AD-A100 782 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 12/1
EFFICIENT COMPUTER IMPLEMENTATIONS OF FAST FOURIER TRANSFORMS.(U)
DEC 80 J D BLANKEN
UNCLASSIFIED AFIT/GE/EE/80D-9 NL

1 of 4

AFIT/GE/EE/80D-9

EFFICIENT COMPUTER IMPLEMENTATIONS OF FAST FOURIER TRANSFORMS

THESIS

AFIT/GE/EE/80D-9 John D. Blanken
Captain USAF

(14) AFIT/GE/EE/80D-9

(6)
EFFICIENT COMPUTER IMPLEMENTATIONS OF FAST FOURIER TRANSFORMS

[illegible]

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

[illegible]

by

John D. Blanken, B.S.E.E.
Capt. USAF

Graduate Electrical Engineering

Acknowledgments

Dr. John Hines and Mr. Harold Noffke of the Air Force Wright Aeronautical Laboratory proposed this topic. I am indebted to them for both the topic and their support.

Gratitude is due Drs. Burrus and Parks of Rice University for many helpful discussions and ideas throughout this effort. They graciously provided the Prime Factor Algorithm studied and tested in this paper.

A special thanks is extended to Mr. Jim Thompson of Control Data Corporation and Dr. Poirier of Aeronautical Systems Division for their invaluable assistance in quantifying the multiply and add speed on the Cyber 74. A debt of gratitude is owed to my thesis readers Capt. Larry Kizer and Dr. Kabrisky. Their willingness to let me work independently was greatly appreciated.

My very deepest appreciation is extended to my faculty advisor Dr. Pedro Rustan for his patience, confidence, and skilled guidance. Most of all I appreciate his enthusiasm and hard work which motivated me to continue my efforts.

Finally, I am indebted the most to my wife, Linda, for her encouragement, typing the rough draft, and allowing me to sacrifice our family life and complete my work at AFIT.

John D. Blanken

This thesis was typed by Niki Maxwell

Contents

Contents

Contents

List of Figures

List of Figures

List of Figures

List of Tables

Glossary of Terms

1. Butterfly: The DFT computation of Figure 3.4 provides the notation whose appearance is that of a "butterfly".

2. Fixed Radix: The term "radix" is commonly used to describe a specific FFT decomposition. The term "fixed" radix means that all the factors of N are the same.

3. Mixed Radix: All the factors of N are not identical.

4. Relatively Prime: The numbers in a given set are said to be relatively prime when no number in the set is divisible (with no remainder) by any other number in the set. Example, (2, 3, 7, 9) are not relatively prime sets because 9 is divisible (with no remainder) by 3. The following example is relatively prime: (2, 3, 5, 7).

5. Square and Square-free Factors: For the case where $N = 4 \cdot 3 \cdot 7 \cdot 4$, the "4s" are square factors and the 3 and 7 are square-free.

6. Twiddle Factors: The term refers to the complex multipliers of Figure 3.8 which pre-multiply the FFT butterflies. They are sometimes called phase or rotation factors.

Abstract

A comprehensive comparison of the most efficient Discrete Fourier Transform (DFT) techniques is presented. The DFT algorithms selected are the fixed radix Fast Fourier Transform (FFT), mixed radix FFT, the Winograd Fourier Transform Algorithm (WFTA), and the Prime Factor Algorithm (PFA). Comparison of the alqorithms is based on the number of real multiplications, additions, and memory arrays required as a function of sequence length N. This paper reviews the literature, selects the most efficient DFT FORTRAN programs available, develops the number of real multiplications and additions as a function of N, and compares the algorithms using tables and plots of real multiplications, additions, and memory arrays. This comparison shows that the WFTA and PFA require the least real multiplications and additions, but the fixed radix and mixed radix FFTs require the least memory. The mixed radix FFT is much more flexible than WFTA or PFA since N can be any length sequence. The WFTA and PFA are closely studied and tradeoffs between the two are discussed. The PFA uses less additions but more multiplications for most sequence lengths which means the WFTA is more efficient when multiplications are "costly" relative to additions. The PFA uses less memory than the WFTA making the PFA preferable when the machine memory is limited. Based on the results of the paper, an algorithm is presented to select the most efficient DFT for an N length sequence given the multiply speed, add speed, and memory size of the computer.

I. Introduction

1.1 Background

Computing the Discrete Fourier Transform (DFT) of N points has many applications in scientific and engineering calculations. In 1965 Cooley and Tukey described an algorithm which became known as the Fast Fourier Transform (FFT) because it reduced the number of complex operations required to compute the DFT from N^2 to $N \log_2 N$ where $N=2^m$, m an integer. Using ideas proposed in the Cooley-Tukey paper a mixed radix algorithm was written and published in 1969 by Singleton which permitted N to be any positive integer length sequence.

In 1976 Winograd proposed a mixed radix DFT algorithm which (1) converted the DFT to circular convolution, (2) used fast convolution algorithms to perform "short-DFTs", and (3) nested these short-DFTs into a structure to perform long Fourier transforms on complex data sequences. This algorithm became known as the Winograd Fourier Transform Algorithm (WFTA). The WFTA maintained the real additions count at the FFT levels while significantly reducing the real multiplications required.

Kolba and Parks, 1977, used Winograd's fast convolution algorithms and proposed a new Prime Factor Algorithm

(PFA). This new algorithm modified the short-DFTs to use "shifts" instead of multiplication by 1/2 and did not use the nested structure of WFTA. As a consequence the PFA uses more real multiplications and less additions relative to the WFTA for a given length sequence N.

1.2 Problem

Both Winograd, 1976, and Kolba-Parks, 1977, compared their operations count to that of the FFT but did not include all possible WFTA and PFA sequence lengths. Further, no comparisons were made on the basis of memory arrays required by each algorithm as a function of N. This paper presents a comprehensive comparison of fixed radix FFTs, mixed radix FFTs, WFTA, and PFA based on real operations and memory arrays. This comparison provides the information needed to select the most efficient algorithm to perform the DFT based on machine size, machine speed, and real operations.

1.3 Scope

This paper reviews the literature, selects DFT algorithms for comparison, studies the theory of each algorithm selected, develops the real operation and memory count as a function of N, compares these algorithms using tables and plots of operation and memory counts, and presents an algorithm to select the most efficient techniques.

The DFT algorithms selected for study and comparison are:

(1) Radix-2 FFT

(2) Radix-3 FFT

(3) Radix-3 FFT in the R(u) field

(4) Radix-5 FFT

(5) Mixed radix FFT written by the author

(6) Mixed radix FFT written by Singleton

(7) Mixed radix FFT available from International Mathematical Subroutine Library (IMSL) on the CDC Cyber 74

(8) WFTA

(9) PFA.

Each of these algorithms has a particular advantage which makes selection of the best algorithm dependent on the machine size, machine speed, and sequence length.

1.4 Assumptions

To a first approximation, the speed of an FFT algorithm is proportional to the number of complex multiplications used. The number of times the data array is indexed is, however, an important secondary factor (Singleton, 1969). Kolba and Parks, 1977, substantiated this assumption by timing the PFA and FFTs on an IBM 370/155 for several sequence lengths and showing that the FORTRAN coded PFA (having less real additions and multiplications) was faster than the FFT FORTRAN algorithms.

In 1978 Morris demonstrated that the sequence of arithmetic operations in a DFT algorithm's internal structure can result in different execution times "between ostensibly equivalent algorithms on a given machine" and that the computer dependent algorithm/architecture interactions may also alter relative performance of the different algorithms. He modified the FORTRAN coded radix-4 FFT and WFTA programs and matched them to the PDP 11/55 and IBM 370/168 architecture and showed that the WFTA offered neither time or space advantages over the radix-4 FFT. Morris achieved these results because "the radix-4 FFT appears almost ideally matched to the PDP-11 architecture" whereas the WFTA "has extra load/store burdens" and requires extra data array indexing.

Morris demonstrated that it may be possible to optimize DFT algorithms to match a certain machine, however, this type of optimization of the FORTRAN DFT algorithms is outside the scope of this paper. It is assumed that existing FORTRAN coded DFT algorithms will not be modified and selecting an algorithm which minimizes real operations produces the most efficient algorithm.

This paper derives and tabulates real operations counts as a function of N for the algorithms listed in Section 1.3. The most efficient DFT algorithms are timed on the CDC Cyber 74 computer and compared to the predicted execution time based on real operations. These predicted times are shown to be consistent with the timing results.

1.5 Approach and Presentation

A literature review is presented in Chapter II which starts with the 1965 Cooley-Tukey paper and follows the various DFT algorithm developments up through Kolba-Parks' 1977 article. The review puts Rader's 1968 landmark paper in perspective with Winograd's "nested" DFT algorithm and the subsequent work by Kolba and Parks.

Next, the theory behind the DFT algorithms is reviewed, the real operations count developed, and the memory array count needed for a sequence length N is determined. The general expressions for real operations and memory array counts are developed from published articles or from the background theory and then plotted and tabulated as a function of N. The readers familiar with the FFT and Winograd background theory may wish to skip Sections 3.1 and 3.2.

In Chapter IV comparison tables and plots of the DFT algorithms make it possible to select the most efficient algorithm based on real operations and memory array required. Timing results from the CDC Cyber 74 system for representative sequence lengths are tabulated to substantiate the assumption that minimizing real operations equates to maximizing efficiency. An algorithm is also presented at the end of Chapter IV which uses the tables in this paper to select the most efficient DFT technique given the sequence length, memory size, and computer add and multiply speed.

Conclusions and recommendations are presented in Chapter V.

II. LITERATURE REVIEW

The calculation of the Discrete Fourier Transform (DFT) is a central operation performed in digital signal processing but was not widely used for other than trivial sequence lengths because of the cumbersome DFT evaluation:

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-j2\pi nk/N) \tag{2.1}$$

which required on the order of N^2 complex operations.

In 1965 Cooley and Tukey published "An Algorithm for the Machine Calculation of Complex Fourier Series" which stimulated the widespread use of an algorithm which became known as the "Fast Fourier Transform" (FFT). Their paper proposed an efficient method of computing the DFT by factoring an N length sequence into its prime components:

$$N = n_1\ n_2\ \ldots\ n_m \tag{2.2}$$

and then decomposing Eq (2.1) into m steps with N/n_i transformations within each step. If $n_1=n_2= \ldots n_m=2$, the operations are reduced to the $N \log_2 N$ level from the previous N^2 level.

Most of the early work on the FFT (Bergland, 1968) was directed toward the special cases where $N=2^m$ which yielded simple and efficient algorithms. These algorithms are efficient because no multiplications are needed to evaluate the 2-point DFT butterflies which can reduce the operations count below the $N \log_2 N$ level.

Other "fixed radix" algorithms were studied and Dubois and Venetsanopoulos published "A New Radix-3 Algorithm" in 1978 which demonstrated that a radix-3 butterfly could be computed without multiplications by defining a new basis (1,u) instead of using the complex plane (1,i) basis, where u is the complex cube root of unity. This technique was later shown to be limited to the special cases of 3^m and 6^m (Burrus and Parks, 1979).

Based on Cooley and Tukey's paper "mixed-radix" algorithms were written by Brenner and Singleton. The most efficient and popular of these algorithms was "An Algorithm For Computing the Mixed Radix Fast Fourier Transform" published in 1969 by Singleton and is frequently used in digital signal processing where a wider choice of N is needed. The Singleton algorithm can perform the DFT using FFT techniques of any length sequence N but becomes most efficient when N is highly composite from the set of integers 2, 3, 4, and 5. If N is a prime number the algorithm performs a DFT using N^2 operations. The Singleton algorithm became the standard against which all future DFT techniques were measured.

In 1968 Rader presented "DFTs when the Number of Data Samples Is Prime" which showed that a prime number length sequence contains an (N-1) point circular convolution. He showed how to isolate the convolution by applying a permutation to the (N-1) signal points x(1), x(2), ... , x(N-1). He also gave the permutation applied to the complex multipliers from the set $[\exp(-j2\pi nk/N), k=1,2, \ldots, N-1]$. Both of the permutations were generated by using a "primitive" root which exists for N length prime sequences (McClellan and Rader, 1979). Rader's paper was largely overlooked for many years but took on new significance when Winograd presented his new DFT algorithm "On Computing the Discrete Fourier Transform" in 1976.

Winograd combined Rader's idea of converting a DFT to circular convolution with his own fast convolution algorithms to produce a new DFT method called the "Winograd Fourier Transform Algorithm" (WFTA). Winograd provided the fast convolution algorithms for short prime and prime power length sequences and proposed that longer transforms be computed by "nesting" the short-high speed transforms. He presented a table comparing the WFTA to the radix-2 FFT operations and showed that the number of additions remained at the FFT levels while the number of multiplications was significantly reduced.

Kolba and Parks published "A Prime Factor FFT Algorithm Using High Speed Convolution" in 1977 which modified Winograd's fast convolution algorithms to permit "shifts" instead of multiplications by 1/2. They also changed the nested structure of the WFTA in favor of a conventional FFT decomposition. The decomposition of the sequence was based on an algorithm proposed by Thomas, 1963, in his article "Using a Computer to Solve Problems in Physics" which uses an index mapping based on the Chinese Remainder Theorem.

Kolba and Parks selected several N length sequences and compared their operations count to WFTA and FFT.

Paralleling Winograd's fast convolution work are the studies into number theoretic transforms (NTTs) which have been proposed for digital cyclic convolution and digital filtering. The NTTs were first published by Pollard, 1971, in "The Fast Fourier Transform in the Finite Field". He showed that an analogous transform to the DFT exists in the finite (or Galois) field where $\exp(j2\pi nk/N)$ terms are replaced by r^{nk} in the DFT expression such that:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, r^{nk} \qquad (2.3)$$

Notice that Pollard chose the alternative definition of the DFT where the exponent of e is positive. The r term is defined in the Galois field (GF) such that the same cyclic convolution properties exist in GF and in the complex field for the DFT. He then proved that this analogous DFT could apply prime factor decomposition to the N length sequence and perform N/n_i transformations to reduce the operations in GF to the $N \log_2 N$ level which provided the FFT in GF. Pollard proposed that this technique be applied to cyclic convolutions in GF, multiplication of polynomials over $GF(p^n)$, aperiodic convolution of integer sequences, multiplication of very large integers, division of polynomials over GF(p), and a chirp-Z-transform for NTTs (McClellan and Rader, 1979).

Pollard's paper stimulated more study of the NTTs. Reed and Truong's 1975 paper, "The Use of Finite Fields to Compute Convolutions", includes complex valued NTTs. It was shown that this NTT over GF(q^2) can reduce convolution operations to the FFT levels. If q is sufficiently large the NTT can be used over GF(q^2) to transform a sequence of complex integers x(n) into X(k) on GF(q^2) for which the inverse transform of X(k) on GF(q^2) is precisely the original sequence x(n). Using these ideas filtering or convolutions without roundoff errors can be obtained on a sequence of complex integers.

Most applications of the NTTs have been in the areas of digital filtering and convolution. The author was not able to find any NTT algorithm which could be compared to the FFT, WFTA, or PFA and perform all the same functions as these three algorithms.

PFA, WFTA, and FFT represent the most efficient and flexible FORTRAN programs available to perform the DFT. Each algorithm has its own particular advantage over the other two depending on machine size and speed for a particular sequence length. None of the articles reviewed presents a comprehensive evaluation or comparison of the three algorithms based on real operations and memory arrays required to perform a DFT for any sequence length N. This paper fills that need so that an efficient algorithm can be selected.

III. FFT Theory

The set of algorithms known as the Fast Fourier Transforms (FFT) use a variety of methods to reduce the computation time required to evaluate the Discrete Fourier Transform (DFT). The DFT is the central part in most spectrum analysis problems and the FFT can improve performance by a factor of 100 or more over direct evaluation of the DFT (Rabiner and Gold, 1975). Therefore, the FFT is crucially important to the digital signal processing techniques.

This section begins with "fixed radix" FFT algorithms by discussing a "decimation-in-time" algorithm, the data reordering (bit reversal) theory, the real operations (addition and multiplication) count, a new fixed radix algorithm in the finite field, and then summarizes the memory required to use the fixed radix algorithms. Next the conventional "mixed" radix algorithms are presented by discussing the theory, digit reversal, real operations count, and memory required to utilize the mixed radix algorithms. This theory chapter concludes with a discussion of mixed radix algorithms based on fast convolution. The theory, data reordering, real operations count and memory are also presented for these algorithms.

Before discussing the FFT algorithms comments must be made relative to computing the trigonometric function values needed to evaluate the FFT.

3.1 Computing Trigonometric Function Values

The trigonometric values used in FFTs can be represented as values on the unit circle. The values are based on integer powers of

$$\exp(-j2\pi/N)$$

which can be computed using sine and cosine functions. It is useful to have accurate methods of generating the sine and cosine terms other than the method of repeated use of library sine and cosine functions.

The method most widely used in FFT algorithms (Singleton, 1967) generates the trigonometric functions by a difference equation given by:

$$\cos((k+1)a) = (C \cdot \cos(ka) - S \cdot \sin(ka)) + \cos(ka)$$

$$\sin((k+1)a) = (C \cdot \sin(ka) + S \cdot \cos(ka)) + \sin(ka)$$

where

$$C = -2 \sin^2 (a/2)$$

$$S = \sin(a)$$

$$\cos(0) = 1$$

$$\sin(0) = 0$$

This technique is used for all FFTs presented in this paper (except noted otherwise) because it minimizes using FORTRAN library subroutines cos (•) and sin (•) thereby reducing the overall FFT computation time.

3.2 Fixed Radix Algorithms

While FFT algorithms are well known and widely used, they are relatively intricate and somewhat difficult to grasp at first reading. There are two excellent textbooks (Rabiner and Gold, 1975; Oppenheim and Schafer, 1975) which discuss the FFT theory in great detail and present FFTs based on decimation-in-time and frequency. Both texts spend a great deal of time discussing the radix-2 FFT, which is the most widely known and used. For this reason, the radix-2 development is presented here as a convenience for the reader and provides a theoretical background from which the other fixed radix algorithms are derived.

3.2.1 Development of Radix-2 Theory.

To achieve the reduction in complex operations (defined as four real multiplications and two real additions) from N^2 to $N \log_2 N$ it is necessary to decompose the DFT computation into smaller and smaller DFT computations. As a result, the symmetry and periodicity of the complex exponential $\exp(-j2\pi nk/N) = W_N^{nk}$ can be exploited. This radix-2 algorithm is based on decomposition of the sequence $x(n)$ from the DFT expression:

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-j2\pi nk/N) \qquad (3.1)$$

$$k = 0, 1, \ldots, N-1 \text{ and } N=2^m$$

which is known as a "decimation-in-time" algorithm (Oppenheim and Schafer, 1975). Since N is an even integer, X(k) can be computed by separating x(n) into two N/2 length sequences consisting of even-numbered points and the odd-numbered points in x(n). Using n=2r for n even and n=2r+1 for n odd Eq (3.1) becomes:

$$X(k) = \sum_{r=0}^{T} x(2r)W_N^{2rk} + \sum_{r=0}^{T} x(2r+1)W_N^{(2r+1)k} \tag{3.2}$$

where T=(N/2)-1 and $W_N = \exp(-j2\pi/N)$. By expanding $W_N^{(2r+1)k}$ and factoring out W_N^k Eq (3.2) can be rewritten as:

$$X(k) = \sum_{r=0}^{T} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{T} x(2r+1)(W_N^2)^{rk} \tag{3.3}$$

But $W_N^2 = \exp(-j4\pi/N) = \exp(-j2\pi/(N/2)) = W_{N/2}$ and Eq (3.3) can be written as:

$$X(k) = \sum_{r=0}^{T} x(2r)W_{N/2}^{rk} + W_N^k \sum_{r=0}^{T} x(2r+1)W_{N/2}^{rk}$$

$$= G(k) + W_N^k H(k) \tag{3.4}$$

Each of the sums in Eq (3.4) is an N/2 point DFT, the first sum being the even numbered points of the original sequence and the second sum being the odd numbered points of the original sequence. Although the index k = 0,1,...,N-1, each of the sums in Eq (3.4) need only be computed over k = 0, 1, ..., (N/2)-1, since G(k) and H(k) are periodic in k with period N/2. After the two DFTs in Eq (3.4) are computed, they are then combined to yield the N-point DFT, X(k). Figure 3.1 indicates the computation involved in computing X(k) according to Eq (3.4) for an eight-point

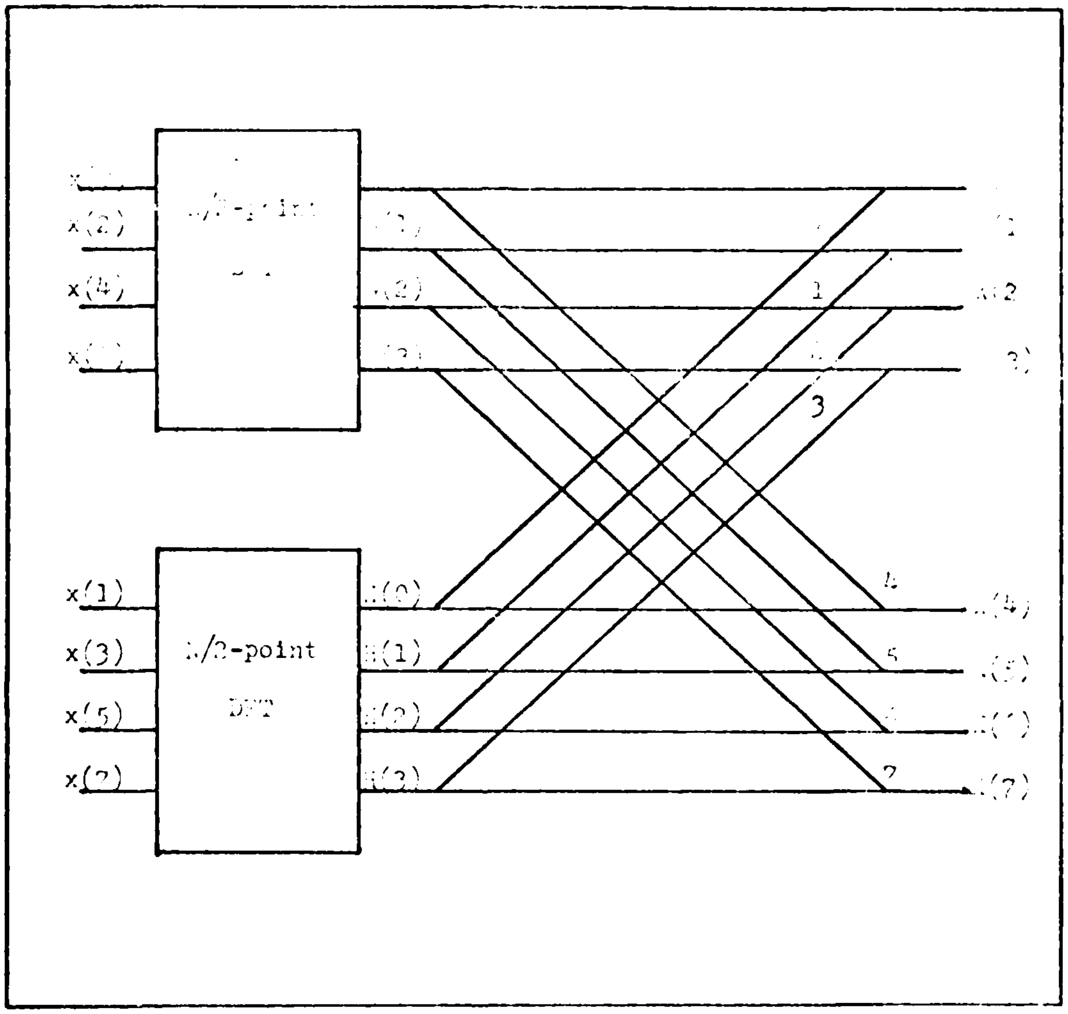


Figure 3.1. Flowgraph of the Decimation-In-Time Decomposition of an N-Point DFT Computation into Two N/2-Point DFT Computations (N=8).

NOTE: The integers on the branches of the flowgraph represent the powers of W_N; i.e., the "4" represents W_N^4.

sequence. Figure 3.1 (Oppenheim and Schafer, 1975) uses the signal flow conventions such that branches entering a node are summed to produce the node variable. When no coefficient is shown the branch transmittance is assumed to be one. For other branches the transmittance of a branch is an integer power of W_N. Note in Figure 3.1 that two four-point DFTs are computed using G(k) and H(k). X(0) is obtained by multiplying H(0) by W_N^0 and adding the product to G(0). X(1) is obtained by multiplying H(1) by W_N^1 and adding the result to G(1). For X(4) it would follow that H(4) is multiplied by W_N^4 and added to G(4), however, since G(k) and H(k) are both periodic in k with period 4, H(4) = H(0) and G(4) = G(0). Thus X(4) results from multiplying H(0) by W_N^4 and adding the produce to G(0).

With the computation of the N-point DFT of Eq (3.4) that number of computations can be compared with the direct DFT computation of Eq (3.1). For the direct computation without using symmetry properties N^2 complex multiplications were required. Eq (3.4) requires computation of two N/2-point DFTs, which require $2(N/2)^2$ complex multiplications and about $2(N/2)^2$ complex additions (Oppenheim and Schafer, 1975). The two N/2-point DFTs must be combined, requiring N complex multiplications corresponding to multiplying the second sum by W_N^k and then N complex additions, corresponding to adding the product to the first sum. As a result, the computation of Eq (3.4) for all values of k requires

$N + 2(N/2)^2$ or $N + (N^2/2)$ complex multiplications and additions. For $N>2$, $N + N^2/2$ is less than N^2.

The expression in Eq (3.4) corresponds to decimating the original N-point sequence into odd and even N/2-point sequences. Since $N=2^m$ the N/2-point sequences are also even and then each G(k) and H(k) can be further decimated into two N/4-point DFTs, which could then be combined to yield the N/2-point DFTs. Decimating the N/2-point sequences in Eq (3.4) into N/4-point sequences gives:

$$G(k) = \sum_{r=0}^{(N/2)-1} g(r)W_{N/2}^{rk}$$

$$= \sum_{p=0}^{(N/4)-1} g(2p)W_{N/2}^{2pk} + \sum_{p=0}^{(N/4)-1} g(2p+1)W_{N/2}^{(2p+1)k}$$

Letting $R = (N/4)-1$,

$$G(k) = \sum_{p=0}^{R} g(2p)W_{N/4}^{pk} + W_{N/2}^{k} \sum_{p=0}^{R} g(2p+1)W_{N/4}^{pk} \qquad (3.5)$$

Similarly,

$$H(k) = \sum_{p=0}^{R} h(2p)W_{N/4}^{pk} + W_{N/2}^{k} \sum_{p=0}^{R} h(2p+1)W_{N/4}^{pk} \qquad (3.6)$$

If the four-point DFT in Figure 3.1 are computed using Eq (3.5) and (3.6) then that computation would be carried out as indicated in Figure 3.2. Inserting the computation in Figure 3.2 into the flowgraph of Figure 3.1 produces the complete flowgraph in Figure 3.3. Note that $W_{N/2} = W_N^2$ was used.

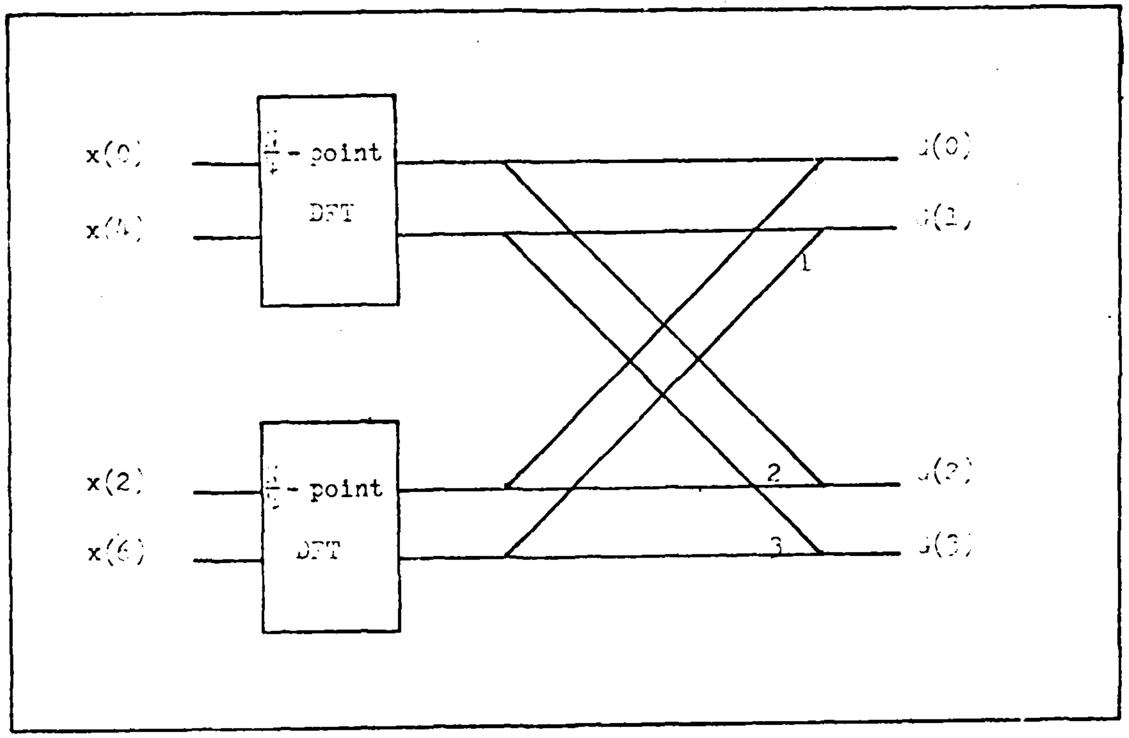


Figure 3.2. Flowgraph of the Decimation-In-Time Decomposition of an N/2-Point DFT Computation into Two N/4-Point DFT Computations (N=8).

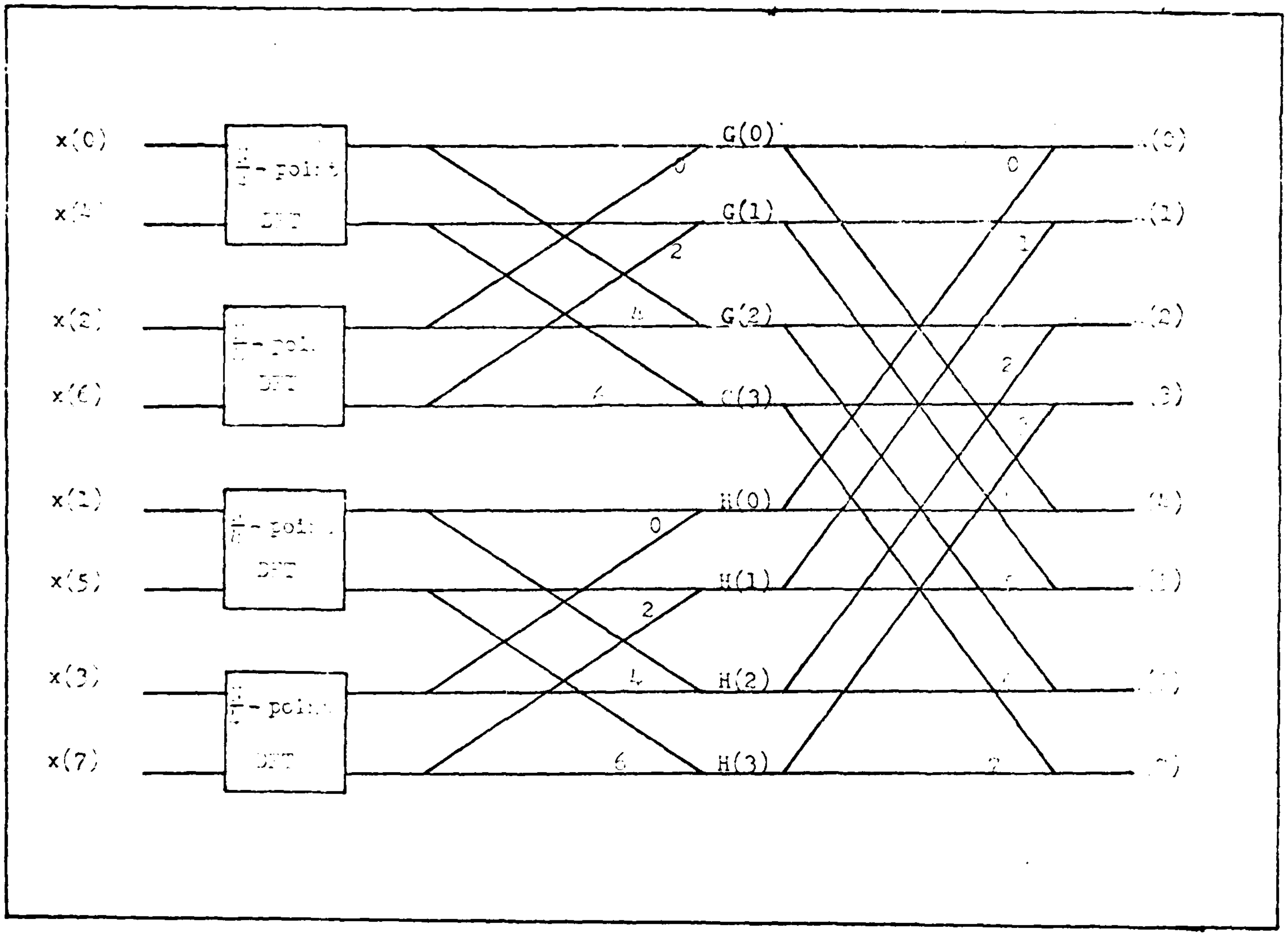


Figure 3.3. Result of Substituting Figure 3.2 into Figure 3.1.

For the 8-point DFT that has been used as an example, the computation has been reduced to a computation of N/4-point DFTs where $N/4=2$. An example, 2-point DFT for $x(0)$ and $x(4)$ is shown in Figure 3.4. The complete flowgraph for the computation of the 8-point DFT is shown in Figure 3.5 and was obtained with the computation of Figure 3.4 and inserting it in Figure 3.3.

Considering the more general case with N a power of 2 greater than 3 the same decimation procedure would be continued by decomposing the N/4-point transforms in Eqs (3.5) and (3.6) into N/8-point transforms. This requires v stages of computation where $v = \log_2 N$. Recall that in the original decomposition of the N-point transform into two N/2-point transforms, the number of complex multiplications and additions required was $N + 2(N/2)^2$. When the N/2-point transforms were decomposed into N/4-point transforms the factor of $(N/2)^2$ is replaced by $N/2 + 2(N/4)^2$ so that the overall computation now requires $N + N + 4(N/4)^2$ complex multiplications and additions. If $N=2^v$ this can be done at most $v = \log_2 N$ times, "so that after carrying out this decomposition as many times as possible the number of complex multiplications and additions is equal to $N \log_2 N$" (Oppenheim and Schafer, 1975).

The flowgraph of Figure 3.5 displays the operations explicitly. By counting branches with transmittances of the form W_N^r it is seen that each stage has N complex

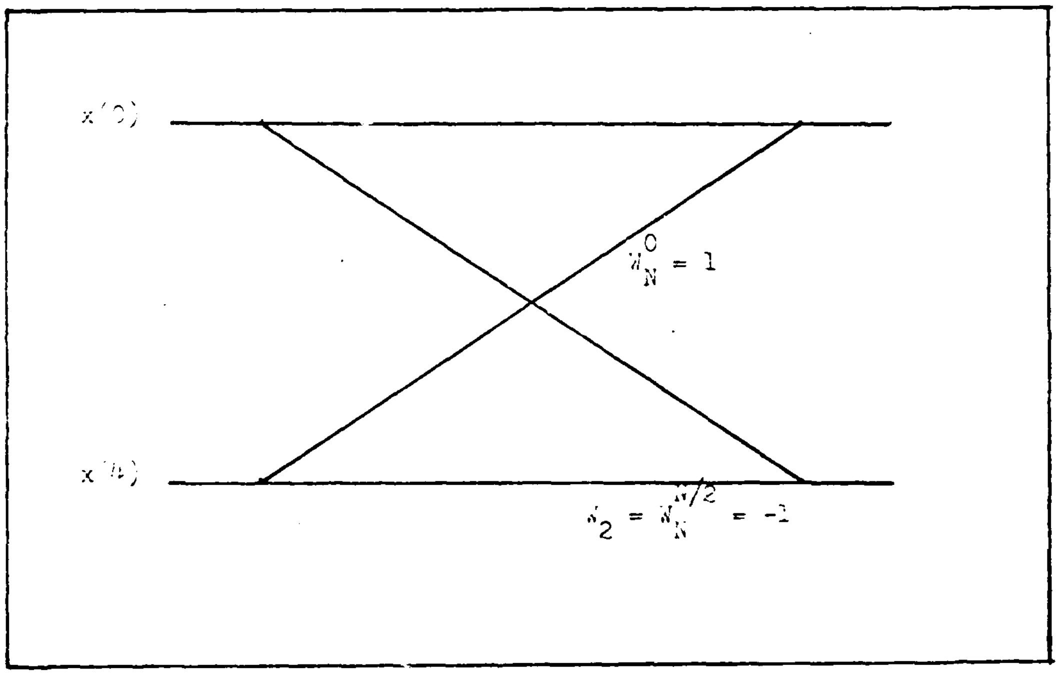


Figure 3.4. Flowgraph of Two-Point DFT.

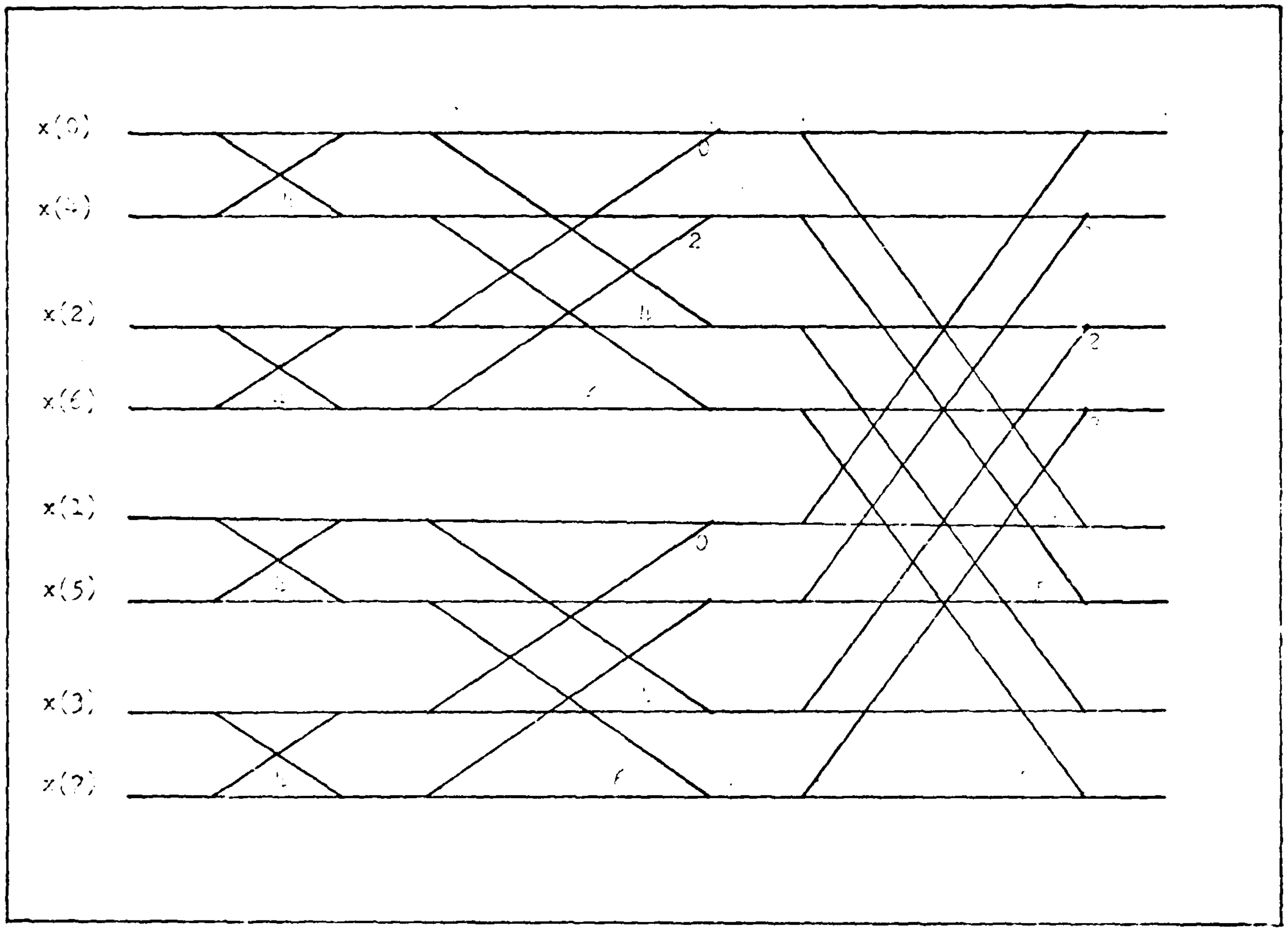


Figure 3.5. Flowgraph of Complete Decimation-In-Time Decomposition of an 8-Point DFT.

multiplications and N complex additions. Since there are $\log_2 N$ stages there are a total of $N \log_2 N$ complex multiplications and additions as shown before. Further reductions in the complex operations count can be achieved by exploiting the symmetry and periodicity of W_N^r.

Note that on each "stage" of Figure 3.5 the computation takes a set of N complex numbers and transforms them into another set of N complex numbers. This process is repeated $v=\log_2 N$ times resulting in the DFT computation. For example, in computing the first stage of Figure 3.5 one set of storage registers would contain the input data sequence and a second set of storage registers would contain the computed results for the first stage. The sequence of numbers resulting from the m th stage of computation is denoted as $X_m(i)$, where $i = 0, 1, \ldots, N-1$ and $m = 1, 2, \ldots, v$. For the following stage, the previous output array, $X_m(i)$, becomes the input array and the new output array is $X_{m+1}(i)$ for the (m+1) stage of computation. Using this notation, it can be seen that the basic flowgraph in Figure 3.5 is given by Figure 3.6. Using the notation of Figure 3.6 the equations of the butterfly are given by:

$$X_{m+1}(p) = X_m(p) + W_N^r X_m(q) \tag{3.7}$$

$$X_{m+1}(q) = X_m(p) + W_n^{r+N/2} X_m(q) \tag{3.8}$$

Because of the appearance of Figure 3.6 the computation of Eqs (3.7) and (3.8) are referred to as the "butterfly" computations.

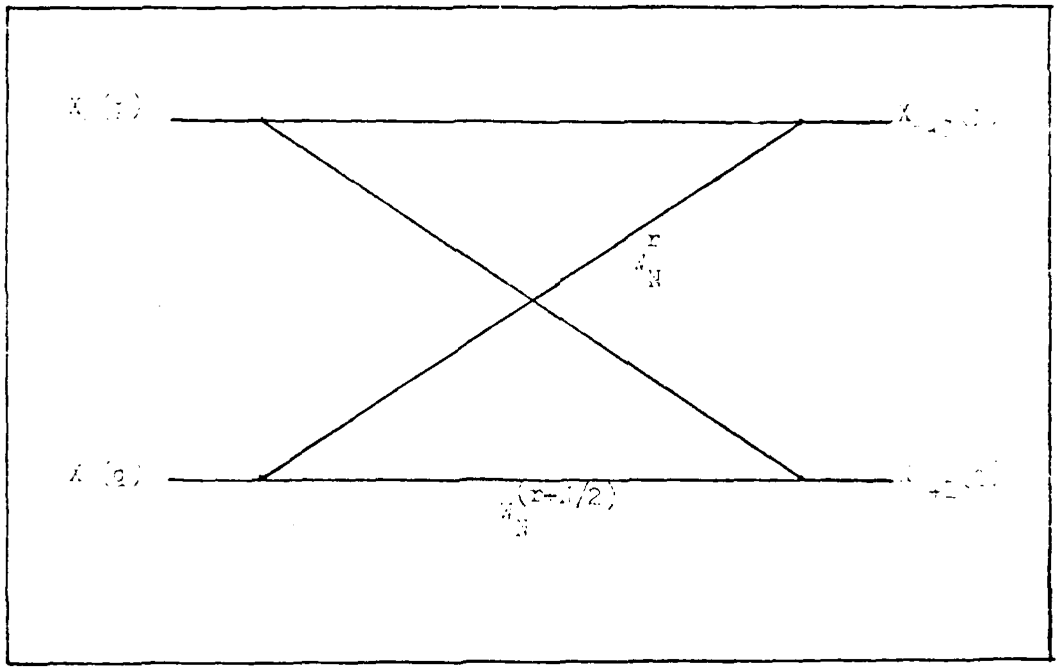


Figure 3.6. Flowgraph of Basic Butterfly Computation.

The number of complex multiplications can be reduced by a factor of 2 using the symmetry:

$$W_N^{N/2} = \exp(-j(2\pi/N) \cdot N/2) = \exp(-j\pi) = -1 \qquad (3.9)$$

so that the Eq (3.7) becomes:

$$X_{m+1}(p) = X_m(p) + W_N^r X_m(q) \qquad (3.10)$$

$$X_{m+1}(q) = X_m(p) - W_N^r X_m(q) \qquad (3.11)$$

Eqs (3.10) and (3.11) are shown in Figure 3.7 which reflects the "twiddle factor" W_N^r out front in the butterfly. Since there are N/2 "butterflies" of the form of Figure 3.7 per stage and $\log_2 N$ stages, the total number of complex multiplications required is $(N/2) \log_2 N$ instead of the $N \log_2 N$ used in Figure 3.5. Using the "twiddle factor" butterfly flowgraph of Figure 3.6 as a replacement for the butterfly of Figure 3.4, the Figure 3.8 is obtained.

3.2.2 <u>Development of Radix-3 FFT Theory</u>. Starting with the restriction that the N-point sequence be an integer power of three ($N = 3^m$, m = 1, 2, 3, ...), the DFT X(k) was computed by separating the discrete time sequence s(n) into three N/3 point sequences. X(k) is given by the DFT expression:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad \text{where } k = 0,1, \ldots, N-1 \text{ and } W_N = \exp(-j2\pi/N) \qquad (3.12)$$

Breaking x(n) into three N/3 point sequences yields x(3r), x(3r+1) and x(3r+2). Substituting these into Eq (3.12) and adjusting the respective summations to (N/3)-1 yields:

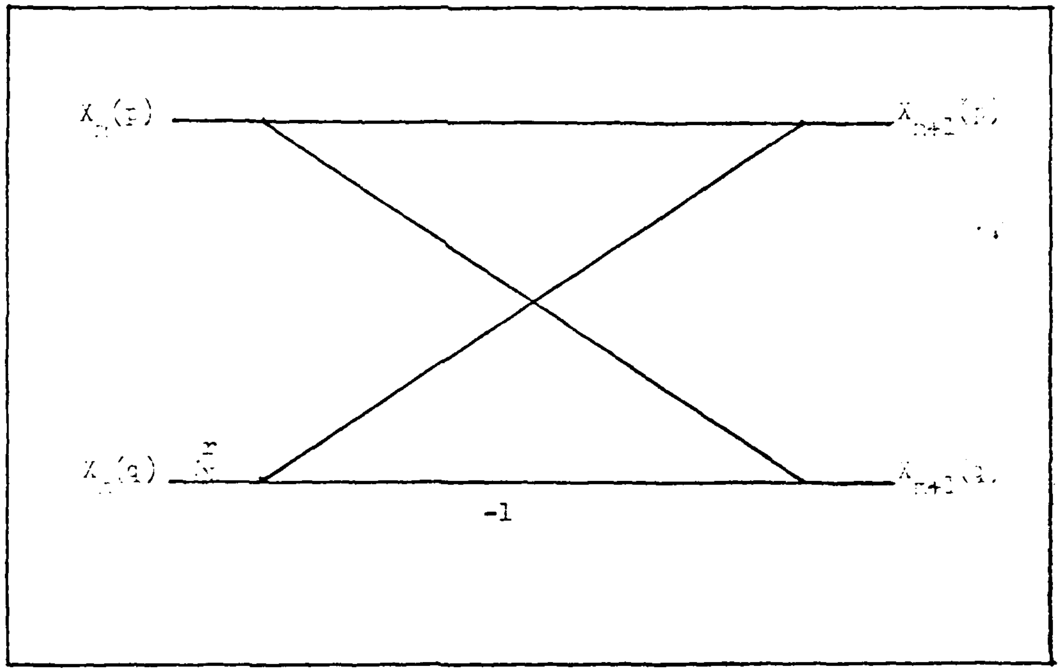


Figure 3.7. Flowgraph of Simplified Butterfly Computation.

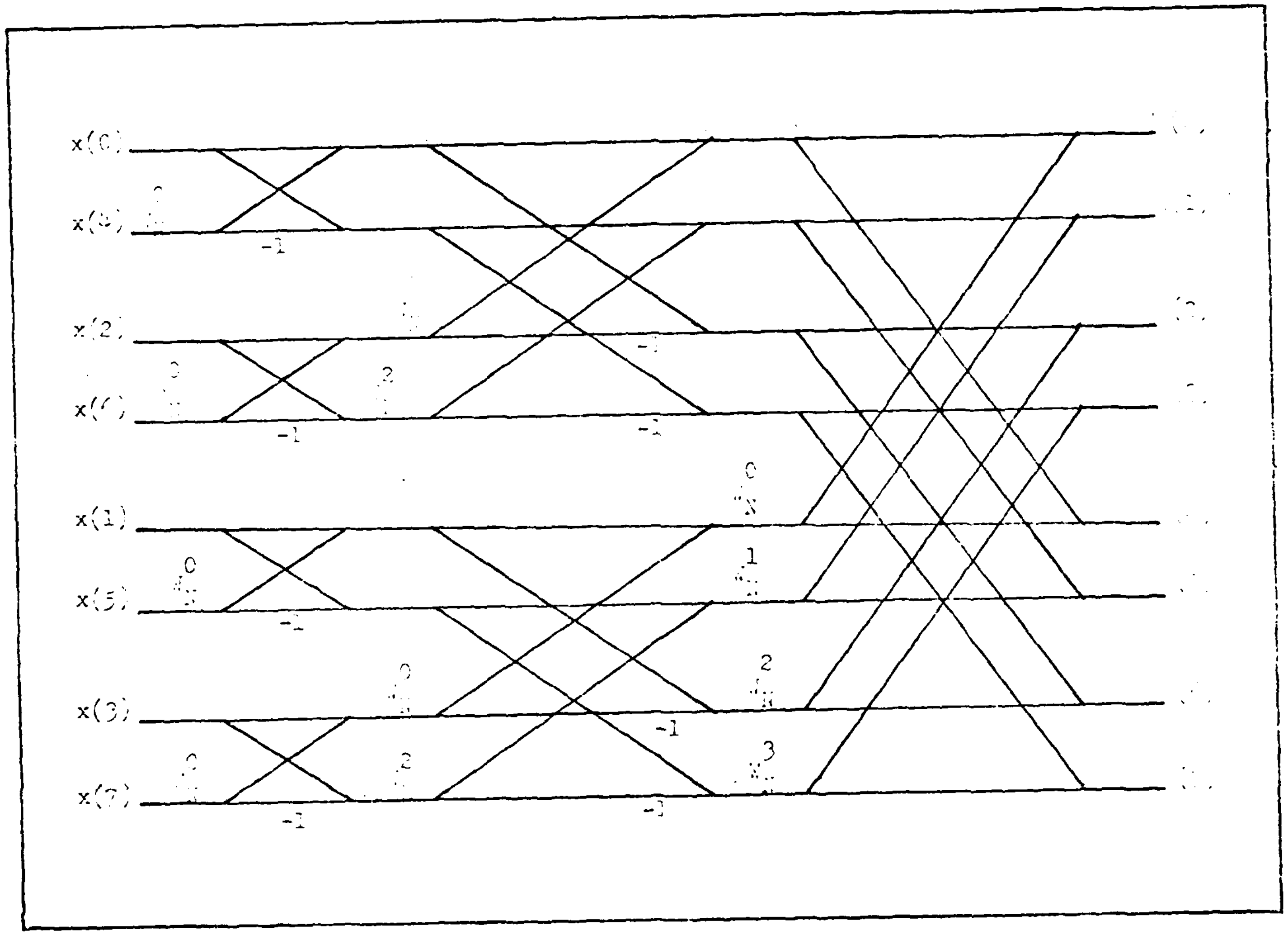


Figure 3.8. Flowgraph of 8-Point DFT Using the "Twiddle Factor" Butterfly of Figure 3.7.

$$X(k) = \sum_{r=0}^{P} x(3r)W_N^{(3r)k} + \sum_{r=0}^{P} x(3r+1)W_N^{(3r+1)k} + \sum_{r=0}^{P} x(3r+2)W_N^{(3r+2)k}$$

where $P = (N/3)-1$ (3.13)

By regrouping the exponents of W_N Eq (3.13) can be rewritten as:

$$X(k) = \sum_{r=0}^{P} x(3r)W_N^{3rk} + W_N^{k}\sum_{r=0}^{P} x(3r+1)W_N^{3rk} + W_N^{2k}\sum_{r=0}^{P} x(3r+2)W_N^{3rk} \quad (3.14)$$

By rewriting W_N^3 as:

$$W_N^3 = \exp(-j6\pi/N) = \exp(-j2\pi/(N/3)) = W_{N/3} \quad (3.15)$$

Eq (3.14) can be expressed as:

$$X(k) = \sum_{r=0}^{P} x(3r)W_{N/3}^{rk} + W_N^{k}\sum_{r=0}^{P} x(3r+1)W_{N/3}^{rk} + W_N^{2k}\sum_{r=0}^{P} x(3r+2)W_{N/3}^{rk} \quad (3.16)$$

Each of the sums in Eq (3.16) represents an N/3 point DFT: the first being the N/3 DFT of the 3r points in the original sequence, the second being the N/3 points of 3r+1, and the third being the N/3 points of 3r+2 points of the original sequence. Although the index k of X(k) ranges over N values (k = 0, 1, ..., N-1) each of the summations in Eq (3.16) needs computation over (N/3)-1 points. Eq (3.16) can be rewritten to reflect this:

$$X(k) = F(k) + W_N^k G(k) + W_N^{2k} H(k) \tag{3.17}$$

Eq (3.17) can be implemented into the butterfly flowgraph in Figure 3.9 using the accepted *notational conventions* (Oppenheim and Schafer, 1975). The convention used for the flowgraph is when no coefficient is shown, the branch transmittance is assumed to be one. For other branches the transmittance (multiplier) is an integer power multiplier of W_N. In Figure 3.9 there are three N/3 point DFTs and these are computed with F(k) designating the three point DFT of the 3r points, G(k) designating the three point DFT of 3r+1, and H(k) designating the DFT of 3r+2 points, where r = 0, 1, ..., (N/3)-1.

X(0) is obtained by (1) multiplying H(0) by a branch transmittance of 1 (which equals W_N^0), (2) multiplying G(0) by 1, (3) multiplying F(0) by 1, and (4) summing the three. Likewise, X(1) is obtained by multiplying H(1) by W_N^2, multiplying G(1) by W_N^1, and adding the results to F(1). X(6) has H(6) multiplied by W_N^{12} and G(6) multiplied by W_N^6 and the products added to F(6) giving:

$$X(6) = F(6) + W_N^6 G(6) + W_N^{12} H(6) \tag{3.18}$$

However, since F(k), G(k), and H(k) are all periodic in k with period N/3=3, the periodicity can be exploited to yield F(6) = F(0), G(6) = G(0), and H(6) = H(0). These results can be substituted into Eq (3.18) to give:

$$X(6) = F(0) + W_N^6 G(0) + W_N^{12} H(0) \tag{3.19}$$

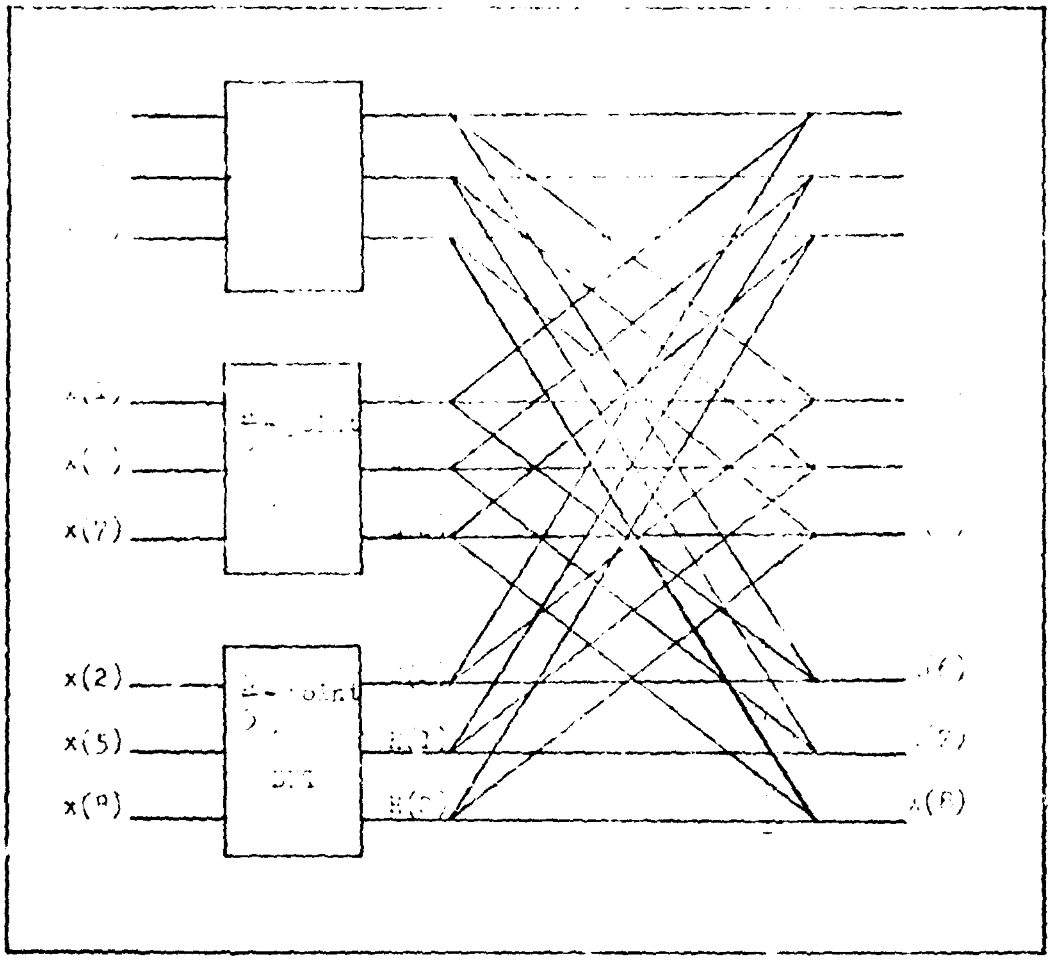


Figure 3.9. Butterfly Flowgraph for First Stage Decimation (N=9).

NOTE: The numbers on the branch transmittances represent exponents of W_9; e.g., 6 represents W_9^6.

Continuing to use the periodic properties, the results for X(0) through X(8) are:

$$X(0) = F(0) + G(0) + H(0) \tag{3.20}$$

$$X(1) = F(1) + W_9^{1}\, G(1) + W_9^{2}\, H(1) \tag{3.21}$$

$$X(2) = F(2) + W_9^{2}\, G(2) + W_9^{4}\, H(2) \tag{3.22}$$

$$X(3) = F(0) + W_9^{3}\, G(0) + W_9^{6}\, H(0) \tag{3.23}$$

$$X(4) = F(1) + W_9^{4}\, G(1) + W_9^{8}\, H(1) \tag{3.24}$$

$$X(5) = F(2) + W_9^{5}\, G(2) + W_9^{10}\, H(2) \tag{3.25}$$

$$X(6) = F(0) + W_9^{6}\, G(0) + W_9^{12}\, H(0) \tag{3.26}$$

$$X(7) = F(1) + W_9^{7}\, G(1) + W_9^{14}\, H(1) \tag{3.27}$$

$$X(8) = F(2) + W_9^{8}\, G(2) + W_9^{16}\, H(2) \tag{3.28}$$

Eqs (3.20) through (3.28) conclude the first stage decimation of the 9-point sequence. The DFT computation has been reduced to computations of N/3-point DFTs where N/3 = 3. An example 3-point DFT for x(0), x(3), and x(6) is shown in Figure 3.10. The complete flowgraph for the computation of the 9-point DFT is shown in Figure 3.11 and was obtained by substituting the computation of Figure 3.10 into Figure 3.9.

Considering the more general case with N a power of 3 greater than two the same decimation procedure would be continued by decomposing the N/3 DFTs into N/9 computations of F(k), G(k), and H(k). The DFT of F(k) is:

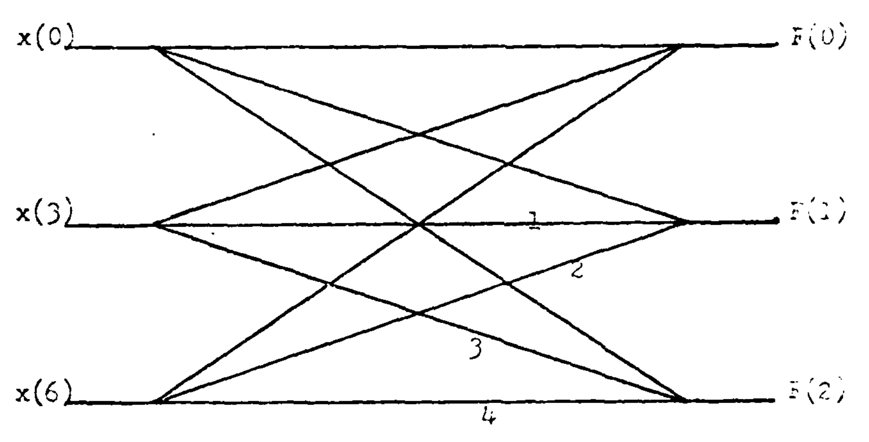


Figure 3.10. Butterfly Flowgraph for F(k).

NOTE: The numbers on branch transmittances refer to powers of W_N.

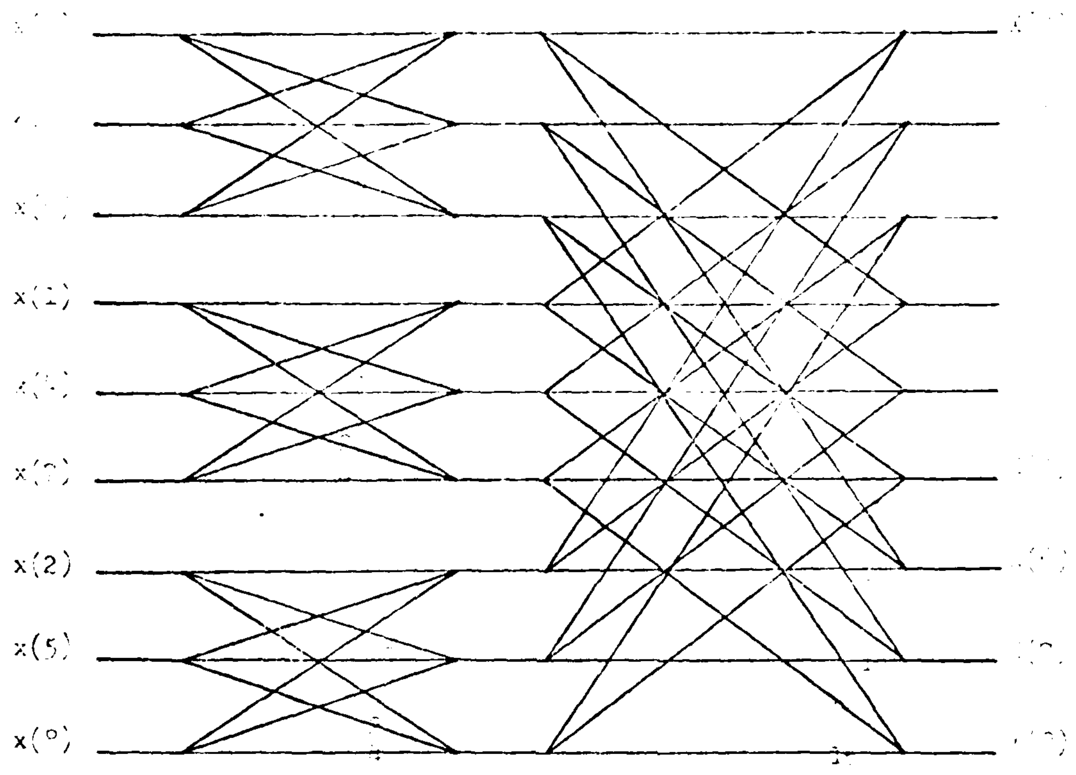


Figure 3.11. Complete Butterfly Flowgraph (N=9).

NOTE: Digits on the branch transmittance refer to powers of W_N.

$$F(k) = \sum_{r=0}^{(N/3)-1} x(r)\, W_3^{rk} \qquad (3.29)$$

This equation, letting $Q = (N/9)-1$, can be divided into three N/9 length sequences:

$$F(k) = \sum_{i=0}^{Q} f(3i) W_{N/3}^{3ik} + \sum_{i=0}^{Q} f(3i+1) W_{N/3}^{(3i+1)k}$$
$$+ \sum_{i=0}^{Q} f(3i+2) W_{N/3}^{(3i+2)k} \qquad (3.30)$$

Expanding the exponents of $W_{N/3}$ Eq (3.30) can be rewritten:

$$F(k) = \sum_{i=0}^{Q} f(3i) W_{N/3}^{3ik} + W_{N/3}^{k} \sum_{i=0}^{Q} f(3i+1) W_{N/3}^{3ik}$$
$$+ W_{N/3}^{2k} \sum_{i=0}^{Q} f(3i+2) W_{N/3}^{3ik} \qquad (3.31)$$

Using the substitution $W_{N/3}^{3} = W_{N/9}$,

$$F(k) = \sum_{i=0}^{Q} f(3i) W_{N/9}^{ik} + W_{N/3}^{k} \sum_{i=0}^{Q} f(3i+1) W_{N/9}^{ik}$$
$$+ W_{N/3}^{2k} \sum_{i=0}^{Q} f(3i+2) W_{N/9}^{ik} \qquad (3.32)$$

Similar expressions for G(m) and H(m) can be derived:

$$G(k) = \sum_{i=0}^{Q} g(3i) W_{N/9}^{ik} + W_{N/3}^{k} \sum_{i=0}^{Q} g(3i+1) W_{N/9}^{ik}$$
$$+ W_{N/3}^{2k} \sum_{i=0}^{Q} g(3i+2) W_{N/9}^{ik} \qquad (3.33)$$

$$H(k) = \sum_{i=0}^{Q} h(3i) W_{N/9}^{ik} + W_{N/3}^{k} \sum_{i=0}^{Q} h(3i+1) W_{N/9}^{ik}$$
$$+ W_{N/3}^{2k} \sum_{i=0}^{Q} h(3i+2) W_{N/9}^{ik} \qquad (3.34)$$

Eqs (3.32) through (3.34) can be used to derive the general expression for a radix-3 butterfly flowgraph. Letting N=9 the expressions for F(k), G(k) and H(k) become:

$$F(0) = f(0) + W_3^0 f(1) + W_3^0 f(2)$$

$$F(1) = f(0) + W_3^1 f(1) + W_3^2 f(2)$$

$$F(2) = f(0) + W_3^2 f(1) + W_3^4 f(2) \qquad (3.35)$$

$$G(0) = g(0) + W_3^0 g(1) + W_3^0 g(2)$$

$$G(1) = g(0) + W_3^1 g(1) + W_3^2 g(2)$$

$$G(3) = g(0) + W_3^2 g(1) + W_3^4 g(2) \qquad (3.36)$$

$$H(0) = h(0) + W_3^0 h(1) + W_3^0 g(2)$$

$$H(1) = h(0) + W_3^1 h(1) + W_3^2 g(2)$$

$$H(2) = h(0) + W_3^2 h(1) + W_3^4 g(2) \qquad (3.37)$$

From Eqs (3.35) through (3.37) the general butterfly multipliers are derived (consistent with Oppenheim and Schafer) to be:

$$X(k) = F(k) + W_N^{k} G(k) + W_N^{2k} H(k) \qquad (3.38)$$

$$X(k+r) = F(k) + W_N^{k+r} G(k) + W_N^{2k+2r} H(k) \qquad (3.39)$$

$$X(k+2r) = F(k) + W^{k+2r} G(k) + W_N^{2k+4r} H(k) \qquad (3.40)$$

where r represents the distance between the endpoints of the butterfly. In Figure 3.11 r=1 for stage 1 and r=2 for stage 2. Eqs (3.38) through (3.40) are represented in Figure 3.12 which is the general radix-3 butterfly flowgraph.

The exponents of Figure 3.12 can be rewritten to:

$$W^{k+r} = W^k \, W^r \tag{3.41}$$

$$W^{2k+2r} = W^{2k} \, W^{2r} \tag{3.42}$$

$$W^{k+2r} = W^k \, W^{2r} \tag{3.43}$$

$$W^{2k+4r} = W^{2k} \, W^{4r} \tag{3.44}$$

With these expressions for the butterfly multipliers an alternative arrangement to Figure 3.12 is possible by "premultiplying" or "twiddling" the inputs to G(k) and H(k) (Gentleman and Sande, 1966). The multipliers W_N^k and W_N^{2k} represent the twiddle factors of the butterfly in Figure 3.13. Since N=3r (Oppenheim and Schafer, 1975) the butterfly multipliers can be reduced to:

$$W_N^r = W_{3r}^r = \exp\,(-j2\pi r/3r) = \exp\,(-j2\pi/3) \tag{3.45}$$
$$= -0.5 - j.866$$

$$W_N^{2r} = W_{3r}^{2r} = \exp\,(-j4\pi/3) = -0.5 + j.866 \tag{3.46}$$

$$W_N^{4r} = W_{3r}^{4r} = \exp\,(-j8\pi/3) = -0.5 - j.866 \tag{3.47}$$

Oppenheim and Schafer observed that there is no advantage in Figure 3.12 to the alternate twiddle factor version in Figure 3.13 because "exp(-j2π/3) and all the powers thereof are complex coefficients that require multiplications". However, for the particular FORTRAN FFT radix-3 programs which implemented Figures 3.12 and 3.13, the twiddle factor

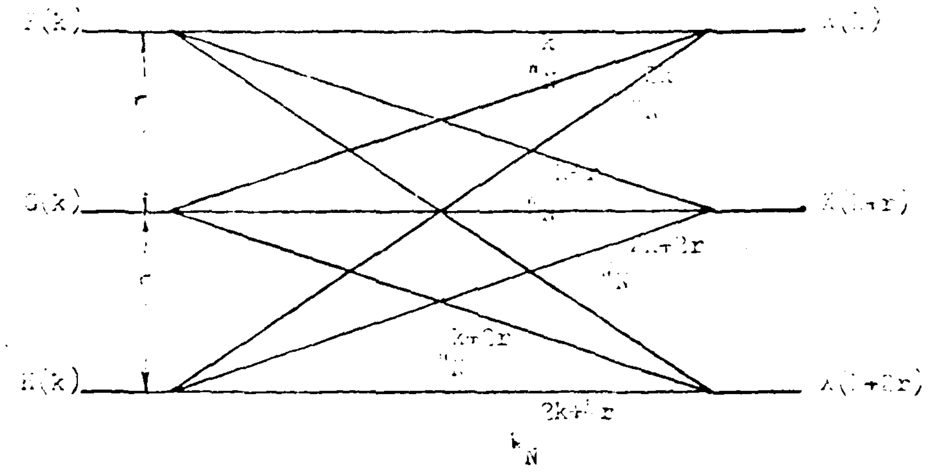


Figure 3.12. General Radix-3 Butterfly Flowgraph.

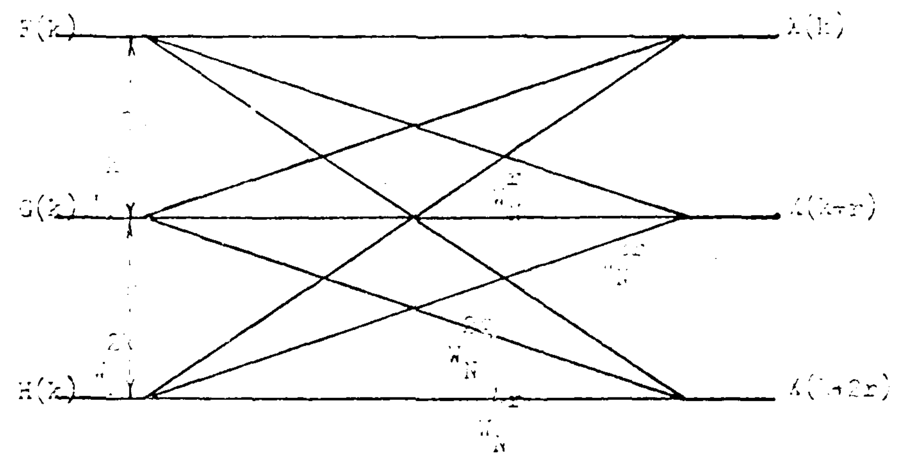


Figure 3.13. Basic Twiddle Factor Radix-3 Butterfly.

version of the radix-3 FFT was much more efficient to implement because only two twiddle factors had to be computed (W^k and W^{2k}) per butterfly and the butterfly multipliers were the constants in Eqs (3.45) and (3.46), the original version of Figure 3.12 requires that all six complex multipliers be computed for each butterfly. The twiddle factor version represents a simplification over the original radix-3 butterfly.

3.2.3 Radix-5 Theory. The theory for the radix-5 algorithm follows a development similar to the radix-3. Because of this similarity only the radix-5 results are given here for comparison to the radix-3, readers interested in detailed development are referred to Appendix D.

The basic butterfly multipliers for the radix-5 are given by:

$$X(k) = A(k) + W_N^{k} B(k) + W_N^{2k} C(k) + W_N^{3k} D(k) + W_N^{4k} E(k) \qquad (3.48)$$

$$X(k+r) = A(k) + W_N^{k+r} B(k) + W_N^{2k+2r} C(k) + W_N^{3k+3r} D(k) + W_N^{4k+4r} E(k) \qquad (3.49)$$

$$X(k+2r) = A(k) + W_N^{k+2r} B(k) + W_N^{2k+4r} C(k) + W_N^{3k+6r} D(k) + W_N^{4k+8r} E(k) \qquad (3.50)$$

$$X(k+3r) = A(k) + W_N^{k+3r} B(k) + W_N^{2k+6r} C(k) + W_N^{3k+9r} D(k) + W_N^{4k+12r} E(k) \qquad (3.51)$$

$$X(k+4r) = A(k) + W_N^{k+4r} B(k) + W_N^{2k+8r} C(k) + W_N^{3k+12r} D(k)$$
$$+ W_N^{4k+16r} E(k) \qquad (3.52)$$

The Eqs (3.48) through (3.52) are shown in the twiddle factor butterfly of Figure 3.14 where "r" is the distance between the butterfly and points. Since N=5r the butterfly multipliers reduce to constant complex multipliers of:

$$W_N^{r} = W_N^{6r} = W_N^{16r} = \cos(2\pi/5) -j \sin(2\pi/5)$$
$$W_N^{2r} = W_N^{12r} = \cos(4\pi/5) -j \sin(4\pi/5)$$
$$W_N^{3r} = (W_N^{2r})^* = W_N^{8r} = \cos(4\pi/5) +j \sin(4\pi/5)$$
$$W_N^{4r} = (W_N^{r})^* = W_N^{9r} = \cos(2\pi/5) +j \sin(2\pi/5)$$

These constant butterfly multipliers are computed once during the FFT computation and used in every radix-5 butterfly.

3.2.4 Digit Reversal Algorithm. In order for the DFT to be computed as discussed above, the input data must be stored in nonsequential order. In fact the order in which the input data are stored is in "bit-reversed" order for the radix-2 FFT and "digit-reversed" order for the other fixed-radix algorithms. To see what is meant by this terminology note that for the 8-point radix-2 flowgraph of Figure 3.8 three binary digits are required to index through the data array. Writing the input indices X_0 in binary form and then reversing the order of th[illegible]s gives:

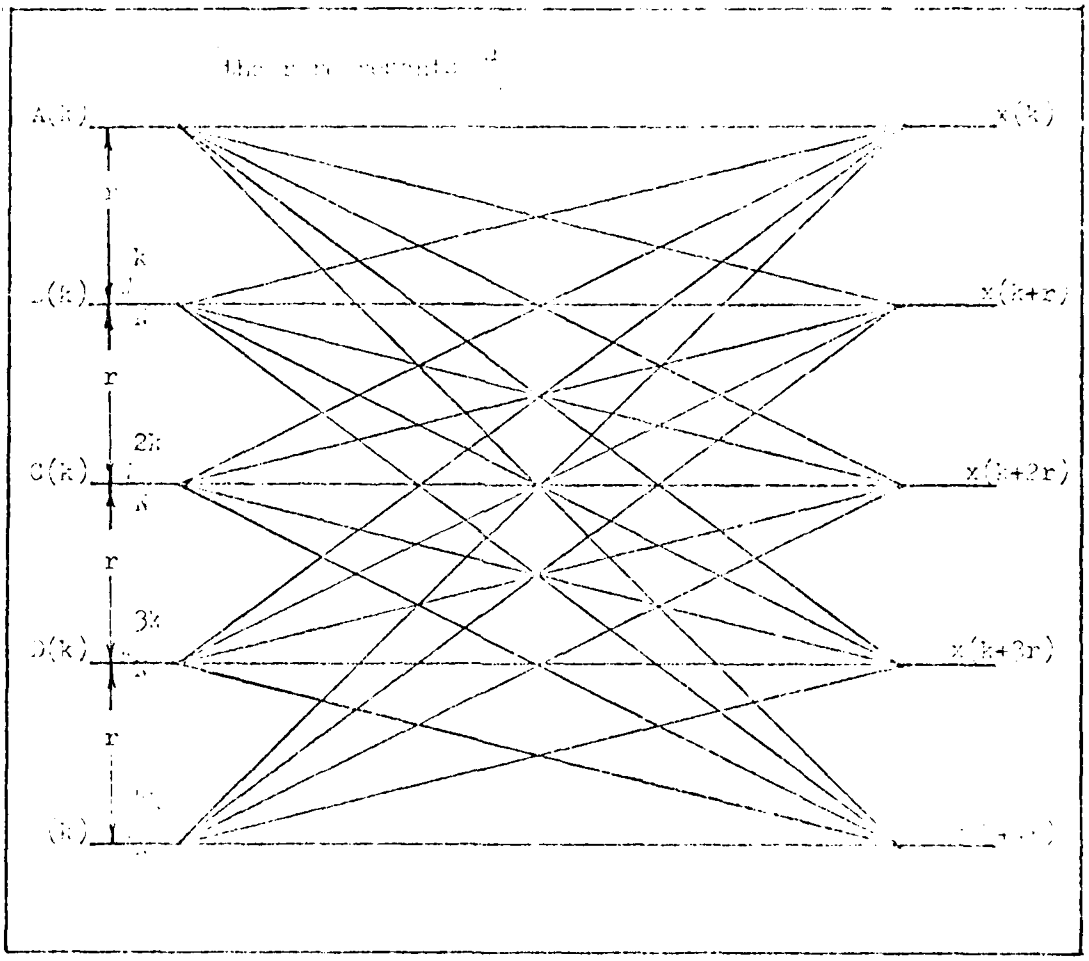


Figure 3.14. Radix-5 Twiddle Factor Butterfly.

$$X_0(0) = X_0(000) = x(000) = x(0)$$

$$X_0(1) = X_0(001) = x(100) = x(4)$$

$$X_0(2) = X_0(010) = x(010) = x(2)$$

$$X_0(3) = X_0(011) = x(110) = x(6) \qquad (3.53)$$

$$\vdots$$

$$X_0(7) = X_0(111) = x(111) = X(7)$$

If $(n_2\ n_1\ n_0)$ is the binary representation of the index of the sequence x(n), then sequence value $s(n_2\ n_1\ n_0)$ is stored in array position $x_0(n_0\ n_1\ n_2)$. That is, in determining the position of $x(n_2\ n_1\ n_0)$ in the input array, the bits of index n must be reversed in order.

For the radix-3 FFT the input array must be in a similar nonsequential order. The order is determined by "digit reversing" the input sequence value using a modulo-3 counter. The digit reversed radix-3 FFT example where N=9 is shown in Figure 3.15. The modulo-3 counter is given by:

$$\text{COUNT} = (b_1 \cdot 3^1) + (b_0 \cdot 3^0) \qquad (3.54)$$

where $b_k = 0, 1, 2$. The reversed count is given by:

$$\text{REVCOUNT} = (b_0 \cdot 3^1) + (b_1 \cdot 3^0) \qquad (3.55)$$

Eqs (3.54) and (3.55) show the modulo-3 counter for N=9 which requires only two b_k bits: b_1 and b_0 to represent the input sequence. For the case where $N=3^3=27$ three bits are needed to represent the input sequence x(n) and the modulo-3 counter becomes:

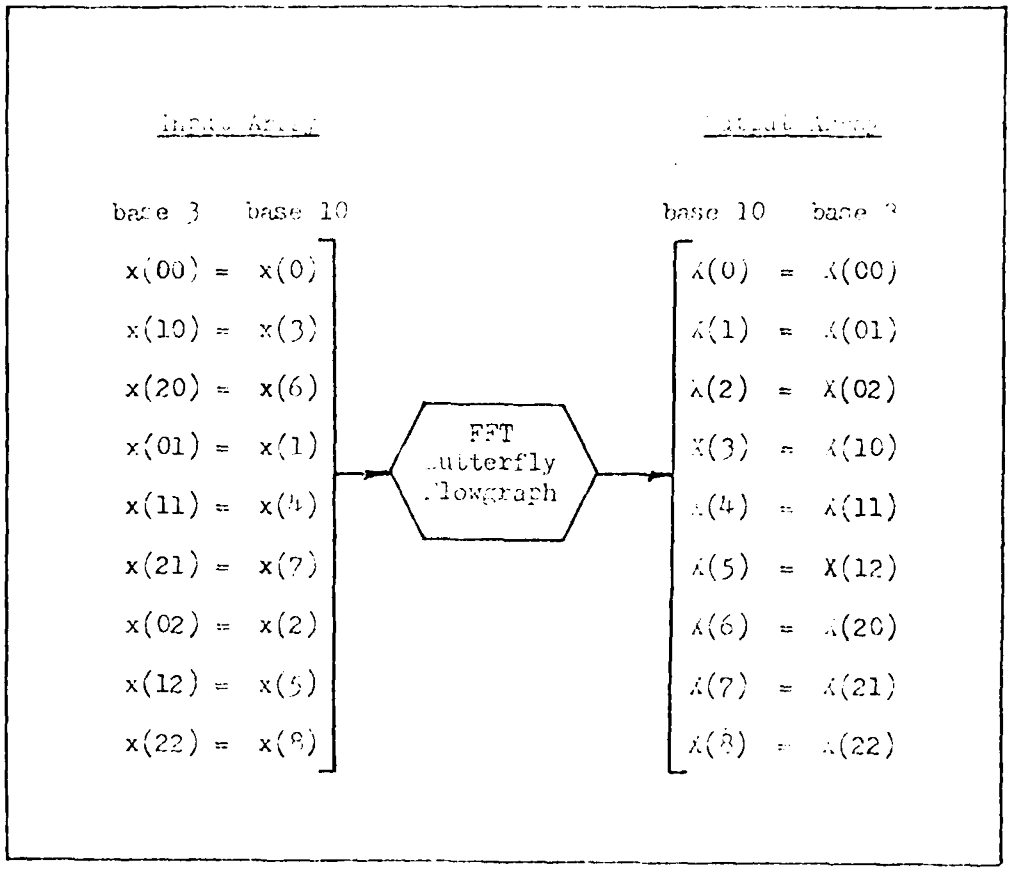


Figure 3.15. Digit Reversed Input and Output Arrays.

$$\text{COUNT} = (b_2 \cdot 3^2) + (b_1 \cdot 3^1) + (b_0 \cdot 3^0) \qquad (3.56)$$

and the reverse digit counter is:

$$\text{REVCOUNT} = (b_0 \cdot 3^2) + (b_1 \cdot 3^1) + (b_2 \cdot 3^0) \qquad (3.57)$$

Similarly the general expressions for COUNT and REVCOUNT can be given where $N=3^m$ and $b_k = 0, 1, 2$:

$$\text{COUNT} = (b_{m-1} \cdot 3^{m-1}) + (b_{m-2} \cdot 3^{m-2}) + \ldots$$
$$+ (b_1 \cdot 3^1) + (b_0 \cdot 3^0) \qquad (3.58)$$

and

$$\text{REVCOUNT} = (b_1 \cdot 3^{m-1}) + (b_2 \cdot 3^{m-2}) + \ldots$$
$$+ (b_{m-2} \cdot 3^1) + (b_{m-1} \cdot 3^0) \qquad (3.59)$$

Once COUNT and REVCOUNT are computed the magnitudes are compared. If REVCOUNT is less than or equal to COUNT a swap of the values indexed by COUNT and REVCOUNT is not required; otherwise exchange the array value indexed in by COUNT with the array value indexed by REVCOUNT. The cojnters are incremented by one and the process continued until all N indices have been tested.

3.2.5 <u>Development of a Radix-3 FFT Based on the Cube Root of Unity</u>. This section presents the theory of a radix-3 FFT algorithm which uses the complex cube root of unity to perform the complex Fourier transformation (butterfly) without using multiplications. The benefit of this technique will also be discussed in the section on real operations count.

While the reference (Dubois and Venetsanopoulos, [illegible]) contains a complete description of this technique, it leaves out several steps which aid in understanding the theory and for that reason it is presented again here.

This algorithm uses basis vectors (1,u) instead of the conventional complex plane vectors (1,j) to perform the complex Fourier transform (where u is the cube root of 1 and j is the square root of -1). The new basis vectors use arithmetic notation:

$$a + bu = R(u) \;;\; a,\ b,\ \text{real numbers} \tag{3.60}$$

Taking u as the cube root of 1 implies:

$$u^3 - 1 = 0 \tag{3.61}$$

or

$$(u-1)(u^2 + u + 1) = 0 \tag{3.62}$$

Since it is known $u \neq 1$, then

$$u^2 + u + 1 = 0 \tag{3.63}$$

or

$$u^2 = -1 - u \tag{3.64}$$

Eq (3.60) is used in the definition of multiplication in the R(u) field:

$$(a + bu)(c + du) = ac + bdu^2 + adu + bcu \tag{3.65}$$

Substituting Eq (3.64) into Eq (3.65) results in:

$$(a + bu)(c + du) = (ac - bd) + (ad + b(c-d))u \tag{3.66}$$

The expression in Eq (3.66) can be expanded and then recombined to reduce the number of multiplications:

$$ad + b(c-d) = ad + bc - bd - bd + bd + ac - ac \tag{3.67}$$

$$= ac + ad + bc + bd - ac - bd - bd \tag{3.68}$$

$$= (a + b)(c + d) - ac - bd - bd \tag{3.69}$$

Substituting Eq (3.69) into Eq (3.66) gives:

$$(a + bu)(c + du) = (ac - bd) \tag{3.70}$$

$$+ ((a + b)(c + b) - ac - bd - bd))u$$

The result in Eq (3.70) requires three real multiplications and six real additions compared with conventional complex multiplication which requires four real multiplications and two real additions. Multiplication in the R(u) field requires one less multiplication but four more additions.

The expression for u^3 is obtained from $u^3 = 1$ by letting $u^3 = (\exp(-j2\pi/3))^3 = 1$. Consequently, $u = \exp(-j2\pi/3) = -1/2 -j(\sqrt{3}/2)$ which is used for conversion between $a + bj$ and $c + du$:

$$c + du = c + d(-1/2-j(\sqrt{3}/2)) = c - d/2-j(\sqrt{3}/2)d \tag{3.71}$$

$$c + du = (c - d/2) + j(-\sqrt{3}/2)d \tag{3.72}$$

To find the conversion from $a + bj$ to $c + du$, solve Eq (3.70) for j:

$$c + du = (c - d/2) + (-\sqrt{3}d/2)\,j$$

$$d/2 + du = (-\sqrt{3}/2)d\ j$$

$$d(1/2 + u) = (-\sqrt{3}/2)d\ j$$

$$1/2 + u = (-\sqrt{3}/2)j$$

$$j = (-2/\sqrt{3})(1/2 + u) \tag{3.73}$$

Using Eq (3.66) and a + bj the conversion to c + du is:

$$a + bj = a + b(-2/\sqrt{3})(1/2 + u)$$

$$= a + b(-2/\sqrt{3})(1/2) + b(-2/\sqrt{3})u$$

$$a + bj = (a - b/\sqrt{3}) + (-2b/\sqrt{3})u \qquad (3.74)$$

Using the R(u) arithmetic developed above, it can be shown that a radix-3 FFT butterfly can be developed which requires no multiplications except for the twiddle factors in Figure 3.13.

Using Eq (3.74) and $W_N^r = \cos(2\pi r/N) + j(-\sin(2\pi r/N))$ produces:

$$c + du = (\cos(2\pi r/N) + \sin(2\pi r/N)/\sqrt{3})$$

$$+ (2 \sin(2\pi r/N)/\sqrt{3})u \qquad (3.75)$$

Using the substitution of N = 3r in Eq (3.75) reduces it to:

$$W_N^r = (\cos(2\pi/3) + \sin(2\pi/3)\sqrt{3}) + 2 \sin(2\pi/3)\sqrt{3})u$$

$$W_N^r = 0 + 1u = u \qquad (3.76)$$

Likewise the remaining W terms in Figure 3.7 can be reduced:

$$W_N^{2r} = (\cos(4\pi/3) + \sin(4\pi/3)/\sqrt{3}) + 2 \sin(4\pi/3)/\sqrt{3})u$$

$$W_N^{2r} = -1 - 1\ u \qquad (3.77)$$

$$W_N^{4r} = 0 + 1\ u = u \qquad (3.78)$$

Substituting Eqs (3.76) through (3.78) into Figure 3.13 produces Figure 3.16.

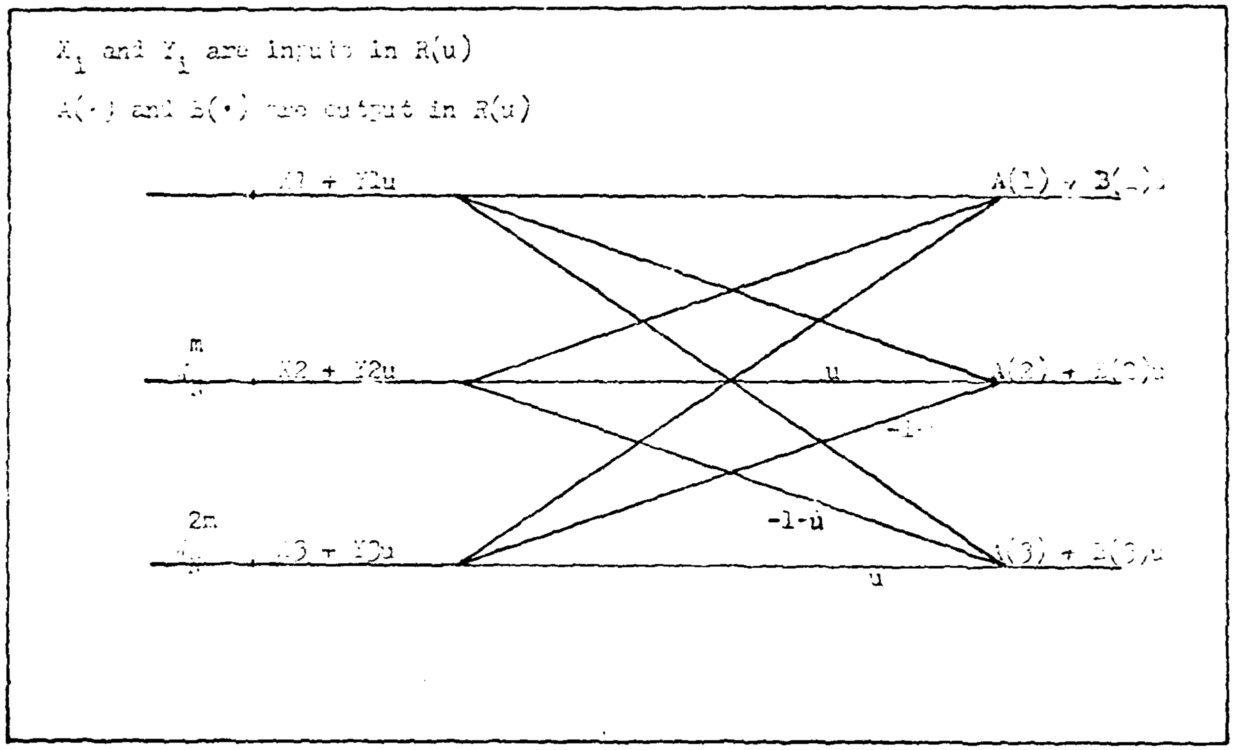


Figure 3.16. Radix-3 Butterfly in R(u) Arithmetic.

Using arithmetic in R(u) and carrying out the operations in Figure 3.1[illegible] shows that only fourteen real adds and no multiplies are required to evaluate the butterfly flowgraph. X_i, Y_i are the butterfly inputs after twiddle factor multiplication and A(·), B(·) are the butterfly outputs in the R(u) field.

$$A(1) + B(1)u = (X1 + X2 + X3) + (Y1 + Y2 + Y3)u \qquad (3.79)$$

$$\begin{aligned} A(2) + B(2)u &= (X2 + Y2u)(0 + u) + (X3 + Y3u)(-1 - u) \\ &\quad + (X1 + Y1u) \end{aligned}$$

$$\begin{aligned} A(2) + B(2)u &= (-Y2) + (X2 + Y2\ (-1))u + (-X3 + Y3) \\ &\quad + (-X3)u - X1 + Y1u \qquad (3.80) \\ &= (X1 - Y2 - X3 + Y3) + (Y1 + X2 - Y2 - X3)u \end{aligned}$$

$$\begin{aligned} A(3) + B(3)u &= X1 + Y1u + (X2 + Y2u)(-1 - u) \\ &\qquad + (X3 + Y3u)(0 + u) \\ &= X1 + Y1u + (-X2 + Y2) + (-X2)u + (-Y3) \\ &\quad + (X3 + Y3(\bar{Q}1))u \qquad (3.81) \\ &= (X1 - X2 + Y2 - Y3) + (Y1 - X2 + X3 - Y3)u \end{aligned}$$

There are 16 real additions shown in Eqs (3.80) and (3.81); however, by combining common terms -Y2 - X3 = -R and -X2 - Y3 = -S, the radix-3 butterfly can be evaluated using only fourteen real additions (neglecting the twiddle factors):

$$A(1) = X1 + X2 + X3$$

$$B(1) = Y1 + Y2 + Y3$$

$$A(2) = X1 + Y3 - R$$

$$B(2) = Y1 + X2 - R \qquad \text{where } R = Y2 + X3$$

$A(3) = X1 + Y2 - S$

$B(3) = Y1 + X3 - S$ where $S = X2 + Y3$

3.2.6 Summary. This completes the discussion of fixed radix FFT theory. In this section the general theory was developed using the radix-3 case as an alternative to the more common radix-2 development. A decimation-in-time for N=9 was shown and the basic butterfly equations for radix-3 was derived. Because of the similarity to radix-3 butterflies, the radix-5 theory was not developed but the butterfly equations necessary to implement a radix-5 FFT was given. Finally, a new radix-3 FFT (Dubois and Venetsanopoulos, 1978) was developed.

3.3 Real Operations Count for Fixed Radix FFTs

The speed at which an FFT algorithm can perform the DFT is a (to a first approximation) proportional to the number of complex multiplications used in the algorithm (Singleton, 1969). The number of times the data array is indexed is a secondary factor and is shown to have minimal impact on the results of this paper.

An anomaly in the nomenclature should be pointed out before further discussion of "complex multiplications" related to FFTs. A complex multiplication implies four real multiplications and two real additions. It has been shown (Singleton, 1969) that $(p-1)^2$ real multiplications are required to evaluate a complex transform of dimension p, p odd, where $N=p^m$. Singleton then refers to the $(p-1)^2$

real multiplications as $(p-1)^2$ complex multiplications which is a notational convenience since a complex transform of dimension p requires more than $(p-1)^2/2$ real additions. Throughout this paper all references to multiplications and additions are in terms of real operations and not complex operations.

The real operations are determined from (1) the number of butterflies times the number of real operations required to compute the butterfly and (2) the number of twiddle factors times real operations required per twiddle factor, and (3) the number of trigonometric functions (sine and cosine) which must be computed. The real operations count for a radix-p FFTs are derived as a function of N, m, and p where $N=p^m$.

3.3.1 Number of Butterflies in Fixed Radix-p FFTs. The number of butterflies is dependent on N, m, and p, where $N=p^m$. Examining the radix-2 FFT in Figure 3-8 shows that there are 8 input points and 8 output points for each stage. The radix-2 butterfly in Figure 3.7 has 2 input and 2 output points which means that Figure 3.8 must have 8/2 = 4 butterflies per stage. There are 3 stages in this radix-2 FFT (where $N=2^3$) giving a total of 12 butterflies in this FFT.

In general the number of radix-p butterflies is given by:

$$mn/p \tag{3.82}$$

This equation can be checked for the radix-3 example. Given that N=9, p=3, and m=2 Eq (3.82) gives the total number of butterflies as $2 \cdot 9/3 = 6$. This is verified by Figure 3.11 which has 6 radix-3 butterflies.

3.3.2 Number of Twiddle Factors in Fixed Radix-p FFTs. The twiddle factors are complex multipliers of the form $\exp(-j2\pi rk/N)$ which multiply each radix-p butterfly as shown in Figure 3.8. Notice that each stage has $N/p = 8/2 = 4$ butterflies, each of which requires $p-1 = 2-1 = 1$ complex twiddle factor. The general expression for number of twiddle factors in each stage becomes:

$$N(p-1)/p \tag{3.84}$$

Given that $N=p^m$ there are m stages in a radix-p FFT making the total number of twiddle factors for the FFT equal:

$$mN(p-1)/p \tag{3.85}$$

Some of the complex twiddle factors are $W_N^0 = 1$ and can be eliminated. In any FFT there are N-1 of these unity twiddle factors (Singleton, 1969) which gives the final expression for the number of complex twiddle factors as:

$$mN(p-1)/p - (N-1) \tag{3.86}$$

Using $N = p^m = 2^3 = 8$ in Eq (3.86) the number of twiddle factors is found to be 5. Examining Figure 3.8 for $N=2^3$ shows there are 5 non-unity twiddle factors.

3.3.3 Number of Trigonometric Functions Required for the Fixed Radix Algorithms. The trigonometric functions of sine and cosine are needed to compute the twiddle factors. The fixed radix-2 algorithm uses calls to the FORTRAN library SIN and COS functions as well as the difference equations given in Section 3.1. The radix-3 and 5 FFTs use only sine and cosine difference equations.

The radix-2 algorithm in Appendix A computes one sine and cosine at each stage of the FFT using:

```
W = CMPLX (COS(PI/LE1), SIN(PI/LE1))
```

Each radix-2 FFT has m stages where $N=2^m$ which means the sine and cosine functions are called m times for the FFT. Once the initial sine and cosine are computed for the stage each new twiddle factor in the stage is computed using the complex multiplication:

$$U = U * W$$

where the complex U was originally initialized to U = (1,0). The complex multiplication U * W effectively implements the sine and cosine difference equations in Section 3.1. The number of times U * W is computed for each FFT stage is a function of the number of different twiddle factors in the stage m_i. In Figure 3.8 the first stage has only one type of twiddle factor W_N^0, the second stage has two types: W_N^0 and W_N^2, while stage has four: W_N^0, W_N^1, W_N^2, W_N^3. The general expression for the types of twiddle factors in each stage is:

$$TF = 2^{k-1}$$

Thus for stage 1, k=1 and $TF=2^0=1$, which gives one type of twiddle factor; for stage 2, k=2 and $TF=2^1=2$ giving two types of twiddle factors; and finally for the last stage in this example k=3 and $TF=2^2=4$, or four types of twiddle factors are required. In general for the radix-2 FFT in

Appendix A the complex multiplication U * W is evaluated a total of

$$\sum_{k=1}^{m} (2^{k-1})$$

times, where m is the number of stages for $N=2^m$. Given that the complex multiplications requires 4 real multiplications and 2 additions, the number of operations required to compute sines and cosines for this radix-2 FFT is:

$$\text{real mult} = 4 \sum_{k=1}^{m} (2^{k-1}) \quad (3.87)$$

$$\text{real add} = 2 \sum_{k=1}^{m} (2^{k-1}) \quad (3.88)$$

$$\text{sine and cosine calls} = m \quad (3.89)$$

.ne real operations required to compute the sine and cosine lookup tables for the radix-3 and 5 algorithms is less complex than the radix-2 FFT. In these algorithms the difference equation from Section 3.1 is used to compute sine and cosine lookup tables which have length N. Because of the symmetry of sin(k) = -sin(-k) only N/2 computations of the difference equations are required. The equations are given by:

WKC(I) = C * WKC(I-1) - S * WKS(I-1) + WKC(I-1)

WKS(I) = C * WKS(I-1) + S * WKC(I-1) + WKS(I-1)

which need a total of 4 real multiplications and 10 additions to compute. For an N length sequence computing the lookup tables require:

$$\text{real mult} = 4(N/2) = 2N \qquad (3.90)$$

$$\text{real add} = 10(N/2) = 5N$$

3.3.4 Number of Real Operations in Radix-p FFTs.
Based on the general expressions in Eqs (3.82) through (3.91) the total number of real multiplications can be determined given $N=p^m$ where N, p, and m are integers. First, each radix-p butterfly computation requires multiplications or additions or both to be evaluated. The exact number of multiplies and adds is determined from the FORTRAN code as shown below. Second, each complex twiddle factor multiplication requires 4 real multiplications and 2 real additions. Third, the number of real operations to compute the sines and cosines is added to the butterflies and twiddle factors to give the total operations count for each algorithm.

For the case of $N=2^m$ it was shown in the radix-2 Section 3.2.1 that the radix-2 butterfly can be computed with 4 real additions and no multiplications. This radix-2 butterfly can be computed with 4 real additions and no multiplications. This radix-2 FFT does not eliminate all multiplications by W_N^0. Therefore each radix-2 butterfly is multiplied by a complex twiddle factor as shown in Figure 3.8. For this particular radix-2 FFT the number of twiddle factors equal the number of butterflies. Combining all sources of real operations for the radix-2 FFT gives a total of:

$$\begin{aligned}\text{real mult} &= (\text{\# mult per butterfly}) * (\text{\# butterflies}) \\ &\quad + 4\ (\text{\# twiddle factors}) \\ &\quad + 4\ (\text{\# types of twiddle factors})\end{aligned} \tag{3.92}$$

Substituting the appropriate values for the radix-2 gives:

$$\begin{aligned}\text{real mult} &= (0) * (mN/2) + 4*(mN/2) + 4 * \left(\sum_{k=1}^{m} 2^{k-1}\right) \\ &= 2mN + 4 \sum_{k=1}^{m} 2^{k-1}\end{aligned} \tag{3.93}$$

Likewise for the number of real additions:

$$\begin{aligned}\text{real adds} &= (\text{\# adds per butterfly}) * (\text{\# butterflies}) \\ &\quad + 2\ (\text{\# twiddle factors}) \\ &\quad + 2\ (\text{\# types of twiddle factors})\end{aligned} \tag{3.94}$$

$$\begin{aligned}\text{real adds} &= 4 * (mN/2) + 2*(mN/2) + 2 * \left(\sum_{k=1}^{m} 2^{k-1}\right) \\ &= 3mN + 2 \sum_{k=1}^{m} 2^{k-1}\end{aligned} \tag{3.95}$$

For the radix-p FFTs where p is an odd prime it has been shown by Singleton, 1969, that these butterflies can be evaluated using $(p-1)^2$ real multiplications. The FORTRAN coded radix-3 and radix-5 in Appendices B and D require 4 real multiplications and 12 additions for radix-3 butterflies and 16 real multiplications and 30 additions for radix-2 butterflies. Using these in Eqs (3.87) and (3.91) yields the total real operations for the radix-3 as:

$$\begin{aligned}\text{real mult} &= (4 \text{ mult per butterfly}) * mN/3 \\ &\quad + 4(mN(3-1)/3 - (N-1)) + 2N \\ &= 4mN/3 + 8mN/3 - 4(N-1) + 2N \\ &= 4mN - 4(N-1) + 2N \qquad (3.96)\end{aligned}$$

$$\begin{aligned}\text{real adds} &= (12 \text{ adds per butterfly}) * mN/3 \\ &\quad + 2(mN(3-1)/3 - (N-1)) + 5N \\ &= 12mN/3 + 4mN/3 - 2(N-1) + 5N \\ &= 16mN/3 - 2(N-1) + 5N \qquad (3.97)\end{aligned}$$

Similarly the real operations count for the radix-5 FFT becomes:

$$\begin{aligned}\text{real mult} &= (16 \text{ mult per butterfly}) * mN/5 \\ &\quad + 4(mN(5-1)/5 - (N-1)) + 2N \\ &= 16mN/5 + 16mN/5 - 4(N-1) + 2N \\ &= 32mN/5 - 4(N-1) + 2N \qquad (3.98)\end{aligned}$$

$$\begin{aligned}\text{real adds} &= (30 \text{ adds per butterfly}) * mN/5 \\ &\quad + 2(mN(5-1)/5 - (N-1)) + 5N \\ &= 30mN/5 + 8mN/5 - 2(N-1) + 5N \\ &= 38mN/5 - 2(N-1) + 5N \qquad (3.99)\end{aligned}$$

The results of Eqs (3.92) through (3.99) are given in Table 3.1 for N between 8 and 16,000. This table also summarizes the possible values of N for the fixed radix-2, 3, and 5 FFTs.

3.3.5 Real Operations Count for the Radix-3 FFT Using the Complex Cube Root of Unity. This algorithm represents an alternative to the conventional radix-3 FFT. It is shown in this section that selective use of this

TABLE 3.1

REAL OPERATIONS COUNT FOR RADIX-2, 3 AND 5

N	Radix	Multiplications	Additions	Trig Library
8	2^3	76	86	3
9	3^2	58	125	1
16	2^4	188	222	4
25	5^2	274	457	1
27	3^3	274	515	1
32	2^5	444	542	5
64	2^6	1020	1278	6
81	3^4	1138	1973	1
125	5^3	2154	3227	1
128	2^7	2300	2942	7
243	3^5	4378	7211	1
256	2^8	5116	6654	8
512	2^9	11260	14846	9
625	5^4	14754	20877	1
729	3^6	15142	25517	1
1024	2^{10}	24572	32766	10
2048	2^{11}	53244	71678	11
2187	3^7	56866	88211	1
3125	5^5	93754	128127	1
4096	2^{12}	1146[illegible]	155646	12
6561	3^8	196834	299621	1
8192	2^{13}	245756	335870	13
15625	5^6	568754	759377	1

algorithm can reduce the number of real operations depending on the sequence length N.

The radix-3 FFT in the R(u) field has four sources or real multiplications (where $N=3^m$):

1. $2mn/3 - (N-1)$ complex twiddle factors derived in Section 3.3.3.

2. Conversion from complex to R(u) of $\sum_{i=2}^{m} 2(3^{i-1} - 1)$ twiddle factors derived from FORTRAN code in Appendix C.

3. Conversion of complex array of length N to the R(u) field derived from the FORTRAN code.

4. Conversion of R(u) array length N back to the complex field derived from the FORTRAN code.

The radix-3 in R(u) has five sources of real additions:

1. mn/3 butterflies derived in Section 3.3.3.

2. The four sources of real multiplies listed above.

Based on the FORTRAN code in Appendix C, there are three real multiplications per complex twiddle factor, two per twiddle factor conversion, two per conversion from complex to the R(u) field, and two per conversion from R(u) to the complex field. Condensing the above into an equation for real multiplications yields:

$$\text{real mult} = 3(2mN/3 - N+1) + 2 \sum_{i=2}^{m} 2(3^{i-1} - 1) + 4N \qquad (3.100)$$

There are 14 real additions per butterfly, six per twiddle factor, one per twiddle factor conversion, one per conversion to R(u) array, and one per conversion to complex array. Expressing the total number of real additions as a function of the above yields:

$$\text{real adds} = 14\,mN/3 + 6(2mN/3 - N+1) + \sum_{i=2}^{m} 2(3^{i-1} - 1) + 2N \qquad (3.101)$$

The results for the number of real multiplications and additions for both radix-3 algorithms is given in Table 3.2 for N=27 to N=19683. Because the R(u) radix-3 requires more multiplications and additions for N=27 and 81 it will always run slower than the complex field radix-3 FFT. But, for N=243 and higher the R(u) radix-3 may run faster depending upon the speed of additions relative to multiplications for the computer being used to perform the FFTs.

Table 3.2 also gives the "Add to Multiply Ratio" required for the R(u) field radix-3 FFT to run faster than the conventional radix-3 FFT. (The ratio is the difference in the number of multiplies divided by the difference in the number of additions.) For the case of N=729, a multiply operation must take 3.77 times longer than an addition before the R(u) field radix-3 can run faster than the complex field radix-3. This means that prior to selecting either of the algorithms the relative costs of additions to multiplications must be known as well as the length of the data sequence.

3.3.6 Memory Requirements for Fixed Radix FFTs. A major consideration for selecting a particular FFT algorithm is the sequence length and memory required to execute the subroutine relative to the memory available in the computer. For this reason the memory requirements

TABLE 3.2

COMPARISON BETWEEN COMPLEX AND R(u)
RADIX-3 FFT FOR REAL OPERATIONS*

	Complex Radix-3		R(u) Radix-3		Add to
N	Real Mult	Real Adds	Real Mult	Real Adds	Mult Ratio
27	220	380	232	624	NA
81	976	1568	1284	2562	NA
243	3892	5996	3140	9796	5.05
729	14584	21872	10912	35714	3.77
2187	52492	77276	37152	126108	3.18
6561	183712	266816	124628	435202	2.85
19683	629360	905420	413308	1476212	2.63

* Does not include computing sine and cosine terms

for the radix-2, 3, and 5 FFTs is given here as a function of sequence length N. The program memory and data array storage requirements for each algorithm are enumerated below.

The program memor, required by each routine was determined from a "load map" generated by the command MAP, PART. The array storage requirements were determined by inspection of the DIMENSION statements in the FORTRAN code for each subroutine listed in Appendix A to D. The results are:

FFT	Program	Arrays
Radix-2	108	2N
Radix-3	301	4N + M + 30
Radix-3 in R(u)	396	4N + M + 30
Radix-5	458	4N + M + 30

The memory arrays required for each algorithm as a function of N are listed in Table 3.3. The program memory was not included because it is dependent on machine word size which varies from machine to machine.

3.4 Mixed Radix FFT Algorithms

Up to this point only fixed radix FFTs ha [illegible]een discussed. Explanation and programming f[illegible] the s[illegible]ecial cases where $N=2^m$ or 3^m or 5^m are simpler than the general case of $N=p_1p_2\ldots p_m$, and for most applications the restricted choice of values is adequate. However, when the application does not permit "zeropacking" of the data sequence to reach one of the special cases a wider choice of N is needed.

TABLE 3.3

FIXED RADIX MEMORY REQUIRED

N	Memory Array
8	16
9	68
16	32
25	132
27	141
32	64
64	128
81	358
125	533
128	256
243	1007
256	512
512	1024
625	2534
729	2952
1024	2048
2048	4096
2187	8820

Singleton first published a mixed radix FFT algorithm in June 1969 which has been widely used and implemented on large and small computers. This algorithm is listed in Appendix F. (The International Mathematical Scientific Library (IMSL) which is available on the WPAFB CDC Cyber 74 computer has a mixed radix FFT based on Singleton's work). Also the author has written and tested a mixed radix algorithm which is listed in Appendix E. The theory, digit reversal, real operations count, and memory requirements for these algorithms is discussed in the following sections.

3.4.1 Mixed Radix Theory. All FFT theory can be developed by representing a one-dimensional sequence N as several two dimensional matrices and performing operations on these matrices. Understanding this approach when exposed to it for the first time is difficult. For this reason the matrix development is presented here and then a specific example of N=30 is treated to increase understanding of the technique.

The complex Fourier transform is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-j2\pi nk/N) \tag{3.102}$$

For k = 0, 1, ..., N-1 where X(k) and x(n) are both complex valued. Eq (3.102) can be expressed as a matrix multiplication: $X = Tx$

The matrix T can be decimated-in-time (Cooley and Tukey, 1965) or frequency (Gentleman and Sande, 1966) to produce equally efficient factoring:

$$T = P\,F_r\,F_{r-1}\,\ldots\,F_2\,F_1$$

where F_i is the decimation corresponding to the factor n_i of: $N = n_r\,n_{r-1}\,\ldots\,n_2\,n_1$

and P is the permutation (digit reversal) matrix (Singleton, 1969). The matrix F_i has only n_i nonzero elements on each row and column and also be partitioned into N/n_i square submatrices of dimension n_i; it is this partition that is the basis for these (mixed radix) algorithms" (Singleton, 1969). The matrices F_i can be further factored into:

$$F_i = R_i\,T_i \tag{3.103}$$

where R_i is the diagonal matrix of twiddle (rotation) factors. Using these twiddle factors enable the trigonometric symmetries and complex multipliers (e.g., $e^{j\pi}$, $e^{j\pi/2}$, ..., $e^{j\pi/N}$) to be exploited in the FFT butterflies and reduce the number of real operations. A specific decimation-in-time example is now considered which uses the above ideas.

Given an N-point sequence for which the N-point DFT is desired, the integer N can be factored into a product of smaller integers assuming N is not prime. The successive factorization of one number into two can result in any possible combination. If N=30, it can be factored as $5 \cdot 6$ and then as $5 \cdot 3 \cdot 2$. The first decomposition is shown in Figure 3.17 and is presented as six 5-point DFTs followed by five 6-point DFTs. The next stage of decomposition is from $5 \cdot 6$ to $5 \cdot 3 \cdot 2$ and is shown in Figure 3.18. Starting with the DFT expression in Eq (3.99) the sequence can

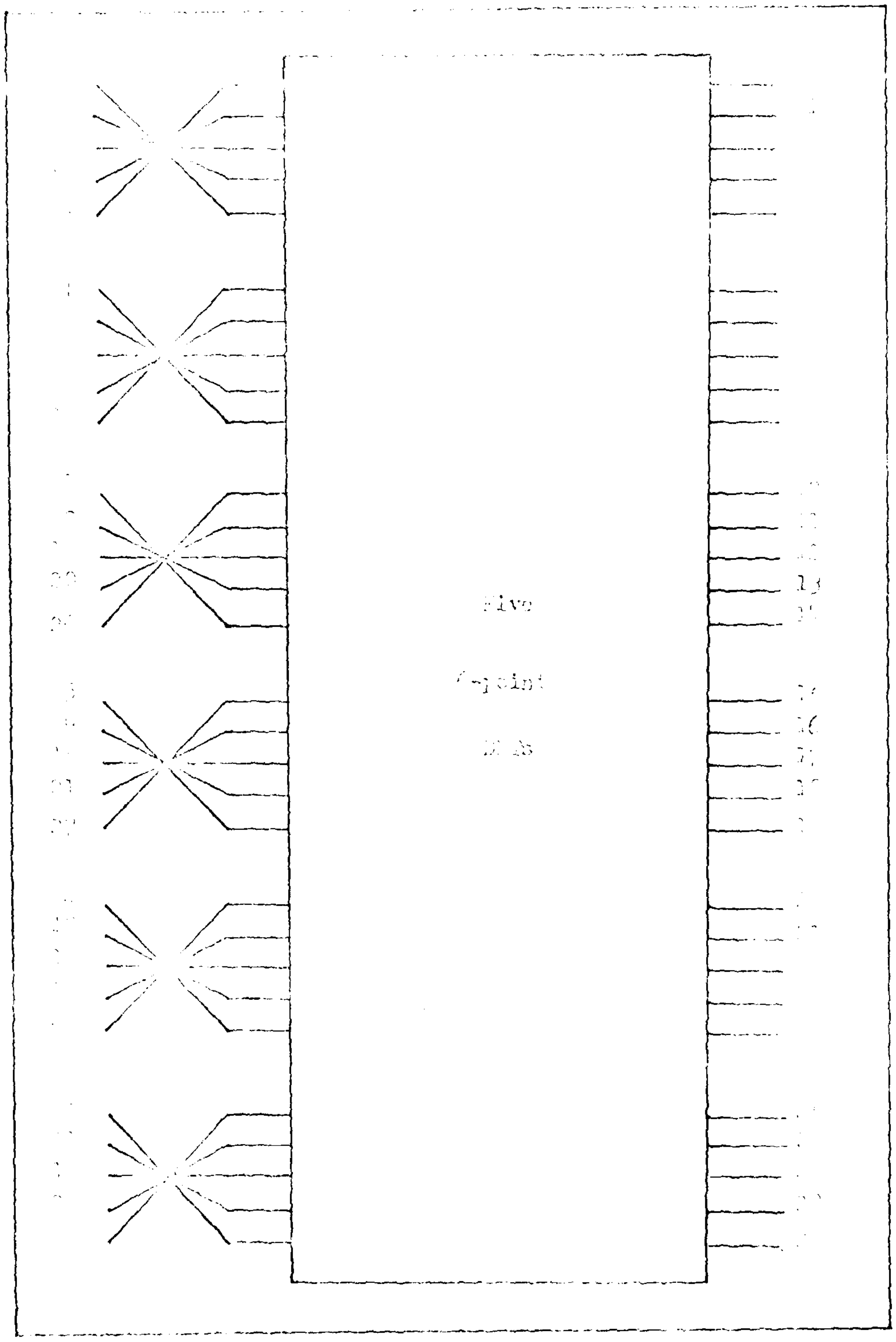


Figure 3.17. First Decomposition N=30.

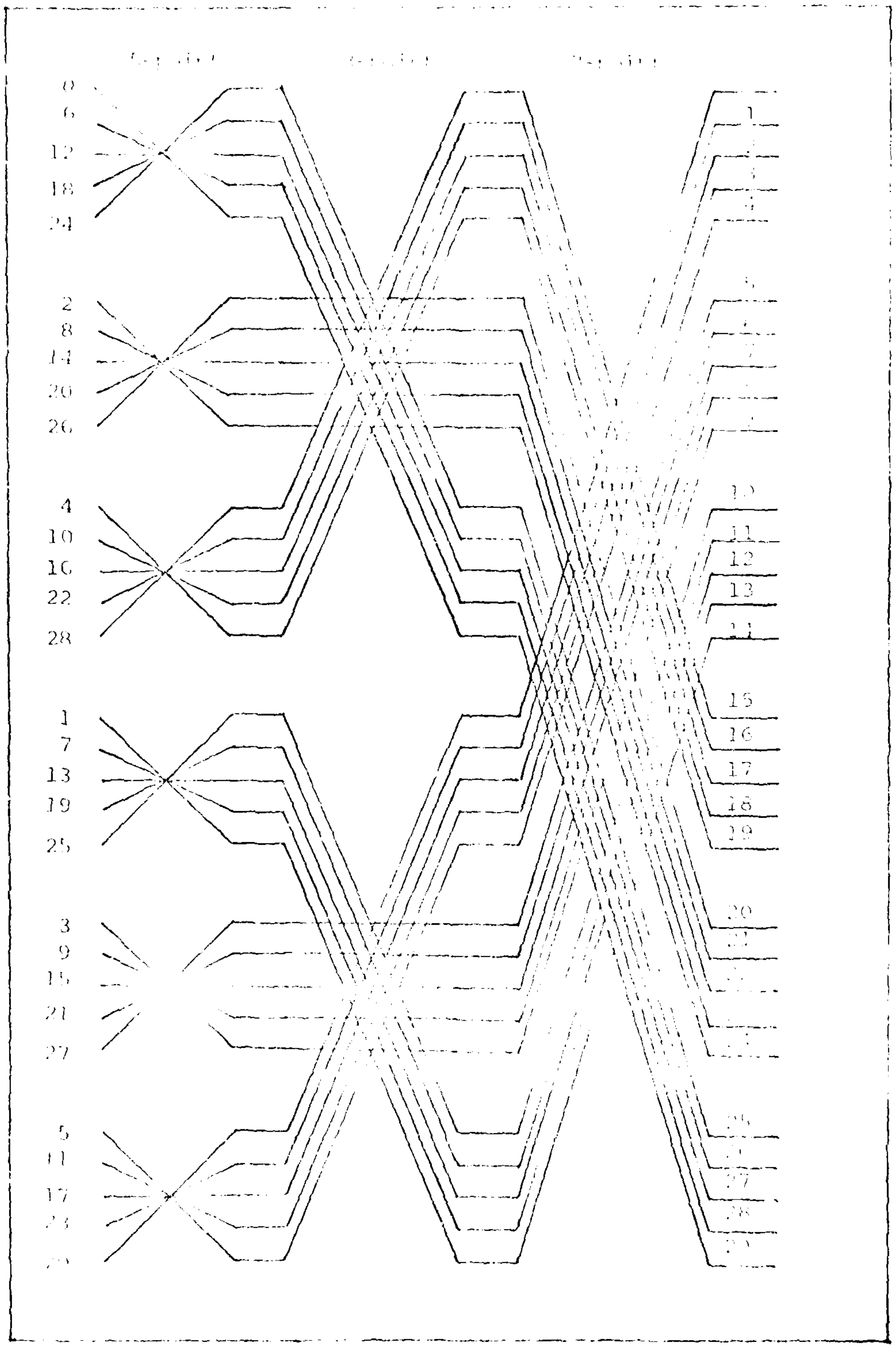


Figure 3.18. Butterfly Flowgraph for N=30.

be factored into $N = p \cdot q = 5 \cdot 6$ (representing a 5 by 6 matrix) and the expression becomes:

$$X(k) = \sum_{m=0}^{p-1} W_N^{mk} \sum_{r=0}^{q-1} x(pr+m) W_N^{prk} \qquad (3.104)$$

Now the inner sums can be expressed as the *q-point* DFTs:

$$G_m(k) = \sum_{r=0}^{q-1} x(pr+m) W_q^{rk} \qquad (3.105)$$

since

$$W_N^{prk} = \exp(-j2\pi prk/N) = \exp(-j2\pi prk/pq) = W_q^{rk} \qquad (3.106)$$

Using p=5 and q=6 in Eq (3.104) produces:

$$X(k) = \sum_{m=0}^{4} W_{30}^{mk} \sum_{r=0}^{5} x(5r+m)\, W_{30}^{5rk} \qquad (3.107)$$

The inner sum in Eq (3.107) is a 6-point DFT which can be decomposed into a 3 by 2 matrix by dividing the sequences x(5r+m) into three sequences, each two points long. The inner summation in Eq (3.107) can be represented using the notation of Eq (3.104) as:

$$G(k) = \sum_{s=0}^{p-1} W_N^{sk} \sum_{t=0}^{q-1} g(pt+s) W_N^{ptk} \qquad (3.108)$$

where N is now equal $p \cdot q = 3 \cdot 2$. Substituting p and q yields:

$$G(k) = \sum_{s=0}^{2} W_6^{sk} \sum_{t=0}^{1} g(3t+s) W_6^{3tk} \qquad (3.109)$$

This expression in Eq (3.109) can be substituted into Eq (3.107) to give:

$$X(k) = \sum_{m=0}^{4} W_{30}^{mk} \sum_{s=0}^{2} W_{6}^{sk} \sum_{t=0}^{1} x(15t+5s+m) W_{2}^{tk} \tag{3.110}$$

where

$r = 3t+s$

$g(3t+s) = x(5(3t+s)+m) = x(15t+5s+m)$

$W_6^{3tk} = \exp(-j2\pi \cdot 3tk/6) = \exp(-j2\pi \cdot tk/2) = W_2^{tk}$

$m = 0, 1, 2, 3, 4$

$s = 0, 1, 2$

$t = 0, 1$

The complete flowgraph is shown in Figure 3.18 and implements Eq (3.110).

3.4.2 Digit Reversal Algorithm (General). The permutation matrix P is required because the transformed result is in a digit reversed order. Given a factorization of $N = n_m \, n_{m-1} \ldots n_2 \, n_1$, the Fourier coefficient of $X(k)$ with:

$$k = k_m \, n_{m-1} \, n_{m-2} \ldots n_1 + \ldots + k_2 \, n_1 + k_1 \tag{3.111}$$

is found in location:

$$k' = k_1 \, n_2 \, n_3 \ldots n_m + k_2 \, n_3 \, n_4 \ldots n_m + \ldots + k_m \tag{3.112}$$

In general the interchange of k with k' can be done "in place" if N is factored such that (Singleton, 1977):

$$n_i = n_{m-i} \tag{3.113}$$

for i less than n-i. For this factoring k can be counted in natural order and k' in digit reversed order as described for fixed-radix algorithm bit-reversal.

To implement this technique for mixed radices N is factored into its prime factors and the "square" factors arranged symmetrically around the "square-free" factors of N. For example, let N=270 and be factored as:

$$3 \cdot 2 \cdot 3 \cdot 5 \cdot 3$$

Now the reordering, P, is factored into:

$$P = P_1 P_2 \qquad (3.114)$$

The reordering P_1 is "associated with the square factors of n and is done by pair interchanges as previously described, except that the digits of n corresponding to the square-free factors are held constant and the digits of the square factors are exchanged symmetrically" (Singleton, 1977). For example, if:

$$N = n_1 n_2 n_3 n_4 n_5 n_6 n_7 \qquad (3.115)$$

with $n_1 = n_7$, $n_2 = n_6$, and n_3, n_4, n_5 relatively prime, the interchange associated with the square factors n_1, n_7, n_2, and n_6 is given by:

$$k = k_7 n_6 n_5 \ldots n_1 + k_6 n_5 n_4 \ldots n_1 + k_5 n_4 n_3 n_2 n_1$$
$$+ k_4 n_3 n_2 n_1 + k_3 n_2 n_1 + k_2 n_1 + k_1 \qquad (3.116)$$

interchanged with:

$$k' = k_1 n_6 n_5 \ldots n_1 + k_2 n_5 n_4 \ldots n_1 + k_5 n_4 n_3 n_2 n_1$$
$$+ k_4 n_3 n_2 n_1 + k_3 n_2 n_1 + k_5 n_1 + k_6 \qquad (3.117)$$

This reordering P_1 in this example places each element of $X(k)$ in the correct segment of length $N/n_1\ n_2$, grouped in "subsequences" of $n_1\ n_2$ consecutive elements (Singleton, 1977). The next reordering P_2 then finished the reordering of each $n_3\ n_4\ n_5$ subsequences within each $N/n_1\ n_2$ segment.

The above factorization is used in the Singleton and IMSL mixed radix algorithms and generates a complicated FORTRAN code. A simpler alternative factorization was written by the author and used in his mixed radix algorithm. The simpler algorithm requires an additional two arrays of length N to store the intermediate results which detracts from the algorithms utility when longer sequence lengths are transformed. The details of this factorization are presented in Appendix E for interested readers.

3.4.3 Twiddle Factors. In Section 3.4.2 the factoring into F_i was described corresponding to a factor n_i. F_i can be factored to give a product $R_i\ T_i$ where the matrix T_i is one of N/n_i identical Fourier transforms of dimension n_i and R_i is a diagonal twiddle factor matrix. The elements of R_i are specified by the decimation-in-frequency version of the FFT (Singleton, 1977).

The twiddle factor matrix R_i multiplies each transform T_i of dimension n_i by $e^{j(Z)}$ where Z is an angle from the set:

$$0,\ Z,\ 2Z,\ \ldots,\ (n_i-1)Z \qquad (3.118)$$

and $Z = 2\pi/N$. No multiplication is needed for the zero angle which gives at most $N(n_i-1)/n_i$ complex multiplications for each transform stage i (Singleton, 1977). Additionally, the last stage of a decimation in frequency FFT requires no twiddles and the number of complex multiplications can be further reduced by (N-1). Given a factorization of $N = F_m F_{m-1}, \ldots F_2 F_1$ the number of twiddle factors for an N length sequence is

$$\sum_{i=1}^{m} (N(n_i-1)/n_i) - (N-1) \qquad (3.119)$$

This result is used in computing the number of real multiplications and additions required by an N length FFT.

3.4.4 <u>Real Operations Count for Computing Sine and Cosine Difference Equation.</u> Recall from Section 3.1 that trigonometric values used in an FFT can be computed using the difference equations:

$$\cos((k+1)a) = (C \cdot \cos(ka) - S \cdot \sin(ka)) + \cos(ka) \qquad (3.120)$$

$$\sin((k+1)a) = (C \cdot \sin(ka) + S \cdot \cos(ka)) + \sin(ka) \qquad (3.121)$$

where $a = 2\pi/N$ radians

$C = -2 \sin^2(a/2)$

$S = \sin(a)$

$\cos(0) = 1$

$\sin(0) = 0$

In the case of the author's mixed radix FFT the difference equations are computed N times and the sine and cosine results stored in two lookup tables. The difference equations are given by:

```
WKC(I) = C * WKC(I-1) - S * WKS(I-1) + WKC(I-1)     (3.122)
WKS(I) = C * WKS(I-1) + S * WKC(I-1) + WKS(I-1)     (3.123)
```

Eqs (3.122) and (3.123) require 4 real multiplications and 10 real additions each time they are computed. Given they are computed N times, the operations count is given by:

$$\text{real mult} = 4N \qquad (3.124)$$

$$\text{real adds} = 10N \qquad (3.125)$$

The IMSL and Singleton FFTs do not use the sine and cosine lookup tables in order to save memory arrays. Instead the sine and cosine values are computed as needed in the FFT program resulting in an intricate FORTRAN code. It was determined from the FORTRAN coded IMSL and Singleton FFTs that both utilize the same method of computing the sine and cosine difference equations. For this reason only the Singleton FFT algorithm was studied.

An algorithm which computes the number of real operations required was interpolated from "counters" placed in the FFT FORTRAN code in Appendix F. They provided the number of times that each section of the FFT subroutine was used to compute the sine and cosine values for different values of N. The labels for the counters are shown below along with the lines of FORTRAN code where they were positioned. The lines of code are shown in Appendix F.

I2C: Counter for the radix-2 difference equation in lines 2330 - 2340.

I2CL: Counter for the radix-2 sine and cosine library calls in lines 2650 - 2660.

I4C1: Counter for the radix-4 section which computes the sine and cosine terms of the W_N^k leg of the radix-4 in lines 3030 - 3040. Refer to Figure 3.19 which shows the radix-4 butterfly flowgraph.

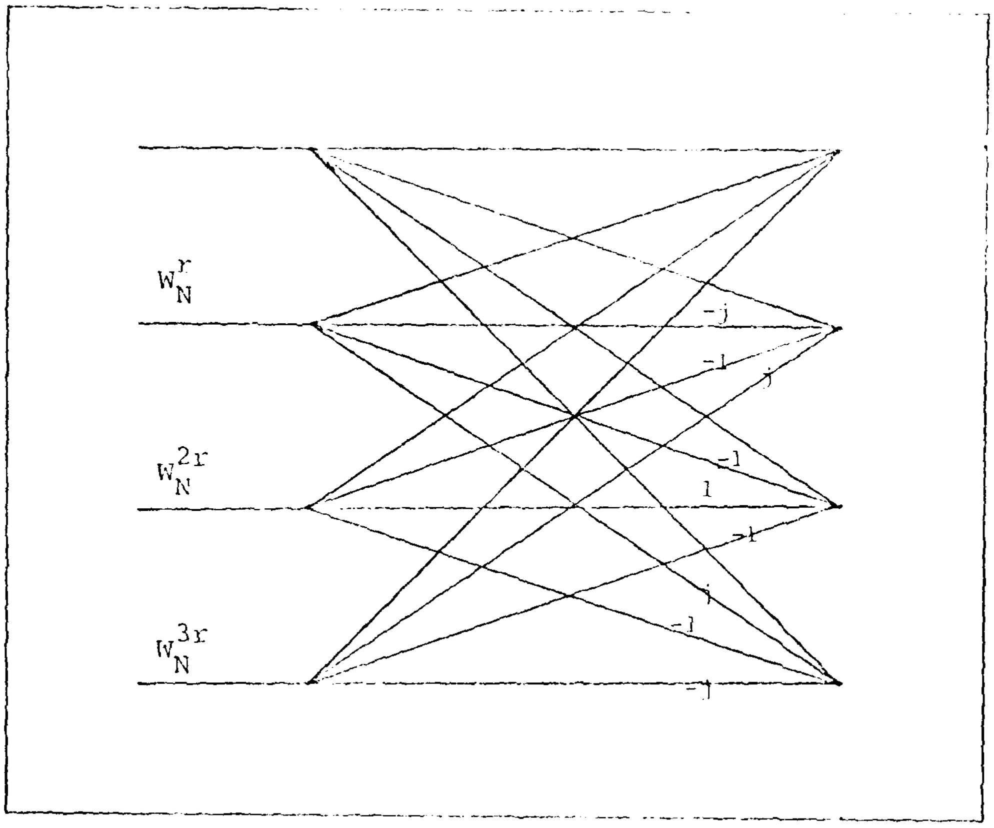


Figure 3.19. Radix-4 Butterfly Flowgraph Showing the 4 Twiddle Factor Multipliers.

I4C2: Counter for the radix-4 section which computes the sine and cosine terms of the W_N^{2k} and W_N^{3k} legs of the radix-4 butterfly flowgraph in lines 3140 - 3170.

I4CL: Counter for radix-4 sine and cosine library calls in lines 3690 - 3700.

IGTF: Counter for the general twiddle factors section in lines 4990 - 5000, which computes the sine and cosine for the W_N^k leg of the general radix-p FFT.

IGTFE: Counter for the general twiddle factors section which computes the sine and cosine for the remainder of the radix-p butterfly legs in lines 5170 - 5190.

IGTFL: Counter for the general radix-p sine and cosine library calls in lines 5290 - 5300.

Data was collected for over 70 values of N using these counters. A subset of the values were the 59 permissible sequence lengths of PFA and WFTA. Based on the results of these tests and study of the FORTRAN code FFT in Appendix F the general expressions for these counters were determined. Given that:

N = sequence length

NFAC(i) = factors of N (as factored by the Singleton subroutine)

M = number of factors of N

$KSPAN_i$ = N/(NFAC(1) * NFAC(2) ... * (NFAC(i-1))

then

$I2C_i$ = ($KSPAN_i$ - 3)/2 for $KSPAN_i \geq 4$ and odd

$I2C_i$ = ($KSPAN_i$ - 2)/2 for $KSPAN_i \geq 4$ and even

$I2C_i$ = 0 for $KSPAN_i < 4$

For the factors of 2 in N the expression for I2C becomes:

$$I2C = \sum_{i=1}^{k} (I2C_i) \quad \text{for } k \text{ factors of 2 in N} \qquad (3.126)$$

The expression for the number of sine and cosine calls during computation of a factor of 2 is $[KSPAN_i/70]$ where $[\cdot]$ represents truncation of the result inside the brackets. Using the "truncation" notation:

$$I2CL = \sum_{i=1}^{k} [KSPAN_i/70] \qquad (3.127)$$

The radix-4 section uses the same notational conventions for KSPAN and truncation. The expressions for I4C1, I4C2, and I4CL become:

$$I4C2_i = KSPAN_i - 1 \qquad (3.128)$$

$$I4CL_i = [KSPAN_i/32] \qquad (3.129)$$

$$I4C1_i = I4C2_i - I4CL_i \qquad (3.130)$$

For all factors of 4 in N the expression becomes:

$$I4C2 = \sum_{i=1}^{k} (KSPAN_i-1) \qquad (3.131)$$

$$I4CL = \sum_{i=1}^{k} [KSPAN_i/32] \qquad (3.132)$$

$$I4C1 = I4C2 - I4CL$$

where there are k factors of 4 in N.

The general expressions for IGTF, IGTFE, and IGTFL were derived to be:

$$IGTFL_i = [KSPAN_i/32] \qquad (3.134)$$

$$IGTF_i = KSPAN_i - IGTFL_i - 2 \qquad (3.135)$$

$$\mathrm{IGTFE}_i = (\mathrm{KSPAN}_i - 1)(\mathrm{NFAC}(i) - 1) \qquad (3.136)$$

The result for the general radix-p section becomes:

$$\mathrm{IGTF} = \sum_{i=1}^{k} \mathrm{IGTF}_i \qquad (3.137)$$

$$\mathrm{IGTFL} = \sum_{i=1}^{k} \mathrm{IGTFL}_i \qquad (3.138)$$

$$\mathrm{IGTFE} = \sum_{i=1}^{k} \mathrm{IGTFE}_i \qquad (3.139)$$

Eqs (3.124) through (3.139) were programmed in FORTRAN and then tabulated as a function of N in Table 3.4. These results identically match the tests conducted using the counters.

Examining the FORTRAN code where the counters were located gives the number of operations performed each time one of the counters was incremented. These results are presented in Table 3.5 for all the counters. The number of real operations, sine and cosine library calls, and exponentiations can be determined for all N length sequences by using Tables 3.4 and 3.5. The general expressions are given by:

$$\mathrm{KADD} = 4(\mathrm{I2C} + \mathrm{I4C2} + \mathrm{IGTF}) + 3(\mathrm{I4C1}) + 2(\mathrm{IGTFL}) \qquad (3.140)$$

$$\mathrm{KMULT} = 4(\mathrm{I2C} + \mathrm{I4C2} + \mathrm{IGTF} + \mathrm{IGTFE}) + 6(\mathrm{I4C1}) \qquad (3.141)$$

$$\mathrm{KEXP} = 2(\mathrm{I4C1}) \qquad (3.142)$$

3.4.5 Real Operations Count for Mixed FFTs. The real operations count is derived from the number of complex twiddle factors, the number of butterflies, and the number of sine and cosine terms computed using difference equations.

TABLE 3.4a

RESULTS OF [illegible] IN SINGLETON'S FFT

N	[illegible]	[illegible]	14C1	14C2	14CL	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	0	0	[illegible]	[illegible]	[illegible]
3	[illegible]	[illegible]	[illegible]	0	0	[illegible]	[illegible]	[illegible]
4	[illegible]	[illegible]	0	0	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	0	0	[illegible]	0	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	0	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	0	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	2	0	0	1	[illegible]	4
[illegible]	1	[illegible]	0	0	0	[illegible]	[illegible]	[illegible]
[illegible]	2	[illegible]	[illegible]	0	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	3	3	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	0	0	4	[illegible]	[illegible]
[illegible]	4	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	4
[illegible]	[illegible]	[illegible]	[illegible]	0	0	5	[illegible]	11
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	0	1
[illegible]	6	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	6	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	5	[illegible]	14
[illegible]	[illegible]	[illegible]	2	2	0	10	[illegible]	22
4	13	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	4
42	9	[illegible]	[illegible]	0	0	8	[illegible]	12
43	[illegible]	[illegible]	[illegible]	[illegible]	0	14	[illegible]	[illegible]
44	[illegible]	[illegible]	11	11	0	2	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	[illegible]
[illegible]	14	[illegible]	0	0	0	8	0	[illegible]
[illegible]	[illegible]	0	0	0	0	21	0	32
[illegible]	16	0	0	0	0	5	0	24
72	7	[illegible]	0	0	0	22	0	[illegible]

Table 3.4b

RESULTS OF COUNTERS IN SINGLETON'S FFT

N	[illegible]	12OL	14O1	14O2	14OL	[illegible]	[illegible]	[illegible]
60	0	0	19	19	0	2	[illegible]	12
64	21	0	0	0	0	12	[illegible]	32
90	6	0	0	0	0	29	[illegible]	66
105	[illegible]	1	0	0	0	17	1	[illegible]
112	0	0	27	27	0	2	0	[illegible]
120	43	[illegible]	0	0	0	6	[illegible]	22
126	9	0	0	0	0	41	1	94
140	33	1	0	0	0	12	[illegible]	58
144	0	0	34	35	1	12	[illegible]	[illegible]
168	61	1	0	[illegible]	[illegible]	12	0	[illegible]
180	14	0	0	0	0	61	1	[illegible]
210	57	1	0	0	0	37	1	[illegible]
240	0	0	56	59	1	21	0	[illegible]
252	20	0	0	0	0	84	1	[illegible]
280	100	3	0	0	0	12	0	58
315	0	0	0	0	0	120	3	[illegible]
336	0	0	81	83	2	28	1	[illegible]
360	43	0	0	0	0	119	3	[illegible]
420	101	3	0	0	0	76	2	[illegible]
504	61	1	0	0	0	165	5	364
560	0	0	135	139	4	29	[illegible]	[illegible]
630	57	1	0	0	0	[illegible]	[illegible]	[illegible]
720	0	[illegible]	174	179	5	69	1	[illegible]
840	304	9	0	0	0	76	1	[illegible]
1008	0	[illegible]	244	251	7	92	[illegible]	[illegible]
1260	101	3	0	0	0	448	14	[illegible]
1680	0	[illegible]	405	419	13	162	4	[illegible]
2520	304	9	0	0	0	[illegible]	[illegible]	[illegible]
5040	0	[illegible]	1220	1259	39	[illegible]	[illegible]	[illegible]

TABLE 3.5

OPERATIONS EXECUTED FOR EACH COUNTER

Counter	Real Add	Real Mult	Exponentiation	Sine Calls	Cosine Calls
I2C	4	4	0	0	0
I2C1	0	0	0	1	1
I4C1	3	6	2	0	0
I4C2	4	4	0	0	0
I4CL	0	0	0	1	1
IGTF	4	4	0	0	0
IGTFE	2	4	0	0	0
IGTFL	0	0	0	1	1

AD-A100 782 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 12/1
EFFICIENT COMPUTER IMPLEMENTATIONS OF FAST FOURIER TRANSFORMS.(U)
DEC 80 J D BLANKEN
UNCLASSIFIED AFIT/GE/EE/80D-9 NL

2 of 4
AD
A100782

Given that N is factored as:

$$N = p_1 \, p_2 \cdots p_m \tag{3.143}$$

the number of twiddle factors has been shown (Singleton, 1969) to be:

$$\sum_{i=1}^{m} (N(p_i - 1)/p_i) - (N-1) \tag{3.144}$$

where m is the total number of factors of N. The number of butterflies required for an N length sequence is given by:

$$\sum_{i=1}^{m} (N/p_i) \tag{3.145}$$

The total real operations count is determined by adding (a) the number of real multiplications and additions required per butterfly times Eq (3.145), plus (b) the complex twiddle factor multiplications times Eq (3.144), plus (c) the number of additions and multiplications given by Eq (3.140) and (3.141).

Assuming a complex multiplication requires four real multiplications and two additions a general expression for the real operations count can be determined for the mixed radix FFTs.

Singleton's mixed radix algorithm contains special transform sections for factors of 2, 3, 4, and 5 as well as a general section for other odd factors. This requires that N be represented as:

$$N = 2^r \, 3^s \, 4^t \, 5^u \, p_1^{m1} \, p_2^{m2} \cdots p_k^{mk} \tag{3.146}$$

The IMSL mixed radix FFT (FFTCC) does not have a special section for factors of 5 and uses the general section to transform these factors. The author's mixed radix FFT (FFTMR) has sections for 2, 3, 4, and 5 but does not have the general transform section. Only the detailed development of operations count for Singleton's algorithm is presented here because the other two algorithms are subsets thereof. The general expressions for real operations versus N are given for the other two algorithms in Appendix G and H.

The radix-2 section of the FORTRAN code for Singleton's algorithm is shown in Figure 3.20. For factors of two the twiddle (rotation) factor complex multiplications are computed in this section rather than the "general rotation section" to reduce the array indexing required. Using Eq (3.144) the total number of butterflies is rN/2 and from Eq (3.145) the total number of twiddle factors is rN/2 (neglecting the -(N-1) term which will be subtracted once the complete real operations count for all factors has been developed). The transform for factor of 2 (refer to Figure 3.20) is computed in lines 2200-2230 using 4 real additions, if no twiddles are required, or it is computed in lines 2450-2500 if twiddles are necessary. The general expression for factors of two becomes:

$$\text{real mult} = 4(rN/2) = 2rN \qquad (3.147)$$

$$\text{real adds} = 4(rN/2) + 2(rN/2) = 3rN \qquad (3.148)$$

The factors of 3 section shown in Figure 3.21 performs only the butterfly in this section and uses the general

```
[illegible]
      C1=[illegible]
[illegible]
      C1 = 0.5/(AK**2+S1**2) + 0.5
      [illegible]
      C1 = C1*AK
      [illegible]
   60 K2 = KK + KSPAN
      AK = A(KK) - A(K2)
      BK = B(KK) - B(K2)
      A(KK) = A(KK) + A(K2)
      B(KK) = B(KK) + B(K2)
      A(K2) = C1*AK - S1*BK
      B(K2) = S1*AK + C1*BK
      KK = K2 + KSPAN
      IF (KK.LT.NT) GO TO 60
      K2 = KK - NT
      [illegible]
      [illegible]
      [illegible]
      IF (KK.GT.K2) GO TO 60
      [illegible]
      IF (KK.LT.K2) GO TO 90
      [illegible]
      [illegible]
      IF (KK.LT.[illegible]) GO TO [illegible]
      GO TO 10
   80 [illegible]
      [illegible]
      [illegible]
      MM = MIN0(K1/2,MM+KLIM)
      GO TO 80
```

Figure 3.20. Radix-2 Section of Singleton's FFT.

```
[illegible]
2700=C  TRANSFORM FOR FACTOR OF 3 (OPTIONAL CODE)
2710=C
2720=100  K1 = KK + KSPAN
2730=        K2 = K1 + KSPAN
2740=        AK = A(KK)
2750=        BK = B(KK)
2760=        AJ = A(K1) + A(K2)
2770=        BJ = B(K1) + B(K2)
2780=        A(KK) = AK + AJ
2790=        B(KK) = BK + BJ
2800=        AK = -0.5*AJ + AK
2810=        BK = -0.5*BJ + BK
2820=        AJ = (A(K1)-A(K2))*S120
2830=        BJ = (B(K1)-B(K2))*S120
2840=        A(K1) = AK - BJ
2850=        B(K1) = BK + AJ
2860=        A(K2) = AK + BJ
2870=        B(K2) = BK - AJ
2880=        KK = K2 + KSPAN
2890=        IF (KK.LT.NN) GO TO 100
2900=        KK = KK - NN
2910=        IF (KK.LE.KSPAN) GO TO 100
2920=        GO TO 250
2930=C
```

Figure 3.21. Radix-3 Section of Singleton's FFT.

rotation (twiddle) section to twiddle the data (the general twiddle factor section is shown in Figure 3.24). Using Eqs (3.144) and (3.145) the number of butterflies for factors of 3 is sN/3 and the number of complex twiddles is s(2N/3). Examining lines 2760-2870 in Figure 3.21 shows 4 real multiplications and 12 real additions. Each complex twiddle requires 4 real multiplications and 2 real additions. The expression for the factors of 3 section becomes:

$$\begin{aligned}\text{real mult} &= 4(N/3)s + 4(2/3)Ns \\ &= 4sN \qquad (3.149)\end{aligned}$$

$$\begin{aligned}\text{real adds} &= 12(N/3)s + 2(2/3)Ns \\ &= 16sN/3 \qquad (3.150)\end{aligned}$$

The factors of 4 section in Figures 3.22a and b include the twiddles in the butterfly section to minimize array indexing. The number of butterflies computed for t factors of 4 is tN/4 and the number of complex twiddles is t(3N/4) from Eqs (3.144) and (3.145). From lines 3210-3320 and 3540-3570 the number of real additions per butterfly is 16. Every complex twiddle requires 4 real multiplications and 2 additions. Combining the butterfly and twiddle operations results in the general expression for factors of 4:

$$\text{real mult} = 4(3N/4)t = 3tN \qquad (3.151)$$

$$\begin{aligned}\text{real adds} &= 2(3N/4)t + 16(N/4)t \\ &= 3tN/2 + 8tN/2 = 11tN/2 \qquad (3.152)\end{aligned}$$

The transform section for factors of 5 shown in Figure 3.23 computes the butterflies for the u factors of 5. There are uN/5 butterflies and u(4N/5) complex twiddles based on

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
3100=       C1 = 0.5/(C2**2 + S1**2) + 0.5
3110=C      S1=C1*S1
3120=C      C1 = C1*C2
3130=       C1=C2
3140= 140   C2 = C1**2 - S1**2
3150=       S2 = C1*S1*2.0
3160=       C3 = C2*C1 - S2*S1
3170=       S3 = C2*S1 + S2*C1
3180= 150   K1 = KK + KSPAN
3190=       K2 = K1 + KSPAN
3200=       K3 = K2 + KSPAN
3210=       AKP = A(KK) + A(K2)
3220=       AKM = A(KK) - A(K2)
3230=       AJP = A(K1) + A(K3)
3240=       AJM = A(K1) - A(K3)
3250=       A(KK) = AKP + AJP
3260=       AJP = AKP - AJP
3270=       BKP = B(KK) + B(K2)
3280=       BKM = B(KK) - B(K2)
3290=       BJP = B(K1) + B(K3)
3300=       BJM = B(K1) - B(K3)
3310=       B(KK) = BKP + BJP
3320=       BJP = BKP - BJP
3330=       IF (ISN.LT.0) GO TO 180
3340=       AKP = AKM - BJM
3350=       AKM = AKM + BJM
3360=       BKP = BKM + AJM
3370=       BKM = BKM - AJM
3380=       IF (S1.EQ.0.0) GO TO 190
3390= 160   A(K1) = AKP*C1 - BKP*S1
3400=       B(K1) = AKP*S1 + BKP*C1
3410=       A(K2) = AJP*C2 - BJP*S2
3420=       B(K2) = AJP*S2 + BJP*C2
3430=       A(K3) = AKM*C3 - BKM*S3
3440=       B(K3) = AKM*S3 + BKM*C3
3450=       KK = K3 + KSPAN
3460=       IF (KK.LE.NT) GO TO 150
3470= 170   KK = KK - NT + JC
3480=       IF (KK.LE.MM) GO TO 130
```

Figure 3.22a. Radix-4 Section of Singleton's FFT.

```
[illegible]
3580=        IF [illegible] TO 1?0
3590= 140    A(J) = [illegible]
3600=        B(K1) = [illegible]
[illegible]
3620=        A(K2) = [illegible]
3630=        B(K2) = [illegible]
3640=        B(K3) = [illegible]
3650=        KK = K3 + KSPAN
3660=        IF (KK.LE.NT) GO TO 150
3670=        GO TO 120
3680= 160    C1 = [illegible]
3690=        C1 = COS(C1)
3700=        S1 = SIN(C1)
3710=        MM = MIN0([illegible])
3720=        GO TO 140
3730=
```

Figure 3.22b. Radix-4 Section of Singleton's FFT.

```
[illegible]

[illegible]

[illegible]
[illegible]
[illegible]
3820=        AKP = A(K1) + A(K4)
[illegible]
[illegible]
3850=        BKM = B(K1) - B(K4)
[illegible]
3870=        AJM = A(K2) - A(K3)
3880=        BJP = B(K2) + B(K3)
3890=        BJM = B(K2) - B(K3)
[illegible]
3910=        BB = B(KK)
3920=        A(KK) = AA + AKP + AJP
3930=        B(KK) = BB + BKP + BJP
3940=        AK = AKP*C72 + AJP*C2 + AA
3950=        BK = BKP*C72 + BJP*C2 + BB
3960=        AJ = AKM*S72 + AJM*S2
3970=        BJ = BKM*S72 + BJM*S2
3980=        A(K1) = AK - BJ
3990=        A(K4) = AK + BJ
4000=        B(K1) = BK + AJ
4010=        B(K4) = BK - AJ
4020=        AK = AKP*C2 + AJP*C72 + AA
4030=        BK = BKP*C2 + BJP*C72 + BB
4040=        AJ = AKM*S2 - AJM*S72
4050=        BJ = BKM*S2 - BJM*S72
4060=        A(K2) = AK - BJ
4070=        A(K3) = AK + BJ
4080=        B(K2) = BK + AJ
4090=        B(K3) = BK - AJ
4100=        KK = K4 + KSPAN
4110=        IF [illegible]
4120=        KK = KK - NN
4130=        IF [illegible]
[illegible]
4150=C
```

Figure 3.23. Radix-5 Section of Singleton's FFT.

Eqs (3.144) and (3.145). Examination of lines 3820-4090 in Figure 3.23 shows 16 real multiplications and 32 real additions are required per butterfly. Combining the butterfly and complex twiddle operations provides the general expression for real operation for factors of 5:

$$\text{real mult} = 16(N/5)u + 4(4N/5)u$$
$$= 32uN/5 \qquad (3.153)$$
$$\text{real adds} = 32(N/5)u + 2(4N/5)u$$
$$= 8uN \qquad (3.154)$$

where u is the number of factors of 5 in N.

The general transform section for odd prime factors is more complex than the special factors sections. To aid in describing the number of real operations a p-radix is defined such that p is an odd prime greater than 5 with an associated "mi" integer power. The real operations count for the general section does not include additions associated with array indexing nor does it count multiplications and additions needed to recursively compute the sine and cosine terms.

Based on the FORTRAN program for the odd factors shown in Figure 3.24a and b there are five sources of real operations for each p_i factor. The first source shown in lines 4310-4360 is computing the $(p_i-1)/2$ complex multipliers for the butterfly legs which require:

$$\text{real mult} = 4(p_i-1)/2 = 2(p_i-1) \qquad (3.155)$$
$$\text{real adds} = 2(p_i-1)/2 = (p_i-1) \qquad (3.156)$$

```
[illegible]
[illegible]
[illegible]
[illegible]
4280=       C1 = 1.
4290=       S1 = 0.0
4300=       J = 1
4310= 240   CK(J) = CK(K)*C1 + SK(K)*S1
4320=       SK(J) = CK(K)*S1 - SK(K)*C1
4330=       K = K - 1
4340=       CK(K) = CK(J)
4350=       SK(K) = -SK(J)
4360=       J = J + 1
4370=       IF (J.LT.K) GO TO 240
4380= 250   K1 = KK
4390=       K2 = KK + KSPNN
4400=       AA = A(KK)
4410=       BB = B(KK)
4420=       AK = AA
4430=       BK = BB
4440=       J = 1
4450=       K1 = K1 + KSPAN
4460= 260   K2 = K2 - KSPAN
4470=       J = J + 1
4480=       AT(J) = A(K1) + A(K2)
4490=       AK = AT(J) + AK
4500=       BT(J) = B(K1) + B(K2)
4510=       BK = BT(J) + BK
4520=       J = J + 1
4530=       AT(J) = A(K1) - A(K2)
4540=       BT(J) = B(K1) - B(K2)
4550=       K1 = K1 + KSPAN
4560=       IF (K1.LT.K2) GO TO 260
4570=       A(KK) = AK
4580=       B(KK) = BK
4590=       K1 = KK
4600=       K2 = KK + KSPNN
4610=       J = 1
4620= 270   K1 = K1 + KSPAN
4630=       K2 = K2 - KSPAN
4640=       JJ = J
4650=       AK = AA
4660=       BK = BB
4670=       AJ = 0.0
4680=       BJ = 0.0
4690=       K = 1
4700= 280   K = K + 1
[illegible]
```

Figure 3.24a. General Factor Section of Singleton's FFT.

```
[illegible]=        IF [illegible].LT.[illegible] GO TO [illegible]
[illegible]
4870=        IF [illegible] GO TO [illegible]
[illegible]=        [illegible] = [illegible]
[illegible]=        IF [illegible] GO TO [illegible]
[illegible]=C
[illegible]=C MULTIPLY BY ROTATION FACTOR (EXCEPT FOR FACTORS OF 2 AND 4)
4920=C
4930= 290  IF (I.EQ.M) GO TO 3[illegible]
4940=        KK = JC + 1
4950= 300  C2 = 1.0 - CD
4960=        S1 = SD
4970=        MM = MIN0(KSPAN,KLIM)
4980=        GO TO 320
4990= 310  C2 = C1 - (CD*C1+SD*S1)
5000=        S1 = S1 + (SD*C1-CD*S1)
5010=C
5020=C THE FOLLOWING THREE STATEMENTS COMPENSATE FOR TRUNCATION
5030=C ERROR.  IF ROUNDED ARITHMETIC IS USED, THEY MAY
5040=C BE DELETED.
5050=C
5060=C       C1 = 0.5/(C2**2+S1**2) + 0.5
5070=C       S1 = C1*S1
5080=C       C2 = C1*C2
5090= 320  C1 = C2
5100=        S2 = S1
[illegible]=        KK = KK + KSPAN
5120= 330  AK = A(KK)
5130=        A(KK) = C2*AK - S2*B(KK)
[illegible]
5150=        KK = KK + KSPNN
5160=        IF (KK.LE.NT) GO TO 330
5170=        AK = S1*S2
5180=        S2 = S1*C2 + C1*S2
5190=        C2 = C1*C2 - AK
5200=        KK = KK - NT + KSPAN
5210=        IF (KK.LE.KSPNN) GO TO 330
5220=        KK = KK - KSPNN + JC
5230=        IF (KK.LE.MM) GO TO 310
5240=        IF (KK.LT.KSPAN) GO TO 340
5250=        KK = KK - KSPAN + JC + INC
```

Figure 3.24b. General Factor Section of Singleton's FFT.

These complex multipliers are computed only once for each new factor p_i, e.g., for N=28=7·4, the factor 7 requires (7-1)/2 complex multipliers. If N=196=7·7·4 there are still only (7-1)/2 complex multipliers needed.

The second source of real operations is produced by computing the butterfly transmittances which require only real additions. From Eq (3.145) there are $(mi)N/p_i$ butterflies required for the (mi) factors of p_i. For each butterfly there are $(p_i-1)/2$ transmittances which require only real additions. Examining lines 4470-4540 in Figure 3.24a show that the $(p_i-1)/2$ transmittances require 6 additions. Combining these results produces the general expression for the real additions:

$$\begin{aligned}\text{real adds} &= (6(p_i-1)/2)(mi)N/p_i \\ &= 3N(mi)(p_i-1)/p_i \qquad (3.157)\end{aligned}$$

The third source of operations is produced by the $(p_i-1)^2/4$ butterfly transmittances which require real multiplications and additions. Lines 4510-4750 in Figure 3.24b show there are 4 real multiplications and 4 real additions needed. Combining this with the number of transmittances and butterflies gives:

$$\begin{aligned}\text{real mult} &= 4((mi)N/p_i)((p_i-1)^2/4) \\ &= (mi)N(p_i-1)^2/p_i \qquad (3.158)\end{aligned}$$

$$\text{real adds} = (mi)N(p_i-1)^2/p_i \qquad (3.159)$$

The fourth source results from computing the $(p_i-1)/2$ butterfly additions for each $(mi)N/p_i$ butterfly. Examining lines 4800-4830 show that this function requires 4 real additions. Combining these results give the total as:

$$\text{real adds} = ((mi)N/p_i)4(p_i-1)/2$$
$$= 2(mi)N(p_i-1)/p_i \tag{3.160}$$

The final source of real operations is shown in Figure 3.24b lines 5120-5140 which performs the complex twiddle multiplications. From Eq (3.144) there are $(mi)N(p_i-1)/p_i$ complex twiddles which provide the general expression:

$$\text{real mult} = 4(mi)N(p_i-1)/p_i \tag{3.161}$$

$$\text{real adds} = 2(mi)N(p_i-1)/p_i \tag{3.162}$$

Combining Eqs (3.145) through (3.162) give the expression for the real operations in the general odd factors section:

$$\text{real mult} = \sum_{i=1}^{k} 2(p_i-1) + (mi)N(p_i-1)^2 p_i$$
$$+ 4(mi)N(p_i-1)/p_i \tag{3.163}$$

$$\text{real adds} = \sum_{i=1}^{k} ((p_i-1) + 3N(mi)(p_i-1)/p_i$$
$$+ (mi)N(p_i-1)^2/p_i + 2(mi)N(p_i-1)/p_i$$
$$+ 2(mi)N(p_i-1)/p_i)$$
$$= \sum_{i=1}^{k} (p_i-1) + 7N(mi)(p_i-1)/p_i$$
$$+ (mi)N(p_i-1)^2/p_i \tag{3.164}$$

Assuming that the sequence can be factored into $N = 2^r\ 3^s\ 4^t\ 5^u\ p_1^{m1}\ p_2^{m2} \ldots p_k^{mk}$ the expressions for the total number of real operations can be written using Eqs (3.140) through (3.164) as:

$$\text{real mult} = 2rN + 4sN + 3tN + 32uN/5 + \sum_{i=1}^{k} (2(p_i-1) + (mi)N(p_i-1)^2/p_i + 4(mi)N(p_i-1)/p_i) - 4(N-1) + \text{KMULT} \quad (3.165)$$

$$\text{real adds} = 3rN + 16sN/3 + 11tN/2 + 8uN + \sum_{i=1}^{k} ((p_i-1) + 7N(mi)(p_i-1)/p_i + (mi)N(p_i-1)^2/p_i) - 2(N-1) + \text{KADD} \quad (3.166)$$

Notice that Eqs (3.165) and (3.166) have the corresponding 4(N-1) and 2(N-1) real operations subtracted from the total multiplications and additions because the first stage of any FFT decimation-in-time does not require the "twiddle factors" (likewise with the last stage of an FFT decimation-in-frequency). These equations also include KADD and KMULT which are the real operations required to compute the recursive sine and cosine difference equation.

Similar expressions and derivations were performed for the IMSL FFT and the author's FFT but due to the redundancy they were derived in Appendices G and E respectively. The general expression for real operations required by the IMSL mixed radix FFT (where $N = 2^r\ 3^s\ 4^t\ p_i^{m1}\ p_2^{m2} \ldots p_k^{mk}$) is given by:

$$\text{real mult} = 2rN + 4sN + 3tN + \sum_{i=1}^{k} (2(p_i-1) + 4(mi)N(p_i-1)/p_i + (mi)N(p_i-1)^2/p_i) - 4(N-1) + KMULT \quad (3.167)$$

$$\text{real adds} = 3rN + 6sN + 1tN/2 + \sum_{i=1}^{k} ((p_i-1) + 8(mi)N(p_i-1)p_i + N(mi)(p_i-1)^2/p_i) - 2(N-1) + KADD \quad (3.168)$$

where KMULT and KADD are the multiplies and adds needed to compute the sine and cosine terms. The general expression for real operations required by the author's mixed radix FFT (where $N = 2^r\, 3^s\, 4^t\, 5^u$) is given by:

$$\text{real mult} = 2rN + 4sN + 3tN + 32uN/5 - 4(N-1) + 4N \quad (3.169)$$

$$\text{real adds} = 3rN + 16sN/3 + 11tN/2 + 8uN - 2(N-1) + 10N \quad (3.170)$$

The real operations count for Singleton's mixed radix FFT is shown for $N \leq 200$ in Figures 3.26 and 3.27. The operations count plotted includes only the additions and multiplications for the butterfly and twiddle factors in order to demonstrate the N^2 "upper bound" and the $N \log_2 N$ "lower bound". The N^2 upper bound occurs in the mixed radix FFTs when a prime number must be transformed. The $N \log_2 N$ lower bound is reached when $N=2^m$. In between the N^2 and $N \log_2 N$ bounds there are other "bounds" which are observed in Figure 3.25. The dashed lines represent numbers

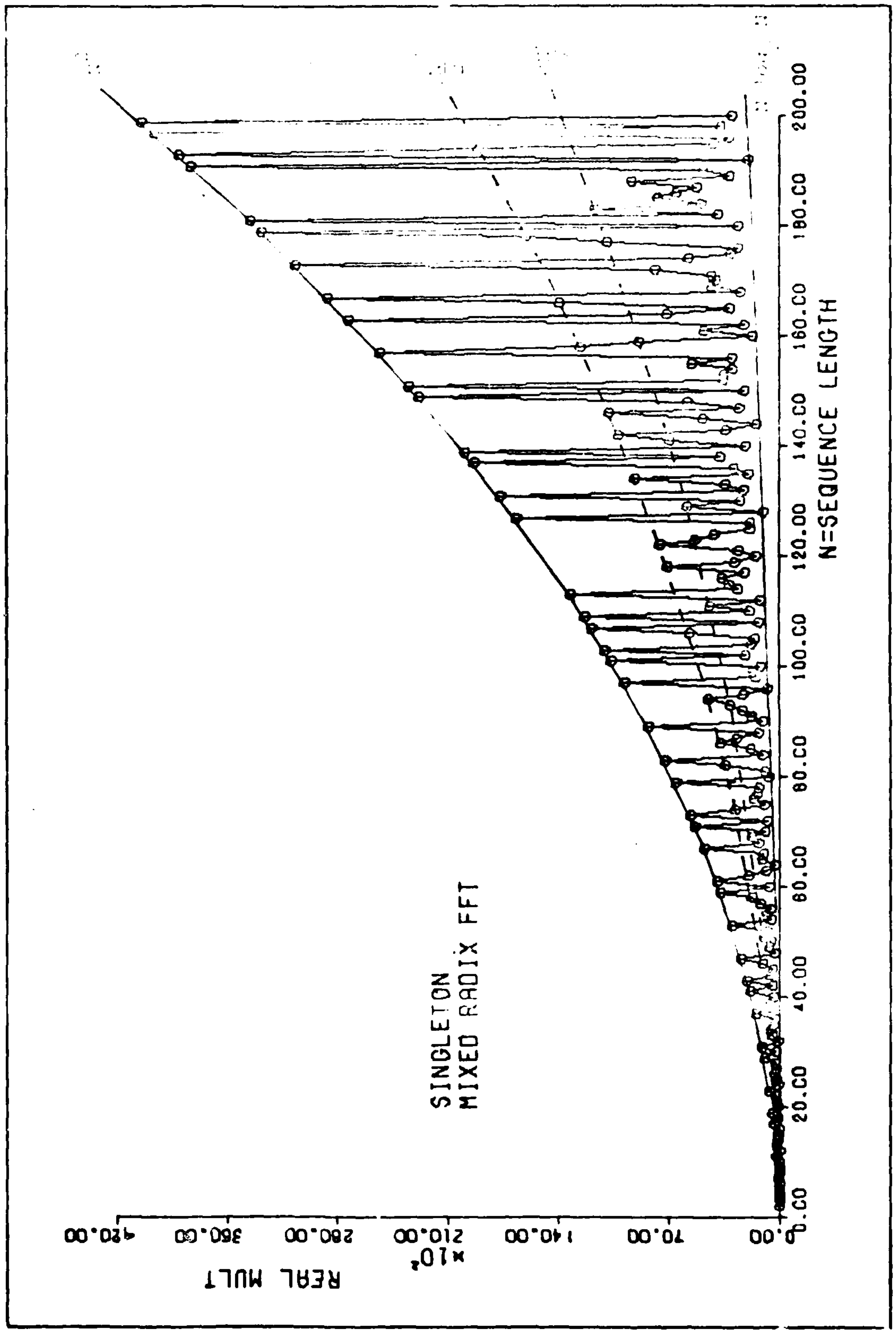


Figure 3.25. Multiplications vs N for Singleton's FFT ($N \leq 200$).

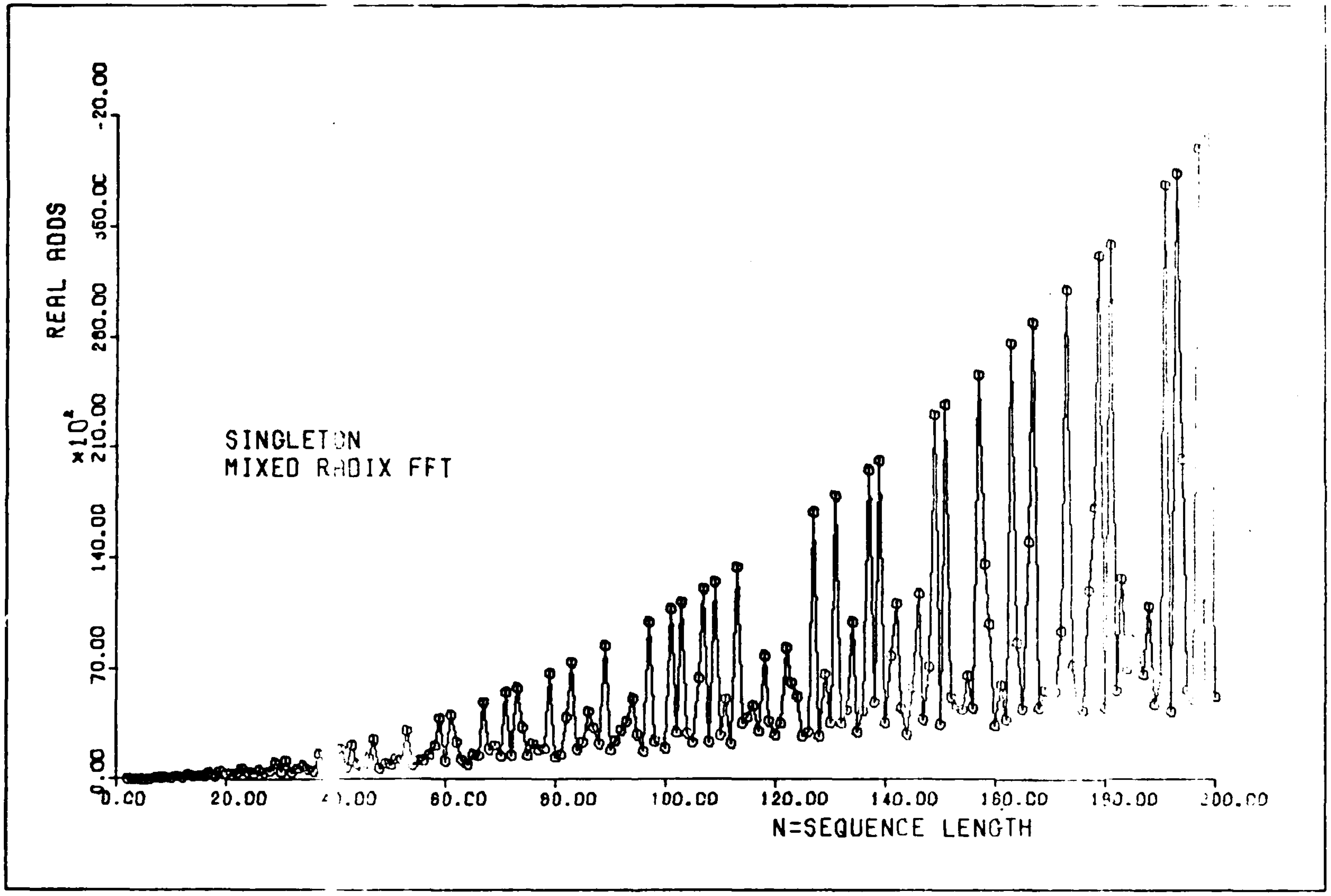


Figure 3.26. Additions vs N for Singleton's FFT (N≤200).

which are not primes, but are not highly factorable either. The dashed line approaches $N \log_2 N$ as N becomes more factorable.

The relative efficiency of radix 2, 3, 4 and 5 FFTs is observed in Figures 3.27 and 3.28. These figures plot real operations counts for the mixed radix FFT for N less than 250 (where N is divisible by 2, 3, 4 and 5 only) and annotate the integer powers of 2, 3, 4 and 5. Notice that the fixed radix-2 and 4 provide the "lower bound" and the radix-3 and 5 provide the "upper bound" on the number of real operations which shows that integer powers of 2 and 4 require the least number of real operations and radix-3 and 5 the most. Other combinations of factors, i.e., N=120=5*4*3*2, have real operations counts which fall between the "bounds".

3.4.6 Memory Requirements for Mixed Radix FFTs. As in the case of fixed radix algorithms, a major consideration in selecting a particular mixed radix algorithm is the memory required to execute the FFT subroutine given the memory storage limitations of the computer to be used. The memory requirements for the three mixed radix FFTs is given here as a function of the sequence length N. Each algorithm has program and memory array requirements which are listed below.

All the algorithms were compiled on the CDC Cyber system at AFIT and the program memory required by each subroutine was determined from a "load map" generated by the

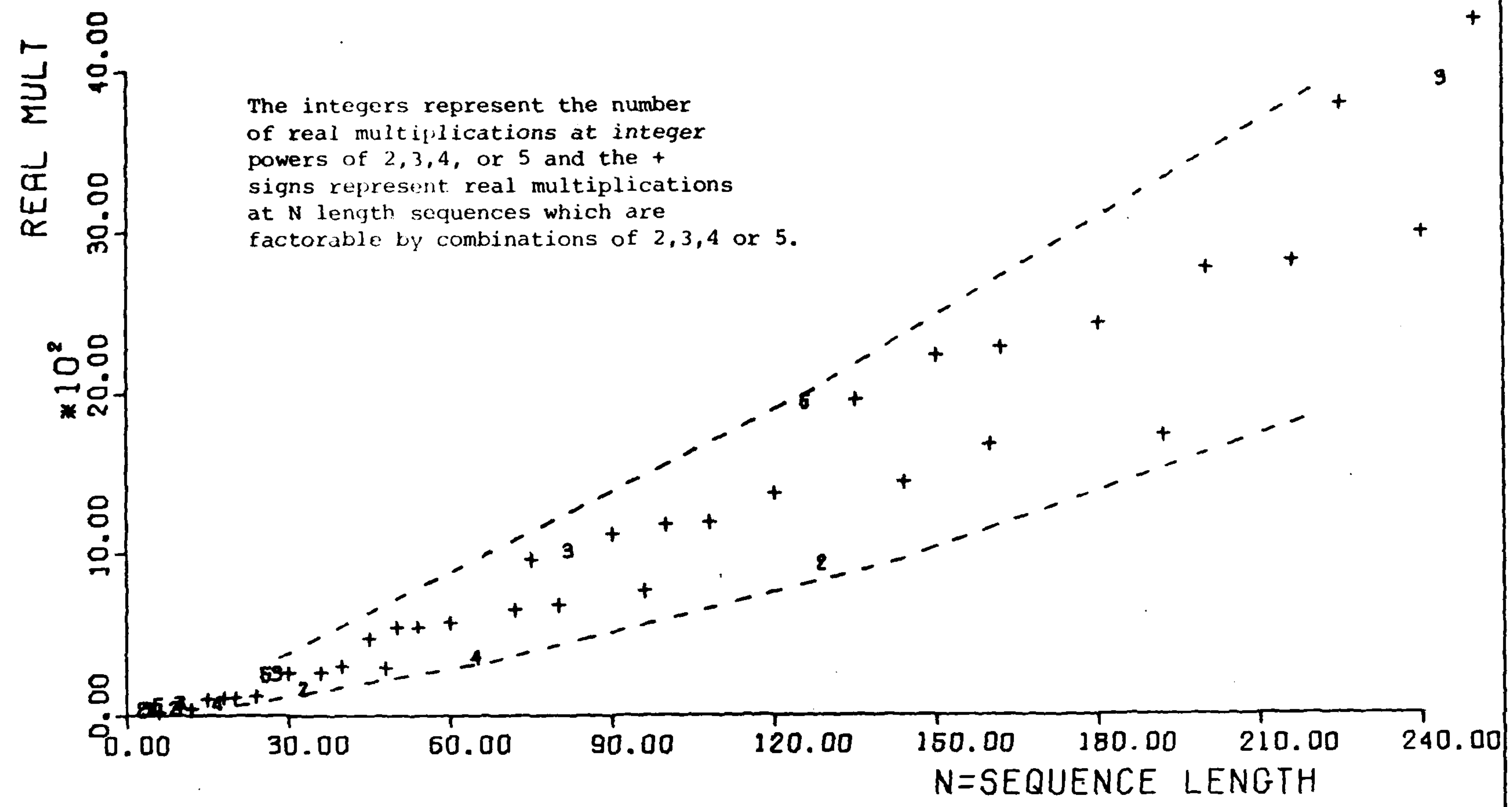
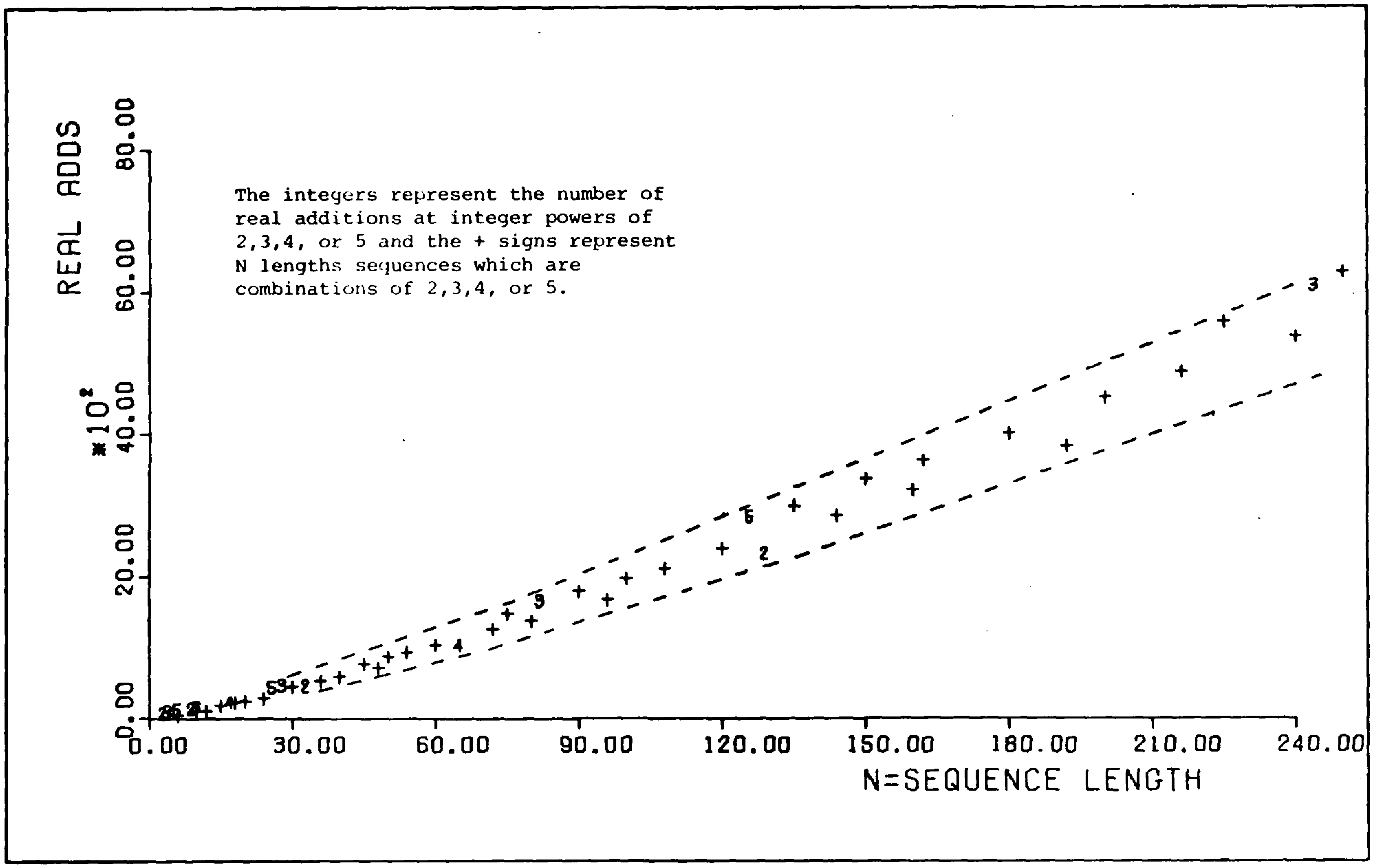


Figure 3.27. Multiplications vs N for Multiples of 2,3,4 and 5.



The integers represent the number of real additions at integer powers of 2,3,4, or 5 and the + signs represent N lengths sequences which are combinations of 2,3,4, or 5.

Figure 3.28. Additions vs N for Multiples of 2,3,4, and 5.

command MAP, PART. This load map gives the size of all programs used during execution. The array storage requirements were determined from the FORTRAN coded programs and reference material provided with the IMSL and Singleton FFT subroutines. The general expression for memory requirements for each FFT subroutine (as a function of N) is given below.

The subroutine written by the author requires 899 words of program memory. This subroutine (FFTMR) also requires the "calling" program to dimension 6 arrays (A, B, AT, BT, WKS, and WKC) to length N. (Use of these arrays is explained in Appendix E). This gives the total memory array required as:

$$\text{FFTMR memory} = 6N \qquad (3.171)$$

The mixed radix subroutine written by Singleton (FFTSNG) requires 1100 words of program memory. Four arrays (AT, BT, CK, SK) are dimensioned to equal the maximum prime factor of N. If there are no prime factors greater than 5 these arrays may be reduced to 1. A fifth array (NP) is dimensioned to at least one less than the product K of the square-free factors (see Glossary) of N. If N contains at most one square-free factor this array can be reduced to M + 1 where M is the maximum number of prime factors of N. Two more arrays, (XR, and XI) are dimensioned to length N. The total memory array storage becomes:

$$\text{FFTSNG memory} = 2 \cdot N + 4 \cdot \text{MAXPF} + (K-1 \text{ or } M+1) \qquad (3.172)$$

where

N	=	Sequence length
MAXPF	=	Maximum prime factor of N
K	=	Product of square-free factors
M	=	Maximum number of prime factors

NOTE: K-1 or M+1 is selected in Eq (3.172) based on the number of square-free factors of N as described in the preceding paragraph.

The mixed radix subroutine (FFTCC) provided as part of the IMSL package on the CDC Cyber system requires 1061 words of program memory. A complex array (A) must be dimensioned to length N and two other arrays (IWK and WK) are dimensioned to length "IWORD", where:

$$\text{IWORD} = 3 \cdot M + 3 + \text{MAX}\,(4 \cdot M + 7 + 6 \cdot K,\; KB + 1 + 2 \cdot JK) \qquad (3.173)$$

To define the quantities M, K, KB and JK a prime factor decomposition of N is required such that:

$$N = f_1^2\, f_2^2 \ldots f_{KT}^2\, f_{KT+1} \ldots f_{KT+JT}$$

where each f_j is a prime number (other than 1) and $f_i \neq f_r$ given that:

$$i, r \geq KT + 1$$

$$KT \geq 0;\; JT \geq 0$$

Then:

$$M = 2KT + JT \qquad (3.174)$$

is the number of prime factors in N and:

$$K = \max_{1 \leq j \leq KT + JT} (f_j) \qquad (3.175)$$

is the largest prime factor of N. KB and JK are defined as follows:

$$JK = 1 \cdot f_1 \cdot f_2 \ldots F_{KT} \qquad (3.176)$$

where JK = 1 if KT = 0 and

$$KB = N/(JK)^2 - 2 \qquad (3.177)$$

Once M, K, JK, and KB are determined they are substituted into Eq (3.173) to determine the value of IWORD, the actual work storage requirement. Counting only the arrays for the work vectors (IWK and WK) and the data arrays (A and B) gives the total array memory required for the IMSL FFT:

$$\text{Memory} = 2 * N + \text{IWORD} * 2 \qquad (3.178)$$

An example of N=2100 is used to demonstrate the use of Eqs (3.172) through (3.178) in computing the memory array required by the IMSL and Singleton subroutines. For N=2100 the factors are $2^2 \cdot 5^2 \cdot 3 \cdot 7$ for which FFTSNG memory becomes:

N = 2100 = sequence length

MAXPF = 7 = maximum prime factor in N

K = 3·7 = 21 = product of the square free factors

M = 6 = maximum number of prime factors

Using Eq (3.172) the expression for FFTSNG memory array is given by

$$2 \cdot 2100 + 4 \cdot 7 + (20 \text{ or } 7) = 4248 \qquad (3.179)$$

NOTE: There are two square-free factors 3 and 7, therefore choose 20 for the last term of Eq (3.179).

If this subroutine were used on the Cyber 74 computer, the program memory is added to the memory array to give a total memory of:

$$\text{memory} = 4248 + 1100 = 5348 \text{ words} \tag{3.180}$$

The same example of N=2100 is applied to the IMSL memory equation where:

$$N = f_1^2\ f_2^2 \cdots f_{KT}^2\ f_{KT+1} \cdots f_{KT+JT}$$

$$= 2^2 \cdot 5^2 \cdot 3 \cdot 7 = 2100 \tag{3.181}$$

From Eq (3.174) the expression for M becomes:

$$M = 2 \cdot KT + JT = 2 \cdot 2 + 2 = 6 \tag{3.182}$$

which is the number of prime factors in N. The largest prime factor in N is given by Eq (3.175):

$$K = \max_{\underline{1} \leq j \leq KT+JT} (f_j) = 7 \tag{3.183}$$

JK, which is the product of the "square-factors", is:

$$JK = \underline{1} \cdot f_1 \cdot f_2 \ldots f_{KT} = 2 \cdot 5 = 10 \tag{3.184}$$

and KB is

$$KB = N/(JK)^2 - 2 = 2100/100 - 2 = 19 \tag{3.185}$$

The results of Eq (3.181) through (3.185) provide the size of the work vector IWORD given by Eq (3.173).

$$IWORD = 3M + 3 + MAX\ (4M + 7 + 6K,\ KB+1+2JK)$$

$$= 18 + 3 + MAX\ (24 + 7 + 42,\ 19+1+20)$$

$$= 21 + MAX\ (73,\ 40) = 94$$

Substituting IWORD=72 and N=2100 into Eq (3.178) gives the memory array for FFTCC as:

$$2N + 2IWORD = 4200 + 94 = 4294 \tag{3.186}$$

Using this subroutine on the Cyber 74 computer requires 1061 words of program memory which makes the total memory required equal to:

$$4294 + 1061 = 5355 \text{ words} \tag{3.187}$$

For this length N=2100 sequence the Singleton FFTSNG used less memory (5348) than the IMSL FFTCC (5355).

The array memory requirements given by Eq (3.172) and (3.178) are plotted in Figures 3.29 and 3.30 for N less than 200. It is readily observed that selective adjustment of N to be highly factorable (composite) minimizes the memory required by subroutines FFTCC or FFTSNG. As an example of how prime numbers increase the memory array sizes, consider N = 2099 for each algorithm. For FFTSNG the variables are MAXPF = 2099, K = 2099, and M = 1. Since N = 2099 contains only one square-free factor the array NP can be dimensioned to M+1=2. The memory array for FFTSNG becomes:

$$2N + 4 \cdot MAXPF + 2 = 12594 \text{ words of memory array}$$

Adding the program memory of 1100 yields the total memory required to execute the FFTSNG on the Cyber 74:

$$\text{memory} = 12594 + 1100 = 13694 \tag{3.188}$$

For the IMSL FFT the variables are K = 2099, JK = 1, KT = 0, JT = 1, KB = 2097, and M = 1. The expression for IWORD becomes:

$$\begin{aligned} IWORD &= 3M + 3 + MAX(4M+7+6K,\ KB+1+2JK) \\ &= 3 + 3 + MAX(12605,\ 2100) = 12611 \end{aligned}$$

The total memory assuming execution on the Cyber 74 system is:

$$2N + 2 \cdot IWORD = 2 \cdot 2099 + 2 \cdot 12611 = 29420 \tag{3.189}$$

which is 5.5 times larger than the total memory for N=2100.

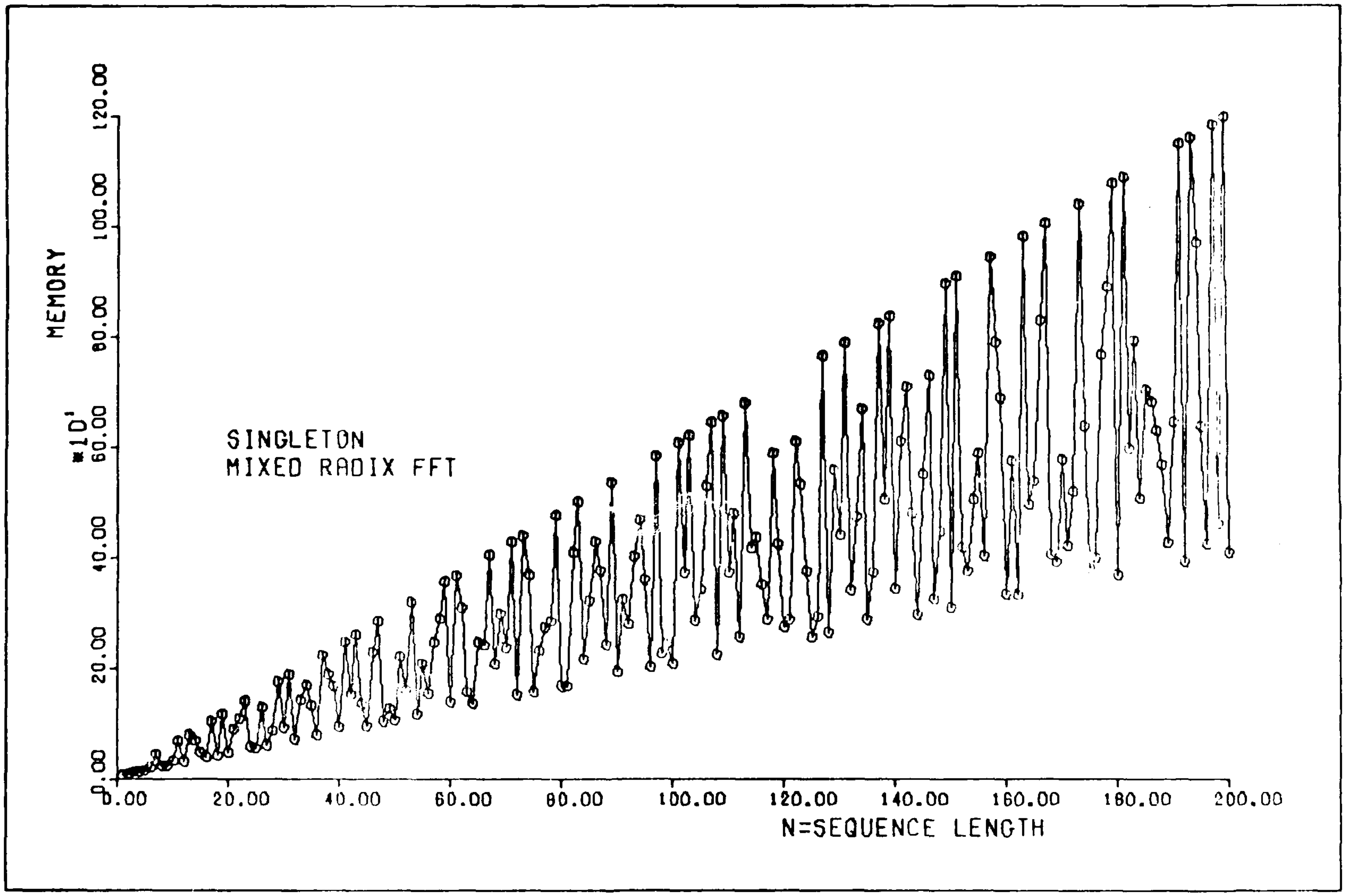


Figure 3.29. Memory Array vs N (≤ 200) for Singleton's FFT.

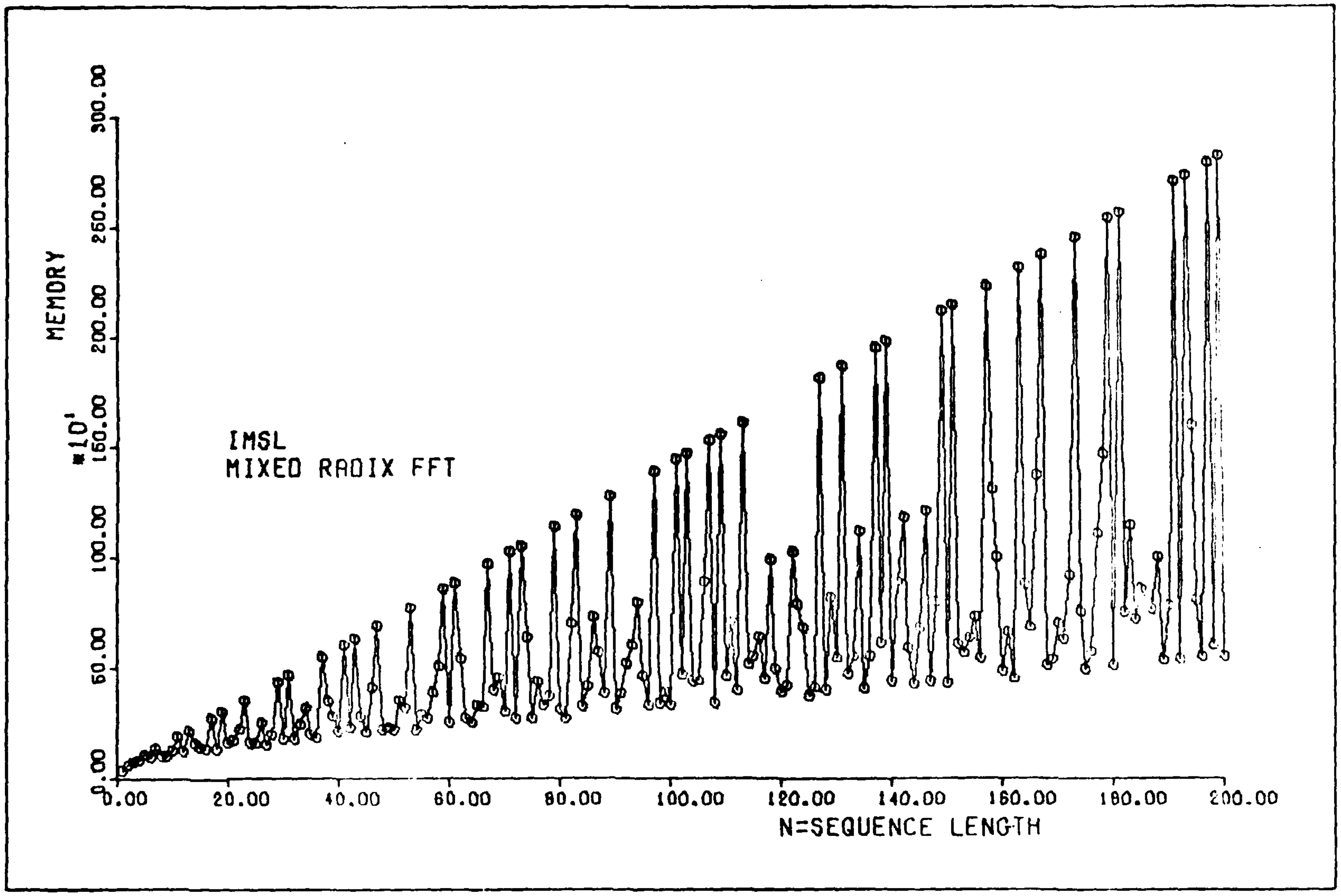


Figure 3.30. Memory Array vs N (≤ 200) for IMSL's FFT.

3.5 Fourier Transforms Using Fast Convolution Algorithms

The paper by Cooley and Tukey, 1965, had a major impact on digital signal processing by stimulating the development and wide use of the FFT. Recently several new ideas have been used to compute the DFT which have impacted digital signal processing. In 1968 it was observed by Rader that computation of the DFT could be changed to circular convolution by rearranging the data when N is prime. Now, if given a fast way to do circular convolution, one has a fast DFT method. Winograd showed the minimum number of multiplications for circular convolution of primes and prime power length sequences. He then proposed that these high speed prime power convolutions be "nested" into long transforms to minimize multiplications. The Winograd nested algorithm has been studied and programmed (Silverman, 1977; McClellan and Nawab, 1979; Zohar, 1979) for computing the DFT of complex valued sequences.

An alternative to the Winograd algorithm was proposed by Kolba and Parks and combined the concept of fast convolution with conventional DFT techniques to give another efficient DFT implementation. Kolba and Parks' prime factor algorithm (PFA) uses the same reordering technique as the Winograd Fourier transform algorithm (WFTA). The original PFA (Kolba and Parks, 1978) has been modified (Burrus and Eschenbacher, 1980) so it can transform the same sequence lengths as the WFTA.

This section presents the theory of the WFTA "small-N" algorithms, the data reordering (which is the same for PFA and WFTA), the PFA theory, the real operations count, and the memory array requirements for both PFA and WFTA. Since both algorithms follow a similar development the conversion of a DFT to circular convolution and data reordering are only presented once and apply to both algorithms.

3.5.1 Converting a DFT to Circular Convolution. To convert the DFT expression to a circular convolution the DFT matrix [W] must be "mapped" into the circular convolution matrix $[W_c]$. The mapping between these two matrices, and hence the basis for the WFTA and PFA was developed by Rader in 1968.

Rader showed that if "N is prime, there is some number g, not necessary unique, such that a one-to-one mapping from the integers i = 1,2, ..., N-1 to the integers j=1,2, ..., N-1 is given by:

$$j = ((g^i))_N \tag{3.190}$$

where the notation $((x))_N$ implies x modulo N." The example of N=7 and g=3 using the mapping of Eq (3.190) gives:

i	1	2	3	4	5	6
j	3	2	6	4	5	1

The number g is referred to as a "primitive root" in number theory. The mapping of Eq (3.190) provides the convolution matrix $[W_c]$ from the DFT matrix [W]. Examples of this mapping are extensively treated in the references

(Silverman, 1977; Kolba and Parks, 1977) and are not repeated in this paper.

A brief example of using the results of the convolution matrix is presented to aid in developing the small-N algorithm operations count. Consider the following 3-point DFT written in matrix notation as:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \end{bmatrix} = \begin{bmatrix} W^0 W^0 W^0 \\ W^0 W^1 W^2 \\ W^0 W^2 W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \end{bmatrix} \tag{3.191}$$

where W_3 is assumed and $W_3^4 = W_3^1$. The circular convolution is given by:

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \end{bmatrix} = \begin{bmatrix} W^1 & W^2 \\ W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix} \tag{3.192}$$

which provides $\overline{X}(1)$ and $\overline{X}(2)$. Then the DFT in Eq (3.191) can be rewritten using Eq (3.192) to give:

$$\begin{aligned} X(0) &= W^0(x(0) + x(1) + x(2)) \\ X(1) &= W^0x(0) + \overline{X}(1) \\ X(2) &= W^1x(0) + \overline{X}(2) \end{aligned} \tag{3.193}$$

Using similar techniques to the one presented here, convolution expressions to perform DFTs have been developed for N = 2, 4, 5, 7, 8, 9 and 16.

3.5.2 Reordering the Data Arrays. Implementing the WFTA or the PFA into a useful form involves making long transforms from the short, fast-convolution transforms for 2, 3, 4, 5, 7, 8, 9, and 16. The general idea is "to convert a one-dimensional length $N = M_1 M_2 \ldots M_i$ transform into a i-dimensional transform requiring computation of i shorter length M_k transforms for $k = 1, 2, \ldots, i$." (Kolba and Parks, 1977). The mapping from one-dimension to i-dimensions is based on the Chinese Remainder Theorem which requires relatively prime factors $M_1 M_2 \ldots M_i$. The example for two mutually prime factors given by Kolba and Parks, 1977, is presented here because the mapping is common to both WFTA and PFA.

In the DFT:

$$X(k) = \sum_{N=0}^{N-1} x(n)\, W^{nk} \qquad (3.194)$$

the index n of the input sequence is referred to as the input index, and the index k of the output sequence X(k) is called the output index. Mapping from one-to-two dimensions maps the input index n into a pair of indices (n_1, n_2).

$$n_1 = r_1 n \bmod M_1 \qquad n_1 = 0_1 \ldots, M_1-1 \qquad r_1 = M_2 \bmod M_1$$

$$n_2 = r_2 n \bmod M_2 \qquad n_2 = 0, \ldots, M_2-1 \qquad r_2 = M_1 \bmod M_2$$

The output index is

$$k_1 = k \bmod M_1 \qquad k_1 = 0, \ldots, M_1-1$$

$$k_2 = k \bmod M_2 \qquad k_2 = 0_1 \ldots, M_2-1$$

The inverse mapping from two-to-one dimension for the output index is:

$$k = (s_1k_1 + s_2k_2) \bmod N \qquad (3.195)$$

where

$s_1 \equiv 1 \bmod M_1$ and $s_2 \equiv 0 \bmod M_1$

$s_1 \equiv 0 \bmod M_2$ and $s_1 \equiv 1 \bmod M_2$

While the same inverse mapping in Eq (3.195) could be used for the input index n, it is more convenient (Kolba and Parks, 1977) to use:

$$n = (M_2n_1 + M_1n_2) \bmod N \qquad (3.196)$$

When the mappings in Eqs (3.195) and (3.196) are used the DFT becomes:

$$X(k_1,k_2) = \sum_{n_1=0}^{M_1-1} \sum_{n_2=0}^{M_2-1} x(n_1,n_2)\, W_{M_2}^{n_2k_2}\, W_{M_1}^{n_1k_1} \qquad (3.197)$$

At this point the WFTA and PFA approach the implementation of Eq (3.197) differently as seen below.

3.5.3 The Winograd Fourier Transform. A new algorithm for computing the DFT was proposed by Winograd in July 1975. The WFTA has properties such that the number of real additions remained at the FFT level while the number of real multiplications necessary to evaluate the DFT was reduced (Silverman, 1977). This paper will not derive the "small-N" algorithms. Readers interested in derivation of the WFTA are referred to the articles which extensively treat the topic (Winograd, 1976; Silverman, 1977; Kolba and Parks, 1977; Zohar, 1979).

Winograd's proof started with the N by N matrix with elements:

$$W_N^{ir} = W_N^{ir \bmod N} = Q_N(i,r) \qquad (3.198)$$

which can be decomposed to:

$$Q_N = O_N D_N I_N \qquad (3.199)$$

where I_N is a u by N incidence matrix with values of 0, 1, and -1 only, D_N is a u by u diagonal matrix, and O_N is an N by u incidence matrix (Silverman, 1977). The decomposition of Q_N is possible with large values of u relative to N (i.e., $u=N^2$). Winograd solved the more difficult problem of decomposing $Q_N=O_N D_N I_N$ given an incidence matrix which has dimension u smaller than N^2. Winograd applied field theory to give solutions where u approximately equals N for small values of N, where N = 2, 3, 4, 5, 7, 8, 9, and 16 (Silverman, 1977).

Not only did Winograd prove the minimum multiplication count for the above small-N DFTs but he also proposed a special structure of Eq (3.197) using Eq (3.199). The two dimensional transform in Eq (3.197) may be implemented by first calculating M_1 length M_2 DFTs:

$$y(n_1,k_2) = \sum_{n_2=0}^{M_2-1} x(n_1,n_2)W^{n_2k_2} \qquad (3.200)$$

and then calculating M_2 length M_1 DFTs:

$$X(k_1,k_2) = \sum_{n_1=0}^{M_1-1} y(n_1,k_2)W^{n_1k_1} \qquad (3.201)$$

Using the notation of Eq (3.199) the M_1 short transform can be written in terms of the input additions $i^{(1)}$, output additions $0^{(1)}$, and multiplications $d^{(1)}$. The length M_2 transform uses $i^{(2)}$, $0^{(2)}$, and $d^{(2)}$ (Kolba and Parks, 1977). The Eq (3.200) becomes:

$$y(n_1,k_2) = \sum_{r=0}^{u_2-1} 0^{(2)}_{k_2r}\, d^{(2)}_r \sum_{n_2=0}^{M_2-1} i^{(2)}_{rn_2}\, x(n_1,n_2) \qquad (3.202)$$

$X(k_1,k_2)$ in Eq (3.201) is a length M_1 transform of $y(n_1,k_1)$ which can also be written:

$$X(k_1,k_2) = \sum_{m=0}^{u_1-1} 0^{(1)}_{k_1m}\, d^{(1)}_m \sum_{n_1=0}^{M_1-1} i^{(1)}_{mn_1}\, y(n_1,k_2) \qquad (3.203)$$

Substituting Eq (3.202) into Eq (3.203) gives:

$$X(k_1,k_2) = \sum_{m=0}^{u_1-1} 0^{(1)}_{k_1m}\, d^{(1)}_m \sum_{n_1=0}^{M_1-1} i^{(1)}_{mn_1}$$

$$\times \sum_{r=0}^{u_2-1} 0^{(2)}_{k_2r}\, d^{(2)}_r \sum_{n_2=0}^{M_2-1} i^{(2)}_{rn_2}\, x(n_1,n_2) \qquad (3.204)$$

The order of summation may be interchanged to "nest" the multiplications in the center which gives Eq (3.204) rewritten as:

$$X(k_1,k_2) = \sum_{r=0}^{u_2-1} 0^{(2)}_{k_2r} \sum_{m=0}^{u_1-1} 0^{(1)}_{k_1m}\, d^{(1)}_m\, d^{(2)}_r$$

$$\times \sum_{n_1=0}^{M_1-1} i^{(1)}_{mn_1} \sum_{n_2=0}^{M_2-1} i^{(2)}_{rn_2}\, x(n_1,n_2) \qquad (3.205)$$

Eq (3.205) is the form that was implemented into FORTRAN code (McClellan and Nawab, 1979) and listed in Appendix II.

As an example of the "nesting" structure for the WFTA consider the case of N=3 given in Eqs (3.190) through (3.192). First, let

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \end{bmatrix} = \begin{bmatrix} M_1/2 + M_2/2 \\ M_1/2 + M_2/2 \end{bmatrix} \tag{3.206}$$

then equating Eqs (3.206) and (3.191) gives:

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \end{bmatrix} = \begin{bmatrix} M_1/2 + M_2/2 \\ M_1/2 - M_2/2 \end{bmatrix} = \begin{bmatrix} x(1)W^1 + x(2)W^2 \\ x(1)W^2 + x(2)W^1 \end{bmatrix} \tag{3.207}$$

Substituting,

$$W^1 = \exp(-j2\pi/3) = -1/2-j(\sqrt{3}/2)$$

$$W^2 = \exp(-j4\pi/3) = -1/2+j(\sqrt{3}/2)$$

into Eq (3.207) provides:

$$\begin{aligned} M_1/2 + M_2/2 = {} & -x(1)/2 - j(x(1)\sqrt{3}/2) \\ & -x(2)/2 + j(x(2)\sqrt{3}/2) \end{aligned} \tag{3.208}$$

$$\begin{aligned} M_1/2 - M_2/2 = {} & -x(1)/2 + j(x(1)\sqrt{3}/2) \\ & -x(2)/2 - j(x(2)\sqrt{3}/2) \end{aligned} \tag{3.209}$$

Solving for M_1 and M_2 gives:

$$\begin{aligned} M_1 &= -(1/2)(x(1) + x(2)) \\ M_2 &= -j(\sqrt{3}/2)(x(1)-x(2)) \end{aligned} \tag{3.210}$$

For the algorithm to be used in Winograd's algorithm the multiplications by $W^0=1$ must be accounted for and minimized. This is accomplished by modifying the length 3 DFT to:

$$\begin{aligned} a_1 &= x(1) + x(2) \\ a_2 &= x(1) - x(2) \\ a_3 &= x(0) + a_1 \end{aligned} \tag{3.211}$$

$$\begin{aligned} M_1 &= (-1/2 - 1)a_1 = -(3/2)a_1 \\ M_2 &= -j(\sqrt{3}/2)a_1 \\ M_3 &= W^0 a_3 = a_3 \end{aligned} \tag{3.212}$$

$$\begin{aligned} C_1 &= M_3 + M_1 \\ X(0) &= M_3 \\ X(1) &= C_1 + M_2 \\ X(2) &= C_1 - M_2 \end{aligned} \tag{3.213}$$

Eqs (3.211) through (3.213) result in 2 multiplications, 1 multiplication by W^0, and 6 additions which can now be expressed in the $X = 0 \cdot D \cdot I \cdot x$ notation as:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 1 & -3/2 & 0 \\ 1 & 0 & -j\sqrt{3}/2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ x(2) \end{bmatrix} \tag{3.214}$$

and then rewritten into summations as:

$$X(k) \sum_{r=0}^{u-1} 0_{kr}\, d_r \sum_{n=0}^{N-1} i_{rn} x(n) \tag{3.215}$$

The fast convolution cases for N=2,4,5,7,8,9, and 16 were developed similar to the method used for N=3 above. The explicit equations for these cases provided the small-N operations count shown in Table 3.6 which is used in computing the real operations count as a function of N for the WFTA.

3.5.4 The Prime Factor Algorithm Theory. An alternative to the nested algorithm proposed by Winograd was developed by Kolba and Parks. Because of the algorithms structure it is called the prime factor algorithm (PFA) and uses a modified version of Winograd's high-speed convolution technique.

Converting the DFT to circular convolution and reordering the data arrays for the PFA is identical up through Eq (3.197)

where $W_{M_1} = \exp(-j2\pi/M_1)$,

$W_{M_2} = \exp(-j2\pi/M_2)$, with M_1 and M_2 relatively prime.

The transform in Eq (3.197) may be performed by calculating M_1 length M_2 DFTs:

$$y(n_1,k_2) = \sum_{n_2=0}^{M_2-1} x(n_1,n_2)W^{n_2k_2} \tag{3.216}$$

then calculating M_2 length M_1 DFTs:

$$X(k_1,k_2) = \sum_{n_1=0}^{M_1-1} y(n_1,k_2)W^{n_1k_1} \tag{3.217}$$

The expressions in Eqs (3.216) and (3.217) are implemented as short DFTs instead of "nested" operations as shown in Eq (3.205).

TABLE 3.6

SMALL-N OPERATIONS COUNT FOR WFTA

N	Mult	Mult by W^0	Adds
2	0	2	2
3	2	1	6
4	0	4	8
5	5	1	17
7	8	1	36
8	2	6	26
9	12	1	44
16	10	8	74

For both algorithms structure the small-N equations are the same, only the implementation is different. In the case of the PFA structure the small-N algorithms are modified to permit a "shift operation" instead of a multiplication by 1/2. For the N=3 example Eqs (3.211) through (3.213) are modified to:

$$a_1 = x(1) + x(2)$$
$$a_2 = x(1) - x(2)$$
$$a_3 = x(0) + a_1 \qquad (3.218)$$

$$M_1 = -(1/2)a_1$$
$$M_2 = -j(\sqrt{3}/2)a_2 \qquad (3.219)$$

$$C_1 = x(0) + M_1$$
$$X(0) = a_3$$
$$x(1) = C_1 + M_2$$
$$X(2) = C_1 - M_2 \qquad (3.220)$$

Eqs (3.218) through (3.220) have 1 multiplication, 1 shift (multiplication by 1/2) and 6 additions.

Similar small-N DFTs result for N=2,4,5,7,8,9 and 16 to produce the operations count for PFA small-N algorithms shown in Table 3.7 (Burrus and Eschenbacher, 1980). (Complex valued sequences require the count in Table 3.7 to be doubled.) If the implementation of the PFA does not use "shifts" the multiplication count must be adjusted to reflect the multiplications by 1/2. The original FORTRAN program written (Kolba, 1977) did not include the factor of 16. Later modifications (Burrus and Eschenbacher, 1980)

TABLE 3.7

PFA SMALL-N DFT OPERATIONS COUNT

N	Multiplies	Shifts	Adds
2	0	0	2
3	1	1	6
4	0	0	8
5	4	2	17
7	8	0	36
8	2	0	26
9	8	2	49
16	10	0	74

NOTE: For complex sequences the values in the table must be doubled.

included the factor of 16 which made the PFA capable of transforming the same sequence lengths as the WFTA. It should be noted that neither FORTRAN version implemented the "shifts" which increased the number of real multiplications.

3.5.5 Real Operations for WFTA. To use the WFTA the N length sequence must be factorable into R relatively prime factors $N_1 \; N_2 \; \ldots \; N_R$ where each factor corresponds to one of the Winograd small-N algorithms for 2,3,4,5,7,8, 9 and 16. It has been shown (Silverman, 1977) that the number of real multiplications is a function of the factors of N. To aid in the development of the number of real operations the following terms are defined:

M_r = number of real multiplications in factor N_r

A_r = number of real additions in factor N_r

$N_r = r^{th}$ factor of N

Winograd proved that the D_N matrix is an M_R by M_R diagonal matrix with only 0, 1, or -1 for diagonal entries and 0_N and I_N are N by M_N and M_N by N incidence matrices, respectively. To evaluate the nested multiplications of D_N (Silverman, 1977) requires:

$$\text{NMULT} = M_1 \; M_2 \; \ldots \; M_R \qquad (3.221)$$

which is the real multiplications count for real valued sequences. For complex valued transforms Eq (3.221) must be multiplied by 2.

All previous multiplications counts (Winograd, 1976; Kolba and Parks, 1977; Silverman, 1977) use only Eq (3.221) as the source of real multiplications for the WFTA. The multiplications in Eq (3.221) are all performed by the MULT subroutine in Figure 3.31. Other real multiplications are required in the WFTA for computing the multiplier coefficients and determining the input and output permutation vectors of the INISHL subroutine in Figure 3.31.

The DFT multiplier coefficients are computed in lines 1450-1510 of the WFTA listed in Appendix H and require:

$$\text{real mult} = 3 * \text{NMULT} \qquad (3.222)$$

where N MULT was computed in Eq (3.221). Determining the output permutation vector in lines 2080-2170 requires:

$$\text{real mult} = 4 * N \qquad (3.223)$$

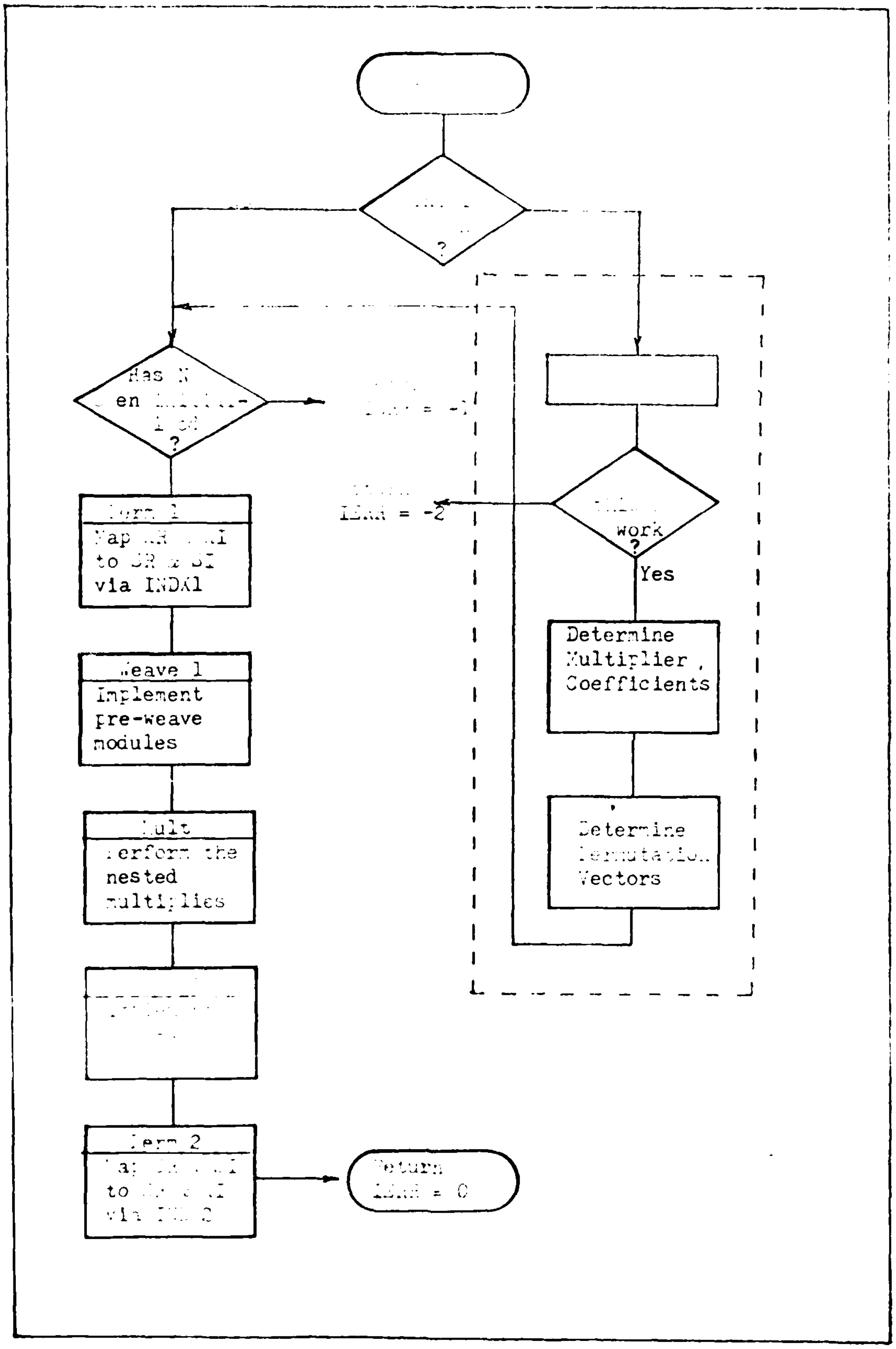
where N is sequence length to be transformed. Combining Eqs (3.222) and (3.223) provides the number of real operations required for initializing the WFTA. Subsequent transforms of the same sequence length do not require initialization. The first complex transform of length N using the WFTA requires:

$$\text{real mult} = 2 * \text{NMULT} + 3 * \text{NMULT} + 4 * N \qquad (3.224)$$

Subsequent complex transforms require:

$$\text{real mult} = 2 * \text{NMULT} \qquad (3.225)$$

Counting the number of real additions is more complicated because the factorization order of N will change the real additions count (Silverman, 1977). For a given factorization of $N = N_1 N_2 \ldots N_R$ the number of real additions



Has N
en
?
IERR = -2
work
?
Yes
Determine Multiplier Coefficients
Determine Permutation Vectors
Perm 1
Map
to
via INDX1
Weave 1
Implement pre-weave modules
Mult
Perform the nested multiplies
Perm 2
to
Return
IERR = 0

Figure 3.31. Flow Control in WFTA Program.

required to evaluate a complex valued sequence can be determined from the INISHL initialization subroutine and the WEAVE1 and WEAVE2 subroutines in Figure 3.31. First the real additions from the "WEAVEs" can be developed by considering the special case of $N = N_1 N_2$. N_1 is defined as the "innermost" factor and N_2 is the "outermost" factor. For two factors of N Silverman has shown the number of real additions to be:

$$A(2) = N_1 A_2 + M_2 A_1 \tag{3.226}$$

(Recall A_2 equal real adds to evaluate factor N_2 and M_2 equal real multiplies to evaluate N_2.) Now consider $N = N_1 N_2 N_3$ where $(N_1 N_2)$ is considered to be the "innermost" factor. The number of real additions becomes:

$$\begin{aligned} A(3) &= (N_1 N_2) A_3 + M_3 A(2) \\ &= N_1 N_2 A_3 + M_3 N_1 A_2 + M_3 M_2 A_1 \end{aligned} \tag{3.227}$$

By iterative substitution the number of additions for $N = N_1 N_2 N_3 N_4$ becomes:

$$\begin{aligned} A(4) &= (N_1 N_2 N_3) A_4 + M \;\; A(3) \\ &= N_1 N_2 N_3 A_4 + M_4 N_1 N_2 A_3 \\ &\quad + M_4 M_3 N_1 A_2 + M_4 M_3 M_2 A_1 \end{aligned} \tag{3.228}$$

Eqs (3.226) through (3.228) are used to write a compact expression for the number of real additions needed in the WEAVE subroutines:

$$A(R) = 2 \left(\sum_{r=1}^{R} \left(\prod_{j=1}^{R-1} N_j \right) (A_r) \left(\prod_{j=R-r+2}^{R} M_j \right) \right) \qquad (3.229)$$

The expression in Eq (3.229) represents only real additions used in WEAVE1 and WEAVE2. Other additions are required by the INISHL initialization subroutine to index the DFT coefficient array and compute the output index vector.

The DFT coefficient array is indexed with a J counter in line 1500 of the FORTRAN WFTA program in Appendix H. This part of the INISHL subroutine requires NMULT real additions. The input index array INDX1 requires another J counter in line 1720 which uses N real additions. The output index array INDX2 uses a J counter in line 2160 which uses N real additions. Also the INDX2 computation requires 8N real additions in line 2120.

Totaling the real additions in the initialization subroutine gives:

$$\text{real adds} = \text{NMULT} + 10\text{N} \qquad (3.330)$$

Adding the results of Eq (3.330) to Eq (3.229) gives the total additions needed to transform an N length sequence for the first time. Subsequent transforms at the same N sequence length requires only the number of adds in Eq (3.229).

The FORTRAN WFTA program written by McClellan and Nawab, 1979, decreased the number of real multiplications for N=9 from 13 to 11 while the number of additions remained constant at 44. Modifying Table 3.6 to reflect the new multiply count for N=9 gives the McClellan and Nawab real operations count for the [illegible] algorithm [illegible] in Table 3.8.

Using Eqs (3.229) and (3.330) with Table 3.8 gives the number of real operations for all permissible WFTA sequence lengths shown in Table 3.9a and b. The columns labeled "REAL MULTI" and "REAL ADDI" represent the operations for the initial transform of length N. The columns labeled "REAL MULT" and "REAL ADD" give the operations count for subsequent transformations of the same sequence length. The number of real operations are plotted as a function of N in Figures 3.32 and 3.33. These graphs demonstrate the large reduction possible after the WFTA has been initialized for an N length sequence.

3.5.6 Memory Requirements for WFTA. The FORTRAN subroutine WFTA listed in Appendix H requires 2348 words of program memory when compiled for the CDC Cyber 74 computer. The memory array requirements are given by:

XR, XI, INDX1, INDX2:	length N
COEF, SR, SI:	length NMULT=M_1 M_2 M_3 M_4 which is the number multiplies required by the factors of N. NMULT is listed in Table 3.9a and b.
CO3, CO4, CO5, CO8, CO16, CDA, CDB, CDC, CDD:	Total of 88

The original version of WFTA dimensioned INDX1, INDX2, COEF, SR, and SI to their maximum possible lengths of 5040, 5040, 10692, 10692, and 10692 respectively. This made the memory

TABLE 3.8

McCLELLAN AND NAWAB'S WFTA

REAL OPERATIONS FOR THE SMALL-N ALGORITHMS

N	M(N)	A(N)
2	2	2
3	3	6
4	4	8
5	6	17
7	9	36
8	8	26
9	11	44
16	18	74

TABLE 3.9a

REAL OPERATIONS AND MEMORY FOR McCLELLAN AND NAWAB WFTA

N	REAL MULTI	REAL MULT	REAL ADDI	REAL ADD	MEMORY ARRAY	IMULT
2	1[illegible]	[illegible]	25	[illegible]	1[illegible]2	2
3	2[illegible]	[illegible]	[illegible]	12	1[illegible]	3
[illegible]	30	[illegible]	[illegible]	1[illegible]	1[illegible]	[illegible]
[illegible]	[illegible]	12	[illegible]	[illegible]	12[illegible]	6
[illegible]	[illegible]	12	1[illegible]2	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	1[illegible]	1[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	72	1[illegible]	1[illegible]	[illegible]2	[illegible]	[illegible]
[illegible]	[illegible]	2[illegible]	1[illegible]	[illegible]	[illegible]	[illegible]1
1[illegible]	1[illegible]	2[illegible]	2[illegible]	[illegible]	[illegible]	[illegible]2
[illegible]	1[illegible]	[illegible]	22[illegible]	[illegible]	1[illegible]2	[illegible]
[illegible]	1[illegible]	[illegible]	[illegible]	1[illegible]2	[illegible]	[illegible]
1[illegible]	[illegible]	3[illegible]	3[illegible]	1[illegible]2	[illegible]2	[illegible]
1[illegible]	[illegible]	3[illegible]	32[illegible]	1[illegible]	2[illegible]	[illegible]
[illegible]	1[illegible]2	[illegible]	[illegible]	2[illegible]2	[illegible]	[illegible]
[illegible]	2[illegible]	[illegible]	[illegible]	2[illegible]	[illegible]	[illegible]
21	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	21[illegible]	[illegible]8	[illegible]15	2[illegible]2	2[illegible]	[illegible]
[illegible]	2[illegible]	[illegible]2	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	72	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]1	1[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]2	[illegible]	[illegible]
[illegible]	[illegible]	1[illegible]	21[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	1[illegible]2	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	1[illegible]	11[illegible]	[illegible]	[illegible]2	[illegible]
[illegible]	[illegible]	1[illegible]	1[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	1[illegible]	[illegible]	[illegible]	[illegible]	[illegible]2
[illegible]	[illegible]	1[illegible]	21[illegible]	1[illegible]	[illegible]	[illegible]
[illegible]	[illegible]2	21[illegible]	2[illegible]	1[illegible]2	[illegible]2	[illegible]
[illegible]	[illegible]	17[illegible]	1[illegible]	11[illegible]2	[illegible]	[illegible]

TABLE 3.9b

REAL OPERATIONS AND MEMORY FOR McCLELLAN AND NAWAB WFTA

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

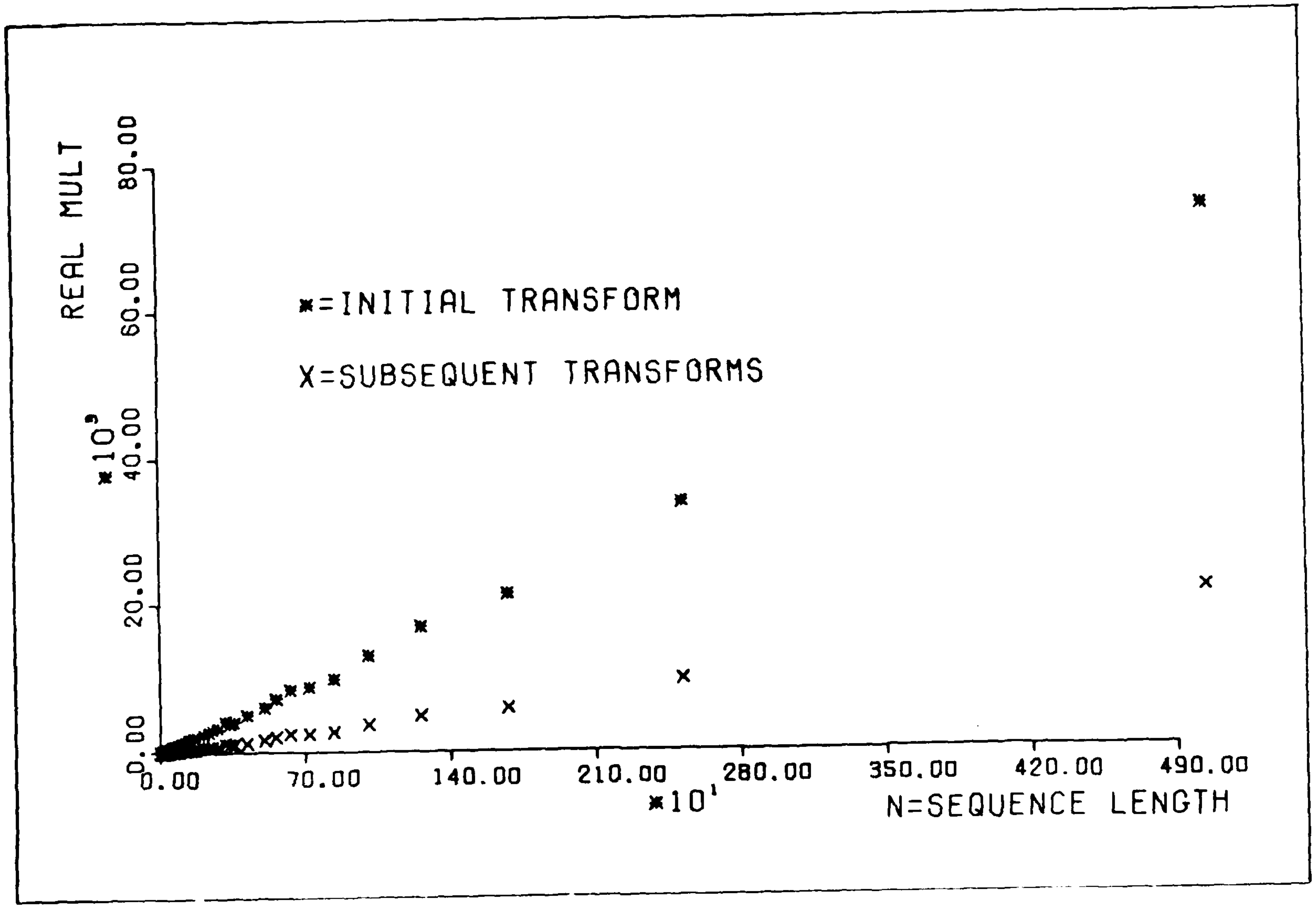


Figure 3.32. Real Multiplications for WFTA.

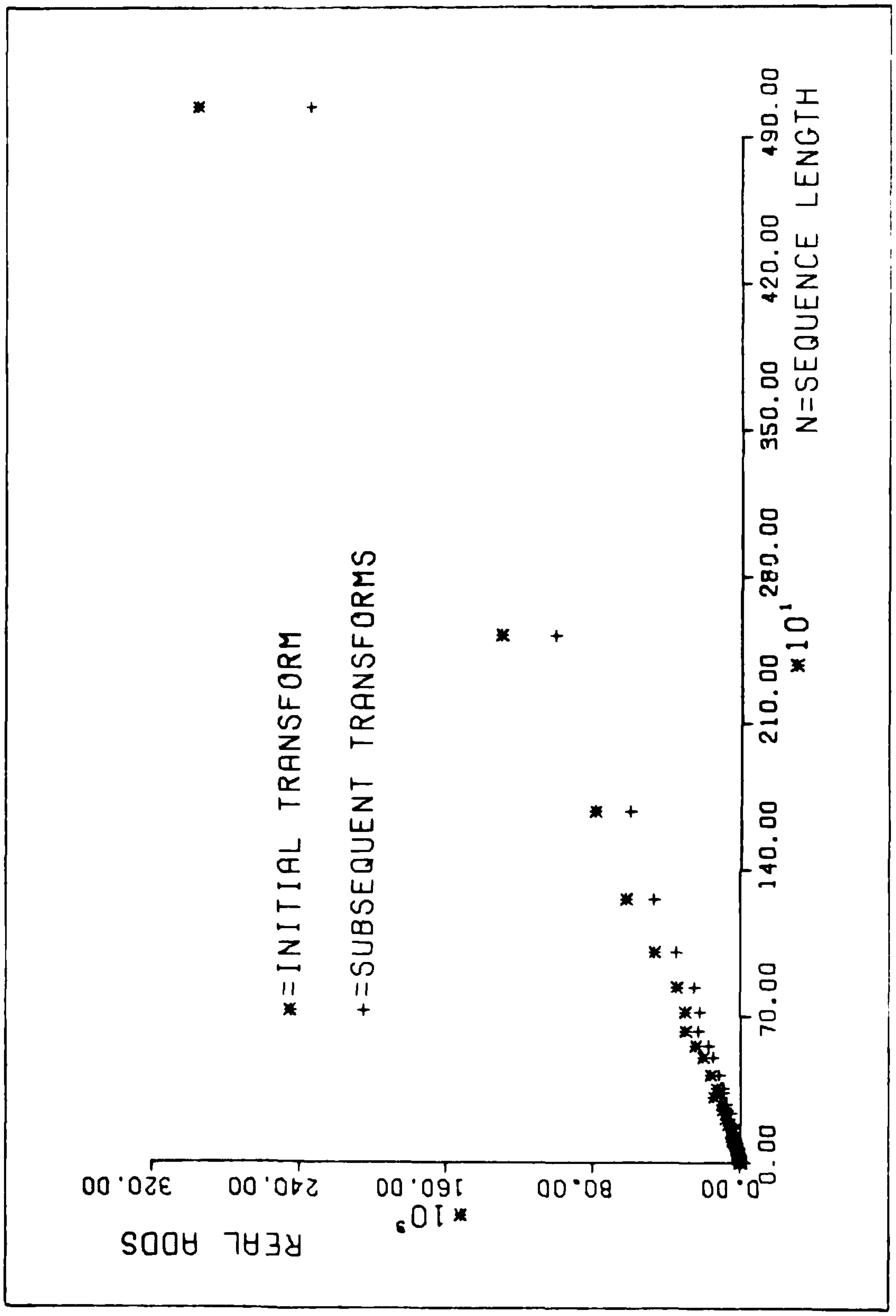


Figure 3.33. Real Additions for WFTA.

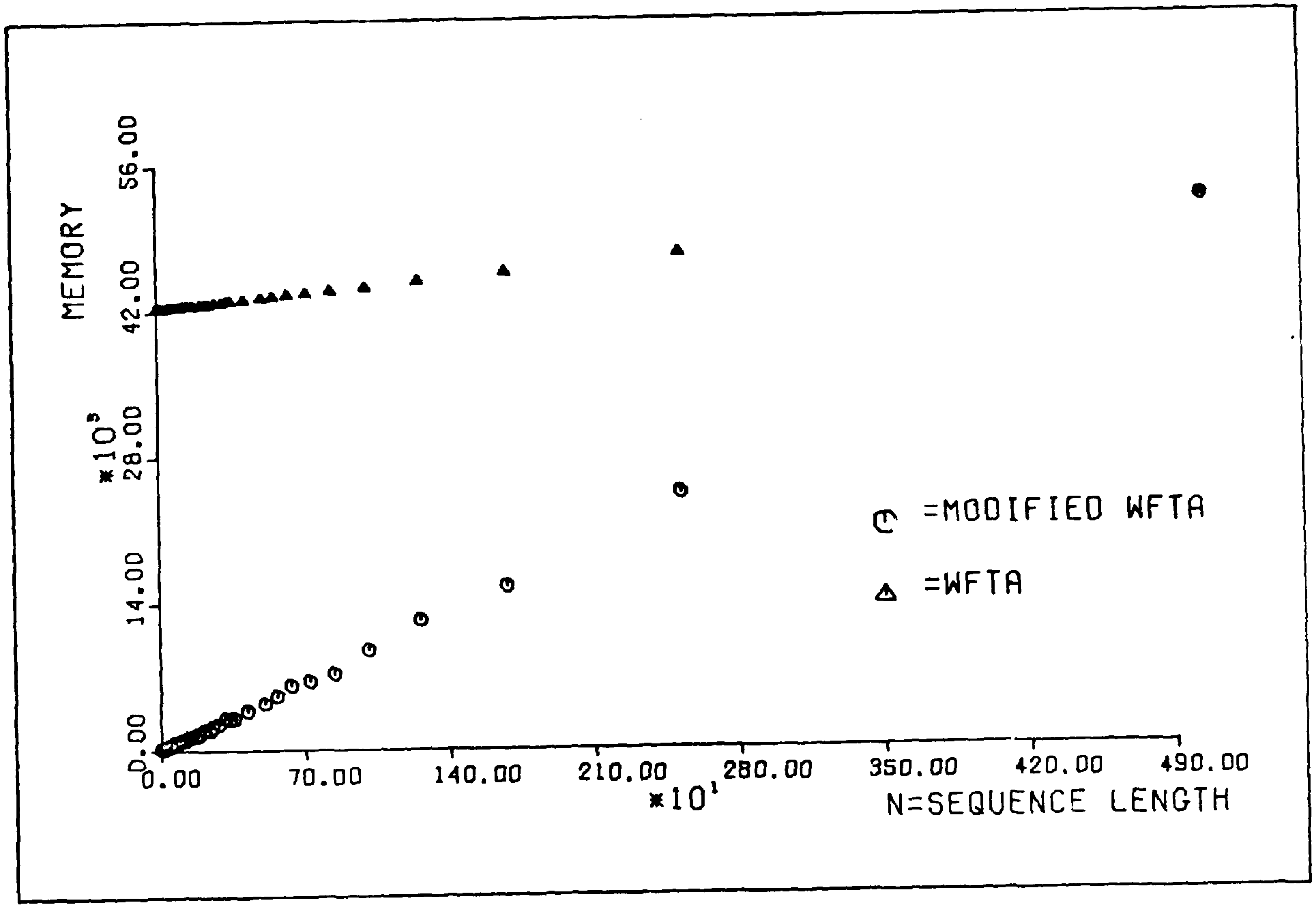


Figure 3.34. Memory Comparison Between Modified and Original WFTA.

array storage very large even for the shortest sequence lengths:

$$\text{memory array} = 2N + 2*5040 + 3*10692 + 88$$

$$= 2N + 42244 \qquad (3.331)$$

The memory arrays INDX1, INDX2, COEF, SR, and SI were variably dimensioned by the author's version of WFTA in Appendix H. This reduced the memory arrays required to:

$$\text{memory array} = 4N + 3\text{NMULT} + 88 \qquad (3.332)$$

The results of Eq (3.332) are listed in Table 3.9a and b for all values of N. A comparison of the memory required by Eqs (3.331) and (3.332) is plotted in Figure 3.34 which shows the drastic savings in memory storage by using the variable dimensions. The "cost" of variable dimensions is more work for the user of WFTA because the dimensions must be passed to the WFTA subroutine using more arguments in the subroutine call. The original version required:

```
CALL WFTA (XR, XI, N, INIT, IERR)
```

The modified WFTA call is:

```
CALL WFTA (N, XR, XI, INIT, IERR, SR, SI, COEF,
           M, INDX1, INDX2)
```

where M = NMULT. The increased complexity of the second call is worth the savings of memory arrays.

3.5.7 Real Operations for the PFA. The real operation sources for the PFA are computed from reordering the data and performing the small-N DFTs. The unscrambling constant which maps the PFA result from arrays X and Y to arrays A and B requires N real additions and no multiplications.

The second source, computing the small-N DFTs using fast convolution, has been proven (Kolba and Park, 1977) for two factors (M_1 M_2) to be:

$$\text{real mult} = 2(M_1 u_2 + M_2 u_1) \quad (3.333)$$

$$\text{real add} = 2(M_1 A_2 + M_2 A_1) \quad (3.334)$$

for three factors (M_1 M_2 M_3):

$$\text{real mult} = 2(M_2M_3u_1 + M_1M_3u_2 + M_1M_2u_3) \quad (3.335)$$

$$\text{real add} = 2(M_2M_3A_1 + M_1M_3A_2 + M_1M_2A_3) \quad (3.336)$$

and for four factors ($M_1M_2M_3M_4$):

$$\text{real mult} = 2(M_2M_3M_4u_1 + M_1M_3M_4u_2 + M_1M_2M_4u_3 + M_1M_2M_3u_4) \quad (3.337)$$

$$\text{real add} = 2(M_2M_3M_4A_1 + M_1M_3M_4A_2 + M_1M_2M_4A_3 + M_1M_2M_3A_4) \quad (3.338)$$

where u_i is the number of multiplications required for M_i and A_i is the number of additions required for M_i. Notice that complex data transforms have been assumed in Eqs (3.333) through (3.338) and the number of multiplications and additions were multiplied by two.

As shown in the PFA theory chapter the small-N algorithms can be implemented by using "shifts" instead of multiplications by 1/2. The FORTRAN programs available do not make use of these shifts. Therefore, the operations count for the PFA small-N DFTs shown in Table 3.7 is modified to produce Table 3.10. Using the results of

TABLE 3.10

PFA SMALL-N DFT OPERATIONS COUNT FOR NO SHIFTS

N	MULT	ADD
2	0	2
3	2	6
4	0	8
5	6	17
7	8	36
8	2	26
9	10	42
16	10	74

Eqs (3.333) through (3.338), the N adds required for the output mapping, and Table 3.10 the number of real multiplications and additions are listed for all permissible N values in Table 3.11a and b. The corresponding graphs in Figures 3.35 and 3.36 show the multiplications and additions as a function of N.

Even though this FORTRAN program did not use a shift to perform multiplication by 1/2, incorporating shifts into the small-N DFTs represents a significant savings of real multiplications. The major benefit would be in small computers where software multiplies are more costly relative to additions. The benefit of performing multiplications by using shifts is given in Table 3.1a and b under the PCT (percentage) column. PCT was calculated by:

$$PCT = ((M-MS)*100)/M \qquad (3.339)$$

where M is the number of multiplications without using shifts and MS is the number using shifts. The percentage savings as a function of N was plotted in Figure 3.37 for all values of N.

3.5.8 Memory Requirements for PFA. The PFA program listed in Appendix I requires 770 words of program memory when compiled for the CDC Cyber 74 computer. The memory array requirements are given by:

X, Y, A, B: length N

The memory array required by PFA is given by:

ry array = 4N

TABLE 3.11a

PFA REAL OPERATIONS AND MEMORY COUNT FOR N≤72

N	REAL MULT	REAL ADDS	MULT W/SHIFT	ROT	MEMORY
2	[illegible]	[illegible]	[illegible]	[illegible]	8
3	[illegible]	1[illegible]	2	[illegible]	12
[illegible]		2[illegible]	[illegible]		1[illegible]
[illegible]	12	30	[illegible]	33	2[illegible]
[illegible]	8	42	[illegible]	[illegible]	2[illegible]
[illegible]	16	70	1[illegible]		2[illegible]
[illegible]	[illegible]	[illegible]	[illegible]		32
9	2[illegible]	83	1[illegible]	2	[illegible]
1[illegible]	2[illegible]	9[illegible]	1[illegible]	33	[illegible]
12	16	1[illegible]	[illegible]	[illegible]	[illegible]
1[illegible]	32	1[illegible]	32	[illegible]	[illegible]
1[illegible]	[illegible]	17[illegible]	3[illegible]	70	[illegible]
16	2	1[illegible]	2[illegible]	[illegible]	[illegible]
18	[illegible]	202	32	2[illegible]	72
2[illegible]	[illegible]	236	32	33	[illegible]
21	7[illegible]	32[illegible]	[illegible]2	1[illegible]	[illegible]
2[illegible]	[illegible]	27[illegible]	32	[illegible]	9[illegible]
28	[illegible]	4[illegible]	[illegible]	[illegible]	112
[illegible]	1[illegible]	[illegible]	[illegible]	[illegible]	12[illegible]
3[illegible]	1[illegible]	673	17[illegible]	1[illegible]	[illegible]
3[illegible]	[illegible]	51[illegible]	[illegible]	[illegible]	[illegible]
4[illegible]	[illegible]	572	[illegible]	[illegible]	[illegible]
[illegible]2	1[illegible]	72[illegible]	12[illegible]	[illegible]	[illegible]
[illegible]	2[illegible]	771	[illegible]2	[illegible]	13[illegible]
[illegible]	[illegible]	[illegible]	[illegible]2	[illegible]	192
[illegible]	[illegible]	99[illegible]	1[illegible]	[illegible]	2[illegible]
[illegible]	[illegible]	9[illegible]	1[illegible]	3[illegible]	[illegible]
[illegible]	[illegible]	12[illegible]	2[illegible]	[illegible]	[illegible]
[illegible]	32[illegible]	[illegible]	2[illegible]	1[illegible]	[illegible]
[illegible]2	[illegible]	1252	1[illegible]	15	[illegible]

TABLE 3.11b

PFA REAL OPERATIONS AND MEMORY COUNT FOR N≥80

N	REAL MULT	REAL ADDS	MULT W/SHIFT	PCT	MEMORY
[illegible]	[illegible]	1304	[illegible]	21	32
[illegible]	[illegible]	162	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	1720	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	2315	[illegible]	[illegible]	[illegible]
112	[illegible]	[illegible]	[illegible]	[illegible]	44
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	17	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	12	[illegible]
[illegible]	692	[illegible]	[illegible]	18	[illegible]
[illegible]	832	[illegible]	[illegible]	26	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	1136	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	15	[illegible]
[illegible]	[illegible]	9637	[illegible]	16	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	13	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	1312	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	13	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	18	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	21	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	22	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

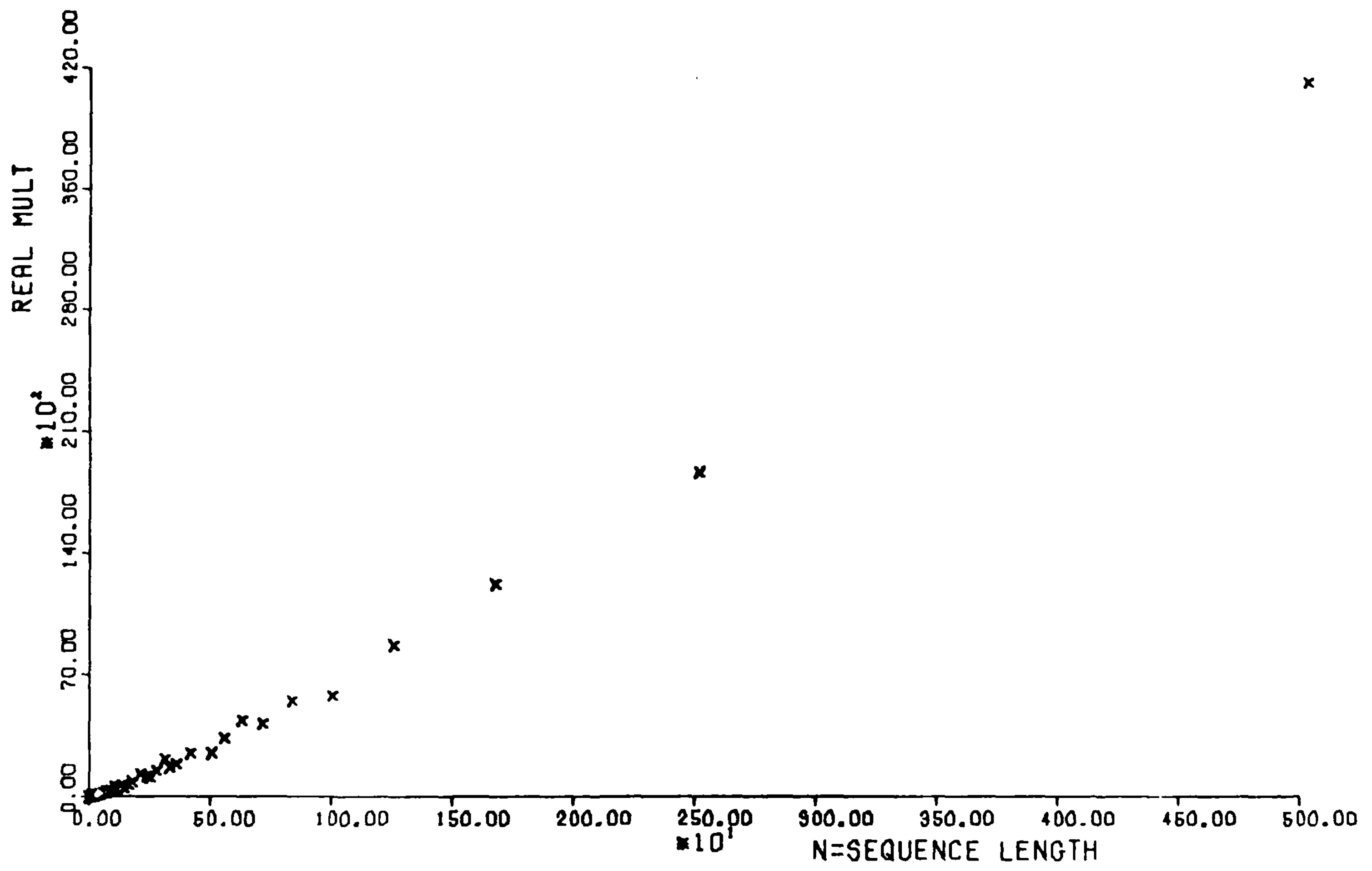


Figure 3.35. Real Multiplications for the PFA.

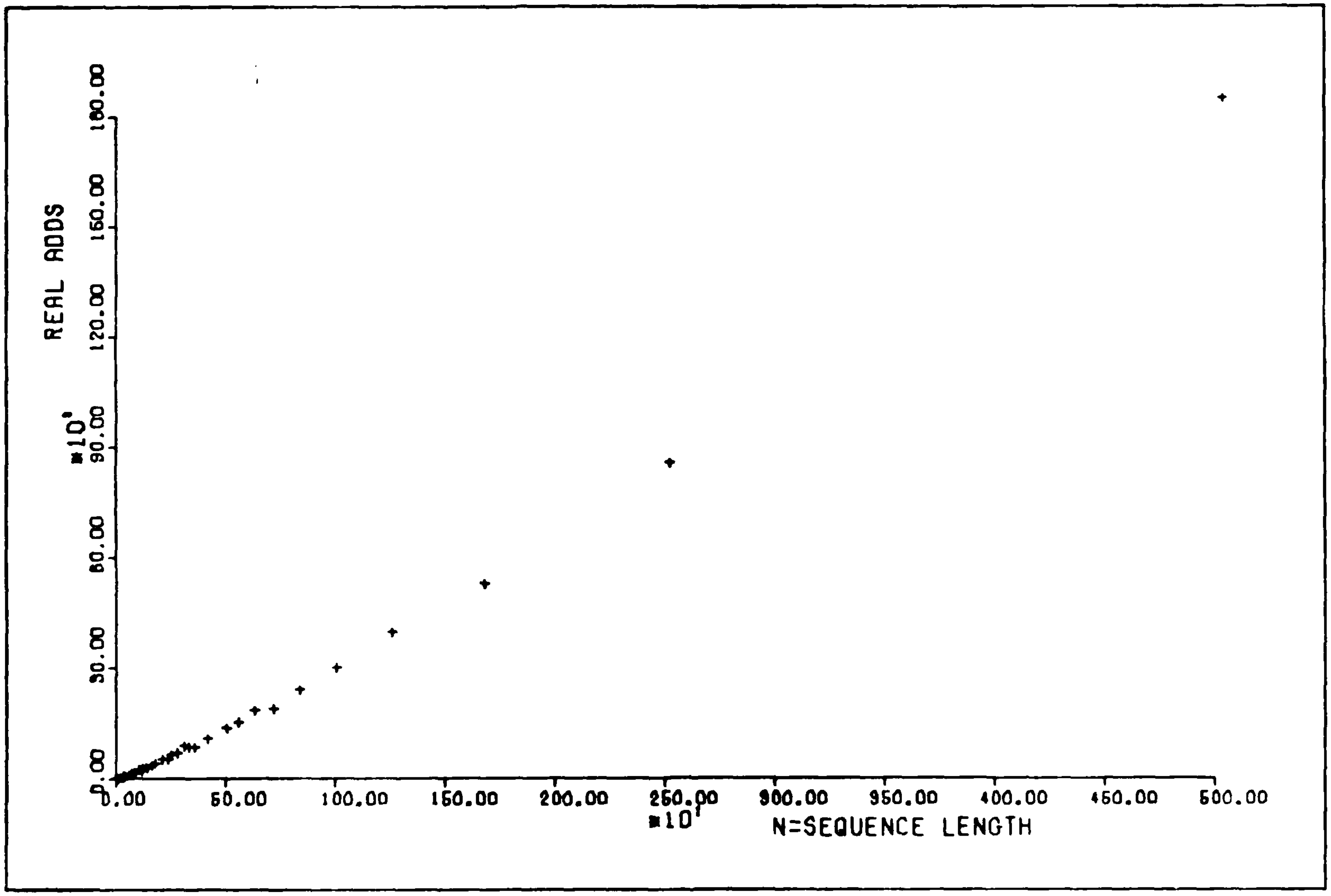


Figure 3.36. Real Additions for the PFA.

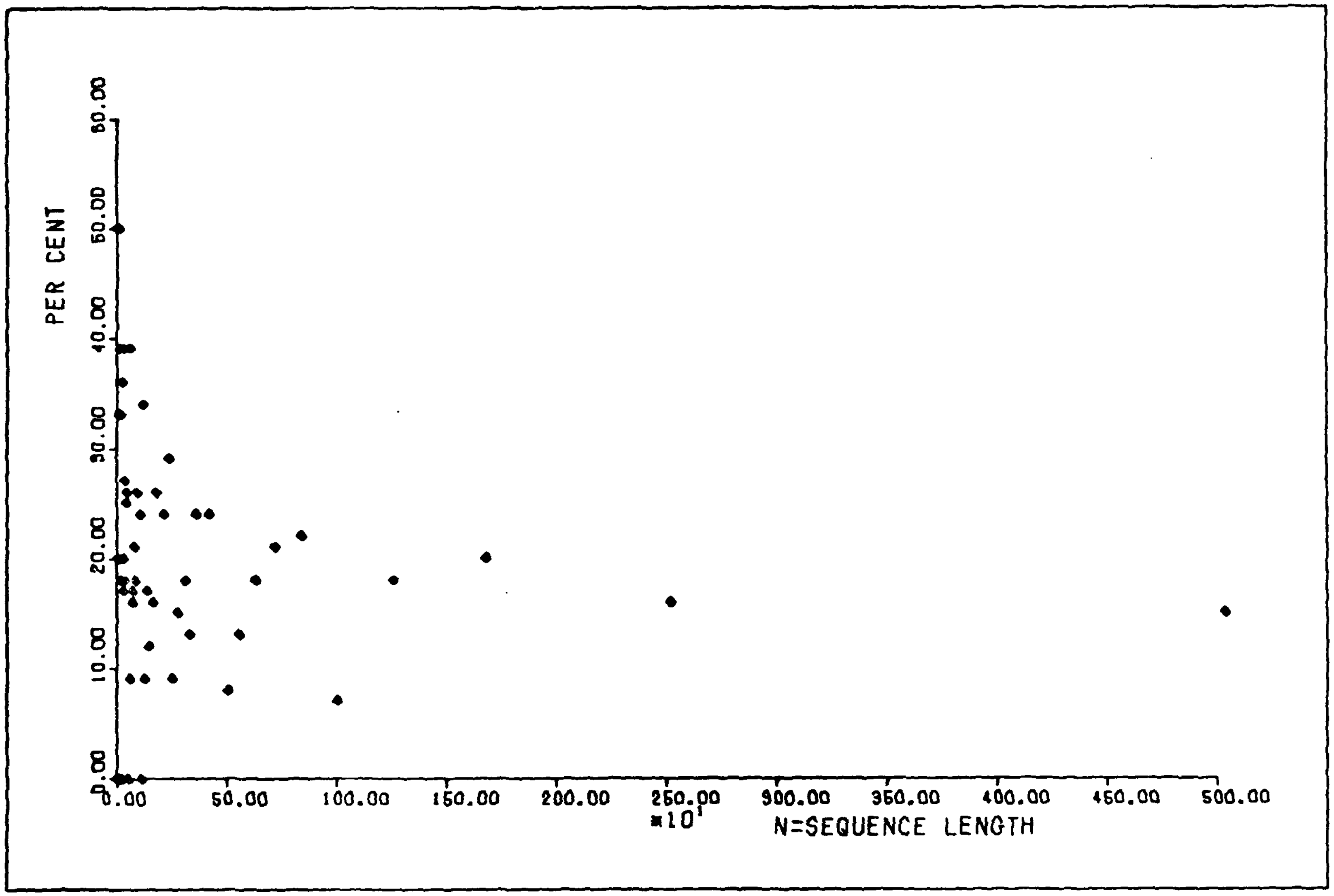


Figure 3.37. Percentage Savings of Multiplications by Using Shifts in PFA.

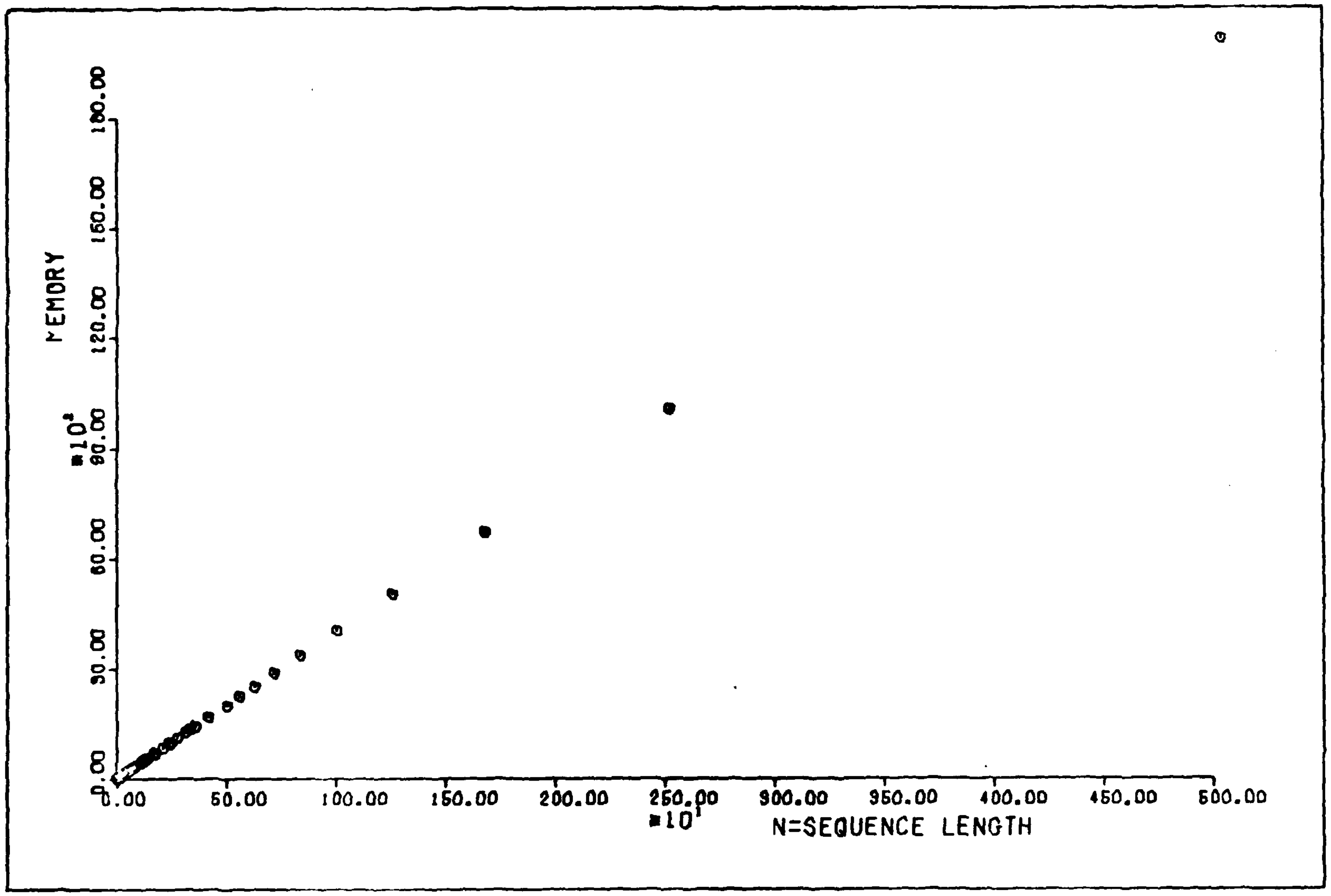


Figure 3.38. Memory Array Required by PFA.

and is listed in Table 3.11a and b and plotted in Figure 3.38.

3.5.9 Summary. Two algorithms which use high-speed convolution techniques have been presented. Both use the convolution for computing small-N DFTs and both require N to be factored into relatively prime factors. This particular factorization used the Chinese Remainder Theorem and the "Sino correspondence" to reorder the data arrays. The theory, structure, and operations count was presented in this section.

IV. Comparison Results of Efficient Discrete Fourier Transforms

4.1 Introduction

Several fixed radix and mixed radix algorithms have been studied and the number of real operations and memory count required have been computed in the preceding sections. The results from these sections are compared and presented here.

Tradeoffs and advantages of fixed radix and mixed radix algorithms are discussed, the justification for selecting Singleton's algorithm over the IMSL and mixed radix FFT is given, tables and graphs comparing the conventional mixed radix FFT with the fast convolution algorithms (WFTA and PFA) are presented and advantages of each are discussed. This chapter concludes with an algorithm which selects the most efficient algorithm based on memory available, machine speed, zeropacking, and sequence length. A flowchart implementation of the algorithm is included.

The timing tests in this section used the Cyber 74 system clock. This clock was accessed using the FORTRAN command SECOND(CP) which provides a timer accurate to .001 seconds. The transforms were all performed using samples from the function $e^{-t}\cos 50\pi t$ which has the magnitude transform shown in Figure 4.1 for N=625.

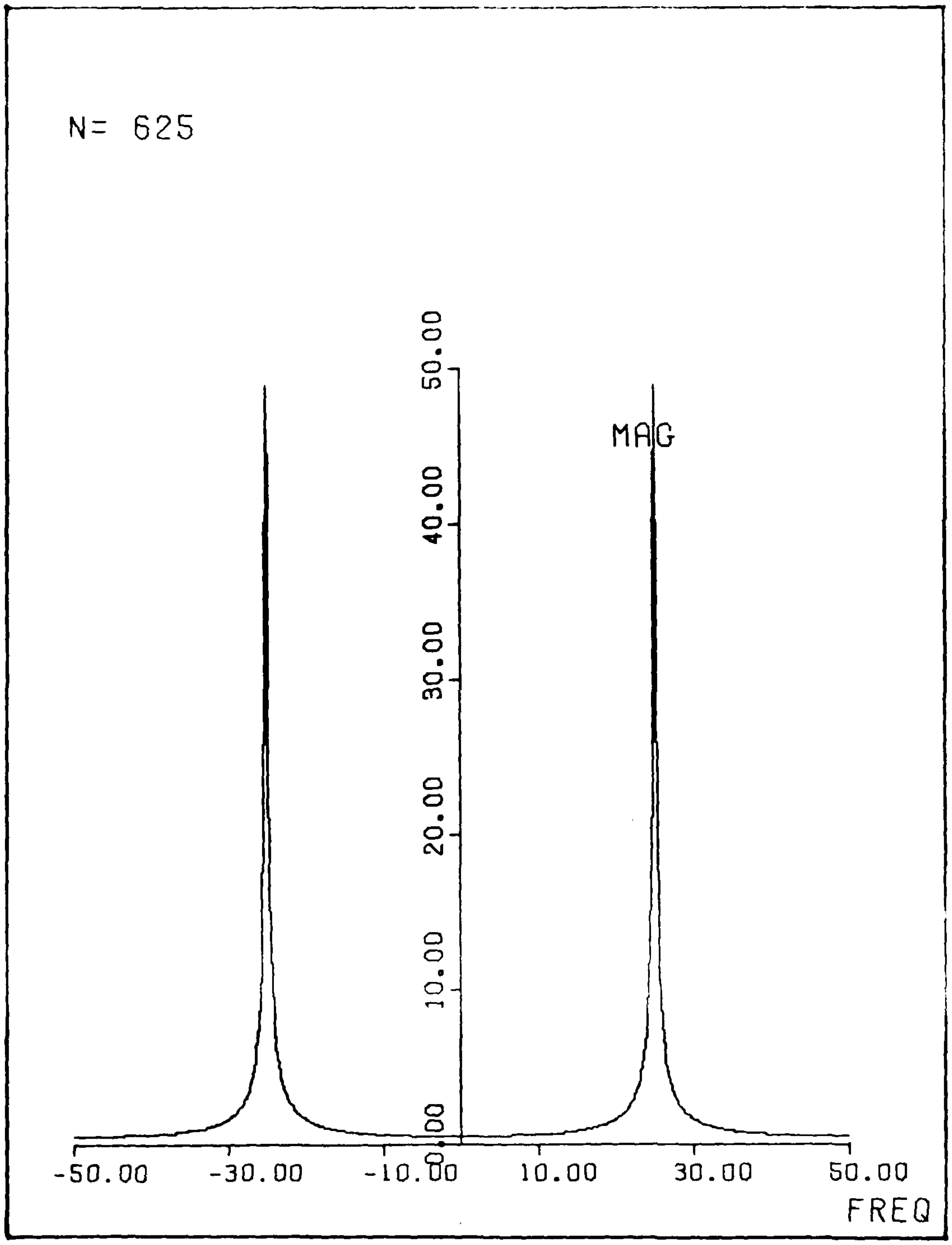


Figure 4.1. Fourier Transform of $e^{-t} \cos 50\pi t$.

The memory comparisons made in this chapter are based on memory array required. The program memory obtained from compilation on the Cyber 74 is not applicable to smaller machines and would not permit valid memory comparisons. The program memory required for the Cyber 74 is given to show the relative sizes of the algorithms.

4.2 Conventional Radix-3 vs R(u) Field Radix-3

In the previous chapter the real operations count for these two radix-3 FFTs was given in Table 3.2. From this table the most efficient radix-3 algorithm can be selected based on machine speed. Validation of this table was performed using the CDC Cyber 74 computer which has a 1.1 multiply-to-add ratio and test data).

With a 1.1 multiply-to-add ratio Table 3.2 indicates that the conventional radix-3 algorithm is more efficient for all sequence lengths shown. The timing results in Table 4.1 verify this conclusion.

4.3 Fixed Radix vs Mixed Radix FFTs

In Sections 3.3 and 3.4 the real operations count and memory requirements developed for the fixed radix and mixed radix FFTs. Using the results from these sections the real operations count and memory requirements are given in Table 4.2 along with results from timing tests conducted on the CDC Cyber 74. This table demonstrates that Singleton's mixed radix FFT (MFFT) minimizes the operations count for factors of 2, 3, and 5 to the level of the fixed radix algorithms.

TABLE 4.1

RADIX-3 TIMING COMPARISON

N	Conventional Radix-3 Time	R(u) Field Radix-3 Time
27	.002	.003
81	.009	.011
243	.026	.034
729	.094	.117
2143	.305	.393

TABLE 4.2

FIXED RADIX (FR) vs MIXED RADIX (MR) FFTs

	Multiplications		Additions		Trig Library		Exponentiation		Memory Array		[illegible]	
N	FR	MR	FR	MR	FR	MR	FR	MR	FR	MR	[illegible]	MR
27	274	313	515	452	1	1	0	0	141	70	[illegible]	.2
32	444	206	542	435	5	1	0	14	64	84	[illegible]	.2
64	1020	504	1278	1056	6	1	0	36	128	148	[illegible]	.4
81	1138	1396	1973	1844	1	1	0	0	358	17[illegible]	[illegible]	.9
125	2154	2448	3027	3072	1	1	0	0	[illegible]	[illegible]	[illegible]	.12
128	[illegible]	1262	2942	2505	7	2	0	66	[illegible]	[illegible]	[illegible]	.11
243	4378	5268	7211	6904	1	3	0	0	1007	[illegible]	[illegible]	[illegible]
256	5116	2850	[illegible]	5683	8	3	0	158	512	[illegible]	[illegible]	[illegible]
512	11260	7988	14846	13802	9	10	0	578	1024	[illegible]	[illegible]	[illegible]
625	14754	16520	[illegible]	20552	1	4	0	0	[illegible]	[illegible]	[illegible]	[illegible]
729	15142	18832	[illegible]	24684	1	10	0	0	[illegible]	[illegible]	[illegible]	[illegible]
1024	24572	14568	[illegible]	28436	10	11	0	652	[illegible]	[illegible]	[illegible]	[illegible]
2048	53244	43366	[illegible]	68127	11	20	0	1242	[illegible]	[illegible]	[illegible]	[illegible]
2187	56866	65376	[illegible]	83718	1	32	0	0	[illegible]	[illegible]	[illegible]	[illegible]

The program memory required by each algorithm is given in Table 4.3. The large size of the MFFT is a result of the extra sections needed to transform any length transform and the extra FORTRAN code required to perform multi-variate transforms. None of the other FFTs are capable of performing multi-variate transforms without a significant amount of additional user programming. Singleton's MFFT can perform up to a tri-variate transform, however, this additional flexibility is a disadvantage on memory limited computers when performing single-variate FFTs.

The fixed radix and mixed radix FFTs are roughly equivalent in efficiency. The fixed radix FFTs offer a memory savings over the MFFT for all radix-2 transform sequence lengths shown in Table 4.2 and some of the radix-3 and 5 transform lengths. The main advantage the MFFT offers is the capability to transform any length sequence N while the fixed radix algorithms are limited to integer powers of 2, 3, and 5.

4.4 Mixed Radix FFT Comparison: IMSL vs Singleton

In Chapter 3 and Appendix G the real operations and memory required for the IMSL and Singleton's mixed radix FFTs were derived as a function of N. Those two algorithms are now compared on the basis of real operations and memory and the best algorithm selected.

TABLE 4.3

PROGRAM MEMORY REQUIRED BY FFTs

FFT	Program Memory
Radix-2	108
Radix-3	301
Radix-5	458
Singleton's Mixed Radix	1100

The expression for real multiplications and additions developed for Singleton's FFT is subtracted from the IMSL FFT expression for real operations to show the extra operations required by IMSL. Recall that both Singleton and IMSL versions of the FFT compute sine and cosine using the difference equation of Section 3.1. Both implement the sine and cosine computation similarly and require the same number of real operations to compute them.

Assuming that N can be factored as:

$$N = 2^r\ 3^s\ 4^t\ 5^u\ p_1^{m1} \ldots p_k^{mk} \tag{4.1}$$

the *difference in real multiplications between IMSL and* Singleton's becomes:

delta multiplies = [IMSL multiplication expression]

- [Singleton multiplication expression]

$$\begin{aligned}
\text{delta multiplies} &= [2rN + 4sN + 3tN + 8 + 32(u)N/5 \\
&+ \sum_{i=1}^{k} (2(p_i-1) + 4(mi)N(p_i-1)/p_i \\
&+ (mi)N(p_i-1)^2/p_i) - 4N-1) + KMULT] \\
&- [2rN + 4sN + 3tN + 32uN/5 \\
&+ \sum_{i=1}^{k} (2(p_i-1) + (mi)N(p_i-1)^2/p_i \\
&+ 4(mi)N(p_i-1)/p_i) - 4(N-1) + KMULT] \\
&= 8
\end{aligned} \tag{4.2}$$

For large values of N the difference in multiplications is negligible.

The difference in real additions is derived from:

$$\text{delta adds} = [\text{IMSL addition expression}] - [\text{Singleton addition expression}]$$

$$\begin{aligned}
\text{delta adds} = {} & [3rN + 6sN + 15tN/2 + 4 + 48(u)N/5 \\
& + \sum_{i=1}^{k} ((p_i-1) + 8(mi)N(p_i-1)/p_i \\
& + N(mi)(p_i-1)^2/p_i) - 2(N-1) + KADD] \\
& - [3rN + 16sN/3 + 11tN/2 + 8uN \\
& + \sum_{i=1}^{k} ((p_i-1) + 7N(mi)(p_i-1)/p_i \\
& + (mi)N(p_i-1)^2/p_i) - 2(N-1) + KADD] \\
= {} & 2sN/3 + 2tN + 8uN/5 + 4 \\
& + N(p_i-1)/p_i
\end{aligned} \tag{4.3}$$

The results from Eqs (4.2) and (4.3) demonstrate that the IMSL has approximately the same number of real multiplications but requires significantly more additions than Singleton's mixed radix algorithm. Based on these results and because the data reordering for the two subroutines is the same, the Singleton FFT is the most efficient of the two subroutines. This conclusion was confirmed by timing tests on the CDC Cyber 74 computer at AFIT. The results are shown in Table 4.4 for selected sequence lengths.

The memory array required for each of the algorithms was derived in the preceding chapter. Those results are now compared for N less than 200 and the percentage of array memory saved by Singleton's FFT over the IMSL FFT was plotted in Figure 4.2 using the equation:

TABLE 4.4

TIMING RESULTS FOR IMSL AND SINGLETON FFTs

N	IMSL Time (sec)	Singleton Time (sec)
60	.010	.008
120	.018	.014
125	.019	.012
128	.013	.011
210	.039	.036
243	.031	.031
256	.028	.021
315	.054	.052
420	.081	.072
504	.090	.082
625	.128	.076
729	.107	.107
840	.163	.150
1008	.151	.157
1024	.126	.092
1250	.275	.158
1260	.268	.231
2048	.269	.224
2187	.366	.364
2520	.565	.495

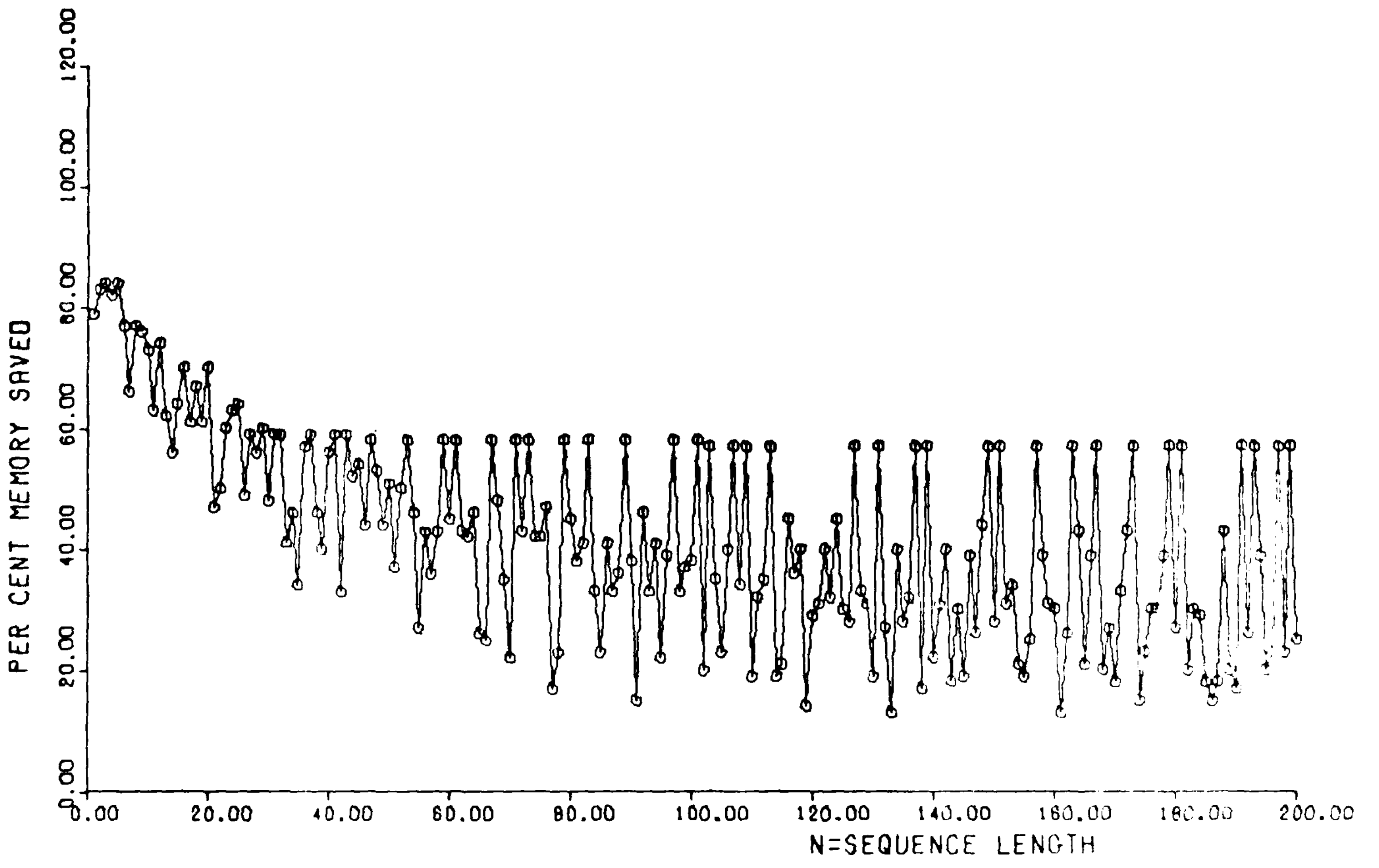


Figure 4.2. Memory Array Saved Using Singleton's Instead of IMSL's FFT.

% savings = (MEMCC − MEMSNG) · 100/MEMCC (4.4)

where MEMCC = IMSL array memory

MEMSNG = Singleton's array memory

From the plot it is evident that Singleton's algorithm uses less memory than the IMSL program. The "flat" portion of the curve approaches 57% which can be verified by examination of Eqs (3.172) through (3.178) for N a prime number. This number represents the memory savings at the points where N is prime.

The values of M, K, KB, and JK used to compute the IWORD constant in Eq (3.173) are M=1, K=N, KB=N-2 and JK=1.

$$\text{IWORD} = 3 \cdot \text{M} + 3 + \text{MAX}\,(4 \cdot \text{M} + 7 + 6 \cdot \text{K}, \text{KB} + 1 + 2 \cdot \text{JK}) \tag{4.5}$$

$$\text{IWORD} = 3 + 3 + \text{MAX}\,(6\text{N} + 11, \text{N} + 1) \tag{4.6}$$

$$\text{IWORD} = 6 \cdot \text{N} + 17 \tag{4.7}$$

Now the memory for IMSL given that N is prime becomes:

$$\text{MEMCC} = 2 \cdot \text{N} + 2(6 \cdot \text{N} + 17) \tag{4.8}$$

$$\text{MEMCC} = 14 \cdot \text{N} + 34 \tag{4.9}$$

The array memory required by Singleton's FFT is based on the values NP and KD. NP is dimensioned to one less than the product of the square free factors of N or if at most one square free factors is present, MP can be dimensioned to M+1 where M is the number of prime factors in N. KD is the size of arrays AT, BT, CK, and SK where KD equals the largest prime factor in N. Using these results the expression for array memory where N is prime becomes:

$$\text{MEMSNG} = 2 \cdot \text{N} + 4 \cdot \text{KD} + \text{NP} \tag{4.10}$$

Substituting for NP and KD this equation is:

$$MEMSNG = 2 \cdot N + 4 \cdot N + 2 \qquad (4.11)$$

$$MEMSNG = 6 \cdot N + 2 \qquad (4.12)$$

Substituting Eqs (4.9) and (4.12) into the percentage expression in Eq (4.4) is seen to approach approximately 57%:

$$\%\ savings = ((14 \cdot N + 34) - (6 \cdot N + 2)) \cdot 100/(14 \cdot N + 34) \qquad (4.13)$$

$$\%\ savings = (8 \cdot N + 36) \cdot 100/(14N + 34) \qquad (4.14)$$

As N gets large Eq (4.14) becomes:

$$\%\ savings \doteq 800N/14N \doteq 57\% \qquad (4.15)$$

which corresponds to the results shown by Figure 4.1.

The memory array must be added to the program memory to determine the size of the program. The program memory required by each algorithm was determined by compiling each algorithm for the CDC Cyber 74. The IMSL FFT used 1061 words and the Singleton FFT used 1100 words. The larger size of the Singleton FFT relative to the IMSL version is because of the extra FORTRAN code needed to perform multi-variate FFTs. These program memory figures are only applicable for the FORTRAN compiler used here at AFIT, however, they do provide a relative measure of the program memory size. Singleton's program requires about 3.7% more program memory.

The results for real operations count and memory required show that Singleton's mixed radix FFT is superior to the IMSL algorithm. For this reason Singleton's algorithm was chosen as the best conventional FFT subroutine available for comparison to the WFTA and PFA in the following sections.

4.5 Conventional vs Fast Convolution Mixed Radix FFTs

Singleton's algorithm (MFFT) is referred to as a "conventional" FFT because it uses the Cooley-Tukey decimation and reordering of the data array. The WFTA and PFA use Winograd's small-N fast convolution algorithms to perform the DFT. The operation and memory array counts are presented in Figures 4.3 and 4.4 and Tables 4.5a and b. as a function of N for comparison of the three algorithms. These tables and plots illustrate the advantages and disadvantages of each algorithm and are used along with the fixed radix results in Table 4.2 to select the most efficient algorithm for a particular sequence length and machine capability (size and speed).

The tables and plots refer to the algorithms as MFFT (Singleton), WFTA (Winograd), and PFA (Kolba-Parks). The PFA used for operation counts and memory comparisons is the one described by Burrus and Eschenbacher which includes prime power factors of 2,3,4,5,7,8,9 and 16. The FORTRAN coded program for PFA was obtained from C. S. Burrus of Rice University and does not make use of "shifts" for multiplications by 1/2. Both the WFTA and MFFT FORTRAN programs were obtained from the IEEE Press "Programs for Digital Signal Processing".

The memory comparison was based on memory array only and did not include program memory. This was done because the program memory changes based on machine word length. The program memory required for the Cyber 74 is given for each algorithm so the relative size can be compared.

4.5.1 Real Operations Count. The mixed radix MFFT written by Singleton includes special sections for factors of 2, 3, 4, and 5 as well as a general section for odd prime factors which permits the transformation of any positive integer N length sequence. Because of the special sections the operations count is less for an N which is highly factorable by 2, 3, 4, or 5 instead of higher prime powers. Figure 4.3 and 4.4 demonstrate the efficiency of Singleton's MFFT relative to the radix-2 complex transform multiplications and additions count of $2N \log_2 N$ and $3N \log_2 N$ respectively (Winograd, 1976). The MFFT operations count shown in Figures 4.3a,b and 4.4a,b are for N factorable by 2, 3, 4, or 5 combinations thereof. The WFTA and PFA counts are shown for all 59 sequence lengths which they can transform. Recall from Section 3.4 and 3.5 that WFTA and PFA sequence lengths are limited by the data reordering algorithm used by the WFTA and PFA. These figures also reflect the WFTA "post-initialization" operations count. As shown in Section 3.5 the post-initialization count is significantly less than the number of operations required for the initial transform of length N.

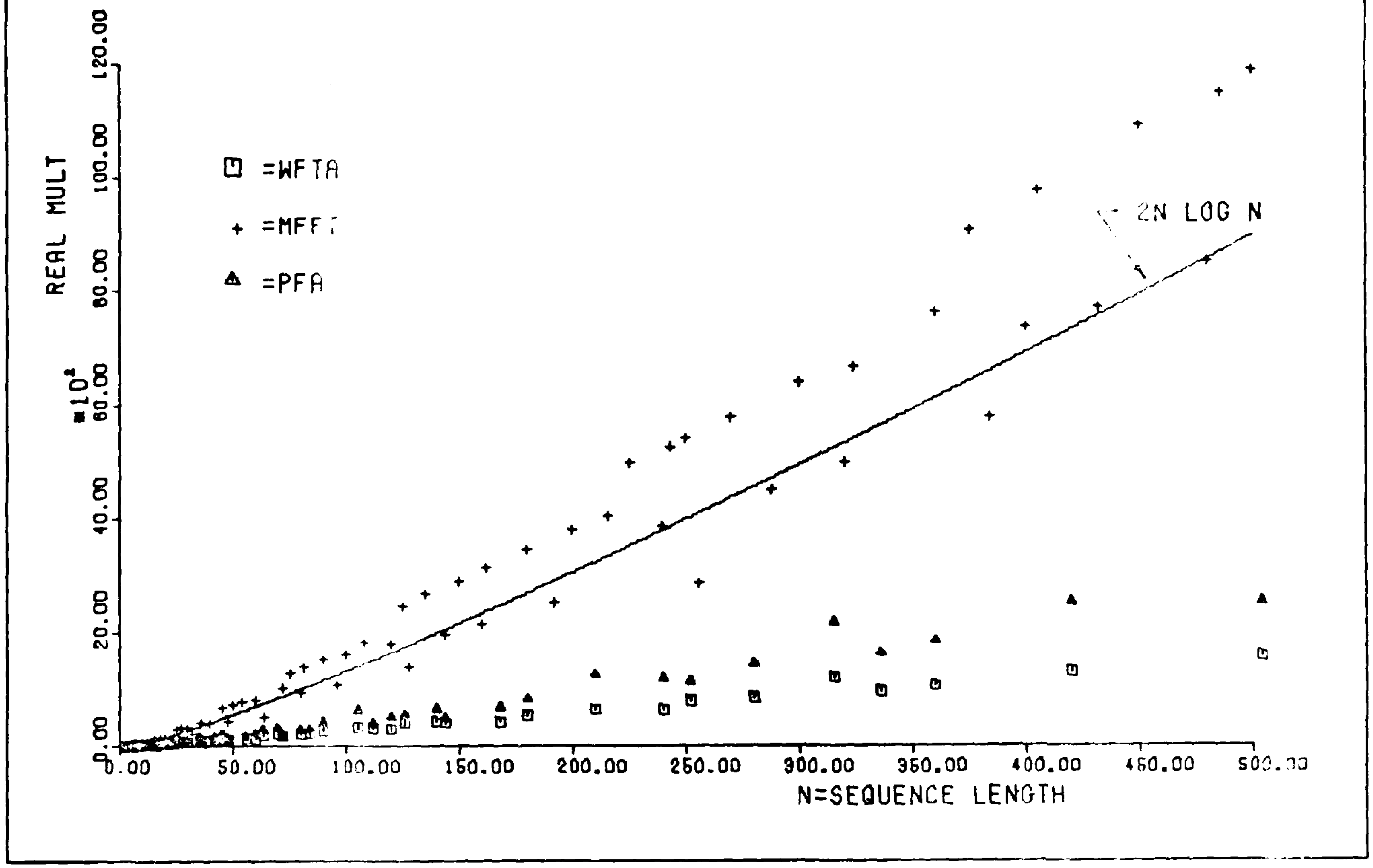


Figure 4.3a. Real Multiplication Comparison for PFA, WFTA, and MFFT (N≤500).

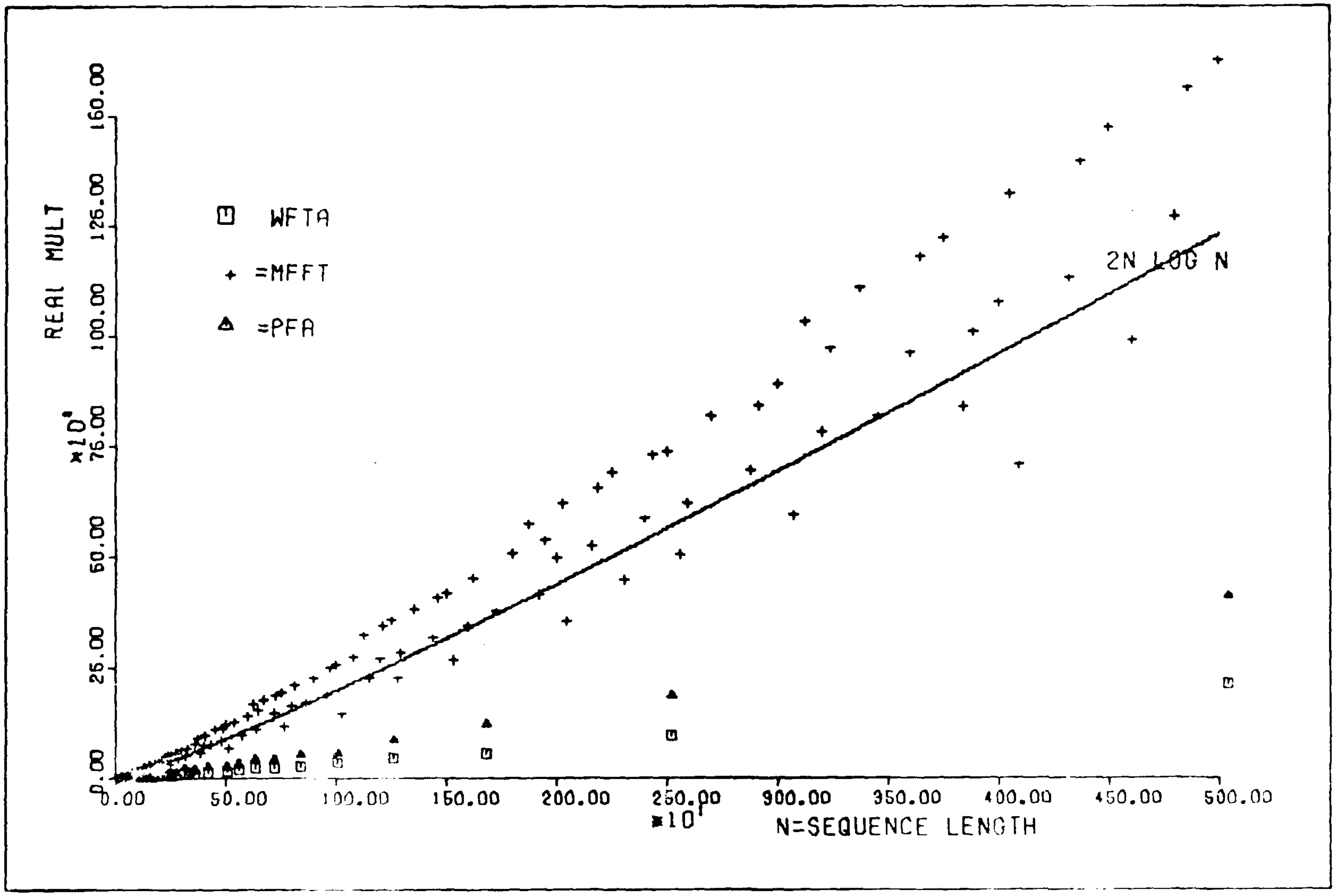


Figure 4.3b. Real Multiplication Comparison for PFA, WFTA, and MFFT (N 5000).

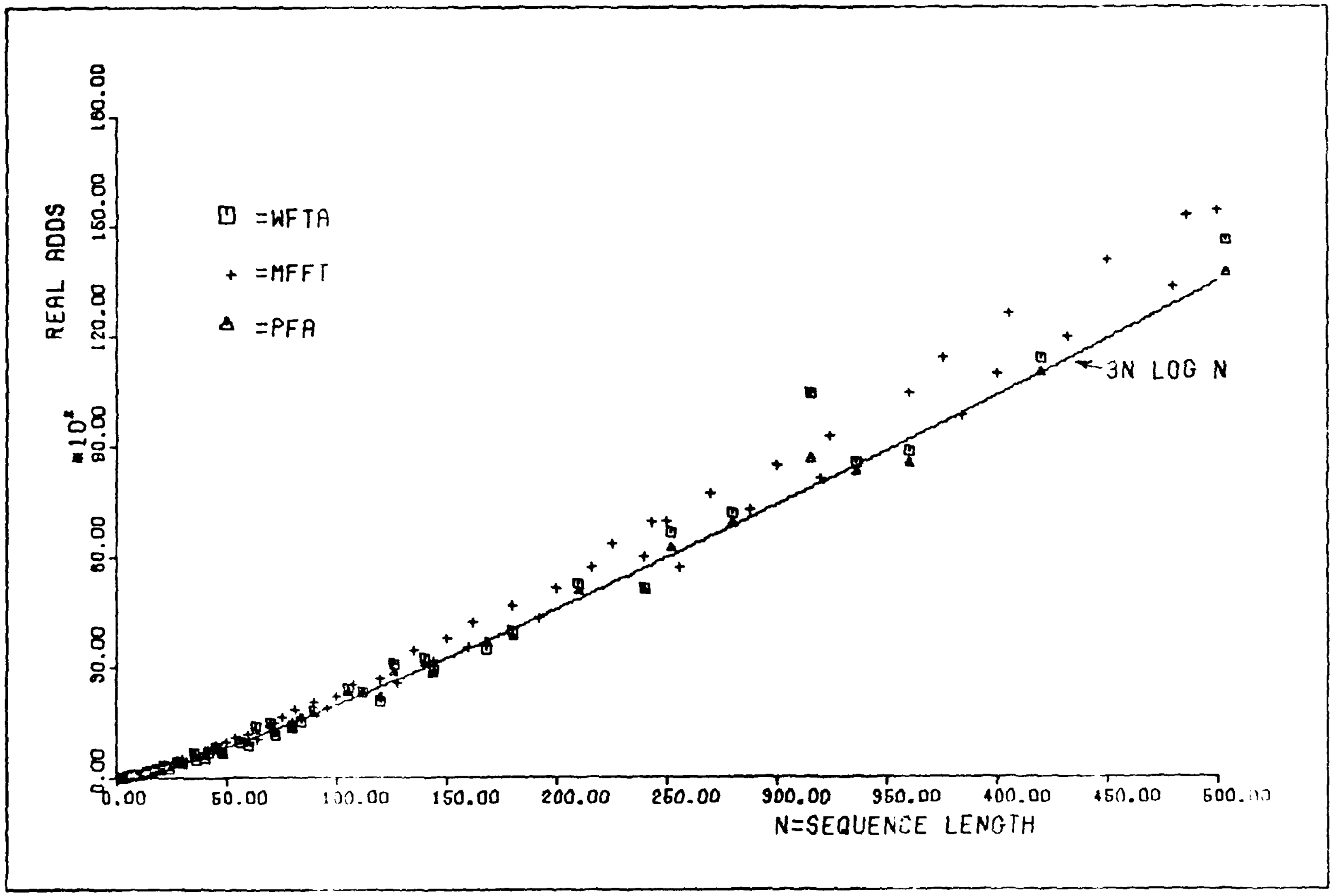


Figure 4.4a. Real Addition Comparison for PFA, WFTA, and MFFT (N≤500).

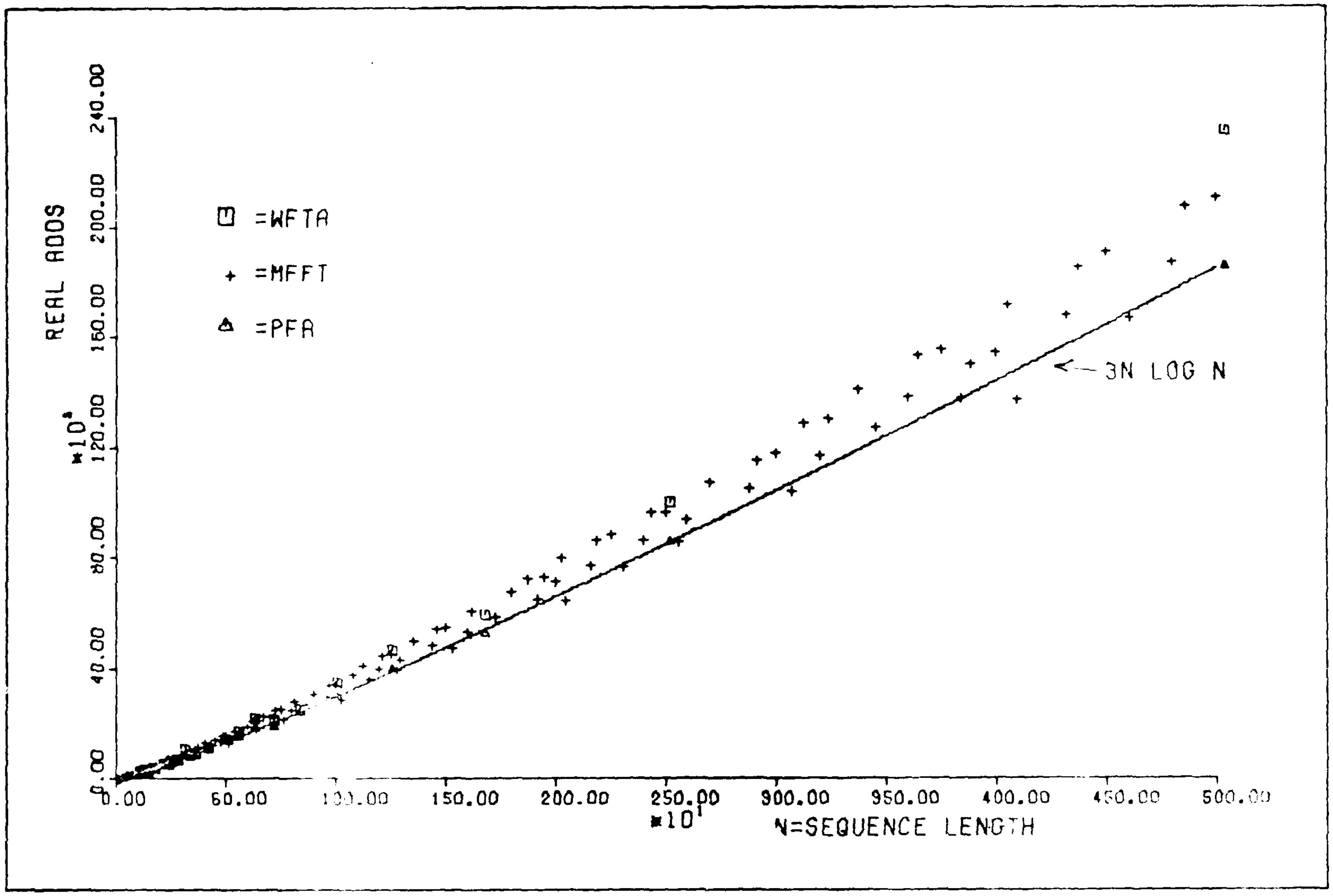


Figure 4.4b. Real Addition Comparison for PFA, WFTA, and MFFT (N=5000).

TABLE 4.5a

OPERATIONS AND MEMORY ARRAY COMPARISON FOR MFFT, WFTA, AND PFA

	REAL MULT			REAL ADDS			MEMORY ARRAY		
N	MFFT	WFTA	PFA	MFFT	WFTA	PFA	MFFT	WFTA	PFA
2	0	4	0	4	4	6	10	102	8
3	4	6	4	12	12	15	12	109	12
4	8	8	0	15	16	20	14	116	16
5	16	12	12	32	34	39	15	126	20
6	16	12	8	40	36	42	22	130	24
7	40	18	15	72	72	73	44	143	28
8	24	16	4	62	52	60	23	144	32
9	60	22	20	92	88	93	24	157	36
10	52	24	24	96	88	98	34	164	40
12	68	24	15	126	96	108	31	172	48
14	115	35	32	186	172	185	70	198	56
15	144	36	55	200	152	177	49	202	50
16	86	36	20	187	148	154	47	205	64
18	158	44	40	248	212	222	42	226	72
20	154	48	45	265	216	235	47	240	80
21	254	54	75	356	310	321	91	253	84
24	184	48	44	330	252	276	58	256	95
28	334	72	54	406	400	428	87	306	112
30	324	72	112	684	384	414	94	316	120
35	516	108	154	676	666	573	133	390	140
36	434	88	90	510	496	516	83	364	144
40	438	96	115	662	532	572	94	332	150
42	[illegible]	153	152	822	584	725	154	418	168
45	472	132	233	580	814	771	97	465	180
48	434	108	124	767	559	584	173	442	192
56	728	144	155	1112	940	335	154	528	224
60	804	144	224	1174	688	348	133	544	240
63	1096	198	234	1440	1408	1229	157	637	252
70	1104	216	323	1550	1472	1475	238	692	280
72	1024	176	135	1482	1172	1212	152	546	286

TABLE 4.5b

OPERATIONS AND MEMORY ARRAY COMPARISON FOR MFFT, WFTA, AND PFA

	REAL MULT			REAL ADDS			MEMORY ARRAY		
N	MFFT	WFTA	PFA	MFFT	WFTA	PFA	MFFT	WFTA	PFA
80	922	216	232	1527	1352	1354	157	732	320
84	1728	216	334	1920	1536	1520	217	748	336
90	1524	264	415	2044	1358	1722	134	844	360
105	2114	324	532	2700	2418	2319	343	934	420
112	1550	324	395	2497	2332	2300	255	1022	448
120	1734	288	503	2590	2076	2195	274	1090	480
126	2424	396	553	3266	3068	2350	294	1166	504
140	2524	432	655	3544	3224	3192	343	1296	560
144	1953	390	[illegible]	3179	2916	2320	295	1258	576
168	2830	432	632	4304	3492	3563	406	1463	672
180	3448	523	332	4533	3976	3334	356	1600	720
210	4298	548	1254	5590	5256	5158	658	1300	840
240	3849	643	1135	5951	5136	5052	499	2028	950
252	5392	792	1135	7326	6640	6204	536	2264	1008
280	5442	864	1452	7592	7148	6334	656	2504	1120
315	7572	1168	2175	9848	10496	8537	693	3130	1260
336	6134	972	1535	9347	8598	8244	721	2890	1344
360	7368	1096	1344	10446	8652	8538	734	3112	1440
420	9434	1296	2523	13076	11352	10355	973	3712	1680
504	11732	1584	2524	15156	14540	13568	1051	4480	2016
560	11510	1944	3324	17089	17168	15378	1183	5244	2240
630	13356	2376	4352	21526	22072	18534	1358	6172	2520
720	14530	2376	4223	21767	21312	19936	1448	6532	2880
840	19540	2592	5475	23378	24814	24012	1918	7336	3360
1008	22520	3564	5884	33531	34668	30178	2049	9455	4032
1260	35236	4752	8724	45868	43664	33538	2533	12256	5040
1680	41550	5832	12212	59565	59054	52544	3493	15556	6720
2520	75058	9504	13553	101146	99528	85475	5133	24424	10080
5040	145252	21384	41115	203737	233928	134312	17143	52324	20160

As a demonstration of the improvement in the post-initiali-zation WFTA consider the timing results of Table 4.6. The data presented here was collected by timing the individual subroutines (INISHL, PERM 1, WEAVE 1, MULT, WEAVE 2, PERM 2) in the WFTA for different sequence lengths and then dividing the time required for each subroutine by the total time for all of the subroutines. Comparing the MFFT and PFA against the post-initialized WFTA is assumed to be valid because most applications of DFTs involve the repeated transform of N length sequences.

A point by point comparison of MFFT, WFTA, and PFA real operations is presented in Table 4.7. The sequence lengths in these tables represent the only lengths permissible for both PFA and WFTA, whereas the mixed radix MFFT can transform any sequence length. The operations count presented in Tables 4.2, 4.7 with a computer's multiply and add speed can predict the most efficient (fastest)DFT technique for that particular computer.

Using the multiply and add speeds determined for the CDC Cyber 74 (see Appendix J) as 1.9×10^{-6} seconds and 1.7×10^{-6} seconds, respectively, the algorithms execution speeds were predicted from the operations count in Tables 3.9 and 4.7. The predicted execution speeds do not account for all of the actual execution time measured as shown in Figure 4.5. The extra time which was not predicted by the real operations count comes from array indexing and data reordering needed in all of

TABLE 4.6

TIMING RESULTS FROM THE WFTA SUBROUTINES

N	INISHL	PERM 1	WEAVE 1	MULT	WEAVE 2	PERM 2
315	48.0%	7.5%	16.3%	4.5%	16.3%	7.4%
360	47.0%	5.9%	15.7%	5.9%	21.6%	3.9%
630	43.9%	5.6%	18.7%	5.6%	21.5%	4.7%
720	44.0%	3.5%	20.0%	6.1%	22.8%	3.6%
840	34.5%	5.5%	23.6%	6.4%	23.6%	6.4%
1008	48.0%	1.7%	19.2%	6.2%	21.5%	3.4%
1260	38.2%	5.3%	18.1%	6.4%	27.7%	4.3%

Results are given as % of total time to execute WFTA.

TABLE 4.7

MEASURED AND PREDICTED TIMING RESULTS FOR MFFT, WFTA, AND PFA

N	MFFT (m)	MFFT (p)	WFTA1 (m)	WFTA1 (p)	WFTA2 (m)	WFTA2 (p)	PFA (m)	PFA (p)
60	.008	.004	.008	.004	.004 (.002)	.002	[illegible]	.002
126	.019	.010	.018	.011	.012	.006	.010	.006
210	.034	.018	.033 (.002)	.018	.018	.010	.019	.011
252	.037	.023	.032	.022	.020	.013	.019	[illegible]
315	.052	.031	.066 (.002)	.032	.032 (.003)	.020	[illegible]	.019
360	.047	.032	.046 (.002)	.030	.025	.017	.027	.015
420	.072	.041	.057	.037	.034	.022	[illegible] (.002)	.023
504	.082	.050	.074 (.002)	.046	.039	.028	[illegible]	.028
630	.109 (.002)	.068	.110	.066	.064	.042	[illegible]	.040
720	.083	.063	.106	.067	.061	.041	[illegible] (.002)	.042
840	.151	.080	.105 (.003)	.077	.067	.047	.075	.051
1008	.157	.100	.178 (.002)	.104	.091 (.002)	.066	.041 (.002)	.052
1260	.231	.147	.193	.137	.117	.088	.113	.064
2520	.491	.315	.413 (.006)	.285	.255 (.005)	.187	[illegible] (.002)	.141

Notation - all times given in seconds
- (m) is measured and (p) is predicted times for execution
- quantities in parentheses are standard deviations in excess of .001 seconds
- WFTA1 uses the initialization subroutine and WFTA2 was not reinitialized

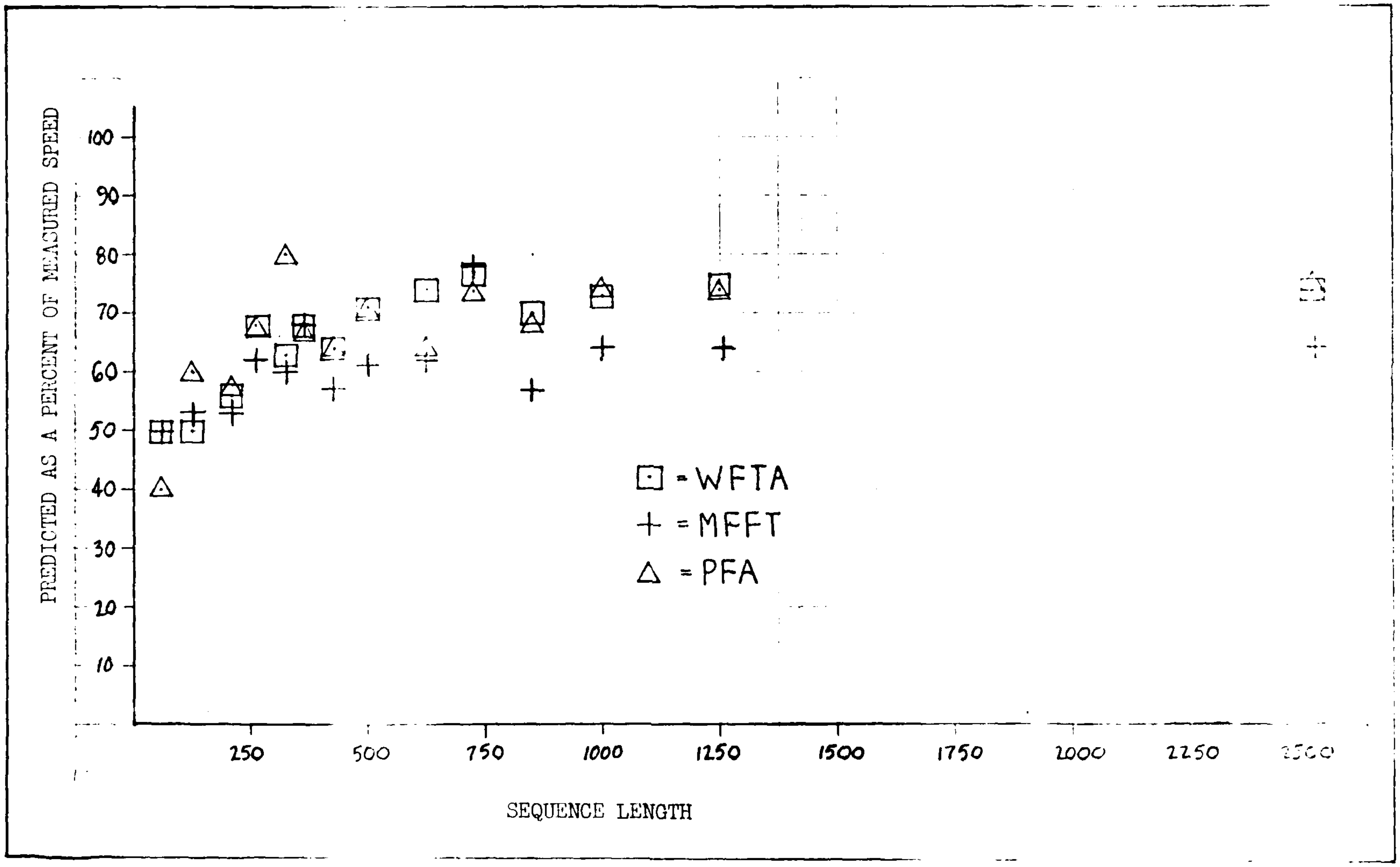


Figure 4.5. Pred[illegible] Times of Execution as a Percentage of Meas[illegible] for the [illegible], WFTA, and PFA.

PRECEDING PAGE BLANK-NOT FILMED

the algorithms, however, the predicted execution [illegible] based only on real operations are sufficient to select the most efficient algorithm as demonstrated by Table 4.7.

The timing results in Table 4.7 compare one-to-one with the predicted times (given the standard deviations shown in parentheses) for all three algorithms. Several observations can be made from Table 4.7. First, the WFTA1 which represents the initial transform made by WFTA may be slower than MFFT for certain sequence lengths. An example of this is N=315, 630, and 720, all of which were correctly predicted to be slower from the operations counts in Tables 3.9 and 4.6. Second, the post-initialized WFTA2 and the PFA were predicted to be, and are, faster than MFFT for all sequence lengths. Third, the PFA and WFTA2 (post-initialization) are close in efficiency for all sequence lengths.

4.5.2 <u>Memory</u>. The memory array for MFFT, WFTA, and PFA was compiled from the previous chapter and presented in Figure 4.6 and Table 4.5a and b. The figure clearly demonstrates how much less memory array is required by MFFT.

These results are due to the efficient data reordering technique of MFFT which can essentially be done in place with very little additional memory relative to the sequence length. The WFTA and PFA base their data reordering on the Chinese Remainder Theorem and require an additional two length N arrays for PFA. The WFTA uses even more memory array because of the algorithm's structure which "nest" multiplications inside all the additions. This requires

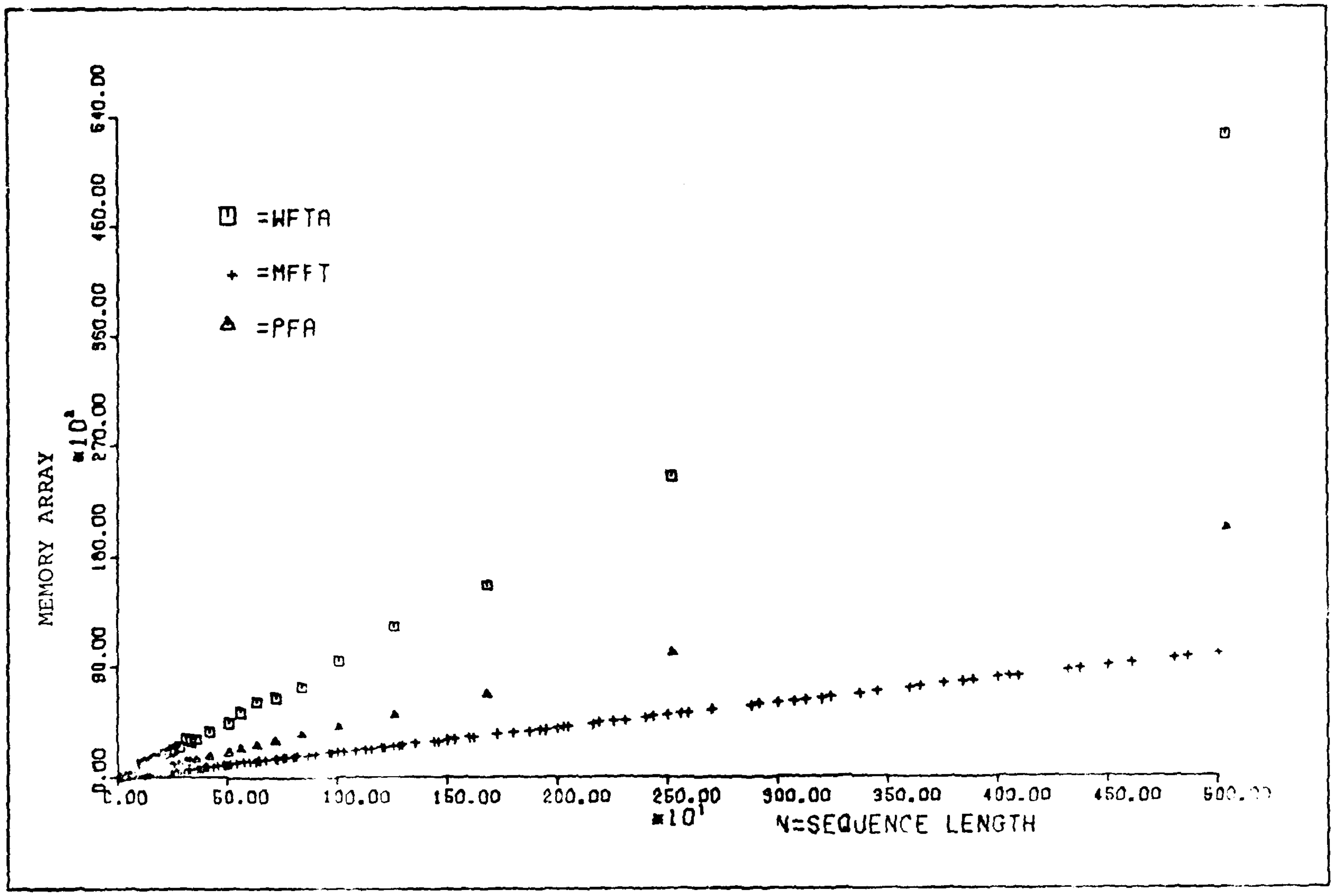


Figure 4.6. Memory Arrays Required by MFFT, WFTA, and PFA.

three additional arrays of length $M = m_1$ [illegible] are the multiplications required for the C_i [illegible] store the multiplication coefficients and provide working array storage because the WFTA is not computed in-place.

The program memory was not included in the tabulations for comparison because program memory required depends on the machine word size. The program memory required on the Cyber 74 for each algorithm is:

PFA program memory = 770 words

WFT program memory = 2348 words

FFT program memory = 1100 words

These results were achieved from the standard compiler command FTN for the FORTRAN IV language. For short sequences these program memory requirements contribute significantly to the choice of the most memory efficient algorithm.

4.5.3 <u>WFTA vs PFA Operations Count</u>. The tradeoffs between WFTA and PFA for real multiplications and additions can be seen in Figures 4.3 and 4.4. In most cases the WFTA requires less multiplications but more additions than PFA. The selection of the most efficient algorithm then becomes dependent on machine speed of real addition compared to real multiplication. As an example of this tradeoff between additions and multiplications consider the case of N=630. For this sequence length the PFA requires 4352 multiplications and 18534 additions while the WFTA requires 2376 multiplications and 22072 additions. Assuming the machine add speed of 1.7×10^{-6} seconds and a multiply speed of

1.9 x 10^{-6} [illegible] d execution speed for the [illegible] TA speed is .042 seconds. For the selected add and multiply speed PFA was faster. However, consider the case where a multiply requires three times the addition speed or 5.2 x 10^{-6} seconds. For the same N=630 the PFA speed is predicted to be .054 seconds and the WFTA speed is .050 seconds. With the increase in multiply time from 1.9 to 5.1 microseconds the WFTA became the more efficient algorithm. This example illustrated why the add and multiply speed must be known to select the fastest algorithm for a particular sequence length N.

The effects of changing the multiply to add ratio from 1 to 20 is shown in Figure 4.7a, b, and c for MFFT, WFTA, and PFA. For the sequences N=315 and 1008 the PFA is most efficient at the low multiply to add ratios but as the multiplies are "more costly" the WFTA soon becomes the most efficient. For N=30 the WFTA is the most efficient for all ratios.

4.6 Flexibility of the FFT Algorithms

It is clear from the plots in Figures 4.3, 4.4, and data in Table 4.2 that the fixed radix FFT, PFA, and WFTA are somewhat limited in permissible sequence lengths, whereas the mixed radix FFT provides a much more "dense" selection even for sequence lengths factorable by only 2, , 4, or 5. The restriction in possible values for N

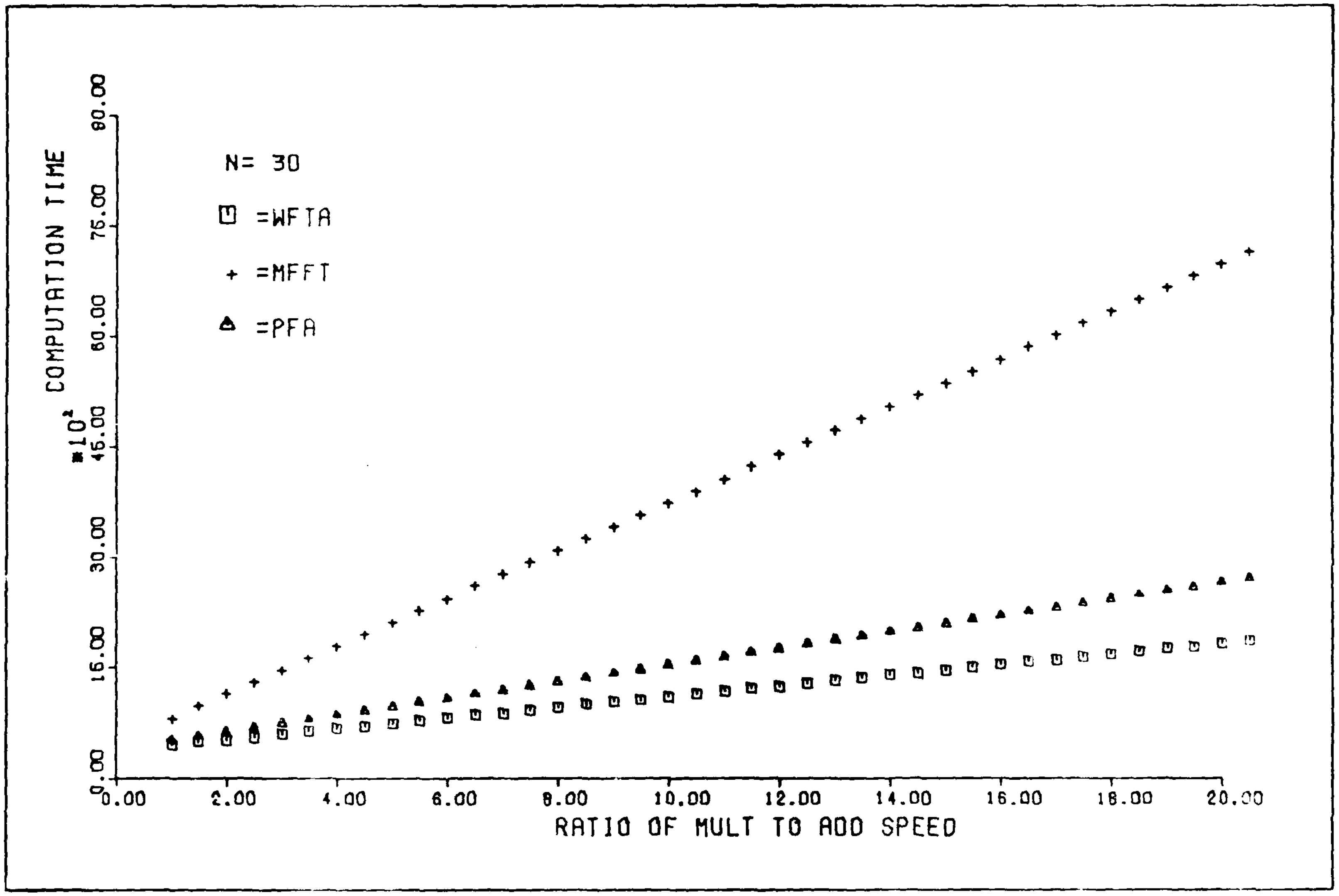


Figure 4.7a. Relative Efficiencies of MFFT, WFTA, and PFA.

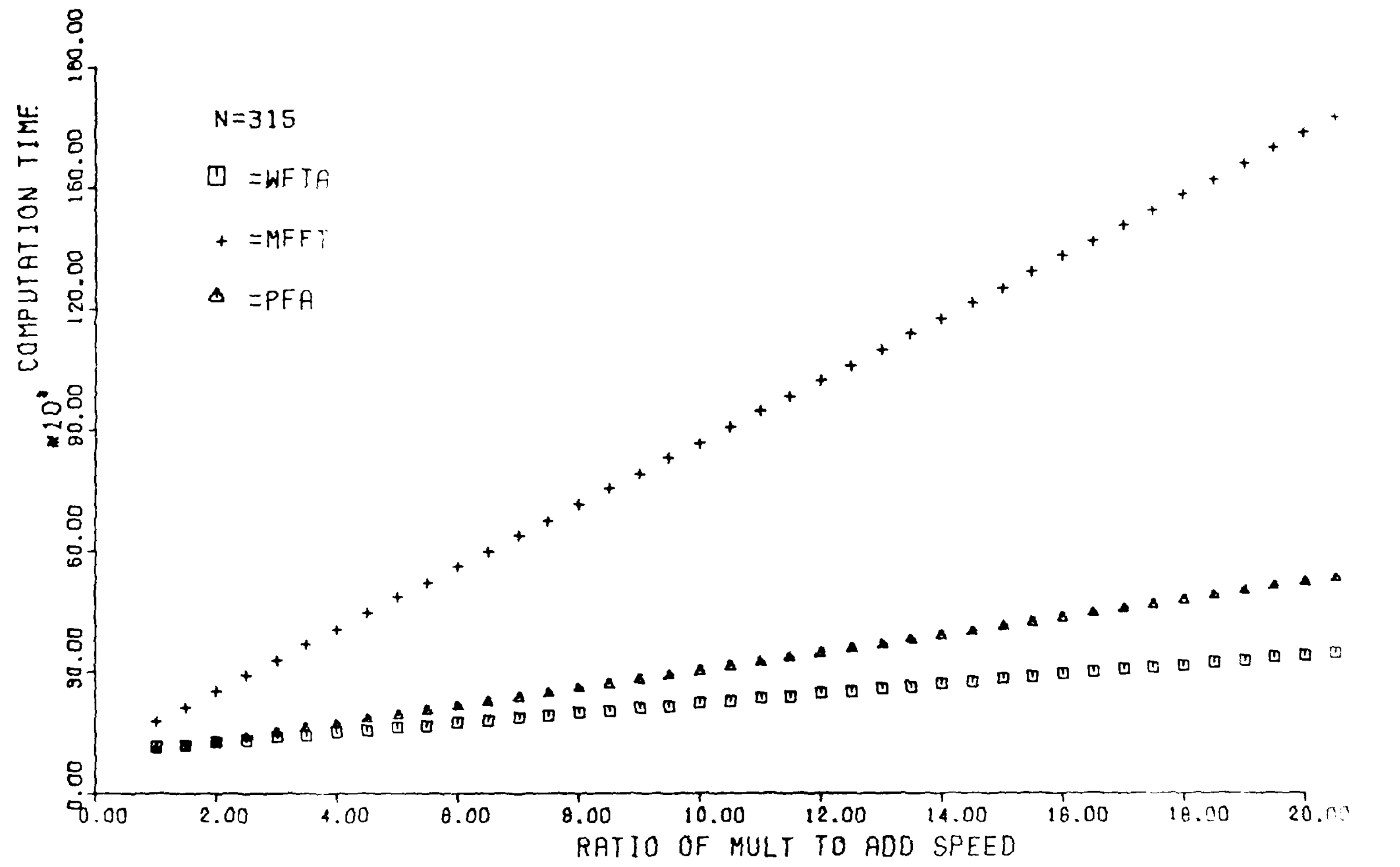


Figure 4.7b. Relative Efficiencies of MFFT, WFTA, and PFA.

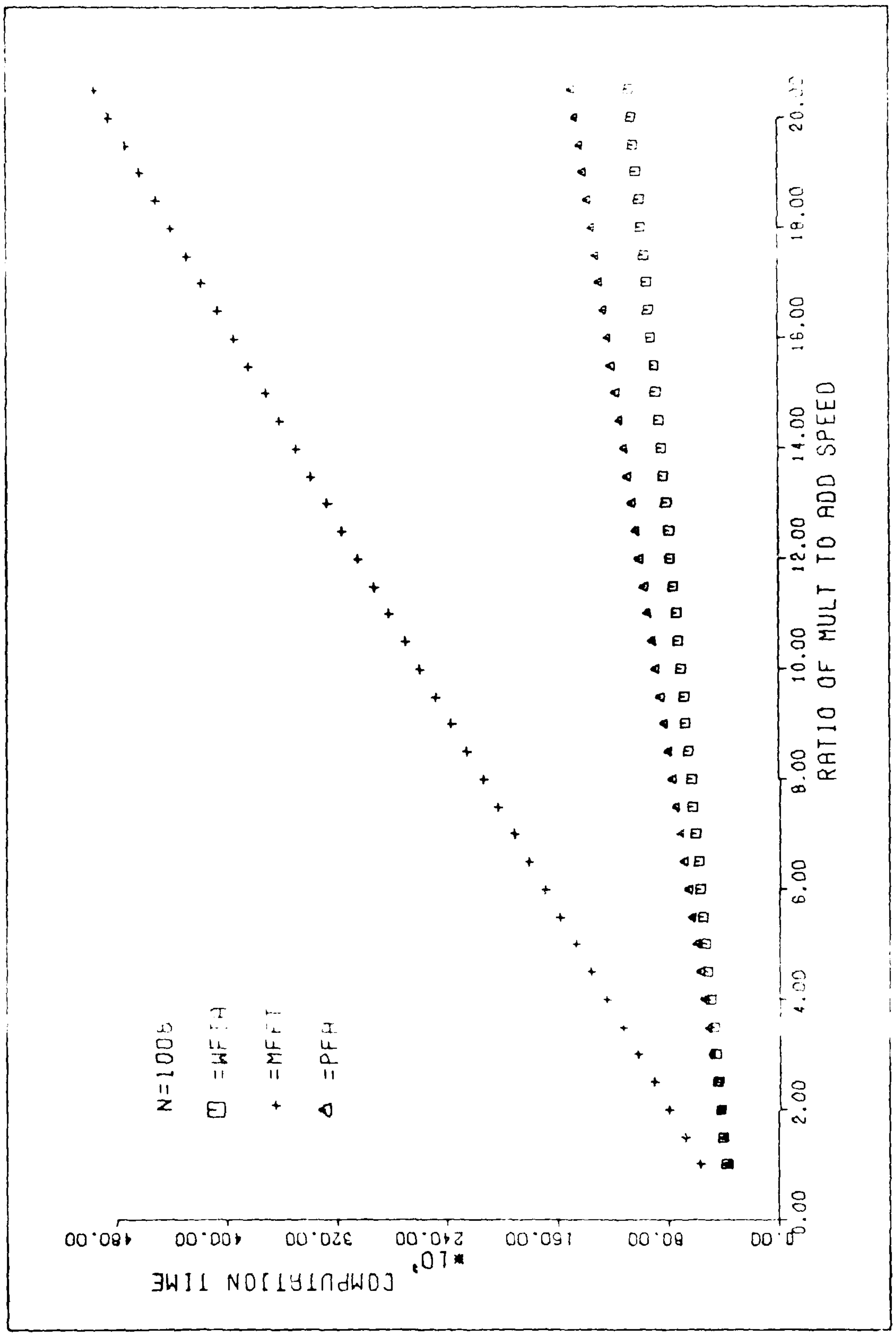


Figure 4.7c. Relative Efficiencies of MFFT, WFTA, and PFA.

in PFA and WFTA is because of the data reordering require-
m[illegible] that N [illegible] factored [illegible] from the set
2,3,4,5,7,8,9 and 16. This limits N to four factors and a
maximum value of 5040. The fixed radix algorithms are even
more restricted than PFA or WFTA because they can transforr
only sequence length which are an integer power of 2, 3, or 5.

4.7 <u>An Algorithm to Select the Most Efficient DFT Technique</u>.

The results of this chapter are used to develop a systematic approach to selecting the most efficient DFT method from the fixed radix FFTs, mixed radix FFT (MFFT), WFTA, and PFA. A flowchart is presented which selects the most efficient algorithm based on real operations, computer memory, machine speed, and sequence length. The algorithm requires inputs of machine speed for add and multiply, sequence length, zeropack limits, and computer memory. This algorithm also assumes that the same length sequence will be repeatedly transformed such that the WFTA is initialized only once.

4.7.1 <u>Arguments</u>. The algorithm requires inputs:

MEM:	Computer memory available for use
N:	Sequence length to be transformed
NP:	The upper limit to which the sequence length can be filled to reach an efficient transform length.
A:	Machine addition speed
M:	Machine multiplication speed

AD-A100 782 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/6 12/1
EFFICIENT COMPUTER IMPLEMENTATIONS OF FAST FOURIER TRANSFORMS.(U)
DEC 80 J D BLANKEN
UNCLASSIFIED AFIT/GE/EE/80D-9 NL

3 of 4
AD
A100782

4.7.2 Usage. The algorithm is presented as a flowchart. The basic logic of the algorithm is:

(1) Zeropack (if permitted) to the nearest WFTA or PFA sequence length.

(2) Determine the memory requirements for the WFTA and PFA.

(3) If WFTA and PFA both fit in computer memory available, select between the two by using real operations and computer speed.

(4) If only PFA or WFTA fit in computer memory, select the one that fits.

(5) If neither PFA nor WFTA will fit in computer memory, zeropack to nearest N an integer power of 2, 3, or 5. Choose the most efficient algorithm from the fixed radix FFT and MFFT based on real operations counts and machine speed.

(6) If fixed radix FFT cannot be used, zeropack to nearest N factorable by 2, 3, or 5 and use the mixed radix FFT.

Using the flow diagram of Figure 4.8a, b, and c along with the specified tables selects the most efficient algorithm.

An example for N=410 demonstrates the use of Figure 4.8 and the tables in this paper to select the most efficient DFT. Given that A=450 nanoseconds (ns), M=1000 ns, 10% zeropacking permitted, and no memory limitations, the most efficient algorithm can be selected.

(1) MEM is very large and is not a limitation

(2) N=410

(3) NP=410 + .10(410) = 451

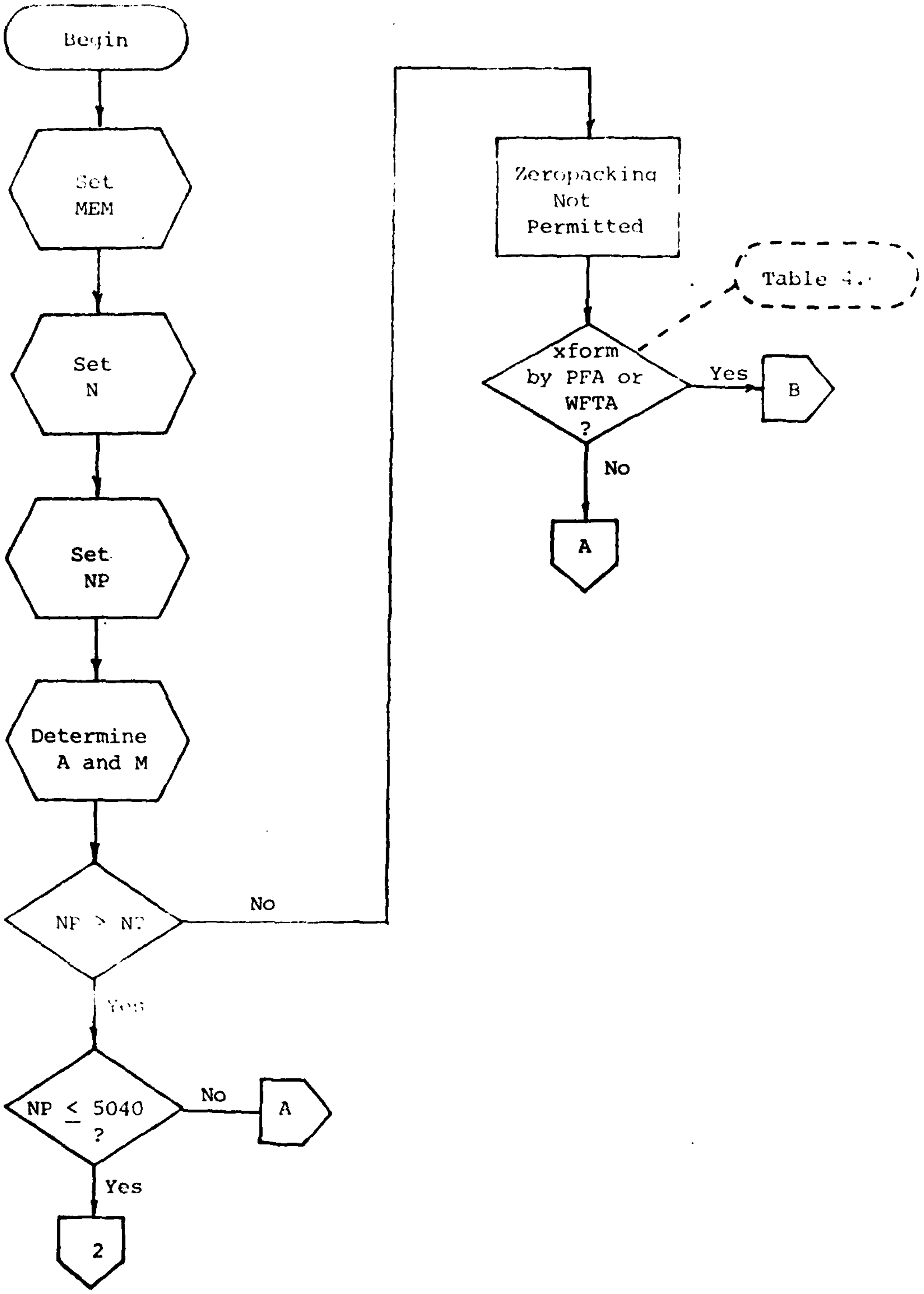
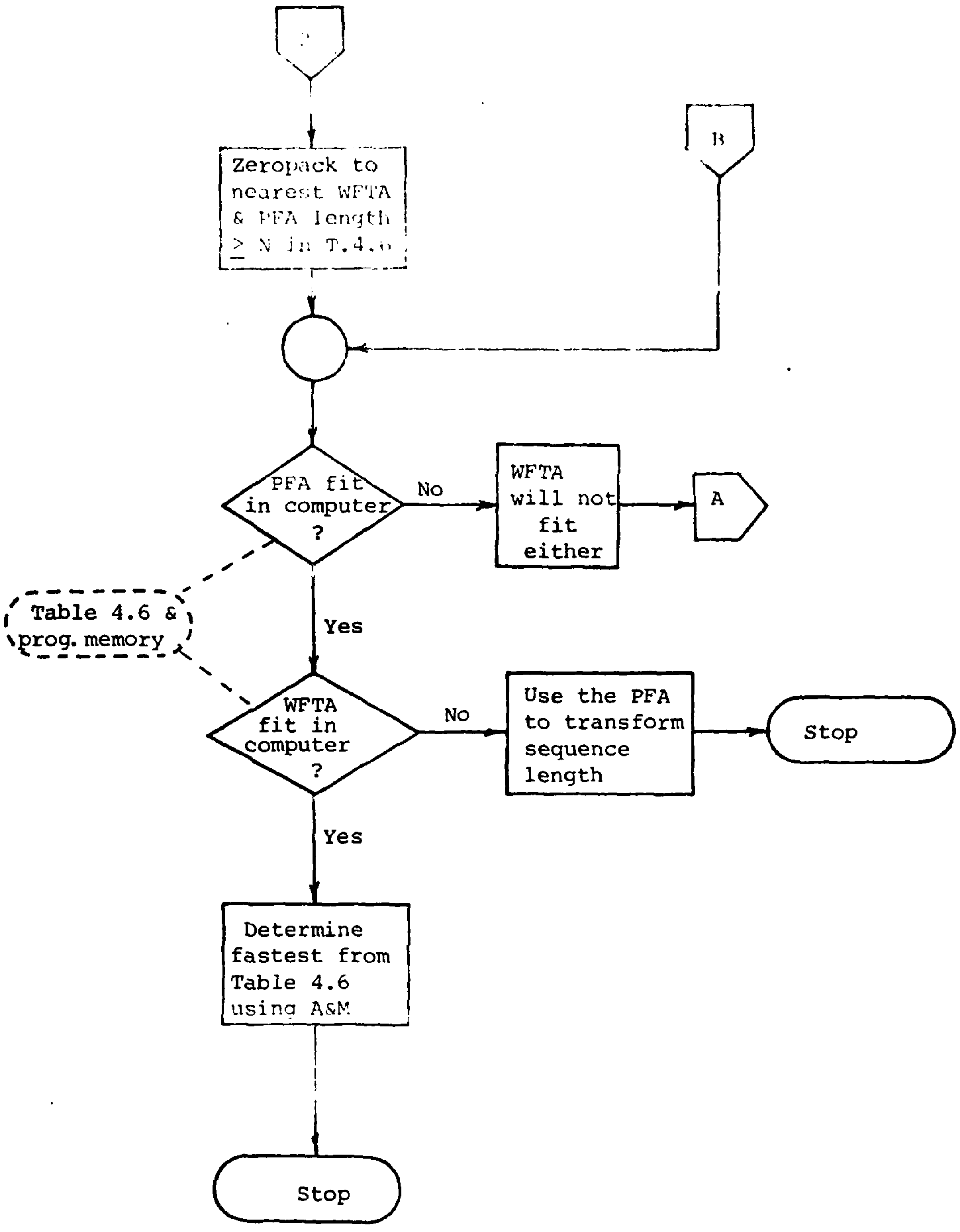


Figure 4.8a. Flowchart to Select Most Efficient Algorithm.



B
Zeropack to nearest WFTA & PFA length ≥ N in T.4.6
PFA fit in computer ?
No
WFTA will not fit either
A
Table 4.6 & prog. memory
Yes
WFTA fit in computer ?
No
Use the PFA to transform sequence length
Stop
Yes
Determine fastest from Table 4.6 using A&M
Stop

Figure 4.8b. Flowchart to Select Most Efficient Algorithm.

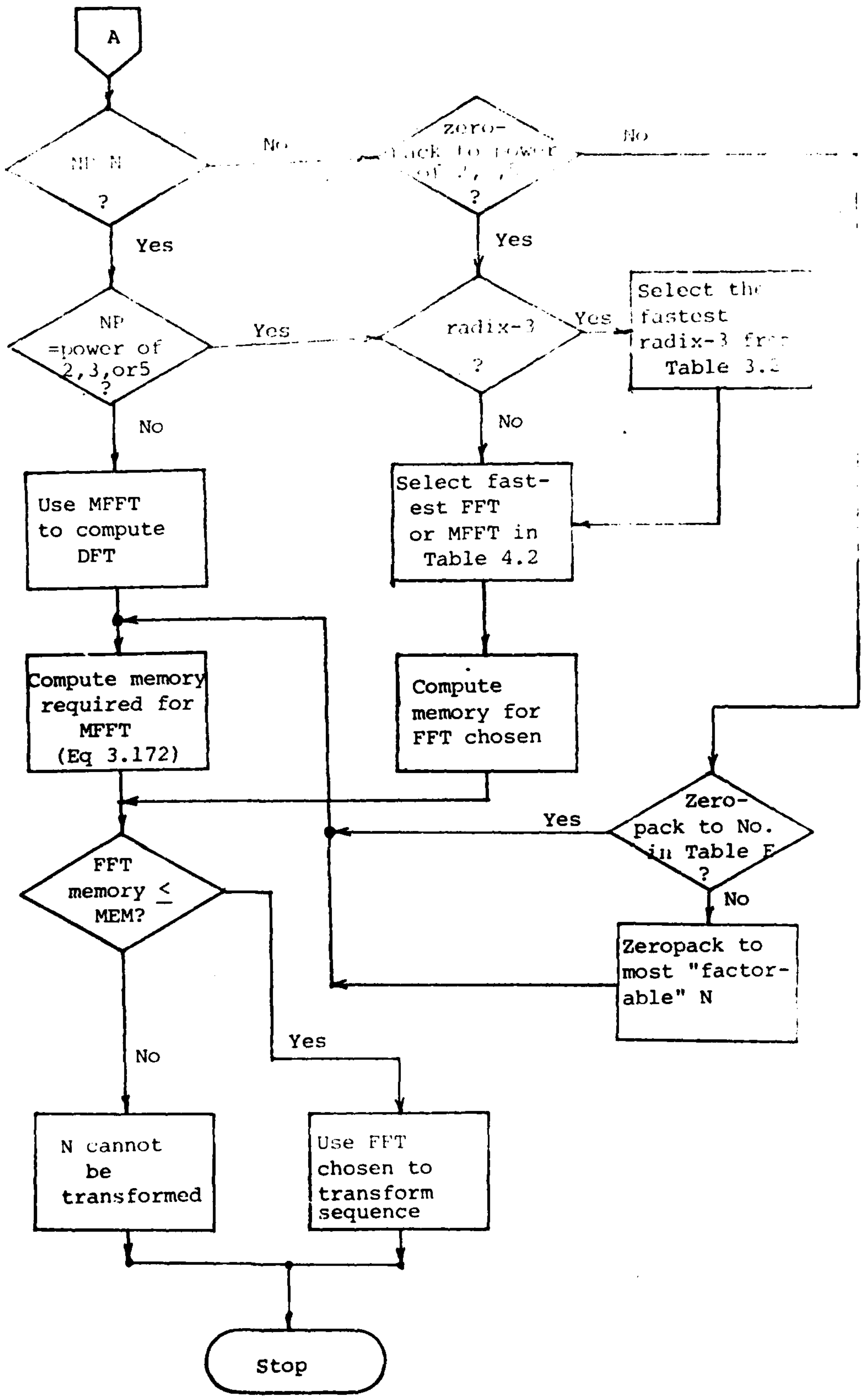

A
NP N ?
No
zero-pack to power of 2, 3, 5 ?
No
Yes
Yes
NP =power of 2,3,or5 ?
Yes
radix-3 ?
Yes
Select the fastest radix-3 from Table 3.2
No
No
Use MFFT to compute DFT
Select fastest FFT or MFFT in Table 4.2
Compute memory required for MFFT (Eq 3.172)
Compute memory for FFT chosen
Zero-pack to No. in Table E ?
Yes
No
FFT memory ≤ MEM?
Zeropack to most "factorable" N
No
Yes
N cannot be transformed
Use FFT chosen to transform sequence
Stop

Figure 4.8c. Flowchart to Select Most Efficient Algorithm.

(4) NP=N? No, continue

(5) NP≤5040? Yes, continue

(6) Zeropack to nearest WFTA PFA length given in Table 4.6 which is NP=420.

(7) PFA fit in computer? Yes, continue

(8) WFTA fit in computer? Yes, continue

(9) Determine fastest algorithm between WFTA and PFA from Table 4.6. For N=420,

	WFTA	PFA
Mult	1296	2528
Add	11352	10956

Using A=450 ns and M=1000 ns the predicted speeds are: WFTA = 6.4 milliseconds

PFA = 7.5 milliseconds

For this sequence N=420 and for the add and multiply speeds given the WFTA is the fastest algorithm. However, if this sequence were only being transformed once for a particular utilization and the WFTA could not be repeatedly used without initialization the WFTA counts must be taken from Table 3.11 where 4920 multiplications and 16200 additions are used to initialize the WFTA and perform the transform. Now the WFTA is predicted to use 56.5 milliseconds to transform N=420. When selecting between WFTA and PFA the particular utilization must be considered.

It should also be noted that the predicted times from Table 4.6 are based only on real operations which do not account for all of the execution time required as shown by the timing tests. For the cases tested in Table 4.7 on the CDC Cyber 74 the real operations accounted for average 67% of the PFA, 65% of the WFTA, and 61% of the MFFT actual execution speed.

V. Conclusions

This paper, for the first time, presented a capability to select the most efficient DFT based on real operations. These real operations were tabulated and plotted as a function of N. The algorithms studied and compared for real operations and memory include:

1. Radix-2 FFT from Rabiner and Gold.
2. Radix-3 FFT written by the author.
3. Radix-3 FFT in R(u) from Dubois and Venetsanopolous.
4. Radix-5 FFT written by the author.
5. Mixed radix FFT for factors of 2, 3, or 5 written by the author.
6. IMSL mixed radix FFT which can transform any sequence length N.
7. Singleton's mixed radix FFT which can transform any sequence length N.
8. Winograd Fourier transform algorithm (WFTA) written by McClellan and Nawab.
9. Prime Factor Algorithm (PFA) written by Burrus and Eschenbacher.

5.1 Results and Conclusions

The two radix-3 FFTs were compared for real operations and memory required to perform the DFT of N length sequences where $N=3^m$. Selection criteria were developed and tabulated based on machine speed. The new radix-3 FFT in the R(u) field uses less multiplications but more real additions than the conventional Radix-3 FFT. The more efficient of the two algorithms depends on the relative costs of multiplications and additions. The Radix-3 in R(u) is most efficient when multiplications are costly.

All of the fixed radix algorithms were compared to the Singleton mixed radix FFT for real operations and memory. The operations counts show that the most efficient algorithm depends on multiplication and addition speed of the computer. Data was tabulated for selecting the best algorithm based on this criteria. The FFT algorithm using the least memory can also be selected from Tables 4.2 and 4.3. The limited choice of sequence lengths possible with the fixed radix FFTs reduce their utility compared to Singleton's mixed radix FFT.

Three conventional mixed radix FFT algorithms were compared for efficiency, memory array, and flexibility. The author's mixed radix FFT was very efficient but required more memory array and was not as flexible since N was limited to factors of 2, 3, 4, and 5. It was shown that Singleton's mixed radix FFT was more efficient, flexible, and used less memory array than the IMSL mixed radix FFT and was chosen as the best conventional mixed radix FFT.

Singleton's mixed radix FFT (labeled MFFT) and the fixed radix FFTs were compared to the WFTA and PFA. The real operations and memory required was tabulated and plotted for all of the N length sequences permitted by WFTA and PFA.

This comparison showed that the WFTA and PFA required less real operations but that the FFTs requires less memory. The MFFT was much more flexible than WFTA or PFA since N can be any length sequence.

The WFTA and PFA were then more closely studied and the tradeoffs between the two were discussed. The PFA uses less additions but more multiplications for most N length sequences which means WFTA is more efficient when multiplications are "costly" relative to additions. The PFA uses less memory than the WFTA which makes PFA preferable when the machine is memory limited. Further criteria considered in selecting between these two algorithms are the (1) machine language and (2) the particular application of the algorithms. If the machine language permits "shifts" to be used for multiplication by 1/2 the PFA performance can be improved. (The percentage improvements have been tabulated for all permissible PFA sequence lengths). The second consideration affects the WFTA since any repeated use of WFTA for the same length N sequence does not require the algorithm to re-initialize the multiplier coefficients. Improvements in operating speeds of 40% over the initial WFTA were realized on the Cyber 74 for various sequence lengths.

An algorithm to select the most efficient DFT method from WFTA (Winograd), MFFT (Singleton), fixed radix FFTs, and PFA (Kolba and Parks) was presented. This selection is based on: minimizing real operations and minimizing memory size for the machine used. Minimizing real operations is the best "first order" criteria (Singleton, 1969) and was verified by timing the transforms on the CDC Cyber 74. A summary of the above conclusions is presented in Table 5.1.

The PFA was chosen as the best DFT technique because it minimizes real operations well below the FFT levels, requires substantially less memory than WFTA, and is more flexible than the fixed radix FFTs. Of course, the "optimum" algorithm depends on the specific application and computer, but for general applications the PFA provides the best mix of minimizing real operations and memory.

5.2 Recommendations

The above conclusions related to an algorithm's efficiency were based on real operations and then verified by timing tests on the CDC Cyber 74. The Cyber 74 is a representative large main frame computer with very high speed additions and multiplications.

To further substantiate the conclusions of this paper it is recommended that similar timing tests be made on other computers (large and small) available at AFIT and the results compared to the predicted efficiencies based on real additions and multiplications. All of the data necessary to perform these tests is available in this paper.

TABLE 5.1

COMPARISON OF DFT ALGORITHMS

	Real Operations	Memory Array	Program Memory	Flexibility	Overall
MFFT	Fair	Excellent	Fair	Excellent	Good
FFTs	Fair	Good	Excellent	Poor	Fair
WFTA	Excellent	Poor	Fair	Fair	Fair
PFA	Excellent	Good	Excellent	Fair	Excellent

Ratings: Excellent, Good, Fair, Poor

Bibliography

1. Bergland, G.D. "A Fast Fourier Transform Using Base-8 Interactions," Math. Comp., Vol. 22, pp. 275-279, April 1968.

2. Brenner, N.M. "Three FORTRAN Programs that Perform the Cooley-Tukey Fourier Transform," M.I.T. Lincoln Lab, Lexington, MASS, Tech Note 1967-2, July 1967.

3. Brigham, Oran E. The Fast Fourier Transform. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1974.

4. Burrus, C.S. and P.W. Eschenbacher. "An In-Place, In-Order Prime Factor FFT Algorithm," Unpublished Article. School of Electrical Engineering, Rice University, Houston, TX 77001, 1980.

5. Burrus, C.S. and T.W. Parks. "Efficient Techniques for Signal Processing", Final Report, Contract DASG50-78-C-0082, Ballistic Missile Defense Advanced Defence Center, 30 June 1979 (AD B039920L).

6. Cooley, J.W. and J.W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series", Mathematics of Computation, Vol. 19, No. 90, pp. 297-301, 1965.

7. Dubois, E. and A.N. Venetsanopoulos. "A New Algorithm for the Radix-3 FFT", IEEE Trans. on Acoustics, Speech, and Sig. Processing, Vol. ASSP-26, No. 3, pp. 222-225, June 1978.

8. Gentleman, W.M. and G. Sande. "Fast Fourier Transforms for Fun and Profit," 1966 Fall Joint Computer Conf., AFIPS Proc., Vol. 29, Washington, D.C., Spartan, pp. 563-578, 1966.

9. Kolba, D.P. A Prime Factor Algorithm Using High Speed Convolution. MS Thesis. Houston, Texas: Rice University School of Engineering, May 1977.

10. Kolba, D.P. and T.W. Parks. "A Prime Factor Algorithm Using High Speed Convolution," IEEE Trans. on Acoustics, Speech, and Sig. Processing, Vol. ASSP-25, No. 4, August 1977.

11. McClellan, J.H. and H. Nawab. "Complex General-N Winograd Fourier Transform Algorithm (WFTA)", Programs for Digital Signal Processing, Edited by Digital Signal Processing Committee IEEE Acoustics, Speech, and Sig. Processing Society, IEEE Press. New York: John Wiley and Sons, Inc., 1979.

12. McClellan, J.H. and C.M. Rader. Number Theory in Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.

13. Morris, L.R. "A Comparative Study of Time Efficient FFT and WFTA Programs for General Purpose Computers," IEEE Trans. on Acoustics, Speech, and Sig. Processing, Vol. ASSP-26, No. 2, April 1978.

14. Oppenheim, A.V. and R.W. Schafer. Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1975.

15. Pollard, J.M. "The Fast Fourier in the Finite Field", Math Comp., Vol. 25, April 1971.

16. Rabiner, L.R. and B. Gold. Theory and Application of Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975.

17. Rader, C.M. "Discrete Fourier Transforms When the Number of Data Samples is Prime", Proc. IEEE, Vol. 56, pp. 1107-8, June 1978.

18. Reed, I.S. and T.K. Truong. "The Use of Finite Fields to Compute Convolutions", IEEE Trans. on Inf. Theory, Vol. IT-21, No. 2, March 1975.

19. Silverman, H.F. "An Introduction to Programming the Winograd Fourier Transform Algorithm", IEEE Trans. on Acoustics, Speech, and Sig. Processing, Vol. ASSP-25, No. 2, April 1977.

20. Singleton, R.C. "On Computing the Fast Fourier Transform", Communications of the ACM, Vol. 10, No. 10, p. 651, October 1967.

21. Singleton, R.C. "An Algorithm for Computing the Mixed Radix Fast Fourier Transform", IEEE Trans. on Audio and Electroacoustics, Vol. AU-17, pp. 39-103, June 1969.

22. Thomas, L.H. "Using a Computer to Solve Problems in Physics", Applications of Digital Computers, Boston, MASS: Ginn, 1963.

23. Winograd, S. "Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants", IBM T.J. Watson Res. Ctr., Yorktown Heights, NY, IBM Res. Rep., Rc 5669, 10 October 1975.

24. Winograd, S. "On Computing the Discrete Fourier Transform", Proc. Nat. Acad. Sci. USA, Vol. 73, No. 4, pp. 1005-6, April 1976.

25. Zohar, S. "Fast Fourier Transformation: The Algorithm of S. Winograd", NASA, Jet Propulsion Lab, California Institute of Technology, Pasadena, CA, 15 February 79 (AS N79-19733).

Appendix A. Radix-2 FFT Algorithm

This appendix presents an algorithm for computing the complex fast Fourier transform (FFT) defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N)$$

where $k = 0,1, \ldots, N-1$

and $n=2^M$, M integer.

A FORTRAN subroutine is listed for computing the radix-2 FFT of a single-variate forward complex Fourier transform or calculates one variate of a multi-variate transform.

Arguments.

A = The complex array to be transformed which is dimensioned to length N.

N = The integer sequence length to be transformed which must have length equal 2^M.

M = The integer power of 2.

Usage. For a single variate forward transform:

(1) Specify the input complex sequence A along with parameters M and N.

(2) Dimension complex array A to length N.

(3) Call FFT2C (A,M,N).

(4) A contains the complex output vector X(k).

```
[illegible]
C     [illegible] TO [illegible]
C     [illegible] # OF DATA POINTS [illegible]
C  THIS FFT WAS TAKEN FROM RABINER & GOLD,
C  "THEORY AND APPLICATION OF DIGITAL SIGNAL
C  PROCESSING", P.367
      COMPLEX A(1),U,W,T
      N=2**M
      NV2=N/2
      NM1=N-1
      J=1
      DO 7 I=1,NM1
      IF(I.GE.J) GO TO 5
      T=A(J)
      A(J)=A(I)
      A(I)=T
 5    K=NV2
 6    IF(K.GE.J) GO TO 7
      J=J-K
      K=K/2
      GO TO 6
 7    J=J+K
      PI=3.14159265358[illegible]
      DO 20 L=1,M
      LE=2**L
      LE1=LE/2
      U=(1.,0.)
[illegible]
      T=A(IP)*U
      A(IP)=A(I)-T
 10   A(I)=A(I)+T
 20   U=U*W
      RETURN
      END
```

Appendix B. Radix-3 FFT Algorithm

This section presents an algorithm for computing the fast Fourier transform (FFT) based on a method called decimation-in-time described in Chapter III. This algorithm is an efficient method for computing the transformation:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N), \quad k = 0,1,2,\ldots, (N-1)$$

where X(k) and x(n) are complex valued. This algorithm requires that the sequence length be $N=3^m$, $m=0,1,2,\ldots$.

This appendix lists a FORTRAN subroutine for computing the radix-3 FFT. This subroutine computes the single-variate complex Fourier transform or calculates one variate of a multivariate transform.

Arguments.

A = The real part of the array to be transformed which is dimensioned to length N.

B = The imaginary part of the array to be transformed which is dimensioned to length N.

M = The exponent of 3.

N = The length of the data sequence ($N=3^M$).

IW = A work vector of length M.

WKS and WKC = Storage arrays of length N used for sine and cosine lookup tables.

Usage. For a single variate forward transform:

(1) Specify the input sequences A and B along with the parameters M and N.

(2) Dimension A,B,IW,WKS and WKC to the correct lengths.

(3) Call FFT3TM (A,B,M,N,IW,WKC,WKS).

(4) A and B are the output real and imaginary portion of the complex vector X(k).

```
[illegible]
[illegible]

150=C
160=C PURPOSE:
170=C      COMPUTE THE [illegible]
[illegible]

[illegible]
220=C ARGUMENTS:
230=C  A:     REAL VECTOR OF LENGTH N=2**M.   ON INPUT A CONTAINS
240=C      THE    REAL VALUE TO BE TRANSFORMED.
250=C      ON OUTPUT A IS REPLACED BY THE FOURIER TRANSFORM.
260=C  B:     REAL VECTOR OF LENGTH N=2**M.   ON INPUT B CONTAIN
270=C      THE     IMAG VALUE  TO BE TRANSFORMED.
280=C      ON OUTPUT B IS REPLACED BY THE FOURIER TRANSFORM.
290=C  M: INPUT EXPONENT TO WHICH 2 IS RAISED,I.E.,N=2**M.
300=C IW: WORK VECTOR OF LENGTH M.
310=C WKC: ARRAY FOR STORAGE OF THE COSINE TERMS USED TO TWIDDLE RESU
LTS
320=C WKS: ARRAY USED TO STORE THE SINE TERMS TO TWIDDLE THE RESULTS
330=C  N: LENGTH OF THE ARRAY TO BE TRANSFORMED.
340=C
350=C  REMARKS:
360=C      FFT IS COMPUTED USING BUTTERFLY FLOWGRAPH SHOWN BY
370=C      "DIGITAL SIGNAL PROCESSING" BY OPPENHEIM&SCHAFER P.314.
380=C
390=C  AUTHOR: JOHN D. BLANKEN, CAPT, USAF              4 AUG 80
400=C
410=C*************************************************************
420=C
430=C START OF SUBROUTINE FFTSTM:
440=C *** DIMENSION ALL ARRAYS: ***
450=C
460=       REAL A(N),B(N),WKS(N),WKC(N)
470=       DIMENSION IW(M),NCOUNT(15),NBASE(15)
480=C
490=C *** DECLARE INTEGER,REAL,COMPLEX,CONSTANTS:  **
500=C
510=       INTEGER D,TYPE,P,P02,P04
520=       PI=3.14159265358979
530=       N=2**M
540=C
550=C
560=       XIN2=SECOND(CP)
570=C COMPUTE THE SINE & COSINE LOOKUP TABLES USING THE RECURSIVE
580=C RELATIONSHIP
590=C
600=       THETA=2.0*PI/N
610=       SINSQ=SIN(THETA/2.0)
620=       C=-2.0*(SINSQ**2.0)
630=       S=SIN(THETA)
640=       WKC(1)=1.
```

```
      DO 30 I=1,NLOWER
      [illegible]
   30 WKC(I)=[illegible]
C FIL[illegible] LOCATION [illegible]
[illegible]
[illegible]
      [illegible]
      PRINT*,"NUPPER=",NUPPER
      J=NLOWER
      DO 81 I=NUPPER,N
      WKC(I)=WKC(J)
      WKS(I)=-WKS(J)
   81 J=J-1
      XOUT2=SECOND(CP)-XIN2
      PRINT*,"LOOK UP COMP TIME=",XOUT2
      PRINT*,"TEST 10"
      XIN1=SECOND(CP)
C *** SHUFFLE THE INPUT ARRAY USING BASE 3 COUNTER
C     AND THEN DIGIT REVERSE THE NOS: ***
C
      DO 22 M1=1,M
   22 IW(M1)=3
C
C     COMPUTE BASE NOS. OF COUNTER
      MFAC=M
      NBASE(1)=1
      DO 21 J=2,M
      NBASE(J)=IW(MFAC)*NBASE(J-1)
   21 MFAC=MFAC-1
C     ZERO THE NCOUNT ARRAY
      DO 24 J=1,M
   24 NCOUNT(J)=0
C     SET THE INDICES FOR A & B ARRAY
      ICOUNT=1
   28 NREV=1
      K=M
      DO 25 I=1,M
      NREV=NREV+NCOUNT(K)*NBASE(I)
   25 K=K-1
C     CK FOR SWAP
      IF(NREV.LE.ICOUNT) GO TO 26
      TA=A(NREV)
      TB=B(NREV)
      A(NREV)=A(ICOUNT)
      B(NREV)=B(ICOUNT)
      A(ICOUNT)=TA
      B(ICOUNT)=TB
   26 DO 27 I=1,M
      NCOUNT(I)=NCOUNT(I)+1
      IF(NCOUNT(I).LE.2) GO TO 28
      NCOUNT(I)=0
   27 CONTINUE
   28 ICOUNT=ICOUNT+1
```

```
      IF(ICOUNT.GE.N) GO TO 28
      GO TO [illegible]
   28 CONTINUE
      XOUT1=SECOND(CP)-XIN1
      PRINT*,"SHUFFLE TIME=",XOUT1
C
C *** ARRAYS ARE NOW SHUFFLED.***
C
C
C *** COMPUTE THE TRANSFORM: ***
C
C
C  COMPUTE BF CONSTANTS
      CONST=.8660254037844
C
C THIS LOOP COUNTS THE STAGE NO.  THE NO. OF STAGES =M
C
      XINN=SECOND(CP)
      DO 30 L=1,M
C
C THE INTEGER D IS THE DISTANCE BETWEEN BUTTERFLIES(BF)
C WHICH HAVE THE SAME COMPLEX TWIDDLE FACTORS :(TF)
C
      D=3**L
C
C TYPES OF BF IN STAGE WHICH USE DIFFERENT TF:
C
      LM1=L-1
      TYPE=3**LM1
C
C  INITIALIZE THE TWIDDLE FACTORS:
C
      TFA1=1.
      TFB1=0.
      TFA2=1.
      TFB2=0.
      TFA3=1.
      TFB3=0.
C
C COMPUTE DISTANCE BETWEEN BF ENDPOINTS FOR THIS STAGE:
C
      R=TYPE
      RX2=R*2
      RX4=R*4
C
C
C COMPUTE INDEX CONSTANTS FOR COS & SIN LOOKUP TABLES
C
      K1=N/D
      K2=2*K1
C  THIS LOOP INDEXES THE BF WITH SAME TFS, INDEXES THE TFS,
C
      DO 40 J=1,TYPE
C       FIRST STAGE HAS NO TF!S SO SKIP TF COMPUTATION
      IF(L.EQ.1) GO TO 60
      IF(J.EQ.1) GO TO 60
```

```
[illegible]
C
C TWIDDLE THE BF INPUTS AND [illegible]
C
       A1T=A(I1)
       B1T=B(I1)
C      IF L = 1, NO TF'S ARE REQUIRED:
       IF(L.EQ.1) GO TO 61
       IF(J.EQ.1) GO TO 61
       A2T=(A(I2)*TFA2)-(B(I2)*TFB2)
       B2T=(A(I2)*TFB2)+(B(I2)*TFA2)
       A3T=(A(I3)*TFA3)-(B(I3)*TFB3)
        B3T=(A(I3)*TFB3)+(B(I3)*TFA3)
       GO TO 62
 61    A2T=A(I2)
       B2T=B(I2)
       A3T=A(I3)
       B3T=B(I3)
C
C COMPUTE THE BF:
C
 62    A2A3=A2T+A3T
       B2B3=B2T+B3T
       A(I1)=A1T+A2A3
       B(I1)=B1T+B2B3
       PBM2=CONST*(A2T-A3T)
       PAM2=CONST*(B3T-B2T)
       PAM1=A1T-0.5*A2A3
       PBM1=B1T-0.5*B2B3
       A(I2)=PAM1-PAM2
       B(I2)=PBM1-PBM2
       A(I3)=PAM1+PAM2
       B(I3)=PBM1+PBM2
 50    CONTINUE
 40    CONTINUE
 30    CONTINUE
       XOVER=SECOND(CP)-XINN
       PRINT*,"BF TIME =", XOVER
C
C *** END OF TRANSFORM COMPUTATION. ***
       RETURN
C
C END OF FFT3TM SUBROUTINE
C
C***********************************************************************
C***********************************************************************
       END
```

Appendix C. Radix-3 FFT in R(u)

This appendix presents an algorithm for computing the radix-3 FFT based on a method which transforms the array from the complex domain (1,i) to the R(u) domain (1,u).

Arguments

A = Real portion of the complex data sequence to be transformed. It is dimensioned to length N.

B = Imaginary portion of the complex data sequence to be transformed. It is dimensioned to length N.

M = The exponent of 3.

N = The length of the data sequence ($N=3^M$).

IW = Work vector dimensioned to length M.

WKC and WKS = Storate array dimensioned to length N and used for sine and cosine look up tables.

RTEST = Set equal to zero or one. If the data sequence is real, RTEST=1; if the data sequence is complex, RTEST=0.

Usage. This algorithm is an efficient method for computing the transformation:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N) \quad k = 0,1, \ldots$$

where X(k) and x(n) are complex valued. This algorithm restricts N to equal 3^M where $M = 0,1,2, \ldots$.

For a single variate forward transform:

(1) Specify the input sequences A and B along with parameters M, N, and RTEST.

(2) Dimension A,B,WKC,WKS, and IW.

(3) Call FFT3RU (A,B,M,N,IW,WKC,WKS,RTEST).

(4) A and B are the output real and imaginary portion of the complex vector X(k).

```
      ●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●
                                                              ●●●●●●●
      [illegible]

[illegible]
[illegible]
[illegible]
[illegible]
230=C ARGUMENTS:
240=C  A:     REAL VECTOR OF LENGTH N=3**M.  ON INPUT A CONTAINS
250=C     THE    REAL VALUE TO BE TRANSFORMED.
260=C     ON OUTPUT A IS REPLACED BY THE FOURIER TRANSFORM.
270=C  B:     REAL VECTOR OF LENGTH N=3**M.  ON INPUT B CONTAINS
280=C     THE     IMAG VALUE TO BE TRANSFORMED.
290=C     ON OUTPUT B IS REPLACED BY THE FOURIER TRANSFORM.
300=C  M: INPUT EXPONENT TO WHICH 3 IS RAISED,I.E.,N=3**M.
310=C IW: WORK VECTOR OF LENGTH M.
320=C  N: LENGTH OF THE ARRAY TO BE TRANSFORMED.
330=C RTEST:         TEST FLAG=1 IF REAL TRANSFORM
340=C                          TEST FLAG = 0 IF IF COMPLEX TRANSFORM
350=C
360=C
370=C
380=C  REMARKS:
390=C     FFT IS COMPUTED USING BUTTERFLY FLOWGRAPH SHOWN BY
400=C     "DIGITAL SIGNAL PROCESSING" BY OPPENHEIM&SCHAFER P.314.
410=C     THIS SUBROUTINE UTILIZES ARITHMETIC IN THE R(W) FIELD DEFIN
ED
420=C     BY "A NEW ALGORITHM FOR THE RADIX-3 FFT",IEEE TRANS ON ACOU
,SPEE,
430=C     AND SIG PROC,JUNE 78.
440=C
450=C ●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●
460=C
470=C [illegible]:
480=C ●●● [illegible] : ●●●
490=C
500=       REAL A(N),B(N),WR[illegible](N),WRC(N)
510=       DIMENSION IW(M)
520=C
530=C ●●● DECLARE INTEGERS,REALS,COMPLEX,CONSTANTS: ●●●
540=C
550=       INTEGER M1,ICOUNT,NREV,M2,MMM2,L,D,TYPE,P,P[illegible]2,LM1,P[illegible]4
560=       INTEGER [illegible]1,[illegible]2,J,I1,I2,I3,RTEST
570=       REAL ARG,TA,TB,TFA1,TFA2,TFA3,TFB1,TFB2,TFB3,AA,BB,CC,DD
580=       REAL A3B2T,A2B3T,A1T,A2T,A3T,B1T,B2T,B3T,INVS03
590=       XIN1=SECOND(CP)
600=       S03=SQRT(3.0)
610=       INVS03=1./S03
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
 720=          [illegible]
 730=          [illegible]
 740=          S=SIN(THETA)
 750=         WKC(1)=1.
 760=         WKS(1)=0.
 770=         NLOWER=N/2+1
 780=         PRINT*,"NLOWER=",NLOWER
 790=         DO 80 I=2,NLOWER
 800=         WKS(I)=C*WKS(I-1)+S*WKC(I-1)+WKS(I-1)
 810= 80      WKC(I)=C*WKC(I-1)-S*WKS(I-1)+WKC(I-1)
 820=C FILL WKS & WKC ARRAY LOCATIONS FROM N/2+2 THRU N WITH COMPLEX
 830=C CONJUGATE OF LOCATIONS 2 THRU N/2+1
 840=         NUPPER=NLOWER+1
 850=         PRINT*,"NUPPER=",NUPPER
 860=         J=NLOWER
 870=         DO 81 I=NUPPER,N
 880=         WKC(I)=WKC(J)
 890=         WKS(I)=-WKS(J)
 900= 81      J=J-1
 910=         XOUT1=SECOND(CP)-XIN1
 920=         PRINT*,"LOOK UP TIME=",XOUT1
 930=C
 940=         XIN2=SECOND(CP)
 950=C *** SHUFFLE THE INPUT ARRAY USING BASE 2 COUNTER
 960=C        AND THEN DIGIT REVERSE THE NO.: ***
 970=C
 980=C ZERO THE DIGIT ARRAY:
 990=         DO 22 M1=1,M
[illegible]
[illegible]
[illegible]
[illegible]
1040=   22    M1=1
1050=C
1060=C COMPUTE THE REVERSED DIGIT VALUE OF N:
1070=   25    NREV=1
1080=         DO 24 M2=1,M
1090=         MMM2=M-M2
1100=   24    NREV=NREV+IN(M2)*(2**MMM2)
1110=C
1120=C CHECK IF SHUFFLE IS REQUIRED ON THIS PAIR:
1130=         IF(NREV.LE.ICOUNT) GO TO 26
1140=C NO, SWAP A(NREV) & A(ICOUNT):
1150=         TA=A(NREV)
1160=         TB=B(NREV)
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
      [illegible]
      IF([illegible].EQ.1) GO TO [illegible]
      GO TO [illegible]
 29   CONTINUE
      DUTG=SECOND([illegible])-[illegible]
      PRINT*,"[illegible] SHUFFLE TIME=",[illegible]
C
C *** ARRAYS ARE NOW SHUFFLED.***
C
C  R=1: REAL ARRAY & CONVERSION IS NOT REQUIRED
C  R=0: COMPLEX ARRAY & CONVERSION TO R(U) FIELD REQUIRED
C
C     CONVERT A + BU
C                        TEST FOR REAL OR COMPLEX ARRAY
      IF(RTEST.EQ.1) GO TO 64
      XCON1T=SECOND(CP)
      DO 63 I=1,N
      AA=A(I)
      BB=B(I)
      BBXSQ3=BB*INVSQ3
      A(I)=AA-BBXSQ3
 63   B(I)=-2.0*BBXSQ3
      PRINT*," TIME TO CONVERT FROM C TO R(U) = ",XCON[illegible]
 64   CONTINUE
C
C *** COMPUTE THE TRANSFORM: ***
[illegible]
[illegible]
[illegible]
[illegible]
      [illegible]=1
      DO 30 L=1,M
C
C THE INTEGER D IS THE DISTANCE BETWEEN BUTTERFLIES (BF)
C WHICH HAVE THE SAME COMPLEX TWIDDLE FACTORS (TF)
C
      D=[illegible]**[illegible]
C
C TYPES OF BF IN STAGE WHICH USE DIFFERENT TF
C
      LM1=L-1
      TYPE=3**LM1
C
```

```
[illegible]
C [illegible]
      [illegible]
      [illegible]
      [illegible]
C  THIS LOOP INDEXES THE BF WITH SAME TFS, INDEXES THE TF[illegible]
C  AND CONVERTS TO R(U) NOTATION
C
      DO 40 J=1,TYPE
C       FIRST STAGE HAS NO TF!S SO SKIP TF COMPUTATION
      IF(L.EQ.1) GO TO 60
      IF(J.EQ.1) GO TO 60
      JM1=J-1
        AA=WKC(JM1*K1+1)
        BB=-WKS(JM1*K1+1)
      BBXS03=BB*INVS03
      TFA2=AA-BBXS03
      TFB2=-2.0*BBXS03
        AA=WKC(JM1*K2+1)
        BB=-WKS(JM1*K2+1)
      BBXS03=BB*INVS03
      TFA3=AA-BBXS03
      TFB3=-2.0*BBXS03
      INCUNT=INCUNT+1
40    CONTINUE
[illegible]

      DO 61 I1=[illegible]
      I2=I1+R
      I3=I1+R*2
C
C  TWIDDLE THE BF INPUTS AND STORE RESULTS IN TEMP LOCATIONS:
C
      A1T=A(I1)
      B1T=B(I1)
C     IF L = 1, NO TF!S ARE REQUIRED:
      IF(L.EQ.1) GO TO 61
      IF(J.EQ.1) GO TO 61
      AXC=A(I2)*TFA2
      BXD=B(I2)*TFB2
```

```
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]=
2440=C  [illegible]TE THE BF:
2450=C
[illegible]
2470=       B(I1)=B1T+B2T+B3T
2480=       A3B2T=A3T+B2T
2490=       A2B3T=A2T+B3T
2500=       A(I2)=A1T+B3T-A3B2T
2510=       B(I2)=B1T+A2T-A3B2T
2520=       A(I3)=A1T+B2T-A2B3T
2530=       B(I3)=B1T+A3T-A2B3T
2540= 50    CONTINUE
2550= 40    CONTINUE
2560= 30    CONTINUE
2570=       XOVER=SECOND(CP)-XINN
2580=       PRINT*,"XOVER= ",XOVER
2590=C
2600=C***CONVERT BACK TO A+BJ NOTATION
2610=C
2620=       XCONST=SECOND(CP)
2630=       DO 70 I=1,N
2640=       CC=A(I)
2650=       DD=B(I)
2660=       A(I)=CC-DD*0.5
[illegible]
[illegible]
[illegible]

2710=C***[illegible]
2720=       PRINT*,"NO. OF TF CONVERSIONS = ",ICOUNT
2730=       RETURN
2740=C
2750=C END OF SUBROUTINE [illegible]
2760=C
2770=C*****************************************************************
2780=C*****************************************************************
2790=       END
2800=*EOR
2810=*EOF
```

Appendix D. Radix-5 FFT Algorithm

This section presents an algorithm for computing the FFT based on decimation-in-time of the discrete Fourier transform defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N) \quad k = 0,1,2, \ldots N-1$$

where X(k) and x(n) are complex valued. This algorithm restricts the length of the sequence to be $N=5^m$ where m is an integer.

In this appendix a FORTRAN subroutine FFT5TF is listed for computing the radix-5 FFT. This subroutine computes the single-variate complex Fourier transform or performs the calculation for one variate of a multivariate transform.

Arguments

A = Real portion of the complex data sequence to be transformed. It is dimensioned to length N.

B = Imaginary portion of the complex data sequence. It is dimensioned to length N.

M = Exponent of 5, where $N=5^M$.

N = Length of the data sequence ($N=5^M$).

IW = Work vector of length M.

WKC and WKS = Storage arrays dimensioned to length N and used for sine and cosine look up tables.

Usage. For a single variate forward transform:

(1) Specify the input sequences A and B along with the parameters M and N.

(2) Dimension A,B,IW,WKS and WKC to correct lengths.

(3) A and B are the output real and imaginary portion of the complex vector X(k).

```
200=C  [illegible]: [illegible] FFT5TF(A,[illegible]) [illegible]
210=C
[illegible]
230=C  A: [illegible] OF LENGTH [illegible]
240=C      THE [illegible] REAL VALUE TO BE TRA[illegible].
250=C      ON OUTPUT A IS REPLACED BY THE [illegible]
[illegible]
270=C      THE [illegible] VALUE TO BE TRAN[illegible]D.
280=C      ON OUTPUT B IS REPLACED BY THE FOURIER TRANSFORM.
290=C  M: INPUT EXPONENT TO WHICH 5 IS RAISED,I.E.,N=5**M.
300=C  N: LENGTH OF THE ARRAY TO BE TRANSFORMED.
310=C IW: WORK VECTOR OF LENGTH M.
320=C WKC: ARRAY OF LENGTH N USED TO STORE COSINE TERMS
330=C WKS: ARRAY OF LENGTH N USED TO STORE SINE TERMS
340=C
350=C       AUTHOR:  JOHN D. BLANKEN, CAPT, USAF
360=C
370=C
380=C*************************************************************
390=C
400=C START OF SUBROUTINE FFT5TF:
410=C *** DIMENSION ALL ARRAYS: ***
420=C
430=      REAL A(N),B(N),WKC(N),WKS(N)
440=      DIMENSION NCOUNT(15),NBASE(15)
450=      DIMENSION IW(M)
460=C
470=C *** [illegible]
[illegible]
520=C
530=      [illegible]
540=C COMPUTE THE SINE & COSINE LOOKUP TABLES USING THE [illegible]
550=C RELATIONSHIP
560=C
570=      THETA=[illegible]
580=      [illegible]
590=      [illegible]
600=      S=SIN(THETA)
610=      WKC(1)=1.
620=      WKS(1)=0.
630=      NLOWER=N/2+1
640=      PRINT*,"NLOWER=",NLOWER
```

```
 790=        PRINT*, [illegible] TIME=, [illegible]
 800=C
 810=C ***  SHUFFLE THE INPUT ARRAY USING BASE 5 COUNTER
 820=C      AND THEN DIGIT REVERSE THE [illegible] : ***
 830=C
 840=       DO 22 M1=1,M
 850=   22  IW(M1)=5
 860=C      COMPUTE BASE NOS. OF COUNTER
 870=       MFAC=M
 880=       NBASE(1)=1
 890=       DO 21 J=2,M
 900=       NBASE(J)=IW(MFAC)*NBASE(J-1)
 910=   21  MFAC=MFAC-1
 920=C      ZERO TH ENCOUNT(J) ARRAY
 930=       DO 24 J=1,M
 940=   24  NCOUNT(J)=0
 950=C      SET THE INDICES FOR A & B ARRAY
 960=       ICOUNT=1
 970=   23  NREV=1
 980=       K=M
 990=       DO 25 I=1,M
1000=       NREV=NREV+NCOUNT(K)*NBASE(I)
1010=   25  K=K-1
1020=C      CK FOR SWAP
1030=       IF(NREV.LE.ICOUNT) GO TO 26
[illegible]
1050=       TB=[illegible]
[illegible]
1080=       A(ICOUNT)=[illegible]
1090=       B(ICOUNT)=TB
1100=   26  DO 27 I=1,M
1110=       NCOUNT(I)=NCOUNT(I)+1
1120=       IF(NCOUNT(I).LE.4) GO TO 28
1130=       NCOUNT(I)=0
1140=   27  CONTINUE
1150=   28  ICOUNT=ICOUNT+1
1160=       IF(ICOUNT.GE.N) GO TO 29
1170=       GO TO 23
1180=   29  CONTINUE
1190=C
1200=C ***  ARRAYS ARE NOW SHUFFLED.***
```

```
        [illegible]
        TIN4=.[illegible]
C THI[illegible] LOOP [illegible]
C
        TINN=[illegible]
        [illegible]
C
C THE INTEGER D IS THE DISTANCE BETWEEN BUTTERFLIES(BF)
C WHICH HAVE THE SAME COMPLEX TWIDDLE FACTORS :(TF)
C
        D=5**L
C
C TYPES OF BF IN STAGE WHICH USE DIFFERENT TF& DISTANCE BETWEEN
C BF ENDPOINTS FOR THIS STAGE
C
        LM1=L-1
        R=5**LM1
C
C  INITIALIZE THE TWIDDLE FACTORS
C
        TFA1=1.
        TFB1=0.
        TFA2=1.
        TFB2=0.
        TFA3=1.
        TFB3=0.
        TFA4=[illegible].
        [illegible].
        [illegible].
        [illegible].
C
C       COMPUTE THE [illegible] SIN TABLE INDICES
C
        K1=N/D
        K2=2*K1
        K3=3*K1
        K4=4*K1
C
C  THIS LOOP INDEXES THE BF WITH SAME TFS & COMPUTES THE TFS:
C
        DO 40 J=1,R
C        FIRST STAGE HAS NO TF!S SO SKIP TF COMPUTATION
        IF(L.EQ.1) GO TO 60
```

```
1900=C
1910=       DO 50 I1=[illegible]
1920=       I2=I1+R
[illegible]
1940=       I4=I3+R
1950=       I5=I4+R
1960=C
1970=C  TWIDDLE THE BF INPUTS AND STORE RESULTS IN TEMP LOCATIONS:
1980=C
1990=       A1T=A(I1)
2000=       B1T=B(I1)
2010=C      IF L = 1, NO TF!S ARE REQUIRED:
2020=       IF(L.EQ.1) GO TO 61
2030=       IF(J.EQ.1) GO TO 61
2040=       A2T=(A(I2)*TFA2)-(B(I2)*TFB2)
2050=       B2T=(A(I2)*TFB2)+(B(I2)*TFA2)
2060=       A3T=(A(I3)*TFA3)-(B(I3)*TFB3)
2070=        B3T=(A(I3)*TFB3)+(B(I3)*TFA3)
2080=       A4T=(A(I4)*TFA4)-(B(I4)*TFB4
2090=       B4T=(A(I4)*TFB4)+(B(I4)*TFA4)
2100=       A5T=(A(I5)*TFA5)-(B(I5)*TFB5)
2110=        B5T=(A(I5)*TFB5)+(B(I5)*TFA5)
2120=       GO TO 62
2130= 61    A2T=A(I2)

[illegible]
2190=       A5T=A(I5)
2200=       B5T=B(I5)
2210=C
2220=C COMPUTE THE BF:
2230=C
2240=  [illegible]
2250=  [illegible]
2260=       B3MB4=B3T-B4T
2270=       B2MB5=B2T-B5T
2280=       A4MA3=A4T-A3T
2290=       A5MA2=A5T-A2T
2300=       B3PB4=B3T+B4T
```

```
[illegible]
[illegible]
[illegible]
[illegible]
2450=      C2B3B4=C[illegible]*[illegible]
2460=      S4A5A2=S[illegible]*[illegible]
[illegible]
[illegible]
2490=      CA34=A1T+C4A2A5+C2A3A4
2500=      SA25=S4B3B4+S2B2B5
2510=      DA34=S4B2B5-S2B3B4
2520=      CB25=B1T+C4B3B4+C2B2B5
2530=      CB34=B1T+C4B2B5+C2B3B4
2540=      SB25=S4A4A3+S2A5A2
2550=      DB34=S4A5A2-S2A4A3
2560=      A(I1)=A1T+A3PA4+A2PA5
2570=      A(I2)=CA25+SA25
2580=      A(I3)=CA34+DA34
2590=      A(I4)=CA34-DA34
2600=      A(I5)=CA25-SA25
2610=      B(I1)=B1T+B3PB4+B2PB5
2620=      B(I2)=CB25+SB25
2630=      B(I3)=CB34+DB34
2640=      B(I4)=CB34-DB34
2650=      B(I5)=CB25-SB25
2660= 50   CONTINUE
2670= 40   CONTINUE
2680= 30   CONTINUE
[illegible]
[illegible]
[illegible]
2740=C
2750=C END OF FFTSTF SUBROUTINE
2760=C
2770=C*********************************************************
2780=C*********************************************************
2790=      END
2800=*ACE
2810=*EOF
```

Radix-5 FFT Theory

This section presents the theory of the radix-5 FFT starting with the DFT definition and then decomposing the DFT equation using the decimation-in-time algorithm (Cooley and Tukey, 1965). This development closely parallels the radix-3 development presented earlier and consequently the radix-5 theory will be brief.

The DFT X(k) is computed by separating the discrete time sequence X(n) into five N/5 point sequences (n must be of length 5^m, m = 0,1,2, ...). X(k) is given by the DFT expression:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad \text{where } k = 0,1, \ldots, N-1 \text{ and } W_N = \exp(-j2\pi/N) \qquad (D.1)$$

Breaking X(n) into five N/5 point sequences yields X(5r), X(5r+1), X(5r+2), X(5r+3), and X(5r+4). Using these sequences and Eq (D.1) gives:

$$X(k) = \sum_{r=0}^{N/5-1} x(5r) W_N^{5rk} + \sum_{r=0}^{N/5-1} x(5r+1) W_N^{(5r+1)k} + \sum_{r=0}^{N/5-1} x(5r+2) W_N^{(5r+2)k}$$
$$+ \sum_{r=0}^{N/5-1} x(5r+3) W_N^{(5r+3)k} + \sum_{r=0}^{N/5-1} x(5r+4) W_N^{(5r+4)k} \qquad (D.2)$$

By regrouping exponents and making the substitution of:

$$W_N^{5r} = W_{N/5}^{r} \qquad (D.3)$$

then Eq (D.2) can be written in final form as:

$$X(k) = \sum_{r=0}^{N/5-1} x(5r) W_{N/5}^{rk} + W_N^{k} \sum_{r=0}^{N/5-1} w(5r+1) W_{N/5}^{rk}$$

$$+ W_N^{2k} \sum_{r=0}^{N/5-1} x(5r+2) W_{N/5}^{rk} + W_N^{3k} \sum_{r=0}^{N/5-1} x(5r+3) W_{N/5}^{rk}$$

$$+ W_N^{4k} \sum_{r=0}^{N/5-1} x(5r+4) W_{N/5}^{rk} \qquad (D.4)$$

Each of the N/5 point DFTs in Eq (D.4) represents an N/5 length sequence and the W_N terms in front of the summations are the butterfly multipliers.

Eq (D.4) can be rewritten to reflect the N/5 point DFTs as:

$$X(k) = A(m) + W_N^{k} B(m) + W_N^{2k} C(m) + W_N^{3k} D(m) + W_N^{4k} E(m) \qquad (D.5)$$

For $N=5^2=25$ the Eq (D.5) representation is shown in Figure D.1 and uses a less cumbersome FFT notation (Rabiner and Gold, 1975). X(k) is obtained by evaluating Eq (D.5) as:

$$X(0) = A(0) + B(0) + C(0) + D(0) + E(0)$$

$$X(1) = A(1) + W_{25}^{1} B(1) + W_{25}^{2} C(1) + W_{25}^{3} D(1) + W_{25}^{4} E(1)$$

$$X(2) = A(2) + W_{25}^{2} B(2) + W_{25}^{4} C(2) + W_{25}^{6} D(2) + W_{25}^{8} E(2)$$

$$\vdots$$

$$X(6) = A(0) + W_{25}^{6} B(0) + W_{25}^{12} C(0) + W_{25}^{18} D(0) + W_{25}^{24} E(0)$$

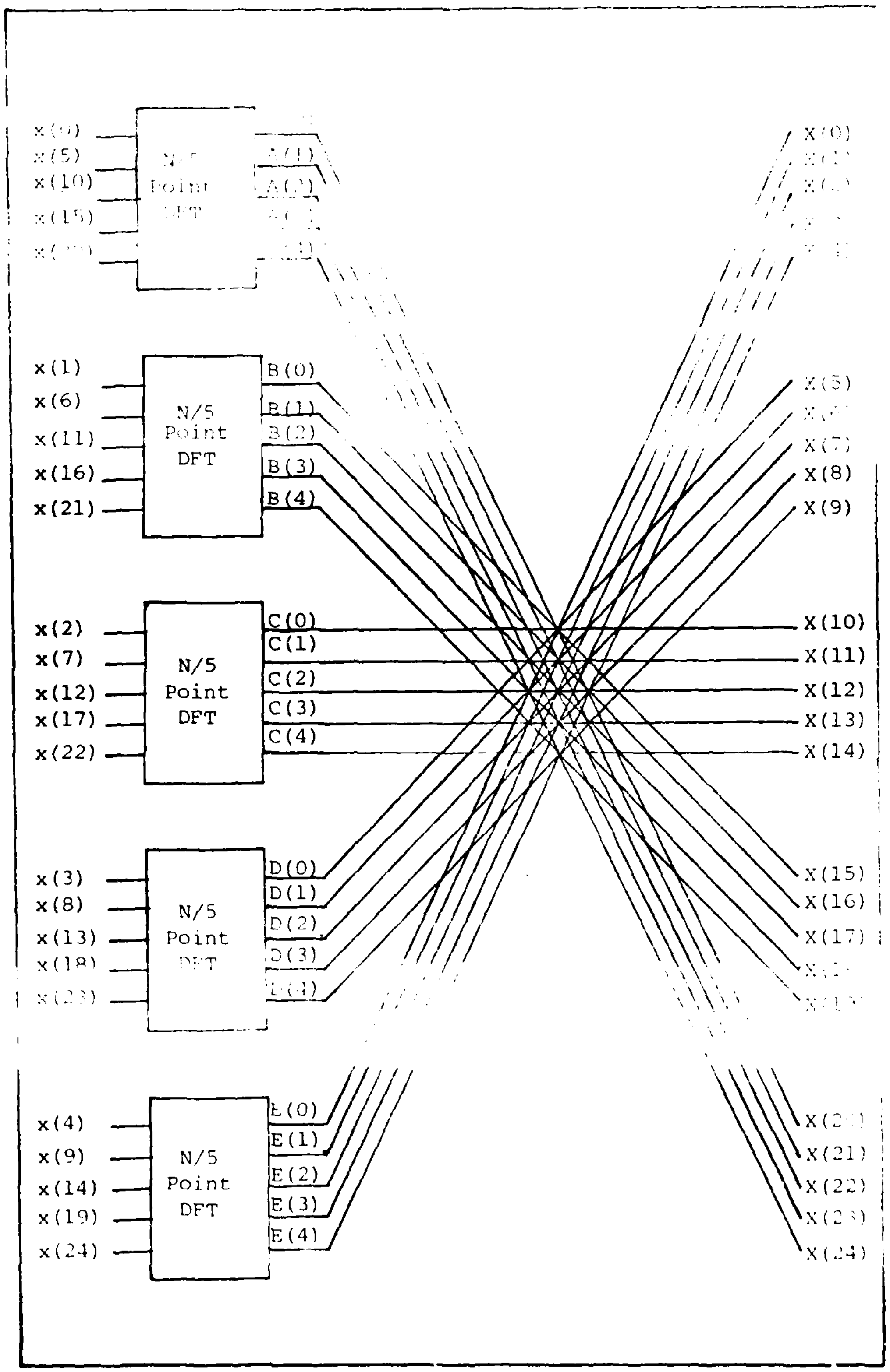


Figure D.1. First Stage Decimation for N=25.

$$X(7) = A(1) + W_{25}^{7}\ [illegible] + W_{25}^{14}\ [illegible] + W_{25}^{21}\ [illegible] + W_{25}^{28} E(1)$$

$$\vdots$$

$$X(23) = A(3) + W_{25}^{23} B(3) + W_{25}^{46} C(3) + W_{25}^{69} D(3) + W_{25}^{92} E(3)$$

$$X(24) = A(4) + W_{25}^{24} B(4) + W_{25}^{48} C(4) + W_{25}^{72} D(4) + W_{25}^{96} E(4)$$

The above expressions explicitly describe the first stage decimation for N=25. The next step is to evaluate A(m) - E(m) which are also 5-point DFTs. The 5-point DFT for A(m) can be evaluated as:

$$A(m) = \sum_{r=0}^{N/5-1} x(r) W_{N/5}^{rm} \tag{D.6}$$

which results in five N/25 length sequences:

$$A(m) = \sum_{i=0}^{N/25-1} a(5i) W_{N/25}^{5im} + W_{N/5}^{m} \sum_{i=0}^{N/25-1} a(5i+1) W_{N/25}^{5im}$$
$$+ W_{N/5}^{2m} \sum_{i=0}^{N/25-1} a(5i+2) W_{N/25}^{5im} + W_{N/5}^{3m} \sum_{i=0}^{N/25-1} a(5i+3) W_{N/25}^{5im}$$
$$+ W_{N/5}^{4m} \sum_{i=0}^{N/25-1} a(5i+4) W_{N/25}^{5im}$$
$$m = 0, 1, \ldots, 4 \tag{D.7}$$

It can be seen from Figure D.1 that a(5i) = x(0), a(5i+1) = x(5), a(5i+2) = x(10), a(5i+3) = x(15), and a(5i+4) = x(20) for the 5-point DFT of A(m). The final expression for the A(m) 5-point DFT is given from Eq (D.7) where N=25:

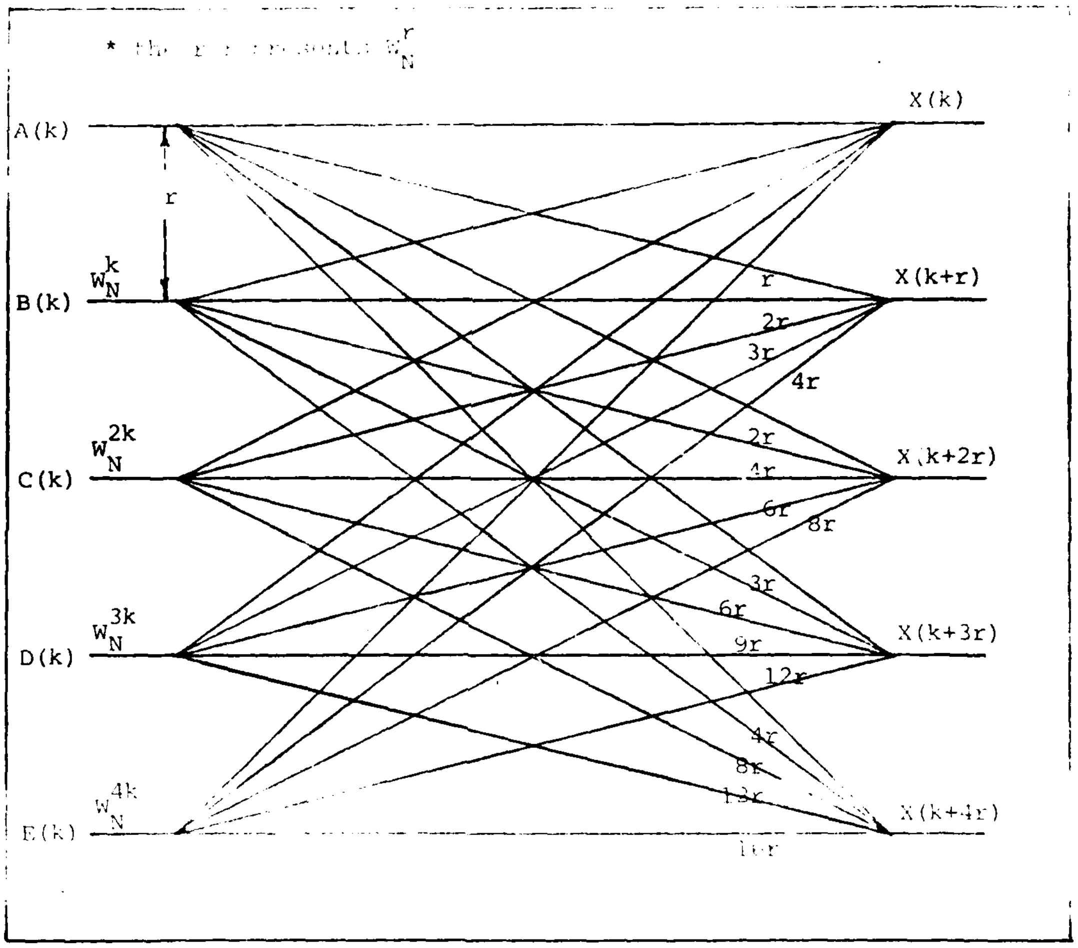


Figure D.2. Basic Radix-5 Butterfly Using Twiddle Factors.

$$A(0) = a(0) + W_5^0 a(1) + W_5^0 a(2) + W_5^0 a(3) + W_5^0 a(4) \quad (D.8)$$

$$A(1) = a(0) + W_5^1 a(1) + W_5^2 a(2) + W_5^3 a(3) + W_5^4 a(4) \quad (D.9)$$

$$A(2) = a(0) + W_5^2 a(1) + W_5^4 a(2) + W_5^6 a(3) + W_5^8 a(4) \quad (D.10)$$

$$A(3) = a(0) + W_5^3 a(1) + W_5^6 a(2) + W_5^9 a(3) + W_5^{12} a(4) \quad (D.11)$$

$$A(4) = a(0) + W_5^4 a(1) + W_5^8 a(2) + W_5^{12} a(3) + W_5^{16} a(4) \quad (D.12)$$

From Eqs (D.8) - (D.12) the basic butterfly multipliers are derived to be:

$$X(k) = A(k) + W_N^k B(k) + W_N^{2k} C(k) + W_N^{3k} D(k) + W_N^{4k} E(k) \quad (D.13)$$

$$X(k+r) = A(k) + W_N^{k+r} B(k) + W_N^{2k+2r} C(k) + W_N^{3k+3r} D(k) + W_N^{4k+4r} E(k) \quad (D.14)$$

$$X(k+2r) = A(k) + W_N^{k+2r} B(k) + W_N^{2k+4r} C(k) + W_N^{3k+6r} D(k) + W_N^{4k+8r} E(k) \quad (D.15)$$

$$X(k+3r) = A(k) + W_N^{k+3r} B(k) + W_N^{2k+6r} C(k) + W_N^{3k+9r} D(k) + W_N^{4k+12r} E(k) \quad (D.16)$$

$$X(k+4r) = A(k) + W_N^{k+4r} B(k) + W_N^{2k+8r} C(k) + W_N^{3k+12r} D(k) + W_N^{4k+16r} E(k) \quad (D.17)$$

The Eqs (D.13) - (D.17) are shown in the twiddle factor butterfly of Figure D.2 where "r" is the distance between the butterfly and points. Since N=5r the butterfly multipliers reduce to constant complex multipliers of:

$$W_N^{r} = W_N^{6r} = W_N^{16r} = \cos(2\pi/5) - j\sin(2\pi/5)$$

$$W_N^{2r} = W_N^{12r} = \cos(4\pi/5) - j\sin(4\pi/5)$$

$$W_N^{3r} = (W_N^{2r})^{*} = W_N^{8r} = \cos(4\pi/5) + j\sin(4\pi/5)$$

$$W_N^{4r} = (W_N^{r})^{*} = W_N^{9r} = \cos(2\pi/5) + j\sin(2\pi/5)$$

These constant butterfly multipliers are computed once during the FFT computation and used in every radix-5 butterfly.

Appendix E. Mixed Radix FFT Algorithm

This section presents an algorithm for computing the FFT based on the discrete Fourier transform:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N)$$

The algorithm described here can accept an N length sequence which is factorable by 2, 3, 4, or 5. To aid in selecting an appropriate length sequence for this algorithm a list of numbers less than 50,000 containing no prime factors larger than five is listed in Table E.

Arguments

A = The real portion of the complex data sequence to be transformed. It is dimensioned to length N.

B = Imaginary portion of the complex data sequence to be transformed. It is dimensioned to length N.

M = Number of factors of N.

WKC and WKS = Storage arrays dimensioned to length N and used for sine and cosine look up tables.

N = Length of the sequence to be transformed. N must be an integer power of 2, 3, 4, 5, or a combination thereof.

AT and BT = Arrays used in the subroutine for temporary storage of A and B during the data reordering (digit reversal).

NFAC = Contains all the factors of N. NFAC is computed by the user and passed to the subroutine in the argument list. Dimensioned to length M.

IWK - Contains the powers of 2, 3, 4, and 5 and is dimensioned to length 4.

```
IWK(1) = powers of 5
IWK(2) = powers of 4
IWK(3) = powers of 3
IWK(4) = powers of 2 (must be 0 or 1)
```

Usage. The subroutine listed permits a maximum of 11 factors which is adequate for any N less than 2^{16} with the factoring used by this subroutine.

(1) Dimension arrays A,B,AT,BT,WKC, and WKS to length N and array NFAC to length M.

(2) Factor N and store them in array NFAC. Array NFAC must contain the factors of N starting with the highest prime factor, 5, and continuing to the lowest, 2.

```
E.G.  N=480
NFAC(1) = 5, NFAC(2) = 4, NFAC(3) = 4
NFAC(4) = 3, NFAC(5) = 2.
```

(3) Specify the integer powers of 2, 3, 4, and 5 in the array IWK.

```
E.G.  N=480
IWK(1) = 1, IWK(2) = 2, IWK(3) = 1, IWK(4) = 1
```

In general,

$$N = 2^m \; 3^n \; 4^p \; 5^q \quad \text{and}$$

IWK(1) = q, IWK(2) = p, IWK(3) = n, IWK(4) = m.

(4) Specify values for A the real part of data sequence and B the imaginary part of the data sequence.

(5) Call FFTMR(A,B,M,N,WKC,WKS,AT,BE,NFAC,IWK).

(6) A and B contain the real and imaginary part of the transform X(b).

To aid the [illegible] selecting appropriate length N sequences, Table B.1 provides possible choices for N less than 50000 which have integer powers of 2, 3, 4, or 5 or combinations thereof.

TABLE II.

[illegible] THAN 5

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
320	324	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
600	625	640	648	675
720	729	750	768	800
810	864	900	960	972
1000	1024	1080	1125	1152
1200	1215	1250	1280	1296
1350	1440	1458	1500	1536
1600	1620	1728	1800	1875
1920	1944	2000	2025	2048
2160	2187	2250	2304	2400
2430	2500	2560	2592	2700
2880	2916	3000	3072	3125
3200	3240	3375	3456	3600
3645	3750	3840	3888	[illegible]
4050	4096	4320	4374	[illegible]
4608	4800	4860	5000	5120
5184	5400	5625	5760	5832
6000	6075	6144	6250	[illegible]
6480	6561	6750	6912	7200
7290	7500	7680	7776	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
13500	13824	14400	14580	15000
15360	15552	15625	16000	16200
16384	16875	17280	17496	18000
18225	18432	18750	19200	19440
19683	20000	20250	20480	20736
21600	21870	22500	23040	23328
24000	24300	24576	25000	25600
25920	26244	27000	27648	28125
28800	29160	30000	30375	30720
31104	31250	32000	32400	32768
32805	33750	34560	34992	36000
36450	36864	37500	38400	38880
39366	40000	40500	40960	41472
43200	43740	45000	46080	46656
46875	48000	48600	49152	50000

```
[illegible]
[illegible]
[illegible]
[illegible]  THE [illegible] VALUE TO [illegible]
   240=C  ON OUTPUT A IS REPLACED BY THE [illegible] TRANSFORM.
[illegible]
[illegible]  B: REAL [illegible]
   270=C  THE IMAGINARY VALUE TO BE TRANSFORMED.
   280=C  ON OUTPUT B IS REPLACED BY THE FOURIER TRANSFORM.
   290=C  B IS DIMENSIONED TO LENGTH N
   300=C  M: NUMBER OF FACTORS N.
   310=C  N: THE LENGTH OF THE SEQUENCE TO BE TRANSFORMED. N MUST BE AN
   320=C  INTEGER POWER OF 2,3,4, OR5.
   330=C  WKC: DIMENSIO D TO LENGTH N AND CONTAINS THE COSINE TERMS FOR
FFTMR.
   340=C  WKS: DIMENSIONED TO LENGTH N AND CONTAIN THE SINE TERMS FOR TH
E FFT.
   350=C  AT: DIMENSIONED TO LENGTH N AND IS USED TO STORE THE A ARRAY D
URING
   360=C  SHUFFLE
   370=C  BT: DIMENSIONED TO LENGTH N AND IS USED TO STORE THE B A[illegible] D
URING
   380=C  SHUFFLE
   390=C  NFAC: CONTAINS THE FACTORS OF N. MUST BE PASSED TO THIS [illegible]
TIME
   400=C  IN THE ARGUMENT LIST. THE FACTORS [illegible] LISTED IN NFA[illegible]
THE
[illegible]
[illegible]

   440=C      IWK(2)= [illegible]
   450=C      IWK(3)= POWER OF [illegible]
   460=C      IWK(4)= POWER OF 2. MUST BE [illegible] OR [illegible] ONLY.
   470=C
   480=C  CALLS : NONE
   490=C  AUTHOR: JOHN D. BLANKEN, CAPT,USAF          31 JULY 80
   500=C
   510=C-----------------------------------------------------------------
-----
   520=       DIMENSION A(N),B(N),WKC(N),WKS(N),IWK(4),AT(N),BT(N)
   530=       DIMENSION NCOUNT(11),NBASE(11),NDIGIT(11),NFAC(M)
   540=       INTEGER R,TYPE,SAME,D
```

```
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
740=C      SHUFFLE THE INPUT ARRAY TO REVERSE DIGIT ORDER
750=C
760=       SHIN=SECOND(CP)
770=       DO 135 I=1,N
780=       AT(I)=A(I)
790= 135   BT(I)=B(I)
800=C      COMPUTE THE BASE NUMBERS OF THE COUNTER
810=C
820=       NFAC=M
830=       NBASE(1)=1
840=       DO 100 J=2,M
850=       NBASE(J)=NFAC(NFAC)*NBASE(J-1)
860= 100   NFAC=NFAC-1
870=C
880=C      COMPUTE THE COUNTER LIMITS OF EACH DIGIT
890=C
900=       DO 110 J=1,M
910=110    NDIGIT(J)=NFAC(J)-1
920=C
930=C      ZERO THE NCOUNT(I) ARRAY
[illegible]


[illegible]
990=       ICOUNT=1
1000=C
1010=C     COMPUTE THE DIGIT REV VALUE OF N
1020=C
1030= 115  NREV=1
1040=      K=M
1050=      DO 130 I=1,M
1060=      NREV=NREV+NCOUNT(K)*NBASE(I)
1070= 130  K=K-1
1080=C
1090=C     CHECK IF SHUFFLE IS NECESSARY ON THIS PAIR
1100=C
```

```
[illegible]
[illegible]
[illegible]
1250= 150   CONTINUE
1260=       IROUT=[illegible]
[illegible]

1290=888    PRINT*, [illegible]
1300=C
1310=C      ARRAY HAS BEEN SHUFFLED
1320=C
1330=C
1340=C      COMPUTE THE TRANSFORM                              RADIX 5 SECTION
1350=C
1360=C      ARE THERE POWERS OF 5?
1370=       PRINT*,'IWK(1)= ',IWK(1)
1380=       FFTIN=SECOND(CP)
1390=       IF (IWK(1).LE.0) GO TO 200
1400=C  DEFINE  RF CONSTANTS
1410=       COS2=.30901699437494
1420=       SIN2=.95105651629515
1430=       COS4=-.80901699437494
1440=       SIN4=.58778525229247
1450=       LM1=IWK(1)-1
1460=       IWK1=IWK(1)
1470=       DO 170 L5=1,IWK1
1480=C
[illegible]

1540=       R=0.5
1550=       TYPE=R
1560=C
1570=C      INITIALIZE TWIDDLE FACTOR (TF)
1580=       TFA1=1.
1590=       TFA2=1.
[illegible]
1610=       TFA4=1.
1620=       TFA5=1.
1630=       TFB1=0.
1640=       TFB2=0.
1650=       TFB3=0.
```

```
[illegible]
[illegible]
[illegible]

      TFB2=-WKS(JM1*K1+1)
      TFA3=WKC(JM1*K2+1)
      TFB3=-WKS(JM1*K2+1)
      TFA4=WKC(JM1*K3+1)
      TFB4=-WKS(JM1*K3+1)
      TFA5=WKC(JM1*K4+1)
      TFB5=-WKS(JM1*K4+1)
   60 CONTINUE
C
C     THIS LOOP PERFORMS THE 5-PT DFT. THE LOOP IS INCREMENTED BY
C     DISTANCE D WHICH SELECTS THE NEXT BF WITH THE SAME TF:
      DO 130 I1=J,N,D
      I2=I1+P
      I3=I2+P
      I4=I3+P
      I5=I4+P
C
C  TWIDDLE THE BF INPUTS AND STORE RESULTS IN TEMP LOCATI[illegible]
C
      A1T=A(I1)
      B1T=B(I1)
[illegible]
[illegible]
[illegible]

      B2T=(A(I2)*TFB2)+(B(I2)*TFA2)
      A3T=(A(I3)*TFA3)-(B(I3)*TFB3)
       B3T=(A(I3)*TFB3)+(B(I3)*TFA3)
      A4T=(A(I4)*TFA4)-(B(I4)*TFB4)
      B4T=(A(I4)*TFB4)+(B(I4)*TFA4)
      A5T=(A(I5)*TFA5)-(B(I5)*TFB5)
       B5T=(A(I5)*TFB5)+(B(I5)*TFA5)
      GO TO 62
   61 A2T=A(I2)
      B2T=B(I2)
      A3T=A(I3)
      B3T=B(I3)
      A4T=A(I4)
```

```
[illegible]
2370=        [illegible]
[illegible]
2390=        C4A2A5=COS4◆A2PA5
2400=        C2A3A4=COS2◆A3PA4
2410=        S4B2B5=SIN4◆B2MB5
2420=        S2B3B4=SIN2◆B3MB4
2430=        C4B3B4=COS4◆B3PB4
2440=        C2B2B5=COS2◆B2PB5
2450=        S4A4A3=SIN4◆A4MA3
2460=        S2A5A2=SIN2◆A5MA2
2470=        C4B2B5=COS4◆B2PB5
2480=        C2B3B4=COS2◆B3PB4
2490=        S4A5A2=SIN4◆A5MA2
2500=        S2A4A3=SIN2◆A4MA3
2510=        CA25=A1T+C4A3A4+C2A2A5
2520=        CA34=A1T+C4A2A5+C2A3A4
2530=        SA25=S4B3B4+S2B2B5
2540=        SA34=S4B2B5-S2B3B4
2550=        CB25=B1T+C4B3B4+C2B2B5
2560=        CB34=B1T+C4B2B5+C2B3B4
2570=        SB25=S4A4A3+S2A5A2
2580=        [illegible]



[illegible]
2640=        B(I1)=B1T+B3PB4+B2PB5
2650=        B(I2)=CB25+SA25
2660=        B(I3)=CB34+SA34
2670=        B(I4)=CB34-SA34
2680=        B(I5)=CB25-SA25
2690= 160    CONTINUE
2700= [illegible]
2710= 170    CONTINUE
2720=C                                                  END RADIX 5
2730=C                                                  RADIX 4 SECTION
2740= 200    PRINT◆,'IWK(2)=',IWK(2)
2750=C       ARE THERE ANY POWERS OF 4?
2760=        IF(IWK(2).LE.0) GO TO 300
```

```
      [illegible]
      [illegible]
C     COMPUTE THE INDEX CONSTANTS
      K1=SAME
      K2=2*K1
      K3=3*K1
      DO 230 J=1,TYPE
      IF(IWK(I).NE.0) GO TO 210
      IF(L4.EQ.1) GO TO 211
210   IF(J.EQ.1) GO TO 211
      JM1=J-1
      TFA2=WKC(JM1*K1+1)
      TFB2=-WKS(JM1*K1+1)
      TFA3=WKC(JM1*K2+1)
      TFB3=-WKS(JM1*K2+1)
      TFA4=WKC(JM1*K3+1)
      TFB4=-WKS(JM1*K3+1)
211   CONTINUE
C
C     THIS LOOP PERFORMS THE 4 PT DFT
      DO 230 I1=J,N,D
      I2=I1+P
      [illegible]
      [illegible]
      IF(IWK(I).NE.0) GO TO 213
      IF(L4.EQ.1) GO TO 215
213   IF(J.EQ.1) GO TO 215
      A2T=A(I2)*TFA2-B(I2)*TFB2
      B2T=A(I2)*TFB2+B(I2)*TFA2
      [illegible]
      [illegible]
      A4T=A(I4)*TFA4-B(I4)*TFB4
      B4T=A(I4)*TFB4+B(I4)*TFA4
      GO TO 217
215   A2T=A(I2)
      B2T=B(I2)
```

```
          [illegible]
          [illegible]
          A(I4)=A1MA3-B2MA4
          B(I4)=B1MB3+A2MA4
  230     CONTINUE
  280     CONTINUE
  270     CONTINUE
          PRINT*,"RADIX 4 DONE"                         END RADIX 4
C                                                       RADIX 3 SECTION
C
C
  300     PRINT*,"IWK(3)=",IWK(3)
C         ARE THERE ANY POWERS OF 3?
          IF(IWK(3).LE.0) GO TO 400
          CONST=.866025[illegible]7844
          LM1=IWK(3)-1
          IWK3=IWK(3)
          DO 370 L3=1,IWK3
          IAME=(3**LM1)*(2**IWK(4))
          LM1=LM1-1
          [illegible]
          [illegible]

          TFA[illegible]
          TFA[illegible]
          K1=IAME
          K2=2*K1
          DO 380 J=1,IYFF
          IF(IWK(1)[illegible]
          IF(IWK[illegible]
          IF(L3.EQ.1) GO TO 311
  310     IF(J.EQ.1) GO TO 311
          JM1=J-1
          TFA2=WKC(JM1*K1+1)
```

```
[illegible]
[illegible]
[illegible]
[illegible]
      A3T=A(I[illegible]*TF[illegible]
      B3T=A[illegible]*TF[illegible]
      GO TO 3[illegible]
315   A2T=A(I2)
      B2T=B(I2)
      A3T=A(I3)
      B3T=B(I3)
317   A2A3=A2T+A3T
      B2B3=B2T+B3T
      A(I1)=A1T+A2A3
      B(I1)=B1T+B2B3
      PAW2=CONST*(B3T-B2T)
      PBW2=CONST*(A2T-A3T)
      PAW1=A1T-0.5*A2A3
      PBW1=B1T-0.5*B2B3
      A(I2)=PAW1-PAW2
      B(I2)=PBW1-PBW2
      A(I3)=PAW1+PAW2
      B(I3)=PBW1+PBW2
390   CONTINUE
380   CONTINUE
370   CONTINUE
      PRINT*,"RADIX 3 DONE"
C                                                     END RADIX 3
C                                                     [illegible]
C
400   PRINT*,[illegible]
      IF(I[illegible]
      D=N
      SAME=1
      P=D/2
      TYPE=P
      TFA1=1.
      TFB1=0.
      TFA2=1.
      TFB2=0.
      K1=SAME
      DO 430 J=1,TYPE
      IF(J.EQ.1) GO TO 411
      JM1=J-1
      TFA2=WKC(JM1*K1+1)
```

```
          TFB2=-WKI(JM1*K1+1)
     411  I1=J
          I2=I1+P
          A1T=A(I1)
          B1T=B(I1)
          IF(J.EQ.1) GO TO 415
          A2T=A(I2)*TFA2-B(I2)*TFB2
          B2T=A(I2)*TFB2+B(I2)*TFA2
          GO TO 417
     415  A2T=A(I2)
          B2T=B(I2)
     417  A(I1)=A1T+A2T
          B(I1)=B1T+B2T
          A(I2)=A1T-A2T
          B(I2)=B1T-B2T
     480  CONTINUE
          PRINT*,"RADIX 2 DONE"
C                                               END RADIX 2
          FFTOUT=SECOND(CP)-FFTIN
          PRINT*,"TIME TO PERFORM FFT=",FFTOUT
C
C         END OF FFTMP SUBROUTINE
     500  RETURN
          END
*EOP
*EOF
```

Real Operations Count for FFTMR

The operations count for the factorization used in this algorithm is a function of (1) the number of butterflies, (2) the number of complex twiddle factors, and (3) the number of times the cosine and sine difference equations must be computed. The number of butterflies in a mixed radix algorithm has been shown to be (Singleton, 1969):

$$\sum_{i=1}^{m} (N/p_i) \tag{E.1}$$

and the number of complex twiddle factors is:

$$\sum_{i=1}^{m} (N(p_i-1)/p_i) - (N-1) \tag{E.2}$$

where $N=p_1 p_2 \ldots p_m$. The radices in this algorithm are restricted to:

$$N = 2^r \, 3^s \, 4^t \, 5^u \tag{E.3}$$

Given the factorization in Eq (E.3) the radix-2 section (where p=2) has

$$\sum_{i=1}^{r} (N/p_i) = \sum_{i=1}^{r} (N/2) = rN/2 \tag{E.4}$$

butterflies which require four real additions each. The number of complex twiddle factors for the radix-2 is given as:

$$\sum_{i=1}^{r} (N(p_i-1)/p_i) = \sum_{i=1}^{r} (N/2) = rN/2 \tag{E.5}$$

which requires four real multiplications and two real additions each. Notice that the N-1 term has not been subtracted as in Eq (E.2). The N-1 term will be subtracted after the total operations count has been derived for 3, 4, and 5 factors and combined with factors of 2. Using Eqs (E.4) - (E.5) and the number of additions and multiplications required for each provides the operations count for the radix-2 section as:

$$\text{real mult} = 4(rN/2) = 2rN \qquad (E.6)$$

$$\text{real adds} = 4(rN/2) + 2(rN/2) = 3rN \qquad (E.7)$$

The radix-3 section requires 4 real multiplications and 12 real additions per butterfly and 4 real multiplications and two additions per complex twiddle factor. Using Eqs (E.1) and (E.2) the number of butterflies for p=3 is:

$$\sum_{i=1}^{s} (N/p_i) = \sum_{i=1}^{s} (N/3) = sN/3 \qquad (E.8)$$

and the number of twiddle factor (neglecting the N-1 term) is:

$$\sum_{i=1}^{s} (N(p_i-1)/p_i) = 2sN/3 \qquad (E.9)$$

Combining the additions and multiplications required for each butterfly and twiddle factor with Eqs (E.8) - (E.9) gives the operations count for the radix-3 section as:

$$\text{real mult} = 4(sN/3) + 4(2sN/3) = 4sN \qquad (E.10)$$

$$\text{real adds} = 12(sN/3) + 2(2sN/3) = 16sN/3 \qquad (E.11)$$

The radix-4 section has zero real multiplications and 16 real additions per butterfly with 4 real multiplications and 2 real additions per twiddle factor. The number of butterflies, where p=4, is given by:

$$\sum_{i=1}^{t} (N/p_i) = \sum_{i=1}^{t} (N/4) = tN/4 \qquad \text{(E.12)}$$

the number of twiddle factors is:

$$\sum_{i=1}^{t} (N(p_i-1)/p_i) = \sum_{i=1}^{t} (3N/4) = 3tN/4 \qquad \text{(E.13)}$$

Using the number of multiplications and additions per butterfly and twiddle factor in Eqs (E.12) - (E.13) gives the total operations for factors of 4 as:

$$\text{real mult} = 4(3tN/4) = 3tN \qquad \text{(E.14)}$$

$$\text{real adds} = 16(tN/4) + 2(3tN/4) = 11tN/2 \qquad \text{(E.15)}$$

The radix-5 section requires 16 real multiplications and 32 additions per butterfly with 4 real multiplications and 2 additions per twiddle factor. Using Eqs (E.1) and (E.2) where p=5 gives the total butterflies as:

$$\sum_{i=1}^{u} (N/p_i) = \sum_{i=1}^{u} (N/5) = uN/5 \qquad \text{(E.16)}$$

and the number of twiddle factors as:

$$\sum_{i=1}^{u} (N(p_i-1)/p_i) = \sum_{i=1}^{u} (4N/5) = 4uN/5 \qquad \text{(E.17)}$$

Combining Eqs (E.16) - (E.17) with the operations required for butterflies and twiddle factor in the radix-5 section gives the total as:

$$\text{real mult} = 16(uN/5) + 4(4uN/5) = 32uN/5$$
$$\text{real adds} = 32(uN/5) + 2(4uN/5) = 8uN \quad (E.18)$$

Using the results of Eqs (E.4) - (E.18) and subtracting the N-1 complex twiddles provides the number of real operations used for butterflies and twiddle factors for the mixed radix algorithm. The expressions are:

$$\text{real mult} = 2rN + 4sN + 3tN + 32uN/5 - 4(N-1) \quad (E.19)$$

$$\text{real adds} = 3rN + 16sN/3 + 11tN/2 + 8uN - 2(N-1) \quad (E.20)$$

Recall that Eqs (E.19) - (E.20) account for only two of the three sources of real operations in this algorithm. The third source is computing the sine and cosine look up table. From the FORTRAN program in this appendix the expressions computing the look up table are:

```
WKC(I) = C*WKC(I-1) - S*WKS(I-1) + WKC(I-1)   (E.21)
WKS(I) = (*WKS(I-1) + S*WKC(I-1) + WKS(I-1)   (E.22)
```

Each equation requires 5 real additions and 2 real multiplications and they are computed N-1 times for the mixed radix FFT. The real operations required to compute the look up table are:

$$\text{real mult} = 4(N-1) \quad (E.23)$$

$$\text{real adds} = 10(N-1) \quad (E-24)$$

Combining Eqs (E.23) - (E.24) with the real operations for butterflies and twiddle factors provides the total real operations for the mixed radix FFT:

$$\begin{aligned} \text{real mult} &= 2rN + 4sN + 3tN \\ &\quad + 32uN/5 - 4(N-1) + 4(N-1) \\ &= 2rN + 4sN + 3tN + 32uN/5 \end{aligned} \tag{E.25}$$

$$\begin{aligned} \text{real adds} &= 3rN + 16sN/3 + 11tN/2 \\ &\quad + 8uN - 2(N-1) + 10(N-1) \\ &= 3rN + 16sN/3 + 11tN/2 \\ &\quad + 8uN + 8(N-1) \end{aligned} \tag{E.26}$$

Development of the Mixed Radix Digit-Reversed Algorithm

Assuming that the number of points to be transformed satisfies $N=r_1, r_2, \ldots, r_m$, where $r_1, r_2, \ldots, r_m$ are integer values, the indices of x(n) and X(k) can be expressed as (Brigham, 1974):

$$n = n_{m-1}(r_2 r_3 \cdots r_m) + n_{m-2}(r_3 r_4 \cdots r_m) + n_1 r_m + n_0 \qquad \text{(E.27)}$$

$$k = k_{m-1}(r_1 r_2 \cdots r_{m-1}) + k_{m-2}(r_1 r_2 \cdots r_{m-2}) + k_1 r_1 + k_0 \qquad \text{(E.28)}$$

where

$$k_{i-1} = 0, 1, 2, \ldots r_i - 1 \qquad i \leq i \leq m$$

$$n_i = 0, 1, 2, \ldots r_{m-i} - 1 \qquad 0 \leq i \leq m-i$$

For $N=30 = 2x3x5 = r_1 r_2 r_3$ and $m=3$ the input sequence x(n) counter is:

$$n = n_2\,(15) + n_1\,(5) + n_0 \qquad \text{(E.29)}$$

where

$$n_0 = 0, 1, 2, 3, 4$$

$$n_1 = 0, 1, 2$$

$$n_2 = 0, 1$$

The output counter k for X(k) is:

$$k = k_2\,(6) + k_1\,(2) + k_0$$

where

$k_0 = 0, 1$

$k_1 = 0, 1, 2$

$k_2 = 0, 1, 2, 3, 4$

To implement the general digit reversed counter let the input counter n use the digit reversed multipliers of the output counter k:

$$n = n_{m-1} + n_{m-2} (r_1) + \dots + n_1 (r_1 r_2 \dots r_{m-2}) + n_0 (r_1 r_2 \dots r_{m-1}) \qquad (E.30)$$

For the example $r_1 r_2 r_3 = 2x3x5 = 30$ the digit reversed counter becomes:

$$n = n_2 + 2n_1 + 6n_0 \qquad (E.31)$$

where, as before:

$n_0 = 0, 1, 2, 3, 4$

$n_1 = 0, 1, 2$

$n_2 = 0, 1$

Appendix F. Singleton's Mixed Radix FFT

This program was written by R.C. Singleton and published by the IEEE press in "Programs for Digital Signal Processing". It computes the DFT defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N)$$

It also computes the 1/N scaled inverse Fourier transform.

The subroutine listed in this appendix factors N into "square" and "square-free" factors and stores the results in an array NFAC. It then calls subroutine FFTMX to compute the complex Fourier transform, twiddle the data, and reorder the complex array to final order.

Use of this subroutine for multi-variate transforms is described in the comments section at the beginning of the program. A multi-variate transform is basically a single-variate transform with modified indexing (Singleton, 1977).

The subroutine listed permits the sequence length that has 15 or fewer factors.

The smallest number that has more than 15 factors is 12,754,584 and if this condition is encountered an error message is printed.

The transform portion of the subroutine includes sections for factors of 2, 3, 4, or 5 as well as a general section for odd prime factors. The special sections for 2 and 4 include the twiddle factor multiplication in these special sections instead of using the general twiddle factor section. "Performing the transform in this manner produces a 10 percent speed improvement over the general twiddle section" (Singleton, 1969). The special sections for 3 and 5 are similar to the general odd factor section but reduce the indexing required and thus improve the speed (Singleton, 1969).

Arguments. The Singleton FFT for computing a complex single-variate transform is called using the following arguments:

A = The real part of the array to be transformed and is dimensioned to length N.

B = The imaginary part of the array to be transformed and is dimensioned to length N.

N = Length of the input sequence N which must be a positive integer with no more than 15 factors.

NSPN = The spacing of consecutive data values while indexing the current variable (in units determined by the magnitude of ISN).

ISN = The sign of ISN determines the transform direction (negative for forward and positive for inverse). The magnitude of ISN determines the indexing increment for arrays A and B. Normally the magnitude of ISN is unity.

NSEG = An integer value such that NSEG x N x NSPN equals the total number of complex data values.

<u>Usage</u>. For a single-variate forward transform:

(1) Specify the input sequences A and B and parameters NSEG=1, N=transform length, NSPN=1, and ISN= -1.

(2) Dimension A and B to length N.

(3) Call FFTSNG (A,B,NSEG,N,NSPN,ISN).

(4) A and B are the output real and imaginary portion of the complex vector X(b).

To perform a real valued, inverse, or multi-variate transform refer to the comments portion of FFTSNG.

```
   [illegible]
   [illegible]
   [illegible]
   [illegible]
   [illegible]= --------------------------------------------------------------
-------
   [illegible]
   [illegible]
   230=  [illegible] MIXED-RADIX FAST FOURIER TRANSFORM ALGORITHM.
   240=C----------------------------------------------------------------------
-------
   250=C
   260=C
   270=C ARRAYS A AND B ORIGINALLY HOLD THE REAL AND IMAGINARY
   280=C       COMPONENTS OF THE DATA, AND RETURN THE REAL AND
   290=C       IMAGINARY COMPONENTS OF THE RESULTING FOURIER COEFFICIENTS
.
   300=C MULTIVARIATE DATA IS INDEXED ACCORDING TO THE FORTRAN
   310=C       ARRAY ELEMENT SUCCESSOR FUNCTION, WITHOUT LIMIT
   320=C       ON THE NUMBER OF IMPLIED MULTIPLE SUBSCRIPTS.
   330=C       THE SUBROUTINE IS CALLED ONCE FOR EACH VARIATE.
   340=C       THE CALLS FOR A MULTIVARIATE TRANSFORM MAY BE IN ANY [illegible]
.
   350=C
   360=C N IS THE DIMENSION OF THE CURRENT VARIABLE.
   370=C NSPN IS THE SPACING OF CONSECUTIVE DATA VALUES
   380=C       WHILE INDEXING THE CURRENT VARIABLE.
   390=C NSEG*N*NSPN IS THE TOTAL NUMBER OF COMPLEX DATA VALUES.
   [illegible]
   410= [illegible]
   [illegible]
   [illegible]
   440= IF FFT IS CALLED TWICE, WITH OPPOSITE SIGNS ON ISN, AN
   4[illegible]=       [illegible] TRANSFORMATION IS [illegible]...CALLS CAN BE IN [illegible]
[illegible]ORDER.
   460=       THE RESULTS ARE SCALED BY 1/N WHEN THE SIGN OF ISN
[illegible]TIVE.
   4[illegible]
   4[illegible] [illegible] WITH [illegible]
   4[illegible] [illegible]
   500=          CALL FFT(A,B,N2*N3,N1,1,-1)
   510=          CALL FFT(A,B,N[illegible],N2,N1,-1)
   520=          CALL FFT(A,B,1,N3,N1*N2,-1)
   530=
```

[illegible]

[illegible]

```
700=C      DOUBLE PRECISION DSTAK
710=C      INTEGER ISTAK(5000)
720=C      REAL RSTAK(5000)
730=C
740=C      EQUIVALENCE (DSTAK(1),ISTAK(1))
750=C      EQUIVALENCE (DSTAK(1),RSTAK(1))
760=C
770=C DETERMINE THE FACTORS OF N
780=C
790=       M = 0
800=       NF = IABS(N)
810=       K = NF
820=       IF (NF.EQ.1) RETURN
830=       NSPAN = IABS(NF*NSPAN)
840=       NTOT = IABS(NSPAN*NTEG)
850=       IF (ISN*NTOT.NE.0) GO TO 20
860=       IERR = I1MACH(4)
870=       WRITE (IERR,9999) NSEG, N, NSPAN, [illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

1020=  40  IF (MOD(K,J).EQ.0) GO TO 50
1030=      J = J + 2
1040=      JJ = J**2
1050=      IF (JJ.LE.K) GO TO 40
1060=      IF (K.GT.4) GO TO 50
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]       IF ([illegible].LE.[illegible]) MA[illegible] = M + [illegible] + 1
1260=        IF (M+KT.GT.15) GO TO 120
1270=        IF (KT.EQ.0) GO TO 110
1280=        J = KT
1290= 100    M = M + 1
1300=        NFAC(M) = NFAC(J)
1310=        J = J - 1
1320=        IF (J.NE.0) GO TO 100
1330=C
1340= 110    MAXF = M - KT
1350=        MAXF = NFAC(MAXF)
1360=        IF (KT.GT.0) MAXF = MAX0(NFAC(KT),MAXF)
1370=C       [illegible] = [illegible](MAXF*4)
1380=C       [illegible] = [illegible] + MAXF
1390=C       [illegible] = [illegible] + MAXF
1400=        [illegible] = [illegible] + MAXF
1410=C       K = IINTGT(MAXF,2)
1420=C       K = IINTGT(MAXF,2)
1430=C       CALL FFTMX(A, B, NTOT, NF, N, NSPAN, ISN, M, [illegible], [illegible]
1440=C     *      RSTAK([illegible]), RSTAK([illegible]), [illegible], [illegible]
[illegible]

1490=
1490= 120    IERR = I1MACH(4)
1500=        WRITE (IERR,[illegible])
1510= 999    FORMAT (50H ERROR - FFT PARAMETER N HAS MORE THAN [illegible]
 -)
1520= 130    PRINT*,"N HAS MORE THAN 15 FACTORS"
1530=        RETURN
1540=        END
1550=
1560=        SUBROUTINE FFTMX(A,B,NTOT,N,NSPAN,ISN,M,KT,AT,CK,BT,[illegible]
1570=       CT,NF,NFM,NFAC)
1580=C-------------------------------------------------------------------------
-------
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
      ISN = [illegible]
      IF (ISN.GT.0) GO TO 10
      S72 = -S72
      S120 = -S120
      RAD = -RAD
      GO TO 30
C
C SCALE BY 1/N FOR ISN .GT. 0
C
   10 AK = 1.0/FLOAT(N)
      DO 20 J=1,NT,INC
        A(J) = A(J)*AK
        B(J) = B(J)*AK
   20 CONTINUE
C
   30 KSPAN = KS
      NN = NT - INC
      JC = KS/N
C
C SIN, COS VALUES ARE RE-INITIALIZED [illegible]
[illegible]
      [illegible] = [illegible]
[illegible]
      JF = 0
      MAXF = M - KT
      MAXF = NFAC(MAXF)
      IF (KT.GT.0) MAXF = MAX0(NFAC(KT),MAXF)
C
C COMPUTE FOURIER TRANSFORM
C
   40 DR = [illegible]*FLOAT(JC)/FLOAT(KSPAN)
      CD = 2.0*SIN(0.5*DR*RAD)**2
      SD = SIN(DR*RAD)
      KK = 1
      I = I + 1
      IF (NFAC(I).NE.2) GO TO 110
```

```
[illegible]
          I = [illegible]
          [illegible] 1 [illegible]LIM[illegible]
          [illegible] TO [illegible]
          [illegible] = [illegible] 1 - [illegible]D*C1+[illegible]D*[illegible]1)
          [illegible] = [illegible]D*[illegible]1-[illegible]D*[illegible]1) + [illegible]1
[illegible]ING THREE STATEMENTS COMPENSATE FOR TRUNCATION
[illegible] IF ROUNDED ARITHMETIC IS USED, SUBSTITUTE
[illegible]
          [illegible]**[illegible] 1**[illegible] + 0.5[illegible]
          [illegible]
          [illegible]
[illegible]
          [illegible]RAD[illegible]
          [illegible]
          [illegible]
          [illegible]
          [illegible]
[illegible]
          [illegible] 1
          [illegible]
          [illegible] GO TO [illegible]
          [illegible]
          [illegible] GO TO 70
          [illegible] GO TO 40
          [illegible]
2610=     [illegible] = [illegible]RAD[illegible] 2 + [illegible]
2620=     IF [illegible].LE.[illegible]+J[illegible] GO TO 60
2630=     GO TO 40
2640=  40  1 = FLOAT([illegible]-1)/[illegible])*DR*RAD
2650=     C1 = COS([illegible]1)
2660=     S1 = SIN(S1)
```

```
[illegible]
[illegible]
[illegible]
2850=        B(K1) = BK + AJ
2860=        A(K2) = AK + BJ
2870=        B(K2) = BK - AJ
2880=        KK = K2 + KSPAN
2890=        IF (KK.LT.NN) GO TO 100
2900=        KK = KK - NN
2910=        IF (KK.LE.KSPAN) GO TO 100
2920=        GO TO 290
2930=C
2940=C  TRANSFORM FOR FACTOR OF 4
2950=C
2960= 110   IF (NFAC(I).NE.4) GO TO [illegible]
2970=        KSPNN = KSPAN
2980=        KSPAN = KSPAN/4
2990= 120   C1 = 1.0
3000=        S1 = 0
3010=        MM = MIN0([illegible]
3020=        GO TO 150
[illegible]
3040=        [illegible]
[illegible]
3100=        C1 = 0.5/([illegible]**2+[illegible]**2) + 0.5
[illegible]
[illegible]
[illegible]
[illegible] 140  [illegible]
[illegible]
[illegible]
3170=        [illegible] = C2*C1 + S2*S1
3180= 150   K1 = KK + KSPAN
3190=        K2 = K1 + KSPAN
3200=        K3 = K2 + KSPAN
3210=        AKP = A(KK) + A(K2)
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]         IF (S1.EQ.0.0) GO TO [illegible]
[illegible]   [illegible]   A(K1) = [illegible]
[illegible]         B(K1) = [illegible]
3410=         A(K2) = AJP*C2 - BJP*S2
3420=         B(K2) = AJP*S2 + BJP*C2
3430=         A(K3) = AKM*C3 - BKM*S3
3440=         B(K3) = AKM*S3 + BKM*C3
3450=         KK = K3 + KSPAN
3460=         IF (KK.LE.NT) GO TO 150
3470= 170     KK = KK - NT + JC
3480=         IF (KK.LE.MM) GO TO 130
3490=         IF (KK.LT.KSPAN) GO TO 200
3500=         KK = KK - KSPAN + INC
3510=         IF (KK.LE.JC) GO TO 130
3520=         IF (KSPAN.EQ.JC) GO TO 350
3530=         GO TO 40
3540= 180     AKP = AKM + BJM
3550=         AKM = AKM - BJM
3560=         BKP = BKM - AJM
3570=         BKM = BKM + AJM
3580=         IF (S1.NE.0.0) GO TO 160
3590= 190     A(K1) = AKP
[illegible]




[illegible]
[illegible]
[illegible]         GO TO 120
[illegible]  200   [illegible]
[illegible]
[illegible]
[illegible]
[illegible]         GO TO 140
3730=C
3740=C  TRANSFORM FOR FACTOR OF 5 (OPTIONAL CODE)
3750=C
3760= 210   C2 = C72**2 - S72**2
```

```
[illegible]
      BK = BKP*C72 + BJP*C2 + BB
      AJ = AKM*S72 + AJM*S2
      BJ = BKM*S72 + BJM*S2
      A(K1) = AK - BJ
      A(K4) = AK + BJ
      B(K1) = BK + AJ
      B(K4) = BK - AJ
      AK = AKP*C2 + AJP*C72 + AA
      BK = BKP*C2 + BJP*C72 + BB
      AJ = AKM*S2 - AJM*S72
      BJ = BKM*S2 - BJM*S72
      A(K2) = AK - BJ
      A(K3) = AK + BJ
      B(K2) = BK + AJ
      B(K3) = BK - AJ
      KK = K4 + KSPAN
      IF (KK.LT.NN) GO TO 220
      KK = KK - NN
      IF (KK.LE.KSPAN) GO TO 220
[illegible]
      KSPNN = KSPAN
      KSPAN = KSPAN/K
      IF (K.EQ.3) GO TO 100
      IF (K.EQ.5) GO TO 510
      IF (K.EQ.JF) GO TO 250
      JF = K
      S1 = [illegible]
      C1 = COS(S1)
      S1 = SIN(S1)
      CK(JF) = 1.0
      SK(JF) = 0.0
      J = 1
  240 CK(J) = CK(K)*C1 + SK(K)*S1
```

```
[illegible]
4450=          K1 = [illegible]
4460=          K2 = [illegible]
4470=          J = J + 1
4480=          [illegible]
4490=          [illegible]
4500=          BT(J) = B(K1) + B(K2)
4510=          BK = BT(J) + BK
4520=          J = J + 1
4530=          AT(J) = A(K1) - A(K2)
4540=          BT(J) = B(K1) - B(K2)
4550=          K1 = K1 + KSPAN
4560=          IF (K1.LT.K2) GO TO 260
4570=          A(KK) = AK
4580=          B(KK) = BK
4590=          K1 = KK
4600=          K2 = KK + KSPNN
4610=          J = 1
4620=   270    K1 = K1 + KSPAN
4630=          K2 = K2 - KSPAN
4640=          [illegible] = 1
4650=          AK = AA
4660=          BK = BB
4670=          AJ = 0.0
4680=          BJ = 0.0
4690=          K = 1

[illegible]
4750=          BJ = BT(K)*[illegible] + B[illegible]
4760=          [illegible]
4770=          IF ([illegible].GT.[illegible]) [illegible]
4780=          IF (K.[illegible]) GO TO 2[illegible]0
4790=          K = JK - 1
[illegible]
[illegible]
4820=          A(K2) = AK + BJ
4830=          B(K2) = BK - AJ
4840=          J = J + 1
4850=          IF (J.LT.K) GO TO 270
4860=          KK = KK + KSPNN
```

```
[illegible]
5040=C  BE DELETED.
5050=C
5060=C        C1 = 0.5/(C2**2+S1**2) + 0.5
5070=C        S1 = C1*S1
5080=C        C2 = C1*C2
5090= 320     C1 = C2
5100=         S2 = S1
5110=         KK = KK + KSPAN
5120= 330     AK = A(KK)
5130=         A(KK) = C2*AK - S2*B(KK)
5140=         B(KK) = S2*AK + C2*B(KK)
5150=         KK = KK + KSPNN
5160=         IF (KK.LE.NT) GO TO 330
5170=         AK = S1*S2
5180=         S2 = S1*C2 + C1*S2
5190=         C2 = C1*C2 - AK
5200=         KK = KK - NT + KSPAN
5210=         IF (KK.LE.KSPNN) GO TO 330
5220=         KK = KK - KSPNN + JC
5230=         IF (KK.LE.MM) GO TO 310
[illegible]
5280= 340     [illegible]
5290=         C2 = COS(S1)
5300=         S1 = SIN(S1)
5310=         MM = MIN0(KSPAN,MM+KLIM)
5320=         GO TO 320
5330=C
5340=C  PERMUTE THE RESULTS TO NORMAL ORDER---DONE IN TWO STAGES
5350=C  PERMUTATION FOR SQUARE FACTORS OF N
5360=C
5370= 350     NP(1) = KS
5380=         IF (KT.EQ.0) GO TO 440
5390=         K = KT + KT + 1
5400=         IF (M.LT.K) K = K - 1
5410=         J = 1
```

```
[illegible]
C
C [illegible]
C
 370  AK = A(KK)
      A(KK) = A(K2)
      A(K2) = AK
      BK = B(KK)
      B(KK) = B(K2)
      B(K2) = BK
      KK = KK + INC
      K2 = KSPAN + K2
      IF (K2.LT.KS) GO TO 370
 380  K2 = K2 - NP(J)
      J = J + 1
      K2 = NP(J+1) + K2
      IF (K2.GT.NP(J)) GO TO 380
      J = 1
 390  IF (KK.LT.K2) GO TO 370
      KK = KK + INC
      K2 = KSPAN + K2
      IF (K2.LT.KS) GO TO 390
      IF (KK.LT.KS) GO TO 380
      JC = K3
      GO TO 440
C
C PERMUTATION FOR MULTIVARIATE TRANSFORM
C
[illegible]
[illegible]
[illegible]
      A(K2) = AK
      BK = B(KK)
      B(KK) = B(K2)
      B(K2) = BK
      KK = KK + INC
      K2 = K2 + INC
      IF (KK.LT.K) GO TO 410
      KK = KK + KS - JC
      K2 = K2 + KS - JC
      IF (KK.LT.NT) GO TO 400
      K2 = K2 - NT + KSPAN
      KK = KK - NT + JC
      IF (K2.LT.KS) GO TO 400
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
 450  NFAC(JJ) = NFAC(JJ)*NFAC(J+1)
      J = J - 1
      IF (J.NE.KT) GO TO 450
      KT = KT + 1
      NN = NFAC(KT) - 1
      JJ = 0
      J = 0
      GO TO 480
 460  JJ = JJ - K2
      K2 = KK
      K = K + 1
      KK = NFAC(K)
 470  JJ = KK + JJ
      IF (JJ.GE.K2) GO TO 460
      NP(J) = JJ
 480  K2 = NFAC(KT)
      K = KT + 1
      KK = NFAC(K)
      J = J + 1
      IF (J.LE.NN) GO TO 470

      GO TO 500
 490  [illegible]
      KK = NP(K)
      [illegible]
      IF (KK.NE.J) GO TO 490
      [illegible]
      [illegible]
      [illegible]
      IF (KK.LT.0) GO TO 500
      IF (KK.NE.J) GO TO 490
      NP(J) = -J
      IF (J.NE.NN) GO TO 500
      MAXF = INC*MAXF
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
6690=        K1 = K1 - INC
6700=        IF (K1.NE.KK) GO TO 530
6710= 540    K1 = KK + KSPAN
6720=        K2 = K1 - JC*(K+NP(K))
6730=        K = -NP(K)
6740= 550    A(K1) = A(K2)
6750=        B(K1) = B(K2)
6760=        K1 = K1 - INC
6770=        K2 = K2 - INC
6780=        IF (K1.NE.KK) GO TO 550
6790=        KK = K2
6800=        IF (K.NE.J) GO TO 540
6810=        K1 = KK + KSPAN
6820=        K2 = 0
6830= 560    K2 = K2 + 1
6840=        A(K1) = AT(K2)
6850=        B(K1) = BT(K2)
6860=        K1 = K1 - INC
6870=        IF (K1.NE.KK) GO TO 560
[illegible]
[illegible]
6930=        IF (JJ.NE.0) GO TO 510
6940=        RETURN
6950=        END
6960=*EOR
6970=*EOF
```

Appendix G. IMSL Mixed Radix FFT

The International Mathematical Subroutine Library contains a mixed radix subroutine which can perform the FFT of any positive integer length sequence. This subroutine was based on Singleton's article "On Computing the Fast Fourier Transform", Comm. ACM 10(10) 1967 in which he proposed several ideas used in the IMSL subroutine. As stated in Chapter III the program closely resembles Singleton's algorithm published in the open literature but the IMSL version has been copyrighted and the FORTRAN code is not listed in this paper. The IMSL description of the algorithm and its usage are included in this appendix for the convenience of the reader and a detailed development of the real operations count which was not presented in the main text is also in this appendix.

Real Operations Count For IMSL Mixed Radix Algorithm

A copyrighted mixed radix FFT is available through the International Mathematical Scientific Library (IMSL) on the CDC computer used at AFIT. This subroutine (FFTCC) can accept any length sequence N including prime numbers. It is based on an article written by Singleton, "On Computing the Fast Fourier Transform" published in 1967.

Functionally this subroutine has few differences from Singleton's algorithm described in the preceding section. The factoring, twiddle factors, and reordering of the data is the same, however, the special sections for factors of 3 and 4 require 2 and 8 more additions, respectively, than Singleton's subroutine. Also this mixed radix algorithm uses the general factors section for odd prime factors of 5 or greater which further reduces the efficiency compared to Singleton's.

As in the case of Singleton's FFT subroutine the real operations count for the IMSL subroutine is determined from the number of twiddle factors:

$$\sum_{i=1}^{m} (N(p_i-1)/p_i) - (N-1) \qquad (G.1)$$

and the number of butterflies:

$$\sum_{i=1}^{m} N/p_i \qquad (G.2)$$

where $N = p_1\ p_2\ \ldots\ p_m$. In this subroutine the factoring is performed such that $N = 2^r\ 3^s\ 4^t\ p_1^{m1}\ \ldots\ p_k^{mk}$ with the real operations count being derived from the FORTRAN coded subroutine FFTCC and the Eqs (G.1) and (G.2). The radix-2 section of FFTCC includes the twiddle factor multiplications with the butterfly computation. In this case there are rN/2 butterflies and twiddle factors to be computed using 4 real multiplications and 6 real additions giving:

$$\text{\# real mult} = 4(rN/2) = 2rN \qquad (G.3)$$

$$\text{\# real adds} = 6(rN/2) = 3rN \qquad (G.4)$$

The radix-3 section uses sN/3 butterflies and 2sN/3 twiddle factors which require 4 real multiplications and 14 additions per butterfly and 4 real multiplications and 2 real additions per twiddle factor. Combining the butterflies and twiddle factors the real operations count for the radix-3 section is given by:

$$\text{real mult} = 4(2sN/3) + 4(sN/3) = 4sN \qquad (G.5)$$

$$\text{real adds} = 14(sN/3) + 2(2sN/3) = 6sN \qquad (G.6)$$

The radix-4 section uses 24 real additions and no real multiplications for the tN/4 butterflies. The 3tN/4 twiddle factors require 2 real additions and 4 real multiplications. Combining the results gives:

$$\text{real mult} = 3tN \qquad (G.7)$$

$$\begin{aligned}\text{real adds} &= 24tN/4 + 2(3tN/4)\\ &= 15tN/2 \qquad (G.8)\end{aligned}$$

All odd prime factors equal to or greater than 5 use the general transform section. Based on the FORTRAN program written by IMSL there are five sources of real operations in this general radix-p_i transform excluding the array indexing additions. First the complex multipliers are computed for the butterfly transmittance:

$$\text{real mult} = 2(p_i-1) \tag{G.9}$$

$$\text{real adds} = (p_i-1) \tag{G.10}$$

for each new factor p_i, e.g., N=7*4=28 and N=7*7*4=196 each require the same $(p_i-1)=(7-1)$ complex multiplications for the factor $p_i=7$. Second the complex twiddle factor multiplications are performed on the data array. Assuming N can be factored as:

$$N = 2^r\ 3^s\ 4^t\ p_1^{m1}\ p_2^{m2} \dots p_k^{mk}$$

where p_i^{mi} represents the ith factor raised to some positive integer mi, the number of complex twiddles is $(mi)N(p_i-1)/p_i$ $-(N-1)$. The n-1 term is subtracted only once for each FFT, which means the intermediate result can be written as:

$$\text{real mult} = 4(mi)N(p_i-1)/p_i \tag{G.11}$$

$$\text{real adds} = 2(mi)N(p_i-1)/p_i \tag{G.12}$$

The individual butterflies are computed next. The first output of each butterfly requires only $3(p_i-1)/2$ real additions and no multiplications. For each radix-p_i^{mi} there are $(mi)N/p_i$ butterflies in the FFT giving:

$$\text{real adds} = (8(p_i-1)/2)(N(mi)/p_i)$$

$$= 4N(mi)(p_i-1)/p_i \qquad (G.13)$$

Now the remaining portion of each butterfly is computed using $(p_i-1)^2$ real multiplications and additions. This gives a total of:

$$\text{real mult} = N(p_i-1)^2(mi)/p_i \qquad (G.14)$$

$$\text{real adds} = N(p_i-1)^2(mi)/p_i \qquad (G.15)$$

Finally the results of the butterfly operations are stored in the proper array locations requiring 4 real additions times $(p_i-1)/2$ times the number of radix-p_i butterflies. This total is:

$$\text{real adds} = (4(p_i-1)/2)(N(mi)/p_i)$$

$$= 2(mi)N(p_i-1)/p_i \qquad (G.16)$$

Combining Eqs (G.9) - (G.16) the number of real operations for the p_i factor becomes:

$$\text{real mult} = \sum_{i=1}^{k} (2(p_i-1) + 4(mi)N(p_i-1)/p_i$$

$$+ N(p_i-1)^2(mi)/p_i) \qquad (G.17)$$

$$\text{real adds} = \sum_{i=1}^{k} ((p_i-1) + 2(mi)N(p_i-1)/p_i$$

$$+ 4(mi)N(p_i-1)/p_i + N(p_i-1)^2(mi)/p_i$$

$$+ 2(mi)N(p_i-1)/p_i)$$

$$= \sum_{i=1}^{k} ((p_i-1) + 8(mi)N(p_i-1)/p_i$$

$$+ N(p_i-1)^2(mi)/p_i) \qquad (G.18)$$

Using Eqs (G.17) and (G.18) for the odd prime factors and the real operations count for factors of 2, 3, and 4 the total operations cound for $N = 2^r\ 3^s\ 4^t\ p_1^{m1} \ldots p_k^{mk}$ can be written as:

$$\text{real mult} = 2rN + 4sN + 3tN + \sum_{i=1}^{k} (2(p_i-1) + 4(mi)N(p_i-1)/p_i + N(mi)\ (p_i-1)^2/p_i) - 4(N-1) \qquad (G.19)$$

$$\text{real adds} = 3rN + 6sN + 15tN/2 + \sum_{i=1}^{k} ((p_i-1) + 8(mi)N(p_i-1)/p_i + N(MI)(p_i-1)^2/p_i) - 2(N-1) \qquad (G.20)$$

As in any FFT the real operations associated with the twiddle factors have been reduced by (N-1) multiplications and additions because the last stage of decimation-in-frequency or the first stage of a decimation-in-time FFT require no twiddles.

Appendix H. An Algorithm for Computing the WFTA

This program computes the DFT defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-j2\pi nk/N) \ ; \ k=0,\ 1,\ \ldots,\ N-1$$

where the sequence length N is a product of the relative prime factors from the set (2,3,4,5,7,8,9,16).

Program Description. The WFTA consists of the six subroutines PERM 1, PERM 2, MULT, WEAVE 1, WEAVE 2, and INISHL. Step One is to map the sequence x(n) into a u-dimensional array $s(n_1, n_2, \ldots, n_u)$. Step Two implements the "pre-weave" modules in subroutine WEAVE 1, one for each factor of N_j. Each of the pre-weave modules contains only additions. Step Three performs a point by point multiply on the data array (subroutine MULT) of real constants derived from the small-N DFT algorithms. These constant multipliers are a function of the complex exponentials of W_N and are the only complex multiplications required in the algorithm. Step Four implements the post-weave (WEAVE 2 subroutine) module which contains additions, subtractions, and multiplies by j. Step Five maps the u-dimensional array $s(k_1, k_2, \ldots, k_u)$ into the correct one-dimensional DFT x(k) according to the Chinese remainder theorem given in Eq (3.144) (McClellan and Nawab, 1979).

Arguments. The WFTA is called using the following arguments. More arguments exist in this list than in the one given by McClellan and Nawab because array storage is minimized in this WFTA version.

N = Transform length which must be factorable into mutually prime factors from the set 2,3,4,5,7,8,9,16. A list of acceptable sequence lengths is given in the left-most column of Table 3.9a,b.

XR and XI = The real and imaginary arrays to be transformed and are dimensioned to length N in the calling program.

INIT = A flag to specify whether the call to FFTWIN requires initialization. INIT = 0 means initialization is required and INIT $\neq$ 0 skips the phase. Initialization is needed when calling FFTWIN for the first time for a given sequence length.

IERR = Contains an error code upon return from FFTWIN. If the DFT was successful IERR = 0; if an error occurred IERR = -1 or -2. There are two causes for an error:

(1) The transform length is illegal, or

(2) The program has not been initialized for the correct length N sequence.

SR and SI = One dimensional working arrays of length $M = M_1 \times M_2 \times M_3 \times M_4$ which is the product of the multiplies required by the small-N algorithms. The value of M for any permissible N is given in Table H.1 in the right-most column.

COEF = One-dimensional array length M used to store the constant coefficients generated by INISHL for the "weave" modules.

INDX1 and INDX2 = One-dimensional length N mapping vectors for pre- and post-permutations of the data.

<u>Usage</u>

(1) Specify the input sequences XR and XI with parameters N, INIT, IERR, SR, WI, COEF, INDX 1, INDX 2.

(2) Call WFTA (XR, XI, N, INIT, ERR, SR, SI, COEF, INDX 1, INDX 2).

(3) XR and XI are the output real and imaginary vectors. The error code IERR=0 specifies successful completion of the transform.

(4) After the initial call, use INIT≠0 as long as N remains constant.

```
[illegible]
      [illegible]
      [illegible]
      [illegible]
      IF(INIT.EQ.N) [illegible]

      IF(IERR.EQ.0) RETURN
      M=NA*NB*NC*ND
      IF(M.EQ.N) GO TO 100
      IERR=-2
      RETURN
C
C   PROGRAM NOT INITIALIZED FOR THIS VALUE OF N
C
100   NMULT=ND1*ND2*ND3*ND4
C
C     PERMUTE THE INPUT DATA
C
      CALL PERM1(SR,SI,XR,XI,INDX1)
C
C      DO THE PRE-WEAVE MODULES
C
      CALL WEAVE1(SR,SI)
C
C      DO THE NESTED MULTIPLICATIONS
C
      CALL MULT(SR,SI,COEF,NMULT)
C
      [illegible]
      [illegible]
      [illegible]

       PERMUTE THE OUTPUT DATA
C
      CALL PERM2(SR,SI,XR,XI,INDX2)
      RETURN
      END
      SUBROUTINE INITAL(N,COEF,XR,XI,INDX1,INDX2,IERR)
      REAL      COEF(1),XR(1),XI(1)
      [illegible]
      REAL     CO3(3),CO4(4),CO8(8),CO9(11),CO16(18),CO1(18),[illegible]
     1CO3(9),CO4(6)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
```

```
[illegible]
[illegible]     DATA [illegible] 1,[illegible]
[illegible]
[illegible]
[illegible]     1 [illegible]
[illegible]     [illegible] -0.[illegible] -1.0,-1.[illegible]
[illegible]
[illegible]     DATA [illegible] 1.
[illegible]     DATA [illegible] 1.
[illegible]     DATA [illegible] 1.0,-1.[illegible]
[illegible]     1 0.[illegible] -1.
[illegible]     2 0.[illegible]
810=       DATA CD4/1.0,-1.25,-1.538841769,0.5590169944,0.363271264,
820=       1 0.5877852523/
830=C
840=C        FOLLOWING SEGMENT DETERMINES FACTORS OF N AND CHOOSES
850=C        THE APPROPRIATE SHORT DFT COEFFICIENTS.
860=C
870=        IERR=0
880=        ND1=1
890=        NA=1
900=        NB=1
910=        ND2=1
920=        NC=1
930=        ND3=1
940=        ND=1
950=        ND4=1
960=        IF(N.LE.0) GO TO 190
970=        IF(16*(N/16).EQ.N) GO TO 30
980=        IF(8*(N/8).EQ.N) GO TO 40
990=        IF(4*(N/4).EQ.N) GO TO 50
[illegible]
[illegible]
[illegible]

1040=       [illegible]
1050=30     ND1=16
1060=       NA=16
1070=       DO 31 J=1,16
1080=31     CD1(J)=CD16(J)
1090=       GO TO 70
1100=40     ND1=8
1110=       [illegible]
1120=       DO 41 J=1,8
1130=41     CD1(J)=CD8(J)
1140=       GO TO 70
1150=50     ND1=4
1160=       NA=4
```

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
1370=        IF(M.EQ.N) GO TO 250
1380=210     PRINT*,"THIS N DOES NOT WORK"
1390=        IERR=-1
1400=        RETURN
1410=C
1420=C        NEXT SEGMENT GENERATES THE DFT COEFFICIENTS AND
1430=C        THE FLAG ARRAY.
1440=250     J=1
1450=        DO 300 N4=1,ND4
1460=        DO 300 N3=1,ND3
1470=        DO 300 N2=1,ND2
1480=        DO 300 N1=1,ND1
1490=        COEF(J)=CD1(N1)*CD2(N2)*CD3(N3)*CD4(N4)
1500=        J=J+1
1510=300     CONTINUE
1520=C        FOLLOWING SEGMENT FOR INPUT INDEXING.
1530=        J1=0
1540=        J2=0
1550=        J3=0
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
1610=        NN=NA*NB*ND
1620=        N[illegible]=NA*NB*NC
1630=        K=1
1640=        DO 440 N4=1,ND
1650=        DO 430 N3=1,NC
1660=        DO 420 N2=1,NB
1670=        DO 410 N1=1,NA
1680=405     IF(K.LE.N) GO TO 406
1690=        K=K-N
1700=        GO TO 405
1710=406     INDX1(J)=K
```

```
[illegible]=         M=1

[illegible]=         [illegible]
[illegible]=         [illegible]
[illegible]=         [illegible]
[illegible]          [illegible]
[illegible]=         M=1
[illegible]          [illegible]
[illegible]=         [illegible]
[illegible]=         [illegible]
1910=         GO TO 550
1920=560      S2=P1+1
1930=540      IF(NC.EQ.1) GO TO 630
1940=         M=1
1950=620      P1=M*NW-1
1960=         IF((P1/NC)*NC.EQ.P1) GO TO 610
1970=         M=M+1
1980=         GO TO 620
1990=610      S3=P1+1
2000=630      IF(ND.EQ.1) GO TO 660
2010=         M=1
2020=640      P1=M*NN-1
2030=         IF((P1/ND)*ND.EQ.P1) GO TO 650
2040=         M=M+1
2050=         GO TO 640
2060=650      S4=P1+1
2070=660      J=1
2080=         DO 810 N4=1,ND
2090=         DO 810 N3=1,N
2100=         [illegible]

2150=         GO TO [illegible]
2160=810      I=I+1
2170=710      CONTINUE
2180=         RETURN
2190=         END
[illegible]   [illegible]
[illegible]   [illegible]
2220=         REAL [illegible]
2230=         INTEGER IND(1,1)
2240=         J=1
2250=         K=1
2260=         IND1=ND1-NH
```

```
[illegible]         [illegible]+1



[illegible]         [illegible]
[illegible]         [illegible]
[illegible]         [illegible]
[illegible]         [illegible]
[illegible]         COMMON [illegible]
[illegible]         REAL [illegible]
[illegible]         INTEGER [illegible]
[illegible]         J=1
2470=        K=1
2480=        INC1=ND1-NA
2490=        INC2=ND1*(ND2-NB)
2500=        INC3=ND1*ND2*(ND3-NC)
2510=        DO 40 N4=1,ND
2520=        DO 30 N3=1,NC
2530=        DO 20 N2=1,NB
2540=        DO 10 N1=1,NA
2550=        XR(INDX2(K))=SR(J)
2560=        XI(INDX2(K))=SI(J)
2570=        K=K+1
2580=10      J=J+1
2590=20      J=J+INC1
2600=30      J=J+INC2
2610=40      J=J+INC3
2620=        RETURN
2630=        END
2640=        SUBROUTINE WEAVE1(SR,SI)
[illegible]         COMMON NA,NB,NC,ND,ND1,ND2,ND3,[illegible]
[illegible]         REAL [illegible]


      *******************************[illegible]
[illegible]
[illegible]         THE FOLLOWING CODE IMPLEMENT [illegible] POINT [illegible]
[illegible]
[illegible]  ******************************************************
[illegible]
[illegible]         [illegible]
[illegible]         [illegible]
[illegible]         [illegible]=1
2790=        DO 240 N4=1,ND
2800=        DO 230 N3=1,NC
2810=        DO 220 N2=1,NB
```

```
[illegible]=        NR1=NBASE+1
[illegible]=        T0=R(NBASE)+R(NR1)
[illegible]=        R(NR1)=R(NBASE)-R(NR1)
[illegible]         R(NBASE)=T0
[illegible]=        T0=I(NBASE)+I(NR1)
[illegible]=        I(NR1)=I(NBASE)-I(NR1)
[illegible]=        I(NBASE)=T0
[illegible]=820     NBASE=NBASE+2
[illegible]=830     NBASE=NBASE+NLUP22
[illegible]=840     NBASE=NBASE+NLUP23
[illegible]=[illegible]    IF(NN.[illegible]) GO TO [illegible]
[illegible]=C
[illegible]=C  **************************************************
2950=C
2960=C        THE FOLLOWING CODE IMPLEMENTS THE 8 POINT [illegible]
2970=C
2980=C  **************************************************
2990=C
3000=         NLUP22=8*(NB2-NB)
3010=         NLUP23=8*NB2*(NB3-NC)
3020=         NBASE=1
3030=         DO 840 N4=1,ND
3040=         DO 830 N3=1,NC
3050=         DO 820 N2=1,NB
3060=         NR1=NBASE+1
3070=         NR2=NR1+1
3080=         NR3=NR2+1
3090=         NR4=NR3+1
3100=         NR5=NR4+1
3110=         NR6=NR5+1
3120=         NR7=NR6+1
3130=         T3=R(NR3)+R(NR7)
3140=         T7=R(NR3)-R(NR7)
3150=         T0=R(NBASE)+R(NR4)
3160=         R(NR4)=R(NBASE)-R(NR4)
3170=         T1=R(NR1)+R(NR5)
3180=         T5=R(NR1)-R(NR5)
3190=         T2=R(NR2)+R(NR6)
3200=         R(NR6)=R(NR2)-R(NR6)
[illegible]=  R([illegible])=[illegible]+T[illegible]
[illegible]=  R(NR2)=T0-T2
[illegible]=  R(NR1)=T1+T[illegible]
[illegible]   [illegible]
[illegible]=  R(NR5)=T5+T[illegible]
3260=         R(NR7)=T5-T7
[illegible]=  T3=I(NR3)+I(NR7)
3280=         T7=I(NR3)-I(NR7)
3290=         T0=I(NBASE)+I(NR4)
[illegible]=  I(NR4)=I(NBASE)-I(NR4)
[illegible]=  T1=I(NR1)+I(NR5)
[illegible]=  T5=I(NR1)-I(NR5)
[illegible]=  T2=I(NR2)+I(NR6)
3340=         I(NR6)=I(NR2)-I(NR6)
[illegible]=  I(NBASE)=T0+T2
3360=         I(NR2)=T0-T2
```

```
         I(NR1)=I1+I3
         I(N[illegible])=I1-I3
         I([illegible])=I[illegible]+I[illegible]
         I([illegible])=I[illegible]-I[illegible]
[illegible] NBASE=NBASE+8
         N[illegible]=N[illegible]
[illegible] NBASE=NBASE+N[illegible]-8
[illegible] IF(NA,NE,16) GO TO 7[illegible]
C
C ******************************************************
C
C      THE FOLLOWING CODE IMPLEMENTS [illegible]
C
C ******************************************************
C
      NLOOP2=16*(NB2-NB)
      NLUP23=16*NB2*(NB3-NB2)
      NBASE=1
      DO 1640 N4=1,[illegible]
      DO 1630 N3=1,NC
      DO 1620 N2=1,NB
      NR1=NBASE+1
      NR2=NR1+1
      NR3=NR2+1
      NR4=NR3+1
      NR5=NR4+1
      NR6=NR5+1
      NR7=NR6+1
      NR8=NR7+1
      NR9=NR8+1
      NR10=NR9+1
      NR11=NR10+1
      NR12=NR11+1
      NR13=NR12+1
      NR14=NR13+1
      NR15=NR14+1
      NR16=NR15+1
      NR17=NR16+1
      NBASE=NB[illegible]
      DO 1645 J=1,[illegible]
      T(J)=TR(NBASE+[illegible])+TR(NBASE+[illegible])
      T([illegible])=[illegible]
      [illegible]=[illegible]+1
1645  CONTINUE
      DO 1650 J=1,4
      U(J)=T(J)+T(J+4)
      U(J+4)=T(J)-T(J+4)
1650  CONTINUE
      TR(NBASE+[illegible])=U(1)+U(3)
      TR([illegible])=U(1)-U(3)
      [illegible]=[illegible]+U(4)
      TR(NR3)=U(2)-U(4)
      TR(NR5)=U(6)+U(8)
      TR(NR7)=U(6)-U(8)
      TR(NR4)=U(5)
```

```
3920=          R(NR6)=Q(7)
3930=          R(NR8)=T(9)
3940=          R(NR9)=T(10)+T(16)
3950=          R(NR15)=T(10)-T(16)
3960=          R(NR13)=T(14)+T(12)
3970=          R(NR11)=T(14)-T(12)
3980=          R(NR17)=R(NR11)+R(NR15)
3990=          R(NR16)=R(NR9)+R(NR13)
4000=          R(NR10)=T(11)+T(15)
4010=          R(NR14)=T(11)-T(15)
4020=          R(NR12)=T(13)
4030=          JRATE=NRATE
4040=          DO 1745 J=1,8
4050=          T(J)=I(JRATE)+I(JRATE+[illegible])
4060=          T(J+8)=I(JRATE)-I(JRATE+[illegible])
4070=          JRATE=JRATE+1
4080=1745      CONTINUE
4090=          DO 1750 J=1,4
4100=          Q(J)=T(J)+T(J+4)
4110=          Q(J+4)=T(J)-T(J+4)
4120=1750      CONTINUE
4130=          SI(NRATE)=Q(1)+Q(3)
4140=          SI(NR2)=Q(1)-Q(3)
4150=          SI(NR1)=Q(2)+Q(4)
4160=          SI(NR3)=Q(2)-Q(4)
4170=          SI(NR5)=Q(6)+Q(8)
4180=          SI(NR7)=Q(6)-Q(8)
4190=          SI(NR4)=Q(5)
4200=          SI(NR6)=Q(7)
4210=          SI(NR8)=T(9)
4220=          SI(NR9)=T(10)+T(16)
4230=          SI(NR15)=T(10)-T(16)
4240=          SI(NR13)=T(14)+T(12)
4250=          SI(NR11)=T(14)-T(12)
4260=          SI(NR17)=SI(NR11)+SI(NR15)
4270=          SI(NR16)=SI(NR9)+SI(NR13)
4280=          SI(NR10)=T(11)+T(15)
4290=          SI(NR14)=T(11)-T(15)
4[illegible]=          SI(NR12)=T(13)
4[illegible]          [illegible]=[illegible]+1[illegible]
4[illegible]=[illegible]   [illegible]=[illegible]+[illegible]
4[illegible]=1[illegible]   [illegible]=[illegible]+[illegible]
4[illegible]          [illegible]
4[illegible]=          IF(NB.NE.[illegible]) GO TO [illegible]
4[illegible]=C
4[illegible]=C  **************************************************
4[illegible]=C
4[illegible]=C       THE FOLLOWING CODE IMPLEMENT  THE 3 POINT FFE-[illegible]
4[illegible]=C
4[illegible]=C  **************************************************
44[illegible]
44[illegible]=          NCUF2=2*ND1
4440=          NCU[illegible]=2*ND1*(ND[illegible]-N[illegible])
4450=          NRA E=1
4460=          NOFF=ND1
4[illegible]
```

AD-A100 782 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 12/1
EFFICIENT COMPUTER IMPLEMENTATIONS OF FAST FOURIER TRANSFORMS.(U)
DEC 80 J D BLANKEN
UNCLASSIFIED AFIT/GE/EE/80D-9 NL

4 OF 4
AD
A100782

END
DATE
FILMED
7 81
DTIC

```
      DO 340 N4=1,ND
      DO 330 N3=1,NC
      DO 310 N2=1,ND1
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      T1=SR(NR1)+SR(NR2)
      SR(NBASE)=SR(NBASE)+T1
      SR(NR2)=SR(NR1)-SR(NR2)
      SR(NR1)=T1
      T1=SI(NR1)+SI(NR2)
      SI(NBASE)=SI(NBASE)+T1
      SI(NR2)=SI(NR1)-SI(NR2)
      SI(NR1)=T1
310   NBASE=NBASE+1
330   NBASE=NBASE+NLUP2
340   NBASE=NBASE+NLUP23
400   IF(NB.NE.9) GO TO 700
C
C ****************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 9 POINT PRE-WEAVE MODULE
C
C ****************************************************
C
      NLUP2=10*ND1
      NLUP23=11*ND1*(ND3-NC)
      NBASE=1
      NOFF=ND1
      DO 940 N4=1,ND
      DO 930 N3=1,NC
      DO 910 N2=1,ND1
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      NR9=NR8+NOFF
      NR10=NR9+NOFF
      T3=SR(NR3)+SR(NR6)
      T6=SR(NR3)-SR(NR6)
      SR(NBASE)=SR(NBASE)+T3
      T7=SR(NR7)+SR(NR2)
      T2=SR(NR7)-SR(NR2)
      SR(NR2)=T6
      T1=SR(NR1)+SR(NR8)
      T6=SR(NR1)-SR(NR8)
      SR(NR1)=T6
      T4=SR(NR4)+SR(NR5)
      T5=SR(NR4)-SR(NR5)
      SR(NR3)=T1+T4+T7
      SR(NR4)=T1-T7
      SR(NR5)=T4-T1
      SR(NR6)=T7-T4
      SR(NR10)=T2+T5+T8
      SR(NR7)=T8-T2
      SR(NR8)=T5-T8
      SR(NR9)=T2-T5
      T3=SI(NR3)+SI(NR6)
      T6=SI(NR3)-SI(NR6)
      SI(NBASE)=SI(NBASE)+T3
      T7=SI(NR7)+SI(NR8)
      T2=SI(NR7)-SI(NR8)
      SI(NR2)=T6
      T1=SI(NR1)+SI(NR9)
      T8=SI(NR1)-SI(NR9)
      SI(NR1)=T3
      T4=SI(NR4)+SI(NR5)
      T5=SI(NR4)-SI(NR5)
      SI(NR3)=T1+T4+T7
      SI(NR4)=T1-T7
      SI(NR5)=T4-T1
      SI(NR6)=T7-T4
      SI(NR10)=T2+T5+T8
      SI(NR7)=T8-T2
      SI(NR8)=T5-T8
      SI(NR9)=T2-T5
910   NBASE=NBASE+1
930   NBASE=NBASE+NLUP2
940   NBASE=NBASE+NLUP23
700   IF(NC.NE.7) GO TO 500
C
C ****************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 7 POINT PRE-WEAVE MODULE
C
C ****************************************************
C
      NOFF=ND1*ND2
      NBASE=1
      NLUP2=8*NOFF
      DO 740 N4=1,ND
      DO 710 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      T1=SR(NR1)+SR(NR6)
      T6=SR(NR1)-SR(NR6)
      T4=SR(NR4)+SR(NR3)
      T3=SR(NR4)-SR(NR3)
      T2=SR(NR2)+SR(NR5)
      T5=SR(NR2)-SR(NR5)
      SR(NR5)=T6-T3
      SR(NR2)=T5+T3+T6
      SR(NR6)=T5-T6
      SR(NR8)=T3-T5
      SR(NR3)=T2-T1
      SR(NR4)=T1-T4
      SR(NR7)=T4-T2
      T1=T1+T4+T2
      SR(NBASE)=SR(NBASE)+T1
      SR(NR1)=T1
      T1=SI(NR1)+SI(NR6)
      T6=SI(NR1)-SI(NR6)
      T4=SI(NR4)+SI(NR3)
      T3=SI(NR4)-SI(NR3)
      T2=SI(NR2)+SI(NR5)
      T5=SI(NR2)-SI(NR5)
      SI(NR5)=T6-T3
      SI(NR2)=T5+T3+T6
      SI(NR6)=T5-T6
      SI(NR8)=T3-T5
      SI(NR3)=T2-T1
      SI(NR4)=T1-T4
      SI(NR7)=T4-T2
      T1=T1+T4+T2
      SI(NBASE)=SI(NBASE)+T1
      SI(NR1)=T1
710   NBASE=NBASE+1
740   NBASE=NBASE+NLUP2
500   IF(ND.NE.5) RETURN
C
C ******************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 5 POINT PRE-WEAVE MODULE
C
C ******************************************************
C
      NOFF=ND1*ND2*ND3
      NBASE=1
      DO 510 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      T4=SR(NR1)-SR(NR4)
      T1=SR(NR1)+SR(NR4)
      T3=SR(NR3)+SR(NR2)
      T2=SR(NR3)-SR(NR2)
      SR(NR3)=T1-T3
      SR(NR1)=T1+T3
      SR(NBASE)=SR(NBASE)+SR(NR1)
      SR(NR5)=T2+T4
      SR(NR2)=T4
      SR(NR4)=T2
      T4=SI(NR1)-SI(NR4)
      T1=SI(NR1)+SI(NR4)
      T3=SI(NR3)+SI(NR2)
      T2=SI(NR3)-SI(NR2)
      SI(NR5)=T1-T3
      SI(NR1)=T1+T3
      SI(NBASE)=SI(NBASE)+SI(NR1)
      SI(NR3)=T2+T4
      SI(NR2)=T4
      SI(NR4)=T2
510   NBASE=NBASE+1
      RETURN
      END
      SUBROUTINE MULT (SR,SI,COEF,NMULT)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL   SR(1),SI(1),COEF(1)
      DO 10 J=1,NMULT
      SR(J)=SR(J)◆COEF(J)
      SI(J)=SI(J)◆COEF(J)
 10   CONTINUE
      RETURN
      END
      SUBROUTINE WEAVE2(SR,SI)
      REAL   SR(1),SI(1)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL   Q(8),T(16)
      IF(ND.NE.5) GO TO 700
C
C ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
C
C     THE FOLLOWING CODE IMPLEMENTS THE 5 POINT POST-WEAVE MODULE
C
C ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
C
      NOFF=ND1◆ND2◆ND3
      NBASE=1
      DO 510 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      T1=[illegible]
      T3=T1-SR(NR3)
      T1=T1+SR(NR3)
      T4=SI(NR2)+SI(NR5)
      T2=SI(NR4)+SI(NR5)
      SR(NR1)=T1-T4
      SR4=T1+T4
      SR2=T3+T2
      SR(NR3)=T3-T2
      T1=SI(NBASE)+SI(NR1)
      T3=T1-SI(NR3)
      T1=T1+SI(NR3)
      T4=SR(NR2)+SR(NR5)
      T2=SR(NR4)+SR(NR5)
      SI(NR1)=T1+T4
      SI(NR4)=T1-T4
      SI(NR2)=T3-T2
      SI(NR3)=T3+T2
      SR(NR2)=SR2
      SR(NR4)=SR4
510   NFAC=NFAC+1
700   IF(NC.NE.7) GO TO 300
C
C  ************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 7 POINT [illegible]
C
C  ************************************************************
C
      NOFF=N01*N02
      NBASE=1
      NLUP2=8*NOFF
      DO 740 N4=1,N0
      DO 710 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      T1=SR(NR1)+SR(NBASE)
      T2=T1-SR(NR3)-SR(NR4)
      T4=T1+SR(NR3)-SR(NR7)
      T1=T1+SR(NR4)+SR(NR7)
      T6=SI(NR2)+SI(NR5)+SI(NR8)
      T5=SI(NR2)-SI(NR5)-SI(NR6)
      T3=SI(NR2)+SI(NR6)-SI(NR8)
      SR(NR1)=T1-T6
      SR6=T1+T6
      SR2=T2-T5
      SR5=T2+T5
      SR(NR4)=T4-T3
      SR(NR3)=T4+T3
      T1=SI(NR1)+SI(NBASE)
      T2=T1-SI(NR3)-SI(NR4)
      T4=T1+SI(NR3)-SI(NR7)
      T1=T1+SI(NR4)+SI(NR7)
      T6=SR(NR2)+SR(NR5)+SR(NR8)
      T5=SR(NR2)-SR(NR5)-SR(NR6)
      T3=SR(NR2)+SR(NR6)-SR(NR8)
      SI(NR1)=T1+T6
      SI(NR6)=T1-T6
      SI(NR2)=T2+T5
      SI(NR5)=T2-T5
      SI(NR4)=T4+T3
      SI(NR3)=T4-T3
      SR(NR2)=SR2
      SR(NR5)=SR5
```

```
7220=       SR(NR6)=SR6
7230=210    NBASE=NBASE+1
7240=240    NBASE=NBASE+NLUP2
7250=300    IF(NB.EQ.1) GO TO 400
7260=       IF(NB.NE.3) GO TO 900
7270=C
7280=C *********************************************************
7290=C
7300=C      THE FOLLOWING CODE IMPLEMENTS THE 3 POINT POST-WEAV
7310=C
7320=C *********************************************************
7330=C
7340=       NLUP2=2*ND1
7350=       NLUP23=3*ND1*(ND3-NC)
7360=       NBASE=1
7370=       NOFF=ND1
7380=       DO 340 N5=1,NB
7390=       DO 330 N4=1,NC
7400=       DO 310 N2=1,ND1
7410=       NR1=NBASE+NOFF
7420=       NR2=NR1+NOFF
7430=       T1=SR(NBASE)+SR(NR1)
7440=       SR(NR1)=T1-SI(NR2)
7450=       SR2=T1+SI(NR2)
7460=       T1=SI(NBASE)+SI(NR1)
7470=       SI(NR1)=T1+SR(NR2)
7480=       SI(NR2)=T1-SR(NR2)
7490=       SR(NR2)=SR2
7500=310    NBASE=NBASE+1
7510=330    NBASE=NBASE+NLUP2
7520=340    NBASE=NBASE+NLUP23
7530=900    IF(NB.NE.9) GO TO 400
7540=C
7550=C *********************************************************
7560=C
7570=C      THE FOLLOWING CODE IMPLEMENTS THE 9 POINT POST-WEAVE MODULE
7580=C
7590=C *********************************************************
7600=C
7610=       NLUP2=10*ND1
7620=       NLUP23=11*ND1*(ND3-NC)
7630=       NBASE=1
7640=       NOFF=ND1
7650=       DO 940 N4=1,ND
7660=       DO 930 N3=1,NC
7670=       DO 910 N2=1,ND1
7680=       NR1=NBASE+NOFF
7690=       NR2=NR1+NOFF
7700=       NR3=NR2+NOFF
7710=       NR4=NR3+NOFF
7720=       NR5=NR4+NOFF
7730=       NR6=NR5+NOFF
7740=       NR7=NR6+NOFF
7750=       NR8=NR7+NOFF
7760=       NR9=NR8+NOFF
```

```
      NR10=NR9+NOFF
      T3=SR(NBASE)-SR(NR3)
      T7=SR(NBASE)+SR(NR1)
      SR(NBASE)=SR(NBASE)+SR(NR3)+SR(NR3)
      T6=T3+SI(NR10)
      SR(NR3)=T3-SI(NR10)
      T4=T7+SR(NR5)-SR(NR6)
      T1=T7-SR(NR4)-SR(NR5)
      T7=T7+SR(NR4)+SR(NR6)
      SR(NR6)=T6
      T8=SI(NR2)-SI(NR7)-SI(NR8)
      T5=SI(NR2)+SI(NR8)-SI(NR9)
      T2=SI(NR2)+SI(NR7)+SI(NR9)
      SR(NR1)=T7-T2
      SR8=T7+T2
      SR(NR4)=T1-T8
      SR(NR5)=T1+T8
      SR7=T4-T5
      SR2=T4+T5
      T3=SI(NBASE)-SI(NR3)
      T7=SI(NBASE)+SI(NR1)
      SI(NBASE)=SI(NBASE)+SI(NR3)+SI(NR3)
      T6=T3-SR(NR10)
      SI(NR3)=T3+SR(NR10)
      T4=T7+SI(NR5)-SI(NR6)
      T1=T7-SI(NR4)-SI(NR5)
      T7=T7+SI(NR4)+SI(NR6)
      SI(NR6)=T6
      T8=SR(NR2)-SR(NR7)-SR(NR8)
      T5=SR(NR2)+SR(NR8)-SR(NR9)
      T2=SR(NR2)+SR(NR7)+SR(NR9)
      SI(NR1)=T7+T2
      SI(NR8)=T7-T2
      SI(NR4)=T1+T8
      SI(NR5)=T1-T8
      SI(NR7)=T4+T5
      SI(NR2)=T4-T5
      SR(NR2)=SR2
      SR(NR7)=SR7
      SR(NR8)=SR8
410   NBASE=NBASE+1
420   NBASE=NBASE+NLUP2
430   NBASE=NBASE+NLUP23
400   IF(NA.EQ.1) RETURN
      IF(NA.NE.4) GO TO 800
C
C ****************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 4 POINT POST-WEAVE
C
C ****************************************************
C
      NLUP2=4*(ND2-NB)
      NLUP23=4*ND2*(ND3-NC)
      NBASE=1
```

```
8320=      DO 440 N4=1,ND
8330=      DO 430 N3=1,NC
8340=      DO 420 N2=1,NB
8350=      NR1=NBASE+1
8360=      NR2=NR1+1
8370=      NR3=NR2+1
8380=      TR0=SR(NBASE)+SR(NR2)
8390=      TR2=SR(NBASE)-SR(NR2)
8400=      TR1=SR(NR1)+SR(NR3)
8410=      TR3=SR(NR1)-SR(NR3)
8420=      TI1=SI(NR1)+SI(NR3)
8430=      TI3=SI(NR1)-SI(NR3)
8440=      SR(NBASE)=TR0+TR1
8450=      SR(NR2)=TR0-TR1
8460=      SR(NR1)=TR2+TI3
8470=      SR(NR3)=TR2-TI3
8480=      TI0=SI(NBASE)+SI(NR2)
8490=      TI2=SI(NBASE)-SI(NR2)
8500=      SI(NBASE)=TI0+TI1
8510=      SI(NR2)=TI0-TI1
8520=      SI(NR1)=TI2-TR3
8530=      SI(NR3)=TI2+TR3
8540=420   NBASE=NBASE+4
8550=430   NBASE=NBASE+NLUP2
8560=440   NBASE=NBASE+NLUP23
8570=800   IF(NA.NE.8) GO TO 1600
8580=C
8590=C ****************************************************
8600=C
8610=C     THE FOLLOWING CODE IMPLEMENTS THE 8 POINT POST-WEAVE
8620=C
8630=C ****************************************************
8640=C
8650=      NLUP2=8*(ND2-NB)
8660=      NLUP23=8*ND2*(ND3-NC)
8670=      NBASE=1
8680=      DO 840 N4=1,ND
8690=      DO 830 N3=1,NC
8700=      DO 820 N2=1,NB
8710=      NR1=NBASE+1
8720=      NR2=NR1+1
8730=      NR3=NR2+1
8740=      NR4=NR3+1
8750=      NR5=NR4+1
8760=      NR6=NR5+1
8770=      NR7=NR6+1
8780=      T1=SR(NBASE)-SR(NR1)
8790=      SR(NBASE)=SR(NBASE)+SR(NR1)
8800=      TR6=SR(NR2)+SI(NR3)
8810=      SR(NR2)=SR(NR2)-SI(NR3)
8820=      T4=SR(NR4)-SI(NR5)
8830=      T5=SR(NR4)+SI(NR5)
8840=      T6=SR(NR7)-SI(NR6)
8850=      T7=SR(NR7)+SI(NR6)
8860=      SR(NR4)=T1
```

```
8870=        SR(NR1)=T4+T6
8880=        SR(NR3)=T4-T6
8890=        SR(NR7)=T5-T7
8900=        SR(NR5)=T5+T7
8910=        T1=SI(NBASE+8)-SI(NR1)
8920=        SI(NBASE+8)=SI(NBASE+8)+SI(NR1)
8930=        T3=SI(NR2)-SR(NR3)
8940=        SI(NR2)=SI(NR2)+SR(NR3)
8950=        T4=SI(NR4)+SR(NR5)
8960=        T5=SI(NR4)-SR(NR5)
8970=        SI(NR6)=T3
8980=        T6=SR(NR6)+SI(NR7)
8990=        T7=SR(NR6)-SI(NR7)
9000=        SI(NR4)=T1
9010=        SI(NR1)=T4+T6
9020=        SI(NR3)=T4-T6
9030=        SI(NR5)=T5+T7
9040=        SI(NR7)=T5-T7
9050=        SR(NR3)=SR3
9060=        SR(NR5)=SR5
9070=        SR(NR6)=SR6
9080=820     NBASE=NBASE+8
9090=830     NBASE=NBASE+NLUP2
9100=840     NBASE=NBASE+NLUP23
9110=1600    IF(NA.NE.16) RETURN
9120=C
9130=C ****************************************************
9140=C
9150=C       THE FOLLOWING CODE IMPLEMENTS THE 16 POINT POST-WEAVE MODUL
E
9160=C
9170=C ****************************************************
9180=C
9190=        NLUP2=18*(ND2-NB)
9200=        NLUP23=18*ND2*(ND3-NC)
9210=        NBASE=1
9220=        DO 1640 N4=1,ND
9230=        DO 1630 N3=1,NC
9240=        DO 1620 N2=1,NB
9250=        NR1=NBASE+1
9260=        NR2=NR1+1
9270=        NR3=NR2+1
9280=        NR4=NR3+1
9290=        NR5=NR4+1
9300=        NR6=NR5+1
9310=        NR7=NR6+1
9320=        NR8=NR7+1
9330=        NR9=NR8+1
9340=        NR10=NR9+1
9350=        NR11=NR10+1
9360=        NR12=NR11+1
9370=        NR13=NR12+1
9380=        NR14=NR13+1
9390=        NR15=NR14+1
9400=        NR16=NR15+1
```

```
9410=       NR17=NR16+1
9420=       T(2)=$R(NBASE)-$R(NR1)
9430=       $R(NBASE)=$R(NR1)+$R(NBASE)
9440=       T(4)=$R(NR2)+$I(NR3)
9450=       T(3)=$R(NR2)-$I(NR3)
9460=       T(6)=$R(NR4)+$I(NR5)
9470=       T(5)=$R(NR4)-$I(NR5)
9480=       T(8)=-$I(NR6)-$R(NR7)
9490=       T(7)=-$I(NR6)+$R(NR7)
9500=       T(9)=$R(NR8)+$R(NR14)
9510=       T(15)=$R(NR8)-$R(NR14)
9520=       T(13)=-$I(NR10)-$I(NR12)
9530=       T(11)=$I(NR10)-$I(NR12)
9540=       T(16)=$R(NR15)-$R(NR17)
9550=       T(12)=$R(NR11)-$R(NR17)
9560=       T(10)=-$I(NR9)-$I(NR16)
9570=       T(14)=-$I(NR16)+$I(NR13)
9580=       $R(NR2)=T(5)+T(7)
9590=       $R6=T(5)-T(7)
9600=       $R10=T(6)+T(8)
9610=       $R(NR14)=T(6)-T(8)
9620=       Q(7)=T(9)+T(10)
9630=       Q(8)=T(9)-T(10)
9640=       Q(1)=T(11)+T(12)
9650=       Q(2)=T(11)-T(12)
9660=       Q(4)=T(14)+T(15)
9670=       Q(5)=T(15)-T(14)
9680=       Q(3)=T(13)+T(16)
9690=       Q(6)=T(13)-T(16)
9700=       $R(NR1)=Q(3)+Q(7)
9710=       $R(NR7)=Q(7)-Q(3)
9720=       $R9=Q(8)+Q(6)
9730=       $R(NR15)=Q(8)-Q(6)
9740=       $R5=Q(1)+Q(4)
9750=       $R3=Q(4)-Q(1)
9760=       $R13=Q(2)+Q(5)
9770=       $R11=Q(5)-Q(2)
9780=       $R(NR6)=T(2)
9790=       $R(NR4)=T(1)
9800=       $R1=T(4)
9810=       T(2)=$I(NBASE)-$I(NR1)
9820=       $I(NBASE)=$I(NR1)+$I(NBASE)
9830=       T(4)=$I(NR2)-$R(NR3)
9840=       T(3)=$I(NR2)+$R(NR3)
9850=       T(6)=$I(NR4)-$R(NR5)
9860=       T(5)=$I(NR4)+$R(NR5)
9870=       T(8)=$R(NR6)-$I(NR7)
9880=       T(7)=$R(NR6)+$I(NR7)
9890=       T(9)=$I(NR8)+$I(NR14)
9900=       T(15)=$I(NR8)-$I(NR14)
9910=       T(13)=$R(NR10)+$R(NR12)
9920=       T(11)=$R(NR12)-$R(NR10)
9930=       T(16)=$I(NR15)-$I(NR17)
9940=       T(12)=$I(NR11)-$I(NR17)
9950=       T(10)=$R(NR9)+$R(NR16)
```

```
 9960=        SI(NR2)=T(5)+T(7)
 9970=        SI(NR6)=T(5)-T(7)
 9980=        SI(NR10)=T(6)+T(8)
 9990=        SI(NR14)=T(6)-T(8)
10000=        Q(7)=T(9)+T(10)
10010=        Q(8)=T(9)-T(10)
10020=        Q(1)=T(11)+T(12)
10030=        Q(2)=T(11)-T(12)
10040=        Q(4)=T(14)+T(15)
10050=        Q(5)=T(15)-T(14)
10060=        Q(3)=T(13)+T(16)
10070=        Q(6)=T(13)-T(16)
10080=        SI(NR1)=Q(3)+Q(7)
10090=        SI(NR7)=Q(7)-Q(3)
10100=        SI(NR9)=Q(8)+Q(6)
10110=        SI(NR15)=Q(8)-Q(6)
10120=        SI(NR5)=Q(1)+Q(4)
10130=        SI(NR3)=Q(4)-Q(1)
10140=        SI(NR13)=Q(2)+Q(5)
10150=        SI(NR11)=Q(5)-Q(2)
10160=        SI(NR8)=T(2)
10170=        SI(NR4)=T(3)
10180=        SI(NR12)=T(4)
10190=        SR(NR3)=SR3
10200=        SR(NR5)=SR5
10210=        SR(NR6)=SR6
10220=        SR(NR9)=SR9
10230=        SR(NR10)=SR10
10240=        SR(NR11)=SR11
10250=        SR(NR12)=SR12
10260=        SR(NR13)=SR13
10270=1620    NBASE=NBASE+18
10280=1630    NBASE=NBASE+NLUP2
10290=1640    NBASE=NBASE+NLUP23
10300=        RETURN
10310=        END
10320=◆EOR
10330=◆EOF
```

Appendix I. Computing the Prime Factor Algorithm (PFA)

This program computes the DFT defined by:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(+j2\pi nk/N) \; ; \; k=0,1, \ldots, N-1$$

where the sequence length N is a product of the relative prime factors from the set (2,3,4,5,7,8,9 and 16). This algorithm was proposed by Kolba and Parks in 1977 and was modified to the program presented here in 1980 by Burrus and Eschenbacher.

Arguments. The PFA is called using the following arguments.

N = The transform length which must be factored into mutually prime factors from the set 2,3,4,5,7,8 and 16. A list of acceptable sequence lengths is given in Table 3.11 - a,b.

X and Y = The real and imaginary data arrays containing the sequence to be transformed. These arrays are dimensioned to length N.

NI = The array containing the factors of N. If all four factors are not used the unused factors are set equal to 1. For example with N=30, we have NI(1)=5, NI(2)=3, NI(3)=2, and NI(4)=1. The factors of one must be the last of the M's.

M = The number of nonunity factors. For N=30, M=3.

UNSC = An output indexing constant which must be precomputed. UNSC = N/(NI(1) + ... + NI(M)).

A and B = Data arrays of length N which contain the results of the DFT. The real part is in A and the imaginary part is in B.

Usage. To compute the forward single-variate DFT:

(1) Dimension X, Y, A, and B to length N.

(2) Define N, M, and NI(4).

(3) Compute UNSC.

(4) Input the sequence to be transformed in x and y.

(5) Call PFA (X,Y,A,B,N,M,NI,UNSC).

(6) The Fourier transform results are located in A and B.

```
100=      SUBROUTINE PFA(X,Y,A,B,N,M,NI,UNSC)
110=C
120=C   A PRIME FACTOR FFT PROGRAM
130=C
140=C  X,Y    -COMPLEX INPUT DATA ARRAY.  REAL DATA IN X AND
150=C          IMAGINARY VALUES IN Y.
160=C  A,B    -COMPLEX OUTPUT VECTORS.  REAL VALUES IN A AND
170=C          IMAGINARY VALUES IN B.
180=C  M      -THE NUMBER OF FACTORS OF N
190=C  N      -SEQUENCE LENGTH WHICH MUST BE FACTORABLE
200=C          BY MUTUALLY PRIME NOS FROM THE SET (2,3,4,5,7,8,9,16)
210=C  NI     -ARRAY LENGTH M CONTAINING THE FACTORS OF N.
220=C  UNSC   -UNSCRAMBLING CONSTANT EQUAL TO N/NI(1)+N/NI(2) +
230=C          ....+N/NI(M).
240=C
250=C   PROGRAM BY C.S. BURRUS
260=C   RICE UNIVERSITY, AUG 1980
270=C
280=      DIMENSION X(N),Y(N),A(N),B(N)
290=      INTEGER NI(4), I(16), UNSC
300=      DATA C31, C32 / 0.86602540, 0.50000000 /
310=      DATA C51, C52 / 0.95105652, 1.53884180 /
320=      DATA C53,C54 /0.36327126, 0.55901699/
330=      DATA C55       /-1.25 /
340=      DATA C71, C72 / -1.16666667, 0.79015647 /
350=      DATA C73, C74 / 0.055854267, 0.7343022 /
360=      DATA C75, C76 /0.44095855, 0.34087293 /
370=      DATA C77, C78 / 0.53396936, 0.87484229 /
380=      DATA C81       /0.70710678/
390=      DATA C92, C93 / 0.93969262, -0.17364818 /
400=      DATA C94, C96 /0.76604444, -0.34202014 /
410=      DATA C97, C98 / -0.98480775, -0.64278761 /
420=      DATA C162, C163 / 0.38268343, 1.30656297 /
430=      DATA C164, C165/ 0.54119610, 0.92387953/
440=      DO 10 K=1,M
450=      N1=NI(K)
460=      N2=N/N1
470=      DO 20 J=1,N,N1
480=         I(1)=J
490=         IT=J
500=         DO 30 L=2,N1
510=            IT=IT+N2
520=            IF(IT.GT.N)   IT=IT-N
530=            I(L)=IT
540=30    CONTINUE
550=            GO TO(20,102,103,104,105,20,107,108,109,
560=     C        20,20,20,20,20,20,116),N1
570=C
580=C   WFTA N=2
590=C
```

```
600=102    T1=X(I(1))
610=       X(I(1))=T1+X(I(2))
620=       X(I(2))=T1-X(I(2))
630=       T1=Y(I(1))
640=       Y(I(1))=T1+Y(I(2))
650=       Y(I(2))=T1-Y(I(2))
660=       GO TO 20
670=C
680=C   WFTA N=3
690=C
700=103    T1=(X(I(2))-X(I(3)))*C31
710=       U1=(Y(I(2))-Y(I(3)))*C31
720=       R1=X(I(2))+X(I(3))
730=       S1=Y(I(2))+Y(I(3))
740=       T2=X(I(1))-R1*C30
750=       U2=Y(I(1))-S1*C30
760=       X(I(1))=X(I(1))+R1
770=       Y(I(1))=Y(I(1))+S1
780=       X(I(2))=T2+U1
790=       X(I(3))=T2-U1
800=       Y(I(2))=U2-T1
810=       Y(I(3))=U2+T1
820=       GO TO 20
830=C
840=C   WFTA N=4
850=C
860=104    R1=X(I(1))+X(I(3))
870=       R2=X(I(1))-X(I(3))
880=       S1=Y(I(1))+Y(I(3))
890=       S2=Y(I(1))-Y(I(3))
900=       R3=X(I(2))+X(I(4))
910=       R4=X(I(2))-X(I(4))
920=       S3=Y(I(2))+Y(I(4))
930=       S4=Y(I(2))-Y(I(4))
940=       X(I(1))=R1+R3
950=       X(I(3))=R1-R3
960=       Y(I(1))=S1+S3
970=       Y(I(3))=S1-S3
980=       X(I(2))=R2+S4
990=       X(I(4))=R2-S4
1000=      Y(I(2))=S2-R4
1010=      Y(I(4))=S2+R4
1020=      GO TO 20
1030=C
1040=C  WFTA N=5
1050=C
1060=105   R1=X(I(2))+X(I(5))
1070=      R2=X(I(2))-X(I(5))
1080=      S1=Y(I(2))+Y(I(5))
1090=      S2=Y(I(2))-Y(I(5))
1100=      R3=X(I(3))+X(I(4))
1110=      R4=X(I(3))-X(I(4))
```

```
1120=       S3=(Y(I(3))+Y(I(4)))
1130=       S4=(Y(I(3))-Y(I(4)))
1140=       T1=(R2+R4)*C51
1150=       U1=(S2+S4)*C51
1160=       R2=T1-R2*C52
1170=       S2=U1-S2*C52
1180=       R4=T1-R4*C53
1190=       S4=U1-S4*C53
1200=       T1=(R1-R3)*C54
1210=       U1=(S1-S3)*C54
1220=       T2=R1+R3
1230=       U2=S1+S3
1240=       X(I(1))=X(I(1))+T2
1250=       Y(I(1))=Y(I(1))+U2
1260=       T2=X(I(1))+T2*C55
1270=       U2=Y(I(1))+U2*C55
1280=       R1=T2+T1
1290=       R3=T2-T1
1300=       S1=U2+U1
1310=       S3=U2-U1
1320=       X(I(2))=R1+S4
1330=       X(I(5))=R1-S4
1340=       Y(I(2))=S1-R4
1350=       Y(I(5))=S1+R4
1360=       X(I(3))=R3-S2
1370=       X(I(4))=R3+S2
1380=       Y(I(3))=S3+R2
1390=       Y(I(4))=S3-R2
1400=       GO TO 20
1410=C
1420=C
1430=C    WFTA N=7
1440=107    R1=X(I(2))+X(I(7))
1450=       R2=X(I(2))-X(I(7))
1460=       S1=Y(I(2))+Y(I(7))
1470=       S2=Y(I(2))-Y(I(7))
1480=       R3=X(I(3))+X(I(6))
1490=       R4=X(I(3))-X(I(6))
1500=       S3=Y(I(3))+Y(I(6))
1510=       S4=Y(I(3))-Y(I(6))
1520=       R5=X(I(4))+X(I(5))
1530=       R6=X(I(4))-X(I(5))
1540=       S5=Y(I(4))+Y(I(5))
1550=       S6=Y(I(4))-Y(I(5))
1560=       T1=R1+R3+R5
1570=       U1=S1+S3+S5
1580=       X(I(1))=X(I(1))+T1
1590=       Y(I(1))=Y(I(1))+U1
1600=       T1=X(I(1))+T1*C71
1610=       U1=Y(I(1))+U1*C71
1620=       T2=C72*(R1-R5)
1630=       U2=C72*(S1-S5)
1640=       T3=C73*(R5-R3)
```

```
1650=         U3=C73*(S5-S3)
1660=         T4=C74*(R5-R1)
1670=         U4=C74*(S5-S1)
1680=         R1=T1+T2+T3
1690=         R3=T1-T2-T4
1700=         R5=T1-T3+T4
1710=         S1=U1+U2+U3
1720=         S3=U1-U2-U4
1730=         S5=U1-U3+U4
1740=         U1=C75*(S2+S4-S6)
1750=         T1=C75*(R2+R4-R6)
1760=         T2=C76*(R2+R6)
1770=         U2=C76*(S2+S6)
1780=         T3=C77*(R4+R6)
1790=         U3=C77*(S4+S6)
1800=         T4=C78*(R4-R2)
1810=         U4=C78*(S4-S2)
1820=         R2=T1+T2+T3
1830=         R4=T1-T2-T4
1840=         R6=T1-T3+T4
1850=         S2=U1+U2+U3
1860=         S4=U1-U2-U4
1870=         S6=U1-U3+U4
1880=         X(I(2))=R1+S2
1890=         X(I(7))=R1-S2
1900=         Y(I(2))=S1-R2
1910=         Y(I(7))=S1+R2
1920=         X(I(3))=R3+S4
1930=         X(I(6))=R3-S4
1940=         Y(I(3))=S3-R4
1950=         Y(I(6))=S3+R4
1960=         X(I(4))=R5-S6
1970=         X(I(5))=R5+S6
1980=         Y(I(4))=S5+R6
1990=         Y(I(5))=S5-R6
2000=         GO TO 20
2010=C
2020=C    DATA C=F
2030=C
2040=108      R1=X(I(1))+X(I(5))
2050=         R2=X(I(1))-X(I(5))
2060=         S1=Y(I(1))+Y(I(5))
2070=         S2=Y(I(1))-Y(I(5))
2080=         R3=X(I(2))+X(I(8))
2090=         R4=X(I(2))-X(I(8))
2100=         S3=Y(I(2))+Y(I(8))
2110=         S4=Y(I(2))-Y(I(8))
2120=         R5=X(I(3))+X(I(7))
2130=         R6=X(I(3))-X(I(7))
2140=         S5=Y(I(3))+Y(I(7))
2150=         S6=Y(I(3))-Y(I(7))
2160=         R7=X(I(4))+X(I(6))
```

```
2170=         R2=X(I(4))-X(I(6))
2180=          [illegible]
2190=          S2=Y(I(4))-Y(I(6))
2200=         T1=R1+R5
2210=         T2=R1-R5
2220=         U1=S1+S5
2230=         U2=S1-S5
2240=         T3=R3+R7
2250=         R3=(R3-R7)*C81
2260=         U3=S3+S7
2270=         S3=(S3-S7)*C81
2280=         T4=R4-R8
2290=         R4=(R4+R8)*C81
2300=         U4=S4-S8
2310=         S4=(S4+S8)*C81
2320=         T5=R2+R3
2330=         T6=R2-R3
2340=         U5=S2+S3
2350=         U6=S2-S3
2360=         T7=R4+R6
2370=         T8=R4-R6
2380=         U7=S4+S6
2390=         U8=S4-S6
2400=         X(I(1))=T1+T3
2410=         X(I(5))=T1-T3
2420=         Y(I(1))=U1+U3
2430=         Y(I(5))=U1-U3
2440=         X(I(2))=T5+U7
2450=         X(I(8))=T5-U7
2460=         Y(I(2))=U5-T7
2470=         Y(I(8))=U5+T7
2480=         X(I(3))=T2+U4
2490=         X(I(7))=T2-U4
2500=         Y(I(3))=U2-T4
2510=         Y(I(7))=U2+T4
2520=         X(I(4))=T6+U8
2530=         X(I(6))=T6-U8
2540=         Y(I(4))=U6-T8
2550=         Y(I(6))=U6+T8
2560=           RETURN
2570=C
2580=C   WFTA N=9
2590=C
2600=109      R1=X(I(2))+X(I(9))
2610=         R2=X(I(2))-X(I(9))
2620=         S1=Y(I(2))+Y(I(9))
2630=         S2=Y(I(2))-Y(I(9))
2640=         R3=X(I(3))+X(I(8))
2650=         R4=X(I(3))-X(I(8))
2660=         S3=Y(I(3))+Y(I(8))
2670=         S4=Y(I(3))-Y(I(8))
2680=         R5=X(I(4))+X(I(7))
```

```
2690=      T=-([illegible](I(4))-[illegible](I(7)))*C[illegible]1
2700=      S=([illegible](I(4))+([illegible](I([illegible]))
[illegible]      U=-([illegible](I(4))-[illegible](I(7)))*C[illegible]1
2720=      R7=([illegible](I(5))+[illegible](I(6)))
[illegible]      [illegible]=([illegible](I(5))-[illegible](I(6)))
2740=      T7=([illegible](I(5))+Y(I(6)))
2750=      [illegible]=([illegible](I(5))-Y(I(6)))
[illegible]      [illegible]=([illegible](I(1))+R[illegible]
[illegible]      [illegible]=([illegible](I(1))+[illegible]5
[illegible]      T1=([illegible](I(1))-R5)*[illegible]
[illegible]      U1=([illegible](I(1))-[illegible]5)*[illegible]
2800=      [illegible]=([illegible]-[illegible])*[illegible]
2810=      U2=([illegible]-[illegible])*C92
2820=      T3=(R1-R7)*C93
[illegible]      [illegible]=([illegible]1-[illegible]7)*C9[illegible]
[illegible]      [illegible]=([illegible]-[illegible])*[illegible]
2850=      U4=([illegible]1-[illegible])*C94
2860=      R10=R1+R3+R7
2870=      [illegible]10=S1+S3+S7
2880=      R1=T1+T2+T4
2890=      R3=T1-T2-T3
2900=      R7=T1+T3-T4
2910=      S1=U1+U2+U4
2920=      S3=U1-U2-U3
2930=      S7=U1+U3-U4
2940=      X(I(1))=R9+R10
2950=      Y(I(1))=S9+S10
2960=      R5=R9-R10*C32
2970=      S5=S9-S10*C32
2980=      R6=-(R2-R4+R8)*C31
2990=      [illegible]6=-([illegible]2-[illegible]4+[illegible]8)*C31
3000=      T2=(R4+R8)*C96
3010=      U2=([illegible]4+[illegible]8)*C96
3020=      T3=(R2-R8)*C97
3030=      U3=([illegible]-[illegible])*C97
3040=      T4=(R2+R4)*C98
[illegible]      [illegible]4=([illegible]+[illegible]4)*C98
3060=      R[illegible]=T+T2+T4
[illegible]      [illegible]4=[illegible]-T2-[illegible]
[illegible]      [illegible]
3090=      [illegible]2=U+U2+U4
3100=      [illegible]4=U-U2-U3
3110=      [illegible]6=U+U3-U4
3120=      [illegible](I(2))=R1-[illegible]2
3130=      [illegible](I(9))=R1+[illegible]2
3140=      Y(I(2))=[illegible]1+R2
3150=      [illegible](I(9))=[illegible]1-R2
[illegible]      [illegible]
3170=      X(I(8))=R3-[illegible]4
3180=      Y(I(3))=[illegible]-R4
3190=      Y(I(8))=[illegible]3+R4
3200=      X(I(4))=R5-[illegible]6
```

```
3210=        X(I(7))=R5+T6
3220=        Y(I(4))=T5+S6
3230=        Y(I(7))=T5-S6
3240=        X(I(5))=R7-T8
3250=        X(I(6))=R7+T8
3260=        Y(I(5))=T7+S8
3270=        Y(I(6))=T7-S8
3280=        GO TO 20
3290=C
3300=C     [illegible] N=16
3310=C
3320=116     R1=X(I(1))+X(I(9))
3330=        R2=X(I(1))-X(I(9))
3340=        S1=Y(I(1))+Y(I(9))
3350=        S2=Y(I(1))-Y(I(9))
3360=        R3=X(I(2))+X(I(10))
3370=        R4=X(I(2))-X(I(10))
3380=        S3=Y(I(2))+Y(I(10))
3390=        S4=Y(I(2))-Y(I(10))
3400=        R5=X(I(3))+X(I(11))
3410=        R6=X(I(3))-X(I(11))
3420=        S5=Y(I(3))+Y(I(11))
3430=        S6=Y(I(3))-Y(I(11))
3440=        R7=X(I(4))+X(I(12))
3450=        R8=X(I(4))-X(I(12))
3460=        S7=Y(I(4))+Y(I(12))
3470=        S8=Y(I(4))-Y(I(12))
3480=        R9=X(I(5))+X(I(13))
3490=        R10=X(I(5))-X(I(13))
3500=        S9=Y(I(5))+Y(I(13))
3510=        S10=Y(I(5))-Y(I(13))
3520=        R11=X(I(6))+X(I(14))
3530=        R12=X(I(6))-X(I(14))
3540=        S11=Y(I(6))+Y(I(14))
3550=        S12=Y(I(6))-Y(I(14))
3560=        R13=X(I(7))+X(I(15))
3570=        R14=X(I(7))-X(I(15))
3580=        S13=Y(I(7))+Y(I(15))
3590=        S14=Y(I(7))-Y(I(15))
3600=        R15=X(I(8))+X(I(16))
3610=        R16=X(I(8))-X(I(16))
[illegible]
3630=        S16=Y(I(8))-Y(I(16))
3640=        T1=R1+R9
3650=        T2=R1-R9
3660=        U1=S1+S9
3670=        U2=S1-S9
3680=        T3=R3+R11
3690=        T4=R3-R11
3700=        U3=S3+S11
3710=        U4=S3-S11
3720=        T5=R5+R13
```

```
3730=        T6=R5-R13
3740=        U5=S5+S13
3750=        U6=S5-S13
3760=        T7=R7+R15
3770=        T8=R7-R15
3780=        U7=S7+S15
3790=        U8=S7-S15
3800=        T9=C81●(T4+T8)
3810=        T10=C81●(T4-T8)
3820=        U9=C81●(U4+U8)
3830=        U10=C81●(U4-U8)
3840=        R1=T1+T5
3850=        R3=T1-T5
3860=        S1=U1+U5
3870=        S3=U1-U5
3880=        R5=T3+T7
3890=        R7=T3-T7
3900=        S5=U3+U7
3910=        S7=U3-U7
3920=        R9=T2+T10
3930=        R11=T2-T10
3940=        S9=U2+U10
3950=        S11=U2-U10
3960=        R13=T6+T9
3970=        R15=T6-T9
3980=        S13=U6+U9
3990=        S15=U6-U9
4000=        T1=R4+R16
4010=        T2=R4-R16
4020=        U1=S4+S16
4030=        U2=S4-S16
4040=        T3=C81●(R6+R14)
4050=        T4=C81●(R6-R14)
4060=        U3=C81●(S6+S14)
4070=        U4=C81●(S6-S14)
4080=        T5=R8+R12
4090=        T6=R8-R12
4100=        U5=S8+S12
4110=        U6=S8-S12
4120=        T7=C1[illegible]●(T6-T[illegible])
4130=        T[illegible]=C1[illegible]●(T6-T7)
4140=        [illegible]=C1[illegible]4●(T6-T7)
4150=        T10=R2+T4
4160=        T11=R2-T4
4170=        R2=T10+T8
4180=        R4=T10-T8
4190=        R6=T11+T9
4200=        R8=T11-T9
4210=        [illegible]
4220=        U8=C164●U6-U7
4230=        U9=C164●U6-U7
4240=        U10=S2+U4
```

```
4250=       U11=[illegible]-U14
[illegible]
[illegible]
4280=       [illegible]=U11+U9
4290=       [illegible]
4300=       T7=C165*(T1+T5)
4310=       T8=T7-C164*T1
4320=       T9=T7-C163*T5
4330=       T10=S11+T[illegible]
4340=       T11=S11-T[illegible]
4350=       R10=T10+T8
4360=       R12=T10-T8
4370=       R14=T11+T9
4380=       R16=T11-T9
4390=       U7=C165*(U1+U5)
4400=       U8=U7-C164*U1
4410=       U9=U7-C163*U5
4420=       U10=S10+U[illegible]
4430=       U11=S10-U3
4440=       S10=U10+U8
4450=       S12=U10-U8
4460=       S14=U11+U9
4470=       S16=U11-U9
4480=       X(I(1))=R1+R5
4490=       X(I(9))=R1-R5
4500=       Y(I(1))=S1+S5
4510=       Y(I(9))=S1-S5
4520=       X(I(2))=R2+S10
4530=       X(I(16))=R2-S10
4540=       Y(I(2))=S2-R10
4550=       Y(I(16))=S2+R10
4560=       X(I(3))=R9+S13
4570=       X(I(15))=R9-S13
4580=       Y(I(3))=S9-R13
4590=       Y(I(15))=S9+R13
4600=       X(I(4))=R8-S16
4610=       X(I(14))=R8+S16
4620=       Y(I(4))=[illegible]+R16
4630=       Y(I(14))=[illegible]-R16
[illegible]
[illegible]
[illegible]
4670=       Y(I(13))=S3+R7
4680=       X(I(6))=R6+S14
4690=       X(I(12))=R6-S14
4700=       Y(I(6))=S6-R14
4710=       Y(I(12))=S6+R14
4720=       X(I(7))=R11-S15
4730=       X(I(11))=R11+S15
4740=       Y(I(7))=S11+R15
4750=       Y(I(11))=S11-R15
4760=       X(I(8))=R4-S12
```

```
4770=          X(I(10))=R4+T12
4780=          Y(I(9))=T4+R12
4790=          Y(I(10))=T4-R12
4800=          GO TO 20
4810=20         CONTINUE
4820=10         CONTINUE
4830=C
4840=C   UNSCRAMBLING
4850=C
4860=          L=1
4870=          DO 2 K=1,N
4880=          A(K)=X(L)
4890=          B(K)=Y(L)
4900=          L=L+UNSC
4910=          IF(L.GT.N) L=L-N
4920=          IF(L.GT.N) L=L-N
4930=2         CONTINUE
4940=          RETURN
4950=          END
4960=*EOR
4970=*EOF
```

Appendix J. Timing Tests on the CDC Cyber 74

The timing tests on the CDC Cyber 74 used the FORTRAN command SECOND(CP) which, according to the FORTRAN IV reference manual, returns time accurate to "two decimal places", i.e., 0.01 seconds. The results of timing the various DFT algorithms showed this clock was accurate to three decimal places (0.001 seconds) giving a time resolution of 0.002 seconds. Using three decimal places was justified since almost every standard deviation was less than or equal to 0.002 seconds.

To verify the premise that counting the real operations performed in a DFT is the primary factor determining execution speed of the algorithm on a computer, the DFT execution times were measured on the CDC Cyber 74. The execution speeds for the WFTA, PFA, and the mixed/fixed radix FFTs were compared to the "predicted" execution speed of the algorithm. To perform these comparisons the multiply and add speeds were determined for the Cyber 74 computer.

The execution times of the floating point multiply and add instructions are given in the CDC 6000 Series Computer Systems Reference Manual. The execution times for several instructions are listed below and include preparing the <u>next</u> instruction for execution:

Instruction	Assembly Language	Minor Cycles
Floating sum	FX_i	4
Floating product	FX_i	10
Normalize result	NX_i	4
Fetch/store	SA_i	3

where one minor cycle equals 0.1 microsecond (μs). Simply using an add time of 4*0.1μs and a multiply time of 10*0.1μs = 1μs is not sufficient because the operands must be fetched and stored which adds more time. To determine the commands executed by the computer for adds and multiplies the assembly (COMPASS) language was studied and timed for three cases. First, the DO loop with no operations was executed 100,000 times:

```
      DO  102  J = 1,N
  102 CONTINUE
```

The associated COMPASS language code was listed as an output of the program:

```
(AA      BSS      OB
         SBO      B2 + 7B
         SA5      J
         SA4      N
         SX7      X5 + 1B
         IXO      X4 - X7
         SA7      A5
         PL       X5, (AA
```

This loop required an average of 2.70μs (standard deviation 0.03μs) to execute. Next the addition instruction was executed 100,000 times using the FORTRAN code:

```
           DO   102    J = 1,N
      102  TAD = A + B
```

The associated COMPASS code for the addition loop is:

```
    (AA    BSS      OB
           SBO      B2 + 7B
           SA5      A
           SA4      B
           SA3      J
           SA2      N
           FXO      X4 + X5
           NX7      BO, XO
           SX6      X3 + 1B
           IX5      X2 - X6
           SA6      A3
           SA7      TAD
           PL       X5, (AA
```

This add loop required an average of 3.34μs (standard deviation 0.3μs) to execute. Notice the "extra" instructions of the add loop versus the no operation loop:

Command		Minor Cycles
SA5	A	3
SA4	B	3
FXO	X4 + X5	3
NX7	BO, XO	4
SA7	TAD	3
		17

Finally the multiply loop was executed 100,000 times.

The FORTRAN code is:

```
          DO   102   J = 1, N
     102  TAD = A*B
```

and the corresponding COMPASS code loop is:

```
    )AA      BSS        OB
             SBO        B2 + 7B
             SA5        B
             SA4        A
             SA3        J
             SA2        N
             FX7        X4*X5
             SX6        X3 + 1B
             IXO        X2 - X6
             SA6        A3
             SA7        TAD
             PL         X5, )AA
```

The multiply loop averaged 3.37μs (standard deviation 0.03) to execute. The extra instructions required for the multiply loop relative to the no operation loop are:

Command		Minor Cycles
SA5	B	3
SA4	A	3
FX7	X4 * X5	10
SA7	TAD	3
		19

Comparing the measured execution times of the three loops shows the add loop is 0.64μs longer. Based on the minor cycle times for the extra add and multiply commands, the add loop should be 17*0.1μs longer and the multiply loop should be 19*0.1μs = 1.9μs longer than the "no operation" loop. (Notice that every floating point addition must be "normalized" by the command NX7 which requires 4 minor cycles. The floating point sum does not require normalization).

The difference in measured add and multiply speed (0.64μs and 0.67μs) versus the predicted add and multiply speed (1.7μs and 1.9μs) is a result of the very short loops fitting inside the Cyber's "instruction/execution stack" which is a 12 word stack with 60 bits per word. Since the entire loop could fit in the stack the instructions were fetched only once instead of 100,000 times, whereas "all execution times (minor cycles) listed include readying the next instruction for execution". During normal DFT algorithm execution of all of the instructions must be fetched which means the add speed is 1.7μs and the multiply speed is 1.9μs. These numbers were then used to predict execution speed of the DFT algorithms.

Vita

John David Blanken was born on 18 April 1953 in Junction City, Kansas. He graduated from Junction City High School in 1971 and attended Kansas State University from which he received a Bachelor of Science in Electrical Engineering in May 1975. Upon graduation, he was designated an AFROTC distinguished graduate and received a commission in the United States Air Force. He entered active duty 15 July 1975 and was assigned to the Air Force Avionics Laboratory at Wright-Patterson Air Force Base, Ohio. On June 6, 1979 he was assigned to the School of Engineering, Air Force Institute of Technology.

Permanent Address: 716 West 8th Street
Junction City, Kansas
66441

Unclassified

REPORT DOCUMENTATION PAGE

AD-A100 782

[illegible] Implementations of Fast Fourier [illegible]

MS Thesis

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

John P. [illegible], Captain, USAF

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Air Force Institute of Technology (AFIT-EN)
Wright-Patterson AFB, Ohio 45433

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

AFC/[illegible]
Wright-Patterson AFB, Ohio 45433

12. REPORT DATE

December 1980

13. NUMBER OF PAGES

302

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS (of this report)

Unclassified

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17 [illegible] 1981
Frederick C. Lynch, Major, USAF
Director of Public Affairs
Frederic C. Lynch

19. KEY WORDS

Fast Fourier Transform
Discrete Fourier Transform
Winograd Fourier Transform

20. ABSTRACT

A comprehensive comparison of the most efficient Discrete Fourier Transform (DFT) techniques is presented. The DFT algorithms selected are the fixed radix Fast Fourier Transform (FFT), mixed radix FFT, the Winograd Fourier Transform Algorithm (WFTA), and the Prime Factor Algorithm (PFA). Comparison of the algorithms is based on the number of real multiplications, additions, and memory arrays required as a function of sequence length N. This [illegible] the most efficient DFT

DD [illegible]

Unclassified

20. Abstract (Continued)

FOR [illegible] programs available, divided [illegible] the number of real multi-
plications [illegible]
[illegible],
additions, and memory [illegible]. This comparison shows that the
WFTA and PFA require the least number of multiplications and additions,
but the [illegible] mixed radix FFTs require the least
memory. The mixed radix FFT is much more flexible than WFTA
or PFA since N can be any length sequence. The WFTA and PFA
are closely studied and tradeoffs between the two are discussed.
The PFA uses less additions but more multiplications for most
sequence lengths which means the WFTA is more efficient when
multiplications are "costly" relative to additions. The PFA
uses less memory than the WFTA making the PFA preferable when
the machine memory is limited. Based on the results of the paper,
an algorithm is presented to select the most efficient DFT for
an N length sequence given the multiply speed, add speed, and
memory size of the computer.