Implementation of the fast-Fourier-transform algorithm on a parallel processor

M. S. Ganagi and K. Neelakantan,

ANURAG Advanced Numerical Research and Analysis Group, P. O. Kanchanbagh, Hyderabad 500 258, India

We describe an algorithm to implement the fast-Fourier-transform (FFT) algorithm on a parallel computer. This algorithm provides a balanced computation load to all the processors in the system and involves fewer communications compared to implementations described earlier in the literature. We also present results for FFT computations on data of various sizes and on different numbers of processors.

FOURIER transforms are widely used in scientific calculations. When dealing with discrete time signals, one uses the discrete Fourier transform[1] (DFT), defined as

$$X(k) = \sum_{i=0}^{N-1} x_i \exp(-\mathrm{j}\,2\pi ik/N),\ 0 \le k \le N-1, \quad (1)$$

where $\mathrm{j} = \sqrt{-1}$. This is often expressed as

$$X(k) = \sum_{i=0}^{N-1} x_i w^{ik},\ 0 \le k \le N-1. \quad (2)$$

One can see from eq. (1) that the computation of a DFT over N samples in a naive fashion involves $O(N^2)$ complex computations. However, the fast-Fourier-transform[1,2] (FFT) algorithm reduces this to $O(N \log_2 N)$ complex computations.

Since the FFT algorithm is widely used in scientific computations, there has been a lot of interest in adapting this for vector and parallel computers. The FFT does not lend itself to efficient implementations on vector machines[3]. However, the FFT algorithm can be parallelized quite easily. Our purpose, in this paper, is to describe a parallel FFT algorithm for hypercube architectures and its implementation on PACE (ref. 4), the parallel processing system developed at ANURAG.

FFT algorithms are implemented for a sequence of 2^L ($=N$) points. We have parallelized the in-place, radix 2, decimation-in-time algorithm[1]. A 2^L-point FFT algorithm involves L stages of computations as shown in Figure 1 (which shows a 16-point FFT with four stages). In the first stage of the computations, $N/2$ two-point DFTs are computed (the traditional FFT 'butterfly' operations). At the next stage, these two-point DFTs are combined to form $N/4$ four-point DFTs. Next, the four-point DFTs are combined to form eight-point DFTs, and so on. The input sequence is taken in the 'bit-reversed' order as required for a decimation-in-time algorithm.

To parallelize the FFT algorithm to run on P processors ($P = 2^D$, $L > D$) arranged on a D-dimensional hypercube, the data are equally divided into N/P segments. For each segment, the first, $L-D$ stages of the FFT are run independently on the P processors. These $L-D$ stages do not require any communication between the processors. This results in N/P-point DFTs in each of the P processors. Next, the P segments (N/P-point DFTs) are to be combined. This requires D stages of computations. These D stages require communication between the processors.

The straightforward implementation of the last, D stages of the FFT on a parallel processor would involve pairs of processors exchanging their entire data after one member of each pair multiplies its data by w^i. After exchanging the data one processor calculates the sum

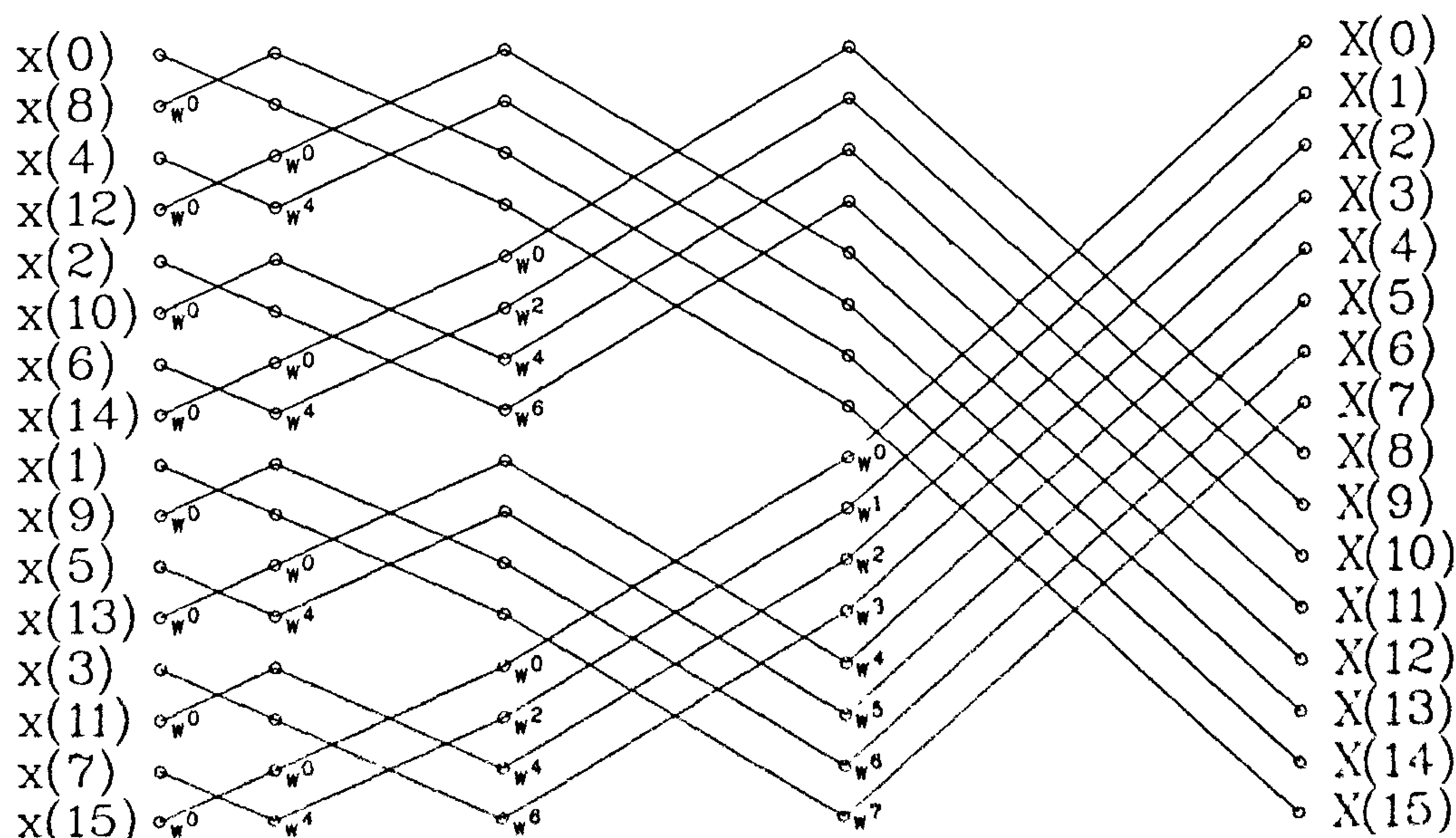

Figure 1. Data-flow diagram of 16-point, radix 2, in-place, decimation-in-time FFT algorithm

while the other calculates the difference. Thus half the processors have to do more computations than the other half and each processor has to communicate its entire data to its partner at each stage[5].

Our implementation also involves pairs of processors exchanging data, but, in each pair, one processor sends the even-numbered data to its partner and receives the odd-numbered data from its partner. Next, each processor of a pair computes the FFT butterfly, i.e. multiplies the appropriate data points by w^i, determines the sums and differences, and stores them in consecutive locations. The data-flow diagrams for the above scheme for a 16-point FFT on two and four processors are shown in Figures 2 and 3 respectively; the dashed lines separate the data assigned to various processors. A communication is required every time the data-flow path (shown as solid lines) crosses the dashed lines.

The processors on a D-dimensional hypercube can be numbered uniquely by D binary bits. The numbering usually follows the Gray-code sequence so that two processors are connected if their binary representations differ in only one bit. For the last, D stages of the parallel FFT algorithm, the processors whose Gray-code representations differ in the ith bit from the least significant bit form pairs in the ith stage. The input to the parallel algorithm is in the bit-reversed order but output comes in an order where even-numbered data are perfectly shuffled and distributed among the even-numbered processors and odd-numbered data are shuffled and distributed among the odd-numbered processors.

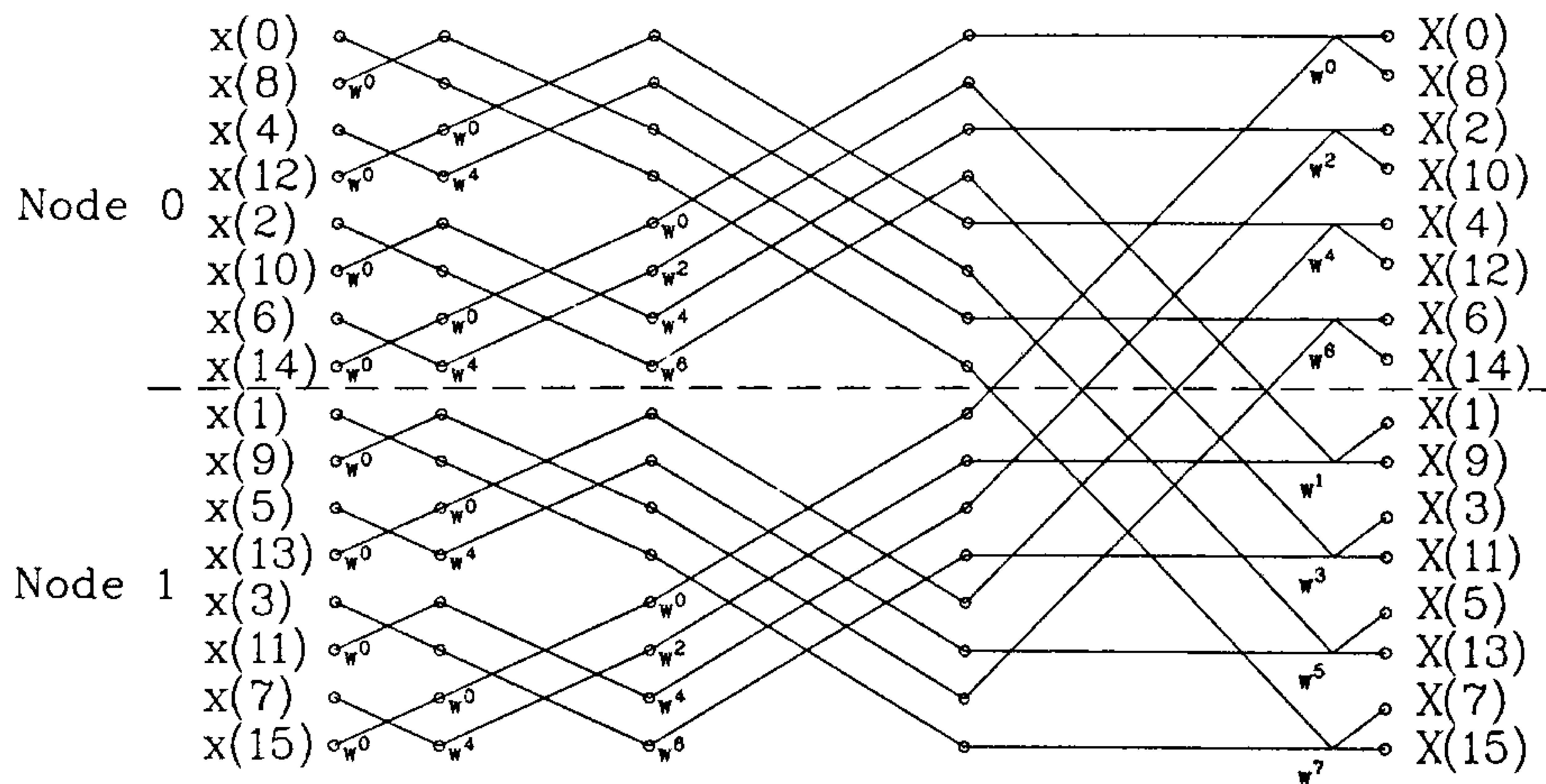

Figure 2. Data-flow diagram of 16-point, radix 2, in-place, decimation-in-time FFT algorithm on one-dimensional hypercube

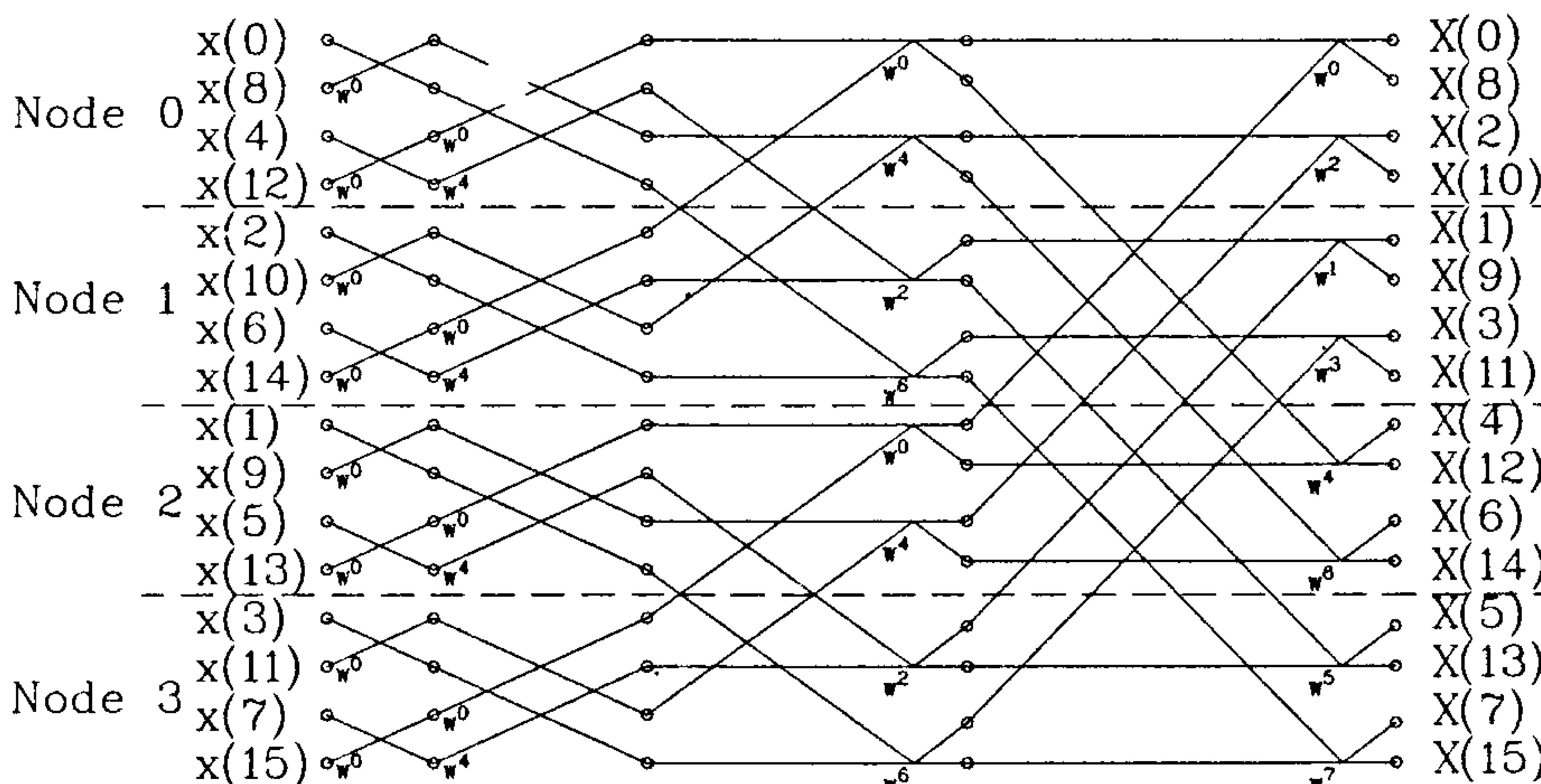

Figure 3. Data-flow diagram of 16-point, radix 2, in-place, decimation-in-time FFT algorithm on two-dimensional hypercube

For the last D stages, the multipliers w^i become functions of the processor number and the stage number. Since $w^p * w^q = w^{p+q}$, one can easily find the starting w^r and common factor w^s such that successive multipliers would be w^r, $w^r * w^s$, $w^r * w^s * w^s$, and so on.

One point to be noted is that the usual algorithms result in computation of the DFT in bit-reversed or normal order depending on the sequence chosen for the input data. In our case, however, the data come out in a shuffled order. But this does not create any problem because one can reorder the data while transmitting them back to the host processor. However, one usually calculates the DFT to perform some operations on the transformed data (as in the computation of convolutions etc.). Later, one needs to compute the inverse transform.

The inverse transform generally follows the same scheme as that used for the direct transform. In the case of the inverse transform, one can start with the data in the shuffled order, such as that obtained from the direct transform. The first, D stages of the computations involve using pairs of processors and exchanging data between them. The remaining, $L-D$ stages are done independently in each processor. The algorithm is shown in Figures 4 and 5 for 16-point inverse DFTs on two and four processors respectively. For the first, D stages, each processor computes the butterfly operations on consecutive data. One processor of each pair sends the difference to its partner while the other processor sends the sum, as shown in Figures 4 and 5.

We have implemented the above algorithm on PACE-8, which has eight processors connected to a host processor. The FFT computations were done for various values of N on one, two, four and eight

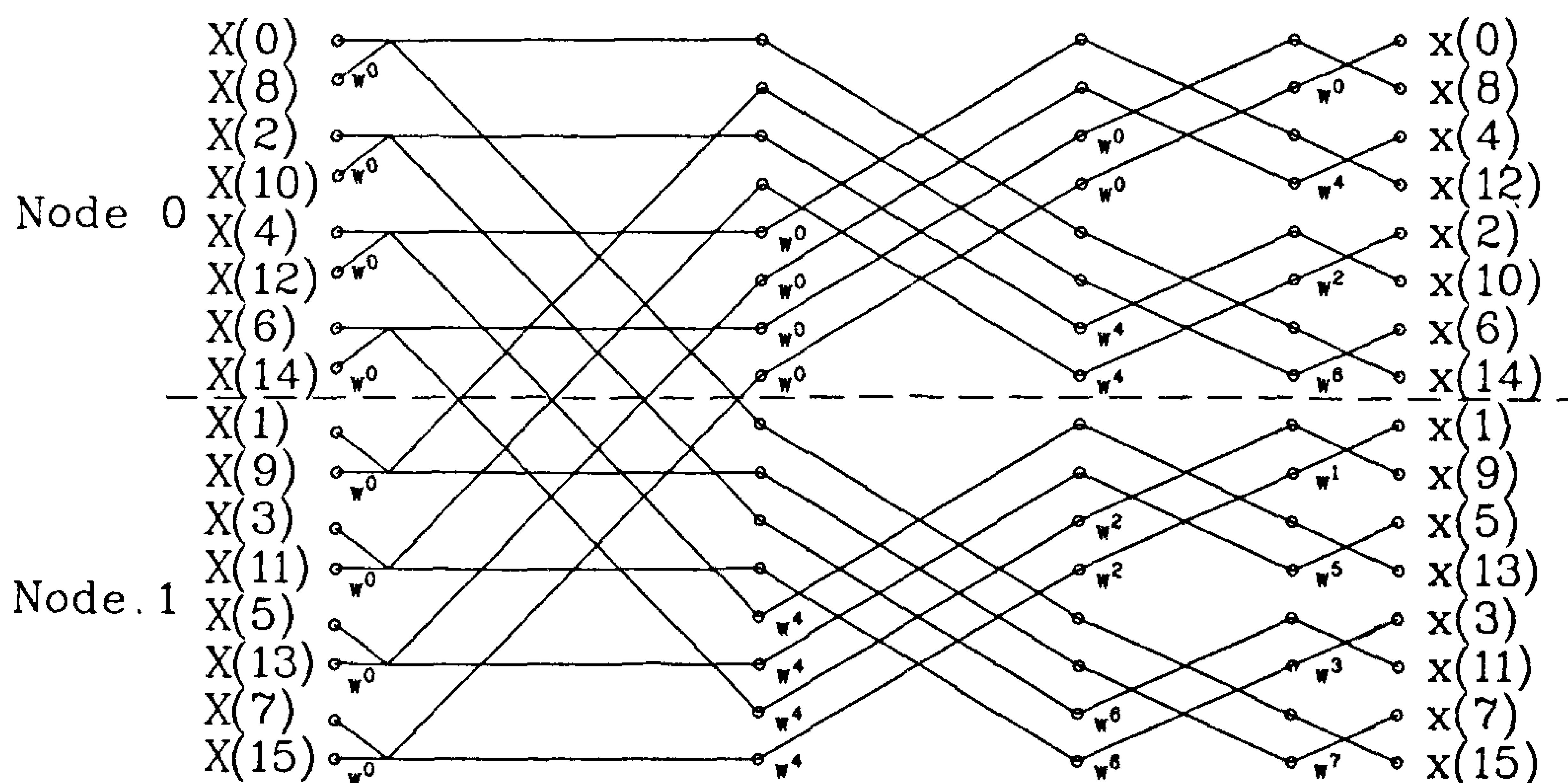

Figure 4. Data-flow diagram of 16-point, radix 2, in-place, decimation-in-time inverse FFT algorithm on one-dimensional hypercube.

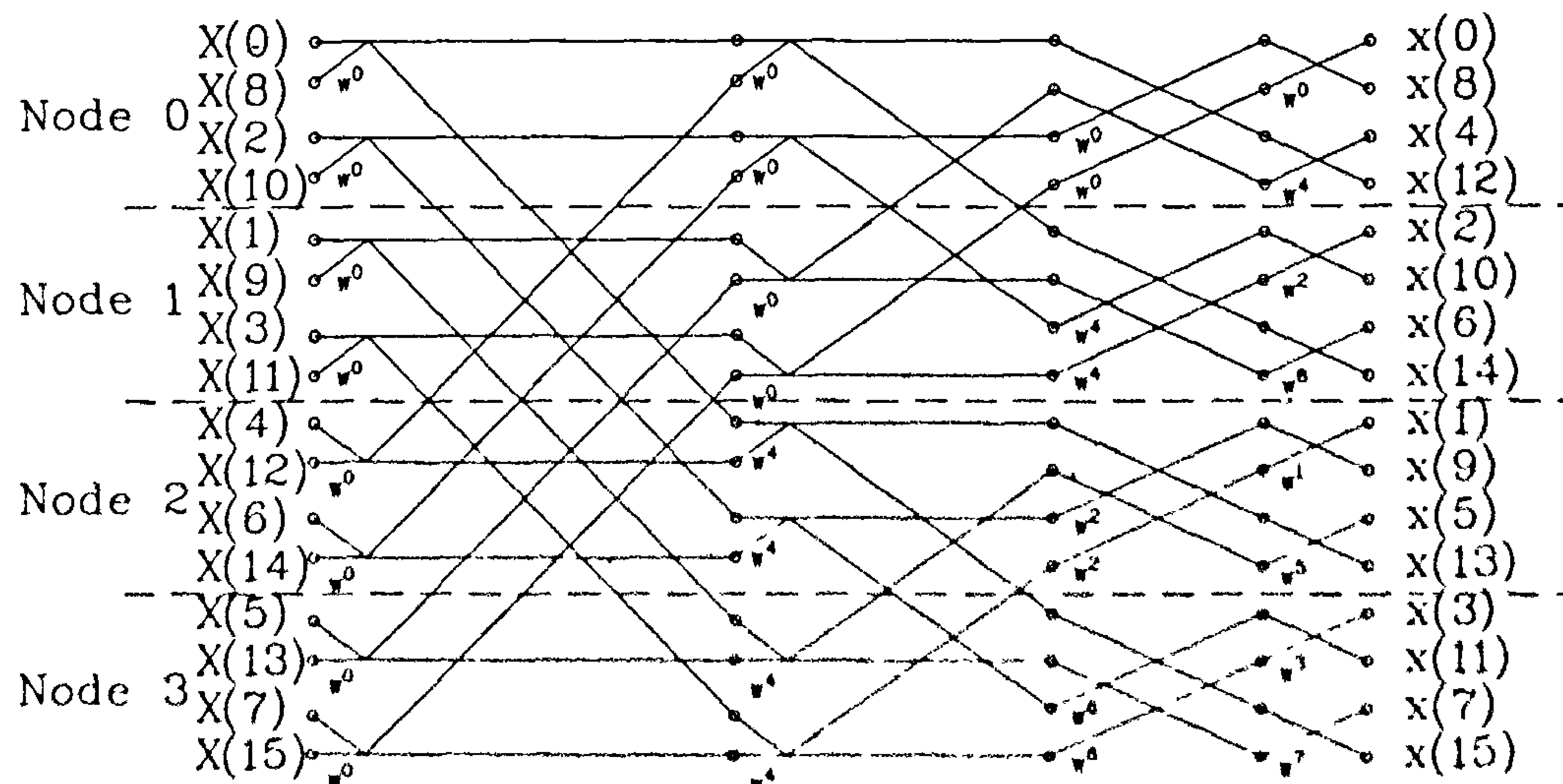

Figure 5. Data-flow diagram of 16-point, radix 2, in-place, decimation-in-time inverse FFT algorithm on two-dimensional hypercube.

Table 1. Performance of the FFT algorithm on sequential and parallel machines.

N	Sequential time	Two processors		Four processors		Eight processors	
		Time	Speedup	Time	Speedup	Time	Speedup
1024	0.251	0.137	1.836	0.075	3.352	0.042	5.954
2048	0.545	0.294	1.851	0.159	3.430	0.087	6.235
4096	1.175	0.631	1.863	0.339	3.471	0.184	6.373
8192	2.523	1.347	1.873	0.719	3.507	0.389	6.485
16,384	5.391	2.867	1.881	1.524	3.539	0.819	6.577
32,768	11.475	6.079	1.888	3.217	3.567	1.725	6.652
65,536	24.332	12.808	1.900	6.772	3.593	3.625	6.713
131,072	51.398	27.016	1.902	14.224	3.613	7.591	6.771

Table 2. Comparison of predicted and observed speedups.

N	Two processors		Four processors		Eight processors	
	Predicted	Experimental	Predicted	Experimental	Predicted	Experimental
1024	1.815	1.836	3.345	3.352	6.056	5.954
2048	1.832	1.851	3.375	3.430	6.239	6.235
4096	1.846	1.863	3.424	3.471	6.378	6.373
8192	1.857	1.873	3.465	3.507	6.489	6.485
16,384	1.867	1.881	3.499	3.539	6.583	6.577
32,768	1.875	1.888	3.529	3.567	6.663	6.652
65,536	1.882	1.900	3.555	3.593	6.736	6.713
131,072	1.889	1.902	3.579	3.613	6.799	6.771

processors and the results are given in Table 1. The times reported in Table 1 are for the actual FFT computation only and do not include the time taken for sending the data to the nodes and for retrieving the results. The speedup for two nodes varies from 1.83 to 1.9 (depending on N), while for 8 nodes the spread is a little more and ranges from 5.95 to 6.77. The lower efficiency on eight nodes for smaller values of N is expected and has been observed for other applications as well[4].

The time for initially sending the data from the host to the nodes was approximately 12.8 μs per point (depending on N). The effective speedup (including the time for distributing the data and collecting the results from the nodes) was 5.03 for a 131,072-point FFT on 8 nodes.

It may be noted that the efficiency of parallelization of the FFT algorithm has been reported[6] to be around 75%. The efficiencies observed by us range from 91.8% to 95% in the case of two processors and from 74% to 84.5% in the case of eight processors. The improved efficiencies are due to the fact that, in our implementation, the communications have been reduced by a factor of 2 and all the processors are equally loaded.

The speedup S of the parallel FFT algorithm for N data points and on P number of processors is given by

$$S = \frac{P}{1 + \frac{2D(t_{start} + t_{send} N/2P)}{5 t_{comp} N L/P}}, \quad (3)$$

where t_{start} is the startup time for a node-to-node communication, t_{send} the time to communicate a real number between the nodes, and t_{comp} the time taken for floating-point computation.

In Table 2, the speedup predicted by eq. (3) and the experimentally observed values are compared for different datasizes and for different numbers of processors. The theoretical speedup is computed by using $t_{start} = 180\ \mu$sec, $t_{send} = 6.4\ \mu$sec, and $t_{comp} = 4\ \mu$sec. It may be seen from Table 2 that the results are in good agreement with the predictions.

1. Rabiner, L. R. and Gold, B., *Theory and Application of Digital Signal Processing*, Prentice Hall of India, New Delhi, 1978.
2. Oppenheim, A. V. and Schafer, R. W., *Digital Signal Processing*, Prentice Hall, New Jersey, 1975.
3. Ashworth, M. and Lyne, G., *Parallel Computing*, 1988, **6**, 217.
4. Neelakantan, K., Ghosh, P. P., Ganagi, M. S., Athithan, G., Atre, M. V. and Venkataraman, G., *Curr. Sci.*, 1990, **59**, 982.
5. Chamberlain, R. M., *Parallel Computing*, 1988, **6**, 225.
6. Walton, S. R.; in *Hypercube Multiprocessors 1987* (ed. Heath, M. J.), SIAM, Phildelphia, 1987, p 530.

Received 30 May 1991; accepted 30 May 1991