Journal of VLSI Signal Processing Systems 13, 57–66 (1996)

FFT Computation with Linear Processor Arrays Using a Data-Driven Control Scheme

DING-MING KWAI AND BEHROOZ PARHAMI
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA

Received March 30, 1995; Revised March 28, 1996

Abstract. For a large number N of data points, linear FFT arrays consisting of $\Theta(\log_2 N)$ processors provide significant economy in hardware. In this paper we discuss the radix-2 decimation-in-frequency Cooley-Tukey algorithm implemented on linear arrays, thereby allowing a continuous real-time application using a word-serial input data stream to the linear arrays. In order to avoid memory access and data path switching under central control, we present a novel data-driven scheme permitting the proposed linear arrays to correctly operate on arbitrarily arriving signal sequences. This distributed control scheme incorporates a control signal propagated with the data signals, in the form of a tag attached to data items. The tag provides control information to initiate the access to the memory containing the coefficients and to select an appropriate data path so that regular data flow can be achieved within the linear array. The cascade structures are well suited for the computation and can operate in pipeline fashion at extremely high data rates. The proposed data-driven control scheme can be used with both synchronous and asynchronous (wavefront) processor arrays.

1. Introduction

The Discrete Fourier Transform (DFT) is widely used in digital signal processing and its realization in hardware has been proposed by many researchers (See, e.g., [1]). For an input signal sequence $\boldsymbol{x} = [x(0), x(1), \ldots, x(N-1)]^T$ and its transformed version $\boldsymbol{y} = [y(0), y(1), \ldots, y(N-1)]^T$, the N-point DFT can be written in matrix form as

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N^1 & \omega_N^2 & \cdots & \omega_N^{(N-1)} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{(N-1)} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}$$

where the entries of the $N \times N$ matrix $\mathbf{W}$ are defined by $[\mathbf{W}]_{ij} = \omega_N^{ij}$ $(0 \le i, j \le N-1)$ with ω_N being the primitive Nth root of unity. Implementation in pipeline fashion is a natural extension for the on-line DFT computation. Kung and Leiserson [2] thus suggested an array of inner-product-step processors connected linearly for a direct matrix-vector multiplication. Approaches along this line, however, are limited by their $O(N^2)$ operation count and generally lead to a total number $\Omega(N)$ of processors; implementation cost thus grows at least linearly with N.

Fast Fourier Transform (FFT) algorithms based on sparse factorization of the DFT matrix $\mathbf{W}$ lead to more efficient implementations. Suppose that the number N of sampled points is initially (or is adjusted upwards to) an integer power of 2, i.e., $N = 2^n$. We can decompose the matrix $\mathbf{W}$ into a product of sparse matrices, $\mathbf{W}_n \cdots \mathbf{W}_2 \mathbf{W}_1$, where each $\mathbf{W}_i (1 \le i \le n)$ has exactly two nonzero entries per row. We then require only $O(N \log_2 N)$ operations by exploiting the sparsity of $\mathbf{W}_i$. Different FFT algorithms have been

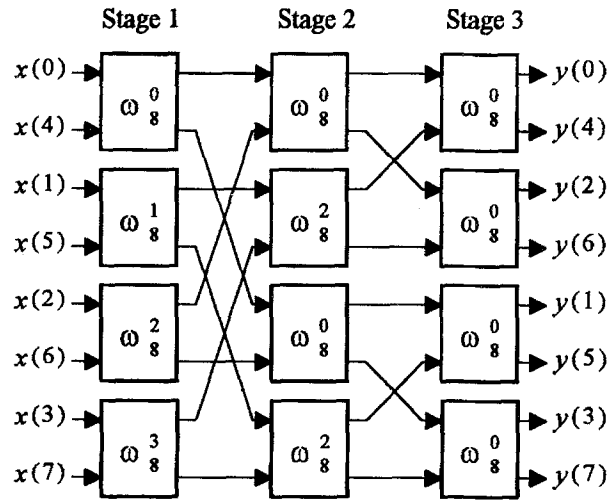


Figure 1. Butterfly FFT network.

developed to operate on different arrangements of data items by using the particular properties of $\mathbf{W}_i$ [3]. Although our scheme applies to other cases as well, the radix-2 decimation-in-frequency Cooley-Tukey algorithm is discussed here due to the fact that a continuous real-time application is allowed by its word-serial input data stream. This arrangement permits data items to be processed in the order of arrival without requiring the inputs to be reassembled into a stream of consecutive data items. A signal flow graph for computing 8-point decimation-in-frequency FFTs is given in Fig. 1. Because of the bit-reversed order, the output vector $\mathbf{y}$ and the DFT matrix become

$$\begin{bmatrix} y(0) \\ y(N/2) \\ y(N/4) \\ \vdots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N^{N/2} & \omega_N^{2(N/2)} & \cdots & \omega_N^{(N-1)(N/2)} \\ 1 & \omega_N^{N/4} & \omega_N^{2(N/4)} & \cdots & \omega_N^{2(N-1)(N/4)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{(N-1)} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}$$

In this paper, we will focus on linear FFT arrays consisting of $\Theta(\log_2 N)$ processors. For a large number N of points, these implementations provide significant economy in hardware. Advantages of linear arrays include low interconnection density for VLSI layout, bounded I/O requirements, ease of global clocking in the presence of signal propagation delays [4], and amenability to (partial) scan design [5] to provide good controllability and observability for testing purposes.

2. Space-Time Mapping

A straightforward way of implementing an FFT network is derived directly from the flow graph of the algorithm. A fully parallel $\log_2 N$-stage network, each stage consisting of $N/2$ butterfly processors, can be obtained (see Figs. 1 and 2). There are two types of operations in these networks. One is a butterfly operation, in which pairs of processors exchange data items and compute weighted sums and differences of the data items exchanged. The other is a permutation operation that alters the order of data items to one that is lexically ascending. It is usually achieved by interconnections among processors.

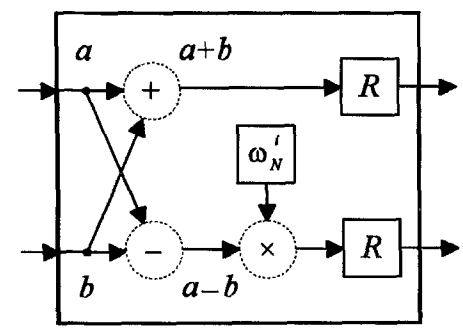


Figure 2. Butterfly processor (B). The square box R represents one word of storage.

In what follows, the butterfly processors will be assumed to have bit-parallel inputs; i.e., all the bits of the two operands a and b in Fig. 2 arrive simultaneously. Since our array designs have only one primary input and one primary output, the I/O requirements are modest and well within the packaging constraints of current VLSI technology. Similarly, with multiple processors placed on a chip, the area overhead due to bit-parallel inter-processor links is not a major concern in view of their regularity and locality.

In terms of matrix factorization, the implementation is derived by decomposing the DFT matrix $\mathbf{W}$ into a product of sparse matrices $\mathbf{W}_i$ $(1 \leq i \leq \log_2 N)$, each corresponding to one stage of the butterfly network. Pairs of nonzero entries in rows and in columns of $\mathbf{W}_i$ are separated by $2^{n-i} - 1$ zero elements. Additionally, nonzero elements appear on only three diagonals of $\mathbf{W}_i$. Since always two input data items are used to compute two output data items simultaneously, the network combines the two input data items in parallel through a butterfly processor. Consequently, $N/2$ butterfly processors are required at each stage. As an example, an 8-point decimation-in-frequency DFT matrix $\mathbf{W}$ can be decomposed into a product of three sparse matrices $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$, such that $\mathbf{W} = \mathbf{W}_3\mathbf{W}_2\mathbf{W}_1$, where

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \omega_8^0 & 0 & 0 & 0 & \omega_8^4 & 0 & 0 & 0 \\ 0 & \omega_8^1 & 0 & 0 & 0 & \omega_8^5 & 0 & 0 \\ 0 & 0 & \omega_8^2 & 0 & 0 & 0 & \omega_8^6 & 0 \\ 0 & 0 & 0 & \omega_8^3 & 0 & 0 & 0 & \omega_8^7 \end{bmatrix}$$

$$\mathbf{W}_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \omega_8^0 & 0 & \omega_8^4 & 0 & 0 & 0 & 0 & 0 \\ 0 & \omega_8^2 & 0 & \omega_8^6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & \omega_8^0 & 0 & \omega_8^4 & 0 \\ 0 & 0 & 0 & 0 & 0 & \omega_8^2 & 0 & \omega_8^6 \end{bmatrix}$$

$$\mathbf{W}_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \omega_8^0 & \omega_8^4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \omega_8^0 & \omega_8^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \omega_8^0 & \omega_8^4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \omega_8^0 & \omega_8^4 \end{bmatrix}$$

Considering the total of $(N/2)\log_2 N$ processors and $2N$ I/O ports, a feasible realization of the butterfly FFT network in hardware is limited to small N, say $N = 8$ [6]. Such a parallel design is clearly impractical when the number of data points becomes very large. Most FFT array designs reallocate and reschedule the operations in the butterfly network onto a smaller number of processors. This can be viewed as a space-time mapping of the flow graph. One possibility is to map the $\log_2 N$ stages onto $\log_2 N$ processors [7]. Another approach is to decompose the multiply-subtract-add (butterfly) operation in each processor into two separate multiply-add (inner-product-step) operations [8]. As we will see in Section 4, the decomposition actually creates three, instead of two, inner-product-step processors per stage.

The rescheduling induces additional storage (in the form of shift registers) inserted between processors along the signal lines to regulate the pipelined communications. The overall cost is therefore determined by two important elements in such a design: the computation modules and the additional storage. The coefficients ω_N^i are assumed to be stored in a circular queue [9], or are located in a memory, with a counter linked to the global clock determining the address before starting the computation. Extending the design to a larger number of data points may be based on reusing the same hardware, although the memory containing the ω_N^i coefficients should be large enough for such an extension.

An important issue in the pipelined design is that the coefficient memory must be initialized to the starting address with the arrival of each new input signal sequence. In view of the periodic resetting feature of shift registers and binary address counters, this will not cause any problem as long as input sequences are continuously fed to the FFT array. However, when input sequences are separated by arbitrary gaps, or idle cycles, it may not be desirable to wait until the current sequence is completely flushed out and then resetting the entire array.

In order to avoid the complexity and added delay of data path switching under central control, we propose a data-driven scheme that allows simple control and provides scalability for the structures. The basic idea of the distributed scheme is to incorporate a set of control signals, propagated with data signals, in the form of a tag attached to the data items [10]. This control tag is then used to activate the counter of the coefficient memory and/or to select an appropriate processor state, hence leading to regular data flow within the linear array. We apply the idea of distributed or data-driven control to

two different cascade implementations in Sections 3 and 4, briefly comparing the two in the concluding Section 5.

3. Feedback Linear Array

The design proposed by Groginsky and Works in 1970 [11] is derived by vertically projecting the FFT flow graph. Figure 3 shows the space-time diagram for computing 8-point FFTs, in which nodes represent butterfly operations scheduled at each processor and edges represent interconnection paths with delays required for data items to move to the next processor. The nodes are arranged in such a way that the *causality* requirement can be satisfied, meaning that an outgoing data item from a node takes at least one time step to reach the next node and two incoming data items (from different nodes) used at the same node arrive simultaneously.

As we can see in the space-time diagram, there are only two input patterns for each processor. For example, the first and the second nodes in the central column (corresponding to processor P_2) receive one input data item delayed by two additional time steps and the other without any delay, while the third and the fourth nodes in the same column receive one input data item delayed by six time steps and the other by four time steps.

An interesting feature of this design is that the shift registers used to delay the input data items can be reused for the generated data items. The data path is established by a 2-state switch appended to each butterfly processor as shown in Fig. 4. In the "through" state, the processor performs the butterfly operation on the input data items received from its primary input and the shift register; two output data items are sent, one to the next processor and the other back to the shift register. In the "cross" state, the processor feeds the input data item directly to the shift register while forwarding the stored output data item from the shift register to the next stage. The pipelining operation can be regarded as two phases of processor activities: computation and transmission, interleaved in consecutive time steps.

The linear array is constructed as follows. One of the two outputs of each butterfly processor is connected to the input of a shift register of appropriate length. The shift register's output is connected to one of the butterfly processor's inputs, thus forming a feedback loop. The remaining primary input and output of the butterfly

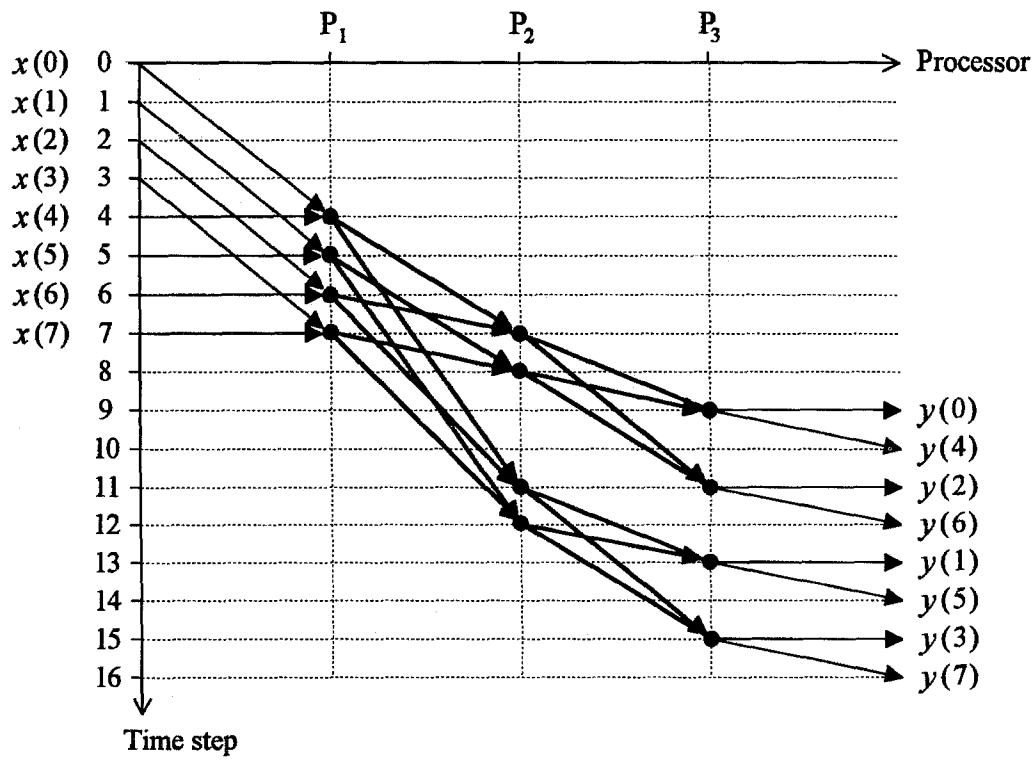


Figure 3. Space-time diagram for the feedback FFT linear array.

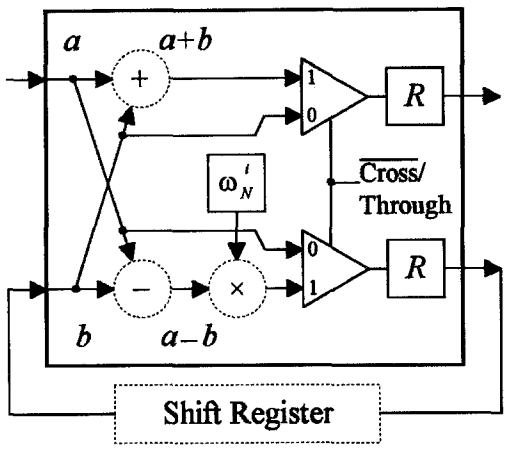


Figure 4. Butterfly processor augmented with data path selection logic.

processors are used to connect the latter into a linear array. The feedback linear array for computing 8-point FFTs is shown in Fig. 5. The length of the shift register for each processor P_i is 2^{n-i} which corresponds to the distance (difference of indices) between two input data items entering the same node in the flow graph or in the same row of the sparse matrix $\mathbf{W}_i$.

It is clear from the space-time diagram that the utilization of the butterfly processors is only 50%. Operationally, this is not a serious drawback since the pipelined operation and throughput are maintained. On the other hand, most concurrent error-detecting schemes actually take advantage of the idle time steps to do a duplicate computation for on-line testing. Thus, the above structure can use the excess node capacity to perform the duplicate computation efficiently.

A drawback of this structure, as originally proposed, is that the control of the switch and the address counting to access the coefficient memory require a central controller. The central controller becomes a bottleneck that may limit the scalability and speed of the linear array. The problem can be avoided by appending a ("routing") tag [12] to each data item, thereby allowing a processor to perform its operation based on the control information provided by the tag. We can assign a tag bit "0" to instruct the processor to switch to the "cross" state and a tag bit "1" to switch to the "through" state. Note that the processor performs a butterfly operation only when its switch is in the "through" state.

The tag assigned to each data item is simply its sequence number (i.e., $0, 1, \ldots, N-1$ in binary form). Therefore, for an N-point FFT, $\log_2 N$ bits are needed to encode the control information. We can further divide the control information into two fields: the most significant bit is used to select the data path which is either "cross" or "through" and the remaining bits are used to address the coefficient memory. By observing the flow graph in Fig. 1, it is clear that for processor P_i the size of the memory is 2^{n-i} (which is equal to the length of the shift register); thus, $n-i$ bits are sufficient to address the memory. To achieve this, we remove the most significant bit after the processor has generated the output data items and append the resulting tags to the generated data items. The operations performed by the processor are then determined solely by the tag received from the primary input.

Figure 6 shows the timing table corresponding to an 8-point FFT under such a control scheme. The timing table contains data items with tags at each location of the linear array, the time steps increasing from top to bottom. The intermediate signal sequences are shown as $\boldsymbol{X} = \mathbf{W}_1 \boldsymbol{x}$, $\boldsymbol{Y} = \mathbf{W}_2 \boldsymbol{X}$, and $\boldsymbol{y} = \mathbf{W}_3 \boldsymbol{Y}$. To make the two phases of operation distinct, we have shaded the generated data items in the shift registers. Each processor waits until it receives an input data item with the most significant tag bit equal to 1 and then performs a butterfly operation using the coefficient ω_8^i selected by the remaining tag bits. The two generated data items are given tags that are obtained from input tags with the most significant bit removed. A data item with a small

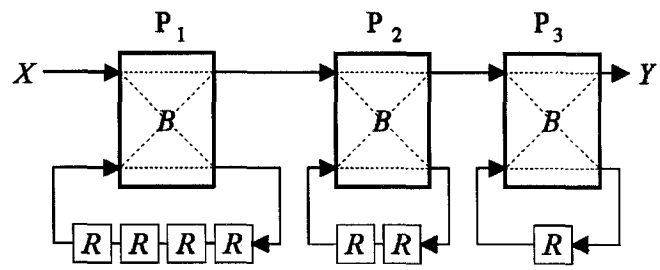


Figure 5. Feedback linear array, for computing 8-point FFTs. The butterfly processors B are augmented with data path selection logic. Each square box R represents one word of storage.

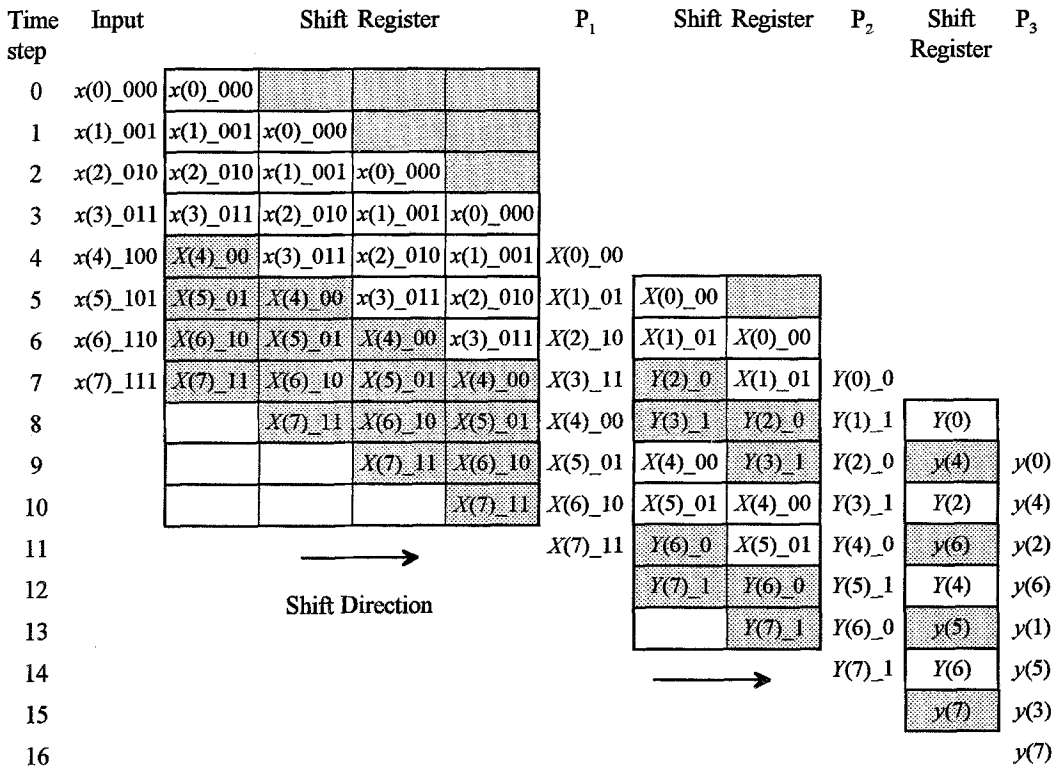


Figure 6. Timing table for computing 8-point FFTs on the feedback linear array.

sequence number is sent directly to the next processor to the right while a data item with a large sequence number is fed back to the shift register. The stored data items are then propagated to the primary output and the lexically ascending order of data items is maintained.

An elegant feature of this control scheme arises from the fact that each processor performs the butterfly operation only when it receives a tag with the most significant bit equal to 1. Thus if we set all the tag bits to 0, the shift registers naturally form a scan path, making it easy to incorporate a built-in self-test (BIST) mechanism into the linear array [6]. Equally simple is the insertion of an arbitrary number of dummy data items between two input signal sequences with all tags set to 0. This is of great value when input signal sequences are separated in time, since regardless of the inter-sequence gaps, the FFTs are computed correctly without mutual interference.

4. Concatenated Unidirectional Linear Array

Boriakoff [8] has recently proposed a concatenated structure consisting of $\log_2 N$ (one uni-directional and $\log_2 N - 1$ bidirectional) linear arrays of inner-product-step processors which takes advantage of the sparsity of decomposed matrices in a different way. As noted in Section 2, the nonzero entries of each sparse matrix are located on three diagonals and this leads to the requirement for three inner-product-step processors per stage. In a manner similar to pipeline chaining in vector processors, the design cascades the linear arrays, each performing a matrix-vector multiplication, such that the output of one linear array is fed to the input of the next array. Here, we adopt the same decomposition technique, but make all linear arrays unidirectional as shown in Fig. 7. Figure 8 shows the space-time diagram for computing 8-point FFTs.

Recall that pairs of entries in the sparse matrix $\mathbf{W}_i$ are separated by 2^{n-i} places in both row and column dimensions. The linear array for the multiplication of $\mathbf{W}_i$ correspondingly contains 2^{n-i} delay elements (registers) between a pair of processors in order to regulate the arrival time of input data items. Use of unidirectional linear arrays provides the added flexibility of being able to insert additional registers in the signal lines between processors without affecting their schedules. These registers, functioning as repeaters in

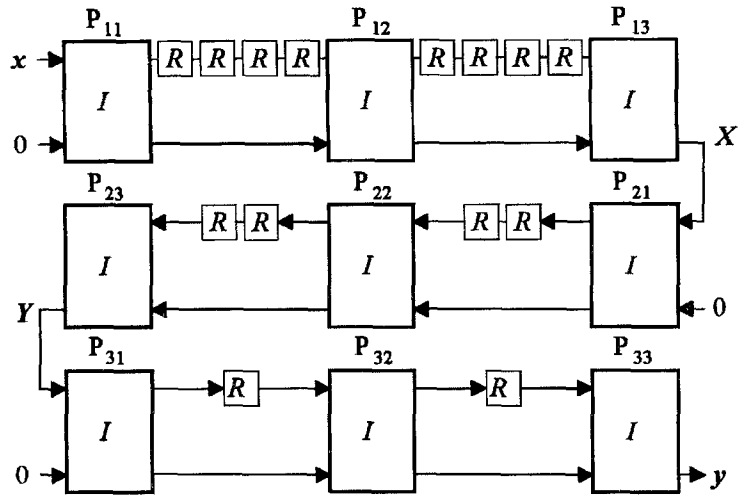


Figure 7. Concatenated unidirectional linear array for computing 8-point FFTs. Segments consisting of three inner-product step processors I are connected in a snakelike arrangement. Each square box R represents one word of storage.

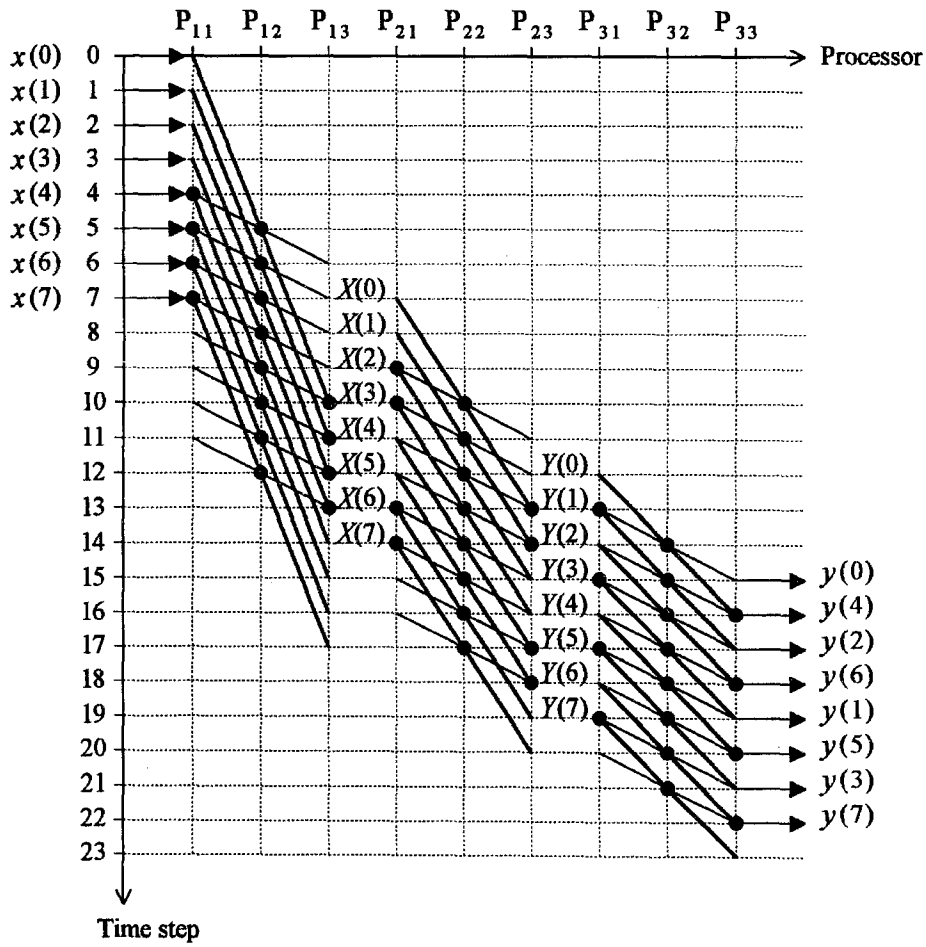


Figure 8. Space-time diagram for the concatenated unidirectional FFT linear array.

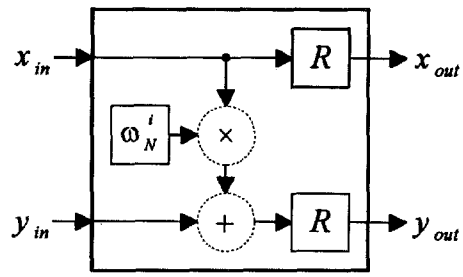


Figure 9. Inner-product-step processor (*I*).

pipelining communication, help improve the transmission rate when long inter-chip or off-chip interconnection wires are required [13]. More importantly, as we will see later, the structure can be driven by a simple control scheme.

An inner-product-step processor that computes $y_{\text{out}} = \omega_N^i \times x_{\text{in}} + y_{\text{in}}$ is shown in Fig. 9, where the three inputs ω_N^i, x_{in}, and y_{in} are complex numbers stored in registers actuated by a global clock signal. The value of x_{out} is the same as x_{in}; hence, this value simply propagates forward in each segment's shift registers. The hardware cost for three inner-product-step processors is higher than that for a single butterfly processor, considering the number of complex multiplier and adder modules used (the disparity is somewhat reduced if we use an array implementation combining the multiplication and addition operations [14]). The significant difference is due to the property $\omega_N^i = -\omega_N^{i+N/2}$ that results in only one complex multiplication (by ω_N^i) in the butterfly processor. The same butterfly computation turns into four complex multiplications (by 1, 1, ω_N^i, and $\omega_N^{i+N/2}$) and is computed at three different inner-product-step processors in the unidirectional linear array. Unlike the butterfly processor, the two multiplications by 1 do not correspond to any saving in hardware, given the path through which data must travel.

We next apply our data-driven control scheme to the unidirectional linear array. The tags are used to access the coefficient memory, since no data path switching is necessary in this structure. The size of the memory for each processor P_{ij} ($1 \le i \le n$, $1 \le j \le 3$) at segment i is 2^{n-i+1}. The memory containing the coefficients can be implemented as a circular queue or as an addressable memory with a counter. The control information in the tag is used to reset the counter to its starting value. The timing table in Fig. 10 shows the execution sequence for the computation $Y = \mathbf{W}_2 X$ that occurs in the middle segment of Fig. 7 (where the tag is attached to the output data stream Y). Because they share the same construction, the top and bottom arrays have similar operations. We can assign a tag bit 1 to reset the counter and a tag bit 0 to increment the counter by

Time step	P_{23}	Shift Reg.	P_{22}	Shift Reg.	P_{21}
0					0, $X(0)$
1				$\quad$ | $X(0)$	0, $X(1)$
2				$X(0)$ | $X(1)$	1, $X(2)$, $Y(0)\_1$
3			1, $X(0)$, $Y(0)\_1$	$X(1)$ | $X(2)$	1, $X(3)$, $Y(1)\_0$
4	0, $Y(0)\_1$	$\quad$ | $X(0)$	1, $X(1)$, $Y(1)\_0$	$X(2)$ | $X(3)$	0, $X(4)$, $Y(2)\_0$
5	0, $Y(1)\_0$	$X(0)$ | $X(1)$	ω_8^4, $X(2)$, $Y(2)\_0$	$X(3)$ | $X(4)$	0, $X(5)$, $Y(3)\_0$
6	ω_8^0, $X(0)$, $Y(2)\_0$	$X(1)$ | $X(2)$	ω_8^6, $X(3)$, $Y(3)\_0$	$X(4)$ | $X(5)$	1, $X(6)$, $Y(4)\_1$
7	ω_8^2, $X(1)$, $Y(3)\_0$	$X(2)$ | $X(3)$	1, $X(4)$, $Y(4)\_1$	$X(5)$ | $X(6)$	1, $X(7)$, $Y(5)\_0$
8	0, $X(2)$, $Y(4)\_1$	$X(3)$ | $X(4)$	1, $X(5)$, $Y(5)\_0$	$X(6)$ | $X(7)$	0, $Y(6)\_0$
9	0, $X(3)$, $Y(5)\_0$	$X(4)$ | $X(5)$	ω_8^4, $X(6)$, $Y(6)\_0$	$X(7)$ | $\quad$	0, $Y(7)\_0$
10	ω_8^0, $X(4)$, $Y(6)\_0$	$X(5)$ | $X(6)$	ω_8^6, $X(7)$, $Y(7)\_0$	←	
11	ω_8^2, $X(5)$, $Y(7)\_0$	$X(6)$ | $X(7)$		Shift Direction	
12	0, $X(6)$	$X(7)$ | $\quad$			
13	0, $X(7)$	←			

Figure 10. Timing table for computing $Y = \mathbf{W}_2 X$ on the middle segment of the unidirectional linear array.

one. The tag bit stream (1 followed by $2^{n-i+1} - 1$ 0s) is propagated through the linear array and inserted every 2^{n-i+1} cycles. The three inputs ω_N^i, x_{in}, and y_{in} for each inner-product-step processor are placed together in the same column.

Note that the tag bit 1 also indicates the arrival of a new signal sequence. If the next input signal sequence is delayed, we may insert dummy data items (with "0" tags attached) to the linear array. The processor does not reset the memory counter until a tag "1" appears. Once the input signal sequence arrives, the tag bit 1 propagates along the linear array and resets each memory counter in a pipeline fashion.

5. Conclusions

FFT arrays are generally treated as special-purpose engines which contain processors, memories, and communication links. Previous designs addressed the issue of how to connect processors with a suitable number of delay elements in the communication links; however, they tended to leave the problem of memory access or data path switching to an unspecified controller. When two sequences coexist in the same array, the memory access and data path switching scheme must remain correct for each of them. Hence, the task of the controller may not be trivial, especially for real-time applications.

In this paper, we have presented two linear array structures for computing N-point FFTs based on sparse factorization of the Cooley-Tukey matrix. In order to avoid memory access and data path switching under central control, we proposed a data-driven scheme allowing the linear arrays to be controlled simply and precisely. The basic idea of the distributed scheme is to incorporate a set of control signals, in the form of a tag attached to the data items, that propagates with data signals. The control tag is used to access the appropriate element of the memory containing the coefficients and/or to direct the processor to a desirable state so that regular data flow can be achieved within the linear array.

The hardware cost and performance of these two implementations are summarized in Table 1. For $N = 1024$, the feedback linear array requires 10 butterfly processors and 10 tag bits to set the data path and to address the coefficient memory. The storage requirement for both the shift registers and coefficient memories is 1023 words. The delay time from input to the emergence of the first transformed data is 1033 time steps. The latency for the whole transform is 2056 time steps; only 50% of these steps are used for computation. The concatenated unidirectional linear array, on other hand, requires more (30) inner-product-step processors but only 1 tag bit to activate the circulating coefficient memory. The storage requirement for the shift registers and the coefficients is higher (2046 words and 6138 words, respectively). It also implies slightly longer delay and latency (1053 time steps and 2076 time steps, respectively). The advantages of the concatenated structure lie primarily in simple control mechanism and ease of expansion.

Table 1. Hardware cost and performance of two FFT arrays.

	Feedback linear array	Concatenated unidirectional linear array
Number of processors	$\log_2 N(B)$	$3\log_2 N(I)$
Number of tag bits	$\log_2 N$	1
Size of storage	$N-1$	$2(N-1)$
Size of coefficient memory	$N-1$	$6(N-1)$
Delay time to first result	$N+\log_2 N-1$	$N+3\log_2 N-1$
Latency of completion	$2N+\log_2 N-2$	$2N+3\log_2 N-2$

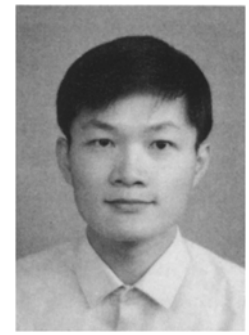
Obviously, introduction of fault tolerance capability into the FFT arrays is also important. We have examined some possibilities for incorporating built-in testing and concurrent error detection for a feedback linear array. Intuitively, the unidirectional data flow of the concatenated linear array design simplifies the provision and analysis of fault tolerance mechanisms, since any permanent or extended transient fault is exercised only once and thus is likely to have a much more localized effect. However, comprehensive treatments of error detection, location, and reconfiguration issues remain for future research.

In closing, we note that our data-driven control scheme can be used with synchronous systolic arrays [2] or their asynchronous counterparts known as wavefront arrays [15, 16]. Wavefront arrays are also "data-driven". However, the data-driven aspect of wavefront arrays only applies to the activation of processors; i.e., it is the availability of input data, rather than the rising or falling edge of a clock signal, that triggers the processor's computation. Our method of embedding instructions within the data stream augments the above capability of wavefront arrays and can also be applied, with equal ease, to synchronous systolic arrays. The examples presented in this paper involved linear arrays in which the single input data stream carried the needed

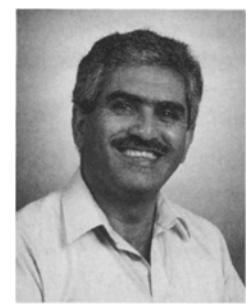
instructions. With multi-dimensional arrays, or multiple data streams within linear arrays, the instructions can be encoded in various ways by attaching suitable indicators or tags to the various data streams. The resultant optimization problem to obtain the most efficient tagging scheme is quite interesting [10].

References

1. C.D. Thompson, "Fourier transform in VLSI," *IEEE Transactions on Computers*, Vol. C-32, pp. 1047–1057, 1983.
2. H.T. Kung and C.E. Leiserson, "Algorithms for VLSI processor arrays," in *Introduction to VLSI Systems*, C. Mead and L. Conway (Eds.), Reading, MA: Addison-Wesley, pp. 271–292, 1980.
3. C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, Philadelphia, PA: Society for Industrial and Applied Mathematics, 1992.
4. A.L. Fisher and H.T. Kung, "Synchronizing large VLSI processor arrays," *IEEE Transactions on Computers*, Vol. C-34, pp. 734–740, 1985.
5. R. Gupta, R. Gupta, and M.A. Breuer, "The BALLAST methodology for structured partial scan design," *IEEE Transactions on Computers*, Vol. 39, pp. 538–544, 1990.
6. K. Yamashita et al., "A wafer-scale 170000-gate FFT processor with built-in test circuits," *IEEE Journal of Solid-State Circuits*, Vol. 23, pp. 336–342, 1988.
7. J. Choi and V. Boriakoff, "A new linear systolic array for FFT computation," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, Vol. 39, pp. 236–239, 1992.
8. V. Boriakoff, "FFT computation with systolic arrays, a new architecture," *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, Vol. 41, pp. 278–284, 1994.
9. M. Aloqeely and C.Y. Chen, "Sequencer-based data path synthesis of regular iterative algorithms," *Proc. 31st ACM/IEEE Design Automation Conf.*, pp. 155–160, 1994.
10. D.-M. Kwai and B. Parhami, "A data-driven control scheme for linear processor arrays," Submitted for publication.
11. H.L. Groginsky and G.A. Works, "A pipeline fourier transform," *IEEE Transactions on Computers*, Vol. 19, pp. 1015–1019, 1970.
12. S. Sarkar and A.K. Majumdar, "Fast fourier transform using linear tagged systolic array," *IEEE Region 10 Conf. Computer and Communication Systems*, pp. 289–293, 1990.
13. D. Audet, Y. Savaria, and N. Arel, "Pipelining communications in large VLSI/ULSI systems," *IEEE Trans. Very Large Integration (VLSI) Systems*, Vol. 2, pp. 1–10, 1994.
14. M.O. Ahmad and D.V. Poornalah, "Design of an efficient VLSI inner-product processor for real-time DSP applications," *IEEE Transactions on Circuits and Systems*, Vol. 36, pp. 324–329, 1989.
15. S.Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proceedings IEEE*, Vol. 72, pp. 867–884, 1984.
16. S.Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice-Hall, 1988.



Ding-Ming Kwai received the B.S. degree from the National Cheng Kung University, Tainan, Taiwan, in 1987, and the M.S. degree from the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, in 1989. He was with the Chung Cheng Institute of Technology, Taoyuan, Taiwan, as a reserve officer from 1989 to 1991 and with the Hualon Microelectronics Corporation, Hsinchu, Taiwan, as a design engineer from 1991 to 1993. He is currently pursuing the Ph.D. degree in Computer Engineering at the University of California, Santa Barbara. His research interests include parallel processing, VLSI architectures, and fault-tolerant computing.



Behrooz Parhami received the Ph.D. degree from University of California, Los Angeles, U.S.A., in 1973. During his 14-year affiliation with Sharif University of Technology, Tehran, Iran, he carried out research in several areas of computer architecture and was also instrumental in national projects in technology transfer, educational planning, curriculum development, and standardization. He was the principal founder of the Informatics Society of Iran and served as its first President and Editor-in-Chief for 5 years, while at the same time guiding the IEEE Iran Section through a turbulent decade. Since 1988, he has been Professor of Computer Engineering at University of California, Santa Barbara, with research interests in computer arithmetic, parallel processing, and fault-tolerant computing. His current projects in these areas emphasize architectures and algorithms for scalable massively parallel systems and their VLSI implementations. Dr. Parhami is a Fellow of the British Computer Society, a Senior Member of IEEE, a Distinguished Member of the Informatics Society of Iran, and a Member of the Association for Computing Machinery. He received the IEEE Centennial Medal in 1984.