Mathematics of Multidimensional Fourier Transform Algorithms

Second Edition

Springer Science+Business Media, LLC

Richard Tolimieri Myoung An Chao Lu

Mathematics of Multidimensional Fourier Transform Algorithms

Second Edition

C.S. Burrus
Consulting Editor

With 21 Illustrations

Springer

Signal Processing and Digital Filtering

Synthetic Aperture Radar
J.P. Fitch

Multiplicative Complexity, Convolution and the DFT
M.T. Heideman

Array Signal Processing
S.U. Pillai

Maximum Likelihood Deconvolution
J.M. Mendel

Algorithms for Discrete Fourier Transform and Convolution
T. Tolimieri, M. An, and C. Lu

Algebraic Methods for Signal Processing and Communications Coding
R.E. Blahut

Electromagnetic Devices for Motion Control and Signal Processing
Y.M. Pulyer

Mathematics of Multidimensional Fourier Transform Algorithms, Second Edition
R. Tolimieri, M. An, and C. Lu

Lectures on Discrete Time Filtering
R.S. Bucy

Distributed Detection and Data Fusion
P.K. Varshney

Richard Tolimieri
Department of Electrical Engineering
City College of CUNY
New York, NY 10037, USA

Myoung An
A.J. Devaney Associates
52 Ashford Street
Allston, MA 02134, USA

Chao Lu
Department of Computer and
Information Sciences
Towson State University
Towson, MD 21204, USA

Consulting Editor
Signal Processing and Digital Filtering

C.S. Burrus
Professor and Chairman
Department of Electrical and
Computer Engineering
Rice University
Houston, TX 77251-1892, USA

Library of Congress Cataloging-in-Publication Data
Tolimieri, Richard, 1940–
Mathematics of multidimensional Fourier transform algorithms /
Richard Tolimieri, Myoung An, Chao Lu. — 2nd ed.
p. cm. — (Signal processing and digital filtering)
Includes bibliographical references and index.
ISBN 978-1-4612-7352-3 ISBN 978-1-4612-1948-4 (eBook)
DOI 10.1007/978-1-4612-1948-4
1. Fourier transformations. I. An, Myoung. II. Lu, Chao.
III. Title. IV. Series
QA403.5.T655 1997
515′.723—dc21 97-16669

Printed on acid-free paper.

Originally published by Springer-Verlag New York, Inc. in 1997
Softcover reprint of the hardcover 1st edition 1997

Production managed by Steven Pisano; manufacturing supervised by Joe Quatela.
Photocomposed pages prepared from the authors' LaTeX files.

9 8 7 6 5 4 3 2 1

ISBN 978-1-4612-7352-3

Preface

The Fourier transform of large multidimensional data sets is an essential computation in many scientific and engineering fields, including seismology, X-ray crystallography, radar, sonar, and medical imaging. Such fields require multidimensional arrays for complete and faithful modeling. Classically, a set of data is processed one dimension at a time, permitting control over the size of the computation and calling on well-established 1-dimensional programs. The rapidly increasing availability of powerful computing chips, vector processors, multinode boards, and parallel machines has provided new tools for carrying out multidimensional computations. Multidimensional processing offers a wider range of possible implementations as compared to 1-dimensional processing, due to the greater flexibility of movement in the data indexing set. This increased freedom, along with the massive size data sets typically found in multidimensional applications, places intensive demands on the communication aspects of the computation. The writing of code that takes into account all the algorithmic possibilities and matches these possibilities to the communication capabilities of the target architecture is an extremely time-consuming task.

In the first chapter, we will review the tensor product and its role in modeling 1-dimensional Fourier transform algorithms. Tensor product formalism provides powerful tools for keeping track of an algorithm's granularity and index calculations. The algebra of the tensor product, especially that of an associated class of permutations, the stride permutations, plays a fundamental role in establishing simple rules, in the form of tensor product identities for modifying the communication requirements of a computation and for greatly simplifying the generation of code, which can take advan-

tage of computational power and communication bandwidth of a target processor. A large part of code generation can take place on the level of algebraic manipulation rather than in the more time-consuming and error-prone level of programming manipulation.

The second chapter extends the role of tensor product to multidimensions. The main idea is that 1-dimensional stride permutations have exact analogues in a multidimensional setting. Vector segmentation, which plays such an important role in describing 1-dimensional stride permutations, is replaced by array segmentation. In form, 1-dimensional FFT is exactly the same as multidimensional FFT except that the interpretation of parallel and vector operations in terms of vector segmentation must now be made in relation to array segmentation. This class of multidimensional FFT was introduced by G.E. Rivard (1977) and D.B. Harris et al. (1977) and goes by the name of Vector–radix FFT.

A major goal of this text is to provide an abstract treatment of the Fourier transform that highlights the relatively few basic mathematical principles that underlie a large part of 1-dimensional and multidimensional Fourier transform theory and implementation. By emphasizing the unity underlying these algorithms, we can achieve deeper insights into their differences and learn how these differences can be exploited in implementation.

In chapter 3, we begin the mathematics required for the abstract definition of Fourier transform. Finite abelian groups will play a central role in this theory. The concept of duality will be introduced. In chapter 4, we define the Fourier transform of a finite abelian group that is canonical in the sense that it is coordinate-free. This approach brings out the role played by duality in Fourier transform theory and implementation. Duality is built into the definition of the Fourier transform. The duality between periodization and decimation will play a fundamental role in the design of Fourier transform algorithms. The results on periodization and decimation unify all additive 1- and multidimensional Fourier transform algorithms, including the Cooley–Tukey FFT, discussed in chapter 5, and the reduced transform algorithm, discussed in chapter 8.

The algorithms of chapter 5 and chapter 8 are based on the relationship between the Fourier transform of data indexed by some finite abelian group A and the Fourier transform of data indexed by subgroups and quotient groups of A. Theorems in chapter 4 explicitly describe this relationship in terms of duality and periodization-decimation.

In chapters 6 and 7 we distinguish a special collection of subgroups of A called lines and planes and describe their relationship to duality. These results are directly applied to the reduced transform algorithms of chapter 8. However, this is but one of several important applications. In applications where a complete Fourier transform may not be required, these results can be used to compute partial Fourier transforms, i.e., the Fourier transform only on some line or plane of the output without having to compute the entire Fourier transform.

The reduced transform algorithm completes the computation by computing the Fourier transform on a covering set of lines or planes. The general structure of the Cooley–Tukey FFT as described in chapter 5 and that of the reduced transform algorithm are very similar. The first step in the Cooley–Tukey FFT is a collection of generalized periodizations that involve complex multiplications, while the first step in the reduced transform algorithm is a collection of periodizations that involve only additions.

The generalized periodizations of the Cooley–Tukey FFT can be computed by Fourier transforms, but in order to do so full data transposition is required. The periodizations in the reduced transform algorithms greatly increase the number of additions required to carry out the computation. However, in a highly parallel environment, the reduced transform algorithm can have the advantage, since these additions can be carried out in parallel during the data-loading process (for some parallel architectures). Depending on the degree of parallelism and transform size, the reduced transform algorithm may avoid data transposition entirely, greatly reducing the need for interprocessor communication, which is typically limited on large multinode processors. The role of these algorithms and hybrids produced by combining the two is still an active research area.

In chapter 9, multiplicative DFT algorithms are discussed. We have concentrated on the case where the indexing set is a field. We bring out the role of the reduced transform algorithm in deriving several well-known multiplicative multidimensional DFT algorithms due to Nussbaumer–Quandalle and Auslander–Feig–Winograd. Starting from Rader's 1-dimensional algorithm, we show that a multidimensional field algorithm can be derived using exactly the same approach. In fact, the basic algorithm depends only on the fact that the indexing set is a field. The interpretation of a field algorithm as a multidimensional DFT is a simple consequence of the results in chapter 4. A purely multiplicative derivation of the Auslander et al. result is given in terms of factorizations of skew-circulant matrices.

In chapters 10 and 11 we explore the implementation of different DFT algorithms on RISC and parallel architectures. We introduce machine parameters and show how they are related to algorithmic parameters and how to write code based on optimal matching of these parameters.

We wish to thank Dr. James W. Cooley for his many valuable discussions on algorithm design and implementation, and DARPA for its support during the formative years of writing this book. We also wish to thank AFOSR for its support during the last two years, in which time the ideas of this book have been tested and refined in applications to electromagnetics, multispectral imaging, and imaging through turbulance. Whatever improvements in this revision are due to the routines written in these applications.

Richard Tolimieri
Myoung An
Chao Lu

Contents

1
Tensor Product

1.1 Introduction

Tensor product notation can be used to mathematically model, in terms of matrix factorizations, computations from many diverse fields, including digital signal processing, linear systems, and numerical analysis. Typically, large data sets are processed by algorithms characterized by intricate index calculations. The problem of analyzing and writing code for such algorithms that is tailored to a specific architecture or processor is both time-consuming and error-prone. The formalism of the tensor product notation provides powerful tools for keeping track of these index calculations and for establishing simple rules, in the form of tensor product identities, that can be used to modify an algorithm for optimal performance as data size and target architecture vary. By mapping certain basic tensor product operations onto code or hardware, a large array of algorithms can be implemented by simple algebraic manipulations rather than more time-consuming programming manipulations.

The basic algebraic properties of tensor products were established many years before the invention of the computer. They played important roles in Riemannian geometry at the beginning of this century. The tensor product as an implementation tool for the finite Fourier transform dates from a paper of Pease in 1968 [26]. In the years following, tensor product identities have been rediscovered many times. Their role in applications has varied during this period from that of a notational convenience for describing complex computation to that of an interactive programming tool. In this latter

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

role, the algebra of the tensor product, especially that of an associated class of permutations, the *stride permutations*, is fundamental. This algebra is a branch of mathematics, divorced from applications. However, its impact on code design is considerable. It permits the systematic study of a large number of programming options by providing algorithmic parameters that can be identified with such machine parameters as locality, granularity, and communication bandwidth.

Recent interest in the tensor product is largely motivated by the critical problem of controlling memory hierarchy and communication on vector and parallel machines. Stride permutations and tensor products of stride permutations govern the addressing between stages of a tensor product decomposition. These permutations are reflected in code by interchange in the orders of nested loops. On many processors, it can even be the case that certain stride permutations are carried out by machine instructions, greatly simplifying code writing. The algebra of stride permutations can be used to build highly intricate readdressing code based on the judicious combining of a relatively small number of core routines.

In this chapter, we review and summarize, usually without proof, basic definitions and results. Since our prime interest is in analyzing an algorithm's communication requirements and in designing tools for mapping this communication onto code or hardware, our emphasis will be on the algebra of stride permutations.

1.2 Tensor Product

Let $\mathbf{C}^M$ denote the M-dimensional vector space of M-tuples of complex numbers. A typical point $\mathbf{x} \in \mathbf{C}^M$ is a column vector

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{M-1} \end{bmatrix}.$$

If the size of $\mathbf{x} \in \mathbf{C}^M$ is important, we write $\mathbf{x}$ as $\mathbf{x}^M$. The tensor product of two vectors $\mathbf{x} \in \mathbf{C}^M$ and $\mathbf{y} \in \mathbf{C}^L$ is the vector

$$\mathbf{x} \otimes \mathbf{y} = \begin{bmatrix} x_0\mathbf{y} \\ x_1\mathbf{y} \\ \vdots \\ x_{M-1}\mathbf{y} \end{bmatrix} \in \mathbf{C}^{ML},$$

formed by replacing each component x_m of $\mathbf{x}$ by the L-tuple $x_m\mathbf{y}$, $0 \le m < M$. The tensor product is bilinear.

$$\mathbf{x} \otimes (\mathbf{y} + \mathbf{z}) = \mathbf{x} \otimes \mathbf{y} + \mathbf{x} \otimes \mathbf{z}, \qquad \mathbf{x} \in \mathbf{C}^M, \ \mathbf{y}, \mathbf{z} \in \mathbf{C}^L, \tag{1.1}$$

$$(\mathbf{x}+\mathbf{y})\otimes\mathbf{z}=\mathbf{x}\otimes\mathbf{z}+\mathbf{y}\otimes\mathbf{z},\qquad \mathbf{x},\mathbf{y}\in\mathbf{C}^M,\ \ \mathbf{z}\in\mathbf{C}^L. \tag{1.2}$$

Denote by e_m^M, $0\le m<M$, the vector in $\mathbf{C}^M$ with a 1 in the m-th place and 0 elsewhere. The set of vectors

$$\{e_m^M:\quad 0\le m<M\} \tag{1.3}$$

is a basis of $\mathbf{C}^M$, called the *standard basis.* Set $N=ML$ and form the set of tensor products

$$e_m^M\otimes e_l^L,\qquad 0\le m<M,\ 0\le l<L. \tag{1.4}$$

Since

$$e_{l+mL}^N=e_m^M\otimes e_l^L,\qquad 0\le m<M,\ 0\le l<L,$$

the set (1.4) ordered by choosing l the faster running parameter is the standard basis of $\mathbf{C}^N$. In particular, the set of tensor products of the form

$$\mathbf{x}^M\otimes\mathbf{y}^L$$

spans $\mathbf{C}^N$.

Let X denote an $M\times R$ matrix and Y denote an $L\times S$ matrix. Set $N=LM$ and $T=RS$. The $N\times T$ matrix

$$\begin{bmatrix} x_{0,0}Y & x_{0,1}Y & \cdots & x_{0,R-1}Y \\ x_{1,0}Y & x_{1,1}Y & \cdots & x_{1,R-1}Y \\ \vdots & & & \\ x_{M-1,0}Y & x_{M-1,1}Y & \cdots & x_{M-1,R-1}Y \end{bmatrix}$$

is called the *tensor product* of X by Y and is denoted by $X\otimes Y$. The tensor product $X\otimes Y$ acts on $\mathbf{C}^T$, and the action on tensor products

$$\mathbf{x}^R\otimes\mathbf{y}^S$$

is given by

$$(X\otimes Y)(\mathbf{x}^R\otimes\mathbf{y}^S)=(X\mathbf{x}^R)\otimes(Y\mathbf{y}^S). \tag{1.5}$$

More generally, we have the following result.

Theorem 1.1 Multiplication Theorem of Tensor Products *If X is an $M\times R$ matrix, Y an $L\times S$ matrix, A an $R\times U$ matrix, and B an $S\times V$ matrix, then*

$$(X\otimes Y)(A\otimes B)=(XA)\otimes(YB).$$

The proof follows by computing the action of both sides on vectors of the form $\mathbf{x}^U\otimes\mathbf{y}^V$ and applying (1.5). Similar arguments prove the following tensor product identities.

For matrices of appropriate sizes, we have the

Associative Law

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

Distributive Law

$$(A + B) \otimes C = (A \otimes C) + (B \otimes C),$$

$$A \otimes (B + C) = (A \otimes B) + (A \otimes C).$$

To prove the associative law, we first prove it for vectors and then use (1.5) to extend it to matrices. The associative law permits the inductive definition of multidimensional tensor products of vectors and matrices. Multidimensional tensor products provide a natural setting for multidimensional problems. In particular, if $N = N_1 N_2 \cdots N_K$, then the set of tensor products

$$e_{n_1}^{N_1} \otimes \cdots \otimes e_{n_K}^{N_K}, \qquad 0 \le n_k < N_k, \quad 1 \le k \le K,$$

ordered by choosing N_K as the fastest running parameter followed in order by $N_{K-1}, \ldots, N_1$ is the standard basis of $\mathbf{C}^N$. We will often prove tensor product identities by showing that two matrix actions are equal on the set of tensor products of the form

$$\mathbf{x}_1^{N_1} \otimes \mathbf{x}_2^{N_2} \otimes \cdots \otimes \mathbf{x}_K^{N_K}.$$

Such tensor products span $\mathbf{C}^N$.

The tensor product is not commutative. This noncommutativity is mainly responsible for the richness of the tensor product algebra, and, as we will see, naturally leads to a distinguished class of permutations, the stride permutations. The first important consequence of this lack of commutativity can be seen in the relationship between tensor product and *matrix direct sum*

$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix},$$

where the 0's represent blocks of zeros of size necessary to form the matrix. We have the

Left Distributive Law

$$(A \oplus B) \otimes C = (A \otimes C) \oplus (B \otimes C),$$

but not the *right distributive law.*

The action of matrix direct sum can be implemented by parallel actions of the direct sum factors. In particular,

$$I_M \otimes B = \oplus_1^M B, \qquad B \text{ an } L \times L \text{ matrix},$$

can be viewed as a *parallel operation.* Its action on a vector $\mathbf{x}$ can be performed by computing the independent action of B on each of the M

consecutive segments of $\mathbf{x}$ of size L. Ideally, the action can be computed in parallel by M independent processors each capable of computing the action of B. The appearance of $I_M \otimes B$ in a matrix factorization representing an algorithm attaches a *parallelism* M and *granularity* B to the corresponding stage of the calculation. One goal of algorithm design is to match such algorithmic parameters to target machine parameters.

By the multiplication theorem,

$$A \otimes B = (A \otimes I_L)(I_M \otimes B) = (I_M \otimes B)(A \otimes I_L), \tag{1.6}$$

where A is an $M \times M$ matrix and B is an $L \times L$ matrix. Set $N = LM$. The factor

$$A \otimes I_L$$

cannot be directly implemented in parallel in the sense described above. In fact, it is natural to view $A \otimes I_L$ as the *vector operation* defined by A on vectors of size L. By definition,

$$A \otimes I_L = \begin{bmatrix} a_{0,0}I_L & a_{0,1}I_L & \cdots & a_{0,M-1}I_L \\ a_{1,0}I_L & a_{1,1}I_L & \cdots & a_{1,M-1}I_L \\ \vdots & & & \\ a_{M-1,0}I_L & a_{M-1,1}I_L & \cdots & a_{M-1,M-1}I_L \end{bmatrix}.$$

To compute the action of $A \otimes I_L$ on a vector $\mathbf{x} \in \mathbf{C}^N$, we segment $\mathbf{x}$ into M consecutive segments of size L,

$$\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{M-1},$$

and compute the vector operations

$$a_{m,0}\mathbf{x}_0 + \cdots + a_{m,M-1}\mathbf{x}_{M-1}, \qquad 0 \le m < M.$$

As before, the term vector operation and the reference to vector size establish algorithmic parameters and do not refer to implementation on a specific machine.

The number of factors of a tensor product is called the *dimension of the tensor product* (as distinguished from the dimension of the space on which the tensor product acts). Thus $A \otimes B$ is a 2-dimensional tensor product. Formula (1.6) decomposes $A \otimes B$ into the product of a parallel operation and a vector operation. *Mixed type tensor products* will also occur. For example,

$$I_M \otimes X \otimes I_L$$

can be viewed as M parallel vector operations $X \otimes I_L$ or as the vector operation defined by $I_M \otimes X$ on vectors of size L.

In the next section, we begin the study of the tensor product algebra, paying special attention to the properties of a special class of permutations,

the stride permutations. This algebra is based on a few simple rules that take the form of tensor product identities. Some have been described in this section, but the most far-reaching, the commutation theorem, requires that we define stride permutations and describe their properties. These tensor product identities generate a wide range of other identities and factorizations, especially of the stride permutations themselves, which, when attached to implementation problems, serve as major tools for modifying tensor product factorizations and hence modifying algorithms for computing on specific machines.

1.3 Stride Permutations

Divide-and-conquer algorithms are often based on a sequence of stages beginning with a mapping of a linear array (vector) onto a multidimensional array, followed by computations on the multidimensional array, and completed by a mapping of the multidimensional array back onto a linear array. The action of the tensor product $A \otimes B$ on a vector can be placed in this framework. Stride permutations are the building blocks for readdressing stages, and implement the global and local transpositions occurring through the computation.

Consider a vector $\mathbf{x} \in \mathbf{C}^N$, where $N = LM$. Segment the vector $\mathbf{x}$ into M vectors of size L

$$\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{M-1},$$

where $\mathbf{x}_m$ is formed from the L components of $\mathbf{x}$ beginning with x_{mL}. Form the $L \times M$ matrix

$$\mathrm{Mat}_{L\times M}(\mathbf{x}) = [\mathbf{x}_0\ \mathbf{x}_1\ \cdots\ \mathbf{x}_{M-1}].$$

The l, m coefficient of $\mathrm{Mat}_{L\times M}(\mathbf{x})$ is given by

$$x_{l,m} = x_{l+mL}, \qquad 0 \le l < L,\ \ 0 \le m < M.$$

Conversely, if X is an $L \times M$ matrix, the *lexicographic ordering* on X is the ordering on the elements of X defined by running in order down the columns of X. Denote by $\mathbf{x} \in \mathbf{C}^N$ the vector determined by the lexicographic ordering on X. Clearly, the vector corresponding to $\mathrm{Mat}_{L\times M}(\mathbf{x})$ is $\mathbf{x}$ itself. Denote the transpose of $\mathrm{Mat}_{L\times M}(\mathbf{x})$ by $\mathrm{Mat}^t_{L\times M}(\mathbf{x})$.

Definition Let $N = ML$. The *stride permutation* $P(N, L)$ is the $N \times N$ permutation matrix defined by

$$\mathbf{y} = P(N, L)\mathbf{x}, \quad \mathbf{x} \in \mathbf{C}^N,$$

where $\mathbf{y}$ is the vector corresponding to the matrix

$$\mathrm{Mat}^t_{L\times M}(\mathbf{x}).$$

On the level of 2-dimensional arrays, stride permutation is equivalent to matrix transpose. Since matrix transpose reverses the coefficient index order, we have

$$y_{m+lM} = x_{l+mL}, \quad 0 \le l < L, \ 0 \le m < M.$$

$P(N, L)$ reorders the coordinates at stride L into L consecutive segments of M elements.

Stride permutations are closely related to the concept of tensor product. Suppose $\mathbf{a} \in \mathbf{C}^M$ and $\mathbf{b} \in \mathbf{C}^L$. Set $\mathbf{c} = \mathbf{a} \otimes \mathbf{b}$. The vector $\mathbf{c}$ is naturally segmented into M consecutive subvectors of size L. In fact,

$$\mathrm{Mat}_{L\times M}(\mathbf{c}) = [a_0\mathbf{b} \ \ a_1\mathbf{b} \cdots \ a_{M-1}\mathbf{b}].$$

The transpose

$$\mathrm{Mat}^t_{L\times M}(\mathbf{c}) = [b_0\mathbf{a} \ \ b_1\mathbf{a} \cdots \ b_{L-1}\mathbf{a}] = \mathrm{Mat}_{M\times L}(\mathbf{b} \otimes \mathbf{a}),$$

proving

$$P(N, L)(\mathbf{a}^M \otimes \mathbf{b}^L) = \mathbf{b}^L \otimes \mathbf{a}^M. \tag{1.7}$$

More generally, we have the following fundamental theorem.

Theorem 1.2 Commutation Theorem *If A is an $M \times R$ matrix and B is an $L \times S$ matrix, then setting $N = LM$ and $T = RS$,*

$$P(N, L)(A \otimes B) = (B \otimes A)P(T, S).$$

Proof Compute the action of both sides on vectors of the form $\mathbf{a}^R \otimes \mathbf{b}^S$; then apply formulas (1.5) and (1.7).

Several special cases are worth noting. Suppose $M = R$ and $S = L$. Then $N = T$, and we have

$$P(N, L)(A \otimes B)P(N, L)^{-1} = B \otimes A.$$

Application of this leads to

$$I_L \otimes A = P(N, L)(A \otimes I_L)P(N, M),$$

$$B \otimes I_M = P(N, L)(I_M \otimes B)P(N, M).$$

We have used

$$P(N, L)^{-1} = P(N, M), \quad N = LM,$$

which immediately follows from (1.7). Stride permutations interchange parallel and vector operations. The readdressing prescribed by $P(N, M)$ on input and $P(N, L)$ on output turns the vector operation $A \otimes I_L$ into the

parallel operation $I_L \otimes A$ and the parallel operation $I_M \otimes B$ into the vector operation $B \otimes I_M$. Continuing in this way, we can write

$$\begin{aligned} A \otimes B &= (A \otimes I_L)(I_M \otimes B) \\ &= P(N,M)(I_L \otimes A)P(N,L)(I_M \otimes B), \end{aligned}$$

which can be used to compute the action of $A \otimes B$ as a sequence of two parallel operations. We can also write

$$A \otimes B = (A \otimes I_L)P(N,M)(B \otimes I_M)P(N,L),$$

which can be used to compute the action of $A \otimes B$ as a sequence of two vector operations. The stride permutations intervene between computational stages, providing a mathematical language for describing the readdressing.

We can now justify the definition of $A \otimes B$ as a two-dimensional tensor product by expressing the action of $A \otimes B$ in terms of the mapping $\mathrm{Mat}_{L \times M}$. For $\mathbf{a} \in \mathbf{C}^M$, $\mathbf{b} \in \mathbf{C}^L$,

$$\mathrm{Mat}_{L \times M}(\mathbf{a} \otimes B\mathbf{b}) = B\ \mathrm{Mat}_{L \times M}(\mathbf{a} \otimes \mathbf{b}), \tag{1.8}$$

$$\mathrm{Mat}_{M \times L}(\mathbf{b} \otimes A\mathbf{a}) = A\ \mathrm{Mat}_{M \times L}(\mathbf{b} \otimes \mathbf{a}). \tag{1.9}$$

Since

$$A\mathbf{a} \otimes \mathbf{b} = P(N,M)(\mathbf{b} \otimes A\mathbf{a}),$$

we use the definition of $P(N,M)$ to write

$$\mathrm{Mat}_{L \times M}(A\mathbf{a} \otimes \mathbf{b}) = (\mathrm{Mat}_{M \times L}(\mathbf{b} \otimes A\mathbf{a}))^t,$$

which by (1.9) becomes

$$\mathrm{Mat}_{L \times M}(A\mathbf{a} \otimes \mathbf{b}) = \mathrm{Mat}_{L \times M}(\mathbf{a} \otimes \mathbf{b})A^t.$$

Formula (1.8) implies

$$\mathrm{Mat}_{L \times M}(A\mathbf{a} \otimes B\mathbf{b}) = B\ \mathrm{Mat}_{L \times M}(\mathbf{a} \otimes \mathbf{b})A^t.$$

More generally, we have the following result.

Theorem 1.3 *For $\mathbf{x} \in \mathbf{C}^N$, A an $M \times M$ matrix, and B an $L \times L$ matrix,*

$$\mathrm{Mat}_{L \times M}((A \otimes B)\mathbf{x}) = B\ \mathrm{Mat}_{L \times M}(\mathbf{x})A^t.$$

1.4 Algebra of Stride Permutations

The tensor product identities of the preceding section form the basis of a rich algebra of stride permutations. The algebra can be used to design general parallel or vector algorithms characterized by computational stages

that are all parallel operations or all vector operations. This will be reviewed in the next section. Such algorithms can be self-sorting, i.e., requiring no readdressing at input or output, or can have uniform readdressing between computational stages. Algorithmic parameters such as degree of parallelism, vector length, and granularity are clearly displayed at each stage of the algorithm. Usually, an algorithm must be considered in greater detail to take advantage of a specific architecture. Local readdressing and commuting of multidimensional tensor products can result in factors having both parallel and vector operation characteristics. Modification of algorithms on this microscopic level is greatly simplified by the tensor product identities. In general, global and local commuting of multidimensional tensor products is accompanied by permutations that admit factorizations in terms of tensor products of the form $I_M \otimes P \otimes I_L$, where P is some stride permutation. These factorizations offer a variety of index-readdressing procedures that can be tested on the algorithmic level rather than in code for their suitability in application.

A permutation of the form $I_M \otimes P \otimes I_L$ acts on a multidimensional tensor product by

$$(I_M \otimes P(T,S) \otimes I_L)(\mathbf{a}^M \otimes \mathbf{x}^R \otimes \mathbf{y}^S \otimes \mathbf{b}^L) = \mathbf{a}^M \otimes \mathbf{y}^S \otimes \mathbf{x}^R \otimes \mathbf{b}^L,$$

where $T = RS$. Products of such tensor product of stride permutations frequently occur in commuting tensor products and can be interpreted as permutations of a tensor product basis rather than arbitrary permutations of the full vector.

Several results will be used throughout this and the following chapter.

Theorem 1.4 Multiplication Theorem of Stride Permutations *If* $N = KLM$, *then*

$$P(N, LM) = P(N, L)P(N, M).$$

In particular, if $N = p^R$ *for an integer* $p \geq 2$, *then the set*

$$\{P(N, p^r) : \ 0 \leq r < R\}$$

is a cyclic group of order R *generated by* $P(N, p)$.

Tensor products of stride permutations of the form

$$I_M \otimes P, \qquad P \otimes I_L$$

are especially important. $I_M \otimes P$ permutes the elements within the segments of the input vector, and $P \otimes I_L$ permutes the segments themselves. Products of these tensor products can decompose a stride permutation. A useful factorization of this type is given by

$$P(MRS, S) = (P(MS, S) \otimes I_R)(I_M \otimes P(RS, S)).$$

The right-hand side product decomposes the stride by S permutation on MRS points into two addressing permutations. The first strides by S on each of the M segments of the input vector of size RS. The second permutes the segments by striding through the segments by S. The result is easily proved by computing the action of both sides on multidimensional tensor products of the form $\mathbf{x}^M \otimes \mathbf{x}^R \otimes \mathbf{x}^S$, where $\mathbf{x}^M$ is an arbitrary vector in $\mathbf{C}^M$.

1.5 Tensor Product Factorization

Let $N = N_1 N_2 \cdots N_K$. Set $N(k) = N_1 N_2 \cdots N_k$, $1 \leq k \leq K$, and $N(0) = 1$. Let

$$A_{N_k}, \quad 1 \leq k \leq K,$$

denote arbitrary $N_k \times N_k$ matrices. Define the tensor products

$$\begin{aligned} X_1 &= A_{N_1} \otimes I_{N_2 \cdots N_K}, \\ &\vdots \\ X_k &= I_{N(k-1)} \otimes A_{N_k} \otimes I_{N/N(k)}, \\ &\vdots \\ X_K &= I_{N_1 \cdots N_{K-1}} \otimes A_{N_K}. \end{aligned}$$

The first factor X_1 can be viewed as a vector operation acting on vectors of length $N_2 \cdots N_K$. As we go down the list, parallelism is increased, but vector length decreases. Detailed discussion of the implementation of such factors emphasizing the loop structure of code and the implications for implementation on vector or parallel machines can be found in Johnson et al. (1990) [21] and Tolimieri et al. (1989) [39] and from a nontensor product perspective in Swarztrauber (1984) [34].

Theorem 1.5 Fundamental Factorization

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = X_1 X_2 \cdots X_K.$$

The proof uses the multiplication theorem for tensor products, which also implies that the factors commute.

The factor X_k can be modified into the parallel operation

$$Y_k = I_{N/N_k} \otimes A_{N_k}$$

by the intervention of stride permutations in two ways. Set

$$P_k = P(N, N(k)), \tag{1.10}$$

$$Q_k = I_{N(k-1)} \otimes P(N/N(k-1), N_k). \tag{1.11}$$

In particular, $P_1 = Q_1 = P(N, N_1)$ and $P_K = Q_K = I_N$. Parallelization can be achieved by either of the following two tensor product identities.

$$X_k = P_k Y_k P_k^{-1},$$

$$X_k = Q_k Y_k Q_k^{-1}.$$

Placing the first into the fundamental factorization, we have

$$\begin{aligned} A_{N_1} \otimes \cdots \otimes A_{N_K} &= (P_1 Y_1 P_1^{-1}) \cdots (P_K Y_K P_K^{-1}) \\ &= P_1 Y_1 (P_1^{-1} P_2) Y_2 (P_2^{-1} P_3) Y_3 \cdots (P_{K-1}^{-1}) Y_K. \end{aligned}$$

Since

$$P_k^{-1} P_{k+1} = P(N, N_{k+1}),$$

we have the following factorization.

Parallel Tensor Product Factorization I

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = P(N, N_1) Y_1 P(N, N_2) Y_2 \cdots P(N, N_K) Y_K.$$

This factorization can be used to decompose the action of $A_{N_1} \otimes \cdots \otimes A_{N_K}$ into a sequence of K stages. In a typical stage, given by

$$P(N, N_k) Y_k,$$

the input vector is acted on by the parallel action Y_k and then readdressed by the stride permutation $P(N, N_k)$ on output. Since initial input data is not readdressed but final output data is readdressed, we call the parallel tensor product factorization I a *decimation-in-frequency factorization.*

The second identity (1.11) leads to another factorization.

Parallel Tensor Product Factorization II

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = Q_1 Y_1 (Q_1^{-1} Q_2) Y_2 \cdots Q_{K-1}^{-1} Y_K.$$

The readdressing between the parallel operations is now given by the product of tensor products,

$$Q_k^{-1} Q_{k+1}, \qquad 1 \le k < K. \tag{1.12}$$

Coding the action of (1.12) is like coding a stride permutation in the sense that component positions are exchanged. The action of (1.12) on the multidimensional tensor product

$$\mathbf{x}^{N_1} \otimes \cdots \otimes \underbrace{\mathbf{x}^{N_k}} \otimes \mathbf{x}^{N_{k+2}} \otimes \cdots \otimes \mathbf{x}^{N_K} \otimes \underbrace{\mathbf{x}^{N_{k+1}}}$$

exchanges the positions k and $k+1$, resulting in the vector

$$\mathbf{x}^{N_1} \otimes \cdots \otimes \mathbf{x}^{N_{k-1}} \otimes \underbrace{\mathbf{x}^{N_{k+1}}} \otimes \mathbf{x}^{N_{k+2}} \otimes \cdots \otimes \mathbf{x}^{N_K} \otimes \underbrace{\mathbf{x}^{N_k}}.$$

There are two global techniques for modifying factorizations. The first is the *transpose method*, which simply involves taking the matrix transpose on both sides of a tensor product identity and using the rules

$$\begin{aligned}(A \otimes B)^t &= A^t \otimes B^t, \\ P^t &= P^{-1}, \quad P \text{ a permutation matrix.}\end{aligned}$$

For notational simplicity, assume that each of the matrices A_{N_k}, $1 \leq k \leq K$, is symmetric; i.e., $A_{N_k}^t = A_{N_k}$ and $Y_k^t = Y_k$, $1 \leq k \leq K$.

Parallel Tensor Product Factorization III

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = Y_K P(N, N/N_K) \cdots Y_1 P(N, N/N_1).$$

Parallel Tensor Product Factorization IV

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = Y_K Q_{K-1} \cdots Y_2 (Q_2^{-1} Q_1) Y_1 Q_1^{-1}.$$

In III and IV readdressing occurs at input (*decimation-in-time*).

Vector tensor product factorizations result from commuting stride permutations across parallel operations. Set

$$Z_k = A_{N_k} \otimes I_{N/N_k}.$$

Then we have

$$P(N, N_k) Y_k = Z_k P(N, N_k).$$

Placing this identity into parallel factorization I, we have the first vector tensor product factorization.

Vector Tensor Product Factorization I

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = Z_1 P(N, N_1) \cdots Z_K P(N, N_K).$$

The following vector tensor product factorization comes from the second parallel factorization.

Vector Tensor Product Factorization II

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = Z_1 R_1 Z_2 R_2 \cdots Z_K R_K,$$

where the permutation

$$R_k = P(N, N_k) Q_k^{-1} Q_{k+1} P(N, N/N_{k+1})$$

is characterized by the property that it maps the multidimensional tensor product

$$\underbrace{\mathbf{x}^{N_{k+1}}} \otimes \mathbf{x}^{N_1} \otimes \cdots \otimes \mathbf{x}^{N_{k-1}} \otimes \underbrace{\mathbf{x}^{N_k}} \otimes \mathbf{x}^{N_{k+2}} \otimes \cdots \otimes \mathbf{x}^{N_K}$$

onto

$$\underbrace{\mathbf{x}^{N_k}} \otimes \mathbf{x}^{N_1} \otimes \cdots \otimes \mathbf{x}^{N_{k-1}} \otimes \underbrace{\mathbf{x}^{N_{k+1}}} \otimes \mathbf{x}^{N_{k+2}} \otimes \cdots \otimes \mathbf{x}^{N_K}.$$

Transposing the vector factorizations with $A_{N_k}^t = A_{N_k}$ leads to the following.

Vector Tensor Product Factorization III

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = P(N, N/N_K) Z_K \cdots P(N, N/N_1) Z_1.$$

Vector Tensor Product Factorization IV

$$A_{N_1} \otimes \cdots \otimes A_{N_K} = R_K^{-1} Z_K \cdots R_2^{-1} Z_2 R_1^{-1} Z_1.$$

The parallel and vector tensor product factorizations I–IV have a variety of global characteristics, such as parallel operations, vector operations, permutation on input, and permutation on output, and clearly describe algorithmic parameters such as degree of parallelism, vector length, and granularity. Another important algorithmic parameter is stride size, both on the components of a segment and on the segments themselves. For example, if the transform size N has been written as the product of many small factors $N_1, ..., N_K$, then the parallel factorization I requires small strides, while the parallel factorization III requires large strides between computational stages. The concise mathematical description of the readdressing permutations is a major guide to code generation even on serial machines.

Many multidimensional computations can be written as multidimensional tensor products of corresponding 1-dimensional computations. Multidimensional finite Fourier transform and convolution are prime examples. The multidimensional tensor product factorizations decompose these multidimensional computations into sequences of 1-dimensional computations with stride permutations and tensor product of stride permutations detailing the required readdressing between computational stages. This methodology produces code that includes 1-dimensional core routines reflecting the 1-dimensional computations and global routines that rebuild the global computation from these core routines. The criteria for judging the core routines may differ from the criteria for judging the global routines, although the two can never be strictly independent. However, certain machine characteristics can dominate the choice, for example, of a core routine but play a smaller role in the choice of a global routine. The scale of the computation also impacts on the problem and, at least at certain thresholds, will closely tie together the choice of core and global routines.

1.6 Fast Fourier Transform Algorithms I

Fast Fourier transform (FFT) algorithms are derived from two distinct sources: the additive and the multiplicative structure of the data indexing set. A partial list of contributions to each is given at the end of this

chapter. In this section, we will review the tensor product formulation of 1-dimensional additive FFT algorithms. One-dimensional multiplicative DFT algorithms and the Prime Factor FFT algorithm will not be included in this account. A detailed treatment can be found in the authors' first text in this series [39]. As examples of tensor product factorization they share all the implementation advantages described in the previous sections. Stride permutations will again play a major role in describing the readdressing between computational stages including the famous "butterfly" of the additive FFT algorithms.

For many applications, hybrids of additive and multiplicative DFT algorithms provide the best code. This is especially true for large size 1-dimensional and for multidimensional computations, where large data sets must first be partitioned into sufficiently small chunks for a machine's computational units. One FFT may break up the problem while another FFT computes on the pieces. From this point of view, the first FFT prepares the data for computation by the second FFT. Although this is a nonstandard way of looking at such algorithms, it is often useful in a parallel environment and fits into the framework of several recently developed Reduced Transform algorithms for multidimensional finite Fourier transform. These will be discussed in chapter 8.

The *N-point Fourier transform matrix* $F(N)$ is defined by

$$F(N) = [w^{jk}]_{0 \le j,k < N}, \qquad w = e^{2\pi i/N}.$$

Several properties of the N-point Fourier transform matrix are useful in derivations and applications. First, $F(N)$ is a *Vandermonde* matrix defined by the first row (the top row consists of all 1's and will be referred to as the 0th row),

$$1,\ w,\ w^2,\ \ldots, w^{N-1}.$$

The componentwise product of the jth row with the rth row is the $(j+r)$th row, $(j+r)$ taken modulo N. As we will discuss in more detail in chapter 4, the rows of $F(N)$ are the additive characters of the abelian group $\mathbf{Z}/N$. The sum of the components in every row but the 0th is equal to 0. The sum of the components in the 0th row is N. With these properties in mind, it is easy to show that

$$F(N)F(N)^* = NI_N,$$

where * denotes complex conjugation. Since $F(N)^t = F(N)$, we have $F(N)^{-1} = \frac{1}{N}F(N)^*$. Also, $F(N)^2 = NR_N$, where R_N is the time reversal matrix

$$R_N = \begin{bmatrix} 1 & 0 & \cdot & \cdot & 0 & 0 \\ 0 & \cdot & & & 0 & 1 \\ \cdot & & & & \cdot & \\ \cdot & & & \cdot & & \\ \cdot & & \cdot & & & \\ 0 & 1 & 0 & & & 0 \end{bmatrix},$$

and hence $F(N)^4 = N^2 I_N$.

Suppose $N = LM$. The additive FFT can be derived using properties of certain submatrices of $F(N)$. For example, the $N \times L$ matrix formed from the multiple of M columns of $F(N)$ is

$$1_M \otimes F(L),$$

where 1_M is the vector in $\mathbf{C}^M$ all of whose components are equal to one. Similar results hold for the $N \times L$ matrix formed from the

$$m,\ M+m,\ \ldots,(L-1)M+m$$

columns of $F(N)$, except that now the matrix is multiplied by a diagonal matrix. Before describing the first Cooley–Tukey (C–T) FFT algorithm, we will introduce a language for dealing with this diagonal matrix.

Set $w = e^{2\pi i/N}$, and define the $N \times N$ diagonal matrix

$$D(N) = \mathrm{diag}(1,\ w,\ \ldots, w^{N-1}).$$

Suppose $N = LM$. Define the $L \times L$ diagonal matrix

$$D_L(N) = \mathrm{diag}(1,\ w,\ \ldots, w^{L-1}),$$

and observe that

$$D(N) = D(M) \otimes D_L(N).$$

The diagonal matrices $D_L(N)$ will be the building blocks for the *twiddle factors* $T_L(N)$, which play a major role in the C–T. They are defined by

$$T_L(N) = \bigoplus_{m=0}^{M-1} D_L^m(N),$$

where $\oplus$ denotes the direct sum of matrices. The matrix $T_L(N)$ is a block diagonal matrix made up of M diagonal matrix blocks of size $L \times L$. Stride permutations reverse the role of L and M in twiddle factors by the formula

$$T_M(N) = P(N,L) T_L(N) P(N,L)^{-1}.$$

Although a version of an additive FFT was given by I. Good in 1958 [18], the additive FFT described in 1965 by J.W. Cooley and J.W. Tukey [11] initiated an era of intense activity, producing several closely related versions distinguished mainly by data flow. The initial FFT was based on transform size a power of a fixed number, with the power of 2 case dominating most early applications, but several authors, including Singleton (1969) [31], extended the FFT to mixed-radix. Mathematically these variations are related by the same rules that govern the various tensor product factorizations. We begin with the *mixed-radix* extension of what was originally published in 1965.

C–T A1 (Decimation-in-Time)

$$F(N) = (F(M) \otimes I_L)T_L(N)(I_M \otimes F(L))P(N,M).$$

One interpretation of C–T A1 is that $P(N,M)$ forms M vectors of size L on which the second factor $I_M \otimes F(L)$ and the twiddle factor act in parallel. This interpretation is reflected in the block diagonal structure of $I_M \otimes F(L)$ and $T_L(N)$. The vector operation $F(M) \otimes I_L$ acts on vectors of size L.

Stride permutations and transpose can be used to derive other versions.

C–T A2 (Decimation-in-Frequency)

$$F(N) = P(N,L)(I_M \otimes F(L))T_L(N)(F(M) \otimes I_L).$$

This form is due to Gentleman and Sande (1966) [17] and is simply the transpose of C–T A1. The output is now readdressed, and a vector operation initiates the computation.

Stride permutations produce parallel versions and vector versions of the C–T algorithm.

C–T A3 Parallel Form

$$F(N) = P(N,M)(I_L \otimes F(M))P(N,L)T_L(N)(I_M \otimes F(L))P(N,M).$$

C–T A4 Vector Form

$$F(N) = (F(M) \otimes I_L)T_L(N)P(N,M)(F(L) \otimes I_M).$$

1.7 General 1-Dimensional FFT

C–T A1 can be used in an inductive argument to derive extensions to many factors. Suppose $N = N_1 N_2 \cdots N_K$ is a factorization of N. Set $N(0) = 1$ and

$$\begin{aligned} N(k) &= N_1 N_2 \cdots N_k, \quad 1 \le k \le K, \\ N'(k) &= N/N(k) = N_{k+1} \cdots N_K, \quad 0 \le k \le K. \end{aligned}$$

Define

$$X_k = (I_{N(k-1)} \otimes F(N_k) \otimes I_{N'(k)})T_k,$$

where T_k is the diagonal matrix $T_k = I_{N(k-1)} \otimes T_{N'(k)}(N'(k-1))$. An induction applying C–T A1 proves the following.

C–T B1 $N = N_1 \cdots N_K$,

$$F(N) = X_1 \cdots X_K Q^{-1},$$

where $Q = Q(N)$ is the generalized *bit-reversal* uniquely defined by

$$Q(\mathbf{x}^{N_1} \otimes \cdots \otimes \mathbf{x}^{N_K}) = \mathbf{x}^{N_K} \otimes \cdots \otimes \mathbf{x}^{N_1}.$$

Taking the transpose on both sides, we have the next result.

C–T B2 $N = N_1 \cdots N_K$,

$$F(N) = QX_K^t \cdots X_1^t.$$

X_k^t simply reverses the order of the diagonal matrix and the Fourier transform factor in X_k.

The factors X_k, $1 \leq k \leq K$, vary from the vector operation $X_1 = (F(N_1) \otimes I_{N/N_1})T_1$ to the parallel operation $X_K = I_{N(K-1)} \otimes F(N_K)$. The general factor X_k is of mixed type. We can argue exactly as in section 5 to produce several variants of C–T B1 and C–T B2 in which all mixed type factors are replaced by parallel operations or vector operations.

Define

$$Y_k = (I_{N/N_k} \otimes F(N_k))T_k',$$

where $T_k' = P(N, N'(k))T_k P(N, N(k))$. Since

$$X_k = P(N, N(k))Y_k P(N, N'(k))$$

and

$$P(N, N'(k))P(N, N(k+1)) = P(N, N_{k+1}),$$

we have the following.

C–T B3 Parallel Form I $N = N_1 \cdots N_K$,

$$F(N) = P(N, N_1)Y_1 \cdots P(N, N_K)Y_K Q^{-1}.$$

Transposing both sides leads to the next form.

C–T B4 Parallel Form II $N = N_1 \cdots N_K$,

$$F(N) = QY_K^t P(N, N/N_K) \cdots Y_1^t P(N, N/N_1).$$

The computational factors are now all parallel operations. The two parallel forms are distinguished by input or output bit-reversal and the addressing strides required between the computational stages.

Define

$$Z_k = (F(N_k) \otimes I_{N/N_k})T_k'',$$

where $T_k'' = T_{N'(k)}(N'(k-1)) \otimes I_{N(k-1)}$.

C–T B5 Vector Form I $N = N_1 \cdots N_K$,

$$F(N) = Z_1 P(N, N_1) \cdots Z_K P(N, N_K)Q^{-1}.$$

Taking the transpose on both sides, we have

C–T B6 Vector Form II $N = N_1 \cdots N_K$,

$$F(N) = QP(N, N/N_K)Z_K^t \cdots P(N, N/N_1)Z_1^t.$$

In both C–T B5 and C–T B6, the computational stages are vector operations. They differ in two respects. C–T B5 is a decimation-in-time FFT, since bit-reversal is required at input, while C–T B6 is a decimation-in-frequency FFT, since bit-reversal occurs at output. For large, highly composite N, the small strides N_k of C–T B5 are replaced by large strides N/N_k of C–T B6.

In the following, the bit-reversal is distributed throughout the computation, resulting in a class of FFT called *auto-sorting*, since bit-reversal is not required at input or output. Define the *bit-reversal* Q_k by

$$Q_k(\mathbf{x}^{N_1} \otimes \cdots \otimes \mathbf{x}^{N_k}) = \mathbf{x}^{N_k} \otimes \cdots \otimes \mathbf{x}^{N_1},$$

and set $R_k = Q_k \otimes I_{N'(k)}$. Define

$$Z'_k = R_k X_k R_k^{-1},$$

which by the commutation theorem can be written as

$$Z'_k = (F(N_k) \otimes I_{N/N_k}) T'''_k,$$

where $T'''_k = R_k T_k R_k^{-1}$.

C–T B7 Auto-Sorting $N = N_1 \cdots N_K$,

$$F(N) = Z'_1 P'_2 Z'_2 \cdots P'_K Z'_K,$$

where $P'_k = R_{k-1} R_k^{-1}$. Direct computation shows that

$$P'_k = P(N(k), N(k-1)) \otimes I_{N'(k)}.$$

Three additional auto-sorting FFT algorithms can be derived using transpose and the commutation theorem.

In C–T B3 to C–T B6, if $N = r^K$ for an integer r, then except for twiddle factor variation, computational and addressing stages remain constant throughout the factorization. M.C. Pease (1968) [26] derived the power of 2 case of C–T B3, emphasizing this property as being especially important for parallel processing on specialized hardware. The tensor product played an important role in Pease's work. The vector version of the Pease FFT was derived by D.G. Korn and J.J. Lambiotte (1979) [24] for implementation on the STAR 100. The extension of the Korn–Lambiotte FFT to mixed-radix was given by R.C. Agarwal and J.W. Cooley (1986) [1] for implementation on the IBM 3090 vector facility. The last work makes no use of tensor products. In these works, especially the last, great care is taken to optimize cache, minimize data transfers, and avoid memory bank conflict. A summary of many of these issues from a non-tensor-product point of view can be found in Swarztrauber [30–32]. In contrast, many authors, including Drubin [15], Rose (1980) [29], Davio (1981) [13], Henderson and

Searle (1981) [19], Temperton (1988) [36–38], and Van Loan (1992) [40], use the tensor product freely. Related work investigating computer manipulation of tensor products can be found in [2,27] and De Boor (1979) [14]. Work by the authors and their close colleagues can be found in Johnson et al. (1990) [21], Tolimieri et al. (1989) [39], and Rodriguez (1988) [28]. Architecture-specific implementation of the FFT can be found in many places, including [3–8, 12, 16, 22, 23, 25, 35, 37]. In Chu's thesis (1988) [9] the tensor product was used as an implementation tool for the Intel iPSC system. In Cochran et al. (1967) [10], an auto-sorting FFT, attributed to Stockham, was presented. Temperton (1983) [36] extended the Stockham FFT using tensor product ideas to mixed-radix exploring implementation on the Cray 1.

The stride permutations intervening between computational stages in C–T B3 through C–T B6 permit *in-place computation*, which requires no additional temporary memory. H.W. Johnson and C.S. Burrus (1984) [20] designed an FFT that is both auto-sorting and in-place. These results have recently been extended by Temperton (1991) [38], with special emphasis on memory management on the Cray X-MP and Y-MP.

In the following list, we specialize several of the C–T FFT algorithms to the power-of-2 cases.

For arbitrary $N \times N$ matrices $X_1, \ldots, X_k$, set

$$\prod_{l=1}^{k} X_l = X_1 X_2 \cdots X_k.$$

I. $N = 2^k$

$$\begin{aligned} T(2^{l+1}) &= T_{2^l}(2^{l+1}), \\ P(2^{l+1}) &= P(2^{l+1}, 2), \\ Q(2^k)(\mathbf{x}_1^2 \otimes \cdots \otimes \mathbf{x}_k^2) &= \mathbf{x}_k^2 \otimes \cdots \otimes \mathbf{x}_1^2 \quad \text{(bit reversal)}. \end{aligned}$$

Cooley–Tukey

$$\begin{aligned} F(N) &= \left(\prod_{l=1}^{k} (I_{2^{l-1}} \otimes F(2) \otimes I_{2^{k-l}})\, T_l \right) Q^{-1}(N), \\ T_l &= I_{2^{l-1}} \otimes T(2^{k-l+1}). \end{aligned}$$

Pease

$$\begin{aligned} F(N) &= \left(\prod_{l=1}^{k} (P(2^k)(I_{2^{k-1}} \otimes F(2)) T_l') \right) Q^{-1}(N), \\ T_l' &= P(2^k, 2^{k-l}) T_l P(2^k, 2^l). \end{aligned}$$

Korn–Lambiotte

$$\begin{aligned} F(N) &= \left(\prod_{l=1}^{k}\left((F(2)\otimes I_{2^{k-1}})T_l''P(2^k)\right)\right)Q^{-1}(N), \\ T_l'' &= T(2^{k-l+1})\otimes I_{2^{l-1}}. \end{aligned}$$

Auto-Sort

$$\begin{aligned} F(N) &= \prod_{l=1}^{k}\left((P(2^l,2^{l-1})\otimes I_{2^{k-l}})(F(2)\otimes I_{2^{k-1}})T_l'''\right), \\ T_l''' &= R_l T_l R_l^{-1}, \quad R_l = Q_l\otimes I_{2^{k-l}}. \end{aligned}$$

References

[1] Agarwal, R.C. and Cooley, J.W. (1986), "Vectorized Mixed Radix Discrete Fourier Transform Algorithms," March, *IBM Report.*

[2] Andrews, H.C. and Kane, J. (1970), "Kronecker Matrices, Computer Implementation, and Generalized Spectra," *J. Assoc. Comput. Mach.* **17**, 260–268.

[3] Auerbuch, A., Gabber, E., Gordissky, B., and Medan, Y. (1990), "A Parallel FFT on a MIMD Machine," *Parallel Comput.* **15**, 61–74.

[4] Bailey, D.H. (1990), "A High Performance Fast Fourier Transform Algorithm for the Cray-2," *J. Supercomputing* **1**, 43–60.

[5] — (1990), "A High Performance Fast Fourier Transform Algorithm for Vector Supercomputers," *J. Supercomputer Appl.* **2**, 82–87.

[6] Brass, A. and Pawley, G.S. (1986), "Two and Three Dimensional FFTs on Highly Parallel Computers," *Parallel Comput.* **3**, 167–184.

[7] Briggs, W.L., Hart, L., Sweet, R., and O'Gallagher, A. (1987), "Multiprocessor FFT Methods," *SIAM J. Sci. Statist. Comput.* **8**, s27–s42.

[8] Chamberlain, R.M. (1988), "Gray Codes, Fast Fourier Transforms, and Hypercubes," *Parallel Comput.* **6**, 225–233.

[9] Chu, C.Y. (1988), "The Fast Fourier Transform on Hypercube Parallel Computers," Ph.D. dissertation, Cornell University.

[10] Cochran, W.T. et al. (1967), "What is the Fast Fourier Transform?" *IEEE Trans. Audio Electroacoust.* **15**, 45–55.

[11] Cooley, J.W., Tukey, J.W. (1965), "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comp.* **19**, 297–301.

[12] Cvetanovic, Z. (1987), "Performance Analysis of the FFT Algorithm on a Shared-Memory Architecture," *IBM J. Res. Devel.* **31**, 435–451.

[13] Davio, M. (1981), "Kronecker Products and Shuffle Algebra," *IEEE Trans. Comput.* **C-30**(2), 116–125.

[14] De Boor, C. (1979), "Efficient Computer Manipulation of Tensor Products," *ACM Trans. Math. Software* **5**(2), 173–182.

[15] Drubin, M. (1971), "Kronecker Product Factorization of the FFT Matrix," *IEEE Trans. Comput.* **C-20**, 590–593.

[16] Fornberg, B. (1981), "A Vector Implementation of the Fast Fourier Transform," *Math. Comp.* **36**, 189–191.

[17] Gentleman, W.M. and Sande, G. (1966),"Fast Fourier Transform for Fun and Profit," *Proc. AFIPS, Joint Computer Conference* **29**, 563–578.

[18] Good, I.J. (1958), "The Interaction Algorithm and Practical Fourier Analysis," *J. Roy. Stat. Soc. Ser. B* **20**, 361–372.

[19] Henderson, H.V. and Searle, S.R. (1981), "The Vec-Permutation Matrix, The Vec Operator and Kronecker Products: A Review," *Linear and Multilinear Algebra* **9**, 271–288.

[20] Johnson, H.W. and Burrus, C.S. (1984), "An In-Place In-Order Radix-2 FFT," *IEEE ICASSP*, San Diego.

[21] Johnson, J., Johnson, R., Rodriguez, D., and Tolimieri, R. (1990), "A Methodology for Designing, Modifying, and Implementing Fourier Transforms on Various Architectures," *IEEE Trans. Circuits Systems* **9**(4).

[22] Johnson, S.L. and Krawitz, R.L. (1991), "Cooley–Tukey FFT on the Connection Machine", Division of Applied Sciences Report TR-24-91, Harvard University (to appear in *Parallel Comput.*).

[23] Johnson, S.L., Krawitz, R.L., Frye, R., and MacDonald, D. (1989), "A Radix-2 FFT on the Connection Machine," *Proceedings Supercomputer 89*, ACM Press, New York, 809-819.

[24] Korn, D.G. and Lambiotte, J.J. (1979), "Computing the Fast Fourier Transform on a Vector Computer," *Math. Comp.* **33**, 977–992.

[25] Norton, A. and Silberger, A. (1987), "Parallelization and Performance Prediction of the Cooley–Tukey FFT Algorithm for Shared Memory Architectures," *IEEE Trans. Comput.* **C-36**, 581-591.

[26] Pease, M.C. (1968), "An Adaptation of the Fast Fourier Transform for Parallel Processing," *J. ACM* **15**, 252–264.

[27] Pereyra, V. and Scherer, G. (1973), "Efficient Computer Manipulation of Tensor Products with Applications to Multidimensional Approximation," *Math. Comp.* **27**, 595–605.

[28] Rodriguez, D. (1988), "On Tensor Product Formulation of Additive Fast Fourier Transform Algorithms and Their Implementation", Ph.D. dissertation, CUNY.

[29] Rose, D.J. (1980), "Matrix Identities of the Fast Fourier Transform," *Parallel Comput.*

[30] Singleton, R.C. (1967), "On Computing the Fast Fourier Transform," *J. ACM* **10**, 647–654.

[31] — (1969), "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.* **17**, 93–103.

[32] Sorensen, H.V., Katz, C.A., and Burrus, C.S. (1990), "Efficient FFT Algorithms for DSP Processors Using Tensor Product Decomposition," *Proc. ICASSP-90*, Albuquerque, NM.

[33] Swarztrauber, P.N. (1982), "Vectorizing the FFTs," in *Parallel Computations*, G. Rodrique (ed.), Academic Press, New York, 490-501.

[34] — (1984), "FFT algorithms for Vector Computers," *Parallel Comput.* **1** 45-63.

[35] — (1987), "Multiprocessor FFTs," *Parallel Comput.* **5**, 197-210.

[36] Temperton, C. (1983), "Self-Sorting Mixed Radix Fast Fourier Transforms," *Computational Physics* **52**, 1–23.

[37] — (1984), "Fast Fourier Transforms on the Cyber 205," in *High Speed Computation*, J. Kowalik (ed.), Springer-Verlag, Berlin.

[38] — (1991), "Self-Sorting In-Place Fast Fourier Transforms," *SIAM J. SCL Stat. Comp.* **12**, July, 808-823.

[39] Tolimieri, R., An, M., and Lu, C. (1989), *Algorithms for Discrete Fourier Transform and Convolution*, Springer-Verlag, New York.

[40] Van Loan, C. (1992), *Computational Frameworks for the Fast Fourier Transform,* in Frontiers in Appl. Math., SIAM, Philadelphia.

Problems

1. Write code implementing each stage X_l, $1 \leq l \leq k$, of the Gentleman–Sande algorithm.

2. Write code implementing bit-reversal.

3. For N= 8, 16, 32, and 64, describe the twiddle factor T_l in the Pease algorithm.

4. Derive the general form of the twiddle factor in the Pease factorization.

5. From the Pease algorithm, design an algorithm reversing the order of permutations and the twiddle factors.

6. From the Pease algorithm, design an algorithm having bit-reversal at output.

7. Determine the general form of the twiddle factors in the Stockham factorization.

8. Describe the twiddle factors in the mixed radix Agarwal–Cooley FFT algorithm.

9. Describe the twiddle factors in the mixed radix Auto-Sort FFT algorithm.

2

Multidimensional Tensor Product and FFT

2.1 Introduction

Multidimensional signal processing is a child of the computer revolution. Several fields of application such as weather, crystallography, sonar/radar, medical imaging, and seismology require multidimensional data arrays for complete and faithful modeling. The processing of such multidimensional data, one dimension at a time, introduces an error-prone and time-consuming reconstruction stage. Redundancies introduced by time-frequency, time-scale, and space-frequency analysis of speech and vision data naturally lead to multidimensional computations.

The massive-size data sets typically found in multidimensional processing place intensive demands on the handling of the communication aspects of the computation. Memory utilization—transfer of data between main storage and computation units—must be precisely matched to the communication capabilities of the target processor. Communication bandwidth between memory and processing unit and when available between processing units must be fully utilized. *Cache misses* can cause delays that overwhelm those caused in computation. The care given to arithmetic complexity, especially to reducing multiplicative complexity in traditional serial programming efforts is now transferred to communication complexity. Multidimensional computations offer a wider range of possible implementations as compared to 1-dimensional computations due to the greater freedom of movement in the data indexing set. This flexibility has already played a central role in 1-dimensional FFT algorithms where computation takes place on multidimensional arrays built from initial linear arrays.

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

In this chapter, we will derive a class of multidimensional FFT algorithms, called *vector-radix FFT*, introduced by G.E. Rivard (1977) [8] and D.B. Harris et al. (1977) [3]. In 1981, a paper by R. Mersereau and T.C. Speake [6] unified various 2-dimensional Cooley–Tukey FFT algorithms. In the following year a paper by Auslander–Tolimieri–Winograd [1] described a generalized Cooley–Tukey FFT on arbitrary finite abelian groups that included the others as special cases. This work uses extensive number-theoretic terminology since its main goal was to understand the relationship between Hecke quadratic reciprocity and generalized Cooley–Tukey FFT.

In form, these FFT algorithms extend the 1-dimensional C–T to multidimensions. For the most part, the discussion takes place in 2-dimensions, as the general multidimensional case can be handled by the same arguments. The 1-dimensional C–T is based on applying periodization-decimation to 1-dimensional arrays. The vector radix FFT takes advantage of the greater flexibility of movement in the indexing set and applies periodization-decimation to the full multidimensional array. This generates a large class of new algorithms whose computational and addressing stages are best described as multidimensional operations, independent of actual machine data storage and memory transfer that usually take place on 1-dimensional arrays.

In this chapter, we derive the vector-radix FFT by manipulating and interpreting multidimensional tensor products. Tensor products of stride permutations will now be described as actions on multidimensional arrays. In chapter 5, we will present a more abstract approach that exploits the subgroup structure of the indexing set.

2.2 Multidimensional Fourier Transform

The multidimensional finite Fourier transform can be written as a multidimensional tensor product whose factors are 1-dimensional finite Fourier transforms. First we consider the 2-dimensional case.

An $L \times M$ *2-dimensional array* A is any complex-valued function

$$A(l,m), \quad 0 \le l < L, \ 0 \le m < M,$$

defined on the Cartesian product $\mathbf{Z}/L \times \mathbf{Z}/M$. We will often view A as a complex $L \times M$ matrix. The $L \times M$ *2-dimensional Fourier transform* of A, denoted by

$$F(L,M)A,$$

is the $L \times M$ 2-dimensional array B defined by

$$B(r,s) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} A(l,m) v^{rl} w^{sm}, \quad 0 \le r < L, \ 0 \le s < M, \tag{2.1}$$

where $v = e^{2\pi i/L}$ and $w = e^{2\pi i/M}$. Definition (2.1) can be written compactly in matrix form as

$$B = F(L)AF(M). \tag{2.2}$$

The *row–column method* computes B by a sequence of 1-dimensional finite Fourier transforms of the columns of A followed by a sequence of 1-dimensional finite Fourier transforms of the resulting rows. This amounts to computing the matrix product (2.2) in the following two stages:

1. $C = F(L)A$.
2. $B = CF(M)$.

The first computes the L-point 1-dimensional Fourier transform on each of the M columns of A, while the second computes the M-point 1-dimensional Fourier transform on each of the L rows of C. Under the assumption that computations are made on columns, two transpositions of the full data array are required: one between the computational stages and one at output (if the output is needed in proper order). The row–column method also refers to the computations

1. $D = AF(M)$,
2. $B = F(L)D$,

which first computes on the rows of A and then computes on the columns of D. Two full transpositions are still required, with the output transposition replaced by an input transposition.

In general, we distinguish two levels of algorithm design. The global level decomposes the desired computation into a set of smaller computations and controls the addressing strategy at input, output, and between these smaller computations. In the row–column method, the array transpositions make up the global level. The local level implements the set of smaller computations. FFT algorithms computing the 1-dimensional finite Fourier transforms of the row–column method can be viewed as part of the local level. There is no hard and fast rule that can be used to distinguish between levels. For example, code for these 1-dimensional finite Fourier transforms can be built from a global strategy that decomposes these computations into even smaller computations. The initial global decomposition will often differ from the global decomposition used for the local computations. These decisions are based on the global and local communication capabilities of the target processor. Similar considerations govern the choice of local computations. In particular, an important part of algorithm design in the row–column method is the code implementing the 1-dimensional finite Fourier transforms.

Full array transpositions characterize the row–column method. For large-size problems, carrying out these transpositions by transferring between

main and local memory can be the most time-consuming part of the computation. The feasibility of a computation can depend on the skill with which the number of such transfers can be minimized and localized.

Formula (2.2) can be written as a tensor product. The lexicographic ordering on the $L \times M$ array A is the ordering on the elements of A defined by reading in order down the columns of A. Denote by $\mathbf{a}$ the vector in $\mathbf{C}^N$, $N = ML$, determined by the lexicographic ordering on A. Then we can write (2.2) as

$$\mathbf{b} = (F(M) \otimes F(L))\mathbf{a}.$$

The results of chapter 1 can now be applied. Any 1-dimensional FFT algorithm, including any 1-dimensional multiplicative DFT algorithm in the form of a matrix factorization, can be used in place of $F(M)$ and $F(L)$. In fact, depending on the application, $F(M)$ can be replaced by one such factorization and $F(L)$ by another. Tensor product identities, especially the multiplication theorem of tensor products, can be used to generate a wide range of algorithms. The following sections will explore some of these possibilities in greater detail. Special emphasis will be given to the tensor products of stride permutations that result from this procedure.

Consider the K-dimensional array

$$A(n_1, \ldots, n_K), \quad 0 \le n_1 < N_1, \ldots, 0 \le n_K < N_K.$$

The *lexicographic ordering* on the $N_1 \times \cdots \times N_K$ array A is the ordering defined on the elements of A by having N_1 as the fastest running parameter followed in order by N_2, ..., N_K. Denote by $\mathbf{a}$ the vector in $\mathbf{C}^N$, $N = N_1 N_2 \cdots N_K$, determined by the lexicographic ordering on A. The $N_1 \times \cdots \times N_K$ *K-dimensional Fourier transform* of A, denoted by

$$F(\mathbf{N})A, \quad \mathbf{N} = (N_1, \ldots, N_K),$$

is the $N_1 \times \cdots \times N_K$ K-dimensional array B defined by

$$B(l_1, \ldots, l_K) = \sum_{n_K=0}^{N_K-1} \cdots \sum_{n_1=0}^{N_1-1} A(n_1, \ldots, n_K) v_1^{l_1 n_1} \cdots v_K^{l_K n_K},$$

where $0 \le l_k < N_k$ and $v_k = e^{2\pi i/N_k}$. Tracing through the sequence of summations, we can rewrite this as the multidimensional tensor product

$$\mathbf{b} = (F(N_K) \otimes \cdots \otimes F(N_1))\mathbf{a}.$$

2.3 2-Dimensional Operations

Consider an $L \times M$ 2-dimensional array A and the vector $\mathbf{a}$ in $\mathbf{C}^N$, $N = LM$ corresponding to A. The action of any $N \times N$ matrix Z on $\mathbf{a}$ induces an

action on A. Define ZA to be the $L \times M$ matrix corresponding to the vector $Z\mathbf{a} \in \mathbf{C}^N$. We call this operation on A a *2-dimensional operation.* In particular, an $N \times N$ permutation matrix P induces a permutation of A called a *2-dimensional permutation.* We can view this permutation as a reordering of the elements of A. In general, if $Z = X \otimes Y$ where X is an $M \times M$ matrix and Y is an $L \times L$ matrix, then

$$ZA = YAX^t.$$

Set $L = L_1L_2$, $M = M_1M_2$ and

$$P_1 = I_{M_2} \otimes P(M_1L_2, L_2) \otimes I_{L_1}.$$

We will see that P_1 can be described as follows. Partition A into L_2M_2 contiguous subblocks of size $L_1 \times M_1$ and reorder the elements of A by placing the lexicographic ordering on the matrix of subblocks and on the elements of each of the subblocks. P_1 generalizes to two dimensions the process of segmenting a vector into contiguous subvectors. We will show that the matrix

$$X_1 = I_{M_2} \otimes F(M_1) \otimes I_{L_2} \otimes F(L_1)$$

acts on A by computing in parallel the $L_1 \times M_1$ 2-dimensional Fourier transform $F(L_1, M_1)$ on each of the L_2M_2 subblocks formed by P_1, placing the results back in place. We will call matrices of this form *parallel 2-dimensional operations.* Since

$$X_1 = P_1^{-1}(I_{M_2L_2} \otimes F(M_1) \otimes F(L_1))P_1,$$

the permutation P_1 changes the parallel 1-dimensional operation $I_{M_2L_2} \otimes F(M_1) \otimes F(L_1)$ into the parallel 2-dimensional operation X_1.

The matrix

$$X_2 = F(M_2) \otimes I_{M_1} \otimes F(L_2) \otimes I_{L_1}$$

acts on A by computing the $L_2 \times M_2$ 2-dimensional Fourier transform $F(L_2, M_2)$ as a vector operation on the L_2M_2 contiguous subblocks of size $L_1 \times M_1$ formed by P_1, placing the results back in place. We will call matrices of this form *vector 2-dimensional operations.* Since

$$X_2 = P_1^{-1}(F(M_2) \otimes F(L_2) \otimes I_{M_1L_1})P_1,$$

the permutation P_1 changes the vector 1-dimensional operation $F(M_2) \otimes F(L_2) \otimes I_{M_1L_1}$ into the vector 2-dimensional operation X_2.

The permutation of A

$$P = P(M, M_2) \otimes P(L, L_2)$$

is called a *2-dimensional stride permutation* since it describes the read-dressing that permits interchange between parallel and vector 2-dimensional operations. The action of P on A is given by the matrix product

$$P(L, L_2)AP(M, M_1).$$

The matrix product

$$P(L, L_2)A$$

reorders the rows of A by stride L_2, while the matrix product

$$AP(M, M_1)$$

reorders the columns of A by stride M_2. In particular, we can write in block matrix form

$$P(L, L_2)AP(M, M_1) = [A_{r,s}]_{0 \le r < L_2,\ 0 \le s < M_2},$$

where $A_{r,s}$ is the $L_1 \times M_1$ matrix

$$A_{r,s} = [A(r + l_1 L_2, s + m_1 M_2)]_{0 \le l_1 < L_1, 0 \le m_1 < M_1}.$$

The case

$$P(L, 2)AP(M, M/2), \quad 2 \mid L,\ 2 \mid M,$$

is called the *2-dimensional inverse perfect shuffle*, which we can write as

$$\begin{bmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{bmatrix},$$

where $A_{0,0}$ consists of all even/even indices, $A_{0,1}$ of all even/odd indices, $A_{1,0}$ of all odd/even indices, and $A_{1,1}$ of all odd/odd indices.

The matrix

$$P(L, L/2)AP(M, 2)$$

is called the *2-dimensional perfect shuffle*. Its action on an 8×8 array results in the 8×8 array

$$\begin{bmatrix} 0,0 & 0,4 & 0,1 & 0,5 & 0,2 & 0,6 & 0,3 & 0,7 \\ 4,0 & 4,4 & 4,1 & 4,5 & 4,2 & 4,6 & 4,3 & 4,7 \\ 1,0 & 1,4 & 1,1 & 1,5 & 1,2 & 1,6 & 1,3 & 1,7 \\ 5,0 & 5,4 & 5,1 & 5,5 & 5,2 & 5,6 & 5,3 & 5,7 \\ 2,0 & 2,4 & 2,1 & 2,5 & 2,2 & 2,6 & 2,3 & 2,7 \\ 6,0 & 6,4 & 6,1 & 6,5 & 6,2 & 6,6 & 6,3 & 6,7 \\ 3,0 & 3,4 & 3,1 & 3,5 & 3,2 & 3,6 & 3,3 & 3,7 \\ 7,0 & 7,4 & 7,1 & 7,5 & 7,2 & 7,6 & 7,3 & 7,7 \end{bmatrix}.$$

The case

$$P(L, 4)AP(M, M/4), \quad 4 \mid L,\ 4 \mid M,$$

decomposes A into 16 subblocks. The subblock $A_{0,0}$ corresponds to indices both of which are congruent to 0 mod 4. The remaining blocks are distinguished by the values of the indices mod 4. The subblock $A_{r,s}$, $0 \leq r < 4$, $0 \leq s < 4$, consists of all those indices the first of which is congruent to r mod 4, the second of which is congruent to s mod 4. Since $P(L,4) = P(L,2)^2$, $P(M,M/4) = P(M,M/2)^2$, we can view this decomposition in 16 subblocks as a two-stage process where we first decompose into 4 subblocks and then decompose each of these 4 subblocks into 4 subblocks.

Consider

$$P_1 = I_{M_2} \otimes P(M_1L_2, L_2) \otimes I_{L_1}$$

as a 2-dimensional permutation of A. Partition A into M_2 contiguous submatrices of size $L \times M_1$,

$$A_0, \ldots, A_{M_2-1}. \tag{2.3}$$

A_0 is the submatrix formed from the first M_1 columns of A. The vector $\mathbf{a}_0$ corresponding to the lexicographic ordering of A_0 is the same as the vector formed from the first LM_1 components of the vector $\mathbf{a}$. Continuing in this way, we see that partition (2.3) corresponds to the partition of $\mathbf{a}$ into M_2 contiguous segments of size LM_1.

P_1 acts on $\mathbf{a}$ by acting in parallel on each of these M_2 contiguous segments of $\mathbf{a}$. By the above discussion, P_1 acts on A by acting in parallel on each of the submatrices in (2.3).

The action $P(M_1L_2, L_2) \otimes I_{L_1}$ on $\mathbf{a}_0$ can be described as follows. Partition $\mathbf{a}_0$ into M_1L_2 contiguous segments of size L_1. The vector operation $P(M_1L_2, L_2) \otimes I_{L_1}$ acts on $\mathbf{a}_0$ by permuting these M_1L_2 segments by stride L_2. The corresponding action on A_0 is given as follows. Partition A_0 into L_2 contiguous subblocks of size $L_1 \times M_1$,

$$A_{0,0}, A_{1,0}, \ldots, A_{L_2-1,0},$$

and place the lexicographic ordering on the elements of each of these $L_1 \times M_1$ submatrices. The vector formed by first passing through the elements in $A_{0,0}$ and then through the elements in $A_{1,0}, \ldots, A_{L_2-1,0}$ in order equals $(P(M_1L_2, L_2) \otimes I_{L_1})\mathbf{a}_0$. Continuing in this way, the action of P_1 on A can be described by the following sequence of steps.

- Partition the $L \times M$ matrix A into M_2L_2 contiguous submatrices of size $L_1 \times M_1$.
- Order the elements of A by placing the lexicographic ordering on the collection of submatrices and on the elements of each of the submatrices.

Following Shamash (1989) [10], we call P_1 a *2-dimensional partitioning permutation.* The general multidimensional definition of a partitioning permutation can also be found in Shamash (1989).

Consider

$$X_1 = I_{M_2} \otimes F(M_1) \otimes I_{L_2} \otimes F(L_1) = P_1^{-1}(I_{M_2L_2} \otimes F(M_1) \otimes F(L_1))P_1$$

as a 2-dimensional operation on A. As a 1-dimensional permutation, P_1 permutes the elements of $\mathbf{a}$ by reordering the components according to the 2-dimensional partitioning permutation P_1. Then the factor $I_{M_2L_2} \otimes F(M_1) \otimes F(L_1)$ acts in parallel on the M_2L_2 contiguous segments of $P_1\mathbf{a}$. Since these 1-dimensional segments correspond to the M_2L_2 contiguous subblocks of A, this is equivalent to computing in parallel the 2-dimensional operation $F(M_1) \otimes F(L_1)$ on each of these subblocks of A. The inverse permutation P_1^{-1} reorders the resulting $L \times M$ array lexicographically. Thus, X_1 is a parallel 2-dimensional operation as described in the beginning of the section.

2.4 2-Dimensional Cooley–Tukey FFT

The 1-dimensional factorizations C–T A1 to C–T A4 of chapter 1 admit 2-dimensional analogues. Suppose $M = M_1M_2$ and $L = L_1L_2$. Set

$$\begin{aligned} X_1 &= I_{M_2} \otimes F(M_1) \otimes I_{L_2} \otimes F(L_1), \\ X_2 &= F(M_2) \otimes I_{M_1} \otimes F(L_2) \otimes I_{L_1}, \\ P &= P(M, M_2) \otimes P(L, L_2), \\ Y_1 &= F(M_1) \otimes I_{M_2} \otimes F(L_1) \otimes I_{L_2}, \\ Y_2 &= I_{M_1} \otimes F(M_2) \otimes I_{L_1} \otimes F(L_2). \end{aligned}$$

The 2-dimensional analogue of C–T A1 can be derived by replacing $F(M)$ and $F(L)$ in $F(M) \otimes F(L)$ by their C–T A1 factorizations and reorganizing by the tensor product multiplication theorem.

C–T C1 (*Decimation-in-Time*)

$$F(M) \otimes F(L) = X_2TX_1P,$$

where $T = T_{M_1}(M) \otimes T_{L_1}(L)$, X_1 is a parallel 2-dimensional operation, and X_2 is a vector 2-dimensional operation. T is a diagonal matrix called a *2-dimensional twiddle factor.*

Transposing both sides of C–T C1, we have

C–T C2 (*Decimation-in-Frequency*)

$$F(M) \otimes F(L) = P^{-1}X_1TX_2.$$

To derive the corresponding parallel and vector forms, we observe that the 2-dimensional stride permutation P interchanges parallel and vector 2-dimensional operations by the identity

$$P^{-1}X_2P = Y_2.$$

Placing this identity into C–T C1, we have

C–T C3 Parallel Form

$$F(M) \otimes F(L) = PY_2P^{-1}TX_1P.$$

Both computational factors X_1 and Y_2 are parallel 2-dimensional operations. In the same way, we can write

C–T C4 Vector Form

$$F(M) \otimes F(L) = X_2TPY_1.$$

X_2 and Y_1 are vector 2-dimensional operations.

Consider now the multifactor case. Suppose $M = M_1 \cdots M_K$ and $L = L_1 \cdots L_K$. Set

$$\begin{aligned} M(k) &= M_1 \cdots M_k, \quad M'(k) = M/M(k), \\ L(k) &= L_1 \cdots L_k, \quad L'(k) = L/L(k). \end{aligned}$$

Set

$$\begin{aligned} X_k &= (I_{M(k-1)} \otimes F(M_k) \otimes I_{M'(k)}) \otimes (I_{L(k-1)} \otimes F(L_k) \otimes I_{L'(k)}), \\ Q(M,L) &= Q(M) \otimes Q(L), \\ T_k &= (I_{M(k-1)} \otimes T_{M'(k)}M'(k-1)) \otimes (I_{L(k-1)} \otimes T_{L'(k)}L'(k-1)). \end{aligned}$$

We call $Q(M,L)$ the $M \times L$ *2-dimensional bit-reversal.* By C–T B1, we have

C–T D1

$$F(M) \otimes F(L) = X_1T_1 \cdots X_KT_KQ(M,L)^{-1}.$$

The factors $X_1, \ldots, X_K$ are mixed-type 2-dimensional operations having varying parallelism and vectorization. Define

$$\begin{aligned} P_k &= P(M,M_k) \otimes P(L,L_k), \\ Y_k &= I_{M/M_k} \otimes F(M_k) \otimes I_{L/L_k} \otimes F(L_k), \\ T'_k &= (P(M,M'(k)) \otimes P(L,L'(k)))\, T_k\, (P(M,M(k)) \otimes P(L,L(k))). \end{aligned}$$

By C–T B3,

C–T D3 Parallel Form

$$F(M) \otimes F(L) = P_1Y_1T'_1 \cdots P_KY_KT'_KQ(M,L)^{-1}.$$

The computational factors $Y_1, \ldots, Y_K$ are parallel 2-dimensional operations. The 2-dimensional stride permutations $P_1, \ldots, P_K$ describe the addressing between computational stages.

The case $M = L = r^K$ is especially important. We have

$$F(r^K) \otimes F(r^K) = PYT'_1 \cdots PYT'_K Q(M, L)^{-1},$$

where

$$\begin{aligned} Q &= P(r^K, r) \otimes P(r^K, r), \\ Y &= I_{r^{K-1}} \otimes F(r) \otimes I_{r^{K-1}} \otimes F(r). \end{aligned}$$

This is the direct 2-dimensional analogue of the Pease FFT. Readdressing and lower-order Fourier transform factors remain constant through all stages of the computation. The twiddle factors vary.

The 2-dimensional analogue of the Korn–Lambiotte FFT can be derived from C–T B5. Set

$$\begin{aligned} Z_k &= F(M_k) \otimes I_{M/M_k} \otimes F(L_k) \otimes I_{L/L_k}. \\ T''_k &= P_k T'_k P_k^{-1}. \end{aligned}$$

C–T D5 Vector Form

$$F(M) \otimes F(L) = Z_1 T''_1 P_1 \cdots Z_K T''_K P_K Q(M, L)^{-1}.$$

The computational factors $Z_1, \ldots, Z_K$ are vector 2-dimensional operations. Set

$$P'_k = P(M(k), M(k-1)) \otimes I_{M'(k)} \otimes P(L(k), L(k-1)) \otimes I_{L'(k)}.$$

From C–T B7 follows the 2-dimensional analogue of the auto-sort FFT.

C–T D7 Auto–Sorting

$$F(M) \otimes F(L) = Z_1 T'''_1 (P'_2 Z_2 T'''_2) \cdots (P'_K Z_K T'''_K),$$

where T'''_1 is a 2-dimensional twiddle factor. As in the 1-dimensional case, the input bit-reversal has been distributed throughout the computation. However, even in the case $M = L = 2^K$, the addressing varies throughout the computation, having different parallelism and vectorization.

A large number of multiplicative DFT algorithms can be derived by the same methods. In this case, we nest 1-dimensional multiplicative DFTs in multidimensional tensor product, and reorganize by the tensor product multiplication theorem. Many results of this type can be found in Blahut (1985) [2]. The paper of H.W. Johnson and C.S. Burrus (1983) [5] explores the full range of possibilities and establishes computer-driven procedures for searching through this range and choosing a multiplicative DFT factorization appropriate for a specified target architecture.

In later chapters, we will derive multiplicative DFTs algorithm based directly on exploiting global algebraic properties of the indexing set.

References

[1] Auslander, L., Tolimieri, R., and Winograd, S. (1982), "Hecke's Theorem in Quadratic Reciprocity, Finite Nilpotent Groups and the Cooley–Tukey Algorithm," *Advances in Mathematics* **43**, 122–172.

[2] Blahut, R.E. (1985), *Fast Algorithms for Digital Signal Processing*, Addison-Wesley.

[3] Harris, D.B., McClelland, D.S.K., Chan, and Schussler, H.W. (1977), "Vector Radix Fast Fourier Transform," *IEEE ICASSP*, 548–551.

[4] Hoyer, E.A. and Berry W.R. (1977), "An Algorithm for the Two-Dimensional Fourier Transform," *IEEE ICASSP*, 552–555.

[5] Johnson, H.W. and Burrus, C.S. (1983), "The Design of Optimal DFT Algorithms Using Dynamic Programming," *IEEE Trans. Acoust. Speech Signal Proc.* ASSP **31**, 378–387.

[6] Mersereau, R. and Speake, T.T. (1981), "A Unified Treatment of Cooley–Tukey Algorithms for the evaluation of the Multidimensional DFT," *IEEE Trans. Acoust. Speech Signal Proc.* ASSP **29**, 1011–1018.

[7] Pease, M.C. (1968), "An Adaptation of the Fast Fourier Transform for Parallel Processing," *J. ACM* **15**, 252–265.

[8] Rivard, G.E. (1977), "Discrete Fast Fourier Transform of Bivariate Functions," *IEEE Trans. Acoust. Speech Signal Proc.* ASSP **25**, 250–252.

[9] Rodriguez, D. (1988), "On Tensor Product Formulation of Additive Fast Fourier Transform Algorithms and Their Implementation," Ph.D. dissertation, CUNY.

[10] Shamash, M. (1989), "VLSI Architectures for Multidimensional Digital Signal Processing," Ph.D. thesis, Technion-Israel Institute of Technology.

[11] Swarztrauber, P.N. (1984), "FFT algorithms for Vector Computers," *Parallel Comput.* **1**, 45–63.

[12] Stone, H.S. (1971), "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers*, 153–161.

[13] Tolimieri, R., An, M., and Lu, C. (1989), "Algorithms for Discrete Fourier Transform and Convolution," Springer-Verlag, New York.

Problems

1. Describe the action of the permutation matrix $P = I_3 \otimes P(8,4) \otimes I_2$ on a vector $\mathbf{a} \in \mathbf{C}^{48}$ and on an 8×6 2-dimensional array A.

2. Describe the action of the permutation matrix $Q = P(9,3) \otimes P(4,2)$ on a vector $\mathbf{a} \in \mathbf{C}^{36}$ and on a 9×4 2-dimensional array A.

3. Describe the action of $I_3 \otimes F(2) \otimes I_4 \otimes F(2)$ as a 1-dimensional and as a 2-dimensional operation.

4. Describe the twiddle factors in C–T D1, D3, and D5.

5. Describe the 8×8 2-dimensional bit-reversal as a 2-dimensional permutation.

6. Describe the 2-dimensional bit-reversal $Q(M) \otimes Q(L)$ on an arbitrary $L \times M$ 2-dimensional array A.

7. Take any 1-dimensional 3-point multiplicative DFT algorithm and design a 3×3 2-dimensional multiplicative DFT algorithm.

8. Derive additional $L \times M$ 2-dimensional C–T algorithms from C–T D1, C–T D3, C–T D5, and C–T D7 by taking the transpose.

9. Describe the permutation P'_k of C–T D7 by its action on tensor products.

10. Describe the radix-4 2-dimensional auto-sorting algorithm.

3

Finite Abelian Groups

3.1 Introduction

The number of elements in a set X will be called the *order* of X and be denoted by $o(X)$.

Suppose throughout that A *is a finite abelian group of order* N. We begin with some preliminary definitions and results.

The *cyclic group* generated by $\mathbf{a} \in A$,

$$gp(\mathbf{a}) = \{n\mathbf{a} : n \in \mathbf{Z}\},$$

has order $o(\mathbf{a})$ dividing N. Moreover, $o(\mathbf{a})\mathbf{a} = 0$ and $n\mathbf{a} = m\mathbf{a}$ if and only if $n \equiv m \bmod o(\mathbf{a})$. A is the *direct sum* of subgroups $A_1, \ldots, A_R$, and we write

$$A = A_1 \oplus \cdots \oplus A_R$$

if every $\mathbf{a} \in A$ can be written uniquely in the form

$$\mathbf{a} = \mathbf{a}^{(1)} + \cdots + \mathbf{a}^{(R)}, \quad \mathbf{a}^{(r)} \in A_r, \quad 1 \le r \le R.$$

A closely related construction, the *group direct product*

$$B_1 \times \cdots \times B_R$$

of groups $B_1, \ldots, B_R$, is the set of R-tuples

$$\mathbf{b} = (\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(R)}), \quad \mathbf{b}^{(r)} \in B_r, \quad 1 \le r \le R,$$

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

under componentwise addition. If $A = A_1 \oplus \cdots \oplus A_R$, then

$$A = A_1 \times \cdots \times A_R.$$

Conversely if A is isomorphic to a group direct product $B_1 \times \cdots \times B_R$, then there exist subgroups $A_1, \ldots, A_R$ such that $A = A_1 \oplus \cdots \oplus A_R$ with A_r isomorphic to B_r, $1 \leq r \leq R$.

The *fundamental theorem of finite abelian groups* will be used extensively in this work. Proofs of the fundamental theorem can be found in most college-level modern algebra texts [3]. For future applications, we will describe two equivalent forms.

- There exist cyclic subgroups $A_1, \ldots, A_R$ of A such that A is the direct sum

  $$A = A_1 \oplus \cdots \oplus A_R.$$

  If the cyclic group A_r has order N_r, then A_r is isomorphic to the group $\mathbf{Z}/N_r$.

- There exist integers $N_1, \ldots, N_R$ such that A is isomorphic to the group direct product

  $$\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R.$$

  Observe that $N = N_1 \cdots N_R$.

Every finite abelian group can serve as the indexing set for a multidimensional Fourier transform. In the above, A is the indexing set for the R-dimensional Fourier transform of size $N_1 \times \cdots \times N_R$. A finite abelian group can usually be written in several ways as the direct product of cyclic groups. The number of cyclic group factors as well as their respective sizes can vary. Each direct product representation of A will be called a *presentation* of A.

The importance of nonuniqueness will be brought out in the following chapters, but we can briefly describe it as follows.

> A Fourier transform will be assigned to each finite abelian group. Each presentation produces a standard multidimensional Fourier transform whose dimension and size are fixed by the presentation. We will prove in chapter 4 that standard multidimensional Fourier transforms corresponding to different presentations differ solely by automorphisms of the finite abelian group. An algorithm computing the multidimensional Fourier transform relative to one presentation can be used to compute the multidimensional Fourier transform relative to any other presentation. Several FFT algorithms, including the Good–Thomas algorithm, will be straightforward consequences of this remark.

3.2 Character Group

The character group A^* of a finite abelian group A will be defined. In analogy with signal processing, we can view A as the time parameter space on which signals are defined and A^* as the frequency parameter space on which Fourier transforms are defined. More generally, although we shall not do so, character groups can be defined for any locally compact abelian group, and Fourier transform theory can be developed to include all the cases of interest to signal processing. In this generality, the character group of the real line $\mathbf{R}$ can be identified with $\mathbf{R}$, and the character group of the circle group $\mathbf{R}/\mathbf{Z}$ can be identified with the group of integers $\mathbf{Z}$.

Denote the multiplicative group of complex numbers of modulus 1 by U and the subgroup of all complex Nth roots of unity by U_N. U_N is a cyclic group of order N having the element $e^{2\pi i/N}$ as a generator. In fact, any element of the form

$$e^{2\pi in/N}, \qquad GCD(n, N) = 1,$$

generates U_N. If M divides N, then U_M is a subgroup of U_N. We can write U_N as a direct product

$$U_N = U_{N_1} \times U_{N_2}$$

whenever $N = N_1 N_2$ and N_1 and N_2 are relatively prime.

A *character* $\mathbf{a}^*$ of A is a group homomorphism of A into U:

$$\mathbf{a}^*(\mathbf{a} + \mathbf{b}) = \mathbf{a}^*(\mathbf{a})\mathbf{a}^*(\mathbf{b}), \qquad \mathbf{a}, \mathbf{b} \in A.$$

We will usually write $\mathbf{a}^*(\mathbf{a})$ as $\langle \mathbf{a}, \mathbf{a}^* \rangle$. Since A has order N, $N\mathbf{a} = 0$ and

$$\langle \mathbf{a}, \mathbf{a}^* \rangle^N = \langle N\mathbf{a}, \mathbf{a}^* \rangle = 1, \qquad \mathbf{a} \in A,$$

for every character $\mathbf{a}^*$ of A. Thus

$$\langle \mathbf{a}, \mathbf{a}^* \rangle \in U_N, \qquad \mathbf{a} \in A,$$

and every character of A is a homomorphism of A into U_N.

Denote the set of all characters of A by A^*. We call A^* the *character group* of A, where an abelian group addition is defined on A^* by the rule

$$\langle \mathbf{a}, \mathbf{a}^* + \mathbf{b}^* \rangle = \langle \mathbf{a}, \mathbf{a}^* \rangle \langle \mathbf{a}, \mathbf{b}^* \rangle, \qquad \mathbf{a}^*, \mathbf{b}^* \in A^*, \ \mathbf{a} \in A.$$

Example 3.1 $A = \mathbf{Z}/N$.
The group A is cyclic of order N. Characters of A are completely determined by their action on any generator, say the element 1 of A. Set $u = e^{2\pi i/N}$. U_N is generated by u. There are N distinct characters of A, each determined by the power of u onto which the generator of A is mapped.

The characters can be defined using the group A as an indexing set. For $a \in A$, define the mapping $\psi(a)$ of A into U by the rule

$$\langle b, \psi(a) \rangle = u^{ab}, \qquad b \in A.$$

The power u^{ab} is well-defined, since a and b are defined modulo N. Direct computation shows that $\psi(a) \in A^*$ and that the mapping

$$\psi : A \to A^*$$

is an isomorphism of A onto A^*.

Example 3.2 $A = \mathbf{Z}/N_1 \times \mathbf{Z}/N_2$, $N = N_1 N_2$.
The elements

$$e_1 = (1, 0), \qquad e_2 = (0, 1)$$

generate A. A typical element $\mathbf{a} \in A$ can be written uniquely as

$$\mathbf{a} = (a_1, a_2) = a_1 e_1 + a_2 e_2, \qquad a_1 \in \mathbf{Z}/N_1, \quad a_2 \in \mathbf{Z}/N_2.$$

Characters of A are determined uniquely by their actions on e_1 and e_2. Set

$$u_1 = e^{2\pi i/N_1}, \qquad u_2 = e^{2\pi i/N_2}.$$

U_{N_1} and U_{N_2} are generated by u_1 and u_2.

Characters must map e_1 into U_{N_1} and e_2 into U_{N_2}. There are N distinct characters of A. We use A to index the characters. For $\mathbf{a} \in A$, the mapping $\psi(\mathbf{a})$ of A into U defined by

$$\langle \mathbf{b}, \psi(\mathbf{a}) \rangle = u_1^{a_1 b_1} u_2^{a_2 b_2}, \qquad \mathbf{b} \in A,$$

is a character of A, and the mapping $\psi : A \to A^*$ is an isomorphism of A onto A^*.

In general, if

$$A = \mathbf{Z}/N_1 \times \mathbf{Z}/N_2 \times \cdots \times \mathbf{Z}/N_R,$$

a typical element $\mathbf{a} \in A$ will be written as

$$\mathbf{a} = (a_1, \ldots, a_R), \qquad a_r \in \mathbf{Z}/N_r, \quad 1 \leq r \leq R.$$

The mapping $\psi(\mathbf{a})$ of A into U defined by

$$\langle \mathbf{b}, \psi(\mathbf{a}) \rangle = u_1^{a_1 b_1} \cdots u_R^{a_R b_R}, \qquad \mathbf{b} \in A,$$

where $u_r = e^{2\pi i/N_r}$, $1 \leq r \leq R$, is a character of A, and the mapping $\psi : A \to A^*$ is an isomorphism of A onto A^*. Since every finite abelian group can be written in this form, we have the following result.

Theorem 3.1 *If A is a finite abelian group, then A is isomorphic to its character group.*

The isomorphism between A and its character group is not canonical, since it depends on presentation and choice of isomorphisms between the cyclic group factors of the presentation. Observe that the isomorphism given above satisfies

$$\langle \mathbf{b}, \psi(\mathbf{a}) \rangle = \langle \mathbf{a}, \psi(\mathbf{b}) \rangle, \qquad \mathbf{a},\ \mathbf{b} \in A.$$

Such an isomorphism is called *symmetric.*

For finite abelian groups A_1 and A_2, set $A = A_1 \times A_2$. A typical element $\mathbf{a} \in A$ will be written as $\mathbf{a} = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)})$, where $\mathbf{a}^{(1)} \in A_1$ and $\mathbf{a}^{(2)} \in A_2$.

Theorem 3.2 *If $A = A_1 \times A_2$ and ψ_1 and ψ_2 are isomorphisms of A_1 and A_2 onto their character groups, then the mapping*

$$\psi : A \to A^*$$

defined by

$$\langle \mathbf{b}, \psi(\mathbf{a}) \rangle = \langle \mathbf{b}^{(1)}, \psi_1(\mathbf{a}^{(1)}) \rangle \langle \mathbf{b}^{(2)}, \psi_2(\mathbf{a}^{(2)}) \rangle, \quad \mathbf{a}, \mathbf{b} \in A,$$

is an isomorphism of A onto its character group.

We write $\psi = \psi_1 \times \psi_2$ and $\psi(\mathbf{a}) = \psi_1(\mathbf{a}) \times \psi_2(\mathbf{a})$ with the understanding that ψ_1 and ψ_2 have been extended to characters on A by the rule

$$\psi_1(\mathbf{a}) = \psi_1(\mathbf{a}^{(1)}), \qquad \psi_2(\mathbf{a}) = \psi_2(\mathbf{a}^{(2)}), \qquad \mathbf{a} \in A.$$

In general, if A is a finite abelian group given by the direct product

$$A = A_1 \times \cdots \times A_M,$$

an isomorphism ψ from A onto A^* can be constructed by the direct product of isomorphisms ψ_r from the factors A_r onto their character groups.

An isomorphism ψ from A onto A^* defines a mapping

$$\Psi_\psi : A \times A \to \mathbf{Z}/N$$

by the rule

$$\langle \mathbf{a}, \psi(\mathbf{b}) \rangle = e^{(2\pi i/N)\Psi_\psi(\mathbf{a},\mathbf{b})}, \qquad \mathbf{a}, \mathbf{b} \in A.$$

Ψ_ψ is a $\mathbf{Z}/N$-bilinear form on $A \times A$:

$$\begin{aligned} \Psi_\psi(\mathbf{a} + \mathbf{a}', \mathbf{b}) &= \Psi_\psi(\mathbf{a}, \mathbf{b}) + \Psi_\psi(\mathbf{a}', \mathbf{b}), \\ \Psi_\psi(\mathbf{a}, \mathbf{b} + \mathbf{b}') &= \Psi_\psi(\mathbf{a}, \mathbf{b}) + \Psi_\psi(\mathbf{a}, \mathbf{b}'), \qquad \mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}' \in A. \end{aligned}$$

Conversely, a $\mathbf{Z}/N$-bilinear form on $A \times A$ uniquely determines a homomorphism of A onto A^*. We call a $\mathbf{Z}/N$-bilinear form *nonsingular* if the corresponding homomorphism is an isomorphism.

Theorem 3.3 *The formula*

$$\langle \mathbf{a}, \phi(\mathbf{b}) \rangle = e^{(2\pi i/N)\Psi_\psi(\mathbf{a},\mathbf{b})}, \qquad \mathbf{a}, \mathbf{b} \in A,$$

determines a one-to-one correspondence between the set of all isomorphisms ψ from A onto A^* and the set of all nonsingular $\mathbf{Z}/N$-bilinear forms Ψ_ψ of A.

Denote the character group of A^* by A^{**}. Theorem 3.1 implies that the groups A, A^*, and A^{**} are isomorphic. We will now define an isomorphism between A and A^{**} that is *canonical* in the sense that it depends solely on group structure and not on presentation. As we will see, this canonical isomorphism is fundamentally tied to the Fourier transform.

For $\mathbf{a} \in A$, define a mapping $\Theta(\mathbf{a})$ of A^* by the rule

$$\Theta(\mathbf{a})(\mathbf{a}^*) = \langle \mathbf{a}, \mathbf{a}^* \rangle, \qquad \mathbf{a}^* \in A^*.$$

The above definition implies that $\Theta(\mathbf{a})$ is a character of A^*, and hence

$$\Theta : A \to A^{**}.$$

This definition also implies that the mapping is a homomorphism. To see that it is an isomorphism, simply observe that if

$$\langle \mathbf{a}, \mathbf{a}^* \rangle = 1 \qquad \text{for all } \mathbf{a}^* \in A^*,$$

then $\mathbf{a} = 0$.

We have left the proof of the following theorem as a problem.

Theorem 3.4 *If $\mathbf{a} \in A$, then*

$$\sum_{\mathbf{a}^* \in A^*} \langle \mathbf{a}, \mathbf{a}^* \rangle = \begin{cases} N, & \mathbf{a} = 0, \\ 0, & \mathbf{a} \neq 0. \end{cases}$$

If $\mathbf{a}^* \in A^*$, then

$$\sum_{\mathbf{a} \in A} \langle \mathbf{a}, \mathbf{a}^* \rangle = \begin{cases} N, & \mathbf{a}^* = 0, \\ 0, & \mathbf{a}^* \neq 0. \end{cases}$$

3.3 Duality

Unless otherwise specified, fix an isomorphism ψ from A onto A^*. Consider a subgroup B of A.

The *dual* $B^\perp$ of B is the set defined by

$$B^\perp = \{\mathbf{a} \in A : \langle \mathbf{b}, \psi(\mathbf{a}) \rangle = 1, \text{ for all } \mathbf{b} \in B\}. \tag{3.1}$$

$B^\perp$ is the *orthogonal complement* of B relative to the bilinear form Ψ_ψ,

$$B^\perp = \{\mathbf{a} \in A : \ \Psi_\phi(\mathbf{b}, \mathbf{a}) = 0, \text{ for all } \mathbf{b} \in B\}.$$

The example below shows that $B^\perp$ is not necessarily a *complement* of the subgroup B of A in the sense that $A = B \oplus B^\perp$. In fact, nontrivial subgroups B exist such that $B^\perp = B$. We say B is *self-dual* in this case.

Since ψ is an isomorphism of A onto A^*,

$$\psi(B^\perp) = \{\psi(\mathbf{b}^\perp) : \ \mathbf{b}^\perp \in B^\perp\}$$

is the subgroup of all characters of A that act trivially on B. Consider the quotient group A/B of B-cosets,

$$\mathbf{a} + B = \{\mathbf{a} + \mathbf{b} : \ \mathbf{b} \in B\}, \qquad \mathbf{a} \in A,$$

with abelian group addition,

$$(\mathbf{a} + B) + (\mathbf{a}' + B) = (\mathbf{a} + \mathbf{a}') + B.$$

The character $\psi(\mathbf{b}^\perp)$, $\mathbf{b}^\perp \in B^\perp$, induces a character $\psi_1(\mathbf{b}^\perp)$ on A/B by the rule

$$\langle \mathbf{a} + B, \psi_1(\mathbf{b}^\perp) \rangle = \langle \mathbf{a}, \psi(\mathbf{b}^\perp) \rangle, \qquad \mathbf{a} \in A, \ \ \mathbf{b}^\perp \in B^\perp.$$

The formula is well-defined by definition of $B^\perp$, i.e., independent of coset representation. The mapping

$$\psi_1 : B^\perp \to (A/B)^*$$

is an isomorphism of $B^\perp$ onto $(A/B)^*$.

A second isomorphism

$$\psi_2 : A/B^\perp \to B^*$$

is defined by the formula

$$\langle \mathbf{b}, \psi_2(\mathbf{a} + B^\perp) \rangle = \langle \mathbf{b}, \psi(\mathbf{a}) \rangle, \qquad \mathbf{a} \in A, \ \ \mathbf{b} \in B.$$

Again, it is easy to see that ψ_2 is well-defined and independent of coset representation.

Example 3.3 $A = \mathbf{Z}/N$, $N = N_1 N_2$.
Define the isomorphism ψ by

$$\langle a, \psi(a') \rangle = e^{2\pi i a' a / N}, \qquad a, a' \in A,$$

and the subgroup B by

$$B = N_1 \mathbf{Z}/N = \{0, N_1, \ldots, (N_2 - 1)N_1\}.$$

Then the dual is given by

$$B^\perp = N_2 \mathbf{Z}/N = \{0, N_2, \ldots, (N_1 - 1)N_2\}.$$

Some special cases are of interest.

1. $N = N_1^2$; then $B = B^\perp$ and B is self-dual.
2. $N = N_1 N_2$ where N_1 and N_2 are relatively prime; then

$$A = B \oplus B^\perp.$$

A coset representation for the quotient group A/B can be given by

$$A/B = \{0, 1, \ldots, N_1 - 1\},$$

relative to which the isomorphism ψ_1 is given by

$$\langle j, \psi_1(kN_2) \rangle = e^{(2\pi i/N_1)jk}, \qquad 0 \le j, k < N_1.$$

Theorem 3.5 *Suppose A_1 and A_2 are finite abelian groups having relatively prime orders and ψ_1 and ψ_2 are isomorphisms of A_1 and A_2 onto their character groups. If B_1 and B_2 are subgroups of A_1 and A_2, and if $B_1^\perp$ and $B_2^\perp$ are their duals relative to ψ_1 and ψ_2, then the dual of $B = B_1 \times B_2$ in $A = A_1 \times A_2$ relative to $\psi = \psi_1 \times \psi_2$ is given by*

$$B^\perp = B_1^\perp \times B_2^\perp.$$

Proof Since A_1 and A_2 have relatively prime orders,

$$\langle \mathbf{b}, \psi(\mathbf{a}) \rangle = 1$$

if and only if

$$\langle \mathbf{b}^{(1)}, \psi_1(\mathbf{a}^{(1)}) \rangle = \langle \mathbf{b}^{(2)}, \psi_2(\mathbf{a}^{(2)}) \rangle = 1.$$

The theorem immediately follows.

Theorem 3.5 easily extends to the case where the finite abelian group A is the direct product of any number of factors whose orders are pairwise relatively prime. In the next section, we will show that every finite abelian group A admits such a factorization, where the orders of the factors are distinct prime powers. Such factors are called *primary factors* (exact definition to be given in the following section). Duality in A can be described using duality in the primary factors of A.

3.4 Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) is a major tool in algorithm design. It is the basis of the Good–Thomas FFT algorithm and the Agarwal–Cooley large-size convolution algorithm [1, 2, 4]. It can be stated in several ways, but in its complete form, it is a statement about rings. The ring version, and especially its use of idempotents, will be important in implementation, where it provides uniformity.

A *primary factorization* of an integer N is any factorization

$$N = N_1 N_2 \cdots N_R \tag{3.2}$$

into pairwise relatively prime integers. We will need the following result about primary factorizations. For $n \in \mathbf{Z}$, if $N_r \mid n$, $1 \leq r \leq R$, then $N \mid n$.

First we will give a weak form using abelian group structure only.

Theorem 3.6 *The direct product of cyclic groups having pairwise relatively prime orders is a cyclic group.*

Proof Suppose

$$A = \mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R, \qquad N = N_1 N_2 \cdots N_R,$$

where the integers N_r, $1 \leq r \leq R$, are pairwise relatively prime. We claim that the element $e = (1, \ldots, 1) \in A$ has order N and thus generates A. Observe that $ke = 0$ in A implies $N_r \mid k$, $1 \leq r \leq R$. Since the integers N_r, $1 \leq r \leq R$, are pairwise relatively prime, we must have $N \mid k$, proving the claim and the theorem.

Consider $\mathbf{Z}/N$ as a ring under addition and multiplication mod N. Then the direct product

$$\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R$$

is a ring under componentwise addition and multiplication. It follows that every presentation of a finite abelian group induces a ring structure. The preceding theorem states that $\mathbf{Z}/N$ and $\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R$ are isomorphic as groups whenever $N = N_1 \cdots N_R$ is a primary factorization. The complete statement of the CRT goes further by explicitly defining a ring isomorphism and its inverse.

Suppose $N = N_1 \cdots N_R$ is a primary factorization. One way is simple. For $n \in \mathbf{Z}/N$, set

$$\phi(n) = (n \bmod N_1, \ldots, n \bmod N_R).$$

Arguing as in Theorem 3.6, we have that ϕ is a ring isomorphism of $\mathbf{Z}/N$ onto $\mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R$. To construct an inverse we require the next result.

Theorem 3.7 *Let $N = N_1 N_2 \cdots N_R$ be a primary factorization. Then there exist uniquely determined elements*

$$e_1, e_2, \ldots, e_R \in \mathbf{Z}/N$$

satisfying

$$\begin{aligned} e_r &\equiv 1 \bmod N_r, \\ e_r &\equiv 0 \bmod N_s, \qquad 1 \leq r, s \leq R, \quad r \neq s. \end{aligned}$$

Proof Since ϕ is a bijection, we can define e_r by the condition that $\phi(e_r)$ is 0 except in the rth component, where it takes the value 1. The theorem follows by direct computation.

The set $\{e_1, e_2, \ldots, e_R\}$ is called the *complete system of idempotents* for the primary factorization $N = N_1 \cdots N_R$.

Several properties of a complete system of idempotents immediately follow from the fact that ψ is a ring homomorphism. We list three of the most important properties that, in fact, characterize complete systems of idempotents.

$$\begin{aligned} e_r^2 &\equiv e_r \bmod N, \quad 1 \le r \le R, \\ e_r e_s &\equiv 0 \bmod N, \quad 1 \le r, s \le R, \quad r \ne s, \\ \sum_{r=1}^{R} e_r &\equiv 1 \bmod N. \end{aligned}$$

The inverse of ϕ will be defined using idempotents.

Theorem 3.8 *Let $N = N_1 N_2 \cdots N_R$ be a primary factorization and e_r, $1 \le r \le R$, the corresponding complete system of idempotents. Then every $n \in \mathbf{Z}/N$ can be written uniquely in the form*

$$n = n_1 e_1 + n_2 e_2 + \cdots + n_R e_R \bmod N, \quad 0 \le n_r < N_r.$$

The coefficients of the expansion satisfy

$$n_r \equiv n \bmod N_r, \quad 1 \le r \le R.$$

Proof If $n = n_1 e_1 + n_2 e_2 + \cdots + n_R e_R \bmod N$, $0 \le n_r < N_r$, $1 \le r \le R$, then by definition $\phi(n) = (n_1, \ldots, n_R)$ with $n_r \equiv n \bmod N_r$, proving uniqueness. The same argument shows that if we define

$$n_r \equiv n \bmod N_r, \quad 0 \le n_r < N, \quad 1 \le r \le R$$

then

$$\phi(n) = \phi(n_1 e_1 + \cdots + n_R e_R)$$

which, since ϕ is an isomorphism, completes the proof.

Since $N_r e_r \equiv 0 \bmod N$, the product $n_r e_r$ is well-defined in $\mathbf{Z}/N$ for all $n_r \in \mathbf{Z}/N_r$, and we can write without ambiguity $n \equiv n_1 e_1 + \cdots + n_R e_R$, $n_r \in \mathbf{Z}/N_r$. We can now define the inverse ϕ^{-1} of ϕ by the formula

$$\phi^{-1}(n_1, \ldots, n_R) \equiv n_1 e_1 + \cdots + n_R e_R \bmod N, \quad n_r \in \mathbf{Z}/N_r.$$

Set

$$A_r = gp(e_r), \quad 1 \le r \le R.$$

Using the ring multiplication in $\mathbf{Z}/N$, Theorem 3.8 implies that if

$$\begin{aligned} \mathbf{a} &= \mathbf{a}^{(1)} + \cdots + \mathbf{a}^{(R)}, \quad \mathbf{a}^{(r)} \in A_r, \\ \mathbf{b} &= \mathbf{b}^{(1)} + \cdots + \mathbf{b}^{(R)}, \quad \mathbf{b}^{(r)} \in A_r, \end{aligned}$$

then

$$\mathbf{ab} = \mathbf{a}^{(1)}\mathbf{b}^{(1)} + \cdots + \mathbf{a}^{(R)}\mathbf{b}^{(R)}, \mathbf{a}^{(r)}\mathbf{b}^{(r)} \in A_r.$$

By the CRT, $\mathbf{Z}/N = \mathbf{Z}/p_1^{\alpha_1} \times \cdots \times \mathbf{Z}/p_M^{\alpha_M}$, where $N = p_1^{\alpha_1} \cdots p_M^{\alpha_M}$ is the factorization of N into distinct prime powers. More generally, we have the following result.

Theorem 3.9 *Suppose A is finite abelian group of order N, with $N = p_1^{\alpha_1} \cdots p_M^{\alpha_M}$, the factorization of N into distinct prime powers. Then A is isomorphic to the direct product*

$$A_1 \times \cdots \times A_M,$$

where A_m is a finite abelian group of order $p_m^{\alpha_m}$ and is isomorphic to a direct product

$$\mathbf{Z}/p_m^{\alpha_m(1)} \times \cdots \times \mathbf{Z}/p_m^{\alpha_m(R)}.$$

Proof By the fundamental theorem,

$$A \cong \mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R, \quad N = N_1 \cdots N_R.$$

Write

$$N_r = p_1^{\alpha_1(r)} \cdots p_M^{\alpha_M(r)}, \quad 1 \le r \le R, \quad \alpha_m(r) \ge 0.$$

By the CRT,

$$\mathbf{Z}/N_r \cong \mathbf{Z}/p_1^{\alpha_1(r)} \times \cdots \times \mathbf{Z}/p_M^{\alpha_M(r)}.$$

Rearranging the factors completes the proof of the theorem.

The factorization in Theorem 3.9 is uniquely determined by the conditions on the orders. In fact, if B_m is the subset of all elements in A having order a power of the prime p_m, then B_m is the subgroup of A isomorphic to A_m, and

$$A = B_1 \oplus \cdots \oplus B_M.$$

We call this direct sum decomposition of A the *primary factorization* of A and refer to the factors B_m as *primary factors*. Many results in theory and algorithm design can first be proved for primary factors and then lifted to the general case using the direct sum. The CRT admits generalization to rings other than $\mathbf{Z}/N$. In particular, the CRT can be proved for the ring of polynomials over any field [3, 4].

3.5 Vector Space $L(A)$

Denote the space of all complex-valued functions on a finite set X by $L(X)$. Then $L(X)$ is a vector space over $\mathbf{C}$ under function addition and complex multiplication:

$$\begin{aligned}(f+g)(\mathbf{x}) &= f(\mathbf{x})+g(\mathbf{x}), \quad f,g\in L(X),\ \mathbf{x}\in X,\\ (\alpha f)(\mathbf{x}) &= \alpha(f(\mathbf{x})), \quad \alpha\in\mathbf{C},\ f\in L(X),\ \mathbf{x}\in X.\end{aligned}$$

$L(X)$ is a finite-dimensional vector space having dimension the order of X. A basis for $L(X)$ can be given by the *evaluation functions*

$$\{e_{\mathbf{x}} : \mathbf{x}\in X\} \tag{3.3}$$

defined by setting

$$e_{\mathbf{x}}(\mathbf{y}) = \begin{cases} 1, & \mathbf{y}=\mathbf{x},\\ 0, & \mathbf{y}\neq\mathbf{x},\end{cases} \qquad \mathbf{y}\in X.$$

The basis of evaluation functions is called a *canonical basis*, since it depends solely on the set X. By abuse of language, we will often denote this basis by X itself. In practice, the set X must be ordered for the set (3.3) to be a basis. The ordering on X determines an ordering on the canonical basis.

An inner product is defined on $L(X)$ by the rule

$$(f,g) = \sum_{\mathbf{x}\in X} f(\mathbf{x})\overline{g}(\mathbf{x}), \qquad f,g\in L(X),$$

where $\overline{g}$ denotes the complex conjugate of g. The basis X is orthonormal,

$$(e_{\mathbf{x}}, e_{\mathbf{y}}) = \begin{cases} 1, & \mathbf{x}=\mathbf{y},\\ 0, & \mathbf{x}\neq\mathbf{y},\end{cases} \qquad \mathbf{x},\mathbf{y}\in X.$$

Suppose now A is a finite abelian group of order N. Consider $A^*\subset L(A)$.

Theorem 3.10 *For* $\mathbf{a}^*,\ \mathbf{b}^*\in A^*$, *we have*

$$(\mathbf{a}^*,\mathbf{b}^*) = \begin{cases} N, & \mathbf{a}^*=\mathbf{b}^*,\\ 0, & \mathbf{a}^*\neq\mathbf{b}^*.\end{cases}$$

Proof Since by definition

$$(\mathbf{a}^*,\mathbf{b}^*) = \sum_{\mathbf{a}\in A}\langle \mathbf{a},\mathbf{a}^*\rangle\overline{\langle \mathbf{a},\mathbf{b}^*\rangle} = \sum_{\mathbf{a}\in A}\langle \mathbf{a},\mathbf{a}^*-\mathbf{b}^*\rangle,$$

the theorem follows from Theorem 3.4.

Theorem 3.10 implies that the set

$$\frac{1}{\sqrt{N}}A^*$$

is orthonormal in $L(A)$ and hence linearly independent, implying the next result.

Theorem 3.11 *The character group A^* of the finite abelian group A is an orthogonal basis of $L(A)$.*

References

[1] Agarwal, R.C. and Cooley, J.W. (1977), " A New Algorithm for Digital Convolution", *IEEE Trans.* **ASSP 25**, 392–410.

[2] Good, I.J. (1958), "The Interaction Algorithm and Practical Fourier Analysis," *J. Roy. Stat. Soc. Ser B* **20**, 361–372.

[3] Lang, S. (1970), *Algebra*, Addison-Wesley, Reading MA.

[4] Tolimieri, R., An, M., and Lu, C. (1988), *Algorithms for Discrete Fourier Transform and Convolution*, Springer-Verlag, New York.

Problems

1. Prove that for a finite abelian group A of order N

$$\sum_{\mathbf{a}\in A}\langle \mathbf{a}, \mathbf{a}^*\rangle = \begin{cases} N, & \mathbf{a}^* = 0, \\ 0, & \mathbf{a}^* \neq 0. \end{cases}$$

$$\sum_{\mathbf{a}^*\in A}\langle \mathbf{a}, \mathbf{a}^*\rangle = \begin{cases} N, & \mathbf{a} = 0, \\ 0, & \mathbf{a} \neq 0. \end{cases}$$

2. Show that the cyclic group $\mathbf{Z}/12$ is not isomorphic to $\mathbf{Z}/2\times\mathbf{Z}/2\times\mathbf{Z}/3$.

3. Determine the collection of all subgroups of $\mathbf{Z}/7$. Do the same for $\mathbf{Z}/p$, where p is any prime integer.

4. Determine the generators of the groups U_5, U_{12}, and U_{25}.

5. Determine the dual of the subgroup $16\mathbf{Z}/48$ in $\mathbf{Z}/48$. Write $\mathbf{Z}/48$ as the direct product of subgroups isomorphic to $\mathbf{Z}/3$ and $\mathbf{Z}/16$.

6. Find the primary factorization of $\mathbf{Z}/18$.

7. Suppose $A = \mathbf{Z}/15 \times \mathbf{Z}/15$ and Ψ is the $\mathbf{Z}/15$-bilinear form on $A \times A$ whose matrix is I_2. Determine the dual of the subgroup $B = 3\mathbf{Z}/15 \times 5\mathbf{Z}/15$ in A.

8. Determine the complete system of idempotents for the factorization $L = 12 \times 15$, for the factorization $M = 3 \times 4 \times 15$, and for the factorization $N = 3^2 \times 4 \times 5$. How are these idempotent systems related?

9. Find the primary factorization of the group $A = \mathbf{Z}/5 \times \mathbf{Z}/15 \times \mathbf{Z}/35$.

10. In problem 9, compute the idempotents of factorization and explicitly write the CRT isomorphism and its inverse.

4

Fourier Transform of Finite Abelian Groups

4.1 Introduction

The standard definition of the multidimensional Fourier transform (MDFT) assumes a fixed coordinate system representation of the indexing set. In this chapter, we will define and explore the MDFT in a more abstract setting, one that removes the dependence on coordinates and solely references the additive abelian group structure of the indexing set. This approach highlights the fundamental role played by the duality between periodization and decimation in MDFT algorithm design. This duality lies at the heart of all standard and recently discovered divide and conquer MDFT algorithms. Emphasizing the unity underlying these algorithms permits a deeper understanding of their differences and how these differences can be exploited in implementation. This is especially true in the design of massively parallel algorithms. Algorithm design is reduced to relatively few basic principles without having to account for the details of specific coordinates.

4.2 Fourier Transform of A

Consider a finite abelian group A. The Fourier transform of A is the mapping F from $L(A)$ into $L(A^*)$ defined by the formula

$$F(f)(\mathbf{a}^*) = \sum_{\mathbf{a} \in A} f(\mathbf{a}) \langle \mathbf{a}, \mathbf{a}^* \rangle, \qquad f \in L(A), \quad \mathbf{a}^* \in A^*.$$

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

The mapping F is clearly linear. We will write F as F_A whenever we want to express the dependence of F on A.

Consider A as a basis of $L(A)$ and A^{**} as a basis of $L(A^*)$. The canonical isomorphism

$$\Theta : A \to A^{**}$$

extends to a linear isomorphism

$$L(\Theta) : L(A) \to L(A^*).$$

Theorem 4.1 *The Fourier transform F of a finite abelian group A maps the basis A of $L(A)$ onto the basis A^{**} of $L(A^*)$. Moreover, $F = L(\Theta)$.*

Proof By definition,

$$\begin{aligned} F(e_{\mathbf{b}})(\mathbf{a}^*) &= \sum_{\mathbf{a}\in A} e_{\mathbf{b}}(\mathbf{a})\langle \mathbf{a}, \mathbf{a}^*\rangle \\ &= \langle \mathbf{b}, \mathbf{a}^*\rangle \\ &= \Theta(\mathbf{b})(\mathbf{a}^*). \end{aligned}$$

As an immediate corollary of Theorem 4.1, we have the following:

Corollary 4.1 *The Fourier transform F of a finite abelian group A is a linear isomorphism of $L(A)$ onto $L(A^*)$.*

Suppose

$$\psi : A \to A^*$$

is any isomorphism of A onto A^*. Assign to ψ the linear transform

$$F_\psi : L(A) \to L(A)$$

defined by

$$F_\psi f = F(f) \circ \psi, \quad f \in L(A).$$

We call F_ψ the *Fourier transform presentation* corresponding to the isomorphism ψ.

Theorem 4.2 *If ψ_1 and ψ_2 are isomorphisms of A onto A^*, then*

$$F_{\psi_1} f(\mathbf{a}) = F_{\psi_2} f(\mathbf{b}), \quad f \in L(A),$$

where

$$\mathbf{a} = \psi_1^{-1}\psi_2 \mathbf{b}, \quad \mathbf{a},\ \mathbf{b} \in A.$$

Proof By definition,

$$\begin{aligned} F_{\psi_1} f(\mathbf{a}) &= \sum_{\mathbf{c}\in A} f(\mathbf{c})\langle \mathbf{c}, \psi_1(\mathbf{a})\rangle \\ &= \sum_{\mathbf{c}\in A} f(\mathbf{c})\langle \mathbf{c}, \psi_2\psi_2^{-1}\psi_1(\mathbf{a})\rangle \\ &= \sum_{\mathbf{c}\in A} f(\mathbf{c})\langle \mathbf{c}, \psi_2(\mathbf{b})\rangle \\ &= F_{\psi_2} f(\mathbf{b}), \end{aligned}$$

where $\mathbf{b} = \psi_2^{-1}\psi_1(\mathbf{a})$. The theorem follows.

F_{ψ_1} and F_{ψ_2} differ by the automorphism $\psi_2^{-1}\psi_1$ acting as a permutation of A. Any algorithm computing F_{ψ_1} can be used to compute F_{ψ_2}. Distinct presentations of A result in MDFTs of varying dimensions and sizes, but the entire collection can be computed by any algorithm that computes one.

Usually, one assigns to the presentation $A = \mathbf{Z}/N_1 \times \cdots \times \mathbf{Z}/N_R$ the isomorphism

$$\psi : A \to A^*$$

defined by the rule

$$\langle \mathbf{b}, \psi(\mathbf{a})\rangle = u_1^{a_1 b_1} \cdots u_R^{a_R b_R}, \quad \mathbf{a}, \mathbf{b} \in A,$$

where $u_j = e^{2\pi i/N_j}$. Then

$$\begin{aligned} F_\psi f(\mathbf{a}) &= Ff(\psi(\mathbf{a})) \\ &= \sum_{\mathbf{b}\in A} f(\mathbf{b})\langle \mathbf{b}, \psi(\mathbf{a})\rangle \\ &= \sum_{b_1 \in Z/N_1} \cdots \sum_{b_R \in \mathbf{Z}/N_R} f(b_1, \ldots, b_R) u_1^{a_1 b_1} \cdots u_R^{a_R b_R}, \end{aligned}$$

which is the formula for the $N_1 \times \cdots \times N_R$ R-dimensional Fourier transform.

4.3 Induced Fourier Transform

Throughout the remainder of this chapter, unless otherwise specified, we fix an isomorphism ψ from A onto A^* and a subgroup B of A of order M with $N = LM$. The isomorphisms

$$\psi_1 : B^\perp \to (A/B)^*, \qquad \psi_2 : A/B^\perp \to B^*,$$

defined in the previous chapter induce linear isomorphisms

$$F_1 : L(A/B) \to L(B^\perp),$$

$$F_2 : L(B) \to L(A/B^\perp)$$

by

$$\begin{aligned} F_1 f &= F_{A/B} f \circ \psi_1, \quad f \in L(A/B), \\ F_2 f &= F_B f \circ \psi_2, \quad f \in L(B). \end{aligned}$$

$F_{A/B}$ and F_B denote the abstract Fourier transforms of A/B and B. By abuse of language, we call F_1 and F_2 the Fourier transform presentations corresponding to ψ_1 and ψ_2. The Fourier transform presentation F_ψ can be built from these induced "lower-order" Fourier transforms F_1 and F_2. The designation lower-order Fourier transform is justified by the following results.

A set of elements in A

$$\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{L-1}$$

is called a *complete system of coset representatives* for A/B if every $\mathbf{a} \in A$ can be written uniquely as

$$\mathbf{a} = \mathbf{a}_l + \mathbf{b}, \quad 0 \le l < L, \, \mathbf{b} \in B.$$

For simplicity of notation, if $f \in L(A/B)$, set $f(\mathbf{a}_l) = f(\mathbf{a}_l + B)$. The following two theorems are straightforward consequences of the definitions of F_1 and F_2.

Theorem 4.3 *If $f \in L(A/B)$ and $\mathbf{a}_l$, $0 \le l < L$, is a complete system of coset representatives for A/B, then*

$$F_1 f(\mathbf{b}^\perp) = \sum_{l=0}^{L-1} f(\mathbf{a}_l) \langle \mathbf{a}_l, \psi(\mathbf{b}^\perp) \rangle, \quad \mathbf{b}^\perp \in B^\perp.$$

In the notation of example 3.3,

$$F_1 f(kN_2) = \sum_{j=0}^{N_1-1} f(j) e^{2\pi i jk/N_1}, \quad 0 \le k < N_1.$$

The right-hand side is the N_1-point finite Fourier transform of the values $f(0), f(1), \ldots, f(N_1 - 1)$.

Theorem 4.4 *If $f \in L(B)$ and $\mathbf{c}_m$, $0 \le m < M$, is a complete system of coset representatives for $A/B^\perp$, then*

$$F_2 f(\mathbf{c}_m) = \sum_{\mathbf{b} \in B} f(\mathbf{b}) \langle \mathbf{b}, \psi(\mathbf{c}_m) \rangle, \quad 0 \le m < M.$$

In the notation of example 3.3,

$$F_2 f(k) = \sum_{j=0}^{N_2-1} f(jN_1)e^{2\pi ijk/N_2}, \quad 0 \le k < N_2.$$

The right-hand side is the N_2-point finite Fourier transform of the values $f(0), f(N_1), \ldots, f((N_2-1)N_1)$.

4.4 Periodic and Decimated Data

A function $f \in L(A)$ is called *B-periodic* if

$$f(\mathbf{a}+\mathbf{b}) = f(\mathbf{a}), \quad \mathbf{a} \in A, \quad \mathbf{b} \in B.$$

A B-periodic function f is constant on the elements of a B-coset and defines a function $f \in L(A/B)$ by

$$f(\mathbf{a}+B) = f(\mathbf{a}), \quad \mathbf{a} \in A.$$

Conversely, a function $f \in L(A/B)$ defines a B-periodic function $f \in L(A)$ by the same formula. We will identify $L(A/B)$ and the space of B-periodic functions in $L(A)$ throughout this work. A function $f \in L(A)$ is called *B-decimated* if f vanishes off of B. The space of B-decimated functions in $L(A)$ can be identified with $L(B)$.

A B-periodic function $f \in L(A)$ is completely described by $L = N/M$ values. It is reasonable to expect that the computation of the Fourier transform Ff reflects the reduced-size input. The next result supports this idea and establishes the first part of the duality between periodization and decimation.

Theorem 4.5 *If $f \in L(A)$ is B-periodic, then its Fourier transform $F_\psi f$ is $B^\perp$-decimated and can be computed by the formula*

$$F_\psi f(\mathbf{b}^\perp) = MF_1(f)(\mathbf{b}^\perp), \quad \mathbf{b}^\perp \in B^\perp.$$

Proof Suppose f is B-periodic. Then

$$\begin{aligned}(F_\psi f)(\mathbf{c}) &= \sum_{l=0}^{L-1} \sum_{\mathbf{b}\in B} f(\mathbf{a}_l+\mathbf{b})\langle \mathbf{a}_l+\mathbf{b}, \psi(\mathbf{c})\rangle \\ &= \sum_{l=0}^{L-1} f(\mathbf{a}_l)\langle \mathbf{a}_l, \psi(\mathbf{c})\rangle \sum_{\mathbf{b}\in B} \langle \mathbf{b}, \psi(\mathbf{c})\rangle.\end{aligned}$$

If $\mathbf{c} \notin B^\perp$, then $\psi(\mathbf{c})$ is not the trivial character restricted to B, and we have by Theorem 3.4

$$\sum_{\mathbf{b}\in B} \langle \mathbf{b}, \psi(\mathbf{c})\rangle = 0, \quad \mathbf{c} \notin B^\perp,$$

implying that $F_\psi f$ is $B^\perp$-decimated. If $\mathbf{c} = \mathbf{b}^\perp \in B^\perp$, then

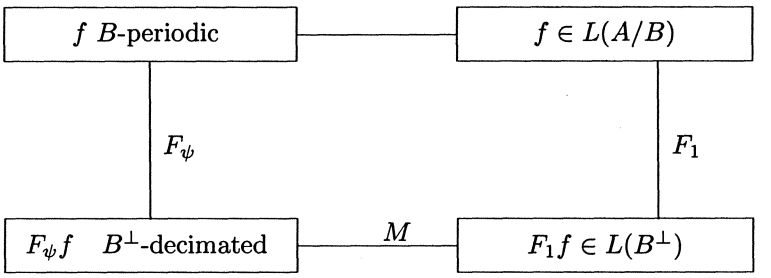
$$\sum_{\mathbf{b}\in B} \langle \mathbf{b}, \psi(\mathbf{b}^\perp)\rangle = M,$$

and we have by Theorem 4.3

$$\begin{aligned}(F_\psi f)(\mathbf{b}^\perp) &= M\sum_{l=0}^{L-1} f(\mathbf{a}_l)\langle \mathbf{a}_l, \psi(\mathbf{b}^\perp)\rangle \\ &= M(F_1 f)(\mathbf{b}^\perp),\end{aligned}$$

completing the proof of the theorem.

Figure 4.1 is a diagram of this relationship.



$$(F_\psi f)(\mathbf{b}^\perp) = M(F_1 f)(\mathbf{b}^\perp), \quad \mathbf{b}^\perp \in B^\perp.$$

Figure 4.1 Fourier transform of B-periodic functions

A partial converse is given by the next result.

Theorem 4.6 *If $f \in L(A)$ is B-decimated, then its Fourier transform $F_\psi f$ is $B^\perp$-periodic, which viewed as a function in $L(A/B^\perp)$ can be computed by the formula*

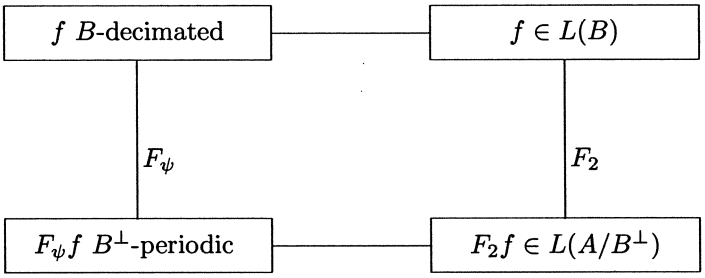
$$F_\psi f(\mathbf{c}) = F_2 f(\mathbf{c}), \quad \mathbf{c} \in A/B^\perp.$$

Proof From

$$\begin{aligned}(F_\psi f)(\mathbf{a}+\mathbf{b}^\perp) &= \sum_{\mathbf{b}\in B} f(\mathbf{b})\langle \mathbf{b}, \psi(\mathbf{a}+\mathbf{b}^\perp)\rangle \\ &= \sum_{\mathbf{b}\in B} f(\mathbf{b})\langle \mathbf{b}, \psi(\mathbf{a})\rangle \\ &= (F_\psi f)(\mathbf{a}),\end{aligned}$$

we have that $F_\psi f$ is $B^\perp$-periodic. The theorem follows from Theorem 4.4.

The relationship between a decimated function and its Fourier transform is pictured in Figure 4.2. Observe that the statement f is B-periodic if and only if $F_\psi f$ is $B^\perp$-decimated holds when ψ is symmetric, but not in general.



$$(F_\psi f)(\mathbf{c}) = F_2 f(\mathbf{c}), \quad \mathbf{c} \in A/B^\perp$$

Figure 4.2 Fourier transform of B-decimated functions

Example 4.1 If f is B-periodic, it is completely described by its values

$$f(0), f(1), \ldots, f(N_1 - 1).$$

The Fourier transform $F_\psi f$ vanishes except at the points

$$0, N_2, \ldots, (N_1 - 1)N_2,$$

where its values are given by

$$F_\psi f(nN_2) = N_1 \sum_{m=0}^{N_1-1} f(m) e^{(2\pi i/N_1)mn}, \quad 0 \le n < N_1.$$

4.5 Periodization and Decimation

For $f \in L(A)$, define $\mathrm{Per}_B f \in L(A)$ and $\mathrm{Dec}_B f \in L(A)$ by

$$\begin{aligned} (\mathrm{Per}_B f)(\mathbf{a}) &= \sum_{\mathbf{b} \in B} f(\mathbf{a} + \mathbf{b}), \quad \mathbf{a} \in A, \\ (\mathrm{Dec}_B f)(\mathbf{a}) &= \begin{cases} f(\mathbf{a}), & \mathbf{a} \in B, \\ 0, & \mathbf{a} \notin B. \end{cases} \end{aligned}$$

$\mathrm{Per}_B f$ is called the *B-periodization* of f, and Per_B is a linear homomorphism of $L(A)$ onto the space of B-periodic functions in $L(A)$. $\mathrm{Dec}_B f$ is

called the *B-decimation* of f, and Dec_B is a linear homomorphism of $L(A)$ onto the space of B-decimated functions in $L(A)$. The following results relate F_ψ with the periodization and decimation operations.

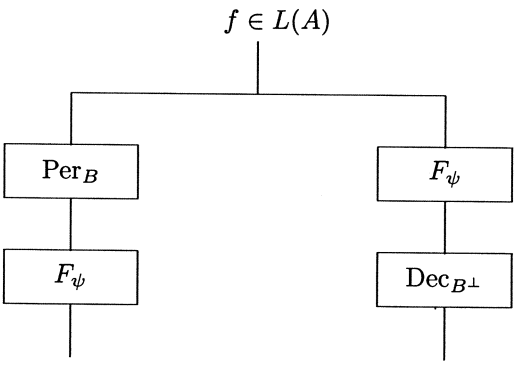
Theorem 4.7 $F_\psi \circ \mathrm{Per}_B = M\mathrm{Dec}_{B^\perp} \circ F_\psi$.

Proof For $f \in L(A)$, $\mathrm{Per}_B f$ is B-periodic and $F_\psi(\mathrm{Per}_B f)$ is $B^\perp$-decimated. By Theorem 3.4,

$$\begin{aligned} F_\psi(\mathrm{Per}_B f)(\mathbf{b}^\perp) &= \sum_{\mathbf{a}\in A}\sum_{\mathbf{b}\in B} f(\mathbf{a}+\mathbf{b})\langle \mathbf{a}, \psi(\mathbf{b}^\perp)\rangle \\ &= \sum_{\mathbf{b}\in B}\sum_{\mathbf{a}\in A} f(\mathbf{a})\langle \mathbf{a}-\mathbf{b}, \psi(\mathbf{b}^\perp)\rangle \\ &= M\sum_{\mathbf{a}\in A} f(\mathbf{a})\langle \mathbf{a}, \psi(\mathbf{b}^\perp)\rangle \\ &= M(F_\psi f)(\mathbf{b}^\perp), \qquad \mathbf{b}^\perp \in B^\perp, \end{aligned}$$

completing the proof of the theorem.

A diagram of Theorem 4.7 is given in Figure 4.3.



$$F_\psi f(\mathbf{b}^\perp) = \frac{1}{M} F_\psi(\mathrm{Per}_B f)(\mathbf{b}^\perp), \qquad \mathbf{b}^\perp \in B^\perp$$

Figure 4.3 Periodization–decimation

Combining Theorems 4.5 and 4.7, we can compute $F_\psi f$ on $B^\perp$ by a lower-order Fourier transform of the B-periodization of f:

$$F_\psi f(\mathbf{b}^\perp) = F_1(\mathrm{Per}_B f)(\mathbf{b}^\perp), \qquad \mathbf{b}^\perp \in B^\perp.$$

This formula is pictured in Figure 4.4. A similar argument shows the dual statement, which is given in Theorem 4.8, and diagrammed in Figure 4.5.

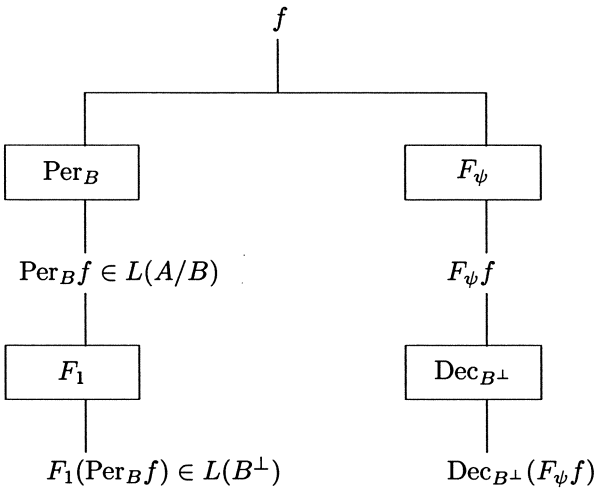
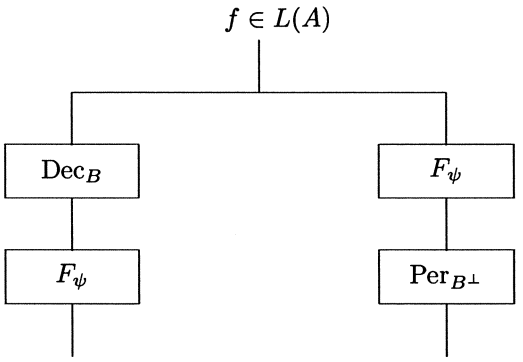


Figure 4.4 $F_\psi f(\mathbf{b}^\perp) = F_1(\mathrm{Per}_B f)(\mathbf{b}^\perp), \quad \mathbf{b}^\perp \in B^\perp$



$$\mathrm{Per}_{B^\perp}(F_\psi f)(\mathbf{c}) = \frac{N}{M} F_\psi(\mathrm{Dec}_B f)(\mathbf{c}), \qquad \mathbf{c} \in A/B^\perp$$

Figure 4.5 Decimation–periodization

Theorem 4.8 $F_\psi \circ \mathrm{Dec}_B = \frac{M}{N} \mathrm{Per}_{B^\perp} \circ F_\psi$.

Combining Theorems 4.6 and 4.8, we can compute the $B^\perp$-periodization of $F_\psi f$ by a lower-order Fourier transform of the B-decimation of f:

$$\mathrm{Per}_{B^\perp}(F_\psi f)(\mathbf{c}) = \frac{N}{M} F_2(\mathrm{Dec}_B f)(\mathbf{c}), \qquad \mathbf{c} \in A/B^\perp.$$

This formula is pictured in Figure 4.6.

Periodization–decimation for finite abelian groups is a special case of the *Poisson summation formula*, which can be stated for locally compact abelian groups. Such groups include all the familiar groups in signal processing applications: integers, real lines, circle, and all multidimensional extensions and p-adic number fields. The universal importance of Poisson summation in Fourier transform theory comes from its role of translating Fourier transform duality to subgroups and quotient groups.

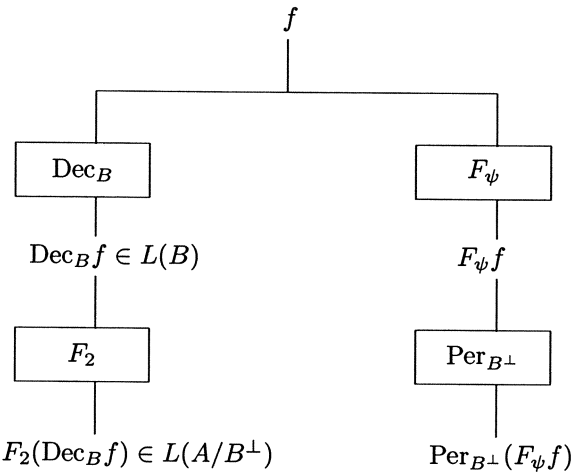


Figure 4.6 $\mathrm{Per}_{B^\perp}(F_\psi f)(\mathbf{c}) = \frac{N}{M} F_2(\mathrm{Dec}_B f)(\mathbf{c}), \quad \mathbf{c} \in A/B^\perp$

The fields of application of Poisson summation include both algebraic and analytic number theory, special function theory, harmonic analysis on groups, algebraic block coding, and signal processing. C.L. Siegel (1935) [7] in a series of papers in the 1930s used Poisson summation to unify many classical number-theoretic results and to build a foundation for subsequent work in automorphic form theory. More closely related to signal processing applications, but still in an abstract mathematical setting, A. Weil (1953, 1964) [9,10] in several fundamental works established Poisson summation as a major tool in harmonic analysis on groups including the Heisenberg group and the theory of theta functions. This work has found its way into more concrete applications in works of L. Auslander and R. Tolimieri (1975, 1985, 1988) [1–4] in the 1970s and 1980s. Some of Weil's ideas had been previously explored in a fundamental paper of J. Zak (1967) [11] introducing the Zak transform. Poisson summation is intimately related to the Zak transform. Concrete examples of these relationships closely following Weil can be found in the references to works of Auslander and Tolimieri where additional references can be found. In signal processing, the mathematics of C.E. Shannon's sampling theory (1948) [5, 6] is based on Poisson summation.

Algebraic block coding is another area of application for Poisson summation in the work of several authors, including that of N.J.A. Sloane (1977) [8].

References

[1] Auslander, L. and Tolimieri, R. (1975), *Abelian harmonic analysis. Theta functions and function algebra on nilmanifolds*, Springer-Verlag, New York.

[2] — (1985), "Radar ambiguity Function and Group theory," *SIAM J. Math. Anal.* **16**(3), 577–601.

[3] Auslander, L. and Tolimieri, R. (1988), "On Gabor Expansion of Signals," *IMA Proceedings on Signal Processing*, Minneapolis.

[4] Auslander, L. and Tolimieri, R. (1988), "Ambiguity Functions and Group Representation," *Proc. of Institute for Math. Science Research*, Berkely, CA.

[5] Shannon, C.E. (1948), "A Mathematical Theory of communication," *Bell System technical Journal* **27**, 379–423.

[6] Shannon, C.E. (1948), "A mathematical Theory of Communication," *Bell System technical Journal* **27**, 623–656.

[7] Siegel, C.L. (1935), "Über die analytische Theorie der quadratischen Formen," Anals of Math. **36**, 527–606 = *Gesammelte Abhandlungen* **I**, 326–405, Springer-Verlag, New York 1966.

[8] Sloane, N.J.A. (1977), "Error-Correcting Codes and Invariant Theory: New Applications of Nineteenth-Century Technique," *Amer. Math. Monthly* **84**, 82–107.

[9] Weil, A. (1953), "L'Intégration dans les groupes et ses applications topologique," *Hermann* **111**, 143–211.

[10] Weil, A. (1964), "Sur certaines groupes d'operateurs unitaires," *Acta Math.* **111**, 143–211.

[11] Zak, J. (1967), "Finite Translations in a Solid Physics," *Phys. Res. lett.* **19**, 1385–1397.

5

Cooley–Tukey and Good–Thomas

5.1 Introduction

The abstract definition of the Fourier transform and the statement of Fourier transform duality as expressed by the periodization–decimation results of chapter 4 provide a unifying principle underlying most 1-dimensional and multidimensional FFTs. The C–T FFTs of chapters 1 and 2 are expressions of this duality. In [1], a vector-radix FFT was derived, extending this duality relative to groups of affine motions on the indexing set. This class of FFT is highly parallelizable.

The Good–Thomas FFT will be seen to be an immediate consequence of the abstract definition when the indexing set A can be written as the direct sum, $A = B \oplus C$, of subgroups B and C. In this generality it includes both 1-dimensional and multidimensional applications and extends what is usually called the Good–Thomas FFT or the Prime Factor FFT. This approach combined with compatibility with respect to a group of affine actions leads to an important class of crystallographic FFTs called *Orbit-Exchange* [1].

In general, we cannot assume that A can be written nontrivially as a direct sum of subgroups. For the C–T FFT, we require only that A contain a nontrivial subgroup B. The basic idea is that the subgroup B partitions A into B-cosets and that the Fourier transform of data restricted to these B-cosets (exact meaning in section 3) can be combined to compute the Fourier transform of the initial data on A.

In chapter 8, the reduced transform algorithm will be introduced. Fourier transform duality is still the main tool, but it will be applied to a special

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

collection of subgroups that cover the indexing set. The distinction between cosets and subgroups is that periodization–decimation can be directly applied to data decimated to or periodized relative to subgroups, but cosets must be translated before direct application. These translations are responsible for the twiddle factors of the C–T FFT and disappear in the RTA. The trade-off is computational redundancy.

5.2 Good–Thomas FFT

Consider a finite abelian group A having the form of a group direct product $A = B \times C$. A typical element $\mathbf{a} \in A$ can be written as $\mathbf{a} = (\mathbf{b}, \mathbf{c})$, $\mathbf{b} \in B$ and $\mathbf{c} \in C$. Suppose ψ_B and ψ_C are isomorphisms from B and C onto their character groups and set $\psi = \psi_B \times \psi_C$. Observe that $C = B^\perp$ with respect to ψ.

For $f \in L(A)$ and $\mathbf{a}' = (\mathbf{b}', \mathbf{c}') \in A$,

$$(F_\psi f)(\mathbf{a}') = \sum_{\mathbf{c} \in C} \left(\sum_{\mathbf{b} \in B} f(\mathbf{b}, \mathbf{c}) \langle \mathbf{b}, \psi_B(\mathbf{b}') \rangle \right) \langle \mathbf{c}, \psi_C(\mathbf{c}') \rangle. \tag{5.1}$$

If $B = \mathbf{Z}/N_1$ and $C = \mathbf{Z}/N_2$, then as described in chapter 2, F_ψ is the $N_1 \times N_2$ 2-dimensional Fourier transform and (5.1) describes F_ψ as the tensor product $F(N_2) \otimes F(N_1)$. In general, F_ψ has the form of the tensor product of F_{ψ_B} and F_{ψ_C} in the following sense.

For each $\mathbf{c} \in C$, defining $f_\mathbf{c} \in L(B)$ by

$$f_\mathbf{c}(\mathbf{b}) = f(\mathbf{b}, \mathbf{c}), \qquad \mathbf{b} \in B,$$

we have that the inner sum is $F_{\psi_B} f_\mathbf{c}(\mathbf{b}')$, $\mathbf{b}' \in B$. For each $\mathbf{b}' \in B$ defining $g_{\mathbf{b}'} \in L(C)$ by

$$g_{\mathbf{b}'}(\mathbf{c}) = F_{\psi_B} f_\mathbf{c}(\mathbf{b}'), \qquad \mathbf{c} \in C,$$

we have that the outer sum computing $F_\psi f(\mathbf{a}')$ is given by

$$F_\psi f(\mathbf{a}') = F_{\psi_C} g_{\mathbf{b}'}(\mathbf{c}'), \qquad \mathbf{a}' \in A.$$

Two data transpositions are required, the first in the definition of $g_{\mathbf{b}'}$ and the second in transferring from $F_{\psi_C} g_{\mathbf{b}'}$ to $F_\psi f$.

In many applications, a fixed isomorphism χ of A onto A^* is given. In this case, we apply Theorem 4.2 to compute $F_\chi f$ by first computing $F_\psi f$ followed by the appropriate array permutation.

The best-known case of the Good–Thomas FFT, sometimes called the *Prime Factor FFT*, is when $N = pq$, for relatively prime p and q, and we want to compute $F(N)f$, the N-point 1-dimensional Fourier transform. In this case, we can take $B = pZ/N$ and $C = qZ/N$. Since typically $F(p)$, $F(q)$, and $F(N)$ are defined relative to particular group-isomorphisms, permutation is required. (See An et al., 1991 [1].)

5.3 Abstract Cooley–Tukey FFT

The Good–Thomas FFT applies whenever A can be written as a direct sum $A = B \oplus C$. More generally, every subgroup B of a finite abelian group A determines an abstract Cooley–Tukey (C–T) FFT even if B no longer splits in A.

First, we will give a straightforward proof of an abstract C–T FFT [1]. Choose a complete system of B-coset representatives

$$\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{L-1}, \qquad \mathbf{a}_0 = 0,$$

and a complete system of $B^\perp$-coset representatives

$$\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{M-1}, \qquad \mathbf{c}_0 = 0.$$

For $f \in L(A)$ we can compute $F_\psi f$ on the coset $\mathbf{c}_m + B^\perp$ by

$$\begin{aligned} & F_\psi f(\mathbf{c}_m + \mathbf{b}^\perp) \\ &= \sum_{l=0}^{L-1} \sum_{\mathbf{b} \in B} f(\mathbf{a}_l + \mathbf{b}) \langle \mathbf{a}_l + \mathbf{b}, \psi(\mathbf{c}_m + \mathbf{b}^\perp) \rangle \\ &= \sum_{l=0}^{L-1} \left[\left(\sum_{\mathbf{b} \in B} f(\mathbf{a}_l + \mathbf{b}) \langle \mathbf{b}, \psi(\mathbf{c}_m) \rangle \right) \langle \mathbf{a}_l, \psi(\mathbf{c}_m) \rangle \right] \langle \mathbf{a}_l, \psi(\mathbf{b}^\perp) \rangle, \end{aligned}$$

with $\mathbf{b}^\perp \in B^\perp$ and $0 \le m < M$. $F_\psi f$ can be computed in the following sequence of steps.

- Form the functions $f_l \in L(B)$, $0 \le l < L$,

$$f_l(\mathbf{b}) = f(\mathbf{a}_l + \mathbf{b}), \qquad \mathbf{b} \in B.$$

- Compute the Fourier transforms $F_2 f_l \in L(A/B^\perp)$, $0 \le l < L$,

$$F_2 f_l(\mathbf{c}_m) = \sum_{\mathbf{b} \in B} f_l(\mathbf{b}) \langle \mathbf{b}, \psi(\mathbf{c}_m) \rangle, \qquad 0 \le m < M.$$

- Form the functions $g_m \in L(A/B)$, $0 \le m < M$,

$$g_m(\mathbf{a}_l) = F_2 f_l(\mathbf{c}_m), \qquad 0 \le l < L.$$

- Compute the products $h_m \in L(A/B)$, $0 \le m < M$,

$$h_m(\mathbf{a}_l) = g_m(\mathbf{a}_l) \langle \mathbf{a}_l, \psi(\mathbf{c}_m) \rangle, \qquad 0 \le l < L.$$

- Compute the Fourier transforms $F_1 h_m \in L(B^\perp)$, $0 \le m < M$,

$$F_1 h_m(\mathbf{b}^\perp) = \sum_{l=0}^{L-1} h_m(\mathbf{a}_l)\langle \mathbf{a}_l, \psi(\mathbf{b}^\perp)\rangle, \qquad \mathbf{b}^\perp \in B^\perp.$$

- Form the transpose

$$F_\psi f(\mathbf{c}_m + \mathbf{b}^\perp) = F_1 h_m(\mathbf{b}^\perp), \qquad 0 \le m < M,\ \mathbf{b}^\perp \in B^\perp.$$

We will now give a second derivation based on the results contained in chapter 4 that will explicitly bring out the underlying periodization–decimation structure of the C–T FFT. A dual C–T FFT can also be derived.

Denote the set of characters of B by

$$\{\chi_m : 0 \le m < M\}.$$

For $f \in L(A)$ and $0 \le m < M$, we define the *generalized periodizations* $\mathrm{Per}_B^{(m)} f \in L(A)$ relative to the subgroup B by

$$\mathrm{Per}_B^{(m)} f(\mathbf{a}) = \sum_{\mathbf{b}\in B} f(\mathbf{a}+\mathbf{b})\langle \mathbf{b}, \chi_m\rangle, \qquad \mathbf{a} \in A. \tag{5.2}$$

Except for $\mathrm{Per}_B^{(0)} f$, the generalized periodizations are not B-periodic but are uniquely determined by their values on the B-coset representatives.

For $0 \le m < M$, denote by $\mathrm{Per}_B(m)$ the set of all $g \in L(A)$ satisfying

$$g(\mathbf{a}+\mathbf{b}) = \langle \mathbf{b}, \chi_m\rangle^{-1} g(\mathbf{a}), \qquad \mathbf{a} \in A,\ \mathbf{b} \in B.$$

Theorem 5.1 *For $f \in L(A)$,*

$$\mathrm{Per}_B^{(m)} f \in \mathrm{Per}_B(m), \qquad 0 \le m < M,$$

and

$$f = \frac{1}{M} \sum_{m=0}^{M-1} \mathrm{Per}_B^{(m)} f.$$

Proof Set $g = \mathrm{Per}_B^{(m)} f$. Then, for $\mathbf{a} \in A$ and $\mathbf{b}' \in B$,

$$\begin{aligned} g(\mathbf{a}+\mathbf{b}') &= \sum_{\mathbf{b}\in B} f(\mathbf{a}+\mathbf{b}+\mathbf{b}')\langle \mathbf{b}, \chi_m\rangle \\ &= \sum_{\mathbf{b}\in B} f(\mathbf{a}+\mathbf{b})\langle \mathbf{b}-\mathbf{b}', \chi_m\rangle \\ &= \langle \mathbf{b}', \chi_m\rangle^{-1} \sum_{\mathbf{b}\in B} f(\mathbf{a}+\mathbf{b})\langle \mathbf{b}, \chi_m\rangle \\ &= \langle \mathbf{b}', \chi_m\rangle^{-1} g(\mathbf{a}), \end{aligned}$$

proving the first part. The second part follows from Theorem 3.4,

$$\begin{aligned}\sum_{m=0}^{M-1} \mathrm{Per}_B^{(m)} f(\mathbf{a}) &= \sum_{\mathbf{b}\in B} f(\mathbf{a}+\mathbf{b}) \sum_{m=0}^{M-1} \langle \mathbf{b}, \chi_m \rangle \\ &= Mf(\mathbf{a}).\end{aligned}$$

By the theorem we can compute $F_\psi f$ by computing $F_\psi(\mathrm{Per}_B^{(m)} f)$, $0 \le m < M$. The initial generalized periodizations can be implemented in several ways, depending on machine architecture and transform size. They can always be implemented by the L M-point Fourier transform or their equivalent in a multidimensional setting. The resulting algorithm is the usual statement of the decimation in frequency C–T. In chapter 8, we will describe the reduced transform algorithm that has an initial step consisting of pure periodizations followed by independent Fourier transforms.

The next result extends the results of chapter 4 to the sets $\mathrm{Per}_B(m)$. Denote by $\mathrm{Dec}_B^{(m)}$, $0 \le m < M$, the set of all $g \in L(A)$ that vanish off of the set

$$\{\mathbf{a} \in A : \langle \mathbf{b}, \chi_m \rangle = \langle \mathbf{b}, \psi(\mathbf{a}) \rangle, \quad \text{for all } \mathbf{b} \in B\}.$$

Theorem 5.2 *If $g \in \mathrm{Per}_B(m)$, then $F_\psi g \in \mathrm{Dec}_B(m)$.*

Proof The case $m = 0$ has been proved in the previous chapter. For $0 < m < M$ and for all $\mathbf{b} \in B$,

$$\begin{aligned}F_\psi g(\mathbf{a}) &= \sum_{\mathbf{c}\in A} g(\mathbf{c}) \langle \mathbf{c}, \psi(\mathbf{a}) \rangle \\ &= \sum_{\mathbf{c}\in A} g(\mathbf{c}+\mathbf{b}) \langle \mathbf{c}+\mathbf{b}, \psi(\mathbf{a}) \rangle \\ &= \langle \mathbf{b}, \chi_m \rangle^{-1} \langle \mathbf{b}, \psi(\mathbf{a}) \rangle \sum_{\mathbf{c}\in A} g(\mathbf{c}) \langle \mathbf{c}, \psi(\mathbf{a}) \rangle \\ &= \langle \mathbf{b}, \chi_m \rangle^{-1} \langle \mathbf{b}, \psi(\mathbf{a}) \rangle F_\psi g(\mathbf{a}),\end{aligned}$$

completing the proof of the theorem.

Choosing $\mathbf{c}_m \in A$ such that

$$\langle \mathbf{b}, \chi_m \rangle = \langle \mathbf{b}, \psi(\mathbf{c}_m) \rangle, \quad 0 \le m < M,$$

we have that $\{\mathbf{c}_m : 0 \le m < M\}$ is a complete system of coset representatives for $A/B^\perp$ and

$$\mathbf{c}_m + B^\perp = \{\mathbf{a} \in A : \langle \mathbf{b}, \chi_m \rangle = \langle \mathbf{b}, \psi(\mathbf{a}) \rangle, \quad \text{for all } \mathbf{b} \in B\}.$$

By the theorem, $g \in \mathrm{Per}_B(m)$ implies that $F_\psi g$ vanishes off of the $B^\perp$ coset $\mathbf{c}_m + B^\perp$. The next result computes $F_\psi g$ on $B^\perp$ cosets.

Theorem 5.3 *For* $0 \le m < M$, *if* $g \in \mathrm{Per}_B(m)$ *and* $h \in L(A/B)$ *is defined by*

$$h(\mathbf{a}_l) = \langle \mathbf{a}_l, \psi(\mathbf{c}_m)\rangle g(\mathbf{a}_l), \qquad 0 \le l < L, \tag{5.3}$$

then

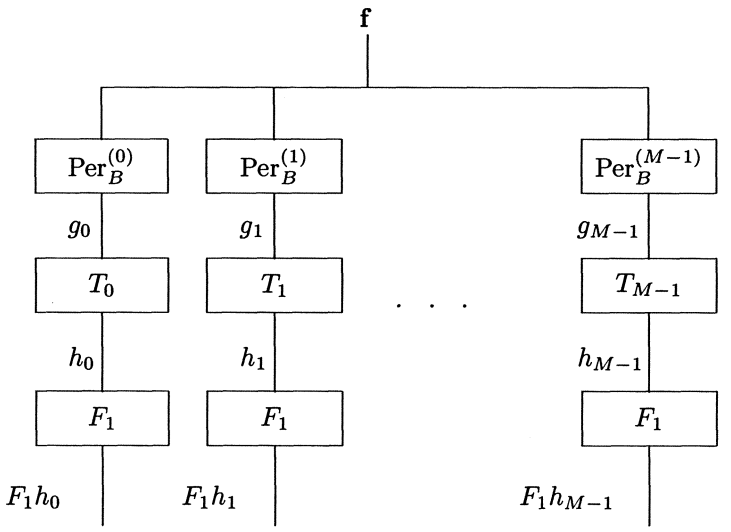
$$F_\psi g(\mathbf{c}_m + \mathbf{b}^\perp) = MF_1 h(\mathbf{b}^\perp), \qquad \mathbf{b}^\perp \in B. \tag{5.4}$$

Proof For $\mathbf{b}^\perp \in B^\perp$,

$$\begin{aligned} F_\psi g(\mathbf{c}_m + \mathbf{b}^\perp) &= \sum_{l=0}^{L-1} \sum_{\mathbf{b}\in B} g(\mathbf{a}_l + \mathbf{b})\langle \mathbf{a}_l + \mathbf{b}, \psi(\mathbf{c}_m + \mathbf{b}^\perp)\rangle \\ &= \sum_{l=0}^{L-1} g(\mathbf{a}_l)\langle \mathbf{a}_l, \psi(\mathbf{c}_m + \mathbf{b}^\perp)\rangle \sum_{\mathbf{b}\in B} \langle \mathbf{b}, \chi_m\rangle^{-1}\langle \mathbf{b}, \chi_m\rangle \\ &= M\sum_{l=0}^{L-1} g(\mathbf{a}_l)\langle \mathbf{a}_l, \psi(\mathbf{c}_m)\rangle\langle \mathbf{a}_l, \psi(\mathbf{b}^\perp)\rangle \\ &= MF_1 h(\mathbf{b}^\perp), \end{aligned}$$

which completes the proof of the theorem.

Combining the preceding results, we have the diagram in Figure 5.1 describing an algorithm computing $F_\psi f$.



$$F_\psi f(\mathbf{c}_m + b^\perp) = F_1 h_m(\mathbf{b}^\perp), \mathbf{b}^\perp \in B^\perp.$$

$$h_m(\mathbf{a}_l) = T_m(g_m)(\mathbf{a}_l) = \langle \mathbf{a}_l, \psi(\mathbf{c}_m)\rangle g_m(\mathbf{a}_l).$$

Figure 5.1 Abstract Cooley–Tukey FFT II

The abstract finite abelian group version of C–T FFT has its origins in works of Weil (1953, 1964) [8, 9], where it can be interpreted in the general setting of locally compact abelian groups as a statement about how the Fourier transform of a group can be built from the Fourier transform of a subgroup and its quotient. A recent account was given in An et al. (1991) [1]. A concrete account of the Good–Thomas FFT can be found in [4, 5]. Burrus (1981) [2] and Burrus and Eschenbacher (1981) [3] have described an in-place in-order Good–Thomas (prime factor) FFT. Temperton (1985, 1988) [6, 7] has implemented a self-sorting in-place prime factor FFT on the Cray-1.

References

[1] An, M., Gertner, I., Rofheart, M., and Tolimieri, R. (1991), "Discrete Fast Fourier Transform Algorithms: A Tutorial Survey," *Advances in Electronics and Electron Physics* **80**.

[2] Burrus, C.S. (1981), "A New Prime Factor FFT Algorithm," *Proc. 1981 IEEE Int. Conf. ASSP*, Atlanta, GA, 335–338.

[3] Burrus, C.S. and Eschenbacher, P.W. (1981), ' An In-Place In-Order Prime Factor FFT Algorithm," *IEEE Trans. ASSP* **29**, 806–817.

[4] Good, I.J. (1958), "The Interaction Algorithm and Practical Fourier Analysis," *J. Roy. Stat. Soc. Ser. B* **20**, 361–372.

[5] — (1971), "The Relationship between Two Fourier Transforms," *IEEE Trans. Comput.* **C-20**, 310–317.

[6] Temperton, C. (1985), "Implementation of a Self-Sorting In-Place Prime Factor FFT Algorithm," *J. Comput. Phys.* **58**, 283–299.

[7] — (1988), "Implementation of a Prime Factor FFT Algorithm on the Cray-1," *Parallel Comput.* **6**, 99–108.

[8] Weil, A. (1953), *L' Intégration dans les groupes et ses applications topologique*, Hermann, Paris.

[9] — (1964), "Sur certaines groupes d'operateurs unitaires," *Acta Math.* **111**, 143–211.

6
Lines

6.1 Introduction

The geometry of finite abelian groups has a long history of significant contributions to many fields in pure and applied mathematics. The arithmetic of this geometry is responsible for some of the deepest and most fruitful formulas in number theory and theta function theory [2]. It forms the basis for many results in algebraic block coding theory [1]. Most often the finite abelian groups come with additional structure, making them into modules and rings relative to which duality can be defined. Duality interleaves the additive group structure with multiplicative structure.

The importance of this geometry in the design of MDFT algorithms has a much shorter history. As we will see in the following chapters, this geometry provides powerful mathematical tools for designing new massively parallel algorithms. In this chapter, we will introduce some of the geometric concepts underlying these new algorithms with special and explicit emphasis on the central role of duality.

Lines are familiar geometric objects in Euclidean spaces admitting simple algebraic description. For example, in 2-dimensional Euclidean space, if a direction is given by the vector

$$\mathbf{a} = (a_1, a_2),$$

then the parametric equation of the line through the origin in this direction is

$$\mathbf{a}(t) = (a_1 t, a_2 t).$$

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

Lines in finite abelian groups can be defined in an analogous way, but some geometric intuition is lost. In general, such lines can double back due to the periodicity of the group addition. We can have two distinct lines intersecting in more than one point.

A *line* in a finite abelian group A is a *maximal* cyclic subgroup of A. If $\mathbf{a} \in A$ generates a line, then denote the line generated by $\mathbf{a}$ by $L(\mathbf{a})$.

Our definition of line always places the origin, or identity of A, on the line. Also, if A is a cyclic group, then the only line in A is A itself.

6.2 Examples

Example 6.1 $A = \mathbf{Z}/5 \times \mathbf{Z}/5$.

Every nonzero element in A has order 5. The lines in A coincide with the nontrivial cyclic subgroups of A. They are given in Table 6.1 (reading across).

Table 6.1 Lines in $A = \mathbf{Z}/5 \times \mathbf{Z}/5$.

0,0	1,0	2,0	3,0	4,0
0,0	1,1	2,2	3,3	4,4
0,0	1,2	2,4	3,1	4,3
0,0	1,3	2,1	3,4	4,2
0,0	1,4	2,3	3,2	4,1
0,0	0,1	0,2	0,3	0,4

There are 6 distinct lines, and except for the origin, each point of A is contained in exactly one line.

Example 6.2 $A = \mathbf{Z}/9 \times \mathbf{Z}/9$.

Table 6.2 Lines in $A = \mathbf{Z}/9 \times \mathbf{Z}/9$

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0
	1,1	2,2	3,3	4,4	5,5	6,6	7,7	8,8
	1,2	2,4	3,6	4,8	5,1	6,3	7,5	8,7
	1,3	2,6	3,0	4,3	5,6	6,0	7,3	8,6
	1,4	2,8	3,3	4,7	5,2	6,6	7,1	8,5
	1,5	2,1	3,6	4,2	5,7	6,3	7,8	8,4
	1,6	2,3	3,0	4,6	5,3	6,0	7,6	8,3
	1,7	2,5	3,3	4,1	5,8	6,6	7,4	8,2
	1,8	2,7	3,6	4,5	5,4	6,3	7,2	8,1
	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8
	3,1	6,2	0,3	3,4	6,5	0,6	3,7	6,8
	6,1	3,2	0,3	6,4	3,5	0,6	6,7	3,8

The maximum order of elements in A is 9. Lines in A are cyclic subgroups of order 9. The distinct lines are given in Table 6.2. There are 12 distinct lines. Every point is contained in some line, but lines can intersect in more than one point.

In the following examples, we will use the Chinese Remainder Theorem (CRT) to describe explicitly an isomorphism between two presentations of an abelian group A.

Example 6.3 $A = \mathbf{Z}/6 \times \mathbf{Z}/6$

By the CRT, $\mathbf{Z}/6$ is isomorphic to $\mathbf{Z}/2 \times \mathbf{Z}/3$. With idempotents $e_1 = 3$, $e_2 = 4$ we can uniquely write

$$\mathbf{a} = a_1 e_1 + a_2 e_2, \qquad \mathbf{a} \in \mathbf{Z}/6, \quad a_1 \in \mathbf{Z}/2, \quad a_2 \in \mathbf{Z}/3.$$

Then

$$A \simeq (\mathbf{Z}/2 \times \mathbf{Z}/2) \times (\mathbf{Z}/3 \times \mathbf{Z}/3),$$

and for $\mathbf{a}, \mathbf{b} \in \mathbf{Z}/6$, we can uniquely write

$$(\mathbf{a}, \mathbf{b}) = (a_1, b_1)(e_1, e_1) + (a_2, b_2)(e_2, e_2), \qquad a_1,\ b_1 \in \mathbf{Z}/2, \quad a_2,\ b_2 \in \mathbf{Z}/3.$$

Set $\mathbf{e}_1 = (e_1, e_1)$, $\mathbf{e}_2 = (e_2, e_2)$. The lines in A are given by

$$L\mathbf{e}_1 + M\mathbf{e}_2,$$

where L is a line in $\mathbf{Z}/2 \times \mathbf{Z}/2$ and M is a line in $\mathbf{Z}/3 \times \mathbf{Z}/3$.

The lines in $\mathbf{Z}/2 \times \mathbf{Z}/2$ are

$$L1: \ (0,0), \ (1,0);$$

$$L2: \ (0,0), \ (1,1);$$

$$L3: \ (0,0), \ (0,1).$$

The lines in $\mathbf{Z}/3 \times \mathbf{Z}/3$ are

$$M1: \ (0,0), \ (1,0), \ (2,0);$$

$$M2: \ (0,0), \ (1,1), \ (2,2);$$

$$M3: \ (0,0), \ (1,2), \ (2,1);$$

$$M4: \ (0,0), \ (0,1), \ (0,2).$$

There are 12 distinct lines in $\mathbf{Z}/6 \times \mathbf{Z}/6$ given by

$$\mathbf{e}_1 L_j + \mathbf{e}_2 M_k, \qquad 1 \le j \le 3,\ 1 \le k \le 4.$$

For example, the line

$$\mathbf{e}_1 L_2 + \mathbf{e}_2 M_3$$

consists of points

$$(0,0),\ (4,2),\ (2,4),\ (3,3),\ (1,5),\ (5,1).$$

There is substantial redundancy in this case, as some points can lie on as many as 4 distinct lines.

The examples so far, have equal size in both dimensions. We will now consider the mixed, or rectangular case. If

$$A = \mathbf{Z}/R \times \mathbf{Z}/S,$$

where R and S are relatively prime, then A is cyclic and has itself as its only line. The only interesting case occurs when there is a common factor.

Example 6.4 $A = \mathbf{Z}/6 \times \mathbf{Z}/15$.

Since

$$\mathbf{Z}/6 \cong \mathbf{Z}/2 \times \mathbf{Z}/3$$

with idempotents $e_1 = 3$, $e_2 = 4$ and

$$\mathbf{Z}/15 \cong \mathbf{Z}/5 \times \mathbf{Z}/3$$

with idempotents $f_1 = 6$, $f_2 = 10$, we have

$$\mathbf{Z}/6 \times \mathbf{Z}/15 \cong (\mathbf{Z}/2 \times \mathbf{Z}/5) \times (\mathbf{Z}/3 \times \mathbf{Z}/3),$$

and we can uniquely write

$$(\mathbf{a}, \mathbf{b}) = (a_1, b_1)(e_1, f_1) + (a_2, b_2)(e_2, f_2),$$

where

$$(\mathbf{a}, \mathbf{b}) \in \mathbf{Z}/6 \times \mathbf{Z}/15,\quad (a_1, b_1) \in \mathbf{Z}/2 \times \mathbf{Z}/5,\quad (a_2, b_2) \in \mathbf{Z}/3 \times \mathbf{Z}/3.$$

Set $\mathbf{e}_1 = (e_1, f_1)$, $\mathbf{e}_2 = (e_2, f_2)$. Since $\mathbf{Z}/2 \times \mathbf{Z}/5$ is a line, the lines in A have the form

$$(\mathbf{Z}/2 \times \mathbf{Z}/5)\mathbf{e}_1 + M\mathbf{e}_2,$$

where M is a line in $\mathbf{Z}/3 \times \mathbf{Z}/3$. There are 4 distinct lines in all, each having order 30.

The elements $(3,1)$, $(1,6)$, $(1,1)$, $(1,11)$ generate the 4 distinct lines. These generators are determined by placing the generator $(1,1)$ of $\mathbf{Z}/2 \times \mathbf{Z}/5$ and the generators $(0,1)$, $(1,0)$, $(1,1)$, and $(1,2)$ of the 4 distinct lines into the above formula.

6.3 Prime Case

Consider the finite abelian group

$$A = \mathbf{Z}/p \times \mathbf{Z}/p,$$

where p is a prime. The order of every nonzero element in A is p. Thus, every nonzero element $\mathbf{a} \in A$ generates a line $L(\mathbf{a})$ in A.

Consider an arbitrary nonzero point

$$\mathbf{a} = (a_1, a_2) \in A.$$

There are two possibilities. First, if $p \not| a_1$, then a_1 is invertible mod p, and we can write

$$\mathbf{a} \equiv a_1(1, a_1^{-1}a_2) \bmod p,$$

where a_1^{-1} is the inverse of a_1 mod p. It follows that

$$L(\mathbf{a}) = L(1, a_1^{-1}a_2),$$

and hence every point $\mathbf{a} \in A$ having $p \not| a_1$ lies on one of the p lines

$$L(1, j), \qquad 0 \le j < p. \tag{6.1}$$

The second possibility is that

$$\mathbf{a} = (0, a_2) = a_2(0, 1), \qquad p \not| a_2,$$

which implies that $\mathbf{a}$ lies on the line

$$L(0, 1). \tag{6.2}$$

From the theory of groups, we have that the order of any subgroup H of a group G must divide the order of G. Since lines in A have prime order, the $p + 1$ lines described in (6.1) and (6.2) are distinct, and any two have trivial intersection. We summarize these results for future reference in the next theorem.

Theorem 6.1 *If A is a finite abelian group of the form*

$$A = \mathbf{Z}/p \times \mathbf{Z}/p, \quad p \text{ prime},$$

then there are $p + 1$ distinct lines in A given by

$$\begin{aligned} &L(1, j), && 0 \le j < p, \\ &L(0, 1). \end{aligned}$$

Every nonzero point $\mathbf{a} \in A$ lies on a unique line.

Suppose

$$A = \mathbf{Z}/p \times \cdots \times \mathbf{Z}/p, \quad p \text{ prime, } N \text{ factors.}$$

Consider a nonzero point

$$\mathbf{a} = (a_1, a_2, \ldots, a_N) \in A.$$

There is a smallest integer J, $1 \leq J \leq N$, such that

$$a_1 \equiv a_2 \equiv \cdots \equiv a_{J-1} \equiv 0 \bmod p$$

and a_J is invertible mod p. We can write

$$\mathbf{a} = a_J(0, \ldots, 0, 1, a_J^{-1}a_{J+1}, \ldots, a_J^{-1}a_N) \bmod p.$$

Arguing as before, we have the following result.

Theorem 6.2 *If A is a finite abelian group of the form*

$$A = \mathbf{Z}/p \times \cdots \times \mathbf{Z}/p, \quad p \text{ prime},$$

with N factors, then there are

$$\frac{p^N - 1}{p - 1} = 1 + p + \cdots + p^{N-1}$$

distinct lines in A given by

$$\begin{aligned} L(1, j_2, \ldots, j_N), &\qquad 0 \leq j_2, \ldots, j_N < p, \\ L(0, 1, j_3, \ldots, j_N), &\qquad 0 \leq j_3, \ldots, j_N < p, \\ &\vdots \\ L(0, \ldots, 0, 1). & \end{aligned}$$

Every nonzero point $\mathbf{a} \in A$ lies on a unique line.

6.4 Prime Power Case

6.4.1 Square case

Consider the finite abelian group

$$A = \mathbf{Z}/p^R \times \mathbf{Z}/p^R, \quad p \text{ prime, } R > 1.$$

Each element $\mathbf{a} \in A$ has order a power of p with the maximum order p^R. A typical point $\mathbf{a} \in A$ has the form

$$\mathbf{a} = (a_1, a_2), \quad a_1, a_2 \in \mathbf{Z}/p^R.$$

The following theorem characterizes the order of elements $\mathbf{a} \in A$ in terms of the greatest common divisors

$$GCD(a_1, a_2, p^R).$$

Since a_1 and a_2 are defined modulo p^R, $GCD(a_1, a_2, p^R)$ is well-defined.

Theorem 6.3 *An element* $\mathbf{a} \in A$ *has order* p^R *if and only if*

$$GCD(a_1, a_2, p^R) = 1.$$

More generally, the order of an element $\mathbf{a}$ *is* p^{R-r} *if and only if*

$$GCD(a_1, a_2, p^R) = p^r.$$

Proof Consider a nonzero $\mathbf{a} \in A$ satisfying $GCD(a_1, a_2, p^R) = p^r$ with $0 \leq r < R$. We can write $\mathbf{a} = p^r\mathbf{b}$, where $p \nmid b_1$ or $p \nmid b_2$. The element $\mathbf{b}$ has order p^R. Since

$$p^{R-r}\mathbf{a} = (p^R b_1, p^R b_2) \equiv (0,0) \bmod p^R,$$

the element $\mathbf{a}$ has order p^{R-r}, proving the theorem in one direction. Conversely, if $\mathbf{a}$ has order p^{R-r}, then

$$p^{R-r}a_1 \equiv 0 \bmod p^R,$$

$$p^{R-r}a_2 \equiv 0 \bmod p^R.$$

It follows that $p^r \mid a_1$ and $p^r \mid a_2$ and r is the largest integer with this property, completing the proof of the theorem.

Corollary 6.1 $\mathbf{b} \in A$ *generates a line if and only if* $GCD(b_1, b_2, p^R) = 1$ *and every point* $\mathbf{a} \in A$ *lies on a line.*

Suppose $\mathbf{a} \in A$ has order p^R. Then $p \nmid a_1$ or $p \nmid a_2$. If $p \nmid a_1$, then a_1 is invertible modulo p^R, and we can write

$$\mathbf{a} = a_1(1, a_1^{-1}a_2) \bmod p^R.$$

The point $\mathbf{a}$ lies on one of the lines

$$L(1, j), \quad 0 \leq j < p^R. \tag{6.3}$$

On the other hand, if $p \mid a_1$ and $p \nmid a_2$, then we can write

$$\mathbf{a} = a_2(a_2^{-1}a_1, 1) \bmod p^R,$$

and the point $\mathbf{a}$ lies on one of the lines

$$L(pk, 1), \quad 0 \leq k < p^{R-1}. \tag{6.4}$$

The lines described in (6.3) and (6.4) are distinct, and every line in A is of this form, leading to the next result.

Theorem 6.4 *The finite abelian group*

$$A = \mathbf{Z}/p^R \times \mathbf{Z}/p^R, \quad p \text{ prime}, \quad R > 1,$$

has

$$p^R + p^{R-1}$$

distinct lines given by

$$\begin{aligned} &L(1,j), && 0 \le j < p^R, \\ &L(pk,1), && 0 \le k < p^{R-1}. \end{aligned}$$

For an integer j, define $\nu_p(j)$ to be the highest power of p dividing j. $\nu_p(j) \ge 0$ with $\nu_p(j) = 0$ if and only if p does not divide j. Distinct lines can have nontrivial intersection. First, consider

$$L(1,j) \cap L(1,k), \quad 0 \le j,k < p^R. \tag{6.5}$$

Set $r = \nu_p(k-j)$. The intersection (6.5) contains the point

$$(p^{R-r}, p^{R-r}j) \equiv (p^{R-r}, p^{R-r}k) \bmod p^R,$$

and hence the subgroup of order p^r

$$gp(p^{R-r}, p^{R-r}j).$$

We will show that

$$L(1,j) \cap L(1,k) = gp(p^{R-r}, p^{R-r}j).$$

A typical point $\mathbf{a} \in L(1,j) \cap L(1,k)$ satisfies

$$\mathbf{a} = (a_1, a_1 j) \equiv (a_1, a_1 k) \bmod p^R,$$

which by the defining condition for r implies $p^{R-r} \mid a_1$. Write $a_1 = b_1 p^{R-r}$. Then

$$\mathbf{a} = b_1(p^{R-r}, p^{R-r}j) \in gp(p^{R-r}, p^{R-r}j),$$

proving the claim. Applying similar arguments, we have the following result.

Theorem 6.5 *For* $0 \le k, j < p^R$,

$$L(1,j) \cap L(1,k) = gp(p^{R-r}, p^{R-r}j), \quad r = \nu_p(k-j).$$

For $0 \le j < p^R$, $0 \le k < p^{R-1}$,

$$L(1,j) \cap L(pk,1) = (0).$$

For $0 \le j, k < p^{R-1}$,

$$L(pk,1) \cap L(pj,1) = gp(p^{R-r}k, p^{R-r-1}), \quad r = \nu_p(k-j).$$

Example 6.5 Suppose $R = 2$. Then $A = \mathbf{Z}/p^2 \times \mathbf{Z}/p^2$ has $p^2 + p$ distinct lines. p^2 lines are given by

$$\begin{array}{cccc} L(1,0) & L(1,p) & \cdots & L(1,(p-1)p) \\ L(1,1) & L(1,p+1) & \cdots & L(1,p^2-p+1) \\ \vdots & \vdots & & \vdots \\ L(1,p-1) & L(1,2p-1) & \cdots & L(1,p^2-1). \end{array}$$

Any two distinct lines in the same column have trivial intersection, while any two distinct lines in the j-th row have intersection $gp(p,pj)$, $0 \le j < p$.

The remaining p lines are given by

$$L(0,1), \quad L(p,1), \quad \ldots, \quad L((p-1)p,1).$$

Any two distinct lines in this collection have intersection $gp(0,p)$.

Set

$$m(\mathbf{a}) = \text{the number of lines containing } \mathbf{a} \in A.$$

We call $m(\mathbf{a})$ the *multiplicity* of the point $\mathbf{a}$. Consider $\mathbf{a} \in A$ having order p^{R-r}. By Theorem 6.3, $\mathbf{a} = p^r\mathbf{b}$, where $p \nmid b_1$ or $p \nmid b_2$. If $p \nmid b_1$, then

$$\mathbf{a} \in L(1,j), \qquad 0 \le j < p^R,$$

if and only if

$$p^r b_1 j \equiv p^r b_2 \bmod p^R,$$

which, since b_1 is invertible mod p, is equivalent to

$$j \equiv b_1^{-1} b_2 \bmod p^{R-r}, \qquad 0 \le j < p^R.$$

There are p^r solutions for j,

$$j = j_0 + mp^{R-r}, \qquad 0 \le m < p^r,$$

where $0 \le j_0 < p^{R-r}$ and $j_0 \equiv b_1^{-1} b_2 \bmod p^{R-r}$. Thus, the point $\mathbf{a}$ of order p^{R-r} lies on exactly p^r lines in A.

If $p \mid b_1$ and $p \nmid b_2$, then

$$\mathbf{a} \in L(kp,1), \qquad 0 \le k < p^{R-1},$$

if and only if

$$p^r b_1 \equiv p^r b_2 kp \bmod p^R. \tag{6.6}$$

Write $b_1 = c_1 p$. Then (6.6) is equivalent to

$$k \equiv c_1 b_2^{-1} \bmod p^{R-r-1}, \qquad 0 \le k < p^{R-1}.$$

There are p^r solutions for k,

$$k = k_0 + np^{R-r-1}, \qquad 0 \le n < p^r,$$

where $0 \le k_0 < p^{R-r-1}$ and $k_0 \equiv c_1 b_2^{-1} \bmod p^{R-r-1}$. Thus, again the point $\mathbf{a}$ of order p^{R-r} lies on exactly p^r lines in A.

We summarize the discussion in the next result.

Theorem 6.6 *An element $\mathbf{a} \in \mathbf{Z}/p^R \times \mathbf{Z}/p^R$ having order p^{R-r} lies on exactly p^r distinct lines in A. Write $\mathbf{a} = p^r\mathbf{b}$, $p \nmid b_1$ or $p \nmid b_2$.*

1. *$p \nmid b_1$; then*

$$\mathbf{a} \in L(1, j), \qquad j \equiv b_1^{-1} b_2 \bmod p^{R-r}, \quad 0 \le j < p^R.$$

2. *$p \mid b_1 = pc_1$ and $p \nmid b_2$; then*

$$\mathbf{a} \in L(pk, 1), \qquad k \equiv c_1 b_2^{-1} \bmod p^{R-r-1}, \quad 0 \le k < p^{R-1}.$$

In particular,

$$m(\mathbf{a}) = \frac{p^R}{GCD(a_1, a_2, p^R)}.$$

A list of distinct lines in $\mathbf{Z}/p \times \mathbf{Z}/p$ and $\mathbf{Z}/p^R \times \mathbf{Z}/p^R$ are given in Tables 6.3 and 6.4.

Table 6.3 Number of lines

A	Number of Lines	Lines
$\mathbf{Z}/p \times \mathbf{Z}/p$	$p+1$	$L(1,j),\ 0 \le j < p$ $L(0,1)$
$\mathbf{Z}/p^R \times \mathbf{Z}/p^R$	$p^R + p^{R-1}$	$L(1,j),\ 0 \le j < p^R$ $L(pk,1),\ 0 \le k < p^{R-1}$

Table 6.4 Intersection of lines, $j \equiv k \bmod p^r$

A	Intersection
$\mathbf{Z}/p \times \mathbf{Z}/p$	Trivial intersection
$\mathbf{Z}/p^R \times \mathbf{Z}/p^R$	$L(1,j) \cap L(1,k) = gp(p^{R-r}, p^{R-r}j)$, $L(pk,1) \cap L(pj,1) = gp(p^{R-r}, p^{R-r-1})$, $L(1,j) \cap L(pk,1) = (0)$

The general prime power case

$$A = \mathbf{Z}/p^R \times \cdots \times \mathbf{Z}/p^R, \quad p \text{ prime, } N \text{ factors,}$$

can be analyzed by the same arguments that served in the case of two cyclic factors. We will simply describe the results.

A point $\mathbf{b} \in A$ generates a line if and only if

$$GCD(b_1, \ldots, b_N, p^R) = 1.$$

Every point $\mathbf{a} \in A$ lies on a line. In fact, if $GCD(a_1, \ldots, a_N, p^R) = p^r$ then $\mathbf{a} = p^r\mathbf{b}$, where $\mathbf{b}$ generates a line. We organize the lines of A into the following distinct subcollections L_M, $1 \leq M \leq N$:

$$Ł_1 = \{L(\mathbf{j}) : j_1 = 1\}.$$

For $1 < M \leq N$,

$$Ł_M = \{L(\mathbf{j}) : j_M = 1 \text{ and } p \mid j_1, \ldots, j_{M-1}\}.$$

The number of lines in $Ł_M$, $1 \leq M \leq N$, is

$$(p^R)^{N-M}(p^{R-1})^{M-1}.$$

Summing over $1 \leq M \leq N$, the number of lines in A is

$$(p^{R-1})^{N-1}\left(\frac{p^N - 1}{p - 1}\right).$$

Lines taken from different subcollections $Ł_{M_1}$ and $Ł_{M_2}$, $M_1 \neq M_2$, have trivial intersection, but two lines in $Ł_M$ can intersect. Consider two lines in $Ł_M$, $L(\mathbf{j})$ and $L(\mathbf{k})$. Set

$$r = \min\{\nu_p(j_l - k_l) : 1 \leq l \leq N\}.$$

Then

$$L(\mathbf{j}) \cap L(\mathbf{k}) = gp(p^{R-r}\mathbf{j}).$$

6.4.2 Rectangular case

Consider the finite abelian group

$$A = \mathbf{Z}/p^S \times \mathbf{Z}/p^R.$$

We assume $S > R$.

The order of lines in A can vary. The elements $(1,0)$ and $(0,1)$ generate lines but have orders p^S and p^R.

Theorem 6.7 *An element* $\mathbf{x} \in A$ *generates a line if and only if*

$$GCD(x_1, x_2, p) = 1.$$

Proof Suppose $\mathbf{x}$ generates a line. If $p \mid x_1$ and $p \mid x_2$, we can write

$$\mathbf{x} = p^r \mathbf{y}, \qquad p \nmid y_1 \text{ or } p \nmid y_2,$$

for some integer $r > 0$. Since $\mathbf{x} \in gp(\mathbf{y})$ and $\mathbf{x}$ generates a line, we have $L(\mathbf{x}) = L(\mathbf{y})$ and $p \mid y_1$ and $p \mid y_2$, a contradiction, proving the theorem one way.

If $GCD(x_1, x_2, p) = 1$ and $\mathbf{x} \in gp(\mathbf{y})$ for some $\mathbf{y} \in A$, then

$$x_1 \equiv n y_1 \bmod p^S,$$

$$x_2 \equiv n y_2 \bmod p^R,$$

for some integer $n > 0$, $p \nmid n$, and there exists an integer m such that

$$mn \equiv 1 \bmod p^S.$$

It follows that

$$y_1 \equiv m x_1 \bmod p^S,$$
$$y_2 \equiv m x_2 \bmod p^R,$$

proving that $gp(\mathbf{x})$ is maximal and $\mathbf{x}$ generates a line. This completes the proof of the theorem.

Arguing as before, we have that every line in A has the form

$$L(1, u), \qquad 0 \le u < p^R, \tag{6.7}$$
$$L(pv, 1), \qquad 0 \le v < p^{S-1}. \tag{6.8}$$

However, (6.7) and (6.8) do not represent distinct lines. Define the subcollections,

$$\begin{aligned}
\text{Ł}_0 &: \quad L(1,u), && 0 \le u < p^R; \\
\text{Ł}_1 &: \quad L(pv,1), && 0 < v < p^{S-1},\ p \nmid v; \\
&\vdots \\
\text{Ł}_r &: \quad L(p^r v,1), && 0 < v < p^{S-r},\ p \nmid v; \\
&\vdots \\
\text{Ł}_{S-1} &: \quad L(p^{S-1}v,1), && 0 < v < p,\ p \nmid v; \\
\text{Ł}_S &: \quad L(0,1).
\end{aligned}$$

There are p^R distinct lines in Ł$_0$, and they have order p^S. If

$$S - r \ge R,$$

then the lines in Ł_r have order p^{S-r}, while if

$$S - r < R,$$

then the lines in Ł_r have order p^R. Any two lines taken from different subcollections are distinct.

Denote by $\varphi(N)$ the number of integers $0 \leq n < N$ such that n and N are relatively prime. φ is called the *Euler function* and satisfies the multiplicative condition

$$\varphi(N_1 N_2) = \varphi(N_1)\varphi(N_2)$$

whenever N_1 and N_2 are relatively prime. We have

$$\varphi(p^R) = (p-1)p^{R-1},$$

since n and p^R are relatively prime if and only if $p \nmid n$.

Consider the lines in Ł_r. If

$$0 \leq u, v < p^{S-r}, \qquad p \nmid u \text{ and } p \nmid v,$$

then

$$L(p^r u, 1) = L(p^r v, 1)$$

implies that there exists $m \equiv 1 \bmod p^R$ such that

$$mu \equiv v \bmod p^{S-r}.$$

If $S - r < R$, then $m \equiv 1 \bmod p^{S-r}$ and

$$u \equiv v \bmod p^{S-r}.$$

The condition $S - r < R$ implies Ł_r consists of $\varphi(p^{S-r})$ distinct lines. If $S - r > R$, then $L(p^r u, 1)$ has order p^{S-r} and the number of m, $0 \leq m < p^{S-r}$ satisfying $m \equiv 1 \bmod p^R$ is p^{S-R-r}. As we run over such m, the corresponding $mu \bmod p^{S-r}$ are distinct. Thus, if $S - r \geq R$, Ł_r has p^{S-R-r} repetitions. Ł_r, in this case, has

$$\frac{\varphi(p^{S-r})}{p^{S-R-r}}$$

distinct lines. We organize this discussion as follows:

Theorem 6.8 *Ł_r has*

$$\varphi(p^{S-r}) = (p-1)p^{S-r-1}$$

distinct lines whenever $S - r < R$, and

$$\frac{\varphi(p^{S-r})}{p^{S-R-r}} = (p-1)p^{R-1}$$

distinct lines whenever $S - r \geq R$. The number of lines in

$$\mathbf{Z}/p^S \times \mathbf{Z}/p^R, \qquad S > R,$$

is $p^R + p^{R-1} + (S-R)(p-1)p^{R-1}$.

6.5 General Square Case

The cases considered in the preceding two sections form the building blocks of the general case. The primary factorization plays a major role in extending these special cases. We will first describe how lines in the factors of a direct product representation produce lines in the direct product, under the condition that the orders of the factors are pairwise relatively prime.

Consider any finite abelian group A having the form

$$A = A_1 \times A_2,$$

where A_1 and A_2 are finite abelian groups having relatively prime orders. Take $\mathbf{a} \in A$ and write $\mathbf{a} = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)})$, $\mathbf{a}^{(1)} \in A_1$ and $\mathbf{a}^{(2)} \in A_2$. Since A_1 and A_2 have relatively prime orders, $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$ have relatively prime orders in A_1 and A_2. By the CRT, the direct product

$$gp(\mathbf{a}^{(1)}) \times gp(\mathbf{a}^{(2)})$$

is a cyclic group.

Theorem 6.9 *If A is a finite abelian group of the form*

$$A = A_1 \times A_2,$$

where A_1 and A_2 have relatively prime orders and $\mathbf{a} \in A$, then

$$gp(\mathbf{a}) = gp(\mathbf{a}^{(1)}) \times gp(\mathbf{a}^{(2)}).$$

Proof Let N_1 and N_2 be the orders of A_1 and A_2, and let $\{e_1, e_2\}$ be the complete system of idempotents for the primary factorization $N = N_1 N_2$.

Since $N_1 \mathbf{a}^{(1)} = 0$ in A_1 and $e_1 \equiv 1 \bmod N_1$, we have $e_1 \mathbf{a}^{(1)} = \mathbf{a}^{(1)}$ in A_1. Since $N_2 \mathbf{a}^{(2)} = 0$ in A_2 and $e_1 \equiv 0 \bmod N_2$, we have $e_1 \mathbf{a}^{(2)} = 0$ in A_2. It follows that $e_1(\mathbf{a}^{(1)}, \mathbf{a}^{(2)}) = (\mathbf{a}^{(1)}, 0)$ in A, and hence

$$(\mathbf{a}^{(1)}, 0) \in gp(\mathbf{a}).$$

The same arguments show that $(0, \mathbf{a}^{(2)}) \in gp(\mathbf{a})$, proving

$$gp(\mathbf{a}^{(1)}) \times gp(\mathbf{a}^{(2)}) \subset gp(\mathbf{a}).$$

Since $\mathbf{a} \in gp(\mathbf{a}^{(1)}) \times gp(\mathbf{a}^{(2)})$ is obvious, we have completed the proof of the theorem.

Theorem 6.10 *If A is a finite abelian group of the form*

$$A = A_1 \times A_2,$$

where the orders of A_1 and A_2 are relatively prime, then every line L in A has a unique representation of the form

$$L = L_1 \times L_2,$$

where L_1 and L_2 are lines in A_1 and A_2.

Proof Suppose $\mathbf{a} \in A$ generates a line. By the previous theorem,

$$L(\mathbf{a}) = gp(\mathbf{a}^{(1)}) \times gp(\mathbf{a}^{(2)}).$$

If $gp(\mathbf{a}^{(1)})$ is strictly contained in a line $L_1 \in A_1$ then $L(\mathbf{a})$ is strictly contained in the cyclic group $L_1 \times gp(\mathbf{a}^{(2)})$ contradicting the maximality of $L(\mathbf{a})$. It follows that $gp(\mathbf{a}^{(1)})$ is a line in A_1. In the same way $gp(\mathbf{a}^{(2)})$ is a line in A_2.

Conversely, if L_1 is a line in A_1 and L_2 is a line A_2, then by the CRT $L = L_1 \times L_2$ is a cyclic subgroup of A. Arguing as above, L is maximal, completing the proof of the theorem.

Several results immediately follow.

Corollary 6.2 *The number of lines in A is equal to the product of the number of lines in A_1 with the number of lines in A_2.*

Corollary 6.3 *If $L = L_1 \times L_2$ and $L' = L_1' \times L_2'$ are lines in A with L_1, L_1' in A_1 and L_2, L_2' in A_2, then*

$$L \cap L' = (L_1 \cap L_1') \times (L_2 \cap L_2').$$

Corollary 6.4 *The multiplicity of a point $\mathbf{a} \in A$ is equal to the product of the multiplicity of the point $\mathbf{a}^{(1)} \in A_1$ with the multiplicity of the point $\mathbf{a}^{(2)} \in A_2$.*

These results extend in a straightforward way to direct products of any number of factors as long as the orders of the factors are pairwise relatively prime. In particular, they apply to the primary factorization of any finite abelian group. Consider the case

$$A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M, \qquad N \text{ factors.}$$

Write $\mathbf{a} = (a_1, \ldots, a_N)$, $a_n \in \mathbf{Z}/M$, $1 \leq n \leq N$. The primary factors of A have the form

$$A_p = \mathbf{Z}/p^R \times \cdots \times \mathbf{Z}/p^R, \qquad N \text{ factors,}$$

where $p \mid M$ and p^R is the highest power of p dividing M. The primary factorization of A will be denoted by

$$A = \prod_{p|M} A_p.$$

Results proved in the previous sections apply to the primary factors and can be extended to A by Theorem 6.10. We state as corollaries some of the immediate consequences.

Corollary 6.5 *A point $\mathbf{a} \in A$ generates a line in A if and only if*

$$GCD(a_1, \ldots, a_N, M) = 1.$$

Corollary 6.6 *Every line L in A can be written uniquely as the direct product $L = \prod_{p|M} L_p$, where L_p is a line in A_p.*

Corollary 6.7 *The number of lines in A is the product of the number of lines in the primary factors of A.*

We will describe the special case

$$A = \mathbf{Z}/M \times \mathbf{Z}/M, \quad M = p^R q^S, \quad p,\ q \ \text{ distinct primes},$$

in detail. The primary factorization is

$$A = A_1 \times A_2,$$

where $A_1 = \mathbf{Z}/p^R \times \mathbf{Z}/p^R$ and $A_2 = \mathbf{Z}/q^S \times \mathbf{Z}/q^S$. The primary factors A_1 and A_2 have $p^R + p^{R-1}$ and $q^S + q^{S-1}$ distinct lines, respectively.

A has

$$(p^R + p^{R-1})(q^S + q^{S-1})$$

distinct lines. Relative to the primary factorization, the distinct lines in A are given by

$$L_1(1,u) \times L_2(1,v), \qquad 0 \le u < p^R,\ 0 \le v < q^S, \tag{6.9}$$

$$L_1(1,u) \times L_2(qv,1), \qquad 0 \le u < p^R,\ 0 \le v < q^{S-1}, \tag{6.10}$$

$$L_1(pu,1) \times L_2(1,v), \qquad 0 \le u < p^{R-1},\ 0 \le v < q^S, \tag{6.11}$$

$$L_1(pu,1) \times L_2(qv,1), \qquad 0 \le u < p^{R-1},\ 0 \le v < q^{S-1}, \tag{6.12}$$

where L_1 denotes lines in the p-primary factor and L_2 denotes lines in the q-primary factor. The structure formulas established in the previous section can be applied to determine structure formulas in this case. In particular, nontrivial intersections can occur only for pairs of lines within each of the four subcollections in (6.9)–(6.12).

Denote the complete system of idempotents for the primary factorization

$$M = p^R q^S$$

by $\{e_1, e_2\}$. We have

$$\mathbf{Z}/M \times \mathbf{Z}/M \cong (\mathbf{Z}/p^R \times \mathbf{Z}/p^R) \times (\mathbf{Z}/q^S \times \mathbf{Z}/q^S),$$

and we can uniquely write $(\mathbf{a}, \mathbf{b}) \in \mathbf{Z}/M \times \mathbf{Z}/M$ as

$$(\mathbf{a}, \mathbf{b}) = (a_1, b_1)(e_1, e_1) + (a_2, b_2)(e_2, e_2), \quad a_1,\ b_1 \in \mathbf{Z}/p^R,\ a_2,\ b_2 \in \mathbf{Z}/q^S.$$

Set $\mathbf{e}_1 = (e_1, e_1)$ and $\mathbf{e}_2 = (e_2, e_2)$. The lines in $\mathbf{Z}/M \times \mathbf{Z}/M$ are all of the form

$$L_1 \mathbf{e}_1 + L_2 \mathbf{e}_2,$$

where L_1 is a line in $\mathbf{Z}/p^R \times \mathbf{Z}/p^R$ and L_2 is a line in $\mathbf{Z}/q^S \times \mathbf{Z}/q^S$.

Consider lines $L_1 = L_1(\mathbf{v}_1)$ and $L_2 = L_2(\mathbf{v}_2)$ in A_1 and A_2. The direct product $L_1 \times L_2$ is isomorphic to the line L in A given by

$$L = L(\mathbf{v}_1\mathbf{e}_1 + \mathbf{v}_2\mathbf{e}_2).$$

For example, if

$$L_1 = L_1(1, j) \subset A_1, \quad 0 \le j < p^R, \tag{6.13}$$

$$L_2 = L_2(1, k) \subset A_2, \quad 0 \le k < q^S, \tag{6.14}$$

then

$$L = L(e_1 + e_2, je_1 + ke_2) = L(1, je_1 + ke_2).$$

Since as j runs over $0 \le j < p^R$ and k runs over $0 \le k < q^S$, $n = je_1 + ke_2$ runs over $0 \le n < M$, the direct products of lines in (6.13) with lines in (6.14) correspond to lines in A of the form $L(1, n)$, $0 \le n < M$. Similar arguments in the other three cases lead to the next result.

Theorem 6.11 *The lines in*

$$A = \mathbf{Z}/M \times \mathbf{Z}/M, \quad M = p^R q^S,$$

are given by

$$\begin{aligned} L(1, n), &\quad 0 \le n < M, \\ L(pqm, 1), &\quad 0 \le m < p^{R-1}q^{S-1}, \\ L(e_2 + jpe_1, e_1 + ke_2), &\quad 0 \le j < p^{R-1}, \quad 0 \le k < q^S, \\ L(e_1 + jqe_2, e_2 + ke_1), &\quad 0 \le j < q^{S-1}, \quad 0 \le k < p^R, \end{aligned}$$

where $\{e_1, e_2\}$ *is the complete system of idempotents for the primary factorization* $M = p^R q^S$.

These results can be extended to an abelian group A having any number of direct product factors $\mathbf{Z}/M$, where M is an arbitrary positive integer. In particular, if M is the product of three distinct prime factors, then the number of lines in the abelian group $A = \mathbf{Z}/M \times \mathbf{Z}/M$ is given by the formula

$$M\left(1 + \frac{1}{p_1} + \frac{1}{p_2} + \frac{1}{p_3} + \frac{1}{p_1p_2} + \frac{1}{p_1p_3} + \frac{1}{p_2p_3} + \frac{1}{p_1p_2p_3}\right).$$

6.6 General Rectangular Case

Consider a finite abelian group

$$A = \mathbf{Z}/M_1 \times \cdots \times \mathbf{Z}/M_N.$$

We no longer assume that the "coordinate sizes" $M_1, \ldots, M_N$ are equal. However, we can still apply the CRT and deduce results about the geometry of lines in A from results about the primary factors of A.

A typical primary factor is now of the form

$$A_p = \mathbf{Z}/p^{R_1} \times \quad \cdots \quad \times \quad \mathbf{Z}/p^{R_N}.$$

where p divides some M_n, $1 \leq n \leq N$, and p^{R_n} is the highest power of p dividing M_n. We can have $R_n = 0$ for some $1 \leq n \leq N$.

Theorem 6.12 *A point* $\mathbf{x} \in A$ *generates a line if and only if*

$$GCD(x_1, \ldots, x_N, D) = 1,$$

where $D = GCD(M_1, \ldots, M_N)$.

Theorem 6.10 can be used to describe the lines in A in terms of the lines in the primary factors A_p. The case $N = 2$ was covered in Section 6.4.2. The general case can be found in [3].

The following special case is often useful in applications. Assume that

$$M_1 = M_1'M, \qquad M_2 = M_2'M,$$

where the integers M_1', M_2', and M are pairwise relatively prime. Then by the CRT,

$$A \simeq (\mathbf{Z}/M_1' \times \mathbf{Z}/M_2') \times (\mathbf{Z}/M \times \mathbf{Z}/M).$$

The group $A_1 = \mathbf{Z}/M_1' \times \mathbf{Z}/M_2'$ is cyclic, since M_1' and M_2' are relatively prime. The group $A_2 = \mathbf{Z}/M \times \mathbf{Z}/M$ is of the type considered in the previous section and the orders of A_1 and A_2 are relatively prime, so we can apply Theorem 6.10. Thus every line L in A can be written in the form

$$L = A_1 \times L_2,$$

where L_2 is a line in A_2. It follows that the number of lines in A is equal to the number of lines in A_2. Results about intersections and multiplicities in A_2 directly extend to the corresponding results in A.

References

[1] Cameron, P.T. and Van Lint, J.H. (1980), *Graphs, Codes and Designs, London Mathematical Society Lecture Note Series* **43**, Cambridge University Press, Cambridge, Great Britain.

[2] Rauch, H.E. and Farka, H.M. (1974), *Theta Functions with Applications to Riemann Surfaces*, Williams and Wilkins, Baltimore.

[3] Vulis, M. and Tsai, D. (1990), "Computing Discrete Fourier Transform on a Rectangular Data Array," *IEEE Trans. ASSP* **ASSP-38** (2).

Problems

1. Determine the number of lines in $\mathbf{Z}/114$.
2. Describe the lines in $\mathbf{Z}/24$.
3. Describe the lines in $\mathbf{Z}/15 \times \mathbf{Z}/15$.
4. Show that every nonzero point in $\mathbf{Z}/7 \times \mathbf{Z}/7$ generates a line.
5. Describe the lines in $\mathbf{Z}/8 \times \mathbf{Z}/12$ containing the point (2,4).
6. Determine the multiplicity of points in $\mathbf{Z}/6 \times \mathbf{Z}/6$.
7. Consider $\mathbf{Z}/27 \times \mathbf{Z}/27$. Show that $L(1,9) \cap L(6,1) = (0)$.
8. Find the intersection of $L(1,6) \cap L(1,9)$ in $\mathbf{Z}/27 \times \mathbf{Z}/27$.
9. Determine the number of lines in $\mathbf{Z}/5 \times \mathbf{Z}/5 \times \mathbf{Z}/5$.
10. Determine the number of lines in $\mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$.

7

Duality of Lines and Planes

7.1 Automorphism Group

A mapping $\alpha : A \to A$ of a finite abelian group A is called a *homomorphism* of A if it satisfies

$$\alpha(\mathbf{a} + \mathbf{b}) = \alpha(\mathbf{a}) + \alpha(\mathbf{b}), \qquad \mathbf{a}, \mathbf{b} \in A.$$

A homomorphism α of A maps subgroups of A to subgroups of A and cyclic subgroups of A to cyclic subgroups of A. A homomorphism α of A is called an *automorphism* of A whenever α is invertible, i.e., there exists a homomorphism β of A such that

$$\alpha\beta = \beta\alpha = 1_A,$$

where 1_A denotes the identity mapping of A. In this case, β is uniquely determined and will be denoted by α^{-1}. An automorphism α of A maps lines in A onto lines in A.

Composition of two automorphisms α and β of A defines a product automorphism $\alpha\beta$,

$$(\alpha\beta)(\mathbf{a}) = \alpha(\beta(\mathbf{a})), \qquad \mathbf{a} \in A,$$

which makes the set of all automorphisms of A into a group, denoted by $\mathrm{Aut}(A)$. The identity of $\mathrm{Aut}(A)$ is 1_A, and the inverse of $\alpha \in \mathrm{Aut}(A)$ is α^{-1}. In general, $\mathrm{Aut}(A)$ is not abelian, but it is finite.

Theorem 7.1 *If B is a primary factor of A, then for all $\alpha \in \mathrm{Aut}(A)$,*

$$\alpha(B) = B.$$

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

Proof Suppose B is the subgroup of all elements in A having order a power of a prime divisor p of N. If $\alpha \in \mathrm{Aut}(A)$, then the order of $\alpha(\mathbf{a})$ equals the order of $\mathbf{a}$ for all $\mathbf{a} \in A$. Thus $\alpha(\mathbf{b})$ has order a power of p for all $\mathbf{b} \in B$ and $\alpha(B) \subset B$. Arguing in the same way with α^{-1} completes the proof of the theorem.

Consider the primary factorization of A

$$A = A_1 \times \cdots \times A_R. \tag{7.1}$$

Let $\alpha \in \mathrm{Aut}(A)$. By Theorem 7.1, the restriction α_r of α to the primary factor A_r is an automorphism of A_r. The automorphism α is completely determined by its primary factor restrictions $\alpha_1, \ldots, \alpha_R$. In fact,

$$\alpha(\mathbf{a}) = \left(\alpha_1(\mathbf{a}^{(1)}), \ldots, \alpha_R(\mathbf{a}^{(R)})\right), \quad \mathbf{a}^{(r)} \in A_r, \quad 1 \le r \le R,$$

and we have

$$\mathrm{Aut}(A) = \mathrm{Aut}(A_1) \oplus \cdots \oplus \mathrm{Aut}(A_R).$$

Suppose

$$A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M \quad (N \text{ factors}).$$

We will identify every $\mathbf{a} \in A$ with the N-tuple column vector

$$\mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix}, \quad a_n \in \mathbf{Z}/M.$$

Denote by e_n the element in A all of whose components are 0 except for the n-th which is 1. If α is a homomorphism of A, we can write

$$\alpha(e_n) = \begin{bmatrix} x_{1n} \\ \vdots \\ x_{Nn} \end{bmatrix}, \quad 1 \le n \le N,$$

and form the $N \times N$ matrix

$$X(\alpha) = [\alpha(e_1) \ \cdots \ \alpha(e_N)].$$

Since α is a homomorphism,

$$\alpha(\mathbf{a}) = a_1\alpha(e_1) + \cdots + a_N\alpha(e_N),$$

and we can write

$$\alpha(\mathbf{a}) = X(\alpha)\mathbf{a}.$$

Denote the group of all $N \times N$ invertible matrices having coefficients in $\mathbf{Z}/M$ by $GL(N, \mathbf{Z}/M)$. For $\alpha \in \mathrm{Aut}(A)$, $X(\alpha) \in GL(N, \mathbf{Z}/M)$.

Theorem 7.2 *The mapping*

$$\mathrm{Aut}(A) \to GL(N, \mathbf{Z}/M)$$

taking $\alpha \in \mathrm{Aut}(A)$ *onto* $X(\alpha) \in GL(N, \mathbf{Z}/M)$ *is a group isomorphism from* $\mathrm{Aut}(A)$ *onto* $GL(N, \mathbf{Z}/M)$.

Matrices in $GL(N, \mathbf{Z}/M)$ have determinants that are invertible mod M. Denote the subgroup of matrices in $GL(N, \mathbf{Z}/M)$ having determinant 1 by $SL(N, \mathbf{Z}/M)$.

For example, the N-point cyclic shift matrix

$$S = \begin{bmatrix} 0 & 0 & \cdot & \cdot & \cdot & & 1 \\ 1 & 0 & \cdot & \cdot & \cdot & & 0 \\ 0 & 1 & & & & & \cdot \\ \cdot & & \cdot & & & & \cdot \\ \cdot & & & \cdot & & & \cdot \\ \cdot & & & & \cdot & & 0 \\ 0 & & & & & 1 & 0 \end{bmatrix} \in SL(N, \mathbf{Z}/M)$$

defines the automorphism α_S of A

$$\alpha_S(\mathbf{a}) = (a_N, a_1, \ldots, a_{N-1}).$$

Theorem 7.3 *If* $\mathbf{a} \in A$ *generates a line, then there exists an* $\alpha \in \mathrm{Aut}(A)$ *such that* $\alpha(e_1) = \mathbf{a}$.

Proof It is sufficient to prove the theorem for $M = p^R$, p a prime. Since $\mathbf{a}$ generates a line, $p \not| a_n$ for some $1 \leq n \leq N$. Without loss of generality we can assume that $p \not| a_1$. Simply apply some power of the cyclic shift automorphism to $\mathbf{a}$. Then a_1 is invertible mod p^R, and the matrix

$$\begin{bmatrix} a_1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ a_2 & a_1^{-1} & 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & 0 & 1 & & & & \\ \cdot & \cdot & & \cdot & & & \\ \cdot & \cdot & & & \cdot & & \\ & & & & & \cdot & \\ a_N & 0 & & & & & 1 \end{bmatrix} \in SL(N, \mathbf{Z}/M)$$

defines an automorphism α of A satisfying the conclusion of the theorem.

The action of $SL(N, \mathbf{Z}/M)$ on lines will play a major role in the duality theory of lines. The first result of this kind is a simple restatement of Theorem 7.3.

Corollary 7.1 *If* $A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M$, N *factors, and* L_0 *and* L *are any two lines in* A, *then there exists* $\alpha \in \mathrm{Aut}(A)$ *such that* $L = \alpha(L_0)$.

7.2 Dual of Lines

In this section, we will consider the special case of the dual of lines in the finite abelian group

$$A = \mathbf{Z}/M \times \mathbf{Z}/M \tag{7.2}$$

relative to the standard bilinear form on A,

$$\Psi(\mathbf{a}, \mathbf{b}) = a_1 b_1 + a_2 b_2, \qquad \mathbf{a},\ \mathbf{b} \in A. \tag{7.3}$$

The matrix of Ψ relative to the presentation (7.2) is I_2, the 2×2 identity matrix.

The mapping J of A

$$J(\mathbf{a}) = (-a_2, a_1)$$

is an automorphism of A satisfying

$$\Psi(J(\mathbf{a}), J(\mathbf{b})) = \Psi(\mathbf{a}, \mathbf{b}), \qquad \mathbf{a}, \mathbf{b} \in A. \tag{7.4}$$

The matrix of J is

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Condition (7.4) is equivalent to the matrix condition

$$J^t \Psi J = J^t I_2 J = I_2 = \Psi.$$

The automorphism J maps lines onto lines by the rule

$$J(L(\mathbf{a})) = L(J\mathbf{a}).$$

Theorem 7.4 *For any line L in A, $L^\perp = J(L)$.*

Proof Since Ψ is bilinear,

$$L^\perp = \{\mathbf{a} \in A : \ \Psi(\mathbf{x}, \mathbf{a}) = 0\},$$

where $\mathbf{x}$ generates the line L. From

$$\Psi(\mathbf{x}, J\mathbf{x}) = x_1(-x_2) + x_1 x_2 = 0$$

we have $J(L) \subset L^\perp$.

Conversely, if $\mathbf{a} \in L^\perp$,

$$a_1 x_1 + a_2 x_2 \equiv 0 \bmod M.$$

Since

$$GCD(x_1, x_2, M) = 1,$$

we have

$$x_1 y_1 - x_2 y_2 \equiv 1 \bmod M,$$

for some $y_1, y_2 \in \mathbf{Z}/M$. Setting

$$b_2 \equiv y_1 a_2 + y_2 a_1 \bmod M,$$

we have

$$a_1 \equiv -x_2 b_2, \qquad a_2 \equiv x_1 b_2 \bmod M.$$

Thus $\mathbf{a} = b_2(-x_2, x_1) \in J(L)$, completing the proof of the theorem.

A second proof of Theorem 7.4 that brings out the role of the action of $SL(2, \mathbf{Z}/M)$ and can be generalized to direct products of an arbitrary number of $\mathbf{Z}/M$ will now be given. The main idea is that it is easy to prove Theorem 7.4 for any "coordinate axis," say

$$L = L(e_1).$$

Direct computation shows

$$L(e_1)^{\perp} = L(e_2). \tag{7.5}$$

Theorem 7.3 can be applied to extend (7.5) to any line in A. The reasoning goes as follows.

Select $X \in SL(2, \mathbf{Z}/M)$ such that $X(e_1) = \mathbf{x}$, where $\mathbf{x}$ generates a line in A. Then $\mathbf{a} \in L(\mathbf{x})^{\perp}$ if and only if

$$\mathbf{a}^t \Psi \mathbf{x} = \mathbf{a}^t \mathbf{x} \equiv 0 \bmod M. \tag{7.6}$$

We can write (7.6) as

$$(\mathbf{x}^t \mathbf{a})^t e_1 \equiv 0 \bmod M,$$

which implies by (7.5) that

$$L(\mathbf{x})^{\perp} = X^* L(e_1)^{\perp} = X^* L(e_2),$$

where $X^* = (X^t)^{-1}$.

Since the second column of X^* is $J\mathbf{x}$, we have

$$X^*(L(e_2)) = L(J\mathbf{x}),$$

verifying once again that

$$L(\mathbf{x})^{\perp} = L(J\mathbf{x}).$$

In particular, the dual of every line in A is a line.

Example 7.1 Let $M = p$, p a prime. The lines in $A = \mathbf{Z}/p \times \mathbf{Z}/p$ are given by

$$L(1, u), \qquad 0 \leq u < p,$$

$$L(0, 1).$$

By Theorem 7.4

$$L(1,u)^{\perp} = L(-u,1),$$
$$L(0,1)^{\perp} = L(1,0).$$

Two cases can be distinguished. If $u_0^2 \equiv -1 \bmod p$, then

$$L(1,u_0)^{\perp} = L(1,u_0).$$

Such a u_0 can be found if and only if $p \equiv 1 \bmod 4$. Otherwise, if $p \equiv 3 \bmod 4$, then no solution can be found and no line is self-dual. We give two special cases in Table 7.1.

Table 7.1 Lines and their duals

$p=5$		$p=7$	
L	$L^{\perp}$	L	$L^{\perp}$
$L(1,0)$	$L(0,1)$	$L(1,0)$	$L(0,1)$
$L(1,1)$	$L(1,4)$	$L(1,1)$	$L(1,6)$
$L(1,2)$	$L(1,2)$	$L(1,2)$	$L(1,3)$
$L(1,3)$	$L(1,3)$	$L(1,3)$	$L(1,2)$
$L(1,4)$	$L(1,1)$	$L(1,4)$	$L(1,5)$
$L(0,1)$	$L(1,0)$	$L(1,5)$	$L(1,4)$
		$L(1,6)$	$L(1,1)$
		$L(0,1)$	$L(1,0)$
$2^2 \equiv -1 \bmod 5$,		$3^2 \equiv -1 \bmod 5$.	

In the composite case, say

$$A = \mathbf{Z}/M \times \mathbf{Z}/M,$$

$M = p^R q^S$, p and q distinct primes, we can use Theorem 7.4 directly or take direct products of duals of lines in the primary factors

$$A_1 = \mathbf{Z}/p^R \times \mathbf{Z}/p^R, \qquad A_2 = \mathbf{Z}/q^S \times \mathbf{Z}/q^S.$$

If $\{e_1, e_2\}$ is the complete system of idempotents for the primary factorization $M = p^R q^S$, then bilinear forms on A_1 and A_2 are defined by

$$\Psi_1(\mathbf{a}^{(1)}, \mathbf{b}^{(1)}) = \frac{e_1}{q^S}(\mathbf{a}^{(1)})^t \mathbf{b}^{(1)}, \qquad \mathbf{a}^{(1)}, \mathbf{b}^{(1)} \in A_1,$$

$$\Psi_2(\mathbf{a}^{(2)}, \mathbf{b}^{(2)}) = \frac{e_2}{p^R}(\mathbf{a}^{(2)})^t \mathbf{b}^{(2)}, \qquad \mathbf{a}^{(2)}, \mathbf{b}^{(2)} \in A_2.$$

Since the matrices of Ψ_1 and Ψ_2 are scalar matrices, Theorem 7.4 still holds in the primary factors. The direct product $\Psi_1 \times \Psi_2$ is the bilinear form Ψ given in (7.3). If L is line in A given by $L = L_1 \times L_2$, where L_1 and L_2 are lines in A_1 and A_2, then by Theorem 3.6, $L^{\perp} = L_1^{\perp} \times L_2^{\perp}$, where

duality in A_1 and A_2 is defined relative to Ψ_1 and Ψ_2 and duality in A is defined relative to $\Psi = \Psi_1 \times \Psi_2$.

Example 7.2 The lines and their duals in $\mathbf{Z}/9 \times \mathbf{Z}/9$ and in $\mathbf{Z}/6 \times \mathbf{Z}/6$ are given in Table 7.2.

Table 7.2 Lines and their duals

$\mathbf{Z}/9 \times \mathbf{Z}/9$		$\mathbf{Z}/6 \times \mathbf{Z}/6$	
L	$L^\perp$	L	$L^\perp$
$L(1,0)$	$L(0,1)$	$L(1,0)$	$L(0,1)$
$L(1,1)$	$L(1,8)$	$L(1,1)$	$L(1,5)$
$L(1,2)$	$L(1,4)$	$L(1,2)$	$L(4,1)$
$L(1,3)$	$L(6,1)$	$L(1,3)$	$L(3,1)$
$L(1,4)$	$L(1,2)$	$L(1,4)$	$L(4,5)$
$L(1,5)$	$L(1,7)$	$L(1,5)$	$L(1,1)$
$L(1,6)$	$L(3,1)$	$L(3,4)$	$L(4,3)$
$L(1,7)$	$L(1,5)$	$L(3,1)$	$L(1,3)$
$L(1,8)$	$L(1,1)$	$L(4,3)$	$L(3,4)$
$L(0,1)$	$L(1,0)$	$L(4,1)$	$L(1,2)$
$L(3,1)$	$L(1,6)$	$L(4,5)$	$L(1,4)$
$L(6,1)$	$L(1,3)$	$L(0,1)$	$L(1,0)$

7.3 Planes

An *r-dimensional plane* P in a finite abelian group A is any direct sum

$$P = L_1 \oplus \cdots \oplus L_r$$

of lines $L_1, \ldots, L_r$ in A. If $P = L_1 \oplus \cdots \oplus L_r$ is an r-dimensional plane and $L_j = L(\mathbf{a}_j)$, $1 \leq j \leq r$, then we will denote P by $P(\mathbf{a}_1, \ldots, \mathbf{a}_r)$.

Theorem 6.10 generalizes in a straightforward way to r-dimensional planes.

Theorem 7.5 *If A is a finite abelian group of the form*

$$A = A_1 \times A_2,$$

where the orders of A_1 and A_2 are relatively prime, then every r-dimensional plane P in A has a unique representation of the form $P = P_1 \times P_2$, where P_1 and P_2 are r-dimensional planes in A_1 and A_2.

Applying Theorem 7.5 to the primary factorization of the finite abelian group A, we can proceed to describe the distinct r-dimensional planes in A in terms of the distinct r-dimensional planes in the primary factors of A.

The next theorem generalizes theorem 7.5 to 2-dimensional planes in $A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M$.

Theorem 7.6 *If P is a 2-dimensional plane in $A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M$, N factors, then there exists an automorphism α of A such that*

$$P = \alpha\left(P(e_1, e_2)\right).$$

Proof Suppose again that $M = p^R$, p a prime, and that $P = P(\mathbf{a}, \mathbf{b})$. As before we can assume without loss of generality that $p \not| a_1$. If for all $1 \leq n \leq N$,

$$p|(a_1 b_n - a_n b_1),$$

then

$$(p^{R-1}a_1)\mathbf{b} = (p^{R-1}b_1)\mathbf{a} \bmod p^R,$$

which contradicts the assumption that $\mathbf{a}$ and $\mathbf{b}$ generate lines having trivial intersection. Arguing as above, with the $(N-1)$-point cyclic shift matrix we can assume that

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

has determinant z not divisible by p. The $N \times N$ matrix

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ x_2 & y_2 & 0 & 0 & & & & \cdot \\ 0 & 0 & z^{-1} & 0 & & & & \cdot \\ \cdot & \cdot & 0 & 1 & & & & \cdot \\ \cdot & \cdot & \cdot & 0 & 1 & & & \\ \cdot & \cdot & & & & \cdot & & \\ & & & & & & \cdot & 0 \\ x_N & y_N & 0 & \cdot & \cdot & \cdot & 0 & 1 \end{bmatrix}$$

defines an automorphism α of A satisfying the conclusion of the theorem.

The previous theorem extends to a plane of any dimension in $A = \mathbf{Z}/M \cdots \times \mathbf{Z}/M$.

Corollary 7.2 *If P is a 2-dimensional plane in A, then there exists an $(N-2)$-dimensional plane Q in A such that $A = P \oplus Q$.*

In applications we will require a procedure for constructing a covering set of r-dimensional planes, i.e., a collection of r-dimensional planes whose union is A. Suppose $A = A_1 \times A_2$, where we make no assumption about the orders of A_1 and A_2. We do assume that if L_1 is a line in A_1, then $L_1 \times \{0\}$ is a line in A, and if L_2 is a line in A_2, then $\{0\} \times L_2$ is a line in A. Lines in a primary factor of a finite abelian group A can never be lines in A unless A has only one primary factor. On the other hand if $A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M$ and $A = A_1 \times A_2$ where A_1 and A_2 are also direct products of $\mathbf{Z}/M$, then lines in A_1 and lines in A_2 are lines in A as described above. The following result provides one method for constructing a covering set of planes.

Theorem 7.7 *Let A be a finite abelian group of the form*

$$A = A_1 \times A_2$$

where lines in A_1 and lines in A_2 are lines in A. If L_1 and L_2 are lines in A_1 and A_2, then $P = L_1 \times L_2$ is a 2-dimensional plane in A. As L_1 runs over all lines in A_1 and L_2 runs over all lines in A_2, the collection of planes $P = L_1 \times L_2$ is a covering set of 2-dimensional planes for A.

Proof If $\mathbf{a} \in A$ and we write $\mathbf{a} = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)})$ where $\mathbf{a}^{(1)} \in A_1$ and $\mathbf{a}^{(2)} \in A_2$, then $\mathbf{a}^{(1)} \in L_1$ and $\mathbf{a}^{(2)} \in L_2$, where L_1 and L_2 are lines in A contained in A_1 and A_2. Thus $\mathbf{a} \in L_1 \times L_2$, completing the proof of the theorem.

7.4 Duality

Throughout this section, fix a finite abelian group A and a symmetric isomorphism ψ from A onto A^*,

$$\langle \mathbf{a}, \psi(\mathbf{b}) \rangle = \langle \mathbf{b}, \psi(\mathbf{a}) \rangle, \qquad \mathbf{a}, \mathbf{b} \in A.$$

Unless otherwise specified, $B^\perp$ is the dual of the subgroup B of A relative to the fixed symmetric isomorphism ψ.

For an automorphism α of A and $\mathbf{b} \in A$, the function $\psi_\alpha(\mathbf{b})$ of A defined by

$$\langle \mathbf{a}, \psi_\alpha(\mathbf{b}) \rangle = \langle \alpha(\mathbf{a}), \psi(\mathbf{b}) \rangle, \qquad \mathbf{a} \in A,$$

is a character of A, and the mapping

$$\psi_\alpha : A \to A^*$$

is an isomorphism of A onto A^*. The automorphism α^t of A defined by

$$\alpha^t = \psi^{-1}\psi_\alpha$$

is called the *transpose* of α relative to ψ. The transpose α^t satisfies

$$\langle \mathbf{a}, \psi(\alpha^t \mathbf{b}) \rangle = \langle \mathbf{a}, \psi_\alpha(\mathbf{b}) \rangle = \langle \alpha(\mathbf{a}), \psi(\mathbf{b}) \rangle, \qquad \mathbf{a}, \mathbf{b} \in A.$$

Unless otherwise specified, we assume that α^t is the transpose of α relative to the fixed symmetric isomorphism ψ.

Theorem 7.8 *For $\alpha \in \mathrm{Aut}(A)$, we have $(\alpha^t)^t = \alpha$.*

Proof For $\mathbf{a}, \mathbf{b} \in A$,

$$\begin{aligned} \langle \mathbf{a}, \psi((\alpha^t)^t(\mathbf{b})) \rangle &= \langle \alpha^t(\mathbf{a}), \psi(\mathbf{b}) \rangle \\ &= \langle \mathbf{b}, \psi(\alpha^t(\mathbf{a})) \rangle \\ &= \langle \alpha(\mathbf{b}), \psi(\mathbf{a}) \rangle \\ &= \langle \mathbf{a}, \psi(\alpha(\mathbf{b})) \rangle, \end{aligned}$$

implying $(\alpha^t)^t = \alpha$.

Suppose for the remainder of this section that

$$A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M, \ N \text{ factors.}$$

Theorem 7.9 *If P is an r-dimensional plane satisfying $P = \alpha(P(e_1, \ldots, e_r))$, for some $\alpha \in \mathrm{Aut}(A)$, then*

$$P^\perp = (\alpha^t)^{-1}(P^\perp(e_1, \ldots, e_r)).$$

Proof Take $\mathbf{a} \in P^\perp$. For all $\mathbf{y} \in P(e_1, \ldots, e_r)$, $\alpha(\mathbf{y}) \in P$ and

$$\langle \mathbf{y}, \psi(\alpha^t(\mathbf{a})) \rangle = \langle \alpha(\mathbf{y}), \psi(\mathbf{a}) \rangle = 1,$$

implying

$$\alpha^t(P^\perp) \subset P^\perp(e_1, \ldots, e_r).$$

Conversely, if $\mathbf{a} \in P^\perp(e_1, \ldots, e_r)$, then for all $\mathbf{y} \in P(e_1, \ldots, e_r)$, we have

$$\langle \alpha(\mathbf{y}), \psi((\alpha^t)^{-1}(\mathbf{a})) \rangle = \langle \mathbf{y}, \psi(\mathbf{a}) \rangle = 1,$$

implying

$$(\alpha^t)^{-1}(P^\perp(e_1, \ldots, e_r)) \subset P^\perp,$$

completing the proof.

Consider symmetric isomorphisms ψ_j, $1 \leq j < N$, of $\mathbf{Z}/M$ onto $(\mathbf{Z}/M)^*$ and form the direct product $\tilde{\psi} = \psi_1 \times \ldots \times \psi_N$. Denote by $B^\perp_{\tilde{\psi}}$ the dual of the subgroup B of A relative to $\tilde{\psi}$.

Theorem 7.10

$$P^\perp_{\tilde{\psi}}(e_1, \ldots, e_r) = P(e_{r+1}, \ldots, e_N).$$

Proof By definition, for $1 \leq j \leq r$,

$$\langle e_j, \tilde{\psi}(\mathbf{a}) \rangle = \langle 1, \psi_j(a_j) \rangle.$$

Since $\mathbf{a} \in P(e_{r+1}, \ldots, e_N)$ if and only if for $1 \leq j \leq r$

$$\langle 1, \psi_j(a_j) \rangle = 1,$$

and $\mathbf{a} \in P^\perp_{\tilde{\psi}}(e_1, \ldots, e_r)$ if and only if for $1 \leq j \leq r$

$$\langle e_j, \tilde{\psi}(\mathbf{a}) \rangle = 1,$$

the theorem follows.

For $\beta \in \mathrm{Aut}(A)$ denote by $\beta^t_{\tilde{\psi}}$ the transpose of β with respect to $\tilde{\psi}$.

Theorem 7.11

$$P^{\perp}(e_1,\ldots,e_r) = (\beta_{\tilde{\psi}}^t)^{-1}P(e_{r+1},\ldots,e_N),$$

where β is the automorphism of A defined by $\beta = \tilde{\psi}^{-1}\psi$.

Proof Take $\mathbf{a} \in P^{\perp}(e_1,\ldots,e_r)$. Since all isomorphisms are symmetric, we have for all $\mathbf{b} \in P(e_1,\ldots,e_r)$,

$$\begin{aligned} 1 &= \langle \mathbf{b}, \psi(\mathbf{a})\rangle = \langle \mathbf{a}, \psi(\mathbf{b})\rangle \\ &= \langle \mathbf{a}, \tilde{\psi}\beta(\mathbf{b})\rangle = \langle \beta(\mathbf{b}), \tilde{\psi}(\mathbf{a})\rangle \\ &= \langle \mathbf{b}, \tilde{\psi}(\beta_{\tilde{\psi}}^t(\mathbf{a}))\rangle, \end{aligned}$$

implying by the preceding theorem that

$$P^{\perp}(e_1,\ldots,e_r) \subset (\beta_{\tilde{\psi}}^t)^{-1}P_{\tilde{\psi}}^{\perp}(e_1,\ldots,e_r) = (\beta_{\tilde{\psi}}^t)^{-1}P(e_{r+1},\ldots,e_N).$$

The converse is proved by reversing the arguments.

Corollary 7.3 $P^{\perp} = (\alpha^t)^{-1}(\beta_{\tilde{\psi}}^t)^{-1}P(e_{r+1},\ldots,e_N)$.

Corollary 7.4 *If P is an r-dimensional plane in A then $P^{\perp}$ is an $(N-r)$-dimensional plane in A.*

In the following examples we take $\psi = \tilde{\psi}$ throughout and construct all transposes and duals relative to $\tilde{\psi}$. Then

$$P^{\perp}(e_1,\ldots,e_r) = P(e_{r+1},\ldots,e_N),$$

and if $P = \alpha(P(e_1,\ldots,e_r))$ then $P^{\perp} = (\alpha^t)^{-1}P(e_{r+1},\ldots,e_N)$.

Example 7.3 Suppose

$$A = \mathbf{Z}/P \times \mathbf{Z}/P \times \mathbf{Z}/P.$$

The dual of a line in A is a 2-dimensional plane in A. The lines in A are given by

$$\begin{aligned} &L(1,j_2,j_3), && 0 \le j_2, j_3 < P, \\ &L(0,1,j_3), && 0 \le j_3 < P, \\ &L(0,0,1). \end{aligned}$$

Their duals are the 2-dimensional planes

$$\begin{aligned} L(1,j_2,j_3)^{\perp} &= P((-j_3,0,1),(-j_2,1,0)) \\ &= L(-j_3,0,1) \oplus L(-j_2,1,0), \\ L(0,1,j_3)^{\perp} &= L(e_1) \oplus L(0,-j_3,1), \\ L(0,0,1)^{\perp} &= L(e_1) \oplus L(e_2). \end{aligned}$$

Since every 2-dimensional plane is the dual of a line, we have described all the 2-dimensional planes in A.

A covering set of planes can be found using Theorem 7.7. Write

$$A = (\mathbf{Z}/p \times \mathbf{Z}/p) \times \mathbf{Z}/p.$$

Applying Theorem 7.7, the set of 2-dimensional planes in A is given by

$$P = L_1 \times \mathbf{Z}/p,$$

where L_1 is a line in $\mathbf{Z}/p \times \mathbf{Z}/p$. Thus A is covered by the $p+1$ 2-dimensional planes

$$\begin{aligned} L(1,u) \times \mathbf{Z}/p &= \{(m, mu, n) : m,\ n \in \mathbf{Z}/p\}, \quad 0 \le u < p, \\ L(0,1) \times \mathbf{Z}/p &= \{(0, m, n) : m,\ n \in \mathbf{Z}/p\}. \end{aligned}$$

Example 7.4 Suppose

$$A = \mathbf{Z}/p \times \mathbf{Z}/p \times \mathbf{Z}/p \times \mathbf{Z}/p.$$

A is covered by the $(p+1)^2$ 2-dimensional planes

$$\begin{aligned} L(1,u) \times L(1,v) &= \{(m, mu, n, nv) : m, n \in \mathbf{Z}/p\}, \quad 0 \le u,\ v < p; \\ L(1,u) \times L(0,1) &= \{(m, mu, 0, n) : m,\ n \in \mathbf{Z}/p\}, \quad 0 \le u < p; \\ L(0,1) \times L(1,v) &= \{(0, m, n, nv) : m,\ n \in \mathbf{Z}/p\}, \quad 0 \le v < p; \\ L(0,1) \times L(0,1) &= \{(0, m, 0, n) : m, \quad n \in \mathbf{Z}/p\}, \end{aligned}$$

and by the $p+1$ 3-dimensional planes

$$\begin{aligned} L(1,u) \times \mathbf{Z}/p \times \mathbf{Z}/p &= \{(m, mu, n, r) : m,\ n,\ r \in \mathbf{Z}/p\}, \quad 0 \le u < p; \\ L(0,1) \times \mathbf{Z}/p \times \mathbf{Z}/p &= \{(0, m, n, r) : m,\ n,\ r \in \mathbf{Z}/p\}. \end{aligned}$$

Example 7.5 Suppose

$$A = \mathbf{Z}/p^R \times \mathbf{Z}/p^R \times \mathbf{Z}/p^R.$$

The 2-dimensional planes are given by the duals of lines.

$$L(1, j_2, j_3)^\perp = L(-j_3, 0, 1) \oplus L(-j_2, 1, 0),$$

$$L(pj_1, 1, j_3)^\perp = L(1, -pj_1, 0) \oplus L(0, -j_3, 1),$$

$$L(pj_1, pj_2, 1)^\perp = L(1, 0, -pj_1) \oplus L(0, 1, -pj_2).$$

Example 7.6 Suppose

$$A = \mathbf{Z}/p \times \mathbf{Z}/p \times \mathbf{Z}/p \times \mathbf{Z}/p.$$

The dual of a 2-dimensional plane is a 2-dimensional plane. Consider the plane

$$P = L(1, j_2, j_3, j_4) \oplus L(0, 1, k_3, k_4).$$

By direct computation,

$$P^{\perp} = L(k_3 j_2 - j_3, -k_3, 1, 0) \oplus L(k_4 j_2 - j_4, -k_4, 0, 1).$$

The results in this and the previous chapter have been known in various fields for a long time. Much of this material can be found in Jordan (1870) [4]. They were rediscovered by Vulis (1989), (1990) [5, 6], Gertner-Shamash (1987) [2], Gertner (1988) [1], and Gertner, Tolimieri (1989) [3] for application to the MDFT algorithms of the next chapter.

References

[1] Gertner, I. (1988), "A New Efficient Algorithm to Compute the Two-Dimensional Discrete Fourier Transform," *IEEE Trans. ASSP* **ASSP-36** (7), 1036–1050.

[2] Gertner, I. and Shamash, M. (1987), "VLSI Architectures for Multidimensional Fourier Transform Processing," *IEEE Trans. Comput.* **C-36**(11), 1265–1274.

[3] Gertner, I. and Tolimieri, R. (1989), "Fast Algorithm to Compute Multidimensional Discrete Fourier Transform," *SPIE Real-Time Signal Processing Proceedings*, San Diego, CA, 132–146.

[4] Jordan, C. (1870), *Traité des Substitutions*, Paris, 96–97.

[5] Vulis, M. (1989), "The Weighted Redundancy Transform," *IEEE Trans. ASSP* **ASSP-37** (11).

[6] Vulis, M. and Tsai, D. (1990), "Computing Discrete Fourier Transform on a Rectangular Data Array," *IEEE Trans. ASSP* **ASSP-38** (2).

Problems

1. Show that every automorphism of a finite abelian group maps lines to lines, and if $L = L(\mathbf{x})$, then $\alpha(L) = L(\alpha(\mathbf{x}))$ for every line L and automorphism α.

2. Consider $A = \mathbf{Z}/9 \times \mathbf{Z}/9$. Show that the points

$$\mathbf{a} = (2,3), \qquad \mathbf{b} = (6,4)$$

generate lines, say L_1 and L_2. Find $\alpha \in SL(2, \mathbf{Z}/9)$ such that $\alpha(L_1) = L_2$.

3. Consider $A = \mathbf{Z}/7 \times \mathbf{Z}/7$ and $L_1 = (1,0)$, $L_2 = (1,1)$, $L_3 = (1,2)$. Find $\alpha, \beta, \gamma \in SL(2, \mathbf{Z}/7)$ such that $\alpha(L_1) = L_2$, $\beta(L_2) = L_3$, and $\gamma(L_1 \oplus L_2) = L_2 \oplus L_3$.

4. Relative to the bilinear form $\Psi(\mathbf{a}, \mathbf{b}) = a_2 b_1 + a_1 b_2$ in $A = \mathbf{Z}/M \times \mathbf{Z}/M$, for every line L in A find an automorphism α of A such that $L^{\perp} = \alpha(L)$.

5. Relative to the bilinear form $\Psi(\mathbf{a}, \mathbf{b}) = a_1 b_1 + a_2 b_2 + a_3 b_3$ in $A = \mathbf{Z}/M \times \mathbf{Z}/M \times \mathbf{Z}/M$, for every line L in A find an automorphism α of A such that $L^{\perp} = \alpha(L)$.

6. Determine a set of 2-dimensional covering planes in $\mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$. Find the corresponding set of 2-dimensional dual planes.

7. Consider $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$. Find the dual of the line $L = L(0,1,1)$.

8. Find a set of 2-dimensional covering planes in $\mathbf{Z}/9 \times \mathbf{Z}/12 \times \mathbf{Z}/15$. Find the corresponding set of 2-dimensional dual planes.

9. Consider $\mathbf{a} = (1,1,1)$ and $\mathbf{b} = (2,1,0)$ in $\mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$ and find $\alpha \in SL(3, \mathbf{Z}/3)$ such that $\alpha \mathbf{a} = \mathbf{b}$.

10. Show that every line in $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$ has a complement; i.e., there exists a 2-dimensional plane P such that $A = L \oplus P$.

8

Reduced Transform Algorithms

8.1 Introduction

Highly parallel computing machines can vary over a wide range of design philosophies, but all depend on some form of space–time concurrency for their potential high-speed computing capacity. Typically, such computers feature a collection of homogeneous processing elements (nodes) together with an interconnection network and can be characterized by the *granularity*, or power, of the node processors, the *degree of parallelism* as measured by the number of independent processing elements and the complexity of node *coupling*, which describes the degree of interaction between nodes.

The Cooley–Tukey type FFT algorithms typically require alternating stages of discrete Fourier transform (DFT) with stages of data exchange. Extensive interprocessor communication is usually required. The algorithms designed in this chapter called *Reduced Transform Algorithms* (RTA) tend to minimize interprocessor communication at the cost of more complex input-output operations. They are based on the geometry of finite abelian groups and the Fourier transform duality between periodization and decimation.

Generally, these RTAs reduce the number of 1-dimensional discrete Fourier transforms needed to compute the multidimensional DFT below the number required by the row–column method. The 2-dimensional RTA is computed by a set of independent 1-dimensional DFT along lines in the output index set. The degree of parallelism of the algorithm is given by the number of lines covering the output, and the node granularity of the

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

algorithm is given by some preadditions followed by a 1-dimensional DFT. An essential property of the algorithm is the uniformity of the independent calculations. In the ideal case, the granularity and degree of parallelism of the algorithm and target machine coincide.

In higher dimensions, we can vary the granularity and degree of parallelism of the algorithm by specifying the dimension of the hyperplanes in the output index set on which the DFT is computed. We can design an N-dimensional DFT that proceeds by a set of independent K-dimensional DFTs. The granularity of the computation is given by some preadditions followed by a K-dimensional DFT, and the degree of parallelism of the algorithm is the number of the K-dimensional planes required to cover the output index set.

The independence of the Fourier transform computations permits simple "segmentation" of the computation. For example, if the machine parallelism is not sufficient to support the computation parallelism, then segmentation is necessary. Assuming one Fourier transform per node, the ratio of the number of lines to the number of nodes equals the number of independent passes through the system required to complete the computation by the line algorithm or 1-dimensional RTA.

Similar statements can be made if higher-dimensional planes are taken to partition the Fourier transform output. Combining RTA with the Cooley–Tukey or the Good–Thomas FFT algorithms can also affect the algorithm's parallelism and granularity, but at the cost of introducing interprocessor communications between arithmetic stages [3].

The RTA described in this chapter decomposes the FT computation by localizing the output computation to planes in some covering set of planes of specified dimension. A dual RTA can easily be derived that localizes by decimating input to such planes. Output periodizations are now required. In [8], various extensions of RTA are described permitting hyperplanes not passing through the origin and variation in hyperplane dimension.

8.2 General Structure

In this section, we will describe the general structure of RTAs for the finite abelian group

$$A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M, \quad N \text{ factors.}$$

Take $f \in L(A)$. We will compute $F_\phi f$ on A by computing $F_\phi f$ on a set of K-dimensional planes that cover A. Since such K-dimensional planes intersect in general, redundant computation is a characteristic of RTAs. The multiplicity results of chapter 7 quantify this redundancy. In [9] algorithms were designed to incorporate this redundancy. However, the uniformity of the basic approach described in this work usually outweighs any modification on highly parallel machines.

Consider an output K-dimensional *decimating plane* P and an input $(N-K)$-dimensional *periodizing plane* Q satisfying $P = Q^{\perp}$. We can write $A = Q \oplus Q^c$, where Q^c is a K-dimensional plane called a *complement* of Q. Any Q-periodic function can be viewed as a function on Q^c. The periodization $\mathrm{Per}_Q f$ is related to the decimation $\mathrm{Dec}_P F_\phi f$ by a K-dimensional Fourier transform

$$\mathrm{Dec}_P F_\phi f(\mathbf{y}) = \sum_{\mathbf{x} \in Q^c} (\mathrm{Per}_Q f)(\mathbf{x}) \langle \mathbf{x}, \phi(\mathbf{y}) \rangle, \qquad \mathbf{y} \in P.$$

To compute $F_\phi f$ on A, we choose

- a covering set of output K-dimensional decimating planes

$$P_1, \ldots, P_T,$$

- a set of input $(N-K)$-dimensional periodizing planes

$$Q_1, \ldots, Q_T$$

satisfying

$$P_t = Q_t^{\perp}, \qquad 1 \le t \le T,$$

- a set of K-dimensional planes

$$Q_1^c, \ldots, Q_T^c$$

satisfying

$$A = Q_t \oplus Q_t^c, \qquad 1 \le t \le T.$$

The algorithm inputs these parameters and proceeds as follows.

- Compute T periodizations

$$g_t = \mathrm{Per}_t f, \qquad 1 \le t \le T,$$

where Per_t denotes periodization relative to the plane Q_t.

- View g_t as a function on Q_t^c, and compute T K-dimensional Fourier transforms

$$G_t(\mathbf{y}) = \sum_{\mathbf{x} \in Q_t^c} g_t(\mathbf{x}) \langle \mathbf{x}, \phi(\mathbf{y}) \rangle, \qquad \mathbf{y} \in P_t.$$

Then $F_\phi f$ is given by

$$F_\phi f(\mathbf{y}) = G_t(\mathbf{y}), \qquad \mathbf{y} \in P_t.$$

This algorithm is given in a diagram in Figure 8.1.

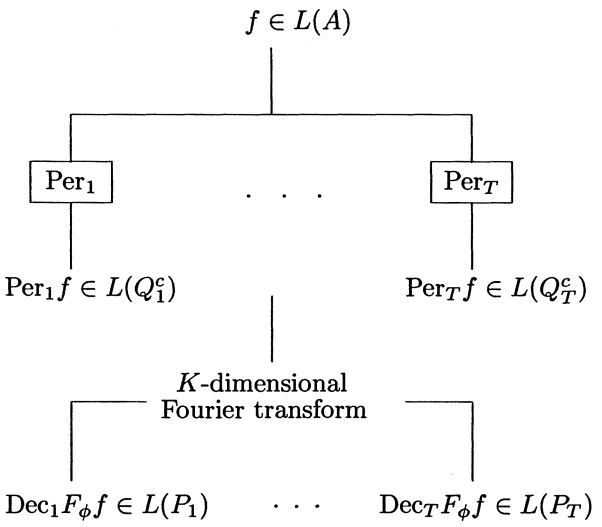


Figure 8.1 RTA

The granularity and parallelism of the RTA can be described in terms of the underlying geometry. The dimension K of the decimating planes determines the size of the K-dimensional Fourier transforms. The larger we take K, the smaller the number of Fourier transforms required. In some applications, it may be useful to remove the restriction that all the output decimating planes have the same dimensions.

The initial addressing and additions required for the periodizations must be supported by broadcast and computational units of the machine. We will discuss this stage in greater detail on specific machines in a subsequent chapter. The K-dimensional Fourier transforms require no interprocessor communication.

8.3 Periodizations

Assume the presentation $A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M$, N factors, and a corresponding product isomorphism ϕ from A onto its character group, throughout. Duality and transpose will be taken with respect to ϕ. Relative to the presentation, we can identify $\mathbf{a} \in A$ with an N-tuple column vector over $\mathbf{Z}/M$ and every $f \in L(A)$ with an M^N-column vector $\mathbf{f}$ over $\mathbf{C}$ by taking the lexicographic ordering on A. We identify $\mathrm{Aut}(A)$ with $GL(N, \mathbf{Z}/M)$.

For $\alpha \in \mathrm{Aut}(A)$, define the linear isomorphism $P(\alpha)$ of $L(A)$ by

$$P(\alpha)f(\mathbf{a}) = f(\alpha(\mathbf{a})), \quad \mathbf{a} \in A,\ f \in L(A),$$

and define the $M^N \times M^N$ permutation matrix $\mathbf{P}(\alpha)$ by the condition that $\mathbf{P}(\alpha)\mathbf{f}$ is the vector corresponding to $P(\alpha)f$.

Consider an output K-dimensional decimating plane

$$P = P(\mathbf{x}_1, \ldots, \mathbf{x}_K).$$

Every $\mathbf{x} \in P$ can be written uniquely in the form

$$\mathbf{x} = x^{(1)}\mathbf{x}_1 + \cdots + x^{(K)}\mathbf{x}_K, \quad x^{(k)} \in \mathbf{Z}/M, \quad 1 \leq k \leq K.$$

Order the elements of P by the lexicographic ordering relative to the elements $x^{(1)}, \ldots, x^{(K)}$. Choose a corresponding input $(N-K)$-dimensional periodizing plane

$$Q = P(\mathbf{y}_1, \ldots, \mathbf{y}_{N-K})$$

and a complement to Q

$$Q^c = P(\mathbf{z}_1, \ldots, \mathbf{z}_K).$$

Define the automorphism α of A by the matrix

$$\alpha = [\mathbf{y}_1 \cdots \mathbf{y}_{N-K}\mathbf{z}_1 \cdots \mathbf{z}_K].$$

Set $P_K = P(e_1, \ldots, e_K)$. Then $P_K^{\perp} = P(e_{K+1}, \ldots, e_N)$. Direct computation shows that

$$\begin{aligned} \alpha(P_{N-K}) &= Q, \\ \alpha(P_{N-K}^{\perp}) &= Q^c, \\ (\alpha^{-1})^t(P_{N-K}^{\perp}) &= P. \end{aligned}$$

The Q-periodization of $f \in L(A)$, viewed as a function on Q^c, is

$$\mathrm{Per}_Q f(\mathbf{y}) = \sum_{\mathbf{x} \in Q} f(\mathbf{y} + \mathbf{x}), \qquad \mathbf{y} \in Q^c. \tag{8.1}$$

Every $\mathbf{x} \in Q$ and $\mathbf{y} \in Q^c$ can be written uniquely in the form $\mathbf{x} = \alpha\mathbf{u}$, $\mathbf{u} \in P_{N-K}$ and $\mathbf{y} = \alpha\mathbf{v}$, $\mathbf{v} \in P_{N-K}^{\perp}$. Formula (8.1) can be written

$$\mathrm{Per}_Q f(\alpha(\mathbf{v})) = \sum_{\mathbf{u} \in P_{N-K}} P(\alpha) f(\mathbf{u} + \mathbf{v}), \qquad \mathbf{v} \in P_{N-K}^{\perp}. \tag{8.2}$$

Both sides of (8.2) are functions on $P_{N-K}^{\perp}$. The right-hand side can be written in vector form as

$$(I_{M^K} \otimes 1_{M^{N-K}}^t)\mathbf{P}(\alpha)\mathbf{f}.$$

Define

$$F_P\mathbf{f} = F_K(M)(I_{M^K} \otimes 1_{M^{N-K}}^t)\mathbf{P}(\alpha)\mathbf{f},$$

where $F_K(M)$ is the K-dimensional tensor product $F(M) \otimes \cdots \otimes F(M)$. $F_P\mathbf{f}$ defines a function $F_P f$ on $P_{N-K}^{\perp}$. We can compute $F_\phi f$ on P by the formula

$$F_\phi f((\alpha^{-1})^t(\mathbf{v})) = F_P f(\mathbf{v}), \qquad \mathbf{v} \in P_{N-K}^{\perp}.$$

By abuse of language, we will say that $F_P\mathbf{f}$ is the vector corresponding to the restriction of $F_\phi f$ on P. The periodization stage has been decomposed into a permutation stage $\mathbf{P}(\alpha)$ followed by a preaddition stage $(I_{M^K} \otimes 1_{M^{N-k}}^t)$. This decomposition is shown in Figure 8.2.

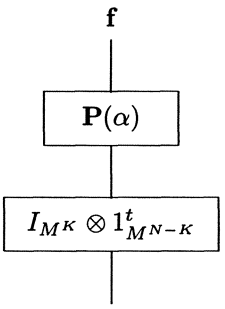
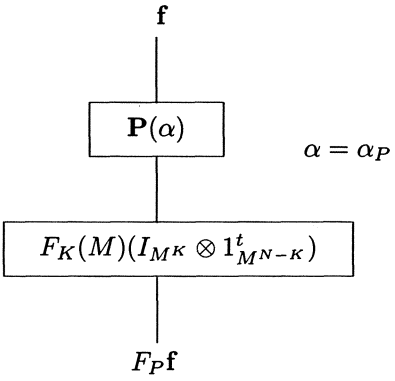


Figure 8.2 RTA periodization stage

The complete computation of $F_\phi f$ on P can be decomposed into the permutation stage $\mathbf{P}(\alpha)$ followed by the computation stage

$$F_K(M)(I_{M^K} \otimes 1_{M^{N-K}}^t).$$

This is diagramed in Figure 8.3.



$$F_\phi f((\alpha^{-1})^t(\mathbf{v})) = F_P\mathbf{f}(\mathbf{v}), \qquad \mathbf{v} \in P_{N-K}^{\perp}$$

Figure 8.3 RTA computation of $F_\phi f$ on P

We see that the computation stage is independent of the K-dimensional plane P. Although α is not uniquely determined by P, since P, Q, Q^c and hence α are usually precomputed, we will express the dependence of α on P by writing α as α_P. In the following sections, we will describe $\mathbf{P}(\alpha)$ in terms of shifts and stride permutations.

8.4 Examples

8.4.1 2-Dimensional $p \times p$, p a prime

Information necessary for designing RTA for $p \times p$ is listed in Table 8.1.

Table 8.1 $0 \leq j < p$

Decimating Line	Periodizing Line	Complement	Automorphism
$L_j = L(1,j)$	$L(-j,1)$	$L(1,0)$	$\alpha_j = \begin{bmatrix} -j & 1 \\ 1 & 0 \end{bmatrix}$
$L_p = L(0,1)$	$L(1,0)$	$L(0,1)$	I_2

Denote the permutation matrix corresponding to α_j by $\mathbf{P}_j$. Consider the case $p = 3$. Then $\mathbf{P}_j\mathbf{f}$ is the vector

$$\begin{bmatrix} f(0,0) \\ f(-j,1) \\ f(-2j,2) \\ f(1,0) \\ f(1-j,1) \\ f(1-2j,2) \\ f(2,0) \\ f(2-j,1) \\ f(2-2j,2) \end{bmatrix}$$

Let S_3 be the 3-point cyclic shift matrix

$$S_3 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

and $P(9,3)$ the 9-point stride by 3 matrix. Define

$$Z_3 = I_3 \oplus S_3 \oplus S_3^2,$$

$$Z_3\mathbf{f} = \begin{bmatrix} f(0,0) \\ f(1,0) \\ f(2,0) \\ f(2,1) \\ f(0,1) \\ f(1,1) \\ f(1,2) \\ f(2,2) \\ f(0,2) \end{bmatrix}.$$

Direct computation shows that

$$\mathbf{P}_0 = P(9,3), \quad \mathbf{P}_1 = P(9,3)Z_3, \quad \mathbf{P}_2 = P(9,3)Z_3^2.$$

Figure 8.4 displays the input permutation stage, while the permutation stage and the computation stage are combined in Figure 8.5.

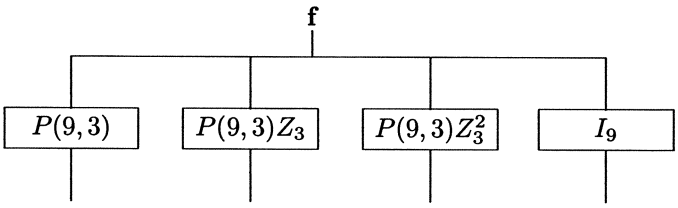


Figure 8.4 RTA permutation stage: 3×3

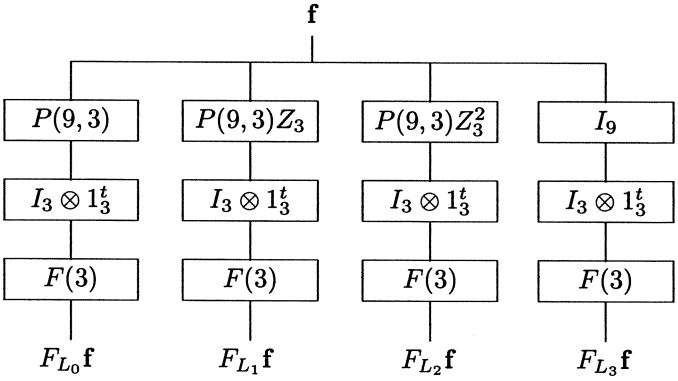


Figure 8.5 RTA: 3×3

The general case $M = p$, p a prime, follows in exactly the same way. We have

$$\mathbf{P}_j = P(p^2, p)Z_p^j,$$

where $Z_p = I_p \oplus S_p \oplus \cdots \oplus S_p^{p-1}$, with S_p the p-point cyclic shift matrix. Figure 8.6 displays this permutation stage. The output of each permutation

is subjected to the same computational stage

$$F(p)(I_p \otimes 1_p^t)$$

with the result

$$F_{L_j}\mathbf{f} = F(p)(I_p \otimes 1_p^t)\mathbf{P}_j\mathbf{f},$$

where $F_{L_j}\mathbf{f}$ is the vector formed from the values of $F_\phi f$ on the line L_j. Figure 8.7 displays the computation.

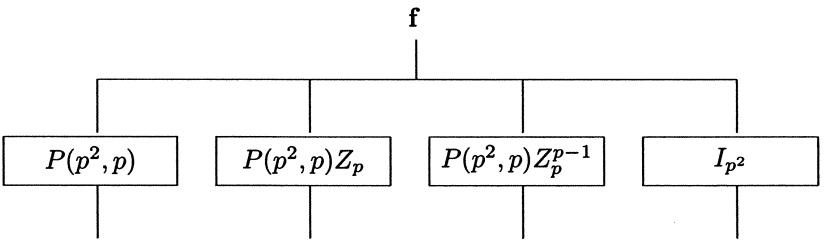


Figure 8.6 RTA permutation stage $p \times p$

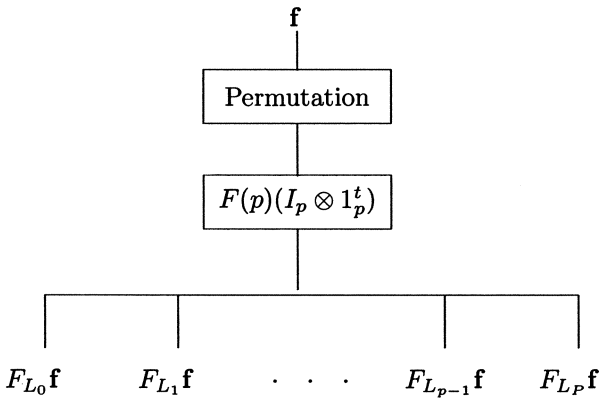


Figure 8.7 RTA: $p \times p$

8.4.2 2-Dimensional $p^2 \times p^2$, p a prime

Set

$$\mathbf{P}_j = \mathbf{P}\left(\begin{bmatrix} -j & 1 \\ 1 & 0 \end{bmatrix}\right), \quad 0 \le j < p^2,$$

$$\mathbf{Q}_k = \mathbf{P}\left(\begin{bmatrix} 1 & 0 \\ -pk & 1 \end{bmatrix}\right), \quad 0 \le k < p.$$

Information necessary for designing RTA for the case $p^2 \times p^2$ is listed in Table 8.2.

Table 8.2 $0 \le j < p$

Decimating Line	Periodizing Line	Complement	Automorphism
$L(1,j)$	$L(-j,1)$	$L(1,0)$	$\begin{bmatrix} -j & 1 \\ 1 & 0 \end{bmatrix}$
$L(pk,1)$	$L(1,-pk)$	$L(0,1)$	$\begin{bmatrix} 1 & 0 \\ -pk & 1 \end{bmatrix}$

We will first work out the case $p = 2$. By direct computation,

$$\mathbf{P}_0 = P(16,4), \quad \mathbf{P}_1 = P(16,4)Z_4,$$
$$\mathbf{P}_2 = P(16,4)Z_4^2, \quad \mathbf{P}_3 = P(16,4)Z_4^3.$$

Since $\mathbf{Q}$ affects the second component, we will first act by $P(16,4)$ on $\mathbf{f}$ to reverse the order of the variables. Set

$$W_4 = I_2 \otimes (I_4 \oplus S_4^2)$$

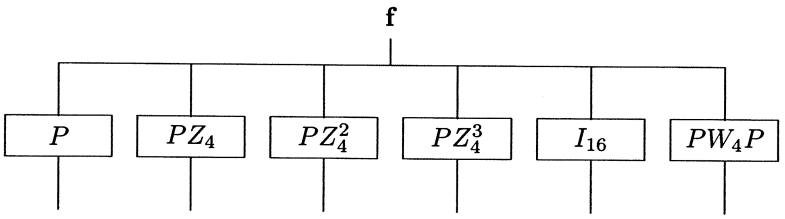
and form the vector $W_4P(16,4)\mathbf{f}$. Reading left to right and then down, this vector is as follows.

$$\begin{array}{llllll} f(0,0) & f(0,1) & f(0,2) & f0,3) & & \\ & f(1,2) & f(1,3) & f(1,0) & f(1,1) & \\ & & f(2,0) & f(2,1) & f(2,2) & f(2,3) \\ & & & f(3,2) & f(3,3) & f(3,0) \quad f(3,1). \end{array}$$

Comparing $\mathbf{Q}_k\mathbf{f}$ with $W_4^kP(16,4)\mathbf{f}$, $k = 0,1$, we see that

$$\mathbf{Q}_0\mathbf{f} = \mathbf{f}, \quad \mathbf{Q}_1\mathbf{f} = P(16,4)W_4P(16,4).$$

The permutation stage is pictured in Figure 8.8.



$$P = P(16,4), \quad Z_4 = \sum_{k=0}^{3} \oplus S_4^k, \quad W_4 = I_2 \otimes (I_4 \oplus S_4^2)$$

Figure 8.8 RTA permutation stage: 4×4

Each periodized output is subjected to the same computational stage

$$F(4)(I_4 \otimes 1_4^t).$$

The output of this stage computes $F_\phi f$ on the decimating lines.

The general case $M = p^2$, p a prime, follows in the same way. The input permutation stage is given by the permutation matrices

$$\begin{aligned}
\mathbf{P}_j &= P(p^4, p^2) Z_{p^2}^j, \quad 0 \le j < p^2, \\
\mathbf{Q}_k &= P(p^4, p^2) W_{p^2}^k P(p^4, p^2), \quad 0 \le k < p, \\
W_{p^2} &= I_p \otimes (I_{p^2} \oplus S_{p^2}^p \oplus \cdots \oplus S_{p^2}^{(p-1)p}), \\
Z_{p^2} &= \sum_{k=0}^{p^2-1} \oplus S_{p^2}^k
\end{aligned}$$

with S_{p^2} the $p^2 \times p^2$ cyclic shift matrix.

The outputs of the permutation stage are acted on by the matrix

$$F(p^2)(I_{p^2} \otimes 1_{p^2}^t).$$

Summarizing,

$$\begin{aligned}
F_{L(1,j)}\mathbf{f} &= F(p^2)(I_{p^2} \otimes 1_{p^2}^t)\mathbf{P}_j\mathbf{f}, \\
F_{L(pk,1)}\mathbf{f} &= F(p^2)(I_{p^2} \otimes 1_{p^2}^t)\mathbf{Q}_k\mathbf{f}.
\end{aligned}$$

Each arithmetic step requires $(p^2 - 1)p^2$ additions to implement the action of $I_{p^2} \otimes 1_{p^2}^t$ followed by the p^2-point Fourier transform. Figure 8.9 summarizes the procedures in RTA for $p^2 \times p^2$.

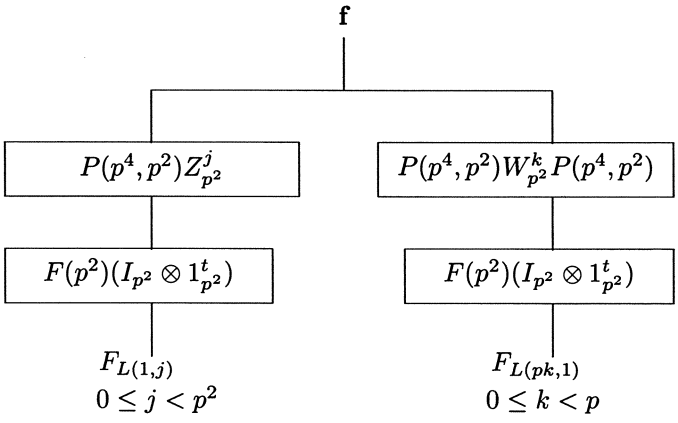


Figure 8.9 2-dimensional $p^2 \times p^2$ RTA

8.4.3 2-dimensional $p^R \times p^R$ RTA

The derivation of RTA for $p^R \times p^R$ is similar to that for $p^2 \times p^2$. The resulting algorithm is given in Figure 8.10.

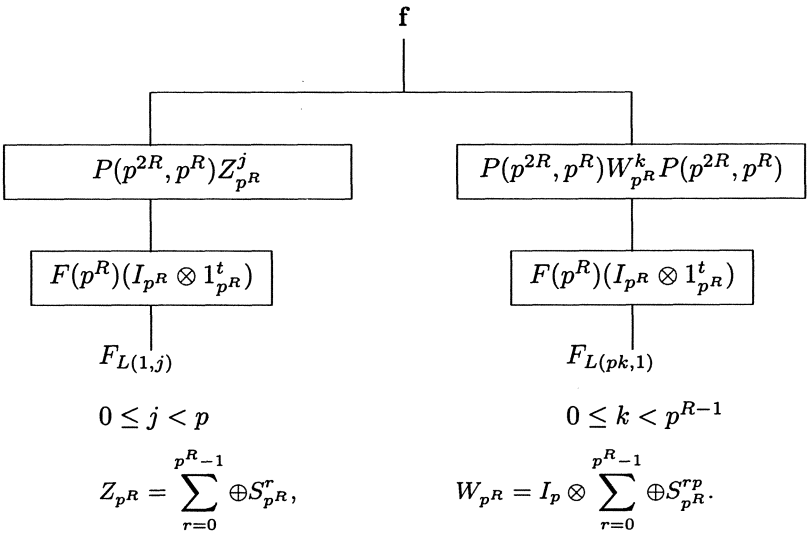


Figure 8.10 2-dimensional $p^R \times p^R$ RTA

8.4.4 2-Dimensional $M = pq$, p and q distinct primes

Suppose $\{e_1, e_2\}$ is the complete system of idempotents for the factorization $M = pq$. By the CRT,

$$A = A_p \times A_q, \tag{8.3}$$

where

$$A_p = \mathbf{Z}/p \times \mathbf{Z}/p, \qquad A_q = \mathbf{Z}/q \times \mathbf{Z}/q. \tag{8.4}$$

Every $\mathbf{a} \in A$ can be written uniquely in the form

$$\mathbf{a} = e_1\mathbf{a}^{(1)} + e_2\mathbf{a}^{(2)}, \qquad \mathbf{a}^{(1)} \in A_p, \quad \mathbf{a}^{(2)} \in A_q. \tag{8.5}$$

A_p and A_q are ordered by the presentations (8.4). The CRT ordering of A is defined by the lexicographic ordering relative to the variables $\mathbf{a}^{(1)}$, $\mathbf{a}^{(2)}$. If $f \in L(A)$, denote by $\mathbf{f}_e$ the vector formed relative to the CRT ordering.

Table 8.3 is a list of necessary information for deriving RTA for the case pq, where p and q are distinct primes.

An automorphism α of A induces automorphisms α_p and α_q of the primary factors A_p and A_q such that

$$\alpha(\mathbf{a}) = e_1\alpha_p(\mathbf{a}^{(1)}) + e_2\alpha_q(\mathbf{a}^{(2)}), \qquad \mathbf{a} \in A.$$

Table 8.3 $u_j = e_1 j + e_2$, $0 \le j < p$, $v_k = e_2 k + e_1$, $0 \le k < q$.

Output Line	Periodizing Line	Complement	Automorphism
$L(1,m)$	$L(-m,1)$	$L(1,0)$	$\begin{bmatrix} -m & 1 \\ 1 & 0 \end{bmatrix}$
$L(e_1,u_j)$	$L(-u_j,e_1)$	$L(e_1,e_2)$	$\begin{bmatrix} -u_j & e_1 \\ e_1 & e_2 \end{bmatrix}$
$L(e_2,v_k)$	$L(-v_k,e_2)$	$L(e_2,e_1)$	$\begin{bmatrix} -v_k & e_2 \\ e_2 & e_1 \end{bmatrix}$
$L(0,1)$	$L(1,0)$	$L(0,1)$	I_2

If α is viewed as a matrix in $GL(2,\mathbf{Z}/pq)$ then the matrices $\alpha_p \in GL(2,\mathbf{Z}/p)$ and $\alpha_q \in GL(2,\mathbf{Z}/q)$ corresponding to the presentation (8.4) are formed by taking the coefficients of α modulo p and modulo q. Denote by $\mathbf{P}_e$ the permutation matrix corresponding to α relative to the CRT ordering of A and by $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ the permutation matrices corresponding to α_p and α_q. Then

$$\mathbf{P}_e = \mathbf{P}^{(2)} \otimes \mathbf{P}^{(1)}. \tag{8.6}$$

Formula (8.5) can be used to describe the input permutation stage. Consider, in particular, the output line $L(e_1,u_j)$, $0 \le j < p$. Then the corresponding automorphisms α_p and α_q are given by

$$\alpha_p = \begin{bmatrix} -j & 1 \\ 1 & 0 \end{bmatrix}, \qquad \alpha_q = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The corresponding permutation matrices $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ are given by

$$\mathbf{P}^{(1)} = P(p^2,p)Z_p^j, \qquad \mathbf{P}^{(2)} = I_q \otimes R_q,$$

where R_q is the q-point time reversal matrix. Then

$$\mathbf{P}_e = (I_q \otimes R_q) \otimes (P(p^2,p)Z_p^j).$$

The remaining cases can be handled in the same way, and they generalize to the case of arbitrary M.

The 2-dimensional $p \times p$ RTA can be combined with the Good–Thomas FFT to produce a hybrid 2-dimensional $pq \times pq$ algorithm. Consider the Good–Thomas factorization,

$$\begin{aligned} F(pq) \otimes F(pq) &= Q_1(F(p) \otimes F(p)) \otimes (F(q) \otimes F(q))Q_2 \\ &= Q_1'(I_{q^2} \otimes F(p,p))P(I_{p^2} \otimes F(q,q))Q_2', \end{aligned}$$

where Q_1', Q_2', and P are permutation matrices with, for example, $P = P(p^2q^2, q^2)$. Computing the 2-dimensional FTs $F(p,p)$ and $F(q,q)$ by RTA, the 2-dimensional $pq \times pq$ FT can be computed using $p^2(q+1)$ 1-dimensional q-point, and $q^2(p+1)$ 1-dimensional p-point FTs. Interprocessor communication is required between these two arithmetic stages. The effect is to increase the number of independent computations and decrease the granularity.

8.4.5 3-D RTA

Suppose

$$A = \mathbf{Z}/M \times \mathbf{Z}/M \times \mathbf{Z}/M.$$

We now have the option of partitioning the output on lines covering A or on 2-dimensional planes covering A. Both options will be discussed.

First we will output on lines. Consider a decimating line

$$P = L(r_1, r_2, r_3),$$

a dual plane

$$Q = L(s_1, s_2, s_3) \oplus L(t_1, t_2, t_3),$$

and a complement

$$Q^c = L(u_1, u_2, u_3).$$

The matrix

$$\alpha = \begin{bmatrix} s_1 & t_1 & u_1 \\ s_2 & t_2 & u_2 \\ s_3 & t_3 & u_3 \end{bmatrix}$$

is in $GL(3, \mathbf{Z}/M)$ and hence defines an automorphism of A relative to which the initial permutation will be defined.

The Fourier transform $F_\phi f$, on the line P, determines a vector $F_P\mathbf{f}$ given by the formula

$$F_P\mathbf{f} = F(M)(I_M \otimes 1^t_{M^2})\mathbf{P}(\alpha)\mathbf{f}.$$

Figure 8.11 shows the steps in this case of RTA.

We will now output on planes. Suppose a plane P is given by

$$P = L(\mathbf{r}) \oplus L(\mathbf{s}), \qquad \mathbf{r}, \mathbf{s} \in A.$$

Denote the dual of P by

$$Q = L(\mathbf{t}), \qquad \mathbf{t} \in A,$$

and a complement of Q by

$$Q^c = L(\mathbf{u}) \oplus L(\mathbf{v}), \qquad \mathbf{u}, \mathbf{v} \in A.$$

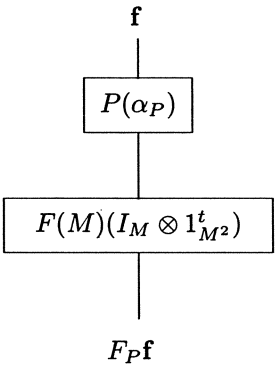


Figure 8.11 RTA output on lines: $M \times M \times M$

The matrix

$$\alpha = \begin{bmatrix} t_1 & u_1 & v_1 \\ t_2 & u_2 & v_2 \\ t_3 & u_3 & v_3 \end{bmatrix}$$

is in $GL(3, \mathbf{Z}/M)$ and hence defines an automorphism of A.

The FT $F_\phi f$ on the plane P determines a vector $F_P\mathbf{f}$ given by the formula

$$F_P\mathbf{f} = F(M, M)(I_{M^2} \otimes 1^t_M)\mathbf{P}(\alpha)\mathbf{f},$$

where $F(M, M) = F(M) \otimes F(M)$, the 2-dimensional $M \times M$ FT matrix. Steps required in this case of RTA is shown in Figure 8.12.

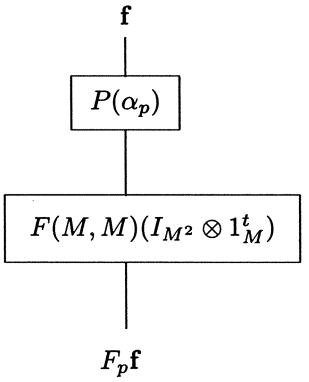


Figure 8.12 RTA output on planes: $M \times M \times M$

As before, the arithmetic part is independent of the output plane, while the initial permutation varies with the output plane.

Denote the number of lines in A by R and the number of lines in $\mathbf{Z}/M \times \mathbf{Z}/M$ by S. The number of covering planes of A is S. In general, $S < R$. The algorithmic parameters are given in Table 8.4. The trade-off is clear. The

plane output RTA has less parallelism at the cost of greater granularity as compared with the line output RTA.

Table 8.4 Parallelism in RTA

Output	Parallelism	Granularity/per node
Lines	R	$M(M^2-1)$ additions $+F(M)$
Planes	S	$M(M^2-1)$ additions $+F(M) \otimes F(M)$

8.5 RTA Permutations

The RTA permutations can be described in terms of cyclic shift matrices and stride permutations. A few simple rules govern the relationship between indexing set automorphisms and data permutation matrices. Suppose the indexing set is given by

$$A = \mathbf{Z}/M \times \cdots \times \mathbf{Z}/M, \qquad N \text{ factors},$$

with the usual ordering. Choose integers $R > 0$, $S > 0$ such that $N = R+S$. The automorphism of A given by the matrix

$$J(R,S) = \begin{bmatrix} 0 & I_R \\ I_S & 0 \end{bmatrix}$$

interchanges the last R coordinates with the first S coordinates. The corresponding data permutation matrix is the stride permutation

$$P(M^N, M^R).$$

Consider the automorphism

$$\alpha_R \oplus I_S, \tag{8.7}$$

where $\alpha_R \in GL(R, \mathbf{Z}/M)$. The indexing set A can be segmented into M^R consecutive "vectors" each of size M^S characterized by the condition that the last S coordinates are fixed, while the first R coordinates range over all possible values. The action of (8.7) keeps each segment invariant, and on each segment it is given by the action of α_R on the first R coordinates. If P_R denotes the M^R-point permutation matrix corresponding to α_R, then the M^N-point permutation matrix corresponding to $\alpha_R \oplus I_S$ is given by

$$I_{M^S} \otimes P_R.$$

In general, the automorphism of A given by

$$\alpha_R \oplus \alpha_S,$$

where $\alpha_R \in GL(R, \mathbf{Z}/M)$ and $\alpha_S \in GL(S, \mathbf{Z}/M)$, gives rise to the permutation matrix

$$P(\alpha_S) \otimes P(\alpha_R),$$

where $P(\alpha_R)$ is the M^R-point permutation matrix corresponding to α_R and $P(\alpha_S)$ is the M^S-point permutation matrix corresponding to α_S.

In the following example, we use the formula

$$P(\alpha\beta) = P(\beta)P(\alpha).$$

Example 8.1 Set

$$r_N = \begin{bmatrix} 0 & \cdot & \cdot & \cdot & 0 & 1 \\ 0 & & & 0 & 1 & 0 \\ \cdot & & & \cdot & & \cdot \\ \cdot & & \cdot & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ 1 & 0 & \cdot & \cdot & \cdot & 0 \end{bmatrix} \in GL(N, \mathbf{Z}/M)$$

and denote the corresponding data permutation matrix by R_N. In particular,

$$R_2 = P(M^2, M).$$

Since

$$r_3 = J(1,2)(r_2 \oplus 1),$$

we have

$$R_3 = (I_M \otimes P(M^2, M))P(M^3, M).$$

Arguing in the same way,

$$R_N = (I_M \otimes R_{N-1})P(M^N, M).$$

For $\mathbf{j} = (j_2, \ldots, j_N)$, $0 \leq j_2, \ldots, j_N < M$, define the matrix $\alpha(\mathbf{j}) \in SL(N, \mathbf{Z}/M)$ by

$$\alpha(\mathbf{j}) = \begin{bmatrix} 1 & -j_2 & \cdot & \cdot & \cdot & -j_N \\ 0 & 1 & & & & 0 \\ 0 & 0 & 1 & & & 0 \\ \cdot & & & \cdot & & \cdot \\ \cdot & & & & \cdot & \cdot \\ 0 & & & & & 1 \end{bmatrix}.$$

The corresponding permutation matrix $P(\mathbf{j})$ is given by

$$P(\mathbf{j}) = \sum_{r_N=0}^{M-1} \cdots \sum_{r_2=0}^{M-1} \oplus S_M^{r_2 j_2 + \cdots + r_N j_N},$$

where S_M is the M-point cyclic shift matrix and the summation is ordered from right to left.

We will use these rules to write the data permutations required in the RTA as cyclic shifts and stride permutations in several cases. Consider the 3-dimensional $p \times p \times p$ RTA. First we output on lines. The automorphisms are given by

$$\alpha = \begin{bmatrix} -j_3 & -j_2 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad 0 \le j_2, j_3 < p,$$

$$\beta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -j_3 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad 0 \le j_3 < p.$$

Since

$$\alpha = \alpha(\mathbf{j}) r_3, \quad \mathbf{j} = (j_2, j_3),$$

we have

$$\begin{aligned} P(\alpha) &= R_3 P(\mathbf{j}) \\ &= \left(I_p \otimes P(p^2, p)\right) P(p^3, p) \left(\sum_{r_3=0}^{p-1} \sum_{r_2=0}^{p-1} \oplus S_p^{r_2 j_2 + r_3 j_3} \right). \end{aligned}$$

Since the output is segmented into p consecutive vectors each of size p^2 and the components of these vectors are summed, we can multiply $P(\alpha)$ by any matrix of the form $I_p \otimes P$, where P is a permutation matrix on p^2 points. Hence we can replace $P(\alpha)$ by

$$P(p^3, p) \left(\sum_{r_3=0}^{p-1} \sum_{r_2=0}^{p-1} \oplus S_p^{r_2 j_2 + r_3 j_3} \right).$$

Arguing in the same way,

$$P(\beta) = \left(P(p^2, p) \left(\sum_{r=0}^{p-1} \oplus S_p^{r j_3} \right) \right) \otimes I_p,$$

which can be replaced by

$$P(p^3, p) \left(I_p \otimes \sum_{r=0}^{p-1} \oplus S_p^{r j_3} \right)$$

in the implementation of the RTA.

Gertner (1988) [1] and Vulis (1989) [9] designed dual versions of RTA. We have presented RTA following the first work. The years following the works extended initial results to a wider range of transform sizes. Originally, RTA was called the *Line Algorithm*, but as hyperplanes were introduced into the picture in Gertner–Tolimieri (1989) [2] and Rofheart (1991) [4], a less focused name was necessary. Shamash (1989) [6] explored the implementation of RTA on several architectures.

References

[1] Gertner, I. (1988), "A New Efficient Algorithm to Compute the Two-dimensional Discrete Fourier Transform," *IEEE Trans. ASSP* **ASSP-36**(7), 1036–1050.

[2] Gertner, I. and Tolimieri, R. (1989), "Fast Algorithm to Compute Multidimensional Discrete Fourier Transform," *SPIE Real-Time Signal Processing Proc.*, San Diego, CA, 132–146.

[3] Kechriotis, G., An, M., Bletzas, M., Tolimieri, R., and Manolakos, E. (1995), "A new approach for computing multidimensional DFT's on parallel machines and its implementation on the iPSC/860 Hypercube," *IEEE Trans. on SP* **43**(1), 272–285.

[4] Rofheart, M. (1991), *Algorithms and Methods for Multidimensional Digital Signal Processing*, Ph.D. thesis, CUNY.

[5] Rofheart, M. and Gertner, I. (1990), "A Parallel Algorithm for 2-D DFT Computation with No Interprocessor Comm.," *IEEE Trans. Paral. Dist. Syst.* **1**(3), 377–382.

[6] Shamash, M. (1989), *VLSI Architectures for Multidimensional Digital Signal Processing*, Ph.D. thesis, Technion-Israel Institute of Technology.

[7] Tsai, D. and Vulis, M. (1990), "Computing Discrete Fourier Transform on a Rectangular Data Array," *IEEE ASSP* **ASSP-38**(2), 271–276.

[8] Tolimieri, R., An, M., Abdelatif, Y., Lu, C., Kechriotis, G., and Anupindi, N. (1995), "Group Invariant Fourier Transform Algorithms," in *Advances in Imaging and Electron Physics*, **93**, Academic Press.

[9] Vulis, M. (1989), "The Weighted Redundancy Transform," *IEEE ASSP* **ASSP-37**(11), 1687–1692.

Problems

1. Let $A = \mathbf{Z}/5 \times \mathbf{Z}/5$. Determine the additive and multiplicative complexity of computing $F(5,5)$ by the row–column method and by the RTA.

2. In problem 1, suppose RTA fully parallelizes on six nodes. What is the additive and multiplicative complexity of computing $F(5,5)$ by RTA?

3. In problem 1, suppose RTA fully parallelizes on three nodes. Write a hybrid FT algorithm that fully utilizes the parallelism. What is its additive and multiplicative complexity?

4. Let $A = \mathbf{Z}/5 \times \mathbf{Z}/5 \times \mathbf{Z}/5$. Determine the additive and multiplicative complexity of computing $F(5,5,5)$ using row–column method, line output RTA, and plane output RTA.

5. A multinode system can implement for "free" the initial and output permutations in RTA. Suppose the system has 64 nodes. Write a 3-dimensional $16 \times 16 \times 16$ RTA on the system.

6. Does the RTA in problem 5 change if only 34 nodes are available? Indicate why or why not by a computation flow chart.

7. Describe the input permutations for the line output RTA in the case $A = \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3 \times \mathbf{Z}/3$.

8. Do problem 7 for the 2-dimensional plane output RTA.

9. Using a hybrid RTA/Good–Thomas algorithm, write an algorithm that computes $F(12,12,12)$.

10. Design an algorithm that computes the 10×10 2-dimensional FT of a function $f \in L(\mathbf{Z}/10 \times \mathbf{Z}/10)$ on the line passing through the origin and the point $(2,3)$.

9
Field Algorithm

9.1 Introduction

In 1968, C. Rader [7] observed that for a prime number p, the p-point 1-dimensional FT could be computed by a $(p-1)\times(p-1)$ skew-circulant matrix action. S. Winograd and others greatly extended the range of Rader's method to include the p^R-point 1-dimensional FT and multidimensional generalizations [3].

In this chapter, we will derive Rader's results and discuss multidimensional extensions that depend on the indexing set being a finite field. This class of *field DFT algorithms* relies on the following observations.

I. The Fourier transform depends only on the additive group structure of the indexing set.

II. The additive group of a field with p^R elements is isomorphic to the group

$$\mathbf{Z}/p \times \cdots \times \mathbf{Z}/p, \qquad R \text{ factors.}$$

III. The multiplicative group $U(K)$ of nonzero elements of a finite field K is a cyclic group.

By property III, the FT of a finite field K can be realized as a skew-circulant matrix by ordering the input and output by successive powers of a generator of $U(K)$. Properties I and II equate the FT of a finite field with p^R elements with the $p \times \cdots \times p$ R-dimensional FT.

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

Rader's method immediately extends from the finite field $\mathbf{Z}/p$ to an arbitrary finite field, say $K = GF(p^R)$. Thus in the R-dimensional case, we obtain a $(p^R - 1) \times (p^R - 1)$ skew-circulant matrix as the *core* of the FT computation. Even for relatively small primes p and dimension R, p^R is too large for efficient computation of the skew-circulant matrix action. Thus we will discuss a factorization method for the $p^R \times p^R$ skew-circulant matrix as well.

The RTA computes the $p \times \cdots \times p$ R-dimensional FT using $\frac{p^R-1}{p-1}$ distinct p-point 1-dimensional FTs, as compared with the row–column method, which uses Rp^{R-1} distinct 1-dimensional FTs. Several authors, including H. Nussbaumer and P. Quandalle (1979) [6] and L. Auslander, E. Feig and, S. Winograd (1983) [3] have derived multiplicative multidimensional DFT algorithm, having the same reduction in the required number of 1-dimensional FT computations. Computing the p-point 1-dimensional FTs in the RTA by Rader's method leads to the result of Auslander et al., while computing these p-point 1-dimensional FTs by the multiplicative character method found in Tolimieri (1986) [8] leads to the Nussbaumer–Quandalle multidimensional DFT algorithm. These results can be found in [2]. In the spirit of this chapter, we will derive the result of Auslander et al. using multiplicative methods. M. Vulis (1989) [10] generalized these ideas to the case when the indexing set is a local ring, which then can be applied to the $\mathbf{Z}/p^R \times \cdots \times \mathbf{Z}/p^R$ N-dimensional FT.

A combination of multiplicative methods and RTA can be used to produce a hybrid DFT that has played a role in the crystallographic FT [9]. Although the details are beyond the scope of this work, the main idea is that generators for a covering set of lines can be taken from successive powers of a generator for the multiplicative group $U(K)$.

9.2 Rader Field Algorithm

For a prime number p, $\mathbf{Z}/p$ is a field. The nonzero elements of $\mathbf{Z}/p$ form a cyclic group under multiplication, denoted by $U(\mathbf{Z}/p)$. Choose any generator a of $U(\mathbf{Z}/p)$ and order $\mathbf{Z}/p$ as

$$0; a^0, \quad a, \quad \ldots, \quad a^{p-2}.$$

This ordering is called a *multiplicative ordering* of $\mathbf{Z}/p$, and the corresponding basis of evaluation functions a *multiplicative basis.*

In this section, the FT will always be presented relative to the standard bilinear form on $\mathbf{Z}/p \times \mathbf{Z}/p$ and we will use the term FT to mean standard presentation. Denote by F the matrix of the FT relative to the

multiplicative basis π. F has the form

$$F = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & 1 \\ \cdot & & & & \\ \cdot & & C(p) & & \\ \cdot & & & & \\ 1 & & & & \end{bmatrix},$$

where $C(p)$ is a $(p-1) \times (p-1)$ matrix, called the *core* of the FT relative to π. Note that $C(p)$ depends on a cyclic generator of $U(\mathbf{Z}/p)$.

An $N \times N$ matrix C is said to be *skew-circulant* if

$$C_{k,l} = C_{k+n,l-n},$$

where the indices of C are computed modulo N.

Theorem 9.1 *For a prime p, the core $C(p)$ of the p-point FT matrix relative to the multiplicative basis π is skew-circulant.*

Proof Since $a^{p-1} = 1$, for a generator a of $U(\mathbf{Z}/p)$,

$$C_{k+n,l-n}(p) = e^{(2\pi i/p)(a^{k+l})} = C_{k,l}(p),$$

where $k+n$ and $l-n$ are computed modulo $p-1$.

Lemma 9.1 *If C is skew-circulant, then CR is circulant, where R is the time-reversal matrix.*

Proof

$$(CR)_{k+n,l+n} = C_{k+n,N-l-n} = C_{k,N-l} = (CR)_{k,l}.$$

Theorem 9.2 *$F(p-1)C(p)F(p-1)$ is a diagonal matrix, where $F(p-1)$ is the $(p-1)$-point FT matrix.*

Proof By Lemma 9.1 and the convolution theorem, we have that $F(p-1)C(p)RF^{-1}(p-1)$ is a diagonal matrix. Since $R = F^2(p-1)$, we have the statement of the theorem.

Observe that the diagonal entries of $F(p-1)C(p)F(p-1)$ are also given by the $(p-1)$-point FT of the first column of $C(p)$.

Example 9.1 2 and 3 are generators of $U(\mathbf{Z}/5)$. The cyclic orderings of $U(5)$ given by the generators are

$$\pi_2 = \{1, \quad 2, \quad 2^2 = 4, \quad 2^3 = 3\},$$

$$\pi_3 = \{1, \quad 3, \quad 3^2 = 4, \quad 3^3 = 2\},$$

giving rise to the following corresponding core matrices.

$$\begin{bmatrix} w & w^2 & w^4 & w^3 \\ w^2 & w^4 & w^3 & w \\ w^4 & w^3 & w & w^2 \\ w^3 & w & w^2 & w^4 \end{bmatrix}, \quad \begin{bmatrix} w & w^3 & w^4 & w^2 \\ w^3 & w^4 & w^2 & w \\ w^4 & w^2 & w & w^3 \\ w^2 & w & w^3 & w^4 \end{bmatrix}.$$

9.3 Finite Fields

9.3.1 Properties of finite fields

In this section we will state, without proof, relevant properties of finite fields. These properties are derived in many books on number theory, including Ireland–Rosen (1980) [5].

Let K be a field with q elements. Denote the multiplicative group of nonzero elements of K by $U(K)$.

I. $U(K)$ is a cyclic group of order $q-1$.

Every element $a \in U(K)$ satisfies

$$a^{q-1} = 1.$$

II. The integer multiples of the identity form a subfield of K isomorphic to $\mathbf{Z}/p$ for some prime number p.

Henceforth, we will identify this subfield with the field $\mathbf{Z}/p$.

III. K is a finite-dimensional vector space over the subfield $\mathbf{Z}/p$.

Thus K has p^R elements, where R is the dimension of the vector space K over $\mathbf{Z}/p$.

IV. Any two fields having p^R elements are isomorphic. Denote by $GF(p^R)$ the (isomorphism class) field of order p^R.

9.3.2 Examples of finite fields

Denote by $K[x]$ the ring of polynomials in the variable x with coefficients in the field K; i.e., $K[x]$ is the set of all expressions of the form

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \qquad a_0, a_1, \ldots, a_n \in K,$$

for a nonnegative integer n, with arithmetic defined by the addition and multiplication of polynomials. $f(x) \in K[x]$ is said to be *reducible* over K if there exist nonconstant polynomials $g(x)$ and $h(x)$ in $K[x]$ such that

$$f(x) = g(x)h(x).$$

Otherwise, $f(x)$ is said to be *irreducible* over K.

Example 9.2 The polynomial $x^2 + 1$ is irreducible in $\mathbf{R}[x]$. However, it is reducible in $\mathbf{C}[x]$, since

$$x^2 + 1 = (x + i)(x - i), \qquad i = \sqrt{-1}.$$

The *degree* of a polynomial is equal to R if $f(x) = a_0 + a_1 x + \cdots + a_R x^R$, where $a_R \neq 0$. If $f(x)$ has degree R, then $K[x]/(f(x))$ denotes the ring

of all polynomials in $K[x]$ having degree $< R$, with addition given by polynomial addition and multiplication given by polynomial multiplication modulo $f(x)$.

Theorem 9.3 *For an irreducible polynomial $f(x) \in K[x]$, the quotient $K[x]/(f(x))$ is a field.*

Example 9.3 $\mathbf{R}[x]/(x^2+1)$ is a field isomorphic to the field of complex numbers $\mathbf{C}$, where the mapping

$$ax + b \rightarrow ai + b, \qquad a, b \in \mathbf{R}, \quad i = \sqrt{-1},$$

is an isomorphism.

We will now restrict our attention to finite fields.

Theorem 9.4 *For each integer $R \geq 1$, there exists an irreducible polynomial of degree R over $\mathbf{Z}/p$.*

Theorems 9.3 and 9.4 guarantee the existence of a finite field of order p^R for every R. Moreover, via Theorem 9.3, we can construct a field of order p^R whenever we find an irreducible polynomial of degree R over $\mathbf{Z}/p$.

Example 9.4 $x^2 + 1$ is irreducible over $\mathbf{Z}/3$. Thus

$$K = (\mathbf{Z}/3)[x]/(x^2+1)$$

is a field. $a = x + 1$ is a generator of $U(K)$.

$$\begin{array}{llll} a^0 = 1, & a = x+1, & a^2 = 2x, & a^3 = 2x+1, \\ a^4 = 2, & a^5 = 2x+2, & a^6 = x, & a^7 = x+2. \end{array}$$

9.4 Fourier Transform of Finite Fields

Let K be a field with p^R elements. Choose a generator a of $U(K)$ and order K as

$$0; a^0, \ldots, a^{p^R-2}.$$

We call such an ordering a *multiplicative ordering* and the corresponding basis of evaluation functions a *multiplicative basis.*

Every nonsingular $\mathbf{Z}/p$-bilinear form Ψ on $K \times K$ determines an FT presentation F_Ψ given by

$$F_\Psi f(b) = \sum_{c \in K} f(c) e^{(2\pi i/p)\Psi(c,b)}, \qquad b \in K.$$

By abuse of notation, we denote by F_Ψ the matrix of F_Ψ relative to a fixed multiplicative basis π. We can write

$$F_\Psi = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & 1 \\ \cdot & & & & \\ \cdot & & C(\Psi) & & \\ \cdot & & & & \\ 1 & & & & \end{bmatrix},$$

where $C(\Psi)$ is the *core* of F_Ψ relative to the multiplicative basis π.

To obtain a skew-circulant matrix as the core of the matrix F_Ψ, Ψ must reflect the cyclic nature of $U(K)$. To this end, define

$$\Psi(a,b) = h(ab), \qquad a,b \in K, \tag{9.1}$$

where h is any linear functional of the $\mathbf{Z}/p$-vector space K *onto* $\mathbf{Z}/p$. A direct computation shows that Ψ is a nonsingular $\mathbf{Z}/p$-bilinear form on $K \times K$ satisfying the important condition

$$\Psi(a,b) = \Psi(ab,1) = \Psi(1,ab), \qquad a,b \in K. \tag{9.2}$$

We will always assume that Ψ has this form. Below we will give two examples of h satisfying the desired onto property.

Using condition (9.2), we can prove that the core matrix is skew-circulant. Denote the k-th row l-th column entry of $C(\Psi)$ by $C_{k,l}(\Psi)$:

$$C_{k,l}(\Psi) = e^{(2\pi i/p)\Psi(a^k,a^l)} = e^{(2\pi i/p)h(a^{k+l})}.$$

Since $a^{p^R-1} = 1$, if $k' + l' \equiv k + l \bmod (p^R - 1)$, $h(a^{k+l}) = h(a^{k'+l'})$, and we have

$$C_{k,l}(\Psi) = C_{k',l'}(\Psi).$$

The remainder of this section is devoted to presenting examples of bilinear forms and their effects on implementations and applications.

9.4.1 Trace map

Let K be a field of order p^R. For $a \in K$, define

$$\mathrm{Tr}(a) = a + a^p + a^{p^2} + \cdots + a^{p^{R-1}}. \tag{9.3}$$

$\mathrm{Tr}(a)$ is called the *trace* of a. We list properties of Tr.

Let a, $b \in K$ and $\alpha \in \mathbf{Z}/p$.

1. Tr maps K onto $\mathbf{Z}/p$.
2. $\mathrm{Tr}(a+b) = \mathrm{Tr}(a) + \mathrm{Tr}(b)$.

3. $\mathrm{Tr}(\alpha a) = \alpha \mathrm{Tr}(a)$.

Properties 1–3 imply that Tr is a linear functional on the $\mathbf{Z}/p$-vector space K onto the subfield $\mathbf{Z}/p$.

Example 9.5 Let $K = (\mathbf{Z}/5)[x]/(x^2+2)$. $U(K)$ is generated by $a = x+1$. Multiplicatively ordered elements of K are given in Table 9.1.

Table 9.1 The Field $K = (\mathbf{Z}/5)[x]/(x^2+2)$

n	a^n	$\mathrm{Tr}(a^n)$	n	a^n	$\mathrm{Tr}(a^n)$
0	1	$1+1=2$	12	4	$4+4=3$
1	$x+1$	$a+a^5=2$	13	$4x+4$	$4\mathrm{Tr}(a)=3$
2	$2x+4$	$a^2+a^{10}=3$	14	$3x+1$	$4\mathrm{Tr}(a^2)=2$
3	x	$a^3+a^{15}=0$	15	$4x$	$4\mathrm{Tr}(a^3)=0$
4	$x+3$	$a^4+a^{20}=1$	16	$4x+2$	$4\mathrm{Tr}(a^4)=4$
5	$4x+1$	$a^5+a^1=2$	17	$x+4$	$4\mathrm{Tr}(a^5)=3$
6	3	$a^6+a^6=1$	18	2	$a^{18}+a^{18}=4$
7	$3x+3$	$3\mathrm{Tr}(a)=1$	19	$2x+2$	$2\mathrm{Tr}(a)=4$
8	$x+2$	$3\mathrm{Tr}(a^2)=4$	20	$4x+3$	$2\mathrm{Tr}(a^2)=1$
9	$3x$	$3\mathrm{Tr}(a^3)=0$	21	$2x$	$2\mathrm{Tr}(a^3)=0$
10	$3x+4$	$3\mathrm{Tr}(a^4)=3$	22	$2x+1$	$2\mathrm{Tr}(a^4)=2$
11	$2x+3$	$3\mathrm{Tr}(a^5)=1$	23	$3x+2$	$2\mathrm{Tr}(a^5)=4$

Define Ψ by

$$\Psi(a,b) = \mathrm{Tr}_K(ab), \qquad a,b \in K. \tag{9.4}$$

Ψ is a nondegenerate bilinear form from $K \times K$ onto $\mathbf{Z}/5$. We now have the presentation F_Ψ of the FT of K:

$$F_\Psi f(b) = \sum_{a \in K} f(a) e^{(2\pi i/p)\Psi(a,b)}, \qquad b \in K. \tag{9.5}$$

A meaningful property of the trace map involves concepts that we will not present here. However, we will state the property in a restricted setting that still serves our interest: We will relate the FT of two isomorphic fields.

A field of p^R elements can be realized in as many ways as the number of irreducible polynomials of degree R in $\mathbf{Z}/p[x]$. Although any two such realizations are isomorphic to each other, one realization may be more natural than another for a given application. Examples are found in coding theory and X-ray crystallography.

In the following theorem we consider two isomorphic fields K_1 and K_2. Denote the corresponding trace maps by Tr_{K_1} and Tr_{K_2}.

Theorem 9.5 *Let $\mathbf{Z}/p$ be a subfield of two isomorphic finite fields K_1 and K_2. If σ is a field isomorphism from K_1 onto K_2 that fixes the elements of $\mathbf{Z}/p$, then for $a_1 \in K_1$,*

$$\mathrm{Tr}_{K_1}(a_1) = \mathrm{Tr}_{K_2}(\sigma(a_1)).$$

A proof of the above theorem can be found in [5].

Example 9.6 Refer to the example above. Set $K_1 = K = (\mathbf{Z}/5)[x]/(x^2+2)$ and $K_2 = (\mathbf{Z}/5)[x]/(x^2+x+1)$. A group homomorphism σ that maps a generator of $U(K_1)$ onto a generator of $U(K_2)$ is a field isomorphism from K_1 onto K_2. The element $a_2 = 3x + 1 \in K_2$ generates $U(K_2)$. Set

$$\sigma(a_1) = a_2.$$

Then σ is a field isomorphism. However, σ does not fix $\mathbf{Z}/5$, since

$$\sigma(3) = \sigma(a_1^6) = a_2^6 = 2.$$

Define a field isomorphism $\tau : K_1 \to K_2$ by

$$\tau(a_1) = a_2^7.$$

Then

$$\tau(3) = \tau(a_1^6) = a_2^{18} = 3.$$

Since 3 is a generator of $U(K)$, τ fixes $\mathbf{Z}/5$.

Defining the bilinear form Ψ_2 by

$$\Psi_2(a_2, b_2) = \mathrm{Tr}_{K_2}(a_2 b_2), \qquad a_2, b_2 \in K_2,$$

we have

$$F_{\Psi_2} g(b_2) = \sum_{a_2 \in K_2} g(a_2) e^{(2\pi i/p)\mathrm{Tr}_{K_2}(a_2 b_2)}, \qquad b_2 \in K_2, \quad g \in L(K_2).$$

Since $\mathrm{Tr}_{K_2}(a_2 b_2) = \mathrm{Tr}_{K_1}(\tau^{-1}(a_2 b_2))$, ordering K_2 by $\tau(K_1)$, we have that the matrix of the FT of K_2 is the same as that of K_1.

The following observation is sometimes useful.

> Since Tr maps K onto $\mathbf{Z}/p$, there exists a_0 in K such that $\mathrm{Tr}(a_0) = 1$. The mapping $a \to \mathrm{Tr}(a_0 a)$, $a \in K$, is a linear functional of the $\mathbf{Z}/p$-vector space K onto $\mathbf{Z}/p$ that acts by the identity on the subfield $\mathbf{Z}/p$.

9.4.2 Evaluation map

In many applications, the field $K = GF(p^R)$ is given as a quotient,

$$K = (\mathbf{Z}/p)[x]/(q(x)), \tag{9.6}$$

where $q(x)$ is an irreducible polynomial over $\mathbf{Z}/p$ of degree R. A typical element in K is written as a polynomial

$$a(x) = a_0 + a_1 x + \cdots + a_{R-1} x^{R-1}, \qquad a_j \in \mathbf{Z}/p.$$

We can identify $a(x)$ with the R-tuple

$$i(a(x)) = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{R-1} \end{bmatrix},$$

which amounts to choosing the monomials $1, x, \ldots, x^{R-1}$ as a basis of K over $\mathbf{Z}/p$. In this case, it is more convenient to choose a nonsingular $\mathbf{Z}/p$-bilinear form on $K \times K$ that reflects the definition of K as a polynomial ring quotient. Define

$$e(a(x)) = a_0$$

and observe that e is a linear functional on the $\mathbf{Z}/p$-vector space K onto $\mathbf{Z}/p$. In fact, identifying $\mathbf{Z}/p$ with the constant polynomials in K, e acts by the identity mapping on the subfield $\mathbf{Z}/p$ of K. Define

$$\Theta(a(x), b(x)) = e(a(x)b(x)) \bmod q(x).$$

Θ is a nonsingular $\mathbf{Z}/p$-bilinear form on $K \times K$ satisfying (9.2).

Example 9.7 Set $K = (\mathbf{Z}/3)[x]/(x^2 + 1)$. For $a(x) = a_0 + a_1 x$ and $b(x) = b_0 + b_1 x$,

$$\Theta(a(x), b(x)) = e(a_0 b_0 + 2a_1 b_1 + (a_1 b_0 + a_0 b_1)x) = a_0 b_0 + 2a_1 b_1.$$

9.5 Factorization of Core Matrices

For a field K of order p^R, $U(K)$ is a cyclic group of order $p^R - 1$. Since $p^R - 1$ is a composite number, $U(K)$ has subgroups. Additive algorithms rely on the additive subgroup structure of the indexing set. Multiplicative subgroups of $U(K)$ lead in an analogous fashion to multiplicative algorithms. This idea was exploited by Agarwal and Cooley (1977) [1] to design the factorization algorithm for cyclic convolution.

Of the many additive algorithms, we will illustrate the analogues of the Cooley–Tukey and Good–Thomas algorithms. Throughout this section, we assume that Ψ is given as in (9.1),

$$\Psi(a, b) = h(ab).$$

Let $p^R - 1 = LM$. Then $U(K)$ has a subgroup of order L. In fact, if $a \in U(K)$ is a generator, then a^M generates a subgroup of order L. Let B be the group generated by a^M,

$$B = \{1, \ a^M, \ a^{2M}, \ \ldots, \ a^{(L-1)M}\}.$$

For the subgroup B in $U(K)$, the set $\{1, \ a, \ \ldots, \ a^{M-1}\}$ is a set of coset representatives,

$$U(K) = \bigcup_{k=0}^{M-1} a^k B,$$

and the union is disjoint.

Order $U(K)$ as follows: First order B by the powers of a^M. Then order $U(K)$ as $B; aB; \cdots; a^{M-1}B$. Relative to this ordering, we have the following decomposition of the core matrix.

$$\begin{bmatrix} C(0,0) & C(0,1) & C(0,2) & \cdots & C(0,M-1) \\ C(1,0) & & & & C(1,M-1) \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ C(M-1,0) & C(M-1,1) & & \cdots & C(M-1,M-1) \end{bmatrix},$$

where for $0 \leq \alpha, \beta \leq M-1$, $C(\alpha,\beta)$ is the $L \times L$ submatrix corresponding to the row index $a^\alpha B$ and the column index $a^\beta B$.

Denote the k-th row l-th column entry of $C(\alpha,\beta)$ by $C(\alpha,\beta)_{k,l}$. Then

$$C(\alpha,\beta)_{k,l} = e^{(2\pi i/p)h(a^{\alpha+kM}a^{\beta+lM})} = e^{(2\pi i/p)h(a^{\alpha+\beta+(k+l)M})}.$$

The following theorems are proved by direct computations.

Theorem 9.6 $C(\alpha,\beta) = C(\beta,\alpha)$.

Theorem 9.7 $C(\alpha,\beta)$ *is skew-circulant.*

Theorem 9.8 $C(\alpha,\beta) = C(\alpha',\beta')$ *if* $\alpha+\beta = \alpha'+\beta'$.

We have the following result in terms of cyclic shift matrices. Denote by $\mathcal{S}_L$ the inverse of the $L \times L$ cyclic shift matrix

$$\mathcal{S}_L = \begin{bmatrix} 0 & 1 & 0 & & \cdot & \cdot & 0 \\ 0 & 0 & 1 & 0 & & & 0 \\ \cdot & & & \cdot & & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & & \cdot & \cdot \\ \cdot & & & & & & 1 \\ 1 & 0 & \cdot & \cdot & \cdot & \cdot & 0 \end{bmatrix}.$$

Theorem 9.9 *If* $\alpha+\beta \equiv \alpha'+\beta' \bmod M$, *then*

$$C(\alpha,\beta) = \mathcal{S}_L^\gamma C(\alpha',\beta'),$$

where $\gamma M = \alpha+\beta-(\alpha'+\beta')$.

Proof Let $\alpha+\beta=\alpha'+\beta'+\gamma M$, for some positive integer γ. Since

$$\alpha+\beta+(k+l)M=\alpha'+\beta'+(\gamma+k+l)M,$$

we have

$$C(\alpha,\beta)_{k,l}=C(\alpha',\beta')_{\gamma+k,l}.$$

Example 9.8 Refer to Example 9.5, $K=(\mathbf{Z}/5)[x]/(x^2+2)$. Set

$$\Psi(a,b)=\mathrm{Tr}_K(ab), \qquad a,\ b\in K.$$

Consider the subgroup

$$B=\{1,\ a^3,\ a^6,\ a^9,\ a^{12},\ a^{15},\ a^{18},\ a^{21}\};$$

then $U(K)=B\cup aB\cup a^2B$. Ordering $U(K)$ by $B;aB;a^2B$ gives rise to the following core matrix.

$$\begin{bmatrix} C(0,0) & C(0,1) & C(0,2) \\ C(1,0) & C(1,1) & C(1,2) \\ C(2,0) & C(2,1) & C(2,2) \end{bmatrix} = \begin{bmatrix} C(0,0) & C(1,0) & C(2,0) \\ C(1,0) & C(1,1) & C(2,1) \\ C(2,0) & C(2,1) & C(2,2) \end{bmatrix}.$$

We will write out 3 of the submatrices explicitly. Set $w=e^{2\pi i/5}$.

$$C(0,0) = \begin{bmatrix} w^2 & 1 & w & 1 & w^3 & 1 & w^4 & 1 \\ 1 & w & 1 & w^3 & 1 & w^4 & 1 & w^2 \\ w & 1 & w^3 & 1 & w^4 & 1 & w^2 & 1 \\ 1 & w^3 & 1 & w^4 & 1 & w^2 & 1 & w \\ w^3 & 1 & w^4 & 1 & w^2 & 1 & w & 1 \\ 1 & w^4 & 1 & w^2 & 1 & w & 1 & w^3 \\ w^4 & 1 & w^2 & 1 & w & 1 & w^3 & 1 \\ 1 & w^2 & 1 & w & 1 & w^3 & 1 & w^4 \end{bmatrix},$$

$$C(1,0) = \begin{bmatrix} w^2 & w & w & w^3 & w^3 & w^4 & w^4 & w^2 \\ w & w & w^3 & w^3 & w^4 & w^4 & w^2 & w^2 \\ w & w^3 & w^3 & w^4 & w^4 & w^2 & w^2 & w \\ w^3 & w^3 & w^4 & w^4 & w^2 & w^2 & w & w \\ w^3 & w^4 & w^4 & w^2 & w^2 & w & w & w^3 \\ w^4 & w^4 & w^2 & w^2 & w & w & w^3 & w^3 \\ w^4 & w^2 & w^2 & w & w & w^3 & w^3 & w^4 \\ w^2 & w^2 & w & w & w^3 & w^3 & w^4 & w^4 \end{bmatrix},$$

$$C(2,2) = \begin{bmatrix} w & w & w^3 & w^3 & w^4 & w^4 & w^2 & w^2 \\ w & w^3 & w^3 & w^4 & w^4 & w^2 & w^2 & w \\ w^3 & w^3 & w^4 & w^4 & w^2 & w^2 & w & w \\ w^3 & w^4 & w^4 & w^2 & w^2 & w & w & w^3 \\ w^4 & w^4 & w^2 & w^2 & w & w & w^3 & w^3 \\ w^4 & w^2 & w^2 & w & w & w^3 & w^3 & w^4 \\ w^2 & w^2 & w & w & w^3 & w^3 & w^4 & w^4 \\ w^2 & w & w & w^3 & w^3 & w^4 & w^4 & w^2 \end{bmatrix} = \mathcal{S}_8 C(1,0).$$

Suppose now that $p^R - 1 = LM$, where L and M are relatively prime. Then there exist idempotents e_1 and e_2 with

$$e_1 \equiv 1 \bmod L, \quad e_1 \equiv 0 \bmod M,$$

$$e_2 \equiv 0 \bmod L, \quad e_2 \equiv 1 \bmod M.$$

Consider the subgroup D generated by a^{e_1},

$$D = \{1, \ a^{e_1}, \ a^{2e_1}, \ \ldots, \ a^{(L-1)e_1}\}.$$

We can write $U(K) = C \times D$, where

$$C = \{1, \ a^{e_2}, \ a^{2e_2}, \ \ldots, \ a^{(M-1)e_2}\}.$$

As before, we decompose the core matrix by ordering $U(K)$ by

$$D; a^{e_2}D; \cdots; a^{(M-1)e_2}D.$$

Denote the submatrices corresponding to $a^{\alpha e_2}D$ along the rows and $a^{\beta e_2}D$ along the columns by $T(\alpha, \beta)$. Then Theorems 9.4–9.6 hold for the submatrices $T(\alpha, \beta)$. In place of Theorem 9.7, we have

Theorem 9.10 *If* $\alpha + \beta \equiv \alpha' + \beta' \ mod \ M$, *then*

$$T(\alpha, \beta) = T(\alpha', \beta').$$

Proof Let $\alpha + \beta = \alpha' + \beta' + \gamma M$, for a positive integer γ.

$$(\alpha + \beta)e_2 + (k + l)e_1 = (\alpha' + \beta')e_2 + \gamma e_2 M + (k + l)e_1.$$

Since $e_2 \equiv 0 \bmod L$, $e_2 M \equiv 0 \bmod p^R - 1$, the theorem follows.

The above theorem implies that the core matrix has the following nested skew-circulant structure,

$$\begin{bmatrix} T(0,0) & T(1,0) & T(2,0) & \cdots & T(M-1,0) \\ T(1,0) & & & & T(0,0) \\ \vdots & & & & \vdots \\ T(M-1,0) & T(0,0) & & \cdots & T(M-2,0) \end{bmatrix},$$

where for $0 \leq \alpha, \beta \leq M - 1$, $T(\alpha, \beta)$ is an $L \times L$ skew-circulant matrix. A beautiful treatment of the theory of circulant and skew-circulant matrices and their applications can be found in [4].

9.6 Auslander–Feig–Winograd DFT

Suppose $K = GF(p^R)$ and h is a linear functional of the $\mathbf{Z}/p$-vector space K onto $\mathbf{Z}/p$. We will apply the results of the previous section to the case where the subgroup B of $U(K)$ is taken as $U(\mathbf{Z}/p)$. If a generates $U(K)$, then $U(\mathbf{Z}/p)$ is generated by a^β, where $\beta = \frac{p^R-1}{p-1}$. The core matrix now has the form

$$\begin{bmatrix} C(0,0) & C(0,1) & C(0,2) & \cdots & C(0,\beta-1) \\ C(1,0) & & & & C(1,\beta-1) \\ \vdots & & & & \vdots \\ C(\beta-1,0) & C(\beta-1,1) & \cdot & \cdots & C(\beta-1,\beta-1) \end{bmatrix},$$

where $C(r,s)$ is the skew-circulant matrix having the first column

$$\begin{bmatrix} v^{h(a^{r+s})} \\ v^{h(a^{r+s})\epsilon} \\ \vdots \\ v^{h(a^{r+s})\epsilon^{p-2}} \end{bmatrix}, \qquad v = e^{2\pi i/p}, \quad \epsilon = a^\beta.$$

As before, $C(r,s)$ depends only on $r+s \bmod p^R - 1$. Define $C_j = C(j,0)$. Denote by $C(p)$ the skew-circulant matrix whose first column is

$$\begin{bmatrix} v \\ v^\epsilon \\ \vdots \\ v^{\epsilon^{p-2}} \end{bmatrix},$$

and observe that the sum of elements in any row and the sum of elements in any column is -1. For $0 \le j < p^R - 1$, $h(a^j) \in \mathbf{Z}/p$. There are two possibilities. The first is that

$$h(a^j) = \epsilon^{\nu(j)} \in U(\mathbf{Z}/p), \qquad 0 \le \nu(j) < p-1,$$

and we have

$$C_j = C(p)\mathcal{S}^{-\nu(j)},$$

where $\mathcal{S}$ is the $p \times p$ cyclic shift matrix introduced in the previous section. The second possibility is that $h(a^j) = 0$, and we have

$$C_j = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix} = C(p) \begin{bmatrix} -1 & \cdots & -1 \\ \vdots & & \vdots \\ -1 & \cdots & -1 \end{bmatrix}.$$

For notational convenience, we set in this case $\nu(j) = \infty$ and define $\mathcal{S}^{\infty}$ by the condition $C_j = C(p)\mathcal{S}^{-\infty}$. We can now write

$$C(\Psi) = (I_\beta \otimes C(p))S,$$

where

$$S = \begin{bmatrix} \mathcal{S}^{-\nu(0)} & \mathcal{S}^{-\nu(1)} & \cdots & \mathcal{S}^{-\nu(\beta-1)} \\ \mathcal{S}^{-\nu(1)} & \mathcal{S}^{-\nu(2)} & \cdots & \mathcal{S}^{-\nu(\beta)} \\ \vdots & & & \vdots \\ \mathcal{S}^{-\nu(\beta-1)} & \mathcal{S}^{-\nu(\beta)} & \cdots & \mathcal{S}^{-\nu(2\beta-2)} \end{bmatrix}, \tag{9.7}$$

which is the statement of the Auslander–Feig–Winograd theorem.

References

[1] Agarwal, R.C. and Cooley, J.W. (1977), "New Algorithms for Digital Convolution," *IEEE Trans. ASSP* **ASSP-25**, 392–410.

[2] An, M. and Tolimieri, R. (1993), "Multiplicative Multidimensional Fourier transform Algorithms," preprint, Aware, Inc.

[3] Auslander, L., Feig, E., and Winograd, S. (1983), "New Algorithms for the Multiplicative Discrete Fourier transform," *IEEE Trans. ASSP* **ASSP-31**(2).

[4] Davis, P.J. (1979), *Circulant Matrices*, John Wiley and Sons, Inc., New York.

[5] Ireland and Rosen (1980), *A Classical Introduction to Modern Number Theory*, Springer-Verlag, New York.

[6] Nussbaumer, H. and Quandalle, P. (1979), "Fast Computation of Discrete Fourier Transforms Using Polynomial Transforms," *IEEE Trans. ASSP* **ASSP-27**, 169–181, April.

[7] Rader, C. (1968), "Discrete Fourier Transforms when the Number of Data Samples is Prime," *Proc. IEEE* **56**, 1107–1108.

[8] Tolimieri, R. (1986), "Multiplicative Characters and the Discrete Fourier transform," *Adv. Appl. Math.* **7**, 344–380.

[9] Tolimieri, R., An, M., Abdelatif, Y., Lu, C., Kechriotis, G. and Anupindi, N (1995), "Group Invariant Fourier Transform Algorithms," *Advances in Imaging and Electron Physics* **93**, 1–55.

[10] Vulis, M. (1989), "The Weighted Redundancy Transform," *IEEE Trans. ASSP* **ASSP-37**(11).

[11] Winograd, S. (1980), "Arithmetic Complexity of Computations," Presented at CBMS-NSF Regional Conf. Series in Appl. Math.

[12] —, (1978), "On Computing the Discrete Fourier transform," *Math. Comput.* **32**, 175–199.

Problems

1. The two generators for $U(\mathbf{Z}/7)$ are 5 and 3. Using one of the generators to index input and output of the 7-point Fourier transform, write the corresponding core matrix.

2. An $M \times M$ matrix C is called *circulant* if

$$C_{k,l} = C_{k',l'},$$

for $k-l \equiv k'-l' \bmod M$. Show that the core matrix of the 7-point FT obtained by indexing the input by successive powers of 5 and output by successive powers of 3 is circulant.

3. If an element $a \in G$ is a cyclic generator of G, then so is a^{-1}. Prove that in general, the corresponding core matrix is circulant if a is used to index the row(/column) and a^{-1} is used to index the column(/row) of the core matrix.

4. Give examples of finite fields of various orders.

5. Let K be a finite field of order p^R containing $\mathbf{Z}/p$.

   (1) For $a, b \in K$, show that $(a+b)^p = a^p + b^p$.

   (2) Show that an element $a \in K$ is an element of $\mathbf{Z}/p$ if and only if $a^p = a$.

   (3) If $a \in \mathbf{Z}/p$, then $Tr(a) = Ra$.

6. Set $K_1 = (\mathbf{Z}/3)[x]/(x^2+1)$ and $K_2 = (\mathbf{Z}/3)[x]/(x^2+x+2)$.

   (1) Find all the isomorphisms of K_1 and K_2. How many are there? Do any of the isomorphisms not fix $\mathbf{Z}/3$? What can one conclude about fields of order a power of 3?

   (2) Compute the trace of the elements of K_1. Using an isomorphism found in (1), find the trace of elements of K_2.

   (3) Write the core matrix corresponding to $U(K_1)$ and the bilinear form defined by the trace map.

(4) $\mathbf{Z}/3$ is a subfield of K_1. Consequently, $U(\mathbf{Z}/3)$ is a subgroup of $U(K_1)$. Using this subgroup, decompose the core matrix $U(K_1)$.

7. Derive the Auslander–Feig–Winograd DFT using the RTA and the Rader Field algorithm.

10

Implementation on RISC Architectures

10.1 Introduction

A wide variety of DFT and convolution algorithms have been designed to optimize computations with respect to the number of arithmetic operations, especially multiplications. Blahut (1985) [1] offers an excellent survey of many algorithms designed using this methodology. Today, with the rapid advance in VLSI technology and the availability of high-speed and inexpensive floating-point processors, the time required to carry out a fixed-point addressing operation or a floating-point addition can effectively be the same as that for the floating-point multiplication. Some advanced architectures have these functional units working in parallel, with multiple operations realized in one or a few cycles at the same time. Traditional algorithm design of trading multiplications for additions, therefore, is not only ineffective but can result in a significant decrease in performance.

In this chapter, we will concentrate on the RISC (reduced instruction set computer) architectures. In section 10.2, efficient algorithm design for RISC architectures will be discussed. In sections 10.3 and 10.4 implementation detail on the IBM RS/6000 and Intel i860 will be provided. Some of the work in this chapter has appeared in [7–10]. For detailed discussion of RISC architectures, see Dewar–Smosna (1990) [3].

The first computers were RISC machines. The development of complex instruction set computers (CICS) was a natural consequence of the need to program increasingly complex computations requiring hundreds of instructions and many addressing modes. To support these tasks, microcoded

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

control units evolved to provide a large instruction set for complicated multitasking. The modern RISC machine is based on an opposing philosophy. The control unit of a CISC is slow and consumes a large percentage of valuable chip area. RISC machines give up the programming advantage offered by the large instruction set of CISC machines and exploit the following two hardware advantages instead.

1. The basic compute cycle is essentially fixed in RISC machines, permitting highly optimized pipelining and other hardware techniques.
2. Valuable chip area is available for dedicating hardware floating-point adders, multipliers, large cache memory, and fixed-point addressing units.

These RISC features have the potential of significantly increasing computation performance, but only if new software design tools can be established.

10.2 Algorithms for RISC Architectures

In this section, algorithms for RISC machines will be studied relative to two abstract computational models (Granata 1990) [4], which are distinguished by the different levels of parallelism between the floating-point adder and multiplier:

- **Model I** The fixed-point unit and floating-point adder and multiplier all operate in parallel, and programmers have independent control of each.
- **Model II** The fixed-point unit and floating-point kernel can operate in parallel, but the floating-point adder and multiplier can operate in parallel (actually pipelined) only when using multiply–add instructions of the form $\pm a \pm b * c$.

It is clear that any algorithm designed for model II RISC is automatically legitimate for model I architectures. The benefit, however, of considering these models separately is derived from the possibility of exploiting the different levels of parallelism to achieve better performance.

A general RISC programming goal is that code should maximize parallelism of the floating-point adder and multiplier and minimize delays caused by address computation and data retrieval. To achieve this goal, instruction flow should be as static as possible in order to control the complex interactions of latencies and loadings. The tensor product provides a major tool in this direction by clustering computations of a given type and thus increasing the static nature of the code. Along these lines, it is important that we make sophisticated use of all the capabilities available on the RISC's addressing unit to unroll complex data shuffles as simple addressing

operations. The size of the instruction cache is an essential characteristic in determining the efficiency with which this can be carried out.

The floating-point adder and multiplier in model I operate in parallel and independently, implying that the number of additions and the number of multiplications required by an algorithm should be balanced and independent. Thus, a fast algorithm for this model, as far as arithmetic is concerned, would minimize $Max(M, A)$, where M denotes the number of multiplications and A denotes the number of additions. Conventional algorithm design strategies that reduce multiplications by increasing additions are not applicable for RISC machines, especially since most DFT and convolution calculations have more additions than multiplications. The ideal algorithm balances the number of additions and the number of multiplications.

For computational model II, the floating-point adder and multiplier operate in parallel only when multiplications and additions are connected in the format

$$\pm a \pm b * c.$$

Thus, a fast algorithm for this model would be one that minimizes $(M) + (A) + (MA)$, where MA denotes the number of *multiply–add* instructions given in (10.1). In general, the architecture of i860 is a RISC model I type with some restrictions on input operands that must be carefully handled for optimal performance. The IBM RS/6000 is a typical model II architecture. Some techniques have been developed by Linzer–Feig (1991) [6], Lu (1991) [7], Lu–Cooley–Tolimieri (1991, 1993) [8, 9] to write code based on this dependency relationship, techniques that at the same time reduce the number of additions. In most DSP algorithms, the total arithmetic cost depends only on the number of additions required.

In the subsequent subsections, we will describe implementation strategy for both of the RISC models. Specific examples of algorithm implementations will be provided in detail.

10.2.1 Implementation on model I RISC

Let F be a matrix and suppose that there exists an algorithm to implement the product $\mathbf{y} = F\mathbf{x}$, where $\mathbf{x}$ and $\mathbf{y}$ are input and output vectors. Assume that the arithmetic complexity of an algorithm to carry out the action F is (M, A) (M the number of multiplications, A the number of additions). The question is, How many computer cycles are required to implement this action on a model I RISC machine?

The number of computer cycles required can be as small as $Max(M, A)$ or as large as $M + A$. For the time being we ignore the number of computer cycles required for loading and storing data and assume that each operation takes one computer cycle. The following two examples show that both extremes are possible.

Example 10.1 Fully independent arithmetic.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & a & 0 & 0 \\ 0 & 0 & 0 & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & c \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

In this example the additions and multiplications are totally independent. The number of computer cycles required to implement this action is $Max(3,3) = 3$.

Example 10.2 Fully dependent arithmetic.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

In this example the additions and multiplications are totally dependent. The number of computer cycles required to implement this action is $M + A = 1 + 5 = 6$.

In the following example of the 5-point Fourier transform computation, we show the importance of having a suitable scheduling strategy for additions and multiplications on a model I RISC machine. This example will be modified in the next subsection to match a model II RISC architecture.

Example 10.3 The 5-point Fourier transform is defined as

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 \\ 1 & w^2 & w^4 & w & w^3 \\ 1 & w^3 & w & w^4 & w^2 \\ 1 & w^4 & w^3 & w^2 & w \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \qquad w = e^{2\pi i/5}.$$

A variant of the Winograd small FFT algorithm computes $\mathbf{y}$ by the factorization [8]

$$\mathbf{y} = HGHA\mathbf{x}, \tag{10.1}$$

where

$$\begin{aligned} H &= 1 \oplus \big(F(2) \otimes I_2\big), \\ G &= 1 \oplus \begin{bmatrix} c_1 & c_2 \\ c_2 & c_1 \end{bmatrix} \oplus \begin{bmatrix} s_1 & s_2 \\ s_2 & -s_1 \end{bmatrix}, \end{aligned}$$

where c_1, c_2, s_1, s_2 are real numbers given by

$$\begin{aligned} c_1 &= \cos(2\pi/5), & c_2 &= \cos(4\pi/5), \\ s_1 &= i\sin(2\pi/5), & s_2 &= i\sin(4\pi/5) \end{aligned}$$

and

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Direct implementation of the matrix operation A requires 8 additions. By interchanging the order of H and A, we have another variant of the algorithm,

$$\mathbf{y} = HGBH\mathbf{x}, \tag{10.2}$$

where

$$B = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The trade-off between the two algorithms is one multiplication for 4 additions. Implementation of the variant algorithm on a model I RISC machine can be scheduled as follows:

$a_{10} = m_6 + m_{79}$
$a_1 = x_1 + x_4$
$a_2 = x_1 - x_4$
$a_3 = x_2 + x_3$ / $m_1 = 2.0 * x_0$
$a_4 = x_2 - x_3$ / $m_2 = a_2 * s_1$
$a_5 = a_1 - m_1$ / $m_3 = a_4 * s_2$
$a_6 = a_3 - m_1$ / $m_4 = a_5 * c_1$
$a_7 = a_1 + a_3$ / $m_5 = a_6 * c_2$
$y_0 = x_0 + a_7$ / $m_6 = a_5 * c_2$
$a_8 = m_4 + m_5$ / $m_7 = a_6 * c_1$
$a_9 = m_2 + m_3$ / $m_8 = a_2 * s_2$
$a_{10} = m_6 + m_7$ / $m_9 = a_4 * s_1$
$a_{11} = m_8 - m_9$
$y_1 = a_8 + a_{10}$
$y_4 = a_8 - a_{10}$
$y_2 = a_9 + a_{11}$
$y_3 = a_9 - a_{11}$

In this example, the arithmetic complexity is $(M, A) = (9, 16)$. The number of computer cycles required to carry out this calculation is $Max(9, 16) =$

16. A direct implementation of the original Winograd 5-point DFT algorithm (Blahut, 1985) [1] on a model I RISC machine requires 17 = $Max(5,17)$ computer cycles. Trading 4 multiplications for one addition produces a more balanced computation, resulting in saving one machine cycle.

Example 10.4 Partially dependent arithmetic.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 \\ 0 & b & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 & 0 \\ 0 & 0 & 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 & e & 0 \\ 0 & 0 & 0 & 0 & 0 & f \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}.$$

We can implement this action by scheduling operations as follows:

$a_6 = x_1 + x_2$
$a_1 = x_0 + x_1$
$a_2 = x_0 + x_2 \quad / \; y_0 = a * a_1$
$a_3 = x_0 + x_3 \quad / \; y_1 = b * a_2$
$a_4 = x_0 + x_4 \quad / \; y_2 = c * a_3$
$a_5 = x_0 + x_5 \quad / \; y_3 = d * a_4$
$a_6 = x_1 + x_2 \quad / \; y_4 = e * a_5$
$y_5 = f * a_6$

In this example, the arithmetic complexity is $(6, 6)$ and the number of computer cycles required to carry out this calculation is 7. In practical applications, the operation of matrix F must be computed many times, and we can achieve the goal of $Max(M, A)$ computer cycles per operation of F. An optimal scheduling strategy on model I RISC architecture implements $(I_n \otimes F)$ in $n \times Max(M, A)$ computer cycles.

Example 10.5 Multiple operations $I_n \otimes F$.

$$\begin{bmatrix} y_0(n) \\ y_1(n) \\ y_2(n) \\ y_3(n) \\ y_4(n) \\ y_5(n) \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & 0 & 0 & 0 \\ 0 & b & 0 & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 & 0 \\ 0 & 0 & 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 & e & 0 \\ 0 & 0 & 0 & 0 & 0 & f \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0(n) \\ x_1(n) \\ x_2(n) \\ x_3(n) \\ x_4(n) \\ x_5(n) \end{bmatrix}.$$

We can achieve optimal performance by scheduling operations as follows.

$a_0 = x_0(i+1) + x_1(i+1)$
$a_0 = x_0(1) + x_1(1)$
do $i = 1, n-1$
$\quad a_1 = x_0(i) + x_2(i) \quad / \; y_0(i) = a * a_0$

$$
\begin{array}{ll}
a_2 = x_0(i) + x_3(i) & / \; y_1(i) = b * a_1 \\
a_3 = x_0(i) + x_4(i) & / \; y_2(i) = c * a_2 \\
a_4 = x_0(i) + x_5(i) & / \; y_3(i) = d * a_3 \\
a_5 = x_1(i) + x_2(i) & / \; y_4(i) = e * a_4 \\
a_0 = x_0(i+1) + x_1(i+1) & / \; y_5(i) = f * a_5
\end{array}
$$

end do

$$
\begin{array}{ll}
a_1 = x_0(n) + x_2(n) & / \; y_0(n) = a * a_0 \\
a_2 = x_0(n) + x_3(n) & / \; y_1(n) = b * a_1 \\
a_3 = x_0(n) + x_4(n) & / \; y_2(n) = c * a_2 \\
a_4 = x_0(n) + x_5(n) & / \; y_3(n) = d * a_3 \\
a_5 = x_1(n) + x_2(n) & / \; y_4(n) = e * a_4 \\
y_5(n) = f * a_5 &
\end{array}
$$

The scheduling techniques presented above require that data loads and stores take place while floating-point units are in action. Most modern RISC architectures provide large banks of registers and data cache on chip to handle data loads and stores with some restrictions. In the following sections, we will discuss scheduling techniques for specific machines in detail.

10.2.2 Implementation on model II RISC

Efficient implementation on RISC model II machines must

- Exploit the dependency between additions and multiplications of the form $\pm a \pm (b * c)$.
- Minimize $M + A + MA$.

Example 10.6 We derive a variant of the 5-point Winograd DFT algorithm that matches the architecture of model II RISC.

Combining the matrix operations of G and B in (10.2), we have

$$\mathbf{y} = HG_BH\mathbf{x}, \tag{10.3}$$

where

$$G_B = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & c_1 & c_2 & 0 & 0 \\ 1 & c_2 & c_1 & 0 & 0 \\ 0 & 0 & 0 & s_1 & s_2 \\ 0 & 0 & 0 & s_2 & -s_1 \end{bmatrix}.$$

The computation

$$\begin{bmatrix} 1 & c_1 & c_2 \\ 1 & c_2 & c_1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}$$

is of the form

$$v_0 + c_1 * v_1 + c_2 * v_2,$$

$$v_0 + c_2 * v_1 + c_1 * v_2,$$

which can be implemented using multiply–add instructions as

$$(v_0 + c_1 * v_1) + c_2 * v_2,$$

$$(v_0 + c_2 * v_1) + c_1 * v_2.$$

The operation of

$$\begin{bmatrix} s_1 & s_2 \\ s_2 & -s_1 \end{bmatrix} \begin{bmatrix} v_3 \\ v_4 \end{bmatrix}$$

is

$$s_1 * v_3 + s_2 * v_4,$$

$$s_2 * v_3 - s_1 * v_4,$$

which can be rewritten as

$$s_2 * (v_4 + t_s * v_3),$$

$$s_2 * (v_3 - t_s * v_4),$$

where $t_s = s_1/s_2$.

The multiplication of s_2 above can be incorporated in the stage of H operation. Implementation of (10.3) on RISC model II can be scheduled as follows.

$a_1 = x_1 + x_4$
$a_2 = x_1 - x_4$
$a_3 = x_2 + x_3$
$a_4 = x_2 - x_3$
$a_5 = x_0 + a_1$
$ma_1 = x_0 + c_1 * a_1$
$ma_2 = x_0 + c_2 * a_1$
$x_0 = a_5 + a_3$
$ma_3 = ma_1 + c_2 * a_3$
$ma_4 = ma_2 + c_1 * a_3$
$ma_5 = a_4 + t_s * a_2$
$ma_6 = a_2 - t_s * a_4$
$y_1 = ma_3 + s_2 * ma_5$
$y_4 = ma_3 - s_2 * ma_5$
$y_2 = ma_4 + s_2 * ma_6$
$y_3 = ma_4 - s_2 * ma_6$

In this example, the arithmetic complexity is $(10, 16)$, and the number of required computer cycles is $A + MA = 16$.

10.3 Implementation on the IBM RS/6000

Much of the work in this section was carried out in collaboration with J. Cooley during the summer of 1990 at IBM [8, 9].

One of the most notable features of the IBM RS/6000 architecture is the separation of components of the processor into functional units. There are three major units: *fixed-point, floating-point*, and *branch*. Each of these components can process instructions in parallel, with the branch unit in overall control and responsible for the integrity of program execution. The branch unit is capable of sending out two instructions per cycle; one to the fixed-point unit and one to the floating-point unit. For optimum system performance, the branch unit should send out one fixed-point and one floating-point instruction in each cycle, and the fixed-point unit and the floating-point unit should execute one operation in each cycle.

Another distinguishing feature of the IBM RS/6000 architecture is the emphasis on floating-point performance. It can perform a 64-bit multiply or add in one machine cycle. A 64-bit multiply-and-add operation can be implemented in one cycle if they are arranged in the format $(\pm a \pm b * c)$. Floating-point performance is in very many cases governed by fixed-point performance. For instance, the Good–Thomas prime factor FFT algorithm requires a significant number of address computations involving fixed-point multiplication and addition for every data load and store. Each fixed-point add requires one cycle, and each fixed-point multiply requires three to five cycles.

The following example shows the possible parallelism among three functional units.

Example 10.7

```
do i=1, n-1
        y1(i) = a11*x1(i) + a12*x2(i) + b1
        y2(i) = a21*x1(i) + a22*x2(i) + b2
end do
```

After initialization the following code updates the arrays $y1$ and $y2$:

```
lp:   lfdu    fp0, x1(i)          #load and update register
      fma     fp3, fp1, a11, b1   #floating-point multiply–add
      lfdu    fp2, x2(i)          # another load and update
      fma     fp4, fp2, a22, b2   #floating-point multiply–add
      fma     fp3, fp2, a12, fp4  #floating-point multiply–add
      stfdu   fp3, y1(i)          #store and update register
      fma     fp4, fp1, a21, fp4  #floating-point multiply–add
      stfdu   fp4, y2(i)          #store and update register
      bc      lp                  #branch and count
```

Note that there is a total of nine instructions in the loop, consisting of four load and store instructions that can be executed in the fixed-point

unit, four floating-point multiply–add instructions that can be executed in the floating-point unit, and one branch instruction that can be executed in the branch unit. Since the multiply–add instruction performs two floating-point operations, these nine instructions result in the execution of thirteen operations. However, parallel implementation will require only four machine cycles for each iteration of the loop.

As stated above, the IBM RS/6000 processor is capable of executing five operations each cycle: two by branch unit (instruction-cache unit ICU), one by fixed-point unit (FXU), and two by floating-point unit (FPU) if multiply–add is considered as two operations. Delays between instructions can occur due to off-chip communication and the pipelined nature of the floating-point unit. For example, in executing a load instruction there is a hold-off of one cycle because the data must be fetched from data cache. Additional delay occurs since an overhead of 2 cycles is required to initiate the computation cycle.

The compiler of the IBM RS/6000 employs highly optimized instruction scheduling techniques to reduce these run-time delays. The 13-point Fourier transform in Example 10.8 shows how the IBM RS/6000 FORTRAN compiler rearranges and schedules operations to reduce delays.

Example 10.8 13-point DFT FORTRAN program and its IBM RS/6000 compiled assembly code.

```
 3 |            SUBROUTINE FT(X,Y,A,B,M)
 4 |C ******************************************
 5 |C MULTIPLICATIVE 13-POINT FT ALGORITHM
 6 |C VARIANT 4 WITH FURTHER FACTORIZATION
 7 |C THIS ALGORITHM CAN BE FOUND IN
 8 |C "ALGORITHMS FOR DISCRETE FOURIER TRANSFORM"
 9 |C CHAPTER 9, SPRINGER 1989
10 |C COMPLEX INPUT DATA IN ARRAY X,Y.
11 |C COMPLEX OUTPUT DATA IN ARRAY A,B.
12 |C ******************************************
13 |              PARAMETER(N=13)
14 |              REAL X(0:N-1,M),Y(0:N-1,M)
15 |              REAL A(M,0:N-1),B(M,0:N-1)
16 |C
17 |              DATA WR1, WR2 /-0.103315417, -0.519484792/
18 |              DATA WR3, WR4 /-0.6627733, 2.65109299/
19 |              DATA WR5, WR6 /0.726109417, 0.3081685/
20 |              DATA WI1, WI2 /1.941883855, 3.43890578/
21 |              DATA WI4, WI5 /-2.770912552, 4.14811571/
22 |              DATA WI6, WS0 /0.20378874, -0.168946715/
23 |              DATA WS1, WS2 /1.546149564, -1.1743318/
24 |              DATA WS3, WS4 /-0.96041485, 0.144528999/
25 |              DATA WL0, WL1 /0.285179429, -2.609876432/
```

```
26 |              DATA WL3, WL4 /1.62116534, -0.243962704/
27 |              DATA WS5, WS6 / 0.6957006, -0.34399223/
28 |              DO J=1, M
29 |              R0=X(1,J)+X(12,J)
30 |              R6=X(1,J)-X(12,J)
31 |              R1=X(2,J)+X(11,J)
32 |              R7=X(2,J)-X(11,J)
33 |              R2=X(4,J)+X(9,J)
34 |              R8=X(4,J)-X(9,J)
35 |              R3=X(8,J)+X(5,J)
36 |              R9=X(8,J)-X(5,J)
37 |              R4=X(3,J)+X(10,J)
38 |              R10=X(3,J)-X(10,J)
39 |              R5=X(6,J)+X(7,J)
40 |              R11=X(6,J)-X(7,J)
41 |C
42 |              A(J,0)=X(0,J)+R0+R1+R2+R3+R4+R5
43 |              S0=R0+R3
44 |              S3=R0-R3
45 |              S1=R1+R4
46 |              S4=R1-R4
47 |              S2=R2+R5
48 |              S5=R2-R5
49 |              R0=S2+S0*WR1+S1*WR2
50 |              R1=S1+S0*WR2+S2*WR1
51 |              R2=S0+S1*WR1+S2*WR2
52 |              S0=S5+S3*WR4+S4*WR5
53 |              S1=S4+S3*WR5-S5*WR4
54 |              S2=S3-S4*WR4-S5*WR5
55 |              R3=X(0,J)+R0*WR3-S0*WR6
56 |              R4=X(0,J)+R1*WR3-S1*WR6
57 |              R5=X(0,J)+R2*WR3-S2*WR6
58 |              R0=X(0,J)+R0*WR3+S0*WR6
59 |              R1=X(0,J)+R1*WR3+S1*WR6
60 |              R2=X(0,J)+R2*WR3+S2*WR6
61 |C
62 |              S0=R9-R6
63 |              S1=R10-R7
64 |              S2=R11-R8
65 |              S3=S2+S0*WI4+S1*WI5
66 |              S4=S1+S0*WI5-S2*WI1
67 |              S5=S0-S1*WI1-S2*WI2
68 |              S0=R8+R6*WS0+R7*WS1
69 |              S1=R7+R6*WS1+R8*WS3
70 |              S2=R6+R7*WS3+R8*WS4
```

```
 71 |                R6=R11+R9*WL3+R10*WL4
 72 |                R7=R10+R9*WL4-R11*WL0
 73 |                R8=R9-R10*WL0-R11*WL1
 74 |                R9=R6-WS6*S3
 75 |                R10=R7-WS6*S4
 76 |                R11=R8-WS6*S5
 77 |                R6=S0+WI6*S3
 78 |                R7=S1+WI6*S4
 79 |                R8=S2+WI6*S5
 80 |C
 81 |                S0=Y(1,J)+Y(12,J)
 82 |                S6=Y(1,J)-Y(12,J)
 83 |                S1=Y(2,J)+Y(11,J)
 84 |                S7=Y(2,J)-Y(11,J)
 85 |                S2=Y(4,J)+Y(9,J)
 86 |                S8=Y(4,J)-Y(9,J)
 87 |                S3=Y(8,J)+Y(5,J)
 88 |                S9=Y(8,J)-Y(5,J)
 89 |                S4=Y(3,J)+Y(10,J)
 90 |                S10=Y(3,J)-Y(10,J)
 91 |                S5=Y(6,J)+Y(7,J)
 92 |                S11=Y(6,J)-Y(7,J)
 93 |C
 94 |                B(J,0)=Y(0,J)+S0+S1+S2+S3+S4+S5
 95 |                SS0=S0+S3
 96 |                SS3=S0-S3
 97 |                SS1=S1+S4
 98 |                SS4=S1-S4
 99 |                SS2=S2+S5
100 |                SS5=S2-S5
101 |                S0=SS2+SS0*WR1+SS1*WR2
102 |                S1=SS1+SS0*WR2+SS2*WR1
103 |                S2=SS0+SS1*WR1+SS2*WR2
104 |                SS0=SS5+SS3*WR4+SS4*WR5
105 |                SS1=SS4+SS3*WR5-SS5*WR4
106 |                SS2=SS3-SS4*WR4-SS5*WR5
107 |                S3=Y(0,J)+S0*WR3-SS0*WR6
108 |                S4=Y(0,J)+S1*WR3-SS1*WR6
109 |                S5=Y(0,J)+S2*WR3-SS2*WR6
110 |                S0=Y(0,J)+S0*WR3+SS0*WR6
111 |                S1=Y(0,J)+S1*WR3+SS1*WR6
112 |                S2=Y(0,J)+S2*WR3+SS2*WR6
113 |C
114 |                SS0=S9-S6
115 |                SS1=S10-S7
```

```
116 |                SS2=S11-S8
117 |                SS3=SS2+SS0*WI4+SS1*WI5
118 |                SS4=SS1+SS0*WI5-SS2*WI1
119 |                SS5=SS0-SS1*WI1-SS2*WI2
120 |                SS0=S8+S6*WS0+S7*WS1
121 |                SS1=S7+S6*WS1+S8*WS3
122 |                SS2=S6+S7*WS3+S8*WS4
123 |                S6=S11+S9*WL3+S10*WL4
124 |                S7=S10+S9*WL4-S11*WL0
125 |                S8=S9-S10*WL0-S11*WL1
126 |                S9=S6-WS6*SS3
127 |                S10=S7-WS6*SS4
128 |                S11=S8-WS6*SS5
129 |                S6=SS0+WI6*SS3
130 |                S7=SS1+WI6*SS4
131 |                S8=SS2+WI6*SS5
132 |C
133 |                A(J,1) =R0-WS2*S6
134 |                A(J,12)=R0+WS2*S6
135 |                B(J,1) =S0+WS2*R6
136 |                B(J,12)=S0-WS2*R6
137 |                A(J,2) =R1-WS2*S7
138 |                A(J,11)=R1+WS2*S7
139 |                B(J,2) =S1+WS2*R7
140 |                B(J,11)=S1-WS2*R7
141 |                A(J,4) =R2-WS2*S8
142 |                A(J,9) =R2+WS2*S8
143 |                B(J,4) =S2+WS2*R8
144 |                B(J,9) =S2-WS2*R8
145 |                A(J,8) =R3-WS5*S9
146 |                A(J,5) =R3+WS5*S9
147 |                B(J,8) =S3+WS5*R9
148 |                B(J,5) =S3-WS5*R9
149 |                A(J,3) =R4-WS5*S10
150 |                A(J,10)=R4+WS5*S10
151 |                B(J,3) =S4+WS5*R10
152 |                B(J,10)=S4-WS5*R10
153 |                A(J,6) =R5-WS5*S11
154 |                A(J,7) =R5+WS5*S11
155 |                B(J,6) =S5+WS5*R11
156 |                B(J,7) =S5-WS5*R11
157 |                END DO
158 |                RETURN
159 |                END
```

```
>>>>>> OBJECT SECTION <<<<<<<<<<<<<<<<<<<<
                        CL. 0;
29 | 000188          1       LFS     fp2 = x(r3,56)
29 | 00018C          1       LFS     fp0 = x(r3,100)
29 | 000190          1       AFS     fp24= fp2, fp0
31 | 000194          0       LFS     fp1 = x(r3,60)
30 | 000198          1       SFS     fp22= fp2, fp0
31 | 00019C          0       LFS     fp0 = x(r3,96)
31 | 0001A0          1       AFS     fp16= fp1, fp0
33 | 0001A4          0       LFS     fp2 = x(r3,68)
32 | 0001A8          1       SFS     fp21= fp1, fp0
33 | 0001AC          0       LFS     fp0 = x(r3,88)
33 | 0001B0          1       AFS     fp19= fp2, fp0
35 | 0001B4          0       LFS     fp1 = x(r3,84)
34 | 0001B8          1       SFS     fp20= fp2, fp0
35 | 0001BC          0       LFS     fp0 = x(r3,72)
35 | 0001C0          1       AFS     fp23= fp1, fp0
37 | 0001C4          0       LFS     fp18= x(r3,64)
36 | 0001C8          1       SFS     fp0 = fp1, fp0
37 | 0001CC          0       LFS     fp3 = x(r3,92)
43 | 0001D0          1       AFS     fp1 = fp24, fp23
39 | 0001D4          0       LFS     fp14= x(r3,76)
37 | 0001D8          1       SFS     fp17= fp18, fp3
39 | 0001DC          0       LFS     fp13 = x(r3,80)
38 | 0001E0          1       SFS     fp3 = fp18, fp3
42 | 0001E4          0       LFSU    fp2,r3= x(r3,52)
39 | 0001E8          1       AFS     fp18= fp14, fp13
42 | 0001EC          1       AFS     fp15= fp2, fp24
40 | 0001F0          1       SFS     fp13= fp14, fp13
42 | 0001F4          1       AFS     fp15= fp16, fp15
44 | 0001F8          1       SFS     fp24= fp24, fp23
42 | 0001FC          1       AFS     fp15= fp19, fp15
42 | 000200          2       AFS     fp15= fp23, fp15
45 | 000204          1       AFS     fp23= fp16, fp17
42 | 000208          1       AFS     fp15= fp17, fp15
46 | 00020C          1       SFS     fp17= fp16, fp17
47 | 000210          1       AFS     fp16= fp19, fp18
42 | 000214          1       AFS     fp15= fp15, fp18
48 | 000218          1       SFS     fp19= fp19, fp18
49 | 00021C          1       FMA     fp14= fp16, fp1, fp31
42 | 000220          1       STFSU   r5,x(r5,4)=fp15
50 | 000224          1       FMA     fp18= fp23, fp1, fp30
51 | 000228          1       FMA     fp15= fp1, fp31, fp23
49 | 00022C          1       FMA     fp1 = fp14, fp23, fp30
50 | 000230          1       FMA     fp23= fp18, fp31, fp16
```

```
52 | 000234          1      FMA     fp14= fp19, fp24, fp29
51 | 000238          1      FMA     fp18= fp15, fp16, fp30
53 | 00023C          1      FMA     fp16= fp17, fp24, fp28
54 | 000240          1      FNMS    fp15= fp24, fp29, fp17
52 | 000244          1      FMA     fp24= fp14, fp17, fp28
55 | 000248          1      FMA     fp1 = fp2, fp1, fp27
56 | 00024C          1      FMA     fp23= fp2, fp23, fp27
57 | 000250          1      FMA     fp2 = fp2, fp18, fp27
55 | 000254          1      FNMS    fp18= fp1, fp24, fp26
53 | 000258          1      FNMS    fp17= fp16, fp29, fp19
54 | 00025C          1      FNMS    fp19= fp15, fp19, fp28
58 | 000260          1      FMA     fp1 = fp1, fp24, fp26
55 | 000264          1      STFL    #SPILL14(r1,200)=fp18
56 | 000268          1      FNMS    fp18= fp23, fp26, fp17
58 | 00026C          1      STFL    #SPILL11(r1,176)=fp1
59 | 000270          1      FMA     fp1 = fp23, fp17, fp26
56 | 000274          1      STFL    #SPILL13(r1,192)=fp19
57 | 000278          1      FNMS    fp18= fp2, fp26, fp19
59 | 00027C          1      STFL    #SPILL10(r1,168)=fp1
60 | 000280          1      FMA     fp1 = fp2, fp19, fp26
57 | 000284          1      STFL    #SPILL12(r1,184)=fp18
62 | 000288          1      SFS     fp2 = fp0, fp22
60 | 00028C          1      STFL    #SPILL9(r1,160)=fp1
...
...
...
...
```

FORTRAN code implementing the 13-point Fourier transform algorithm [8, 9] that embeds all the multiplications within the multiply–add operations has been written. It has a total of 174 floating-point operations, 54 additions, and 120 multiply–add operations. The compiler-generated assembly code for the 13-point Fourier transform kernel from line 276 to line 532 shows that most of the operations have been rescheduled to achieve optimum performance of the floating-point unit. All the data-loading operations have been carried out by the ICU while the floating-point unit is in action except for the first two loads in line 276 and 277, which are the two operands of the first floating-point addition. At line 306 or FORTRAN code line 42 we have a one-cycle delay, since the result of one floating-point addition was needed as one of the operands of the next addition. The rest of the 173 floating-point operations have all been scheduled to execute in one cycle per operation. On the IBM RS/6000 processor each floating-point store operation requires one machine cycle. A total of 212 cycles was actu-

ally used for the computation with some delays occurring when the number of temporary variables exceeds the number of floating-point registers.

The RS/6000 has 32 general registers and 32 floating-point registers. Optimal performance is achieved when loads are interspersed with floating-point operations. However, when the number of temporary variables exceeds the number of registers, extra stores and loads (spills) are required, causing an increase in run time, as seen in the 13-point Fourier transform program.

In general, the RS/6000 compiler does a very good job in generating optimized execution code. As to be expected, special hand-tailored FORTRAN code based on knowledge of the compiler and architecture can improve performance.

Timing results of some prime size routines are listed in Table 10.1.

Table 10.1 Timing on RS/6000 (540/30MHz)

Size	CPUTIME	CPUTIME/$N \log_2 N$
11	5.97 μs	0.157 μs
13	7.58 μs	0.158 μs
17	13.24 μs	0.191 μs

$\mu s = 10^{-6}$sec

10.4 Implementation on the Intel i860

The dedicated hardware of Intel i860 includes a parallel RISC core and floating-point processing units. The Intel i860 is a high-performance RISC processor capable of 80 megaflops of single-precision floating-point throughput. The floating-point processing units include a floating-point adder and a floating-point multiplier that can operate in parallel. Operands are loaded from data cache and main memory by a separate fixed-point addressing unit that operates in parallel with the floating-point units. In this sense the i860 is a typical type I RISC machine. However, serious restrictions are placed on ideal parallel operation, which must be accounted for in coding. Parallel addition and multiplication requires that 6 operands be specified in a single instruction cycle. The pipelining hardware limits the number of floating-point registers that can be used as a source or destination in a single instruction cycle to 3. This forces code to be written to call on the output of either the adder or the multiplier (or temporary storage registers in the floating-point units).

In the pipelined mode, each floating-point addition and each single-precision multiplication is split into three stages. The double-precision multiplication is decomposed into four stages. Complex pipelines can be set up in which the adder or multiplier feed operands to each other. This so-called

dual operation is a major factor in avoiding operand bottleneck. There are 31 dual operation patterns in the i860. Every pattern computes addition and multiplication in parallel with different operands. Using these 31 combinatorial instructions, the i860 can effectively execute two floating-point operations per cycle.

Two of the 31 dual operation patterns are shown in Figure 10.1. We will demonstrate later how to use them in vector dot product and vector scalar addition.

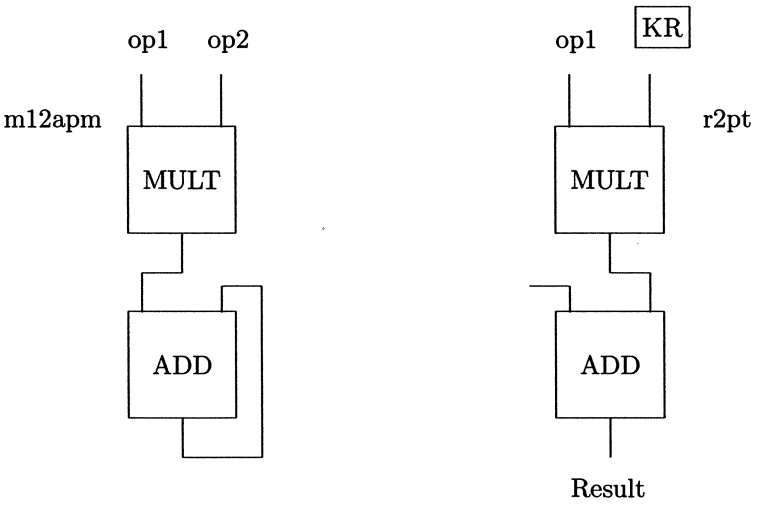


Figure 10.1 Two of the 32 pipelined instructions

The greatest obstacle to performance in the dual operation mode is that there are only two input operands. To achieve the optimum performance of two floating-point operations per cycle, one has to arrange the order of calculations so that the third and fourth operands of the addition and multiplication will be fed back from the output of the adder or the multiplier. In the case of a nonconstant vector computation, the dual operation could become a total failure. As an example, in the form

$$y(i) = v(i) * u(i) \pm x(i) \quad \text{or} \quad y(i) = x(i) * (u(i) \pm v(i)),$$

each operation needs three operands. The i860 would be unable to perform two operations in one cycle. In the case of DFT and convolution computations, special constant registers can be called on to schedule two operations per cycle.

In digital signal processing, most computations involve multiplication of a vector with a precalculated constant coefficient and addition with another vector. In the following example we will show some basic techniques in i860 programming.

Example 10.9

do $i = 1, n$
$\quad y(i) = b * x(i) + a$
end do

Here a and b are constants. We assume that n is a multiple of 8 and that x, y fit the data cache. The assembly code is as follows after initialization: Note that loading and storing are hidden so that the algorithm reaches the ideal of one add and one multiply in each cycle. There are a few points worth emphasizing. The data should be loaded at least two steps earlier than it is used. Otherwise, there will be a one to two cycle penalty. We also notice that the constant b is loaded into a constant register to serve as one operand. Another technique is the use of the delayed slot of the loop branch instruction. The loop branch instruction takes two cycles to complete. By inserting an extra operation in the delayed slot, the whole loop, which contains 8 additions and 8 multiplications, takes only 8 cycles.

In practice, a small DFT routine is going to repeat hundreds of times, making it ideal for pipelining. The example of the 5-point DFT given in Example 10.6 can be implemented on an i860 processor as follows: Observe that there are 4 constants in the implementation, we split the implementation into 2 steps. In the first step, c_1, c_2 are loaded into the constant registers: a_1, a_2, a_3, a_4, a_5, y_0, ma_1, ma_2, ma_3, ma_4 are computed: $y_0, a_2, a_4, ma_3, ma_4$ overwrite x_0, x_1, x_2, x_3, x_4. In the second step, we load a_2, a_4, ma_3, ma_4 into the registers and compute $ma_5, ma_6, y_1, y_2, y_3, y_4$. At this stage, t_s, s_2 will be stored in the constant registers: y_1, y_2, y_3, y_4 will finally replace a_2, a_4, ma_3, ma_4. The detailed assembly code is lengthy and will not be included in this text. The assembly code takes 32 cycles to complete each 5-point FFT. Here we consider the general complex pairs, so the count doubles the result of Example 10.6.

```
        fld.q       16(Xptr)++,Xold   //pre-load 4 elements of x(i)
                                      //auto increase the ptr.
        r2pt.ss     b, f0, f0         //load b into cons. register
        pfmul.ss    a, x1old,f0       //fill the pipeline
        d.pfmul.ss  a, x2old, f0      //switch to dual instruction
        fld.q       16(Xptr)++,Xnew   //x1,x2,x3,x4 are consecutive
        d.pfmul.ss  a, x3old, f0      //last element to pipeline
        nop
lp:     d.r2p1.ss   a, x4old, y1old   //start the lp
        nop
        d.r2p1.ss   a, x1new, y2old
        fld.q       16(Xptr)++,Xold   //load 4 x's for next
        d.r2p1.ss   a, x2new, y3old
        nop
        d.r2p1.ss   a, x3new, y4old   //complete one set of y's
```

```
fst.q       y1old 16(Yptr)++  // store y's in cache
d.r2p1.ss   a, x4new, y1new
nop
d.r2p1.ss   a, x1old, y2new
fld.q       16(Xptr)++,Xnew   // load Xnew for next lp
d.r2p1.ss   a, x2old, y3new   //complete one set of y's
bla         -1, lp-count, lp  // lp branch instruction
d.r2p1.ss   a, x3old, y4new   //delayed slot
fst.q       ynew 16(Yptr)++   // store y's in cache
```

References

[1] Blahut, R.E. (1985), *Fast Algorithms For Digital Signal Processing*, Addison-Wesley, Reading, MA.

[2] Bogoch, S., Bason, I., Williams, J., and Russell, M. (1990), "Supercomputers Get Personal," *BYTE Magazine*, 231–237.

[3] Dewar, R.B. and Smosna, M. (1990), *Microprocessors: A Programmer's View*, McGraw-Hill Publishing Co., New York.

[4] Granata, J.A. (1990), *The Design of Discrete Fourier Transform and Convolution Algorithms For RISC Architectures*, Ph.D. dissertation, the City University of New York.

[5] Hennessy, J.L. (1984), "VLSI Processor Architecture," *IEEE Computers* **C-33**, 1221–1246.

[6] Linzer, E. and Feig, E. (1991), "Implementation of Efficient FFT Algorithms on Fused Multiply–Add Architectures," to appear.

[7] Lu, C. (1991), "Implementation of 'Multiply-Add' FFT Algorithms for Complex and Real Data Sequences," *Proceeding of IEEE International Conference on Circuits and Systems*, Singapore.

[8] Lu C., Cooley, J.W., and Tolimieri, R. (1993),"FFT Algorithms for Prime Transform Sizes and Their Implementations on VAX, IBM 3090VF and RS/6000," *IEEE Trans. Signal Processing* **41 (2)**, February.

[9] — (1991), "Variants of the Winograd Multiplicative FFT Algorithms and Their Implementation on RS/6000," *Proceedings ICASSP-91*, Toronto.

[10] Lu, C., An, M., Qian, Z., and Tolimieri, R. (1992), "Small FFT module Implementation on the Intel i860 Processor," *Proc. ICSPAT*, November, 2–5, Cambridge, MA.

[11] Margulis, N. (1990), *i860 Microprocessor Architecture*, McGraw-Hill Publishing Co., New York.

[12] Patterson, D.A. (1985), "Reduced Instruction Set Computers," *Communications of the ACM* **28**(1), 8–21.

[13] Patterson, D.A. and Sequin, C.H. (1981), "RISC I: A Reduced Instruction Set VLSI Computer," *Proc. 8th Internat. Sympos. Computer Architectures* **ACM**, 443–457.

[14] — (1982), "A VLSI RISC," *IEEE Computer Mag.*, September, 8–22.

[15] Radin, G. (1982), "The 801 Minicomputer," *Computer Architecture News* **10**, 39–47.

[16] Stallings, W. (1990), *Reduced Instruction Set Computers (RISC),* Second Edition. IEEE Computer Society Press.

[17] Tolimieri, R., An, M., and Lu, C. (1989), *Algorithms for Discrete Fourier Transform and Convolutions*, Springer-Verlag, New York.

[18] *IBM Journal of Research and Development: Special Issue on IBM RISC System/6000 Processor*, June, 1990.

[19] Intel, *iPSC/860 Supercomputer Advanced Information Fact Sheet.* Intel 1990.

[20] *AT&T DSP Parallel Processor BT-100 User Manual*, AT&T, 1988.

11

Implementation on Parallel Architectures

11.1 Introduction

In this chapter, we will consider some issues surrounding parallel implementation of several MDFT algorithms on a broadcast mode multiprocessor machine. Such machines typically feature a collection of homogeneous processing elements (nodes) together with an interconnection network of a regular topology for interprocessor communication. The node processors are externally connected by a single I/O channel to a host through which all data loads and unloads are carried out (see Figure 11.1).

The machine must support at least two communication functions between host and node processors:

- *Broadcast*: The function downloads data from the I/O channel to all node processors.
- *Report*: This function allows a distinguished node to upload data to the I/O channel.

Multinode processors are often classified by reference to the following machine parameters.

- The *granularity* of the node processors including their computational complexity and cache size.
- The *degree of parallelism* as measured by the number of independent processing elements.

R. Tolimieri et al., *Mathematics of Multidimensional Fourier Transform Algorithms*

- The *network topology*, which includes the pattern and density of interaction between processors (*coupling*) as well as the distribution of control across the processing elements (*task allocation and synchronization*).

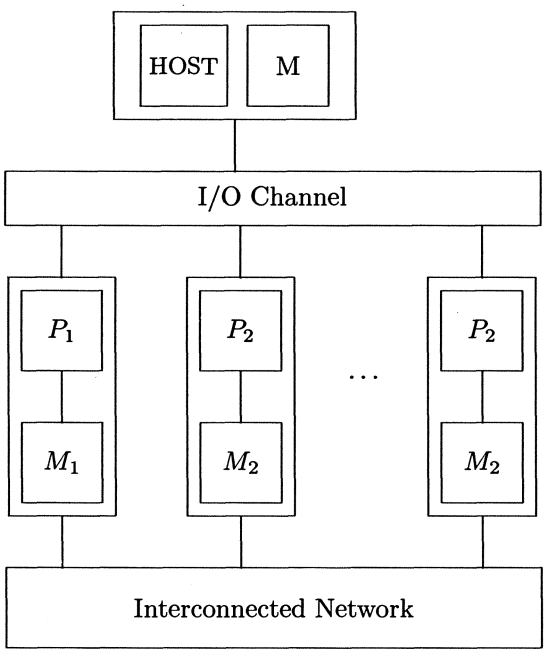


Figure 11.1 A distributed computing model

We will usually assume that the machine model contains powerful node processors (coarse-grained) and has limited interaction between node processors (loosely coupled). Examples of computers within this framework include the AT&T BT-100 [35], Intel Hypercube iPSC/2 [38], iPSC/860 [38], NCUBE 6400 [36], CM-2 of Thinking Machines Co. [11]. Such machines have large computational power but limited *communication bandwidth.* Communication cost, rather than arithmetic operations, can dominate execution time of a parallel algorithm. Two commercially available machines can illustrate this argument. The iPSC/860, composed of up to 128 i860 microprocessors (thus far), can achieve 7.5 GFlops, while its aggregate point-to-point communication bandwidth is only 2.8 Mbytes per channel. In the CM-2, the average time needed to transfer a 32-bit message between two processors is about 250 microseconds, while a floating-point multiplication takes about 24 microseconds.

Therefore, reducing and hiding communication cost for parallel algorithm design becomes absolutely essential to obtain optimum performance of a parallel computation, i.e., algorithms that minimize interprocessor communication.

In the next section, we will describe the implementation of the row–column FFT on a broadcast mode multiprocessor, which will serve as a reference model for the remaining sections. In these later sections we will discuss implementation strategies for the RTA, the vector-radix FFT, and a hybrid algorithm based on the following two-stage decomposition of the DFT computation.

- A *global dimension reduction* stage that downloads the data to the node processors and simultaneously implements node dependent periodizations (RTA) or generalized periodizations (FFT).
- A Local DFT computational stage implemented on the node level.

In general, this strategy computes an R-dimensional DFT by a set of independent S-dimensional DFTs with preassigned $S < R$. The global stage has no data interdependencies and can be implemented in parallel across the node processors. We assume that the arithmetic of the global stage can be computed simultaneously with the data download.

The S-dimensional DFT computations represent the DFT of the initial R-dimensional data restricted to S-dimensional hyperplanes of the indexing set. The advantage of this approach is that the local DFT computations are taken on lower-dimensional data and can be implemented in parallel.

11.2 Parallel Implementation of FFT

Multidimensional FFT algorithms have been extensively studied. Both row–column FFT and vector-radix FFT alternate stages of computation with stages of global data exchange. Parallel implementation of the row–column FFT requires a global transpose operation between the row FFT stages and the column FFT stages. On our machine model, this stage is implemented by having every processing element exchange data with every other processing element. Although the vector-radix algorithm introduces structures that can be more finely tuned to specific architecture, extensive interprocessor communication is still required. Implementation of the row–column FFT on a broadcast mode multiprocessor can be described by the following sequence of steps.

Multiprocessor implementation of the row–column FFT

- Broadcast rows of the input multidimensional data uniformly among the processing elements. If there are more rows than processing ele-

ments, then several rows may be assigned to a particular processing element.

- In parallel, compute the DFT of the rows of the data in each of the processing elements.
- Globally transpose the intermediate results using the interconnection network of the machine.
- In parallel, compute the DFT of the data in each of the processing elements.
- Upload results. The transformed data is stored columnwise in the machine. If data is required in its natural order, then an additional transpose must be carried out at some point of the uploading process.

For 2-dimensional data arrays, two global transposes are required to implement the row–column FFT on a broadcast mode multiprocessor. An additional global transpose is required for each increase in dimension. Implementation of global transpose on a parallel machine depends strongly on the interconnection network and topology. A great deal of research has been carried out for optimal implementation of global transpose specific to particular topologies. We will not consider this problem further at this time.

11.3 Parallel Implementation of the RTA

In this section we will discuss parallel implementation of the RTA on a broadcast mode multiprocessor. The RTA computes the R-dimensional DFT by a set of independent S-dimensional DFTs ($S < R$) in the node processors. The input to the S-dimensional DFTs is formed by a periodizing operation involving only additions on the R-dimensional input data. This periodization operation varies over the nodes. We assume that the periodizations (additions) in this stage can be computed simultaneously with data download. We shall see that the RTA scales to some extent with the degree of parallelism and granularity of the target architecture. The optimal case occurs when the number of S-dimensional hyperplanes is equal to the degree of parallelism of the machine. In section 11.5 we will provide additional tools for scaling by introducing a hybrid algorithm that nests the RTA within the multidimensional FFT.

Under the above assumption, we can assign to each of the node processors an S-dimensional hyperplane in such a way that the collection of assigned S-dimensional hyperplanes covers the R-dimensional indexing set. We will assume that such an assignment has been made and is fixed throughout this section. We can distinguish three stages in the RTA.

- A global dimension reduction stage that downloads and simultaneously periodizes the data to the node processors. Since there is no data interdependency, this stage can proceed in parallel across the node processors. The assignment of a hyperplane to a node uniquely determines the periodization of initial data to the node. In all cases, S-dimensional data is placed in the node for local computation.
- An S-dimensional DFT computation stage on the node level that computes the R-dimensional DFT of the initial data on the S-dimensional hyperplane assigned to the node.
- An uploading stage that places the computed data in its appropriate positions. Redundant data elimination must also be carried out.

Consider the S-dimensional hyperplane RTA computation of the R-dimensional DFT. Optimal parallel implementation on a broadcast mode multiprocessor machine occurs when the number of S-dimensional hyperplanes covering the R-dimensional indexing set equals the degree of machine parallelism. This degree of parallelism varies with S. For example, if each dimension has size p^N, then the degree of parallelism is

$$\left(\frac{p^N}{p}\right)^{R-S}\left(\frac{p^{R-S+1}}{p-1}\right),$$

and the granularity of the DFT computation at the nodes is

$$(p^N)^S.$$

As S increases, the degree of parallelism decreases and the granularity increases. The trade-off is limited to powers of p^N.

In the following subsections we will give some examples of this procedure.

11.3.1 2-dimensional prime case

The 2-dimensional $p \times p$ DFT can be written as

$$\mathbf{y} = (F(p) \otimes F(p))\mathbf{x}. \tag{11.1}$$

Recall the RTA computation of (11.1) for a prime p.

1. Reduction stage. Calculate the $(p+1)$ summations:

$$a_d^{(m,1)} = \sum_{i=0}^{p-1} x(i, d-mi),$$

$$a_d^{(1,0)} = \sum_{i=0}^{p-1} x(d,i),$$

where $d = 0, 1, 2, \ldots, p-1$ and $m = 0, 1, 2, \ldots, p-1$.

2. FFT stage. Compute the $(p+1)$ 1-dimensional DFTs:

$$\begin{aligned} \mathbf{v}_0 &= F(p)\mathbf{a}^{(0,1)}, \\ \mathbf{v}_1 &= F(p)\mathbf{a}^{(1,1)}, \\ &\vdots \\ \mathbf{v}_{p-1} &= F(p)\mathbf{a}^{(p-1,1)}, \\ \mathbf{v}_p &= F(p)\mathbf{a}^{(1,0)}. \end{aligned}$$

3. Unload and remove redundant data from $\{\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_p\}$ to the output data array $\mathbf{y}$.

The implementation of the 2-dimensional $p \times p$ RTA can be given as follows: Each processor requires only $O(p)$ storage for each $\mathbf{a}^{H(m,l)}$ it is assigned. Redundant data is removed in the uploading stage. The redundancy in this case is trivial, since $y(0,0)$ is the only output common to every processor.

Processor m : $(m = 0, 1, 2, \ldots, p-1)$
  for $j = 0$ to $p-1$
    $a^{H(m,1)}(j) = 0.0$
    for $i = 0$ to $p-1$
      $a^{H(m,1)}(j) = a^{H(m,1)}(j) + x(i, j - mi)$
    end
  end$p$-point FT $F(p)\mathbf{a}^{H(m,1)}$
  upload to output data array $y(mk, k)$.
Processor p
  for $j = 0$ to $p-1$
    $a^{H(1,0)}(j) = 0.0$
    for $i = 0$ to $p-1$
      $a^{H(1,0)}(j) = a^{H(1,0)}(j) + x(j, i)$
    end
  end
  p-point FT $F(p)\mathbf{a}^{H(1,0)}$
  upload to output data array $y(k, 0)$.

11.4 Parallel Implementation of FFT

Assume that $N = SM$, where S is much smaller than M. Direct implementation of the

Cooley–Tukey Algorithm

$$F(N) = P(N, M)(I_S \otimes F(M))T_M(N)(F(S) \otimes I_M) \qquad (11.2)$$

is a two-stage calculation. Since we assume that S is relatively small and M large, the second stage M-point DFTs carry much heavier weight than the first stage S-point DFTs. The first stage can be viewed as preparation for the M-point DFT computations. For simplicity, assume that the machine has two processors (degree of parallelism is two), $S = 2$. The implementation of (11.2) can be given as follows:

Processor 1
for $i = 0$ to $M - 1$
$x_1(i) = x(i) + x(i + M)$
end for
M-point Fourier transform $F(M)\mathbf{x}_1$
stride unload (report) to $y(2 * k)$.
Processor 2
for $i = 0$ to $M - 1$
$x_2(i) = (x(i) - x(i + M)) * w_N^i$
end for
M-point Fourier transform $F(M)\mathbf{x}_2$
stride unload (report) to $y(2 * k + 1)$.

The main idea is to perform additions or other operations while loading the data to each node processor. The same strategy can be applied to the

Good–Thomas Algorithm

$$F(N) = P_2(I_S \otimes F(M))(F(S) \otimes I_M)P_1. \tag{11.3}$$

For $S = 2$ and an odd integer M, S and M are relatively prime.

Processor 1
for $i = 0$ to $(M - 1)/2$
$x_1(i) = x(2 * i) + x(2 * i + M)$
end for
for $i = 1$ to $(M - 1)/2$
$x_1(i + (M - 1)/2) = x(2 * i - 1 + M) + x(2 * i - 1)$
end for
M-point FT $F(M)\mathbf{x}_1$
P_2 permutation unload (report) to $y(k)$.
Processor 2
for $i = 0$ to $(M - 1)/2$
$x_2(i) = x(2 * i) - x(2 * i + M)$
end for
for $i = 1$ to $(M - 1)/2$
$x_2(i + (M - 1)/2) = x(2 * i - 1 + M) - x(2 * i - 1)$
end for
M-point FT $F(M)\mathbf{x}_2$
P_2 permutation unload (report) to $y(k)$.

Notice that in the above procedure, we achieve addressing without using modulo operations. Only fixed-point stride addressing is involved.

As in the Cooley–Tukey algorithm, the data array $\mathbf{x}$ is partitioned into two subsets $\mathbf{x}_1$ and $\mathbf{x}_2$ by the first stage S-point FTs. Independent M-point FTs act on each of the subsets.

The key issue in the above implementation strategy is reducing interprocessor communications and taking advantage of parallelism of data loading and node processor operation. The same idea can be applied easily to MDFTs. Hybrid algorithms designed in this way offer substantial flexibility in terms of degree of parallelism and granularity of a target machine. We will use the 2-dimensional FT for the purpose of illustration.

2-Dimensional Vector Radix Cooley–Tukey Algorithm

$$F(\mathbf{N}) = P(\mathbf{N},\mathbf{M})(I_{\mathbf{S}} \otimes F(\mathbf{M}))T_{\mathbf{M}}(\mathbf{N})(F(\mathbf{S}) \otimes I_{\mathbf{M}}, \tag{11.4}$$

where

$$\mathbf{N} = (N_1, N_2), \quad \mathbf{M} = (M_1, M_2), \quad \mathbf{S} = (S_1, S_2),$$
$$N_1 = S_1 M_1, \quad N_2 = S_2 M_2,$$

$$\begin{aligned}
F(\mathbf{N}) &= F(N_1, N_2) = F(N_1) \otimes F(N_2), \\
F(\mathbf{M}) &= F(M_1, M_2) = F(M_1) \otimes F(M_2), \\
F(\mathbf{S}) &= F(S_1, S_2) = F(S_1) \otimes F(S_2), \\
P(\mathbf{N},\mathbf{M}) &= P(N_1, M_1) \otimes P(N_2, M_2), \\
T_{\mathbf{M}}(\mathbf{N}) &= T_{M_1}(N_1) \otimes T_{M_2}(N_2), \\
I(\mathbf{S}) &= I(S_1) \otimes I(S_2).
\end{aligned}$$

Many implementations are possible.

Case I Assume that the machine has two processors and that each processor can operate on half of the data size of input array $\mathbf{x}$. Set $N_2 = 2M_2$.

Processor 1
for $i = 0$ to $M_2 - 1$
for $j = 0$ to $N_1 - 1$
$x_1(j,i) = x(j,i) + x(j,i+M_2)$
end for
end for
2-dimensional $N_1 \times M_2$-point FT $F(N_1, M_2)\mathbf{x}_1$
stride unload (report) to $y(k_1, 2 * k_2)$

Processor 2
for $i = 0$ to $M_2 - 1$
for $j = 0$ to $N_1 - 1$
$x_2(j,i) = (x(j,i) - x(j,i+M_2)) * w_{N_2}^i$
end for

end for
2-dimensional $N_1 \times M_2$-point FT $F(N_1, M_2)\mathbf{x}_2$
stride unload (report) to $y(k_1, 2 * k_2 + 1)$

Case II Assume that the machine has four processors and that each processor can operate on a quarter of the data size of input array $\mathbf{x}$. Set $N_1 = 2M_1$ and $N_2 = 2M_2$.

Processor 1
for $i = 0$ to $M_1 - 1$
for $j = 0$ to $M_2 - 1$

$$x_1(j,i) = x(j,i) + x(j,i+M_2) + x(j+M_1,i) + x(j+M_1,i+M_2)$$

end for
end for
2-dimensional $M_1 \times M_2$-point FT $F(M_1, M_2)\mathbf{x}_1$
stride unload (report) to $y(2 * k_1, 2 * k_2)$

Processor 2
for $i = 0$ to $M_1 - 1$
for $j = 0$ to $M_2 - 1$

$$x_2(j,i) = (x(j,i) - x(j,i+M_2) + x(j+M_1,i) - x(j+M_1,i+M_2)) * w_{N_2}^{j}$$

end for
end for
2-dimensional $M_1 \times M_2$-point FT $F(M_1, M_2)\mathbf{x}_2$
stride unload (report) to $y(2 * k_1, 2 * k_2 + 1)$.

Processor 3
for $i = 0$ to $M_1 - 1$
for $j = 0$ to $M_2 - 1$

$$x_3(j,i) = (x(j,i) + x(j,i+M_2) - x(j+M_1,i) - x(j+M_1,i+M_2)) * w_{N_1}^{i}$$

end for
end for
2-dimensional $M_1 \times M_2$-point FT $F(M_1, M_2)\mathbf{x}_3$
stride unload (report) to $y(2 * k_1 + 1, 2 * k_2)$

Processor 4
for $i = 0$ to $M_1 - 1$
for $j = 0$ to $M_2 - 1$

$$x_4(j,i) = (x(j,i) - x(j,i+M_2) - x(j+M_1,i) + x(j+M_1,i+M_2)) * w_{N_2}^{i} * w_{N_1}^{j}$$

end for
end for
2-dimensional $M_1 \times M_2$ -point FT $F(M_1, M_2)\mathbf{x}_4$
stride unload (report) to $y(2 * k_1 + 1, 2 * k_2 + 1)$

Case III Assume that the machine has only two processors and that each processor can operate on a quarter of the data size of input array $\mathbf{x}$. The case II algorithm can still be used, but each processor will be assigned two sets of operations.

A similar implementation strategy can be applied to the 2-dimensional Good–Thomas vector radix prime factor algorithm.

2-Dimensional Vector-Radix Good–Thomas Algorithm

$$F(\mathbf{N}) = P_2(\mathbf{N})(I_{\mathbf{S}} \otimes F(\mathbf{M}))(F(\mathbf{S}) \otimes I_{\mathbf{M}} P_1(\mathbf{N}), \tag{11.5}$$

where

$$P_1(\mathbf{N}) = P_1(N_1) \otimes P_1(N_2),$$
$$P_2(\mathbf{N}) = P_2(N_1) \otimes P_2(N_2).$$

Implementation of (11.5) is the same as the 2-dimensional Cooley–Tukey algorithm, except that input and output data loading and unloading require different permutations.

The above implementation techniques can be extended easily to the multidimensional vector-radix Cooley–Tukey or Good–Thomas FFT.

11.5 Hybrid Algorithm

The RTA offers a trade-off between degree of parallelism and granularity depending on the dimension of output hyperplanes. The trade-off is limited by transform size, which reduces its potential for scaling with machine parameters. A hybrid algorithm that nests the RTA within the multidimensional FFT will be discussed. The advantage of this hybrid algorithm is that both the dimension of the output hyperplanes and the size of the transform can be manipulated to affect a finer trade-off between degree of parallelism and granularity as compared with the stand-alone RTA.

The multidimensional FFT operates at the global level to decompose the initial DFT computation into smaller-size DFT computations without change in dimension. The RTA can then be applied to each of these smaller size DFTs.

One way to increase control over the trade-off between degree of parallelism and granularity is to modify the size of the computation in each dimension. This can be done by the multidimensional Cooley–Tukey FFT, the vector-radix FFT, or the Good–Thomas prime factor algorithm described in chapters 2 and 5.

The tensor product formulation of the multidimensional Cooley–Tukey FFT has the form

$$F(\mathbf{N}) = Q(F(\mathbf{M}) \otimes I_L)T(I_M \otimes F(\mathbf{L}))P,$$

where P and Q are permutation matrices, T is a diagonal matrix, and

$$\begin{array}{ll} \mathbf{N} = (N_1, \ldots, N_R), & N = N1 \cdots N_R, \\ \mathbf{N} = (N_1, \ldots, N_R), & N = N1 \cdots N_R, \quad N_r = M_r L_r, \\ \mathbf{M} = (M_1, \ldots, M_R), & M = M1 \cdots M_R, \\ \mathbf{L} = (L_1, \ldots, L_R), & L = L1 \cdots L_R. \end{array}$$

Consider the computation of $I_M \otimes F(\mathbf{L})$ not as a parallel operation, but rather as a sequence of M computations of the action of $F(\mathbf{L})$ on vectors determined by the permutation P. We will apply the RTA to each of these computations. Observe that although the dimension of the indexing set for $F(\mathbf{L})$ is the same as the dimension of the indexing for $F(\mathbf{N})$, the size of the transform has been reduced. Choose a dimension S such that the number of S-dimensional hyperplanes covering the R-dimensional indexing set for $F(\mathbf{L})$ equals the degree of parallelism of the target machine. Assign one hyperplane to each node. Apply the RTA to a single $F(\mathbf{L})$ computation. The result is that each node contains an S-dimensional hyperplane of the output of $F(\mathbf{L})$. Continue in this way to implement the remaining computations of $F(\mathbf{L})$. Each node contains the outputs of all the $F(\mathbf{L})$ computations on the fixed preassigned S-dimensional hyperplane. The contents of a single node provide (after twiddle factor multiplications) the input into the second stage of the computation, which is completed by a single R-dimensional $F(\mathbf{M})$ computation.

The following example uses the Good–Thomas prime factor algorithm with RTA.

Example 11.1 Assume that the transform size is $\mathbf{N} = 14 \times 14$, $14 = 2 \times 7$, and the Good–Thomas algorithm is used at the global level to break up the data into two or four subsets corresponding to whether one or two dimensions are being processed. The algorithm is given as

$$F(14,14) = P_2(14,14)(I_4 \otimes F(7,7))(F(2,2) \otimes I_{49})P_1(14,14), \qquad (11.6)$$

where $F(N,N) = F(N) \otimes F(N)$ and $P(N,N) = P(N) \otimes P(N)$.

The 14×14 data array is broken up into 4 data arrays of size 7×7 by the operation $F(2,2) \otimes I_{49}$ as follows:

array 1
  for $i = 0$ to 3
    for $j = 0$ to 3
      $x_1(j,i) = x(2*j, 2*i) + x(2*j, 2*i+7)$
        $+x(2*j+7, 2*i) + x(2*j+7, 2*i+7)$
    end for
    for $j = 1$ to 3
      $x_1(j+3,i) = x(2*j-1, 2*i) + x(2*j-1, 2*i+7)$
        $+x(2*j+6, 2*i) + x(2*j+6, 2*i+7)$
    end for
  end for
  for $i = 1$ to 3

for $j = 0$ to 3
$x_1(j, i+3) = x(2*j, 2*i-1) + x(2*j, 2*i+6)$
$+x(2*j+7, 2*i-1) + x(2*j+7, 2*i+6)$
end for
for $j = 1$ to 3
$x_1(j+3, i+3) = x(2*j-1, 2*i-1) + x(2*j-1, 2*i+6)$
$+x(2*j+6, 2*i-1) + x(2*j+6, 2*i+6)$
end for
end for

array 2
for $i = 0$ to 3
for $j = 0$ to 3
$x_2(j, i) = x(2*j, 2*i) - x(2*j, 2*i+7)$
$+x(2*j+7, 2*i) - x(2*j+7, 2*i+7)$
end for
for j = 1 to 3
$x_2(j+3, i) = x(2*j-1, 2*i) - x(2*j-1, 2*i+7)$
$+x(2*j+6, 2*i) - x(2*j+6, 2*i+7)$
end for
end for
for $i = 1$ to 3
for $j = 0$ to 3
$x_2(j, i+3) = x(2*j, 2*i-1) - x(2*j, 2*i+6)$
$+x(2*j+7, 2*i-1) - x(2*j+7, 2*i+6)$
end for
for $j = 1$ to 3
$x_2(j+3, i+3) = x(2*j-1, 2*i-1) - x(2*j-1, 2*i+6)$
$+x(2*j+6, 2*i-1) - x(2*j+6, 2*i+6)$
end for
end for

array 3
for $i = 0$ to 3
for $j = 0$ to 3
$x_3(j, i) = x(2*j, 2*i) + x(2*j, 2*i+7)$
$-x(2*j+7, 2*i) - x(2*j+7, 2*i+7)$
end for
for $j = 1$ to 3
$x_3(j+3, i) = x(2*j-1, 2*i) + x(2*j-1, 2*i+7)$
$-x(2*j+6, 2*i) - x(2*j+6, 2*i+7)$
end for
end for
for $i = 1$ to 3
for $j = 0$ to 3
$x_3(j, i+3) = x(2*j, 2*i-1) + x(2*j, 2*i+6)$
$-x(2*j+7, 2*i-1) - x(2*j+7, 2*i+6)$

```
        end for
        for j = 1 to 3
            x3(j + 3, i + 3) = x(2 * j − 1, 2 * i − 1) + x(2 * j − 1, 2 * i + 6)
                               −x(2 * j + 6, 2 * i − 1) − x(2 * j + 6, 2 * i + 6)
        end for
    end for
array 4
    for i = 0 to 3
        for j = 0 to 3
            x4(j, i) = x(2 * j, 2 * i) − x(2 * j, 2 * i + 7)
                       −x(2 * j + 7, 2 * i) + x(2 * j + 7, 2 * i + 7)
        end for
        for j = 1 to 3
            x4(j + 3, i) = x(2 * j − 1, 2 * i) − x(2 * j − 1, 2 * i + 7)
                           −x(2 * j + 6, 2 * i) + x(2 * j + 6, 2 * i + 7)
        end for
    end for
    for i = 1 to 3
        for j = 0 to 3
            x4(j, i + 3) = x(2 * j, 2 * i − 1) − x(2 * j, 2 * i + 6)
                           −x(2 * j + 7, 2 * i − 1) + x(2 * j + 7, 2 * i + 6)
        end for
        for j = 1 to 3
            x4(j + 3, i + 3) = x(2 * j − 1, 2 * i − 1) − x(2 * j − 1, 2 * i + 6)
                               −x(2 * j + 6, 2 * i − 1) + x(2 * j + 6, 2 * i + 6)
        end for
    end for
```

Notice that the addressing of the load and addition stage of the above loops is relatively simple, involving only stride loading. No modulo operations are required, as is typical with the Good–Thomas algorithm, since only half the looping size is required.

The above preparation stage can be done at various levels, depending on the target parallel machine. They can be carried out by a host processor at different time periods in parallel with node processors while the node processors carry out the DFT stage on the previous data set.

11.6 An Example Program on iPSC/860

A program example of the RTA algorithm on the Intel iPSC/860 is given in this section. The transform size is a 2-dimensional 257×257.

```
c c This program computes 2-D p by p Fourier transform
```

```
c by the Reduced Transform Algorithm (RTA)
c
c Program Constants:
c
c ALLNODES -1 all active nodes in the cube
c
c DATASIZE 4 * n * n size of data in bytes
c
c       Program rtapxp
        include 'fcube.h'
c ************************************************
c p by p 2-D Fourier transform
c p - prime, p=257
c (p+1) 1-D p-point FFTs are called
c ************************************************
        complex x(0:256,0:256), y(0:256,0:256)
        integer * 4 ALLNODES, DATATYPE, DATASIZE,
     > OUTDSIZE1, OUTTYPE1, OUTDSIZE,
     > OUTTYPE, numnod, mynod
        data ALLNODES /-1/,DATATYPE/20/,OUTDTYPE/10/,
     > DATASIZE /264196/, OUTDSIZE /32896/,
     > OUTTYPE1 /30/, OUTSIZE1 /33924/
c use 3-dimensional cube - 8 node processor
c initialize the data
c these data can be loaded or prestored in memory
        n = 257
        numnod = numnodes()
        mynod = mynode ( )
        write ( * , * ) ' number of nodes', numnod
        if (mynod.eq.0) then
           do i = 0, n-1
              do j = 0, n-1
                 x(j,i)=cmplx(cos(0.3 * i),(0.4 * i * j))
              end do
           end do
c send data to other nodes from node 0
           call csend(DATATYPE,x,DATA5IZE,ALLNCDE5,0)
           s0 = dclock()
c other nodes (1-7) receive data
        else
           call crecv(DATATYPE, x, DATASIZE)
        endif
        goto (1, 2, 3, 4, 5, 6, 7, g) , mynod+1
c Processor # 0
     1  continue
```

```
      r00 = dclock () - s0
      s0 = dclock()
      do i = 0, n-1
         y (i, 0) = x (i, 0)
         do j = 1, n-1
            y (i, 0) = y (i, 0) + x (i, j)
         end do
      end do
      call ftc257 ( y (0, 0) , 1 )
      do k = l, 31
         do i = 0, n-1
            y (i, k) = x (i, 0)
            do j = l, n-1
               jj = i - j * k
               jj = mod (jj, n)
               y(i,k) = y(i,k) + x(jj,j)
            end do
         end do
         call ftc257 ( y(0,k), 1)
      end do
      r01 = dclock () - s0
      write ( * ,10) r00, r01
   10 format(' n0 send = ', f10.6, 'run= ', f10.6)
      goto 999
c Processor # 1
    2 continue
      s1 = dclock()
      do k = 0, 31
         kk = k+32
         do i = 0, n-1
            y (i, k) = x (i, 0)
            do j = l, n-1
               jj = i - j * kk
               jj = mod (jj, n)
               y(i,k) = y(i,k) + x(jj,j)
            end do
         end do
         call ftc257 ( y(0,k), 1)
      end do
      r1 = dclock () - s1
      write ( * ,11) r1
   11 format(' n1 runtime = ', f10.6)
      call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
      goto 999
c Processor # 2
```

```
      3  continue
         s2 = dclock()
         do k = 0, 31
            kk = k+64
            do i = 0, n-1
               y (i, k) = x (i, 0)
               do j = l, n-1
                  jj = i - j * kk
                  jj = mod (jj, n)
                  y(i,k) = y(i,k) + x(jj,j)
               end do
            end do
            call ftc257 ( y(0,k), 1)
         end do
         r2 = dclock () - s2
         write ( * ,12) r2
     12  format(' n2 runtime = ', f10.6)
         call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
         goto 999
c Processor # 3
      4  continue
         s3 = dclock()
         do k = 0, 31
            kk = k+97
            do i = 0, n-1
               y (i, k) = x (i, 0)
               do j = l, n-1
                  jj = i - j * kk
                  jj = mod (jj, n)
                  y(i,k) = y(i,k) + x(jj,j)
               end do
            end do
            call ftc257 ( y(0,k), 1)
         end do
         r3 = dclock () - s3
         write ( * ,13) r3
     13  format(' n3 runtime = ', f10.6)
         call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
         goto 999
c Processor # 4
      5  continue
         s4 = dclock()
         do k = 0, 31
            kk = k+129
            do i = 0, n-1
```

```
                y (i, k) = x (i, 0)
                do j = l, n-1
                   jj = i - j * kk
                   jj = mod (jj, n)
                   y(i,k) = y(i,k) + x(jj,j)
                end do
             end do
             call ftc257 ( y(0,k), 1)
          end do
          r4 = dclock () - s4
          write ( * ,14) r4
       14 format(' n4 runtime = ', f10.6)
          call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
          goto 999
c Processor # 5
        6 continue
          s5 = dclock()
          do k = 0, 31
             kk = k+162
             do i = 0, n-1
                y (i, k) = x (i, 0)
                do j = l, n-1
                   jj = i - j * kk
                   jj = mod (jj, n)
                   y(i,k) = y(i,k) + x(jj,j)
                end do
             end do
             call ftc257 ( y(0,k), 1)
          end do
          r5 = dclock () - s5
          write ( * ,15) r5
       15 format(' n5 runtime = ', f10.6)
          call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
          goto 999
c Processor # 6
        7 continue
          s6 = dclock()
          do k = 0, 31
             kk = k+194
             do i = 0, n-1
                y (i, k) = x (i, 0)
                do j = l, n-1
                   jj = i - j * kk
                   jj = mod (jj, n)
                   y(i,k) = y(i,k) + x(jj,j)
```

```
            end do
         end do
         call ftc257 ( y(0,k), 1)
      end do
      r6 = dclock () - s6
      write ( * ,16) r6
   16 format(' n6 runtime = ', f10.6)
      call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
      goto 999
c Processor # 7
    8 continue
      s7 = dclock()
      do k = 0, 30
         kk = k+226
         do i = 0, n-1
            y (i, k) = x (i, 0)
            do j = l, n-1
               jj = i - j * kk
               jj = mod (jj, n)
               y(i,k) = y(i,k) + x(jj,j)
            end do
         end do
         call ftc257 ( y(0,k), 1)
      end do
      do i = 0, n-1
         y(i,31) = x(0,i)
         do j = 1, n-1
            y(i,31) = y(i,31) + x(j,i)
         end do
      end do
      call ftc257 (y(0,31), 1)
      r7 = dclock () - s7
      write ( * ,17) r7
   17 format(' n7 runtime = ', f10.6)
      call csend(OUTDTYPE, y, OUTDSIZE, 0, mypid())
  999 continue
      if (mynode().eq.0) then
c from node #0
      s0 = dclock()
      do i = 0, n-1
         x(i,0) = y(i,0)
      end do
      do i = 1, n-1
         x(i,i) = y(i,1)
      end do
```

```
      do j = 2, 31
         do i = 0, n-1
            jt = mod(i * j,n)
            x(i,jt) = y(i,j)
         end do
      end do
c output data from node #1
      call crecv(OUTTYPE, y, OUTSIZE)
      do j = 32, 63
         jj= j - 32
         do i = 0, n-1
            jt = mod(i * j,n)
            x(i,jt) = y(i,jj)
         end do
      end do
c output data from node #2
      call crecv(OUTTYPE, y, OUTSIZE)
      do j = 64, 96
         jj= j - 64
         do i = 0, n-1
            jt = mod(i * j,n)
            x(i,jt) = y(i,jj)
         end do
      end do
c output data from node #3
      call crecv(OUTTYPE, y, OUTSIZE)
      do j = 97, 128
         jj= j - 97
         do i = 0, n-1
            jt = mod(i * j,n)
            x(i,jt) = y(i,jj)
         end do
      end do
c output data from node #4
      call crecv(OUTTYPE, y, OUTSIZE)
      do j = 129, 161
         jj= j - 129
         do i = 0, n-1
            jt = mod(i * j,n)
            x(i,jt) = y(i,jj)
         end do
      end do
c output data from node #5
      call crecv(OUTTYPE, y, OUTSIZE)
      do j = 162, 193
```

```
            jj= j - 162
            do i = 0, n-1
               jt = mod(i * j,n)
               x(i,jt) = y(i,jj)
            end do
         end do
c output data from node #6
         call crecv(OUTTYPE, y, OUTSIZE)
         do j = 194, 256
            jj= j - 194
            do i = 0, n-1
               jt = mod(i * j,n)
               x(i,jt) = y(i,jj)
            end do
         end do
c output data from node #7
         call crecv(OUTTYPE, y, OUTSIZE)
         do j = 226, 256
            jj= j - 226
            do i = 0, n-1
               jt = mod(i * j,n)
               x(i,jt) = y(i,jj)
            end do
         end do
c
         do i =1, n-1
            x(0,i) = y(i,31)
         end do
         r00 = dclock() - s0
         write ( * ,18) r00
      18 format('Last stage receive t = ', f10.6)
         endif
         end
```

References

[1] Agarwal, R.C. and Cooley, J.W. (1986), "Fourier Transform and Convolution Subroutines for the IBM 3090 Vector Facility," *IBM J. Res. Devel.* **30**, 145–162.

[2] — (1986), "An Efficient Vector Implementation of the FFT Algorithm on IBM 3090VF," *Proc. ICASSP-86*, 249–252.

[3] — (1987), "Vectorized Mixed Radix Discrete Fourier Transform Algorithms," *IEEE Proc.* **75**(9).

[4] Auslander, L., Feig, E., and Winograd, S. (1983), "New Algorithms for the Multi-Dimensional Discrete Fourier Transform," *IEEE Trans. Acoust., Speech, Signal Processing* **ASSP-31** (2), 388–403.

[5] Berglsnd, G.D. (1972), "A Parallel Implementation of the Fast Fourier Transform Algorithm," *IEEE Trans. Computers* **C-21**(4), 366–370.

[6] Blahut, R.E. (1985), *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading, MA.

[7] Browning, S.A. (1980), *The Tree Machine: A Highly Concurrent Computing Environment*, Ph.D. thesis, CIT, CA.

[8] Burrus, C.S. and Eschenbacher, P.W. (1979), "An In-place, In-order Prime Factor FFT Algorithm," *IEEE Trans. Acoust., Speech, Signal Proc.* **ASSP-29**, 806–817.

[9] Chamberlain, R.M. (1988), "Gray Codes, Fast Fourier Transforms and Hypercubes," *Parallel Computing* **6**, 225–233.

[10] Chu, C.Y. (1988), *The Fast Fourier Transform on Hypercube Parallel Computers*, Ph.D. thesis, Cornell Univ.

[11] "Connection Machine CM-2, Technical Summary," Thinking Machines Co. Technical Report HA87-4, April, 1987.

[12] Fox, G.C. and Otto, S.W. (1984), "Algorithms for Concurrent Processors," *Phys. Today* **37**, 50–59.

[13] Gertner, I. (1988), "A New Efficient Algorithm to Compute the Two-dimensional Discrete Fourier Transform," *IEEE Trans. ASSP* **ASSP-36**(7), 1036–1050.

[14] Gertner, I. and Shamash, M. (1987), "VLSI Architectures for Multidimensional Fourier Transform Processing," *IEEE Trans. Comp.* **C-36**(11), 1265–1274.

[15] Gertner, I. and Rofheart, M. (1990), "A Parallel Algorithm for 2-D DFT Computation with No Interprocessor Communication," *IEEE Trans. Parallel and Dist. Syst.* **1**(3).

[16] Gorin, A.L., Auslander, L., and Silberger, A. (1987), "Balanced Computation of 2-D Transforms on a Tree Machine," *Appl. Math. Letters.*

[17] Hwang, K. and Briggs, F.A. (1984), *Computer Architecture and Parallel Processing*, McGraw-Hill, New York.

[18] Jackson, E., She, Z., and Orszag, S. (1991), "A case Study in Parallel Computing: I. Homogeneous Turbulence on a Hypercube," *J. Scientific Comp.* **6** (1).

[19] Jamieson, L.H., Mueller, P.T., and Siegel, H.J. (1986), "FFT Algorithms for SIMD Processing," *J. Paral. Dist. Comp.*

[20] Jesshope, C.R. (1980), "The Implementation of Fast Radix-2 Transforms on Array Processors," *IEEE Trans. Comp.* **C-29** (1), 20–27.

[21] Johnson, S.L., Krawitz, R.L., Frye, R., and Macdonald, D. (1989), "A Radix-2 FFT on the Connection Machine," *Supercomputing'89.*

[22] Korn, D.G. and Lambiotte, J. Jr. (1979), "Computing the Fast Fourier Transform on a Vector Computer," *Math. Comput.* **33**, 977–992.

[23] Lu, C., An, M., Qian, S., and Tolimieri, R. (1992), "Parallel M-D FFT Algorithms and Their Implementation on Distributed Computing Systems," submitted for publication.

[24] Matsuura, T., Miura, K., and Makino, M. (1985), "Supervector Performance without Toil," *Comput. Phys. Comm.* **37**, 101–107.

[25] Norton, V.A. and Silberger, A.J. (1987), "Parallelization and Performance Analysis of the Cooley–Tukey FFT Algorithm for Shared-memory Architectures," *IEEE Trans. Comp.* **C-36**(5), 581–591.

[26] Pease, M.C. (1968), "An Adaptation of the Fast Fourier Transform for Parallel Processing," *J. Assoc. Comp. Mach.* **15**, 253–264.

[27] Peterson, W.P. (1983), "Vector Fortran for Numerical Problems on Cray-1," *Commun. Assoc. Comput. Mach.* **26**, 1008–1021.

[28] Rofheart, M. (1991), *Algorithms and Methods for Multidimensional Digital Signal Processing*, Ph.D. thesis, the City University of New York.

[29] Swarztrauber, P.N. (1982), "Vectorizing the FFTs," *Paral. Comput.*, Rodrique, G., ed., Academic Press, New York.

[30] — (1986), "Multiprocessor FFT's," *Paral. Comput.*

[31] Temperton, C. (1985), "Implementation of Self-Sorting In-place Prime Factor FFT Algorithm," *J. Comp. Phys.* **58**, 283–299.

[32] — (1991), "Self-Sorting In-place Fast Fourier Transforms," *Siam J. Sci. Stat. Comput.* **12**(4), 6–23.

[33] Tolimieri, R., An, M., and Lu, C.(1989), *Algorithms for Discrete Fourier Transform and Convolutions*, Springer-Verlag, New York.

[34] Zapata, E.L. et al. (1990), "Multidimensional Fast Fourier Transform into SIMD Hypercubes," *Pro. IEE* **137**(4), 253–260.

[35] *AT&T DSP Parallel Processor BT-100*, AT&T, Whippany, NJ, 1988.

[36] *NCUBE 6400 Processor Handbook*, NCUBE Co., Beaverton, Oregon, 1989.

[37] *Intel iPSC/2*, Intel Scientific Computers, Beaverton, 1988.

[38] *Intel iPSC/860 User's Guide*, Intel Co., June, 1990.

Index