A CLASS OF PARALLEL ARCHITECTURES FOR FAST FOURIER TRANSFORM

Chi-Hsiang Yeh and Behrooz Parhami
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA
{yeh@simd.| parhami@}ece.ucsb.edu

ABSTRACT

We propose a new class of parallel architectures called unfolded swapped networks (USN) for fast Fourier transform (FFT) and related problems. The VLSI area of a suitably constructed $N(\log_2 N + o(\log N))$-node USN is no more than $N^2 + o(N^2)$, which is smaller than the best known result for a $\log_2 N$-dimensional butterfly network. USNs can be constructed using small butterfly modules, each built on a chip, and requires fewer pins than a similar-sized butterfly network by a factor of $\Theta(\log N)$. N-point FFT can be executed on a USN at a speed comparable to a butterfly network, assuming constant link delay; it can be executed on a USN considerably faster than on a butterfly when link delay increases with length and/or when inter-chip data transfers are much slower than intra-chip ones.

1. INTRODUCTION

Fast Fourier transform (FFT) is important for numerical computation and digital signal processing. It has been applied to the solution of differential equations, convolution, digital filters, image processing, speech recognition, and many others areas. Since FFT requires intensive computation, there has been a great deal of interest in implementing FFT on parallel computers, and many parallel architectures that support efficient FFT algorithms have been proposed [2, 3, 6, 9].

In this paper, we propose a new class of parallel architectures called *unfolded swapped networks (USNs)*, and present efficient FFT algorithms that run efficiently on USNs. We show that N-point FFT can be executed on a USN at a speed comparable to a *butterfly network* [2], assuming constant delay for data exchange via a link; we also show that FFT can be executed on a USN considerably faster than on a butterfly network when link delay increases with length, and inter-chip communications are more expensive than intra-chip ones. We then present a simple layout for USNs. The VLSI area of a USN is smaller than those of known layouts for a similar-sized butterfly network under the commonly used square grid model. USNs can also be constructed with small butterfly (or any other FFT computation) modules, each built on a chip, and requires fewer pins than a similar-sized N-node butterfly network by a factor of $\Theta(\log N)$. Since pin limitations become a major constraint in the implementations of highly parallel systems [1], this feature implies a considerable reduction in the number of chips, and thus, the hardware cost of USNs. In effect, the construction of USNs provides a systematic method to synthesize large FFT computation networks from various smaller modules.

2. UNFOLDED SWAPPED NETWORKS

Recursive hierarchical swapped networks (RHSNs) [9] form a class of efficient interconnection networks that have small node degrees and a variety of fast algorithms. The node degrees of RHSNs are, however, not constant, so that they may not be suitable for some applications. In this section, we present the definition of URHSN, which can be viewed as unfolding the structure of an RHSN along routing paths. As a natural consequence, URHSNs can efficiently emulate most algorithms developed for RHSNs and inherit many advantages of RHSNs.

We first introduce *butterfly networks*, which can be used as a basic module for building a USN. Algorithms developed for butterfly networks can usually be efficiently emulated on USNs having the same number of rows.

2.1. Butterfly Networks

An n-dimensional butterfly network, denoted by B_n, has $(n+1)2^n$ nodes and $n2^{n+1}$ links (see Fig. 2a). A node in the butterfly network corresponds to a pair (Y, y), where Y is an n-bit binary *row* number and $y \in [0, n]$ denotes the node *dimension* or *column*. Nodes (U, u) and (V, v) are neighbors if and only if $u = v + 1$ or $v = u + 1$ and either

a) $U = V$, or
b) U and V differ only in bit $\min(u, v)$.

Butterfly networks are suitable for the efficient implementation of many important algorithms, including FFT, sorting, and other normal hypercube algorithms [2]. In this paper, we will focus on USNs that use small butterfly networks as basic modules, and show that such USNs can execute FFT efficiently and have several important advantages over a similar-sized butterfly network.

2.2. Formal Definition of USNs

In what follows, we denote the basic building blocks (nucleus graph) for USNs as $G = ((\mathcal{V}_\mathcal{G}, v_\mathcal{G}), \mathcal{E}_\mathcal{G})$, where G is a multistage network and $v_\mathcal{G}$ is the set of stage (column) numbers. We refer to networks and graphs interchangeably in this paper and use the short form $X_{i:j}$ to represent $X_i X_{i-1} \cdots X_{j+1} X_j$.

Definition 1 (Unfolded HSN, UHSN(l, G)):
Let the nucleus graph be $G = ((\mathcal{V}_\mathcal{G}, v_\mathcal{G}), \mathcal{E}_\mathcal{G})$ with $v_\mathcal{G} \in [0, k]$. An l-level unfolded HSN based on the nucleus G is defined as the graph UHSN$(l, G) = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{V = (V_{l:1}, v) | V_i \in \mathcal{V}_\mathcal{G}, i = 1, ..., l, v \in [0, l(k+1) - 1]\}$ is the set of vertices, and $\mathcal{E} = \{(U, V) | U = (U_{l:1}, u), V = (V_{l:1}, v) \in \mathcal{V}$, satisfying $u = v + 1$ or $v = u + 1$ and either

a) $\max(u, v) \bmod (k+1) \neq 0$, and $1) U_{l:1} = V_{l:1}$, or $2) U_{l:2} = V_{l:2}$ and $(U_1, V_1) \in \mathcal{E}_\mathcal{G}$, or
b) $\max(u, v) = i(k+1), U_i = V_1, V_i = U_1$, and $U_j = V_j$ for $j \neq 1, i$, where $1 \leq i, j \leq l$.

$\}$ is the set of edges.

A UHSN(l, B_n) has 2^{ln} rows, $l(n+1)$ columns, and $l(n+1)2^{ln}$ nodes.

0-7803-3636-4/97 $10.00 © 1997 IEEE

Definition 2 (Unfolded RHSN):
An r-deep URHSN$(l_r, l_{r-1}, ..., l_1, G)$ is recursively defined as UHSN$(l_r,$ URHSN$(l_{r-1}, l_{r-2}, ..., l_1, G))$ for $r > 1$ and as UHSN(l_r, G) for $r = 1$.

We can generalize the construction of URHSN to obtain general USN, derived from "unfolding" general swapped networks [8]. The step sizes of USN can be as small as approximately doubling the number of nodes, as opposed to the factor $(1 + 1/l)2^n$ for a UHSN based on B_n. The definition of USNs is, in fact, a systematic method to synthesize large FFT computation networks with various smaller modules, which can be further generalized to any graph in addition to multistage networks. An example is shown in Fig. 2d. Note that the input and output nodes of the basic FFT modules may need to be permuted in order to perform FFT computation.

2.3. Recursive Construction of USNs

To better visualize the modularized construction of USNs, we present the construction of unfolded recursive swapped networks (URSNs) in this subsection. An r-deep URSN based on G is denoted as URSN(r, G) and is a URHSN$(\underbrace{2, 2, ..., 2}_{r}, G)$.

An r-deep butterfly-based URSN, denoted by URSN(r, B_n), begins with a nucleus B_n. To build a URSN$(1, B_n)$, we use 2^{n+1} identical copies of the nucleus B_n arranged in two stages, each containing 2^n copies. We give each nucleus a pair (Y_1', j_1) as its address, where Y_1' is an n-bit string, and $j_1 \in \{0, 1\}$ is the stage number. A node within the nucleus butterfly is addressed by the pair (Y_1'', y_1''), where Y_1'' is the n-bit row number, and $y_1'' \in [0, n]$ is the column number. Node (Y_1'', y_1'') of nucleus (Y_1', j_1) is given a pair $(Y_2'' = Y_1'Y_1'', y_2'' = j_1(n + 1) + y_1'')$ as its address within the URSN$(1, B_n)$. Node $(Y_1'Y_1'', n)$ has a link connecting it to node $(Y_1''Y_1', n + 1)$, which serves to connect stages 0 and 1. To build a URSN$(2, B_n)$, we use 2^{2n+1} identical copies of a URSN$(1, B_n)$. The copies of a URSN$(1, B_n)$ are arranged in two stages, each containing 2^{2n} copies. We give each copy a pair (Y_2', j_2) as its address, where Y_2' is a $2n$-bit string, and $j_2 \in \{0, 1\}$ is the stage number. Node (Y_2'', y_2'') of copy (Y_2', j_2) is given a pair $(Y_2'Y_2'', 2j_2(n + 1) + y_2'')$ as its address within the URSN$(2, B_n)$. Node $(Y_2'Y_2'', 2n+1)$ has a link connecting it to node $(Y_2''Y_2', 2n+2)$. URSNs with higher levels are constructed following the same rule. Structures of a URSN$(1, B_2)$ and a URSN$(2, B_1)$ are shown in Fig. 2. A URSN(r, B_n) has 2^{n2^r} rows and $(n + 1)2^r$ columns and uses $(n + 1)2^{n2^r+r}$ nodes of degree not exceeding 4.

3. FFT COMPUTATION

One of the reasons for our proposing USN is that FFT (and ascend/descend [2]) algorithm maps efficiently on this class of parallel architectures.

The *discrete Fourier transform* of an N-vector $\vec{x}$ is a linear transformation of $\vec{x}$ defined by $\vec{z} = F_N\vec{x}$, where the i, j entry of F_N is ω_N^{ij} for $0 \le i, j < N$ with $\omega_N = e^{-j2\pi/N}$ [2]. The key idea behind the FFT algorithm is to decompose the computation of the N-point DFT into successively smaller DFTs using the divide and conquer paradigm. In other words, we can reduce the problem of computing $\vec{z} = F_N\vec{x}$ to the problem of computing

$$\vec{u} = F_{N/2}\begin{pmatrix} x_0 \\ x_2 \\ x_4 \\ \cdot \\ \cdot \\ x_{N-2} \end{pmatrix} \text{ and } \vec{v} = F_{N/2}\begin{pmatrix} x_1 \\ x_3 \\ x_5 \\ \cdot \\ \cdot \\ x_{N-1} \end{pmatrix},$$

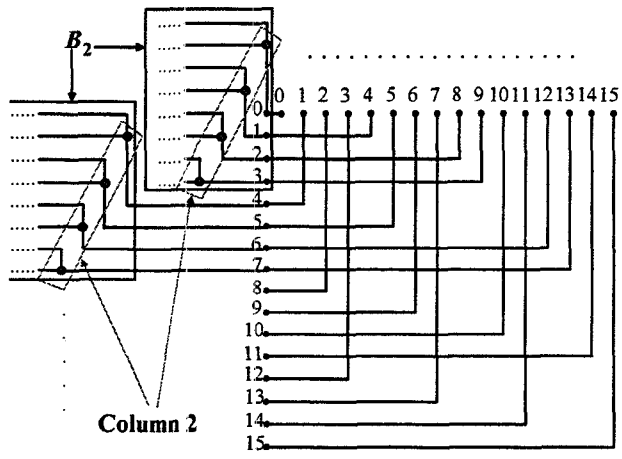

Figure 1. Partial layout of a URSN$(1, B_2)$ = UHSN$(2, B_2)$.

where $\omega_{N/2} = \omega_N^2$. We can then obtain $\vec{z}$ by computing

$$z_i = \begin{cases} u_i + \omega_N^i v_i & \text{for } 0 \le i < N/2, \\ u_{i-N/2} + \omega_N^i v_{i-N/2} & \text{for } N/2 \le i < N, \end{cases}$$

where z_i is the i^{th} element of $\vec{z}$. By unfolding the recursive algorithm, we can easily map the N-point FFT on a $\log_2 N$-dimensional butterfly network [2].

The FFT algorithm for USNs can be viewed as emulating the FFT computation for butterfly networks. An example of performing 8-point FFT on a UHSN$(3, B_1)$ is illustrated in Fig. 3. The initial distribution of data for FFT computation on a UHSN(l, B_n) is the same as that for an nl-dimensional butterfly network. Let $y \in [0, (n+1)l-1]$, $y_1 = \lfloor y/(n+1) \rfloor$, and $y_0 = y \bmod (n+1)$. That is, $y = (y_1, y_0)_{l,n+1}$. If $y_0 = 0$, no computation is required for node $Y = (Y_{l:1}, y)$ in the UHSN; otherwise, node Y in the UHSN emulates the computation required for a node in Column v of the butterfly, where $v = y - y_1$. If $y_1 = 0$, node Y emulates the node in Row $Y_{l:1}$; if $y_1 = 1$, node Y emulates the node in Row $Y_{l:3}Y_1Y_2$; if $y_1 > 1$, node Y emulates the node in the same row as the one in the emulated butterfly that node $(Y_{l:y_1+2}Y_1Y_{y_1:2}Y_{y_1+1}, y-n-1)$ emulates, and so on. The row numbers being emulated on the UHSN$(3, B_1)$ is given in the circles in Columns 2 and 4 of Fig. 3. Note that the output of the FFT computation is unordered.

The time required for these algorithms with problem size N on a URHSN based on B_n is $(1+1/n)\log_2 N - 1$, which is the number of columns minus 1, assuming unit time for addition, multiplication, and data exchange via a link. FFT algorithms on a URHSN can be applied on a USN with similar performance after minor modifications. Pipelining these algorithms on USNs is straightforward and similar to pipelining on butterfly networks. The algorithm can, in fact, be generalized to the class of ascend/descend algorithms, making USNs efficient architectures for many other problems, such as sorting and matrix multiplication.

4. HARDWARE COST AND DELAY

In this section, we present a simple VLSI layout and compute the associated area and delay bounds. We also compare the implementation of USNs with that of similar-sized butterfly networks under the consideration of pin limitation for large parallel architectures.

4.1. VLSI Layout and Area

In this subsection, we present a simple layout for a URSN and an upper bound on the area required under the grid model [6].

The area of the layout of a URSN$(1, B_n)$ is dominated by the links connecting the central columns n and $n+1$. We place the nodes in column n along a vertical line, and the nodes in column $n+1$ along a horizontal line as shown in Fig. 3. A link goes right from a node V in column n and then turns upward toward its neighbor in column $n+1$. Clearly, this part of layout requires area at most equal to $N(N+1)$, where $N = 2^{2n}$ is the number of rows.

It is well known that an n-dimensional butterfly network can be laid out within a $O(N)$ area [6]. However, we have to allocate two horizontal (or vertical) tracks to a node in column n (or $n+1$) for the construction of the nucleus butterfly to which it belongs. Since a nucleus B_n requires only small area $O(N)$, it is easy to build the remaining $\Theta(\sqrt{N})$ B_n's within $o(N^2)$ area. A naive method for constructing columns 0, 1, ..., n is as follows. We first add $\sqrt{N}$ horizontal tracks at the upper-left side of the previous layout. The top two new tracks are used to implement the links incident to node $(0, n)$ and are connected using a vertical wire as shown in Fig. 3. The next two new tracks are for node $(1, n)$ and are connected using a vertical wire located at the left-hand side of the previous vertical wire, and so on. We are then ready to use the construction given in [6] to lay out the nucleus B_n that includes rows 0,1,..., $\sqrt{N}-1$. Note that the positions of the nodes except for node $(0, n)$ are virtually shifted left and up and relocated at the intersection of its lower horizontal track and vertical wire.

To implement the i-th B_n, $i = 1, 2, ..., \sqrt[4]{N} - 1$, we extend the horizontal wires and shift all the nodes to the left of the previous B_n layout, and then construct it using the same rules. We can then repeat this procedure for the second group of $\sqrt[4]{N}$ B_n's by first adding tracks between the tracks originally for rows $\sqrt{N}-1$ and $\sqrt{N}$, and so forth.

Columns $n+1, n+2, ..., 2n+1$, can be built using similar strategies. Since the increased width and height are both $O(\sqrt[4]{N^3})$, the total area required for the URSN$(1, B_n)$ is no more than $N^2 + o(N^2)$. We can see that the layout area can be further reduced by placing columns n and $n+1$ in parallel since the number of parallel tracks can be considerably reduced. Although the nondominant terms of the area may not be negligible for small or moderate N, and the layout for URSNs can be further refined, the obtained upper bound on the area is asymptotically better than known results (under the grid model) for a butterfly network that has the same number of rows. This layout rule can be generalized to URHSNs without difficulty. It can be shown that a USN requires $O(N^2)$ area and $O(\log N)$ time per cycle when pipelined for each column, the required AT^2 is $\Theta(N^2 \log^2 N)$ and is thus optimal for N-point FFT computation [6, 7],

4.2. Pin Limitations

When interconnection networks grow very large, pin limitation becomes a major constraint in their implementations [1]. In this subsection, we give an example to illustrate the advantage of USNs under this consideration.

Assume that N is very large and that no more than $O(\sqrt{N}\log N)$ nodes can be put onto a single chip. In order to implement an N-row URSN(r, B_n), we can partition the network into $2\sqrt{N}$ subnetworks, each of which is a URSN$(r-1, B_n)$. By the definition of URHSN and the construction presented in Subsection 2.3., it is clear that only $\sqrt{N}$ inter-chip links are required to connect the network, and usually $O(\sqrt{N})$ additional inter-chip links are required for I/O. On the other hand, $O(\sqrt{N}\log N)$ off-chip connecting links are needed when building a butterfly network under this assumption. This brief comparison implies that a considerably larger number of chips (or boards) will be required when the number of pins per chip (or connectors per board) is the limiting factor for implementation.

4.3. Computation Delay

Another advantage of USNs is that their communication patterns tend to be localized. For example, only one step during FFT computation requires routing data outside a chip (that is, the central step) in the previous URSN implemented using $O(\sqrt{N})$ chips. This feature leads to better performance than hypercube and butterfly networks when link delay increases with length, and inter-chip communications are more expensive than intra-chip ones. For example, if the delay for an inter-chip transmission is larger by a factor of d than an intra-chip transmission, the FFT computation on a URSN will be faster by a factor of $\Theta(d \log N/(d+\log N))$ than on butterfly networks. As another example, if the delay of a long link is proportional to the logarithm of its length, and the delay of a short link is constant, the time required for FFT computation (which originally ran in $O(\log N)$ time with constant delay links) becomes $\Theta(\log^2 N)$ on hypercube and butterfly networks. In contrast, the delay of FFT computation on USN remains $\Theta(\log N)$, with the hidden constant increased only slightly. If we assume logarithmic delay for all links, the time required for the FFT computation on a URSN(r, B_n) with $r = \log_2 \log_2 N$, is $O(\log N \log\log N)$.

When the delay of a link grows faster than the logarithm of its length, the advantage of FFT computation on a USN over hypercube and butterfly networks will become even more pronounced.

5. CONCLUSION

We have proposed a new class of parallel architectures, unfolded swapped networks (USN), for FFT computation and related problems. We showed that FFT can be executed on a USN at a speed comparable to, or faster than, a butterfly network. USNs have highly modularized constructions, thus facilitating implementation concerns such as routing and layout, and faring quite well on scalability. They appear to be attractive alternatives to butterfly networks, especially when the network size is large.

REFERENCES

[1] Franklin, M.A., D.F. Wann, and W.J. Thomas, "Pin limitations and partitioning of VLSI interconnection networks," *IEEE Trans. Comput.*, Nov. 1982, vol. C-31, pp. 1109-1116.

[2] Leighton, F.T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes,* Morgan-Kaufman, San Mateo, CA, 1992.

[3] Loan, C.V., *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, PA, 1992.

[4] Swartzlander, E.E. Jr., V.K. Jain, and H. Hikawa, "A radix-8 wafer scale FFT processor," *J. VLSI Signal Processing,* May 1992, vol. 4, no. 2-3, pp. 165-176.

[5] Tanno, K., T. Taketa, and S. Horiguchi, "Parallel FFT algorithms using radix 4 butterfly computation on an eight-neighbor processor array," *Parallel Computing,* Jan. 1995, vol. 21, no. 1, pp. 121-136.

[6] Thompson, C.D., "Fourier transforms in VLSI," *IEEE Trans. Comput.*, Nov. 1983, vol. C-32, pp. 1047-1057.

[7] Vuillemin, J., "A combinatorial limit to the computing power of VLSI circuits," *Proc. Symp. Foundations of Computer Science*, 1980, pp. 294-300.

[8] Yeh, C.-H. and B. Parhami, "Swapped networks: unifying the architectures and algorithms of a wide class of hierarchical parallel processors," *Proc. Int'l Conf. Parallel and Distributed Systems,* 1996, pp. 230-237.

[9] Yeh, C.-H. and B. Parhami, "Recursive hierarchical swapped networks: versatile interconnection architectures for highly parallel systems," *Proc. IEEE Symp. Parallel and Distributed Processing*, 1996, to appear.

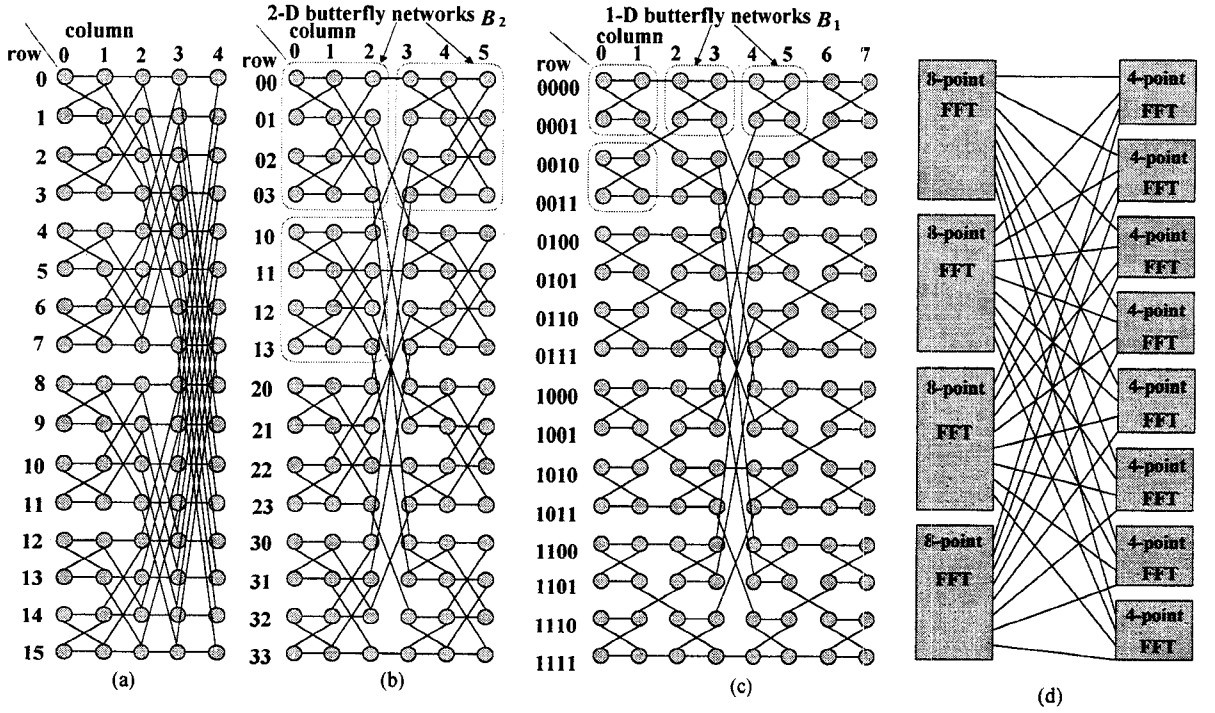

Figure 2. Structures of a butterfly and USNs. (a)B_4.(b) UHSN$(2, B_1)$. Row numbers are represented in radix 4. (c) URSN$(2, B_2)$. (d) Synthesis of a 32-point FFT computation network using 8-input and 4-input modules.

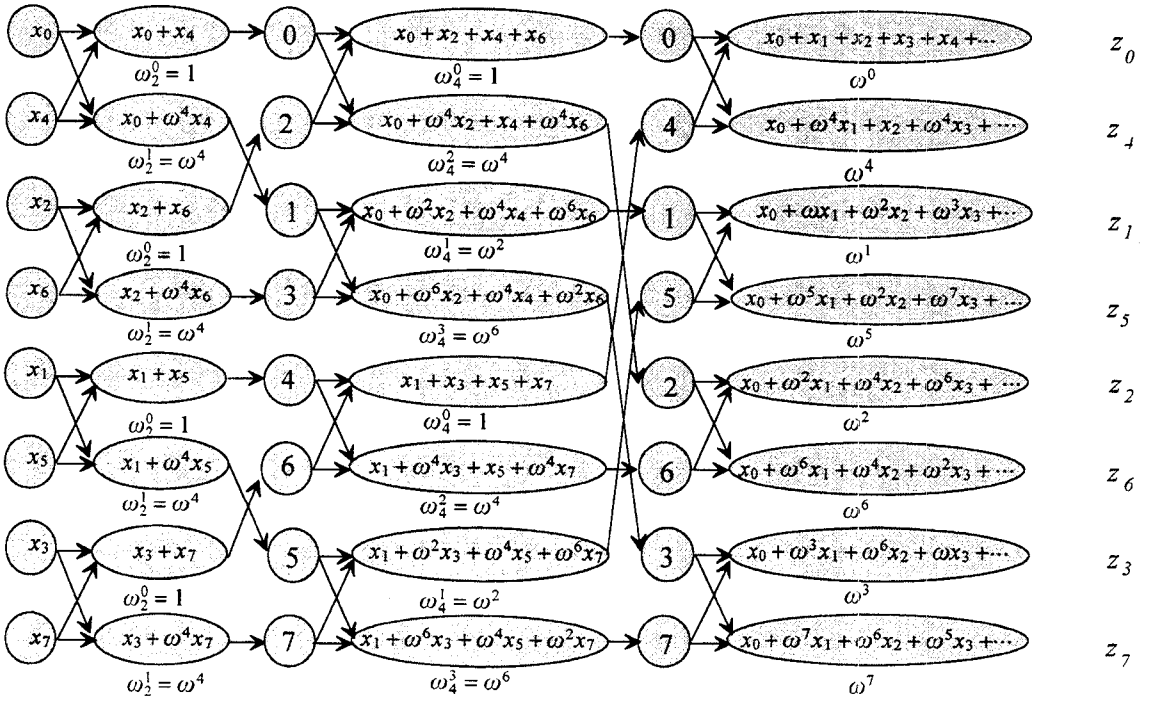

Figure 3. Entire calculation of an 8-point FFT on a UHSN$(3, B_1)$. The row numbers of the B_3 being emulated is given in the circles in Columns 2 and 4.