Abstract

The most effective use of mesh connected computers is achieved by paying careful attention to the organization of the storage and movement of data. For an important class of algorithms the formalization of the different operations they perform lead to an unified treatment for them and may result in important simplifications. In this work we apply this point of view to the Fast Fourier Transform (FFT). In particular, we present a unified view of a set of FFT algorithms on mesh connected computers with non-shared memory. To this end we use a combination of two techniques, 'mapping vector' and index-digit permutations, which allow us to define the organization of the storage and movement of data for any FFT algorithm whose radix is a power of 2. The methodology we have employed is general and can be applied to other algorithms obtained through the divide and conquer strategy.

Keywords: Fast Fourier Transform; Mesh connected computers; Distributed memory multi-processors; Divide and conquer strategy

1. Introduction

During the course of computation on a multiprocessor array computer, the data needs to be rearranged, both between and within the local storage areas of each
processor, in a variety of regular and irregular ways. In general, this involves complex movements of structured data within a structured storage area; visualizing the movement of data and keeping track of the resulting mapping can be difficult. This frequently gives rise to inefficient ad hoc solutions to problems on an array processor.

In [5] we find a general scheme for data and movement organization which can be applied to mesh-connected computers. This method employs a compact representation of the mapping of the data onto the storage, referred to as the ‘mapping vector’. Movement of data is then viewed in terms of changes to the mapping vector. These are expressed as a sequence of more elementary changes whose physical counterparts are readily implemented on an array processor as either macros or subroutines.

The rearrangement of a data array can be generated according to a common permutation of the digits of each element’s index. This type of permutation is known as index-digit permutation. The digit reversed reordering which results from common Fast Fourier Transform algorithms (FFT) is an example. A general formulation of the index-digit permutation is presented in [6]. This formulation provides a clear and precise description of the data reorderings and helps, in many cases, to simplify the algorithms.

The description through index-digit permutations is specially adequate for the FFT. Each computing pass of an FFT may be considered an index digit permutation over the array computer. Implementations of different versions of the FFT on computer networks have been presented, some with partitioning [8], others without [14,1], examples are hypercube computers [11,12], mesh connected computers [5] and vector computers [11], among others.

In this work we use a combination of both mapping techniques, mapping vector and index-digit permutations in order to define the different operations the algorithms carry out on the array computer. This methodology provides us with great flexibility in order to formulate the algorithms and permits a unified approach to a set of parallel FFT algorithms. In particular, we present eight FFT algorithms formulated over mesh connected computers with non-shared memory.

The FFT is the typical algorithm obtained through the application of the divide and conquer technique. This is a general technique for obtaining fast algorithms which finds a large number of applications. Some of these algorithms have a structure that is similar to that of the FFT; as an example we can point out those employed in the calculation of the orthogonal transforms [2,13] and in the solution of tridiagonal systems [7]. The methodology employed in this work could be directly extended to the formulation of these algorithms. In this sense, the eight algorithms we present illustrate the application of this methodology.

The organization of the rest of the work is as follows: in Section 2 we present the structure of the computer we consider, the notation we used and the data movements we will employ in order to describe the parallel FFT algorithms; these algorithms are presented in Section 3; in Section 4 we evaluate them and finally, in Section 5 we present the conclusions.
2. Mesh connected computers

The type of computer considered is an array processor made up of a number of processing elements (PEs), each with its own local storage. Each PE obeys the same instruction simultaneously under the control of a single master control unit. The PEs are arranged in a square two-dimensional array (mesh connected computer). Each PE, except those located on the edges, has four links which connect it to the four immediate neighbouring PEs. A toroid is similar, but the PEs of the top row are connected to the corresponding PEs of the bottom row and the PEs of the leftmost column are connected to the corresponding PEs of the rightmost column forming a torus. This feature is not crucial, but merely improves performance.

Each PE has its own private local memory and processes its own data. The PEs include a very limited form of local control. The local storage areas may be viewed as adding another dimension leading to a three-dimensional structure. Access is random along this dimension, but a communication operation between two PEs is performed through the interconnection network when processing of remote data is required.

Let us consider the mapping of a one-dimensional data sequence of size $N = r^n$, being the radix $r$ a power of 2. The radix is a characteristic parameter of FFT algorithms. We will denote this data sequence using the index-digit representation [6]. In this representation each data item of the sequence is denoted by the base $r$ decomposition of its indices. That is, data item $x(t)$ with index $t = t_n \cdot r^{n-1} + \cdots + t_2 \cdot r + t_1$ is written as

$$[t_n \cdots t_2 t_1].$$  

We assume that the mesh is made up of $V \cdot W$ PEs (distributed as $V$ rows and $W$ columns). Each PE will have a local memory of $U = N/(V \cdot W)$ data items. In the particular case where $V$ and $W$ are powers of the radix ($V = r^v$ and $W = r^w$) and with a cyclic distribution of the data [14], expression (1) can be written as

$$[\text{memory}, \text{row}, \text{column}],$$  

with $\text{memory} \equiv t_n \cdots t_{i+n+1}$, $\text{row} \equiv t_{i+n} \cdots t_{i+n+2}$, and $\text{column} \equiv t_d \cdots t_1$. That is, the data item of index (2) is assigned to the memory position $\text{memory}$ of the PE of coordinates (row, column). A block distribution of the data is also possible, but the efficiency of parallel FFT algorithms is worse. With a block type data distribution, coordinate $\text{memory}$ would be assigned to the least significant digits of the indices.

For the sake of clarity, we will assume that $U$, $V$, and $W$ are powers of the radix. This implies a loss of generality that can be avoided by considering the binary representation of the data indices instead of the index-digit representation. If $U$, $V$, and $W$ are not powers of the radix, we must distribute the bits of the boundary digit between the two adjacent dimensions $\text{memory}$ and $\text{row}$, or $\text{row}$ and $\text{column}$.

We will later show this process in detail. In particular, Lemma 8 establishes the relationship between the two different representations and Fig. 7 displays an example of its application.
2.1 Data movement

In order to formulate the FFT algorithms over the mesh we are going to define a set of algebraic operators which will allow us to describe the operations carried out with the data. These operators are defined by means of different index-digit permutations. For writing the expressions of the operators we will follow the convention of composing operators from left to right. For example, in the operator string $\phi_1 \cdot \phi_2$, we first execute $\phi_1$ and then, $\phi_2$.

We will define two types of operators which correspond to computations and communications respectively. The operators of the first type represent in-place arithmetic operations over the data items in the local memories of the PEs. An in-place operation writes the results of the operation in positions of the data employed in the calculation. We define the following operator which represents computations,

**Definition 1.** The butterfly operator, $B_i$, $n \geq i > v + w$, reads those sets of $r$ data items whose position differs precisely in their $i$th digit, performs the required arithmetic operations over them and writes the $r$ results in the original positions.

As we are only interested in regrouping the data we will take the computations to be carried out as arbitrary. The definition of this operator implies that the set of $r$ data items to be combined is in the local memory of a single PE. This operator will commute with the redistributions of data among PEs that meet this requirement. We will perform redistributions of this type in order to optimize the algorithms. So as to simplify the notation, we will write $B = B_r$. In Fig. 1 we show an example of the application of this operators on a mesh of size $2 \times 2$.

![Fig. 1. Application of the butterfly operator, $B_4$ and $B_3$ radix 2, and $B_2$ radix 4 to a sequence of length 16 in a mesh of size $2 \times 2$. The sets of data we have marked are those that recombine.](image-url)
The operators of the second type represent redistributions of the data among PEs. Thus, the equality provided by a generic operator \( \phi \{ x, y, z \} = \{ x', y', z' \} \), means that the data item located in the \( x \)th position of the local memory of the PE whose coordinates are \( (y, z) \) is moved to the \( x' \)th memory position of the PE whose coordinates are \( (y', z') \). Among the operators that represent communications we define the following,

**Definition 2.** The exchange operator (transposition), \( E_{i,j}, i \geq j \), exchanges the \( i \) and \( j \)th digits of the index-digit representation of the data,

\[
E_{i,j}[t_n \cdots t_i \cdots t_i \cdots t_i \cdots t_i] = [t_n \cdots t_{i+1} \cdots t_{i} \cdots t_{i-1} \cdots t_{i+1} \cdots t_{i} \cdots t_{i-1} \cdots t_i].
\] (3)

This operator is the same as its inverse due to the fact that \( E_{i,j}E_{j,i} = 1 \).

**Definition 3.** The perfect shuffle operator \( \sigma_{i,j}, i \geq j \), performs a cyclic shift to the left between the \( i \) and \( j \)th digits of the index-digit representation of the data.

\[
\sigma_{i,j}[t_n \cdots t_i \cdots t_i \cdots t_i \cdots t_i] = [t_n \cdots t_{i+1} \cdots t_i \cdots t_{i-1} \cdots t_{i+1} \cdots t_i \cdots t_{i-1} \cdots t_i].
\] (4)

**Definition 4.** The perfect unshuffle operator \( \rho_{i,j}, i \geq j \), performs a cyclic shift to the right between the \( i \) and \( j \)th digits of the index-digit representation of the data.

\[
\rho_{i,j}[t_n \cdots t_i \cdots t_i \cdots t_i \cdots t_i] = [t_n \cdots t_{i+1} \cdots t_i \cdots t_{i-1} \cdots t_{i+1} \cdots t_i \cdots t_{i-1} \cdots t_i].
\] (5)

This operator and the previous one are the inverse of each other as \( \sigma_{i,j}\rho_{i,j} = 1 \).

**Definition 5.** The digit reversal operator \( \pi_{i,j}, i \geq j \), performs the reversal of the digits between the \( i \) and \( j \)th digit of the index-digit representation of the data,

\[
\pi_{i,j}[t_n \cdots t_i \cdots t_i \cdots t_i \cdots t_i] = [t_n \cdots t_{i+1} \cdots t_i \cdots t_{i-1} \cdots t_{i+1} \cdots t_i \cdots t_{i-1} \cdots t_i].
\] (6)

This operator coincides with its inverse as \( \pi_{i,j}\pi_{i,j} = 1 \).

For instance, applying these operators to a one dimensional sequence of \( N = 8 \) data items and \( r = 2 \), we have,

\[
(76543210) \xrightarrow{E_{3,1}} (73516240) \quad (7)
\]
\[
(76543210) \xrightarrow{\sigma_{3,1}} (73625140) \quad (8)
\]
\[
(76543210) \xrightarrow{\rho_{3,1}} (75316420) \quad (9)
\]
\[
(76543210) \xrightarrow{\pi_{3,1}} (73516240). \quad (10)
\]

In Fig. 2 we present the implementation of these operators on a mesh of size \( 2 \times 2 \) using a radix 2 index-digit representation, \([t_1, t_2, t_3]\).

We also need to define the following operators,

**Definition 6.** The perfect shuffle operators over the digit subfields, \( \sigma_{i,j,k,l}, i \geq j \geq k \geq l \), performs a cyclic shift to the left from the \( i \)th digit to the \( j \)th and from the \( k \)th to the \( l \)th of the index-digit representation of the data,

\[
\sigma_{i,j,k,l}[t_n \cdots t_i \cdots t_i \cdots t_i \cdots t_i] = [t_n \cdots t_{i+1}t_{i-l} \cdots t_{i-k}t_{i-l} \cdots t_{i-k-1} \cdots t_{i-k}t_{i-l} \cdots t_{i-k-1} \cdots t_i].
\] (11)
Fig. 2. Implementation of the exchange, $E_{3,1}$, perfect shuffle, $\sigma_{3,1}$, perfect unshuffle, $\rho_{3,1}$ and digit reversal, $\pi_{3,1}$ radix 2 operators over a mesh of size $2 \times 2$.

Where, for the sake of clarity we have underlined the digits that are shifted. Note that $\sigma_{i,j,k} \neq \sigma_{i}, \sigma_{j,k}$; however $\sigma_{i,j,k} = \sigma_{i,j,k}E_{i,j}$.

**Definition 7.** The perfect unshuffle operator applied to the two digit subfields, $\Gamma_{i,j,k,l}$, $i \geq j \geq k \geq l$, performs a cyclic shift to the right from digit $i$ to digit $j$ and from digit $k$ to digit $l$ of the index-digit representation of the data,

$$
\Gamma_{i,j,k,l}[t_n \cdots t_l] = [t_n \cdots t_{i+1}t_{i+1} \cdots t_{j-1}t_{j-1} \cdots t_{k+1}t_{k+1} \cdots t_{l+1}t_{l+1} \cdots t_1].
$$

(12)

The objective of these definitions is to have perfect shuffle and perfect unshuffle operators that only modify the memory and column dimensions.

Some relationships between the operators we have defined have been included as lemmas in the Appendix. In addition we can classify the operators into four categories depending on the type of communications they require:

1. If the operator only modifies the memory dimension it does not require communications.
2. If the operator modifies the memory and row dimensions the communications take place in parallel in the columns.
3. If the operator modifies the memory and column dimensions, the communications take place in parallel in the rows.
4. If the operator modifies the row and column dimensions, the communications take place diagonally, traversing both rows and columns.

In the example of Fig. 2, operators $E_{3,1}$ and $\rho_{3,1}$ are of the third type, whereas operators $\sigma_{3,1}$ and $\Gamma_{3,1}$ are of type 4.
3. Formulation of the FFT

The FFT is an algorithm obtained from the Discrete Fourier Transform (DFT) by applying the divide and conquer strategy [4]. The DFT, \( \{X(k), 0 \leq k < N\} \), of a sequence \( \{x(m), 0 \leq m < N\} \), is defined through the following equation,

\[
X(k) = \sum_{m=0}^{N-1} x(m) \exp\left(-j \frac{2\pi km}{N}\right).
\]

(13)

The divide-and-conquer strategy is applied to a DFT of size \( N = r^n \) by dividing the initial data sequence into \( r \) subsequences of length \( N/r \) (radix \( r \) algorithm). Each one of the subsequences obtained is subdivided again into \( r \) subsequences. The scheme is repeated (in \( n \) steps), until minimum subsequences are obtained (\( r \) data). The calculation of DFTs over these minimum subsequences is trivial. After this, the minimum length DFTs are successively recombined into larger DFTs in \( n \) steps. The algorithm thus obtained basically implies two processes: computations and communications, and these processes are repeated during the \( n \) stages that make up the algorithm.

Depending on how the computations and the data flow are organized, a large number of FFT algorithms have been designed [9]. The FFT algorithms under the self-sorting label, unlike the other versions of the FFT, are characterized by the fact that they provide an output sequence that is digit reversed with respect to the input sequence and do not require additional shuffles. There are two different versions of this type of algorithms [3] which using the notation we introduced can be written as follows,

**Algorithm 1.** The radix \( r \) and length \( N = r^n \) FFT can be expressed by means of the following operator string

\[
B\sigma_{n,1} \cdot B\sigma_{n,2} \cdots B\sigma_{n,n} = \prod_{i=1}^{n} B\sigma_{n,i},
\]

(14)

or, by the string,

\[
\prod_{i=1}^{n} \Gamma_{n,n-i+1} B.
\]

(15)

In these expressions, the butterfly operator \( B \), represents the arithmetic operations required by the algorithms whereas the perfect shuffle \( \sigma_{n,i} \), and perfect unshuffle \( \Gamma_{n,i} \) operators represent the communications. In Fig. 3 we schematically illustrate the data flow of these algorithms using the index-digit representation.

These algorithms are self-sorting, and consequently do not require additional shuffles of the data. However, they require diagonal communications over the rows and columns and are difficult to compute in-place. Thus, these algorithms are not very efficient. The communications can be improved by modifying the data redistributions and employing the operators given by Definitions 6 and 7. The resulting algorithms are the following,
Algorithm 2. The radix $r$ and length $N = r^n$ FFT can be expressed by means of the following operator string
\[ \prod_{i=1}^{2w} E_{w+i,j} \prod_{i=1}^{w} B \sigma_{n,2w+i,w,j} \prod_{i=1}^{2w} B \sigma_{n,i} \prod_{i=1}^{n} B \sigma_{n,i}. \] or by the operator string,
\[ \prod_{i=1}^{n-2w} \Gamma_{n,n-i+1} R \prod_{i=n-2w+1}^{n-w} \Gamma_{n,n-i+1} R \prod_{i=n-w+1}^{n} \Gamma_{n,2w+1,w,n-i+1} R \prod_{i=1}^{w} F_{w+i,j}. \]

Proof. We will only prove the first expression. Starting from the first algorithm, Eq. (14), and using the relationship obtained in Lemma 6 (Appendix), we have
\[ \prod_{i=1}^{2w} E_{w+i,j} \prod_{i=1}^{w} B \sigma_{n,2w+i,w,j} \prod_{i=1}^{2w} B \sigma_{n,i} \prod_{i=1}^{n} B \sigma_{n,i}. \]

where we have taken into account that $B$ commutes with $E_{w+i,j}$. \hfill \Box

These algorithms are only defined when the mesh is square (we assume $v = w$). They require a transposition of the data and $n$ computation/communication stages. Considering Eq. (16), the first product represents the transposition of the data between the rows and columns of the mesh and requires diagonal communications. Alternatively, this transposition can be achieved by writing the data in the mesh by rows and reading the results by columns. Out of the $n$ computation/communication stages, $w$ stages (second product) imply parallel communications over
the rows. The next $v = w$ stages (third product), imply parallel communications over the columns, and, finally, the remaining $u$ stages (fourth product) are data shuffles in the local memories and do not require any communications. In Fig. 4 we schematically illustrate the data flow of these algorithms using the index-digit representation.

In general, the perfect shuffle and perfect unshuffle permutations require quite a few communications and are difficult to compute in-place. The exchange of data involves several PEs as several digits of the memory dimension are shifted to the row dimension or to the column dimension. We will now present other FFT algorithms based on the exchange permutation, which is an easier permutation to implement.

**Algorithm 3. The radix $r$ and length $N = r^n$ FFT can be expressed by means of the operator string**

$$\rho_{n-1,1} \prod_{i=1}^{n} BE_{n,i},$$  \hspace{1cm} (19)

**or, by the operator string,**

$$\left(\prod_{i=1}^{n} E_{n,n-i+1} \right) \rho_{n-1,1}. \hspace{1cm} (20)$$

**Proof.** We will only prove the first expression. Starting from the first algorithm, Eq. (14) and using the relationship $\sigma_{n,i} = \sigma_{n-1,i} E_{n,i}$ obtained in Lemma 3 (Appendix),
we have,

\[
\prod_{i=1}^{n} B \sigma_{n,i} = \prod_{i=1}^{n} B \sigma_{n-1,i} E_{n,i} \\
= \prod_{i=1}^{n-1} \sigma_{n-1,i} \prod_{i=1}^{n} BE_{n,i} \\
= \rho_{n-1,1} \prod_{i=1}^{n} BE_{n,i},
\]

(21)

where we have used Lemma 7 (Appendix) and taken into account that \( B \) commutes with \( \sigma_{n-1,i} \).

In these algorithms we can differentiate \( n \) computation/communication stages and one data shuffling stage following the \( \rho_{n-1,1} \) permutation. The data flow in these \( n \) stages is defined by means of the exchange operator; out of them, \( u \) stages do not require communications \( v \) require parallel communications in the columns and \( w \) stages require parallel communications in the rows. Depending on the algorithm, the \( \rho_{n-1,1} \) data shuffle can be carried out over the input sequence or over the output sequence. This shuffling implies diagonal communications over rows and columns. The algorithm given by Eq. (20) is equivalent to the one proposed in [5].

In general, the communications based on the exchange operator are more efficient than those based on the perfect shuffle operator. The exchange operator transfers a single digit from the memory dimension to the row dimension or to the column dimension, and consequently, the number of PEs exchanging data is

\[
\rho_{s,i} \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \\
BE_{k1} \quad 6 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \\
BE_{k2} \quad 5 \quad 1 \quad 2 \quad 3 \quad 4 \quad 6 \\
BE_{k3} \quad 4 \quad 1 \quad 2 \quad 3 \quad 5 \quad 6 \\
BE_{k4} \quad 3 \quad 1 \quad 2 \quad 4 \quad 5 \quad 6 \\
BE_{k5} \quad 2 \quad 1 \quad 3 \quad 4 \quad 5 \quad 6 \\
BE_{k6} \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6
\]

\[
E_{r,i} \quad B \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \\
E_{r,i} \quad B \quad 5 \quad 6 \quad 4 \quad 3 \quad 2 \quad 1 \\
E_{r,i} \quad B \quad 6 \quad 5 \quad 3 \quad 2 \quad 1 \\
E_{r,i} \quad B \quad 6 \quad 5 \quad 4 \quad 2 \quad 1 \\
E_{r,i} \quad B \quad 6 \quad 5 \quad 4 \quad 3 \quad 1 \\
E_{r,i} \quad B \quad 6 \quad 5 \quad 4 \quad 3 \quad 2
\]

Fig. 5. Index-digit scheme of the data redistribution of type 3 algorithms of length \( N = r^6 \). The circles represent the butterfly operator and the lines the exchange operator.
smaller (r PEs for a radix r algorithm). In addition, the algorithm can be easily computed in-place. In Fig. 5 we schematically illustrate the data flow of these algorithms using the index-digit representation.

The following algorithm, also based on the exchange permutation, does not require any additional shuffling of the data implying diagonal communications between rows and columns.

Algorithm 4. The radix r and length N = r^n FFT can be expressed by means of the operator string

\[\prod_{i=1}^{p} BE_{n,i} E_{n,n-i} \prod_{i=p+1}^{n} BE_{n,i},\]  

or by the operator string,

\[\prod_{i=1}^{n-p} E_{n,n-i+1} B \prod_{i=p+1}^{n} E_{n,j-1} E_{n,n-i+1} B.\]  

in both cases \(p = n/2 - 1\) if \(n\) is even or \(p = \lfloor n/2 \rfloor\) if \(n\) is odd.

Proof. We will only prove the first expression. If \(n\) is even, then \(p_{n-1,i} = E_{n-1,i} E_{n-2,2} \cdots E_{n/2+1,n/2-1}\); and if \(n\) is odd, \(p_{n-1,i} = E_{n-1,i} E_{n-2,2} \cdots E_{n/2+1,n/2-1}\). Assuming that \(n\) is even and starting from Algorithm 3.

\[\rho_{n-1,i} = \prod_{i=1}^{n/2-1} E_{n-i,i} \prod_{i=n/2}^{n} BE_{n,i},\]  

where we have taken into account that \(B\) commutes with \(E_{n-i,i}\) and that \(E_{n-i,i} E_{n,i} = E_{n,i} E_{n,n-i}\) (Lemma 2, Appendix). If \(n\) is odd, the proof is similar. □

These algorithms have the two desired characteristics, as they are self-sorting and in-place. They are self-sorting because they only require carrying out \(n\) computation/communication stages, without any need for additional shuffles. They are in-place because the operators defining the data redistributions are exchanges. Some of the stages include two exchange operators, which respectively imply parallel communications over rows and columns. In Fig. 6 we illustrate the data flow of these algorithms using the index-digit representation. A similar algorithm to the one of Eq. (23) is proposed for its implementation on a hypercube computer in [12] although it can be easily adapted for implementation on a mesh.

Algorithms 1 to 4 are based on carrying out a sequence of computation/communication stages. This is not the only possible way to organize a FFT. Another
possibility is to implement a set of computation stages until the data available in the local memories of the PEs is used up (blocks of size $N/(V \cdot W)$) and then introduce a set of communication stages. This scheme is repeated until the set of stages of the FFT is completed. The technique is equivalent to considering a radix $r = N/(V \cdot W)$ algorithm.

To increase the radix does not imply complicating the algorithm. Lemma 8 establishes the expression for radix $r^2$ operators as a function of radix $r$ operators.

---

Fig. 6. Index-digit scheme of the data redistrihutions in the type 4 algorithms of length $N = r^6$. The circles represent the butterfly operator and the lines the exchange operator.

Fig. 7. Index-digit scheme of the data redistrihutions in a type 3 radix 4 algorithm as a function of radix 2 operations of length 256 on a mesh of size $8 \times 2$. The circles represent the butterfly operator and the lines the exchange operators.
This lemma will allow us to implement algorithms over a mesh in which the number of PE rows and columns is not a multiple of the radix. In Fig. 7 we show the data flow of a type 3 radix 4 algorithm as a function of radix 2 operations over a mesh of size 8 \times 2.

4. Evaluation

We have presented a unified version of a set of FFT algorithms on mesh connected computers with non-shared memory. In order to do this we have used the 'mapping-vector' technique proposed in [5] and the index-digit permutation described in [6]. This allows us a clear and precise description of the algorithms, paying special attention to the organization of the storage and the movement of the data.

In particular, we have formulated eight FFT algorithms which due to their characteristics have been grouped into pairs. The second algorithm of each group is obtained by means of the reversal of each of the operations carried out in the first one, and, consequently, their characteristics are similar (they implement the same operations in an inverse order). These eight algorithms are the following:

(1) Two algorithms based on the perfect shuffle and perfect unshuffle permutations, of the not in-place, self-sorting, type and which require diagonal communications over rows and columns.

(2) Two algorithms that are respectively based on the perfect shuffle and perfect unshuffle permutations, of the non in-place type, which require the transposition of the data rows and columns, and with parallel communications over rows and columns.

(3) Two algorithms based on the exchange permutation, of the in-place type which require the permutation of data according to the digit-reversal permutation, either over the input sequence or over the output sequence and with parallel communications between rows and columns.

(4) Two algorithms based on the exchange permutation, of the in-place, self-sorting type and with parallel communication over rows and columns.

Table 1

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Communications</th>
<th>Additional permutations</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3wWN/4</td>
<td></td>
<td>2N</td>
</tr>
<tr>
<td>2</td>
<td>wWN/2</td>
<td>( \Pi_{i=1}^{w} E_{w+i} )</td>
<td>2N</td>
</tr>
<tr>
<td>3</td>
<td>((W-1)N)</td>
<td>( E_{w+1} ) ( \rho_{w-1,1} )</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td>((W-1)N + (2^w - 2^s - p - w - 1)N/2^s )</td>
<td>( \rho_{w-1,1} )</td>
<td>N</td>
</tr>
</tbody>
</table>

\(*\) This term only exists if \( 2w > n - p - 1 \) (\( p = n/2 - 1 \) if \( n \) is even or \( p = \lceil n/2 \rceil \) if \( n \) is odd)
In Table 1 we summarize the number of communications and memory locations required by the algorithms. In this table we have assumed radix 2 algorithms ($N = 2^n$) and square meshes ($W \times W$). The number of communications was obtained adding the cell to cell transfers required for each data item in the computation/communication stages. Thus, an exchange operation $E_{n,i}$ requires $2^N/4$ communications and a shuffle operation $\sigma_{n,i}$ requires $WN/4$ or $WN/2$ communications, if it only affects rows or columns, or if it affects both, respectively (the same as the perfect unshuffle operator $T_{n,i}$). From this we can deduce that algorithms 4 will in general be more efficient. An algorithm of this type (Eq. 23) has been implemented on the Fujitsu AP1000 [10] computer.

The block diagram of the AP1000 computer consists of 64 to 1024 PEs and three independent communication networks (Fig. 8). These are the torus network (T-net) for point-to-point communications between cells, the broadcast network (B-net) for 1-to-$N$ communications in data distribution and collection, and the synchronization network (S-net) for barrier synchronization.

We have implemented a type 4 algorithm (Eq. 23) on the AP1000. This algorithm is one of the most efficient ones as it is in-place, avoiding the duplication of the amount of memory required for data storage, and self-sorting, avoiding the need for additional reordering operations.

We consider a Simple Code Multiple Data (SCMD) programming model. From the host the data is broadcasted to the PEs through the B-net. After this, the successive computation/communication stages begin. The communications between PEs are carried out node to node through the T-net.
Table 2
Execution times (in seconds) measured on the AP1000

<table>
<thead>
<tr>
<th>Number of PEs</th>
<th>Number of data</th>
<th>$2^{14}$</th>
<th>$2^{16}$</th>
<th>$2^{18}$</th>
<th>$2^{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.45</td>
<td>25.42</td>
<td>116.35</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.01</td>
<td>12.66</td>
<td>57.87</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.99</td>
<td>5.04</td>
<td>28.83</td>
<td>130.14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.50</td>
<td>2.27</td>
<td>14.23</td>
<td>64.76</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.25</td>
<td>1.13</td>
<td>5.63</td>
<td>32.26</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0.13</td>
<td>0.56</td>
<td>2.54</td>
<td>15.83</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>0.07</td>
<td>0.28</td>
<td>1.27</td>
<td>6.25</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>0.03</td>
<td>0.14</td>
<td>0.64</td>
<td>2.83</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>0.02</td>
<td>0.07</td>
<td>0.32</td>
<td>1.41</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>0.01</td>
<td>0.04</td>
<td>0.16</td>
<td>0.71</td>
<td></td>
</tr>
</tbody>
</table>

Without any loss of generality, we consider that the trigonometric coefficients required by the computations are precomputed and stored in each PE. The coefficient table is generated using the method proposed in [12]. The trigonometric coefficients of one stage can be computed from those of the previous stage using the relationships

\[
\cos 2\theta = \cos^2 \theta - \sin^2 \theta \\
\sin 2\theta = 2 \cos \theta \cdot \sin \theta.
\] (25)

Using this method, the time needed for the computation of the trigonometric coefficients and the memory required for their storage is reduced. In Table 2 we summarize the times measured for different sizes of the FFT and the mesh, considering floating point simple precision data.

In Fig. 9 we display the speed-up obtained for several sizes of the data sequence. We have obtained an over speed-up in this algorithm when each PE contains $2^{14}$ data items, value that coincides with the size of the cache contained

![Fig. 9. Speed-up obtained for the parallel FFT algorithm on the AP1000 for different values of $N$.](image)
within each PE \((2^{14} \cdot 8\) bytes \(= 128\) Kbytes). These results show that the algorithm we have implemented provides an excellent performance on the AP1000.

5. Conclusions

There is a large number of applications which solve problems using the divide and conquer strategy. This is a general technique for solving computational problems which permits eliminating redundances in them, leading to faster and more efficient algorithms. The typical example of the application of the divide and conquer strategy is the FFT, but we can also cite other orthogonal transforms and the solution of tridiagonal systems, among other. In this work we have applied a methodology based on the combination of two techniques, 'mapping vector' and index digit permutations which allow us to formally describe the operations carried out by the algorithms on mesh connected computers. We have applied this methodology to eight FFT algorithms, but the methodology is general and could be extended to the formulation of other algorithms obtained by means of the divide and conquer strategy.

Appendix

In this appendix we will obtain some relationships verified among operators defined in Section 3.

The butterfly operator \(B_j\) commutes with all those operators that do not modify the \(i\)-th digit, for example, \(B_j E_{i,k} = E_{i,k} B_j (i \neq j, k)\). On the other hand, if an operator modifies the \(i\)-th digit, we will have to take this into account, we can establish the following lemma.

**Lemma 1.**

\[
B_i E_{i,j} = E_{i,j} B_j \quad (26)
\]

\[
\sigma_{n,j} B_i = B_{i-1} \sigma_{n,j}, \quad i > j \quad (27)
\]

\[
B_j \Gamma_{n,j} = \Gamma_{n,j} B_{i-1}, \quad i > j \quad (28)
\]

**Proof.** It is immediate through direct assessment. \(\square\)

The following expressions having to do with the exchange operators are verified:

**Lemma 2.**

\[
E_{i,j} E_{i,k} = E_{i,k} E_{j,k} = E_{j,k} E_{i,j} \quad (29)
\]

\[
E_{i,j} E_{i,k} E_{i,j} = E_{j,k} - E_{i,j} \quad (30)
\]

**Proof.** Immediate through direct assessment. \(\square\)
Lemma 2 establishes different relationships between exchange operations affecting three digits.

We can prove the following relationships regarding the perfect shuffle operator,

**Lemma 3.**

\[ \sigma_{i,j} = \sigma_{i,j+1} F_{j+1,j} = \sigma_{i-1,j} F_{i,j} = F_{i,j} \sigma_{i,j+1} = F_{i,j} \sigma_{i-1,j}. \]  

**Proof.** Immediate by applying the definition of the operators. For instance, the first equation of (31) is proven as follows

\[ \sigma_{i,j+1} F_{j+1,j} \{t_n \ldots t_1\} = F_{j+1,j} \{t_n \ldots t_{i+1}, t_{i-1} \ldots t_j, t_{j-1} \ldots t_1\} \]

\[ = \sigma_{i,j+1} \{t_n \ldots t_1\}. \]  

Lemma 3 provides a relationship between the perfect shuffle permutation and the corresponding permutation that shifts one more digit.

**Lemma 4.**

\[ \sigma_{i,j} = \prod_{k=j}^{i-1} E_{i,k} = \prod_{k=j}^{i-1} E_{k+1,j} = \prod_{k=i-j}^{i-1} E_{i-k+1,j-k}. \]  

**Proof.** Immediate by recursively applying Lemma 3.

Lemma 4 provides the decomposition of the perfect shuffle permutation, \( \sigma_{i,j} \), in \( i-j \) exchange operations. Group theory proves that any permutation over \( i-j+1 \) elements can be expressed as a composition of \( i-j \) exchanges (transpositions). In the case of the perfect shuffle permutation this is the minimum number of exchanges that can be used, although other decompositions with \( i-j \) or more exchanges are possible.

**Lemma 5.**

\[ \sigma_{i,j} = \sigma_{i,k+1} \sigma_{k,j} E_{k+1,j} = E_{i,k} \sigma_{i,k+1} \sigma_{k,j} = \sigma_{i,k+1} E_{k+1,k} \sigma_{k,j}. \]  

The following alternative expressions are also possible

\[ \sigma_{i,j} = \sigma_{k,j} \sigma_{i,k+1} E_{k+1,j} = E_{i,k} \sigma_{k,j} \sigma_{i,k+1} = \sigma_{k,j} E_{i,j} \sigma_{i,k+1}, \]  

where in both cases \( i > k > j \).

**Proof.** Immediate applying Lemma 3.

Lemma 5 establishes that the perfect shuffle permutation \( \sigma_{i,j} \) (which shifts the digit field between the \( i \)th and \( j \)th position) can be decomposed into two permutations which shift the fields \( \{i, \ldots , k+1\} \) and \( \{k, \ldots , j\} \) plus an exchange operation.

The following lemma is verified.
Lemma 6.

\[ \prod_{i=1}^{s} \sigma_{n,i} = \prod_{i=1}^{s} \prod_{j=1}^{s} \sigma_{n,2s+1,s,j}. \]  

**Proof.** By induction on variable \( j \) over the following equation:

\[ \prod_{i=0}^{s} \sigma_{n,i} = (\sigma_{2s,j})^{j-1} \prod_{i=0}^{s} \sigma_{n,2s+1,s,i}. \]  

This equation is verified for \( j = s \), as can be seen by direct substitution. If we assume that it is verified for \( j + 1 \), that is

\[ \prod_{i=0}^{s} \sigma_{n,i} = (\sigma_{2s,j+1})^{j} \prod_{i=0}^{s} \sigma_{n,2s+1,s,i}, \]  

then it is verified for \( j \), as,

\[ \prod_{i=j}^{s} \sigma_{n,i} = \sigma_{n,j} (\sigma_{2s,j+1})^{j-1} \prod_{i=j+1}^{s} \sigma_{n,2s+1,s,i} \]

\[ = (\sigma_{2s,j})^{j-1} \prod_{i=j+1}^{s} \sigma_{n,2s+1,s,i}, \]

as \( \sigma_{n,j} (\sigma_{2s,j+1})^{j-1} = (\sigma_{2s,j+1})^{j-1+1} \sigma_{n,2s+1,s,j} \) which can be seen by direct substitution, and so expression (37) is proven. In addition

\[ (\sigma_{2s,j})^{s} = \prod_{i=1}^{s} E_{s,i}. \]

As we can see by direct substitution, and thus the lemma is proven. \( \Box \)

The following relationships can be established between the perfect shuffle and digit reversal permutations.

**Lemma 7.**

\[ \prod_{k=0}^{i-j} \sigma_{i,j+k} = \prod_{k=0}^{i-j} \sigma_{j+k,j} = \rho_{i,j}, \]  

with \( i \geq j \).

**Proof.** Immediate by induction. \( \Box \)

Similar relationships to those of Lemmas 3 to 7 can be established for the perfect unshuffle permutations \( I_{i,j} \). As \( I_{i,j} = (\sigma_{i,j})^{-1} \), the expressions for the perfect unshuffle operator are obtained from the corresponding expressions for the perfect shuffle operator, inverting each of the operators.

The following lemma relates operators of the same type for different radix.
Lemma 8. Radix $r^2$ operators can be expressed from the corresponding radix $r$ operators as follows

\begin{align}
B_{n}^{r^2} &= \left( \prod_{k=1}^{r} B_{n-k+1}^{r} \right) \rho_{n, n-r-1}^{r} \prod_{k=1}^{r} B_{n-r+k}^{r} \\
E_{i,j}^{r^2} &= \prod_{k=1}^{r} E_{i-j+k, i-j+k}^{r} ^{r} \\
\sigma_{i,j}^{r^2} &= \prod_{k=1}^{r} \sigma_{i-j+k, i-j+k+1}^{r} ^{r} \\
\Gamma_{i,j}^{r^2} &= \prod_{k=1}^{r} \Gamma_{i-j+k, i-j+k+1}^{r} ^{r}
\end{align}

where we have indicated the radix of the operators through an index.

**Proof.** Immediate by direct assessment. \(\square\)

**Acknowledgments**

The authors would like to thank the Centro de Supercomputación de Galicia (CESGA) for the facilities it has provided and Fujitsu Ltd. for making an account on the AP1000 multicomputer available to us.

**References**