Lemma 1: \( \pi(p, n)(I_n \otimes F_p \otimes I_p)\pi(p, n)^T = I_{n/p} \otimes F_p \otimes I_p \).
Lemma 2: \( \pi(p, n)^{n-1} = \pi(p, n)^n \).
Lemma 3: \( \pi(p, n)^T = \pi(n/p, n) \).
Lemma 4: \( \pi(p, n)^T = I_n \).

Using the definitions and the lemmas, a CGF decomposed matrix is modified. The basic \( n \)-point DFT matrix, \( F_n \), can be written as

\[
F_n = (f_m)
\]

where \( f_m = \omega_m^n, \omega_n = \exp(-2\pi i/n) \), \( u \) is the row index, and \( v \) is the column index.

From (A.1), a radix-\( p \) inputs shuffled, outputs ordered, \( n \)-point, CGF [6] can be written as shown in (A.2). If \( n = p^r \),

\[
F_n P(p) = V_P V_{P-1} \cdots V_1
\]

where \( P(p) \) is an input digit reversal shuffling operator:

\[
P(p) = (I_{n/L} \otimes \pi(p, L_1)) \cdots (I_{n/L} \otimes \pi(p, L_m))
\]

where \( V_P \) is a radix-\( p \) constant geometry decomposed matrix:

\[
V_P = (F_p \otimes I_{L_p}) \pi(p, n) T_p = \pi(p, n)^{N/p-1} (I_{n/p} \otimes F_p) \pi(p, n)^{N/p-1} \pi(p, n)^T q
\]

Applying Lemma 3, \( V_P \) can be modified as shown in (A.3):

\[
V_P = \pi(n/p, n) (I_{n/p} \otimes F_p) T_p
\]

REFERENCES


Application-Specific Architecture for Fast Transforms Based on the Successive Doubling Method

E. L. Zapata and F. Argüello

Abstract—The successive doubling method is an efficient procedure for the design of fast algorithms for orthogonal transforms of length \( N = r^k \) where the radix \( r \) is a power of 2. In this correspondence we present a partitioned systolic architecture for the two standard radix successive doubling algorithms: decimation in time (DIT) and decimation in frequency (DIF). We project the index space of the data onto the index space associated with a column of processors interconnected using a perfect unshuffle (DIT) or shuffle (DIF) interconnection network, defined by permutations of order \( r \). The result is a partitioned systolic array with \( 2 \) processors \((Q = r^k, 0 \leq i < n)\), which extracts the maximum spatial and temporal parallelism achieved by the successive doubling algorithm and which can be integrated in VLSI and WSI technologies.

I. INTRODUCTION

There is a class of important orthogonal transforms which can be calculated using a fast algorithm by applying the successive doubling method [4], the objective of which is to minimize the redundant operations. The successive doubling method consists of performing successive bisections of the data until the original sequence is decomposed into \( N/r \) sequences of length \( r (N = r^k) \), where \( r \) is the length of the minimum sequence to be transformed (radix of the transform). Once the \( N/r \) discrete transforms (butterflies) have been calculated, they must be combined to obtain the discrete transform of the original sequence by means of \( \log r \) calculation stages. As a result of the successive bisections of the initial data set, it is necessary to carry out a shuffle of the input sequence (decimation in time (DIT) algorithm) or of the transformed sequence (decimation in frequency (DIF) algorithm) in order to obtain an output sequence \((T(k), 0 \leq k < N)\) in its natural order. The usual way for carrying out this shuffle is by using the bit reversal permutation.

An adequate architecture for integration in VLSI and WSI technologies is the systolic architecture. There are many methods for the systolization and partitioning of algorithms [5], but none of them is applicable to the algorithms based on the successive doubling method. This is why many authors consider the successive doubling method an efficient procedure for the design of fast algorithms for orthogonal transforms of length \( N = r^k \) where the radix \( r \) is a power of 2. In this correspondence we present a partitioned systolic architecture for the two standard radix successive doubling algorithms: decimation in time (DIT) and decimation in frequency (DIF). We project the index space of the data onto the index space associated with a column of processors interconnected using a perfect unshuffle (DIT) or shuffle (DIF) interconnection network, defined by permutations of order \( r \). The result is a partitioned systolic array with \( 2 \) processors \((Q = r^k, 0 \leq i < n)\), which extracts the maximum spatial and temporal parallelism achieved by the successive doubling algorithm and which can be integrated in VLSI and WSI technologies.

1. INTRODUCTION

There is a class of important orthogonal transforms which can be calculated using a fast algorithm by applying the successive doubling method [4], the objective of which is to minimize the redundant operations. The successive doubling method consists of performing successive bisections of the data until the original sequence is decomposed into \( N/r \) sequences of length \( r (N = r^k) \), where \( r \) is the length of the minimum sequence to be transformed (radix of the transform). Once the \( N/r \) discrete transforms (butterflies) have been calculated, they must be combined to obtain the discrete transform of the original sequence by means of \( \log r \) calculation stages. As a result of the successive bisections of the initial data set, it is necessary to carry out a shuffle of the input sequence (decimation in time (DIT) algorithm) or of the transformed sequence (decimation in frequency (DIF) algorithm) in order to obtain an output sequence \((T(k), 0 \leq k < N)\) in its natural order. The usual way for carrying out this shuffle is by using the bit reversal permutation.

An adequate architecture for integration in VLSI and WSI technologies is the systolic architecture. There are many methods for the systolization and partitioning of algorithms [5], but none of them is applicable to the algorithms based on the successive doubling method. This is why many authors consider the successive doubling method an efficient procedure for the design of fast algorithms for orthogonal transforms of length \( N = r^k \) where the radix \( r \) is a power of 2. In this correspondence we present a partitioned systolic architecture for the two standard radix successive doubling algorithms: decimation in time (DIT) and decimation in frequency (DIF). We project the index space of the data onto the index space associated with a column of processors interconnected using a perfect unshuffle (DIT) or shuffle (DIF) interconnection network, defined by permutations of order \( r \). The result is a partitioned systolic array with \( 2 \) processors \((Q = r^k, 0 \leq i < n)\), which extracts the maximum spatial and temporal parallelism achieved by the successive doubling algorithm and which can be integrated in VLSI and WSI technologies.

1. INTRODUCTION

There is a class of important orthogonal transforms which can be calculated using a fast algorithm by applying the successive doubling method [4], the objective of which is to minimize the redundant operations. The successive doubling method consists of performing successive bisections of the data until the original sequence is decomposed into \( N/r \) sequences of length \( r (N = r^k) \), where \( r \) is the length of the minimum sequence to be transformed (radix of the transform). Once the \( N/r \) discrete transforms (butterflies) have been calculated, they must be combined to obtain the discrete transform of the original sequence by means of \( \log r \) calculation stages. As a result of the successive bisections of the initial data set, it is necessary to carry out a shuffle of the input sequence (decimation in time (DIT) algorithm) or of the transformed sequence (decimation in frequency (DIF) algorithm) in order to obtain an output sequence \((T(k), 0 \leq k < N)\) in its natural order. The usual way for carrying out this shuffle is by using the bit reversal permutation.

An adequate architecture for integration in VLSI and WSI technologies is the systolic architecture. There are many methods for the systolization and partitioning of algorithms [5], but none of them is applicable to the algorithms based on the successive doubling method. This is why many authors consider the successive doubling method an efficient procedure for the design of fast algorithms for orthogonal transforms of length \( N = r^k \) where the radix \( r \) is a power of 2. In this correspondence we present a partitioned systolic architecture for the two standard radix successive doubling algorithms: decimation in time (DIT) and decimation in frequency (DIF). We project the index space of the data onto the index space associated with a column of processors interconnected using a perfect unshuffle (DIT) or shuffle (DIF) interconnection network, defined by permutations of order \( r \). The result is a partitioned systolic array with \( 2 \) processors \((Q = r^k, 0 \leq i < n)\), which extracts the maximum spatial and temporal parallelism achieved by the successive doubling algorithm and which can be integrated in VLSI and WSI technologies.

Manuscript received July 17, 1991; revised October 14, 1991. This work was supported by the Ministry of Education and Science (CICYT) of Spain under Contracts TIC 88-0094 and 90-0407, MIC 88-0549 and 90-1264, and Xunta de Galicia XUGA80406888. E. L. Zapata is with the Departamento Arquitectura de Computadores, Universidad de Málaga, 29013 Málaga, Spain. F. Argüello is with the Departamento Electrónica, Facultad de Física, Universidad Santiago de Compostela, 15706 Santiago de Compostela, Spain.

IEEE Log Number 9206018.
doubling algorithm as non-systolizable and, consequently, they have directed their research towards the design of systolic architectures for the discrete transforms [7], [9], [10], or the design of application specific microprogrammable processors [6]. We have recently proposed constant geometry architectures for the radix 2 real fast Hartley and Fourier transforms [3], [12]. The new architectures proposed in this correspondence are more efficient than the ones described in [3] and [12].

Our objective in this work is to design an application specific architecture which permits the exploitation of the parallelism present in the successive doubling method and which is integrable in VLSI and WSI technologies. The constant geometry architecture we present in the following sections is based on the perfect unshuffle or shuffle interconnection networks, which permit efficient mapping and partitioning of the flowchart generated by the successive doubling method without having to use microprogrammed control.

We have structured the rest of this correspondence in the following way. In Sections II and III we present in detail the application specific architecture of the DIT algorithm, defined by the perfect unshuffle permutation. More specifically, in Section II we approach the design of the appropriate processor for obtaining the constant geometry systolic architecture and in Section III we present the application specific parallel architecture. In Section IV we briefly describe the application specific architecture associated with the DIF algorithm, whose constant geometry is determined by the perfect shuffle permutation.

II. THE CONSTANT GEOMETRY ARCHITECTURE

We consider that the size of the transform is \( N = r^u \), where \( r \) is the radix and we will use a two-dimensional representation \([x, y]\) of the index for each data item \( i = 0, 1, \ldots , N - 1 \) in the input sequence

\[
[x, y] = [(x_0, \ldots , x_i), (y_0, \ldots , y_i)]
\]

(1)

where \( x_i \) and \( y_i \) are the digits of the binary representation of \( x \) and \( y \), respectively. The union of \( x \) and \( y \) into one number \( (x \cdot 2^u + y) \) will coincide with the binary representation of the index \( i \) of the data sequence \( (u + v = \log_2 N) \). Finally, we will suppose that the data sequence flows from left to right.

The operators are defined by their effect on the indices of the data items. The decimation operator \( \delta \) converts a row into many by reducing the number of columns:

\[
\delta_{i \downarrow}[x, y] = [(x_0, \ldots , x_i), (y_0, \ldots , y_i)]
\]

(2)

Each row is broken into \( 2^u \) rows, and the operator is well defined if \( k \leq u \). Using this notation, we can define the operator concatenation \( \beta \) which reduces the number of rows of an array by increasing the number of columns:

\[
\beta_{i \uparrow}[x, y] = [(y_0, \ldots , y_i), (x_0, \ldots , x_i)]
\]

(3)

A sequence made of \( 2^u \) rows with \( 2^u \) elements (columns) is transformed into another with \( 2^u \cdot 4^u \) rows of \( 2^u \cdot 4^u \) columns and the operator is well defined if \( k \leq 2^u \). Finally, we define the perfect unshuffle operator \( \Gamma \) of a sequence, which flows from left to right and is organized as a two-dimensional array of \( 2^u \) rows and \( 2^u \) columns:

\[
\Gamma_{i \uparrow}[x, y] = [(y_0, \ldots , y_i), (x_0, \ldots , x_i)]
\]

(4)

\( \Gamma \) performs a rotation to the right of order \( k \) of the binary representation of the index of each element of the sequence and is well defined if \( k \leq u + v \).

We are interested in the efficient hardware implementation of the perfect unshuffle permutation, as it is the base for the design of a constant geometry architecture for radix \( r \) DIT successive doubling algorithm. The output sequence of the processor will have to undergo a perfect unshuffle permutation to maintain the constant geometry in all stages of the transform. From this we deduce that out of all the permutations we can implement with (4) only the particular cases \( \Gamma_{i \uparrow}[x, y] \) and \( \Gamma_{i \uparrow}[x, 0] \) will be of any interest, being \( v = \log_2 r \). Also, these permutations can be obtained by the combination of the operators \( \delta_{i \downarrow} \) and \( \beta_{i \uparrow} \) defined previously.

Lemma 1:

\[
\Gamma_{i \uparrow}[x, y] = \beta_{i \uparrow} \delta_{i \downarrow}[x, y]
\]

(5)

\[
\Gamma_{i \uparrow}[x, 0] = \delta_{i \downarrow} \beta_{i \uparrow}[x, 0]
\]

(6)

Here, the order for the application of the operators is from left to right. The proof of (5) and (6) is immediate.

A. Design of the Processor

The internal structure of the processor will consist of two clearly differentiated sections: Processing (PS) and routing (RS). The PS section will carry out the set of operations associated with the \( r \)-point butterfly (discrete transform of a sequence with \( r \)-points). These operations will depend on which particular transform we are implementing [2]. As an example, Figs. 1(a) and (b) show the PS section corresponding to a 2 and 4 point discrete Walsh transform (DWT), respectively. As we are only interested in the regrouping of the data items and not in the specific computations of each transform, we will consider the PS section as a new operator, the butterfly operator \( B_{i \uparrow} \), which carries out an arbitrary function with \( 2^u \) inputs and \( 2^u \) outputs.

The equalities (5) and (6) guarantee the decomposition of the perfect unshuffle permutation into two elementary permutations which are easily implemented in hardware. Specifically, the concatenation permutation \( \beta_{i \uparrow} \) can be implemented using a FIFO queue of length \( N \) with \( 2^u \) inputs located in cells 0-, 2'-, \ldots , and \((2^u - 1) \cdot 2^u\)th, using a numbering scheme from left to right; the queue must have an output in cell \( N - 1 \).

There are two ways of implementing the decimation permutation \( \delta_{i \downarrow} \). The first can be achieved by means of a FIFO queue of length \( N \) cells \((i = 0, 1, \ldots , N - 1, N = r^u)\) with outputs of the \((N - 1)\), \((N - 2)\), \ldots , and \((N - 2^u)\)th cells having the same numbering scheme as in the previous case; the queue must have an input in cell zero. We can also implement the permutation \( \delta_{i \downarrow} \) by means of a demultiplexer with an input associated with the sequence we wish to decimate, \( 2^u \) outputs and \( v \) control inputs used in a cyclic fashion each clock period. Both solutions require the sequence to be decimated to advance \( 2^u \) positions each cycle.

The hardware implementation of permutation \( \Gamma_{i \uparrow} \) is immediate using permutation \( \beta_{i \uparrow} \) and one of the two alternatives of permutation \( \delta_{i \downarrow} \). This leads to two different designs for the processor, although the internal parallelism is the same in both cases (\( 2^u \) data items are processed in parallel). If we use the FIFO queue as the implementation for operator \( \delta_{i \downarrow} \) (Lemma 1, equality (5)), the design of the processor for the calculation of a stage of DIT successive doubling algorithm is the hardware translation of the following operator string:

\[
B_{i \uparrow} \beta_{i \uparrow} \delta_{i \downarrow}
\]

(7)

where \( v = \log_2 r \). Fig. 2(a) shows the design of the radix 2 processor. We have included a double FIFO queue in order to be able to implement the whole transform by external recirculation of the data, using only one processor with the \( i \)th output connected to the \( i \)th output of the processor for the calculation of a stage of DIT successive doubling algorithm.
input \(i = 0, 1, \cdots, 2^r - 1\). The \(n\) stages (\(n = \log N\)) of the transform are identical, as we apply the operator sequence (7) \(n\) times. At each stage, one queue acts as the output buffer (writing the data generated in the current stage) whereas the other acts as input buffer (reading the data generated in the previous stage) and this function will be exchanged in the next stage, under control of multiplexors MUX0-MUX2. Observe that in the design of Fig. 2(a) the FIFO queues have a length of \(N - 1\) cells \((W = N / r)\) and the inputs associated with permutation \(\beta_{ij}\) have been conveniently distributed. In order to do this we have considered the PS section as a segment of the pipeline made of sections PS and RS.

Fig. 2(b) shows the second alternative for the design of the processor (Lemma 1, equality (6)), we have considered radix 2 again. This design is the hardware translation of the following operator string

\[
\delta_{ij} B_{ij} \beta_{ij} \delta_{ij}
\]

(6)

where \(r = \log_2 r\). For the same reasons as in the design of Fig. 2(a) we have included two FIFO queues of lengths \(N - 1\) with \(2^r\) inputs and only one output in its right end (cell \((N - 2)\)). In this case we can also implement all the stages of DIT successive doubling algorithm using only one processor, with each output path feeding back its corresponding input path.

The design of the processor according to the operator sequences (7) or (8) permits the interpretation of the two-dimensional representation of (1) in the following way: the \(y\) coordinate gives the parallelism for each stage of the transform (a butterfly with \(2^r\) data items is processed each cycle), whereas the \(x\) coordinate establishes the sequentiality for each stage of the transform \((2^r\) butterflies of length \(2^r\)). Therefore, the calculation time for a stage will be \(2^r\) clock cycles, the time used by the processor to compute the butterflies associated with each input vector. With this interpretation of (1), the binary representation of the data consists of two fields [cycle, path]. The data item \((x, y)\) will input the processor through its \(y\)th input path (path = 0, 1, \(\cdots\), \(2^r - 1\)), being a part of the \(x\)th butterfly of the stage (cycle = 0, 1, \(\cdots\), \(2^r - 1\)).

### III. Parallel Architecture

Successive doubling algorithm can be implemented on a rectangular constant geometry array of PE’s \([11]\). Two extremes of this array are the row of \(n\) PE’s and the column with \(Q = r^d\) PE’s, where \(n = \log N\) and \(0 \leq q \leq n - 1\). The row of PE’s permits pipeline design \([8], [11]\), and the sequencing of the transforms without losing cycles. In the other extreme, the column of PE’s impedes the execution of a new transform until the current one has finished, but it permits the extension of the I/O bandwidth directly, as the limitation in the number of PE’s is less restrictive than in the case of the row of PE’s.

To change from a single PE system to a PE column will force us to modify the notation introduced at the beginning of Section II as we need a three-dimensional representation \([x, z, y]\) of the index of each data item in the input sequence:

\[
[x, z, y] = [(x_1, \cdots, x_l), (z_1, \cdots, z_l), (y_1, \cdots, y_l)]
\]

(9)

where \(x_i, y_j, z_i\) are defined as in (1) and the union of \(x, z, y\) into a single number \((x \cdot 2^r)^{\cdot z + y} = 2 + z + y)\) will coincide with the binary representation of the index of the data sequence \((i = 0, 1, \cdots, 2^{r + \cdot r} - 1; w + w + r = n)\). If we consider \(r = \log_2 r\) we can interpret the three-dimensional representation of (9) in the following way: the \(x\) and \(y\) coordinates determine the parallelism of each stage of the transform, \(2^r\) butterflies \((\leq N / r)\) of 

\(2^r\) data items are computed in parallel in one column of \(2^r\) PE’s with \(2^r\) inputs each, and the \(x\) coordinate establishes the sequentiality in each stage of the transform \((2^r\) vector of length \(2^r\)).

The calculation time for a stage will be \(2^r\) clock cycles where the duration of the cycle is the time used by the processor in the computation of a butterfly of \(2^r\) data items.

With this interpretation of equation (9) we are decomposing the binary representation of the index of each data item into three fields [cycle, path]. In each stage, cycle indicates the instant it is processed (cycle = 0, 1, \(\cdots\), \(2^r - 1\)), \(nPE\) indicates the PE where it will be processed \((nPE = 0, 1, \cdots, 2^r - 1)\) and path specifies the path through which it will enter the PE \((path = 0, 1, \cdots, 2^r - 1)\). Thus, for example, let \(N = 64, Q = 4, r = 2\) \((\mu, v) = (3, 2, 1)\). In the first stage of the transform, data item 35 \((100101)\) binary, whose three-dimensional representation is \([100], [01], [11]\), will input PE 1 through path 1 in cycle 4 (35 will be a part of the fifth block of butterflies). \(nPE, cycle, path\) is another possible interpretation with similar characteristics \([2]\).
The perfect unshuffle permutation of order \( k \) on the three-dimensional representation of the data indices is defined as
\[
\Gamma_{[x, z, y]} = \{ [y_k \ldots y_1, x_k \ldots x_1, z_k \ldots z_1], [x_k \ldots x_1, y_k \ldots y_1, z_k \ldots z_1] \}
\]
where \( k \leq u + w + v \). We are again interested in the decomposition of \( \Gamma_{[x, z, y]} \) into elementary permutations. For this reason we are going to generalize \( \Gamma_{[x, z, y]} \), to be able to apply it to two or three dimensions of (9):
\[
\Gamma_{[x, z, y]} = \{ [y_k \ldots y_1, x_k \ldots x_1, z_k \ldots z_1], [x_k \ldots x_1, y_k \ldots y_1, z_k \ldots z_1] \}
\]
where \( k \leq v + u \) and \( k \leq v \), respectively. We define the rest of the permutations of two variables \( \Gamma^{[u, w]}_{[x, z, y]} \), with \( a, b = x, y, z \) \((a \neq b)\) in a similar way. We will also extend the meaning of the operators \( \delta_{[x]} \) and \( \beta_{[x, z, y]} \), which possess a two-dimensional nature, in order to be able to apply them to the new representation of (9). This is, \( \delta_{[x]} \) and \( \beta_{[x, z, y]} \) perform the decimation and concatenation permutations, equations (2) and (3), respectively, on the dimensions \( x \) and \( y \) without modifying dimension \( z \).

**Lemma 2:**
\[
\Gamma_0[x, z, y] = \Gamma_0 \Gamma_0^{[u, w]} \Gamma_0^{[x, z, y]} = \Gamma_0^{[x, z, y]} \Gamma_0^{[u, w]} \Gamma_0[ x, z, y] \quad (12)
\]
where the application order for the operators is from left to right. Its proof is immediate.

Lemmas 1 and 2 guarantee the decomposition of permutation \( \Gamma_{[x, z, y]} \) of a two- or three-dimensional representation of the index of each data item into more elementary permutations, which is the base for the design of a constant geometry architecture. Consequently, we can state the following theorem.

**Theorem 1:** DIT successive doubling radix \( r \) and constant geometry algorithm of a sequence of \( N \) data items (\( N = r^n \)) can be carried out in a column of \( Q \) PE's (\( Q = r \), \( 0 \leq q \leq n - 1 \)) which implements (hardware translation) each stage the following operator string:
\[
\beta_{[x, z, y]} \delta_{[x]} \delta_{[z]} \beta_{[x]} \delta_{[z]} \beta_{[x, z, y]} \Gamma_{[x, z, y]} \quad (13)
\]
where \( r = \log_r N \), the operators are applied from left to right and we consider \([x, z, y]\) interpretation of the three-dimensional representation of the index of each data item.

**Proof of (13):** We have to demonstrate that the operator sequence \( \beta_{[x, z, y]} \delta_{[x]} \delta_{[z]} \beta_{[x]} \delta_{[z]} \beta_{[x, z, y]} \) is equivalent to a perfect unshuffle permutation (10):
\[
\beta_{[x, z, y]} \delta_{[x]} \delta_{[z]} \beta_{[x]} \delta_{[z]} \beta_{[x, z, y]} \Gamma_{[x, z, y]} = \delta_{[y]} \delta_{[x]} \delta_{[z]} \delta_{[x]} \delta_{[z]} \Gamma_{[x, z, y]} \Gamma_{[y, z, x]} \quad (10)
\]

The perfect unshuffle permutation expressed in (10) is immediate from the operator strings in (12) and (13). \( \beta_{[x, z, y]} \delta_{[x]} \delta_{[z]} \) performs the perfect unshuffle permutation of the \( 2^r \) butterflies of \( 2^q \) data items processed sequentially by each PE. Consequently, we can use the same solution as in the single processor case (see (7) and Fig. 2(a)), with the only difference being that in this case the length of the FIFO queues will be \( 2^r - 2^q \) cells, considering the PS section as the only stage of the internal pipeline of the PE. \( \Gamma_{[x, z, y]} \) performs the perfect unshuffle permutation of the outputs of the PE's. Its hardware implementation will use an external interconnection network determined by operator \( \Gamma_{[x, z, y]} \) applied on the dimensions \( [z, y] \). The \( y \)th output path of the \( z \)th PE will be connected to the \( z \)th input path of the \( z \)th PE, where \([x, z, y] = \Gamma_{[x, z, y]} [x, z, y] \). Fig. 4 shows the connections of the PE's for the example of Fig. 3. The number of input and output paths of each PE is only a function of the radix of the transform and not of the number of PE's in the column.
Summarizing, we have introduced the partition of DIT successive doubling algorithm in a natural way by means of the decomposition of permutation \( \sigma \) in a perfect unshuffle which is internal \((\text{cycle}, \text{path})\) and another which is external \((\text{nPE}, \text{path})\) to the PE's. In the particular case of one column with a single PE the whole unshuffle will be internal, we obtain the solution of the previous section \((\text{sequence } (7))\), and the FIFO queues will have a length of \( \text{PE's} \). In the particular case of one column with a single PE the whole unshuffle will be internal, we obtain the solution of the pre-

(Figs. 2(a) and 4) computes the DIT successive doubling algorithm in a natural way by means of the decom-

where the application order for the operators is from left to right. The proof is immediate, using definitions \((15)\) and \((16)\).

Equations \((17)\) and \((18)\) guarantee the decomposition of the perfect shuffle permutation of the three dimensional representation of each data item's index. Consequently, we can establish the following result:

**Theorem 2:** DIF successive doubling radix \( r \) and constant geometry algorithm of a sequence of \( N \) data items \((N = r^n)\) can be carried out in a column of \( Q \) PE's \((Q = r^k, 0 \leq q < n)\) which implement (hardware translation) each stage the following operator string:

\[
B_{r(k)}[x, z, y] = \frac{r^k}{\phi_{r(k)}}[x, z, y] (19)
\]

where \( r = \log_r r \), the operators are applied from left to right and we consider the \([\text{cycle}, \text{nPE}, \text{path}]\) interpretation of the three dimensional representation of the index of each data item.

Its proof is similar to that of Theorem 1. Fig. 5 shows the internal structure of the PE where we have included two FIFO queues to facilitate the overlapping execution of different transform stages. These queues implement the operator string \( r^k[\phi_{r(k)}] \), whereas the partial perfect shuffle operator \( \mu_{[r]} \) determines the interconnection network of the PE column.

V. Final Remarks

Parallelism has long been considered an important solution to fast transform algorithms [1]. The solution is in the design of multiprocessor architectures which permit the exploitation of the inherent parallelism of these algorithms, but we must consider three important problems: partitioning of the algorithm, synchronization between PE's, and pipeline design of the PS section.

The successive doubling method is an efficient procedure for the design of fast algorithms for orthogonal transforms, presenting a high spatial parallelism in the calculation stages and an inherent sequentiality between them. Consequently, the appropriate architecture will be a rectangular array of a pipeline of PE columns. The application specific multiprocessor we have proposed in this work efficiently solves the problems mentioned before. The result is a constant geometry systolic architecture. The geometry is determined by the perfect unshuffle (or perfect shuffle) permutation of the data generated in each stage of the DIT (or DIF) successive doubling algorithm.

The constant character of the geometry permits the implementation of the data flux of the successive doubling algorithm by means of hardwired control. That is, the PE's do not need address arith-
metic units to locate the data. Addressing is inherent to the evolution of the data in the FIFO queues of the RS section and the external interconnection network. Moreover, the partitioning of the algorithm appears in a natural way when this permutation (perfect unshuffle or shuffle) is decomposed into a string of elementary permutations which can be implemented electronically: decimation, concatenation, and partial perfect unshuffle (DIT) or shuffle (DIF). Also, we have chosen the systolic operating mode, which is an effective synchronization method. Finally, a pipelined PS section of the PE will improve the performance of the proposed architecture (see [2] and [12]).

REFERENCES