Unified framework for the parallelization of divide and conquer based tridiagonal systems

J. López a, O. Plata a, F. Argüello b, E.L. Zapata a,*

a Department of Computer Architecture, University of Málaga, Aptdo. 4114, E-29080 Málaga, Spain
b Department of Electronics and Computer Science, University of Santiago de Compostela, E-15706 Santiago de Compostela, Spain

Received 29 March 1996; revised 23 November 1996

Abstract

In this paper we describe a method for the regularization and parallelization of tridiagonal algorithms based on the divide and conquer strategy. The method is based on perfect shuffle and unshuffle permutations which transform the flow of these algorithms into a flow with the same pattern of communications in all the stages (constant geometry). We use a unified parallel architecture defined by a column of $P = r^e$ processors ($1 \leq P \leq N/r$, for systems with $N = r^n$ equations) interconnected by means of a shuffle and a ring network as a framework to compare the most important parallel solvers for tridiagonal systems.

Keywords: r-ary tree; Divide and conquer; Multiprocessor; Perfect shuffle/unshuffle; Cyclic reduction; Tridiagonal solvers; Parallel architecture

1. Introduction

The solution of tridiagonal systems is a topic of great interest in many areas of matrix algebra. The Divide and Conquer (DC) strategy has been extensively used in the formulation of tridiagonal algorithms due to the fact that it extracts their inherent parallelism and generates very regular patterns of data. In this paper we present a transformation of DC-based tridiagonal algorithms which permits the determination of a
unified parallel architecture for the efficient execution of all of them and also permits to compare their arithmetic and communications complexities. This work completes Ref. [13], where Successive Doubling, Recursive Doubling and Parallel Cyclic Reduction types are considered. For this reason, here we will focus in the tree-based tridiagonal solvers. This way, we obtain a general unified parallel architecture for DC-based tridiagonal algorithms and use it as a framework for comparison of the most significant tridiagonal algorithms proposed in the current literature.

The transformation we propose proceeds in three phases. In the first phase we transform the data flow of each algorithm into another one with constant geometry (CG) by shuffling the results obtained in each stage of the algorithm. In the second phase, each stage of the CG algorithm is expressed as an operator string (perfect shuffle, unshuffle and others that will be defined in Section 3). As all the stages are identical (constant geometry), the algorithm consists in successively applying the same operator string. In the third phase this operator string is decomposed as a function of other operators that are easy to translate into hardware. There are several possible decompositions. The one chosen permits designing the architecture of a general or specific purpose processor column that is adequate for the efficient execution of the algorithm. The operator string that expresses each stage determines the internal structure of the processors and the interconnection network. As a result we obtain a unified architecture for tridiagonal system solvers. It consists in a column of processors interconnected by means of a perfect unshuffle and a ring type network with bidirectional buses. The unified architecture allows all communications of the algorithms to be carried out between processors that are physically connected.

We have organized the rest of the paper as follows: in Section 2 we introduce six tree-based tridiagonal algorithms and classify their stages into direct, inverse and extended trees. In Section 3 we obtain the constant geometry versions of the direct, inverse and extended trees and their expression as operator strings. In Section 4 we propose several relationships and decompositions among operators, we obtain the appropriate parallel architecture for the computation of the three types of trees and deduce the unified architecture. Finally, in Section 5 we describe the algorithms introduced in Section 2 and carry out a comparative analysis of the arithmetic and communications complexity of some of the most important tridiagonal solvers.

2. Tree-based tridiagonal algorithms

A tridiagonal system is made up of a set of $N$ linear equations in $N$ unknowns, $A \cdot x = d$, where $A$ is an $N \times N$ tridiagonal matrix of the form

$$
\begin{bmatrix}
  b_0 & c_0 & & & \\
  a_1 & b_1 & c_1 & & \\
 & a_2 & b_2 & c_2 & \\
 & & \ddots & \ddots & \\
 & & & a_{N-1} & b_{N-1}
\end{bmatrix}
$$

(1)

We denote $d$ and $x$ as the $N$-dimensional known and unknown vectors, respectively.
In the last few years, considerable attention has been paid to obtaining new tridiagonal algorithms that are efficient in vector processors and multiprocessor systems. An important group of these presents a tree type data flow. The Cyclic Reduction algorithm is efficient in vector processors [9,12]. Variations of this algorithm with radix \( r > 2 \) have recently been proposed [6–8] in order to optimize the interaction with memory and exploit locality at the level of vector registers. The radix \( r \) Cyclic Reduction (r-CR) algorithms carry out an elimination stage whose data flow belongs to a type of tree we will call extended, and a substitution stage whose data flow is an inverse \( r \)-ary tree. As an example, Fig. 1 displays the data flow of the 2-CR algorithm for a system of \( N = 2^4 \) equations. The first 4 stages of this figure (left half) correspond to the elimination stage. The algorithms by Müller and Scheerer [15], Krechel et al. [11], Cox and Knisely [5] and Gauss Elimination–Cyclic Reduction (GECR) [10] are efficient in multiprocessor systems. They partition the system into \( P \) subsystems \( (P = 2^p) \) and evolve in three stages. In the first one, we will call local reduction, they apply an elimination process independently (or using some equation from neighboring subsystems) to each subsystem. In the second stage a tridiagonal system, which we will call reduced system, is considered. It is made up of \( 2P - 2 \) equations (in the case of Müller and Scheerer algorithm), \( P \) in the other cases. In this second stage, this group of algorithms solves the reduced systems by means of several methods with tree type data flows. Finally, in the third stage (substitution), the solutions of the reduced system permit the calculation of the remaining unknowns.

In this section we carry out a classification of the tree type tridiagonal algorithms mentioned above with respect to the characteristics of their data flows and we briefly comment the operation carried out in each node of the tree. In Section 5 we analyze the arithmetic complexity in detail. Regarding the stability of the algorithms, we direct the reader to the original works.
2.1. Classification

2.1.1. Direct binary tree

This type of data flow is found in the elimination phase of the reduced system in the algorithms by Cox and Knisely [5], Krechel et al. [11] and Müller and Scheerer [15]. In the first two cases, each butterfly (operation carried out in each node) has two inputs and one output, whereas in the last case the butterflies have two inputs and two outputs, but only one of the outputs progresses to the next stage. For Cox–Knisely and Krechel–Plum–Stüben algorithms, the operation executed in each node is the concatenation of the two input data items (all-to-one broadcast). Müller–Scheerer butterflies apply the local reduction algorithm to a 4 equation system. The direct binary tree of the Müller–Scheerer algorithm is the most general, and from now on, we will consider it the representative data flow of a direct binary tree.

2.1.2. Extended r-ary tree

This type of data flow is found in the elimination stage of the r-CR algorithms \((r \geq 2)\) [6] (see left half of Fig. 1 for radix 2). Some data items (one equation) are inputs to two r-CR-butterflies. The butterflies present \(2r - 1\) inputs (equations with unknowns at a distance of \(r' - 1\)). The elimination process executed in each node produces \(r\) outputs (equations with unknowns at a distance of \(r'\)). One output progresses to the next stages of the tree, whereas the \(r - 1\) remaining outputs are eliminated. The 2-CR algorithm is also used for solving the reduced system of the GECR algorithm [10].

2.1.3. Inverse r-ary tree

This type of tree presents a data flow that is the inverse of the direct tree (see right half of Fig. 1). It is found in the stage of unknowns distribution of the reduced system of the Cox–Knisely [5] and Krechel–Plum–Stüben [11] algorithms, and in the substitution stage of the Müller–Scheerer [15], r-CR [6,8,9] and GECR [10] algorithms. In the first two cases the butterflies present the input data in its two outputs (one-to-all broadcast). In the substitution stage of the r-CR algorithms (for \(r = 2\) it is represented in the right half of Fig. 1), each butterfly presents \(r\) inputs and \(r\) outputs. One input is a pair of unknowns and the remaining \(r - 1\) inputs are equations that did not progress in the direct tree. A substitution process that permits the calculation of \(r - 1\) new unknowns is carried out in each node. The \(r + 1\) unknowns are organized into \(r\) groups of two consecutive unknowns that constitute the \(r\) outputs of each r-CR butterfly. The substitution stage of the Müller–Scheerer algorithm is similar \((r = 2)\).

Finally, we must point out that other tridiagonal algorithms with tree type data flows have been proposed. We have not included them here because their characteristics are the same as the ones we have described. Thus, Bondeli [4] and Agüí and Jiménez [1] have proposed a method that is similar to Müller and Scheerer [15] and Bessenrodt and Weberpals [3] and Reale [17] propose algorithms that are similar to Cox and Knisely [5].

3. Constant geometry trees

The central role in constant geometry algorithms is carried out by perfect unshuffle (we will denote as \(\Gamma'\)) and perfect shuffle (denoted as \(\sigma\)) permutations. \(\Gamma'\) transforms
the data flow of a \( r \)-ary direct tree into a constant geometry flow. The basic idea, due to Pease [16], is that given a subsequence of \( r \) elements of a sequence \( S \) whose indices differ only in their \( t \)-th digit (i.e. they are at a distance of \( r^{t-1} \), \( t > 0 \)), \( \Gamma \) places them at a distance of \( r^{t-2} \). Consequently, if we carry out a \( \Gamma \) permutation of the output sequence in each stage of the tree, two elements initially at a distance of \( r^t \) will occupy consecutive positions after \( t \) stages. This way, the inputs to the trees specific butterflies are at distances of 1 in all the stages. The data flow obtained is called constant geometry \( r \)-ary direct tree. In the same way, \( \sigma \) transforms the data flow of an inverse tree into a constant geometry data flow.

In addition to shuffling/unshuffling operators, we define the operator \( B \) to represent the arithmetic operations that transform one stage into the next, that is, the calculations associated to the butterflies. On the other hand, we define the operators \( \Omega \), to denote the operation of reducing the number of data found in the direct tree structure, and, in a similar way, \( \varepsilon \), which expresses the inverse operation to \( \Omega \), found in inverse tree structures.

Without any loss of generality we will restrict the analysis of the algorithms to radix 2 in order to clarify the reasoning. Whenever necessary we will comment on the differences that may exist between radix 2 and the general case, radix \( r \).

### 3.1. Radix \( r \) operators

We will consider trees with \( N = r^n \) leaves \((r \geq 2)\), referencing each of these leaves by means of an index \( i \) \((0 \leq i < N)\), being \([i_n \ldots i_1]\) the base \( r \) representation of such index \( i \).

**Definition 1.** The perfect unshuffle (shuffle) operator \( \Gamma \) (\( \sigma \)) transforms a data sequence into another one of the same length, performing a cyclic rotation of order 1 to the right (left) in the \( r \)-ary representation of the index of each element of the sequence.

\[
\Gamma [i_n \ldots i_2 i_1] = [i_1 i_n \ldots i_2] \quad (2)
\]

\[
\sigma [i_n i_{n-1} \ldots i_1] = [i_{n-1} \ldots i_1 i_n] \quad (3)
\]

**Definition 2.** The reduction operator \( \Omega \) transforms a data sequence into the sequence of its \( N/r \) first elements, eliminating the most significant digit \( (i_n) \) from index \( i \) of an element if \( i_n = 0 \), or eliminating the complete element in any other case.

\[
\Omega [i_n \ldots i_1] = \begin{cases} 
[i_{n-1} \ldots i_1] & \text{if } i_n = 0 \\
nil & \text{otherwise}
\end{cases} \quad (4)
\]

**Definition 3.** The extension operator \( \varepsilon \) is the inverse of operator \( \Omega \), adding a digit to the most significant position of the indices of the elements of data sequence.

\[
\varepsilon [i_{n-1} \ldots i_1] = \{[i_n i_{n-1} \ldots i_1] / 0 \leq i_n < r\} \quad (5)
\]

Through this paper we will consider that the operators in a string are applied from left to right, i.e., \( \alpha \beta(x) = \beta(\alpha(x)) \).
3.2. Direct r-ary constant geometry trees

Fig. 2 shows the constant geometry version of a direct binary tree with 16 leaves. In the most left column of Fig. 2 we find the indices of the initial data \( c_i^0 \). In each column, the shadowed boxes represent the sequence of inputs to the butterflies. At each stage a butterfly operator \( B \) (execute a butterfly for each pair of inputs) is applied to a sequence of half the length of the previous one. White boxes represent results of a step that will not be processed further. In Fig. 2 we carry out the unshuffling of data generated in each \( t \)-th stage, that we name \( z_i^{t-1} \), and eliminate the white boxes to obtain the input data sequence \( c_i^t \) for the \((t+1)\)-th stage. Note that butterfly inputs are at a distance of 1 in all the stages. Having in mind our convention on applying the operators in a string from left to right, we can conclude that a 2-CGT (radix 2 Constant Geometry Tree) algorithm consists in applying operator string \( B^t \) \( n \) times to a sequence of \( N = 2^n \) data:

\[
2 \text{-CGT} = (B^t)^n.
\]  

(6)

In order to view the evolution of the data better, we will assume that the sequence \( c_i^{t-1} \) of initial data of stage \( t \) is distributed as a matrix \( C^{t-1} \) with 2 rows and \( 2^{n-t} \) columns,

\[
C^{t-1} = \begin{pmatrix}
  c_0^{t-1} & c_1^{t-1} & \cdots & c_2^{t-1} \\
  c_0^{2^{n-t+1}-1} & c_1^{2^{n-t+1}-1} & \cdots & c_2^{2^{n-t+1}-1}
\end{pmatrix}
\]  

(7)

The rows of matrix \( C^{t-1} \) are numbered 0 and 1 from top to bottom and the columns 0 to \( 2^{n-t} - 1 \) from right to left. The string \( B^t \) in Eq. (6) determines three stages in the execution of the \( t \)-th stage of a direct binary tree:
(1) B computes a butterfly for each column of matrix $C^{t-1}$, overwriting the results over the data used for its calculation and thus transforming matrix $C^{t-1}$ into another one, $Z^{t-1}$, of the same dimension

$$Z^{t-1} = \begin{pmatrix}
    z_2^{t-1} & \cdots & z_2^{t-1} & z_0^{t-1} \\
    z_3^{t-1} & \cdots & z_3^{t-1} & z_1^{t-1}
\end{pmatrix}$$ (8)

(2) $I$ rearranges the data of $Z^{t-1}$, transforming it into matrix $G^{t-1}$, also of the same dimension,

$$G^{t-1} = \begin{pmatrix}
    z_2^{n-t-1} & \cdots & z_4^{t-1} & z_0^{t-1} \\
    z_5^{t-1} & \cdots & z_6^{t-1} & z_2^{t-1}
\end{pmatrix}$$ (9)

(3) $\Omega$ eliminates half of the columns (starting on the left) of $G^{t-1}$, and thus obtains matrix $C^t$ (with 2 rows and $2^{n-t-1}$ columns), containing the initial data for the $(t+1)$-th stage.

In the general case (radix $r$) the expression of the algorithm as an operator string is identical, and consequently defines the same three stages as in the case of $r = 2$. Matrix $C^{t-1}$ with the initial data of the $r$-th stage is now

$$\begin{pmatrix} c_{(r-1)t+1}^{t-1} & \cdots & c_{r-1}^{t-1} \\
    c_{(r-1)t+2}^{t-1} & \cdots & c_{(r-1)t+r-1}^{t-1} \\
    c_{rt-1}^{t-1}
\end{pmatrix}$$

where $u = n - t$.

3.3. Extended $r$-ary constant geometry trees

The data flow of the elimination stage of the CR algorithm is a direct extended binary tree (see left half of Fig. 1). Each node at stage $t$, where a $B_{CR}$ butterfly is computed, has 3 inputs at a distance of $2^{t-1}$ and 2 outputs. Fig. 3 shows the data flow obtained after applying $I$ to the data generated in each stage. We will call this data flow direct extended binary tree with constant geometry (2-XCGT).

In a similar way as in the previous case (direct trees), in the $r$-th stage of a 2-XCGT we will assume the initial data sequence, $c_i^{t-1}$, distributed as a matrix $C^{t-1}$ (see Eq. (7)). Then we identify four steps in the execution of the $t$-th stage:

(1) Matrix $C^{t-1}$ is transformed into matrix $F^{t-1}$,

$$F^{t-1} = \begin{pmatrix}
    c_2^{t-1} & c_3^{t-1} & c_1^{t-1} & I \\
    c_4^{t-1} & c_2^{t-1} & c_0^{t-1} \\
    c_5^{t-1} & c_3^{t-1} & c_1^{t-1}
\end{pmatrix}$$ (11)

with 3 rows and $2^{n-t}$ columns, where the $j$-th column is made up of the 3 equations we
need for the computation of 2-CR butterflies and \( I \) is the identity equation. Let \( \xi \) be the operator which transforms \( C'^{-1} \) into \( F'^{-1} \).

(2) Let \( B_{CR} \) be the operator that performs a CR butterfly for each column of matrix \( F'^{-1} \) as input. The output matrix will have the same format as Eq. (8).

(3) Phases 3 and 4 are similar to phases 2 and 3 in Section 3.2, that is, the perfect unshuffle operator \( \Gamma \) is applied to matrix \( Z'^{-1} \) of transformed data (third step) and then the number of columns is reduced by a factor of 2 by means of the application of the reduction operator \( \Omega \) (fourth step).

We can conclude that each stage of a 2-XCGT algorithm consists in applying the operator string \( \xi B_{CR} \Gamma \Omega \) to the matrix \( C'^{-1} \), and the complete algorithm in applying this operator string \( n \) times to the initial matrix \( C^0 \),

\[
2-\text{XCGT} = (\xi B_{CR} \Gamma \Omega)^n
\]

In the general case (\( r \)-CR algorithm) each node has \( 2r - 1 \) inputs and \( r \) outputs (in each node we compute a \( B_{r,CR} \) butterfly). \( \xi \) incorporates \( r - 1 \) rows to the top part of \( C'^{-1} \).

### 3.4. Inverse \( r \)-ary constant geometry trees

The constant geometry version of the inverse tree (ICGT) requires the use of a shuffle process of the data in each stage (\( \sigma \) operator). We can minimize the computational cost associated with the permutations if we restrict them to the input data to be processed in each stage. We will achieve this reduction if we previously rearrange the initial data sequence before it is processed by the inverse tree. As an example, Fig. 4 shows an ICGT with \( 2^4 \) leaves. The initial arrangement of the data is displayed in the left column of this figure, the next column to the right displays the same data after rearrangement. In general, this rearrangement is obtained by successively applying operator \( \Gamma \) to the top half of the initial sequence until this half only contains two elements. We will denote as \( d_i^0 \) the initial sequence \( d_i \) after rearrangement. An ICGT can be formulated as the inverse string of Eq. (6) applied to the root of the tree \( d_0^0 \), that is,

\[
2-\text{ICGT} = (\varepsilon \sigma B)^n
\]
4. Unified parallelization

The method we propose for exploiting the inherent parallelism of each stage of CGT, XCGT and ICGT trees is based in a column of processors (PEs), that we can extend to an array of PE columns. We consider the case of a single column made up of \( P \) PEs \( (P = 2^p) \), being the number of PEs lower than the number of data, \( N = 2^n \), that is, \( p < n \).

In order to compute the \( t \)-th stage of the CGT, matrix \( C^{t-1} \) of input data to that stage (see Eq. (10)) should be distributed among the PEs local memories so that each PE evaluates \( 2^{n-p-t} \) butterflies (columns in \( C^{t-1} \)) at stages \( t = 1, \ldots, n - p \). For steps \( t = n - p + 1, \ldots, n \) the number of columns in \( C^{t-1} \) is less than \( P \), so there are idle PEs and active PEs will compute only one butterfly each cycle. From the mapping point of view, we are decomposing the binary indices of the data into three fields: \( PE, cycle \) and \( bus \). The field \( PE \) identifies the processor, the field \( cycle \) refers to the processing cycle in which data is computed and the field \( bus \) identifies the input (and output) buses of each PE [21]. These three fields can be arranged in six different ways, although only two of them generate different parallel architectures. Considering the leftmost digit of the index as the most significant one, these two orderings are (cycle, \( PE \), bus) and (\( PE \), cycle, bus). The first one of them, called cyclic distribution, assigns consecutive butterflies to different PEs (as the \( PE \) field varies more rapidly than the cycle field), whereas the second, called block distribution, assigns a block of consecutive butterflies to each PE [21]. Without any loss of generality, we will restrict this study to the cyclic distribution ordering of the three fields, that is, (cycle, \( PE \), bus). A similar study can be carried out using a block distribution, obtaining exactly the same parallel architecture.

The design of a single PE column for the 2-CGT, 2-XCGT and 2-ICGT algorithms is based on the decomposition of the permutation operators \( \Gamma \) and \( \sigma \) as a string of elementary operators. In the next subsection we will introduce a formal description of this decomposition into elementary operators. In the remaining subsections we will design the parallel architectures with a single PE column for each type of tree with

![Fig. 4. Constant-geometry version of an inverse binary tree (2-ICGT) with 32 leaves.](image-url)
constant geometry considered in detail. The design of PE meshes will not be considered in this paper.

4.1. Decomposition into elementary operators

We will borrow the notation from Section 3.1. We will use the decomposition of the indices of the data \( i \) into three fields, \( i = (s, \rho, \theta) \), with \( c = [s_a \ldots s_1], \rho = [\rho_b \ldots \rho_1], \theta = [\theta_t \ldots \theta_1] \), where \( a + b + c = n \). Also we will use \( \Gamma(s, \rho, \theta) = ([\theta_1 s_a \ldots s_2], [s_1 \rho_b \ldots \rho_2], [\rho_1 \theta_t \ldots \theta_2]) \). In the following we will define the constraint of operators \( \Gamma \) and \( \sigma \) to some of the fields and some others operators.

**Definition 4.** Partial unshuffle and shuffle operators

\[
\Gamma^{s, \theta}(s, \rho, \theta) = ([\theta_1 s_a \ldots s_2], \rho, [s_1 \theta_t \ldots \theta_2])
\]

\[
\sigma^{s, \theta}(s, \rho, \theta) = ([s_a \ldots s_2 \rho_b], [\rho_b \ldots \rho_1 s_a], \theta)
\]

**Definition 5.** We assume that field \( \theta \) has a single digit (\( \theta_1 \)). The decimation operator \( d^{s, \theta} \) introduced in Ref. [21], converts a two dimensional data sequence into a three dimensional one, splitting the field \( s \). Its inverse operator will be noted as \( \tau^{s, \theta} \)

\[
d^{s, \theta}(s, \rho, [1]) = ([s_a \ldots s_2], \rho, [s_1])
\]

\[
\tau^{s, \theta}(s, \rho, \theta) = ([s_a \ldots s_1 \theta_1], \rho, [1])
\]

**Lemma 1.** The perfect unshuffle (shuffle) operators can be decomposed into two partial unshuffles (shuffles)

\[
\Gamma = \Gamma^{s, \theta} \Gamma^{\rho, \theta}
\]

\[
\sigma = \sigma^{\rho, \theta} \sigma^{s, \theta}
\]

**Proof.** \( \Gamma^{s, \theta} \Gamma^{\rho, \theta}(s, \rho, \theta) = \Gamma^{s, \theta}(\Gamma^{\rho, \theta}(s, \rho, \theta)) = \Gamma^{s, \theta}([s_1 s_a \ldots s_2], [\rho_b \ldots \rho_1], [s_1]) = ([s_1 \theta_1 s_a \ldots s_2], [s_1 \rho_b \ldots \rho_2], [\rho_1]) = \Gamma(s, \rho, \theta) \). The second expression can be proven in the same way. \( \square \)

4.2. Processors column for 2-CGT

In this type of trees, the number of butterflies computed in each stage is reduced by a factor of 2 as it progresses through the tree. This reduction has been represented by means of the reduction operator \( \Omega \) (see Eq. (4)), which is really eliminating the most significant binary digit of the indices of the data as we go from one stage of the tree to the next.

As a consequence, the application of the reduction operator \( \Omega \) to the three fields (cycle, PE, bus) classifies the \( n \) stages of a 2-CGT tree into three types. In each of the first \( n - p - 1 \) stages \( (t = 1, 2, \ldots, n - p - 1) \), the PEs locally process \( 2^{n-p-t} \) butterflies, one for each cycle. The operator \( \Omega \) (which we will denote by \( \Omega^{\text{cycle}} \) during these first stages), reduces by a factor of 2 the number of cycles of the stage, because the most
Fig. 5. Data flow of a 2-CGT tree for \( N = 2^5 \) leaves distributed in a cyclic fashion over a column of \( P = 2^2 \) PEs.

significant radix-2 digits of the indices come from the cycle field. At the end of this type of stages, the field cycle disappears from the indices of the data. A second type of stages \( (t = n - p, \ldots, n - 1) \) starts then, where each PE processes a single butterfly per stage. In this case, operator \( \Omega \) (now represented by \( \Omega^{PE} \)) reduces the number of active PEs by a factor of 2 per stage. In the last stage \( (t = n) \), the indices of the data have a single field, bus. One butterfly is computed in the only active PE (number 0) and operator \( \Omega \) (represented by \( \Omega^{bus} \)) acts over the field bus, generating a single element, which is the root of the tree (see, for example, Fig. 5).

Taking this information into account, the design of a general purpose PE column is defined in Theorem 1,

**Theorem 1.** The data flow of a \( N = 2^n \) leaves 2-CGT can be mapped onto a column of \( P = 2^p \) processors that implement in the first \( n - p - 1 \) stages the operator string

\[
B \Gamma^{cycle,bus} \Omega^{cycle} \Gamma^{PE,bus}
\]

in the following \( p \) stages the string

\[
B \Omega^{bus} \delta^{PE,bus}
\]

and in the last stage the operator string

\[
B \Omega^{bus}
\]

**Proof.** We obtained in Section 3.2 that 2-CGT = \((B \Gamma \Omega)^n\). It can be easily seen that the following equation is verified

\[
(B \Gamma \Omega)^n(s, \rho, \theta) = (B \Gamma \Omega^{cycle})^{n-p-1}(B \Gamma \Omega^{PE})^p(B \Gamma \Omega^{bus})(s, \rho, \theta)
\]

where we identify \( s = \text{cycle}, \rho = \text{PE}, \theta = \text{bus} \). The proof of the theorem consists in seeing that operator strings of Eqs. (20)–(22) are equivalent to substrings \( B \Gamma \Omega^{cycle}, B \Gamma \Omega^{PE} \) and \( B \Gamma \Omega^{bus} \), respectively:

(1) By Lemma 1, \( \Gamma = \Gamma^{cycle,bus} \Gamma^{PE,bus} \). String (Eq. (20)) follows after observing that operators \( \Omega^{cycle} \) and \( \Gamma^{PE,bus} \) commute.

(2) In this case, data indices have only two fields \((\text{PE}, \text{bus})\). Then \( \Gamma \Omega^{PE}([\ ], [\rho_k \ldots \rho_1], [\theta_1]) = \Omega^{PE}(\theta_1, \rho_k \ldots \rho_1) = ([\ ], [\rho_k \ldots \rho_2], [\rho_1]). \) On the other
hand, $\Omega^{bus,PE}_{bus}(\cdot, [\rho_1 \ldots \rho_1], [\theta_1]) = \delta^{PE, bus}(\cdot, [\rho_k \ldots \rho_1], [\rho]) = (\cdot, [\rho_k \ldots \rho_2], [\rho_1])$, and Eq. (21) follows.

(3) It is obvious, since there exists a single one-bit field. $\square$

As an example, Fig. 5 shows the data flow of a 2-CGT tree for $N = 2^2$ elements distributed in a cyclic fashion over a column of $P = 2^2$ PEs. In the first stage ($t = 1$), the initial data item $c^0(24) = c^0([11], [00], [0])$ goes to PE number 0 through bus 0 and is processed in cycle 3. String Eq. (20) performs the following transformations: $[11][00][0] \rightarrow [01][00][1] \rightarrow [1][00][1] \rightarrow [1][10][0]$. This means that $c^1(24)$ is introduced into PE number 2 through bus 0 and is processed in cycle 1 in the second stage. In this case, string Eq. (20) generates $[1][1][0]$. Observe that the field cycle is eliminated. Each PE processes a single butterfly in the third stage. In this stage, data item $c^2(24)$ is introduced into PE number 3 through bus 0 and is processed applying string Eq. (21), undergoing transformations $[1][1][1]$. That is, $c^3(24)$ will input PE number 1 (bus 1) and will be processed applying string Eq. (21) which eliminates it, that is, $c^4(24)$ does not participate in the rest of the stages. Hence, a data item progresses until a digit that is different from 0 appears in the least significant position of its index. Observe that the three dimensional representation of the index allows us to follow the evolution of the data in the different stages of the algorithm and carries out the partitioning of the data in a natural and implicit way.

The architecture (computations and communication pattern) of the PE column is defined by Theorem 1. String Eq. (20) is applied in the first $n - p - 1$ stages. The specific $B$ butterflies (2 inputs and 2 outputs) that are computed each cycle determine the arithmetic operations the PEs must carry out in each cycle. The partial perfect unshuffle $\Gamma^{cycle,bus}$ is carried out internally in each PEs local memory (it doesn't modify the field PE) and $\Omega^{cycle}$ eliminates data outputs of operator $B$ with a field cycle not 0. Eliminated data items do not progress to the rest of the stages and are stored in the PE that has generated them. Not eliminated data are directed to the 2 output buses of each PE.

The partial unshuffle $\Gamma^{PE,bus}$ determines the interconnection network (communication pattern) of the PE column, as this operator affects both fields, PE and bus. Observe that operator $\Gamma^{PE,bus}$ transforms data item $([s], [\rho_p \ldots \rho_1], [\theta_1])$ into $([s], [\theta_1 \rho_p \ldots \rho_2], [\rho_1])$. So, all data directed to the output bus $[\theta_1]$ of PE $[\rho_p \ldots \rho_1]$ will be introduced into PE of index $[\theta_1 \rho_p \ldots \rho_2]$ through input bus $[\rho_1]$ in the next stage. The operation it
carries out over the indices is equivalent to moving the data from one PE to another so that the elements provided by an output bus are directed to the same PE without modifying their relative arrangement. Consequently, we must connect output bus \( \theta_1 \) of PE \( \{ p_\ldots p_N \} \) to input bus \( \theta_1 \) of PE \( \{ p_\ldots p_N \} \). Its implementation is carried out by means of the perfect unshuffle interconnection network. As an example, Fig. 6 shows the perfect unshuffle network for 4 PEs (solid lines).

In stages \( t = n - p, \ldots, n - 1 \), operator \( \delta_{\text{PE,bus}} \) moves the least significant bit of the PE field to the field bus and defines an interconnection network that is a particular case of the one defined by operator \( \Gamma_{\text{PE,bus}} \) (only output buses 0 of the active PEs are used). The action of operator \( \delta_{\text{PE,bus}} \) implies dividing by 2 the number of active PEs. In the last stage \( (t = n) \) only one 2 data butterfly is processed in PE number 0 and consequently the interconnection network is not used.

We conclude from Theorem 1 that a 2-CGT can be mapped on a column of PEs interconnected by a perfect unshuffle network in such a way that all communications happen among physically connected PEs. If a block distribution were used instead of a cyclic one, the same case presented in Fig. 5 will be as shown in Fig. 7.

### 4.3. Processors column for 2-XCGT

In Section 3.3 we have obtained the expression of an extended constant geometry tree as an operator string (see Eq. (12)). Comparing this string with Eq. (6), which corresponds to a 2-CGT, we can observe that the data flow of a 2-XCGT can be projected onto a PE column that implements the same operator strings as Theorem 1 but after operator \( \xi \). This operators extends by 1 row matrix \( C^{t-1, \rho} \) (see Eq. (10)) processed by each PE (with index \( \rho \)). The matrix \( F^{t-1, \rho} \) must be obtained in the PE of index \( \rho \) in the \( t \)-th stage, that is

\[
F^{t-1, \rho} = \begin{pmatrix}
C^{t-1}_{2^t - 1, 2^t + 2 \rho - 1} & \cdots & C^{t-1}_{2^t - 1, 2^t + 2 \rho - 1} & C^{t-1}_{2^t - 1, 2^t + 2 \rho - 1} \\
C^{t-1}_{2^t - 1, 2^t + 2 \rho} & \cdots & C^{t-1}_{2^t - 1, 2^t + 2 \rho} & C^{t-1}_{2^t - 1, 2^t + 2 \rho} \\
C^{t-1}_{2^t - 1, 2^t + 2 \rho + 1} & \cdots & C^{t-1}_{2^t - 1, 2^t + 2 \rho + 1} & C^{t-1}_{2^t - 1, 2^t + 2 \rho + 1}
\end{pmatrix}
\]  

(24)

![Fig. 7. Data flow of a 2-CGT tree for \( N = 2^5 \) leaves distributed in a block fashion over a column of \( P = 2^2 \) PEs.](image)
The j-th column of matrix $\mathbf{F}_r^{-1,\rho}$ is made up by the three input data to the B-butterflies. The columns of $\mathbf{F}_r^{-1,\rho}$ have 3 data elements. The set of data incorporated to the j-th column is $\{c_{k-h, j}^{-1,\rho}, 1 \leq h < 2\}$. Data $c_{k-h, j}^{-1,\rho}$ is located in position $(2-h, j)$ of matrix $\mathbf{C}_r^{-1,\rho-1}$ (observe that $k-h = 2jP + 2(\rho - 1) + 2-h$) which is processed in the PE of index $\rho-1$ with the following exceptions:

1. If $\rho = 0$ and $j = 0$ the data item is $I$.
2. If $\rho = 0$ and $j \neq 0$ the data item is $(2-h, j-1)$ of $\mathbf{C}_r^{-1,\rho}$. 

From this description we deduce that all the data operator $\xi$ incorporates to PE $\rho$ are in PE $\rho-1$ (for PE 0 they are in PE $P-1$). Consequently, operator $\xi$ defines a ring network. The appropriate architecture for a 2-XCGT is a column of general purpose PEs which are interconnected by means of a perfect unshuffle and a ring network. The ring interconnection network is obtained by connecting input bus 0 of PE $\rho$ to input bus 2 of PE $\rho-1$ and input bus 0 of PE 0 to input bus 2 of PE $P-1$. These ring buses are represented by dashed lines in Fig. 6 (4 PEs).

4.4. Processors column for 2-ICGT

In Section 3.4 we express a 2-ary ICGT tree as the operator string $(\xi \sigma B)^n$ applied to the root of the tree. An 2-ICGT constructs the inverse tree taking the data from the initial sequence $d^0 = \{d^0_i, 0 \leq i < N\}$. The computation of a 2-ICGT in a PE column requires sequence $d^0$ to be distributed among the PEs. The adequate distribution is obtained by rearranging the original sequence in $n$ stages. In stage $t = 1$, the set of elements $\{d^0_i, 0 \leq i < 2\}$ are assigned to PE 0. In the $t$-th stage ($t = 2, \ldots, n$) the data subset $\{d^0_i, 2^{t-1} \leq i < 2^t\}$ is distributed in a cyclic fashion among the active PEs. Fig. 5 shows the data flow for a 2-ICGT algorithm ($N = 32$, $P = 4$) if we go through the stages and their cycles from right to left. It can be observed that in the first stage the data items of indices 0 and 16 are assigned to PE 0. After this, in the next stages, the data processed in the previous stage is added to the same number of data items taken from the initial sequence.

Following a similar line of argument as that used for 2-ary CGT trees, we will be able to express the operator string $(\varepsilon \sigma B)^n$ in a more convenient way for its physical implementation, as indicated in the following theorem.

**Theorem 2.** The data flow of a 2-ICGT can be mapped onto a column of $P = 2^\rho$ general purpose processors that in the first stage implement the following composition of operators

$$\varepsilon_{\text{bus}}^{\text{bus}} B$$

in the next $p$ stages the string

$$\tau_{\text{PE,bus}, \varepsilon_{\text{bus}}}^{\text{bus}} B$$

and in the last $n-p-1$ stages the operator string

$$\sigma_{\text{PE,bus}, \text{cycle}}^{\varepsilon_{\text{cycle}}, \text{cycle}} B$$
Proof. As the data flow of a 2-ICGT is inverse to the one corresponding to a 2-CGT, the operator strings that implement them are inverse too. This can be easily verified taking into account that operators $\varepsilon$, $\sigma$ and $\tau$ are the inverse of operators $\Omega$, $\Gamma$ and $\delta$, respectively. □

Summarizing, the communication pattern for a 2-ary ICGT tree can be obtained from the architecture designed for the 2-ary CGT tree by exchanging inputs and outputs.

4.5. Unified framework and complexities

In Section 4.2 we have deduced that a column of general purpose processors interconnected by a perfect unshuffle network is an appropriate architecture for the computation of direct 2-ary trees. If we make the buses of the interconnection network bidirectional, we obtain a perfect shuffle network, which is the appropriate network for computing inverse trees. Finally, the expansion by means of a ring network permits the computation of extended direct trees. On the other hand, in Ref. [13] we have established that this architecture is appropriate for computing Successive Doubling (SD), Recursive Doubling (RD) and Parallel Cyclic Reduction (PCR) DC based tridiagonal algorithms. Therefore, we can use this general parallel architecture as a framework for comparing the most significant parallel DC-based algorithms to solve tridiagonal systems that appeared in the literature.

In the rest of this section we examine the arithmetic and communications complexity of the implementation of DC-based tridiagonal algorithms using the unified framework. We assume a number of $N = 2^n$ data elements and $P = 2^p$ PEs, as well as a cyclic data distribution. If a block distribution were used, we would obtain the situation showed in Fig. 7. Note that in this case a load imbalance appears as a consequence of the unshuffling permutations carried out at each stage. These inefficiencies can be avoided suppressing such unshuffling permutations, which permits the reductions to be performed locally, until a unique equation is kept at each PE (as done in the original papers). To solve this reduced system, a tree communication pattern is needed. Consequently, this algorithm is more efficient than the one we are proposing. Of course, it is possible to design a communication pattern to optimize each particular algorithm. However, our purpose is to establish a unified architectural framework to make possible a comparison of the most important proposed tridiagonal systems algorithms.

The number of butterflies computed by the most loaded PE (which is PE 0) is the same for the three types of trees, that is, $N/P + p - 1$. In the $t$-th stage ($1 \leq t \leq n - p - 1$) of a direct or extended tree, it computes $2^{n-p-t}$ butterflies (as the number of butterflies computed is halved from one stage to the next), therefore, a total of $2^{n-p-2}$ butterflies are computed in these first $n - p - 1$ stages. In the remaining $p + 1$ stages, each active PE (which is halved from one stage to the next) computes one butterfly (see, as an example, Fig. 5). This way, the maximum number of butterflies computed by a single PE is $N/P + p - 1$. For the third type of tree, the inverse tree, the calculation is similar, as the computations are the same but carried out in reverse order. The arithmetic complexity will thus be, $N/P + p - 1$ multiplied by the number of arithmetic operations of a $B$-butterfly. Finally, the number of butterflies for the case of SD, RD and PCR
constant geometry algorithms are $nN/2P$, $nN/P$ and $nN/P$, respectively, and the number of arithmetic operations of SD, RD and PCR butterflies are 41, 40 and 24 (see Ref. [13]).

Regarding communications, in the $t$-th stage ($1 \leq t \leq n - p - 1$) of a direct tree, each PE sends/receives 2 messages of size $2^{n-p-t-1}$. In the remaining stages, each active PE sends 1 message (size of 1 data item) and those remaining active in the next stage receive two messages with one data item each (see fourth step in Fig. 5 as an example). In the case of an inverse tree, the same messages are communicated, but in the other way round. Finally, the implementation of an extended tree requires the transmission of the same number of messages through the shuffle network as in the case of a direct tree, but, in addition, each PE sends/receives one message from the adjacent PEs through the ring network. Table 1 summarizes the communication complexity of the three types of trees and also of the SD, RD and PCR algorithms in the unified architecture. The last three algorithms produce the same number of messages ($2n$) with constant sizes of $12N/P$, $6N/P$ and $4N/P$ floating numbers.

5. Comparative analysis of DC-based tridiagonal solvers

In this section we carry out a comparative analysis of the parallel implementation of some DC-based tridiagonal algorithms (constant geometry versions) using the unified framework introduced above.

The first column of Table 2 displays the arithmetic complexity of each algorithm. The algorithm by Müller and Scheerer [15] evolves in three stages (Section 2). In the local reduction stage, two subsystems of $N/P$ equations are assigned to each PE and in each one of them, the unknowns are expressed as a function of two of them ($12N/P$ arithmetic operations). The second stage solves the reduced system. This stage has a communication pattern of a direct tree followed by an inverse tree. The data flow of the
Table 2
Comparative analysis of several constant geometry trees

<table>
<thead>
<tr>
<th>Constant geom. algth.</th>
<th>arithmetic complex</th>
<th>$R/S$ message number</th>
<th>$R/S$ message size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Müller and Scheerer [15]</td>
<td>$(17N/P) + 64(p + 1)$</td>
<td>$4p/4p$</td>
<td>$32/8$</td>
</tr>
<tr>
<td>Cyclic Reduction [9]</td>
<td>$17(N/P) + p - 1$</td>
<td>$5n - p - 5/5n - p - 3$</td>
<td>$16 \cdot 2^{n-p-1}/8 \cdot 2^{l-p-1}$</td>
</tr>
<tr>
<td>r Cyclic Reduction [6]</td>
<td>$(15r - 13K(N/P) + p - 1)$</td>
<td>$(3r - 1)n - p - 5/(3r - 1)n - p - 3$</td>
<td>$16 \cdot r^{n-p-1}/8 \cdot r^{l-p-1}$</td>
</tr>
<tr>
<td>Cox and Knisely [5]</td>
<td>$(41N/2P) + 24P$</td>
<td>$4p/4p$</td>
<td>$16 \cdot 2^{n-p-1}/8 \cdot P$</td>
</tr>
<tr>
<td>GECR [10]</td>
<td>$17(N/P) + p + 1$</td>
<td>$4p/4p$</td>
<td>$16/8$</td>
</tr>
<tr>
<td>Successive Doubling [13]</td>
<td>$41Nn/2P$</td>
<td>$2n/2n$</td>
<td>$12N/P$</td>
</tr>
<tr>
<td>Recursive Doubling</td>
<td>$40Nn/P$</td>
<td>$3n/2n$</td>
<td>$6N/P$</td>
</tr>
<tr>
<td>Parallel Cyclic Reduction</td>
<td>$24Nn/P$</td>
<td>$4n/2n$</td>
<td>$4N/P$</td>
</tr>
</tbody>
</table>
direct tree is treated as a 2-CGT and a system of 4 equations is solved in each node (56 arithmetic operations). The inverse tree is a \( p + 1 \) stages 2-ICGT and two unknowns are substituted in two equations in each node (8 arithmetic operations). Finally, the third stage (substitution of unknowns in \( N/2P \) equations in each PE) has a cost of \( 5N/P \) arithmetic operations in each PE and completes the solution of the system. The arithmetic complexity of the algorithm is \( 17N/P + 64(p + 1) \), as shown in the second row of Table 2. Müller and Scheerer’s algorithm allows also pivoting (see Ref. [18] for a description of this case).

The complexity of the \( r \)-CR algorithm (\( r \geq 2 \), \( N = r^n \), \( P = r^n \)) (Refs. [6-9]) is deduced from Sections 2.1.2 and 2.1.3. Observe that in this case, the algorithm has a tree communication pattern. Hence, there is no local reduction step, as opposed to the Müller and Scheerer’s algorithm and the rest of the tree algorithms considered in this paper. In the \( t \)-th stage of the elimination stage (\( t = 1, \ldots, n \)) the initial data are equations \( E_i^{t-1} \) (\( i = kr^{t-1}, \ 0 \leq k < r^{n-(t-1)} \)) which have unknowns at distance \( r^{t-1} \). Equations \( E_i^t \) (\( i = kr^t, \ 0 \leq k < r^{n-t} \)) are obtained from equation \( E_i^{t-1} \) and the \( 2(r - 1) \) contiguous equations by means of an elimination process [6,8] which has a cost of \( 10r - 8 \) arithmetic operations. Without any additional cost, this process permits modifying equations \( E_i^{t-1} \) so that all present \( i + r^t \) as the third unknown and they differ by a single unknown. This simplifies the substitution stage permitting that each node is calculated with a cost of \( 5(r - 1) \) operations. The complexity of the algorithm is shown in the fourth row of Table 2.

We will not detail the calculation of the arithmetic complexities of the remaining algorithms displayed in Table 2, which were determined in a similar way. The three last rows in Table 2 depict the arithmetic complexity of SD, RD and PCR algorithms as described in Ref. [13].

The communications complexity is displayed in the second and third column of Table 2. The Constant Geometry Cyclic Reduction (CGCR) algorithm has a 2-XCGT data flow (elimination stage) followed by a 2-ICGT (substitution phase). From Table 1 we deduce that the total number of messages received (\( R \)) per PE is \( 3n + p + 1 + 2(n - p - 3) = 5n - p - 5 \), and the number of messages sent (\( S \)), \( 5n - p - 3 \). In the third column of Table 2 we present the size in bytes of the messages in the elimination/substitution stages. The size of the messages is obtained by multiplying the number of data in the message (Table 1) times the size of the data. In the elimination phase (2-XCGT) each data is an equation (4 coefficients, 16 bytes) and the data in the 2-ICGT are two unknowns (8 bytes). In the same way, we have obtained the complexities of the remaining algorithms. In the algorithms by Krechel et al., GECR algorithm, \( r \) Cyclic Reduction and PCR we include the communications between adjacent PEs necessary for obtaining the reduced system.

Observe in Table 2 that constant geometry versions of the CR, GECR, Krechel et al. and Müller and Scheerer algorithms present similar arithmetic complexities. The arithmetic complexity of the \( r \)-CGCR algorithm is higher than that of the CR algorithm by a factor of 2 to the \( p(\log_2 r - 1) \) power.

The number of \( R/S \) messages are similar for the Cox and Knisely, Müller and Scheerer, Krechel et al. and GECR algorithms but the last three are the smallest. For \( r \)-CGCR algorithms the increase of the value of parameter \( r \) leads to a reduction in the
number of stages but the number of messages increases by a factor of $(3r - 1)/\log_2 r$.

From the information in Table 2 we deduce that the constant geometry versions of the Müller and Scheerer, GECR and Krechel et al. algorithms are the most efficient ones for the computation of tridiagonal systems with tree, successive doubling, recursive doubling and parallel cyclic reduction type data flows. The GECR algorithm is based on Wangs method [19] and is found in the excellent work by Johnsson [10]. Surprisingly, the algorithms that have recently been proposed (1990, 1992) do not improve on the GECR algorithm and are less stable, which casts doubts on the efforts devoted to obtaining more efficient tridiagonal algorithms. An interesting direction is to use the great stability of classical methods (Gauss, CR) in order to obtain approximate solutions by solving independently in each PE subsystems with a certain degree of overlap [12].

6. Conclusions

In this paper we have shown that most DC-based tridiagonal algorithms can be implemented on a multiprocessor with a communication pattern defined by the perfect unshuffle/shuffle and ring permutations. A unified parallel framework is obtained by applying an algorithm regularization method based on these permutations, which transforms the algorithms data flows into constant geometry data flows, making the partition of the data arise in a natural way and obtaining an efficient parallel computation of the algorithms.

Transformation of tree algorithms into constant geometry algorithms allows us to make an immediate comparison of all of them from a unified perspective. The evaluation carried out indicates that the algorithms proposed in recent literature do not improve the efficiency of classical algorithms.

Finally, we will indicate that Zapata and Argüello [2,21,22] and Miel [14] have developed constant geometry versions of the successive doubling algorithms for the computation of fast orthogonal transforms. Also, in Ref. [13] we have proposed a unified parallel architecture for tridiagonal solvers based on the Successive Doubling, the Recursive Doubling, or the Parallel Cyclic Reduction flows. If we analyze the results presented in this paper and the architectures proposed in Refs. [2,13,22] we can conclude that it is possible to define a unified constant geometry architecture for most of the classes of divide and conquer algorithms and use it as a framework for their comparison.

References