A Performance Study of Parallel FFT in Clos and Mesh Networks

Rajkumar Kettimuthu[1] and Sankara Muthukrishnan[2]
[1]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA*
[2]*IBM Corporation, Bangalore, Karnataka 560017, India*
kettimut@mcs.anl.gov, sankara_m@in.ibm.com

Abstract

Though it is known that clos interconnection networks have many advantages over the other interconnection networks, it would be interesting to see the performance benefits that clos networks can bring to the real world applications. In this paper, we compare the performance of Fast Fourier Transforms (FFT) in clos networks with that of in mesh networks, a popular interconnection topology. We use both two- dimensional (2-d) and three-dimensional (3-d) formulations of FFT for our studies. We show that the performance of FFT in clos networks is significantly better and we further show that 3-d formulation outperforms the 2-d formulation.

1. Introduction

A clos network [1] is a rearrangeable, which means that it can route any permutation without blocking. Although the property of being rearrangeable is rarely exploited directly in clusters, a rearrangeable network necessarily exhibits full (maximal) bisection, a property that is crucial to any network that claims to be scalable. A message-passing network's minimum bisection, measured in links, is defined mathematically as the minimum number of links crossing any cut that bisects the hosts (half of the hosts on one side of the cut, the other half on the other side). The minimum bisection provides a metric for the minimum traffic-handling capacity of a network irrespective of the communication patterns among the hosts. The upper bound on the minimum bisection of a network of *N* hosts is *N/2* links because there will be bisecting cuts possible across half of the host links irrespective of the internal topology of the network.

The clos topology provides multiple routes between hosts, and all shortest routes are deadlock-free. This allows the traffic to be dispersed in a way that statistically avoids "hot spots," high utilization of specific network links that can result from single-route mappings of the communication patterns of application programs to the topology. This same dispersive routing technique can provide fault tolerance on a much smaller time scale than by periodic mapping, and can exploit multiple ports on host interfaces.

Though theoretically clos networks are supposed to give better performance for the applications, it is important to perform experimental studies using real world applications. We use FFT [2-11], an application that is both computation and communication intensive, to show the benefits that clos networks can provide when compared to the mesh networks [12]. Discrete Fourier Transform (DFT) [13-15] plays an important role in many scientific and technical applications including wave analysis, solutions to partial differential equations, digital signal processing and image filtering. The DFT is a linear transformation that maps *n* regularly sampled points from a cycle of a periodic signal onto equal number of points representing the frequency spectrum of the signal. FFT is an algorithm to compute the DFT of an n-point series. We use a 64-processor and a 128-processor mesh and clos topology systems for our experiments. The paper is organized as follows. Section 2 describes the topology information. The algorithm used for performing parallel FFT is explained in Section 3. Section 4 provides the experimental results and we conclude in Section 5.

2. Topology Information

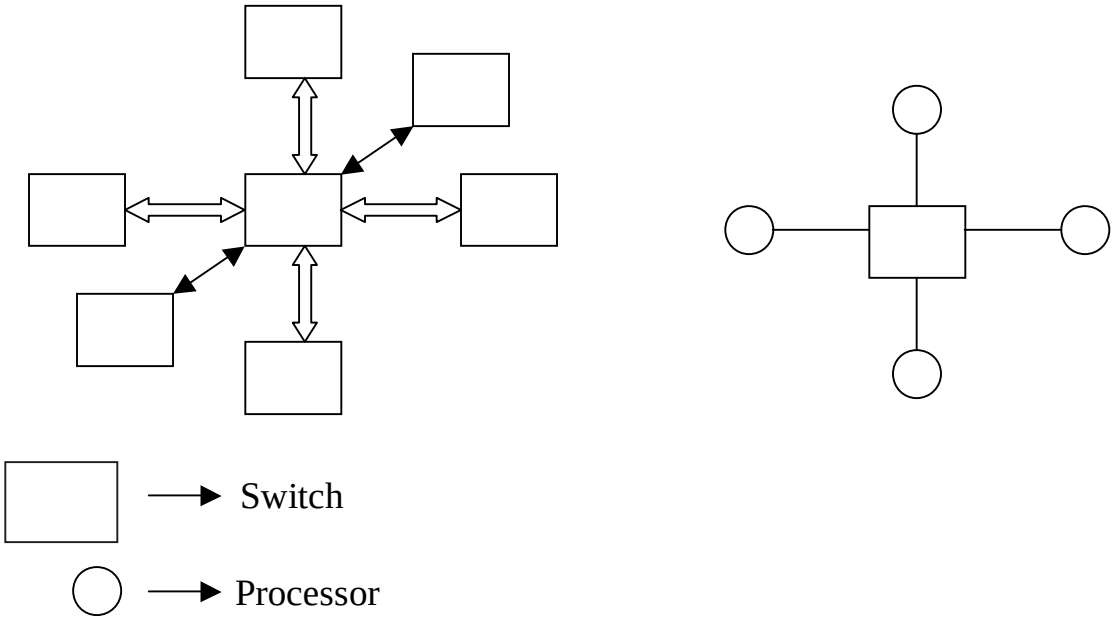
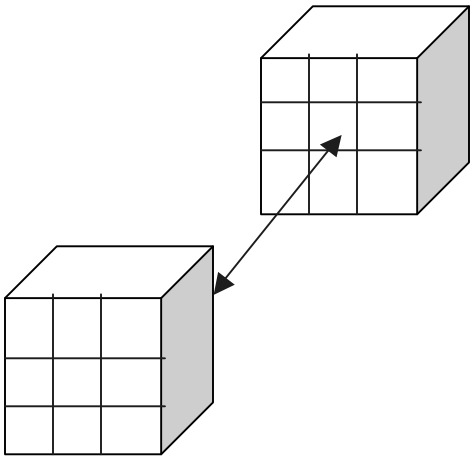


Figure 1. Mesh Topology

For a 64-processor mesh network, 16 switches each containing 16 ports were used. Out of the 16 ports, four ports were used to connect the processors and others were used to connect with the other switches. Similarly for the 128-processor system, 32 switches were used.

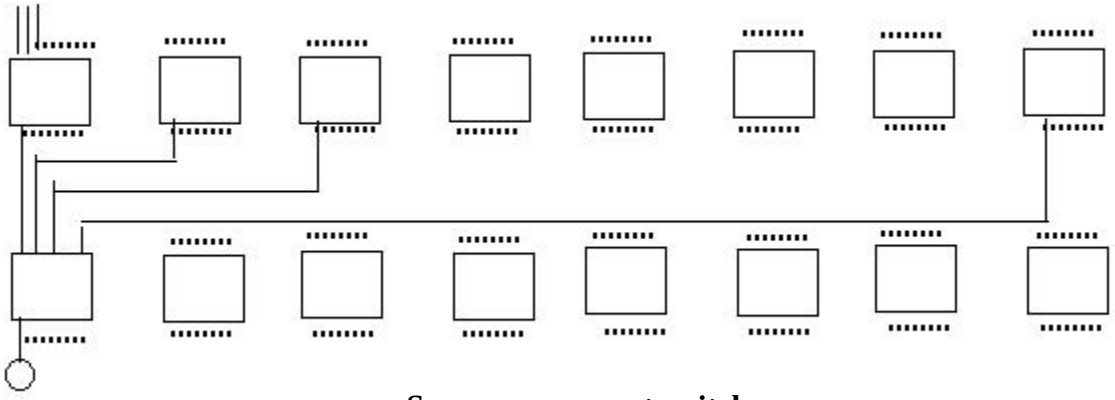


Each level has 64 processors and multiple levels can be added

Figure 2. 3-d Mesh Topology

For a 64-processor clos network, 16 switches - each containing 16 ports - were used (8 switches in the first level and 8 switches in the second level). In the first level, 8 ports of each of

the switches were used to connect to the processors and the other 8 were used to connect to the switches in the upper level. In the second level, 8 ports were to connect to the switches in the lower level and the other 8 were unused. For a 128-processor system, 24 switches were used (16 in the lower level and 8 in the upper level). All the 16 ports in the upper level switches were used here to connect to the switches in the lower level.



Squares represent switches
Circles represent processors

Figure 3. CLOS Topology

3. Algorithm

There has been a great interest in implementing FFT on parallel computers [16-23]. Commonly used parallel formulations of the FFT algorithm are binary-exchange algorithm [16, 18, 19, 21-27] and transpose algorithm [17, 23, 25, 26]. We use the transpose algorithm for our studies. We compute the performance of FFT using both the 2-d and 3-d transpose algorithms. In the two-dimensional transpose algorithm, the input of size D elements is arranged in a $\sqrt{D} \times \sqrt{D}$ 2-d array that is stripe-partitioned among P processors. These processors, although physically connected in a clos or mesh network, can be regarded as being arranged in a logical one-dimensional linear array. We call this as one-dimensional (1-d) layout or two-dimensional FFT.

Thus, in a 1-d layout (2-d FFT), $\sqrt{D} \times \sqrt{D}$ input elements are mapped to the P processors logically arranged as linear array. Each processor will have D/P elements. Each of them will compute the FFT for the (column) elements present in it.

Thus, the computation time = $(\sqrt{D}/P) * S * \sqrt{D} * \log\sqrt{D}$, where S is a constant.

The elements are transposed after this computation and for transposing the matrix, the number of elements that needs to be passed on to the other processors is D / (P * P). After getting the elements from other processors, each processor computes the FFT again

The communication cost = $(P-1) [t_s + t_w * D / (P * P)]$

As an extension of the above mentioned 1-d layout scheme, in a 2-d layout (3-d FFT), the $D^{1/3} \times D^{1/3} \times D^{1/3}$ input elements are mapped to P processors logically arranged as $\sqrt{P} \times \sqrt{P}$ grid. Each processor has $(D^{1/3}/\sqrt{P}) * (D^{1/3}/\sqrt{P})$ columns each of size D1/3. The total computation involves three computation phases and two communication phases. In the first, third and fifth phases, each processor does the computation of the elements present in it. In the second phase, the elements are transposed along **xz** plane and the fourth phase involves transposition along **yz** plane.

The computation cost is $(D^{2/3}/P) * S * D^{1/3} * \log D^{1/3}$

The communication cost is $(\sqrt{P} - 1) [t_s + t_w * D / (P*\sqrt{P})]$

Since this algorithm requires transposing the matrix, in the 2-d FFT, every processor needs to communicate with every other processor and in the 3-d FFT, every processor needs to communicate with every other processor in its column in the first phase of communication and every other processor in its row in the second phase of communication. With the clos networks, the number of hops to reach a node in any of the other switches is only 2 hops for the 64 and 128-processor systems where there are only two levels. In order to optimize the performance, the input elements are distributed across different processors in such a way that the communication for one of the phases need not go to a different switch.

4. Experimental Results

We performed all our experiments using a locally developed discrete event simulator. The time taken for various problem sizes and system sizes were measured for both the 1-d and 2-d layouts. Each FFT point was considered as a complex number. Figure 4 shows the time taken for the 2-d FFT in the 64-processor clos and mesh networks for the problem sizes varying from 64^2 elements to 4096^2 elements.

As can be observed from Figure 4, there is no appreciable difference between mesh and clos for small problem sizes. When the problem size is small, the amount of communication between the processors is less and thus the contention is less in the mesh network. But as the problem size increases, the difference is more pronounced and clos outperforms mesh. For the larger problems, in the mesh networks, the size and the number of messages that need to be communicated among the various processors increase and thus there are a lot of contentions. But, in the clos networks, the contention is less as there are more number of links, and thus it performs much better than the mesh. The percentage improvement is around (40-50%) for large problem sizes.

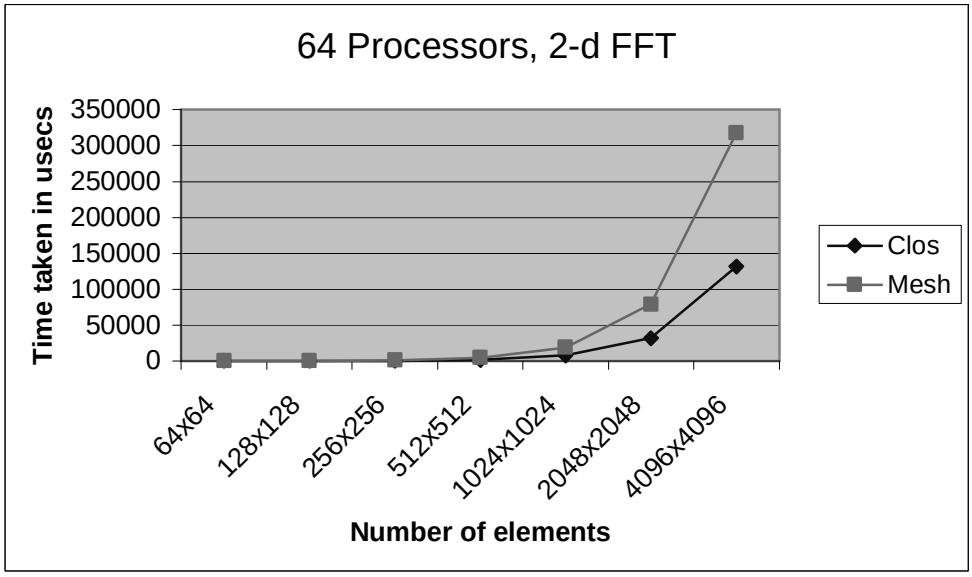


Figure 4. Comparison of 64-processor Clos and Mesh networks for 2-d FFT

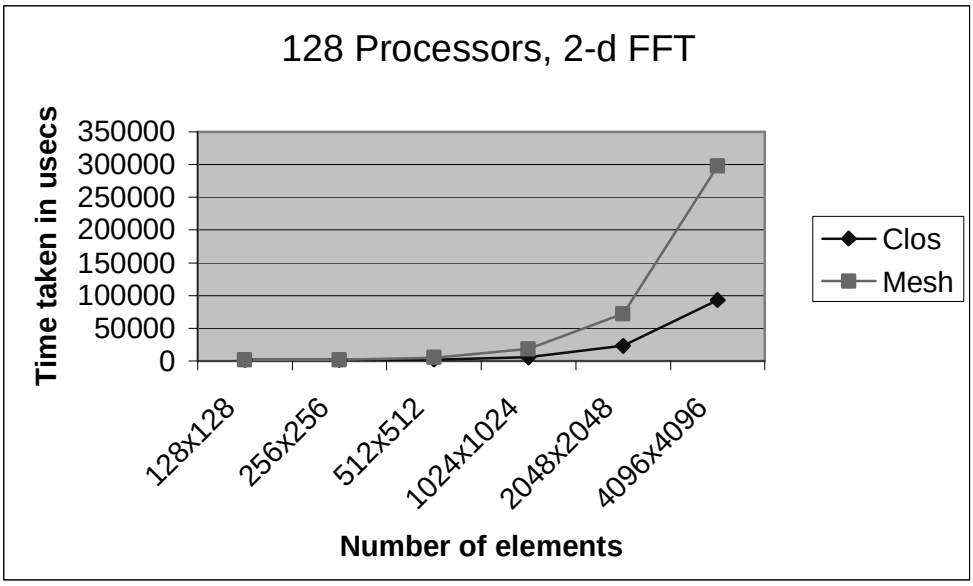


Figure 5. Comparison of 128-processor Clos and Mesh networks for 2-d FFT

Figure 5 shows the time taken for the 2-d FFT in the 128-processor clos and mesh networks for the problem sizes varying from 128^2 elements to 4096^2 elements.

For the 128-processor system, the trends are similar to that of the 64-processor system. We observe that as the number of processors increases from 64 to 128, the total time taken for the small problem sizes increases whereas the total computation time for the large problem sizes decreases.

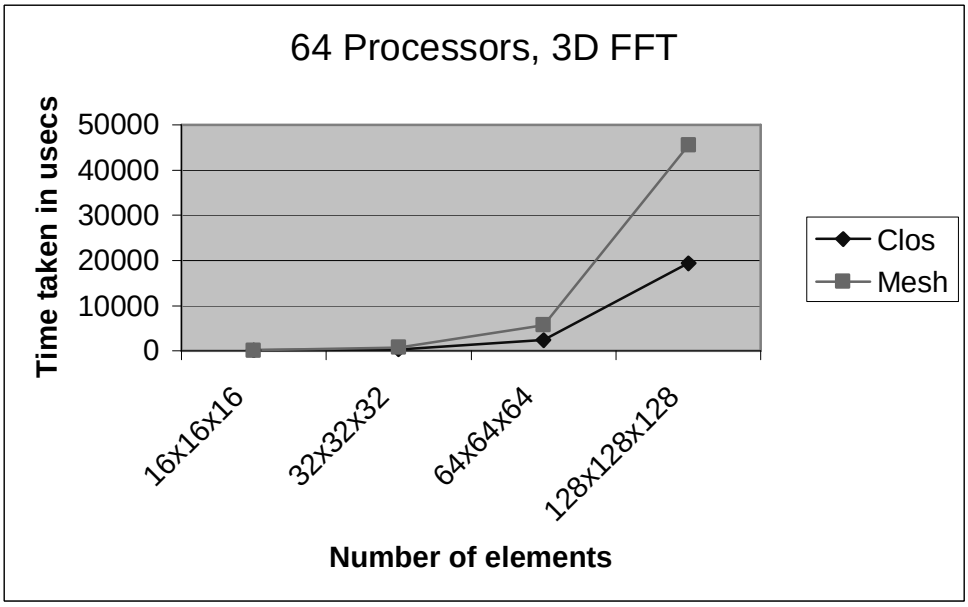


Figure 6. Comparison of 64-processor Clos and Mesh networks for 3-d FFT

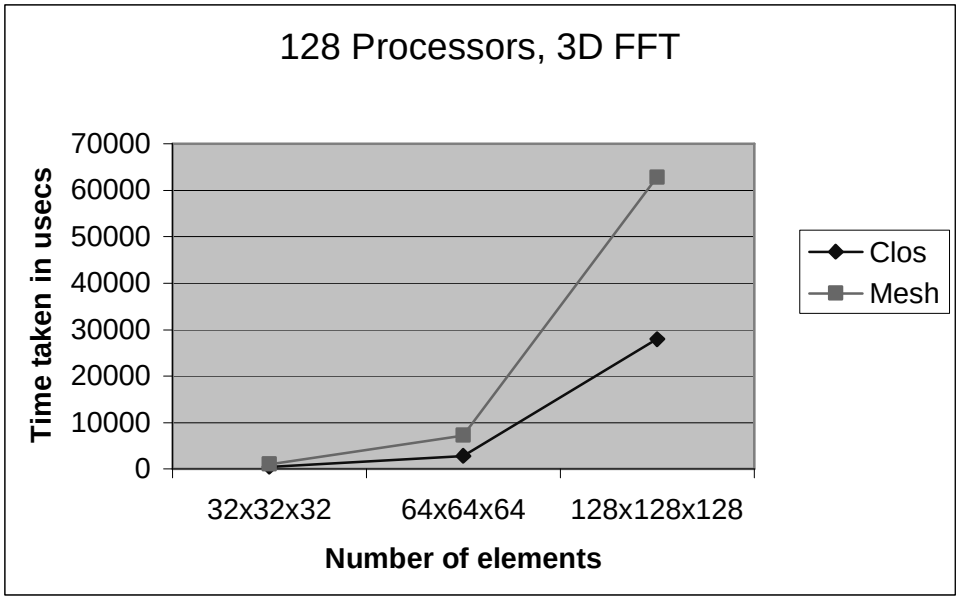


Figure 7. Comparison of 128-processor Clos and Mesh networks for 3-d FFT

When the problem size is small, the communication time is predominant over the computation time. If we increase the number of processors, the number of communications increases. Even though the computation time reduces and the message size reduces, the startup time incurred for each of the communication attributes a significant overhead to the total computation time.

Figure 6 shows the time taken for the 3-d FFT in the 64-processor clos and mesh networks for the problem sizes varying from 16^3 elements to 128^3 elements. Figure 7 shows the time taken for the 3-d FFT in the 128-processor clos and mesh networks for the problem sizes varying from 32^3 elements to 128^3 elements.

The results for 3-d FFT are similar to that of 2-d FFT. It can be observed that the time taken for the total computation in 3-d FFT is less than that of in 2-d FFT for the same problem size. Input size of 64x64x64 in 3-d FFT corresponds to an input size of 512x512. Similarly the problem size of 128x128x128 corresponds to the size between 1024x1024 and 2048x2048.

5. Conclusion

We evaluated the performance of parallel FFT in clos networks and mesh networks and observed that clos networks outperformed mesh networks. An improvement of up to 50% was obtained for larger problem sizes. We have also analyzed the two different formulations of the parallel FFT algorithm namely the two-dimensional form and the three-dimensional form and noted that the three-dimensional form outperformed the two-dimensional form.

Acknowledgments

This work was supported in part by Sandia National Laboratories, the University of Chicago under NSF Grant #SCI-0414407, and the U.S. Department of Energy under Contract W-31-109-ENG-38.

References

[1] C. Clos, “A Study of Non-Blocking Switching Networks,” The Bell System Technical Journal, pages 406–421, 1953.

[2] E. O. Brigham. “The fast fourier transform. Spectrum,” 4:63–70, December 1967.

[3] P. Duhamel and M. Vetterli, “Fast Fourier transforms: a tutorial review and a state of the art,” Signal Processing 19, 259–299, 1990.

[4] W. M. Gentleman and G. Sande, “Fast Fourier transforms—for fun and profit,” Proceedings of *AFIPS* 29, 563–578, 1966.

[5] L. R. Johnson and A. K. Jain. “An efficient two-dimensional fft algorithm,” *PAMI*, 3(6):698–701, November 1981.

[6] J. C. Schatzman, “Accuracy of the discrete Fourier transform and the fast Fourier transform,” *SIAM J. Sci. Comput.* 17 (5), 1150–1166, 1996.

[7] R. C. Singleton, “On computing the fast fourier transform,” *CACM*, 10:647–654, 1967.

[8] R. C. Singleton, “An algol procedure for the fast fourier transform with arbitrary factors,” algorithm 339. *CACM*, 11:776–779, 1968.

[9] R. C. Singleton, “Algol procedures for the fast fourier transform,” algorithm 338. *CACM*, 11:773–776, 1968.

[10] S. W. Smith, “The scientist and engineer’s guide to digital signal processing,” California Technical Publishing, 1997.

[11] E. O. Brigham, “The Fast Fourier Transform and Its Applications, Englewood Cliffs,” NJ: Prentice-Hall, Inc., 448 pp., 1988.

[12] L. E. Miller, "Connectivity Properties of Mesh and Ring-Mesh Networks," April 2001. Available online: http://w3.antd.nist.gov/wctg/netanal/MeshCon.pdf

[13] G. Arfken, “Discrete Orthogonality--Discrete Fourier Transform,” §14.6 in Mathematical Methods for Physicists, 3rd ed. Orlando, FL: Academic Press, pp. 787-792, 1985.

[14] W.H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, “Fourier Transform of Discretely Sampled Data,” §12.1 in Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed. Cambridge, England: Cambridge University Press, pp. 494-498, 1989.

[15] J. O. Smith III, “Mathematics of the Discrete Fourier Transform (DFT), with Music and Audio Applications,” W3K Publishing, ISBN 0-9745607-0-7, 2003.

[16] A. Averbuch, E. Gabber, B. Gordissky, and Y. Medan, “A parallel FFT on an MIMD machine,” Parallel Computing, 15:61–74, 1990.

[17] D. H. Bailey, “FFTs in external or hierarchical memory,” The Journal of Supercomputing, 4:23–35, 1990.

[18] S. Bershader, T. Kraay, and J. Holland, “The giant-Fourier-transform,” In Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications: Volume I, pages 387–389, 1989.

[19] L. Desbat and D. Trystram, “Implementing the discrete Fourier transform on a hypercube vector-parallel computer,” In Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications: Volume I, pages 407–410, 1989.

[20] A.Gupta and V. Kumar, “On the scalability of FFT on parallel computers,” In Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation, 1990. Also available as Technical Report TR 90-53, Department of Computer Science, University of Minnesota, Minneapolis, MN.

[21] S. L. Johnsson, R. Krawitz, R. Frye, and D. McDonald, “A radix-2 FFT on the connection machine,” Technical report, Thinking Machines Corporation, Cambridge, MA, 1989.

[22] A. Norton and A. J. Silberger, “Parallelization and performance analysis of the Cooley-Tukey FFT algorithm for shared memory architectures,” IEEE Transactions on Computers, C-36(5):581–591, 1987.

[23] P. N. Swarztrauber, “Multiprocessor FFTs,” Parallel Computing, 5:197–210, 1987.
[24] S. G. Akl, “The Design and Analysis of Parallel Algorithms,” Prentice-Hall, Englewood Cliffs, NJ, 1989.
[25] V. Kumar, A. Grama, A. Gupta, G. Karypis, “Introduction to Parallel Computing,” Benjamin/Cummings, 1994.
[26] C. V. Loan, “Computational Frameworks for the Fast Fourier Transform,” SIAM, Philadelphia, PA, 1992.
[27] Michael J. Quinn, “Designing Efficient Algorithms for Parallel Computers,” McGraw-Hill, New York, NY, 1987.