INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9130365

Algorithms and methods for multidimensional digital signal processing

Rofheart, Martin, Ph.D.

City University of New York, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Algorithms and Methods for Multidimensional Digital Signal Processing

by

Martin Rofheart

A dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1991

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

3-18-91
Date

Richard Tolmach
Chair of Examining Committee

3/18/91
Date

Gerard J. Lovey
Executive Officer

Michael Anshel

Joseph Barba

Michael Conner

James Cooley

Tarek Saadawi

Supervisory Committee

The City University of New York

Abstract

Algorithms and Methods for Multidimensional Digital Signal Processing

by

Martin Rofheart

Advisor: Professor Richard Tolimieri

One of the major issue of parallel processing is the design of algorithms that minimize interprocessor communications. This research addresses this issue with respect to the parallel computation of the multidimensional discrete Fourier transform.

A tensor product formulation for multidimensional Cooley-Tukey type fast Fourier transform algorithms is developed. The formulation depends on the expression of data flow by the coset decomposition of the underlying index set. This representation allows fast algorithms to be designed by algebraic manipulations of the tensor product formulation.

A new reduced transform algorithm for the computation of the multidimensional discrete Fourier transform is developed. The algorithm computes a $d-$dimensional discrete Fourier transform by a set of independent $k-$dimensional discrete Fourier transforms ($k < d$); it is a reduction algorithm in the sense that it has lowered the dimension of the Fourier transforms that are computed. The $k-$dimensional discrete Fourier transforms are performed on data derived from the input using only additions, and produce $k-$dimensional hyperplanes of the output array.

The major contribution of this research is an intrinsically parallel algorithm for the computation of the $d-$dimensional DFT. The main features of the algorithm are that it requires no interprocessor communications, and that it maps to the degree of parallelism of the target multiprocessor flexibly.

The mapping of the algorithm onto architectures with broadcast and report capabilities is given. Expressions are obtained for estimating the speed on these machines as a function of the size of the $d-$dimensional DFT, the bandwidth C of the communications channel, the time A for an addition, the time $T(FFT)$ for a single processing element to perform a $k-$dimensional DFT ($k < d$), and the degree of parallelism of the machine. For single I/O channel machines that are capable of exploiting the full degree of parallelism of the algorithm, execution times as low as the time required to compute a single $k-$dimensional DFT plus the I/O time for data upload and download are attainable.

Contents

Chapter 1

Introduction

Over the past generation our ability to compute has grown dramatically. Integrated circuit (IC) technology has been the prime mover behind this growth. In the last ten years the complexity of ICs has increased from the hundred thousand transistor range to over the million mark, moving from VLSI to ULSI. Over this same period the cost of computation has fallen just as dramatically.

The computing machines being constructed using this technology span a broad range of design philosophies, each trying to achieve maximal performance in some area, and each exploiting some forms of concurrency, in space, in time or in both. Processor designs range from complex-instruction-set-computers (CISC) to reduced-instruction-set-computers (RISC) and dedicated digital signal processors (DSP). Typically these processors incorporate such advanced features as pipelining, multiple execution units, and multiple bus structures. Highly parallel computing structures are being composed of these processors, or more dedicated processing elements. With continued improvements in VLSI, computer, and algorithm design, these computing structures are attaining computational performance that dwarfs that of yesterday's supercomputers.

1.1 Issues of Concurrent Processing

VLSI circuits are the major building blocks of parallel systems. The main limitation of VLSI circuits has been identified as communications [61]. In VLSI, communications refers to the interconnections between computational elements. Compared to computation and storage, communication is expensive in both area and time, the performance metrics of the VLSI model [58]. The primary causes of this are that the number of I/O pins on a VLSI device is limited, the area on pad drivers required for amplification and internal wiring is large, and the capacitance of wiring introduces signal delays and dissipates power.

As a result of the restriction on communications, structures that localize communications have more favorable cost/performance characteristics than those that do not. The structures developed to satisfy this cost criterion typically feature a collection of homogenous processing elements together with an interconnection network of a regular topology. Specific cases of these structures include the butterfly [68], Cube-Connected-Cycles [49], hypercube [59], mesh [65], and tree [13] networks.

Parallel machines are described in terms of their size, the complexity of the computations that can be performed at each processing element, and in terms of the degree of interaction between the processing elements. These properties are defined by the *granularity*, *degree of parallelism*, and *coupling* of the machine.

The *granularity* [30] of a machine describes the computational complexity of the PEs. Typically, granularity is categorized as *fine grain*, *medium grain*, and *coarse grain*. These are described below.

Fine Grain refers to PEs that perform operations at the bit level. These are bit-serial architectures [8].

Medium Grain refers to PEs that perform operations at the word level. These include systolic arrays [36,35].

Coarse Grain refers to PEs that perform operations at the task level. Parallelism is exhibited by the processors handling different subtasks, and results are obtained by high level team work [24].

The amount of parallelism inherent in a computer is referred to as its *degree of parallelism.* In the case of a multiprocessor, the degree of parallelism refers to the number of independent processing elements. The interaction between processors is described in terms of how they are *coupled.* Systems that feature limited interaction between the processors are termed *Loosely coupled.* This class includes multiprocessors communicating over an interconnection network. *Tightly coupled* systems have a high degree of interaction between the processors. Shared memory machines are examples of tightly coupled systems.

At the processor level, the current trend in VLSI is toward fabrication of more powerful and sophisticated processing elements at reduced costs. However, at the communications level the interconnect structures that are available have either high wire and switch counts, or restricted interprocessor communications. The parallel computing structures whose cost/performance ratios benefit the most from these trends are coarse-grained loosely-coupled multiprocessors. These machines typically have powerful processors at each node, and an interconnection network to provide interprocessor communication [30].

1.2 Multidimensional Signal Processing

Multidimensional signal processing is an emerging field that contains a problem set whose data storage and computational requirements are far beyond those of its one-dimensional counterpart. Feasible solutions to the problems of this field depend on advances in computing machinery and the algorithms they support.

The multidimensional discrete Fourier transform (MD DFT) is one of the major tools for obtaining solutions to multidimensional digital signal processing problems. Applications of the MD DFT include radar [46], sonar [46], beamforming [66], weather forecasting, computer vision [48], and crystallography [1], among others.

1.3 Research Overview

Current technology enables parallel machines that have the computational ability of supercomputers to be constructed from many low-cost VLSI circuits. These are typically coarse-grain distributed memory machines. Examples of these include the AT&T BT-100 [3] and DSP3, Intel Hypercube [31], NCUBE 6400 [41], Encore Multimax, and others. These machines exhibit the inherent characteristics of VLSI machines; they have large computational power but limited communications bandwidth. To highlight the disparity between computation and communication consider that the Intel Hypercube, composed of up to 128 i860 processors, can achieve 7.5 GFlops, while its aggregate point to point communications bandwidth is only 2.8 Mbytes per channel.

These machines require algorithms that minimize interprocessor communications. This research addresses this problem from two directions. The first is the creation of tools to facilitate the design of multidimensional algorithms. The form this takes is a multidimensional extension of the tensor product formulation for Cooley-Tukey type FFT algorithms. The second is the creation of new algorithms specifically designed to eliminate interprocessor communications.

1.3.1 Summary of Contributions

This dissertation contributes three new results. The first of these is a multidimensional generalization of the tensor product formulation of Cooley-Tukey type FFT algorithms. This formulation provides a tool for both the expression and exploration of multidimensional Cooley-Tukey type algorithms. The one-dimensional tensor formulation has already been used to great advantage in designing FFT algorithms. This is due primarily to its concise treatment of the dataflow of FFT algorithms. The multidimensional extension of the tensor formulation finds similar use when designing MD FFT algorithms.

The second contribution is the development of a new reduced transform algorithm for multidimensional DFT computation. The algorithm computes the $d-$dimensional DFT by restricting it to a collection of $k-$dimensional hyperplanes that

cover the output array. The derivation of the algorithm is undertaken in two parts. The first part defines a minimal set of $k-$dimensional hyperplanes that cover the output array. The second part restricts the $d-$dimensional DFT of each of the hyperplanes in the covering set. This gives rise to a set of independent $k-$dimensional DFT. These are computed on data derived from the input using only additions, and produce $k-$dimensional hyperplanes of the output. The algorithm requires the $d-$dimensional array to be the same size in at least $d-k+1$ dimensions. The dimensions that are equal must be of either prime or power of prime (including 2) size.

The major contribution of this research is a multiprocessor algorithm for computing the $d-$dimensional discrete Fourier transform. The main features of the algorithm are that it requires no interprocessor communications, and that it maps to machine size flexibly. The kernel of the multiprocessor algorithm is the hyperplane algorithm developed in this work. A direct mapping of the hyperplane algorithm to a multiprocessor machine model is given. This method is shown to produce a parallel algorithm that requires no interprocessor communications, but imposes limiting restrictions on the degree of parallelism of the target processor. In order to obtain a multiprocessor algorithm with a flexible degree of parallelism, the hyperplane algorithm is married to the multidimensional Cooley-Tukey algorithm. The resulting composite algorithm requires no interprocessor communications and has a flexible degree of parallelism.

1.3.2 Organization

This dissertation is organized as follows: Chapter 2 reviews works related to this research. The review is presented in two parts: the first part defines the pertinent mathematical formulations, and the second part reviews parallel algorithms for MD DFT computation that have been proposed in the literature. Chapter 3 introduces a new tensor product formulation for MD DFT algorithms that exploits the additive properties of the underlying index set. The factorizations of the DFT induced by these algorithms are shown to depend on dual coset decompositions of the input

and output index groups. This fact is used to define a natural representation of the algorithms. Chapter 4 introduces a new reduction algorithm for MD DFT computation. The algorithm is based on the restriction of the $d-$dimensional DFT to a collection of $k-$dimensional hyperplanes that cover the output array. Chapter 5 introduces a parallel algorithm based on the hyperplane algorithm of chapter 4. The algorithm is shown to require no interprocessor communication. A variant of the algorithm is derived that couples the hyperplane algorithm with multidimensional Cooley-Tukey methods. This development is presented in terms of the tensor product formulation of chapter 3. Chapter 6 discusses future research and possible extensions of the methods developed in this work to other computations.

Chapter 2

Theoretical Foundation

Overview

Algorithms that compute the multidimensional discrete Fourier transform (MD DFT) have been roughly categorized as row-column, multidimensional variants of Cooley-Tukey (MD CT), and reduced transform algorithms (RTA). Parallel implementations of the row-column and multidimensional Cooley-Tukey algorithms have been widely studied in the literature. In section 2.3 these implementations are discussed in terms of their communications requirements under a distributed computation model. The feature of these algorithms most pertinent to this research is that they alternate stages of computation with stages of global data exchange.

The row-column algorithm (section 2.1.3) evaluates the $N \times N$ 2D DFT by computing the 1D $N-$point DFT of the rows and columns of the 2D array [10]. Parallel implementations of this algorithm (section 2.3) require a global transpose operation between the row and column FFT stages. Specific implementations of this algorithm differ in the machine model and method used to perform the transpose. For a distributed machine model, the global transpose operation requires every processing element to exchange data with every other processing element. Interconnection networks that can transpose an $N \times N$ array distributed on N processors in $O(N)$ time require $O(N \log N)$ switches [32].

The Cooley-Tukey [16] algorithm (section 2.1.4) was extended to multiple dimensions in [50,26,29]. A unified treatment of these is given in [38]. As in the one

dimensional case, the additive properties of the indexing set are exploited to factor the MD DFT into alternating stages of lower order MD DFT, twiddle multiplication, and data permutation. This decomposition process can be applied recursively to produce algorithms whose dataflow is optimized for a specific architecture [60,32], and that have a reduced number of multiplies relative to the row-column method. However, all variants of the multidimensional Cooley-Tukey algorithm require alternating stages of DFT with stages of data exchange. Like the row-column method, extensive interprocessor communications is required.

Reduced transform algorithms (RTA) will be considered as alternative methods of computation. These algorithms apply number theory or polynomial theory to the computation of the multidimensional DFT. They are reduced in the sense that they compute a $d-$dimensional DFT with fewer 1D DFTs than the dN^{d-1} 1D DFTs required by the row-column method. These algorithms include the polynomial transform algorithm [44,45], the multidimensional AFW algorithm [5], the weighted redundancy transform (WRT) [67,64], and the linear congruence algorithm [20]. Some RTA are shown to possess structures that do not intersperse computation stages with communication stages. The algorithms that satisfy this constraint can be implemented on distributed processors with no interprocessor communication. These algorithms have neither the time nor the hardware cost associated with the data exchange stages of row-column and multidimensional Cooley-Tukey algorithms.

The remainder of this chapter is organized as follows. First, the mathematical preliminaries are presented. Then, reduced transform algorithms are discussed, and a parametric line formulation of the linear congruence algorithm is described. A constructive approach is used to show that a particular instance of this algorithm matches up with the Nussbaumer–Quandalle polynomial transform algorithm. The chapter closes by considering the parallel algorithms that have appeared in the literature. The presentation of these is loosely organized by machine model. The main distinction between the algorithms is seen to be how the communication requirement is handled.

2.1 Mathematical Background

The next few sections present the mathematical entities that are pertinent to this research. The presentation includes the tensor product formulation of the one dimensional Cooley-Tukey FFT, the definition of the multidimensional DFT, the row-column algorithm, and the multidimensional Cooley-Tukey FFT algorithm. This information is required for the development of a new multidimensional FFT tensor product formulation introduced in chapter 3.

2.1.1 Tensor Product Formulation of the Cooley-Tukey FFT

The use of tensor products as a tool for exploring the structure and expressing the formulation of Cooley-Tukey FFT algorithms [16] was begun by Pease in [47]. Tensor products have since been applied to this task on various architectures. Some of the more recent applications are presented in [60,54,43,25]. A definitive work on the application of the language of tensor products to Cooley-Tukey type FFT algorithms is D. Rodriguez [51].

The Cooley-Tukey FFT, and its variants, are composed of three types of stages that are interspersed throughout the computation. These are (1) data movement stages that are expressed by stride permutation matrices, (2) multiplication stages that are expressed by twiddle matrices, and (3) stages of lower order DFT that are expressed in terms of matrices derived from those DFT. The tensor product is used to build up the formulation from these components. Recursive application of the tensor product factorization and application of algebraic properties of the tensor product allow variants of CT type algorithms to be realized. The definition of the tensor product of matrices is given below, then the components of the factorization are developed in terms of it.

The Tensor Product of Matrices

The tensor product of the n and m element vectors

$$\underline{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \quad \text{and} \quad \underline{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{bmatrix}$$

is the nm element vector $\underline{x} \otimes \underline{y}$ defined by

$$\underline{x} \otimes \underline{y} = \begin{bmatrix} x_0\underline{y} \\ \vdots \\ x_{n-1}\underline{y} \end{bmatrix}. \tag{2.1}$$

The tensor product of the $r \times r$ and $s \times s$ matrices A and B is the $rs \times rs$ matrix C defined by

$$C = A \otimes B = \begin{bmatrix} a_{0,0}B & \cdots & a_{0,r-1}B \\ \vdots & \ddots & \vdots \\ a_{r-1,0}B & \cdots & a_{r-1,r-1}B \end{bmatrix}. \tag{2.2}$$

Some useful properties of tensor products are [51]

$$(A \otimes B)(C \otimes D) = AC \otimes BD \tag{2.3}$$

$$[(A \otimes B)(C \otimes D)]^t = (C^t \otimes D^t)(A^t \otimes B^t). \tag{2.4}$$

Let the matrix A_i be of size $n_i \times n_i$, then a ramification of (2.3) is

$$A_1 \otimes A_2 = (A_1 \otimes I_{n_2})(I_{n_1} \otimes A_2) \tag{2.5}$$

$$A_1 \otimes A_2 \otimes \cdots \otimes A_d = \prod_{i=1}^{d} (I_{\prod_{j=0}^{i-1} n_j} \otimes A_i \otimes I_{\prod_{j=i+1}^{d} n_j}). \tag{2.6}$$

If $n_i = n$, then (2.6) can be written

$$A_1 \otimes A_2 \otimes \cdots \otimes A_d = \prod_{i=0}^{d} (I_{n^i} \otimes A_i \otimes I_{n^{d-1-i}}).$$

One Dimensional DFT Definition

The $n-$point discrete Fourier transform (DFT) is defined by the summation

$$Y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk}, \tag{2.7}$$

where

$$\omega_n = e^{\frac{-2\pi i}{n}}, \quad 0 \leq k < n$$

and the matrix-vector product

$$\underline{y} = F_n \underline{x} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \tag{2.8}$$

where F_n defines the Fourier transform matrix.

Tensor Product FFT Formulation

The factorization of F_n into stages of lower order transforms, can be written as [51]

$$F_n = (F_s \otimes I_r) T_{n,s} (I_s \otimes F_r) P_{n,s} \tag{2.9}$$

where

$$n = rs.$$

The stages of this decomposition are defined below.

1. The stride by s permutation $P_{n,s}$. This permutation reorders the elements of a vector $\underline{x}$ so that all elements that had previously been s apart are contiguous

$$(x_0, x_s, \ldots, x_{n-s}, x_1, x_{s+1}, \ldots, x_{n-s+1}, \ldots, x_{n-1})^t = P_{n,s}\underline{x} \tag{2.10}$$

2. The lower order Fourier transform stage

$$I_s \otimes F_r = \begin{bmatrix} F_r & & & O \\ & F_r & & \\ & & \ddots & \\ O & & & F_r \end{bmatrix} \tag{2.11}$$

is computed by performing s independent $r-$point Fourier transforms.

3. The lower order Fourier transform stage

$$F_s \otimes I_r = \begin{bmatrix} I_r & I_r & \cdots & I_r \\ I_r & \omega_s I_r & \cdots & \omega_s^{s-1} I_r \\ \vdots & \vdots & & \vdots \\ I_r & \omega_s^{s-1} I_r & \cdots & \omega_s I_r \end{bmatrix} \tag{2.12}$$

is computed by performing an $s-$point Fourier transform on vectors of length r.

4. The twiddle factor matrix $T_{n,s}$. Where $n = rs$ this matrix is defined by the direct sum

$$T_{n,s} = \sum_{j=0}^{s-1} \oplus D_{n,r}^{j}, \tag{2.13}$$

$$D_{n,r} = diag[1, \omega_n, \ldots, \omega_n^{r-1}]$$

An important property of the stride permutations is the commutation theorem [51] restated below.

Theorem 2.1.1 *If A is an $r \times r$ matrix and B is an $s \times s$ matrix then*

$$P_{n,s}(A \otimes B)P_{n,r} = B \otimes A$$

This property can be used to convert the data flow of one type of DFT stage to the other. Consider

$$(F_s \otimes I_r) = P_{n,s}(I_r \otimes F_s)P_{n,r}.$$

This identity will appear throughout the text.

2.1.2 MD DFT Definition

The summation form of the multidimensional discrete Fourier transform is given by

$$Y(j_1, j_2, \ldots, j_d) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \cdots \sum_{k_d=0}^{n_d-1} x(k_1, k_2, \ldots, k_d) \omega_{n_1}^{j_1 k_1} \omega_{n_2}^{j_2 k_2} \cdots \omega_{n_d}^{j_d k_d} \tag{2.14}$$

$$0 \leq j_i, k_i < n_i, \quad \omega_{n_i} = e^{\frac{-2\pi\sqrt{-1}}{n_i}}.$$

Using the tensor product, the $d-$dimensional Fourier transform matrix $F_{\underline{n}}$, where $\underline{n} = (n_1, \ldots, n_d)$, can be written in terms of one dimensional Fourier transform matrices [1]. $F_{\underline{n}}$ is given by

$$F_{\underline{n}} = (F_{n_1} \otimes F_{n_2} \otimes \cdots \otimes F_{n_d}). \tag{2.15}$$

$F_{\underline{n}}$ operates on a vector $\underline{x}$ formed by writing down the elements of the $d-$dimensional input array $x_{\underline{k}}$ in lexicographic order by dimension.

Letting $n_i = N$, the direct computation of a single output point by the summation definition or by matrix multiplication requires N^d complex multiplies, and N^d complex additions. There are N^d output points, so the evaluation of the entire expression by this method requires $N^d N^d$ complex multiplies, and $N^d N^d$ complex additions. As in the 1D case fast algorithms exist for MD DFT. The most basic of these is the row-column algorithm.

2.1.3 Row-Column Algorithms

Without loss of generality the row-column algorithm will be defined in terms of the 2D DFT using the tensor product of matrices formulation. The row-column algorithm evaluates the $n \times n$ 2D DFT by computing the 1D $n-$point DFT of the rows and columns of the 2D array [10]. The $n \times n$ 2D DFT summation is defined as

$$Y(j_1, j_2) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} x(k_1, k_2) \omega_{n_1}^{j_1 k_1} \omega_{n_2}^{j_2 k_2}, \tag{2.16}$$

and the 2D Fourier transform matrix is

$$F_{\underline{n}} = F_{n,n} = (F_n \otimes F_n). \tag{2.17}$$

Applying the identity of (2.5) to (2.17) yields

$$F_{n,n} = (F_n \otimes I_n)(I_n \otimes F_n). \tag{2.18}$$

If the Fourier matrix $F_{(n,n)}$ is applied to a vector $\underline{x}$ formed from reading the rows of the 2D array $x(k_1, k_2)$, then equation (2.18) is seen to compute one dimensional F_n on the rows then columns of the input array. The row-column method is extended

to any number of dimensions by performing 1D Fourier transforms with respect to each dimension.

There are dN^{d-1} one dimensional Fourier transforms requiried to evaluate the $d-$dimensional Fourier transform by the row-column method. If they are computed directly then $O(dN^{d+1})$ arithmetic operations are required. If $N = 2^n$ and they are computed using the FFT, then the complexity is $O(dN^d \log N)$. This represents a considerable improvement over the $O(N^{2d})$ operations required for direct evaluation of (2.16) or (2.15).

2.1.4 Multidimensional Cooley-Tukey Algorithms

The row-column method applies the Cooley-Tukey FFT algorithm in each dimension to compute the MD DFT. For the 2D case it applies the FFT row-wise then column-wise. Essentially this decimates and transforms the array in each index separately. In [26] a derivation of an algorithm which decimates and transforms both indecies simultaneously is given. This algorithm is a multidimensional Cooley-Tukey algorithm in the sense that it exploits the additive properties of the underlying indexing set to decompose the MD DFT into stages of lower order DFT, stages of permutation, and stages of twiddle multiplication. Below, the multidimensional Cooley-Tukey algorithm is derived for the 2D case, then extended to MD cases. In chapter 3 a new tensor product formulation of the MD CT algorithm is presented.

The $N_1 \times N_2$ 2D DFT is given by

$$Y(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) \omega_{N_1}^{k_1 n_1} \omega_{N_2}^{k_2 n_2}. \tag{2.19}$$

For N_1 and N_2 composite, $N_1 = r_1 s_1$ and $N_2 = r_2 s_2$ let

$$\begin{aligned} n_1 &= s_1 p_1 + u_1, \quad n_2 = s_2 p_2 + u_2 \\ 0 &\le p_i < r_i, \quad 0 \le u_i < s_i. \end{aligned} \tag{2.20}$$

Substituting (2.20) into (2.19)

$$\begin{aligned} Y(k_1, k_2) \;=\; & \sum_{u_1=0}^{s_1-1} \sum_{u_2=0}^{s_2-1} \omega_{N_1}^{k_1 u_1} \omega_{N_2}^{k_2 u_2} \\ & \cdot \sum_{p_1=0}^{r_1-1} \sum_{p_2=0}^{r_2-1} x(s_1 p_1 + u_1, s_2 p_2 + u_2) \omega_{r_1}^{k_1 p_1} \omega_{r_2}^{k_2 p_2} \end{aligned} \tag{2.21}$$

where

$$\omega_{N_i}^{k_i(s_ip_i+u_i)} = \omega_{r_i}^{k_ip_i}\omega_{N_i}^{k_iu_i}.$$

Repeating the same procedure on the output indexes, let

$$\begin{aligned} k_1 = a_1 + r_1b_1, \quad k_2 = a_2 + r_2b_2 \\ 0 \le a_i < r_i, \quad 0 \le b_i < s_i. \end{aligned} \tag{2.22}$$

Substituting (2.22) into (2.21) gives the basic two-dimensional Cooley-Tukey decomposition

$$\begin{aligned} Y(a_1 + r_1b_1, a_2 + r_2b_2) \;=\; & \sum_{u_1=0}^{s_1-1}\sum_{u_2=0}^{s_2-1} \omega_{s_1}^{u_1b_1}\omega_{s_2}^{u_2b_2}\omega_{N_1}^{a_1u_1}\omega_{N_2}^{a_2u_2} \\ & \cdot \sum_{p_1=0}^{r_1-1}\sum_{p_2=0}^{r_2-1} x(s_1p_1 + u_1, s_2p_2 + u_2)\omega_{r_1}^{a_1p_1}\omega_{r_2}^{a_2p_2} \end{aligned} \tag{2.23}$$

where

$$\omega_{N_i}^{(a_i+r_ib_i)u_i} = \omega_{s_i}^{u_ib_i}\omega_{N_i}^{a_iu_i}.$$

The decomposition of (2.24) extends naturally to multidimensional transforms that have a composite number of points in each dimension. Let

$$N_i = s_ir_i, \; k_i = a_i + r_ib_i, \;\text{ and } n_i = s_ip_i + u_i \tag{2.24}$$

where

$$0 \le a_i, p_i < r_i, \quad 0 \le b_i, u_i < s_i, \quad i = 1, 2, \ldots, d$$

then the $d-$dimensional DFT given by (2.14) can be written

$$\begin{aligned} & Y(a_1 + r_1b_1, \ldots, a_d + r_db_d) = \\ & \sum_{u_1=0}^{s_1-1} \cdots \sum_{u_d=0}^{s_d-1} \omega_{s_1}^{u_1b_1} \cdots \omega_{s_d}^{u_db_d}\omega_{N_1}^{a_1u_1} \cdots \omega_{N_d}^{a_du_d} \\ & \cdot \sum_{p_1=0}^{r_1-1} \cdots \sum_{p_d=0}^{r_d-1} x(s_1p_1 + u_1, \ldots, s_dp_d + u_d)\omega_{r_1}^{a_1p_1} \cdots \omega_{r_d}^{a_dp_d} \end{aligned} \tag{2.25}$$

The next chapter will present a new tensor product formulation of the basic MD CT factorization.

2.2 Reduced Transform Algorithms

Reduced transform algorithms (RTA) take their name from the fact that they reduce the number of one-dimensional DFT needed to compute the MD DFT below the dN^{d-1} required by the row-column method. These algorithms apply number theory or polynomial theory to the computation of the MD DFT. They include the Nussbaumer–Quandalle polynomial transform algorithm [44,45], the multidimensional algorithm [5] of Auslander *et al.*, the weighted redundancy transform (WRT) [67,64] of Vulis and Tsai, and the linear congruence algorithm of Gertner [20].

In [5] an algorithm for the computation of multidimensional transforms is introduced. The algorithm depends on the underlying indexing set possessing finite field properties and is therefore restricted to arrays of prime size in each dimension. The WRT algorithm [67,64] depends on ring properties of the indexing set, and operates on any $M \times N$ array. This algorithm computes the MD DFT in terms of a set of lines that cover the output array. The output lines are formed by summation from a set of 1D DFT computed on lines covering the input array. If the computation of the 1D DFT are performed in parallel, then interprocess communication must occur to form the output lines.

The Nussbaumer-Quandalle algorithm [45,44] uses polynomial transforms to map the multidimensional DFT into a set of independent 1D DFT, and the linear congruence algorithm of [20] computes the 2D DFT by a set of independent 1D DFT along linear congruences over the indexing set. Both algorithms operate on 2D square arrays of prime, power of prime, and product of prime sizes in each dimension. Both of these algorithms evaluate the 2D DFT by a set of independent 1D DFT computed on data derived from the input set using only additions. Parallel algorithms that require no interprocessor communication may be developed using these reduction algorithms.

The remainder of this section is devoted to the parametric line formulation of the linear congruence algorithm of [20]. This formulation of the 2D DFT directly restricts computation to a set of lines covering the output grid. A constructive approach is used to show that a specific case of this algorithm matches up with the

Nussbaumer–Quandalle polynomial transform algorithm.

2.2.1 The Parametric Form of the Linear Congruence Algorithm

Below, the restriction of the 2D DFT to a set of lines covering the output grid is described [1]. Using the parametric form of the line, expressions are derived giving the covering set of lines, as well as the redundancy between lines. This formulation is identical to the linear congruence algorithm of [20]. In later work this formulation will be used to express a parallel algorithm for 2D DFT computation with no interprocessor communication. An example is provided at the end of the section.

Prime Case

The $P \times P$ 2D DFT will be computed by restricting it to a set of lines over the indexing set $Z/P \times Z/P$. In general, the line through a point $\underline{a} = (a_1, a_2) \in Z/P \times Z/P$ is defined as the (cyclic) subgroup generated by $\underline{a}$

$$L(\underline{a}) = L((a_1, a_2)) = \{t\underline{a} = (a_1 t, a_2 t) : t = 0, 1, \ldots, P-1\} \tag{2.26}$$

The following theorem gives a set of P point radial lines which cover $Z/P \times Z/P$ [1].

Theorem 2.2.1 *The set of $P+1$ lines*

$$\begin{aligned} &L((m,1)), \qquad m = 0, 1, \ldots, p-1 \\ &L((1,0)) \end{aligned}$$

cover $Z/P \times Z/P$.

Proof. Let $\underline{a} = (a_1, a_2)$ be any element of $Z/P \times Z/P$. Two cases of a_2 describe all points $\underline{a}$.

$a_2 \neq 0$: Then a_2 is invertible modulo p and $\underline{a} = (a_1, a_2)$ is on the line $L((a_2^{-1}a_1, 1))$.

$a_2 = 0$: Then $(a_1, 0)$ is on the line $L((1,0))$.

□

The $P \times P$ 2D DFT can be computed by restricting it to lines covering the $P \times P$ output grid. The points on the lines $L((m,1))$ are evaluated by

$$V(mt,t) = \sum_{k_1=0}^{P-1} \sum_{k_2=0}^{P-1} x(k_1,k_2)\omega_P^{(k_1m+k_2)t}. \tag{2.27}$$

Using the change of variables:

$$\begin{aligned} i &= k_1 \\ d &= k_1m + k_2 \\ k_2 &= d - mi \end{aligned} \tag{2.28}$$

equation (2.27) can be written

$$V(mt,t) = \sum_{i=0}^{P-1} \sum_{d=0}^{P-1} x(i,d-mi)\omega_P^{dt}. \tag{2.29}$$

Similarly, the points on the line $L((1,0))$ are given by

$$V(t,0) = \sum_{d=0}^{P-1} \omega_P^{dt} \sum_{i=0}^{P-1} x(d,i). \tag{2.30}$$

Rearranging terms and identifying

$$a_d^{(m,1)} = \sum_{i=0}^{P-1} x(i,d-mi) \tag{2.31}$$

in equation (2.29), and

$$a_d^{(1,0)} = \sum_{i=0}^{P-1} x(d,i) \tag{2.32}$$

in equation (2.30), the $P \times P$ 2D DFT is seen to be computed by

$$V(mt,t) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(m,1)}, \qquad m = 0,1,\ldots,P-1 \tag{2.33}$$

$$V(t,0) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(1,0)}, \qquad t = 0,1,\ldots,P-1 \tag{2.34}$$

Equations (2.33) and (2.34) evaluate the $P \times P$ 2D Fourier transform with $P+1$ one dimensional Fourier transforms on data derived from the input by the preadditions of (2.31) and (2.32).

By inspection the only redundancy for this case is that the output point $(0,0)$ is common to each of the $P+1$ independent $P-$point DFT.

Power of Prime Case

In a manner similar to the prime case, the $p^n \times p^n$ 2D DFT can also be computed by restricting it to a set of lines over the indexing set $Z/p^n \times Z/p^n$. In general, the line through a point $\underline{a} = (a_1, a_2) \in Z/p^n \times Z/p^n$ is defined as the (cyclic) subgroup generated by $\underline{a}$

$$L(\underline{a}) = L((a_1, a_2)) = \{t\underline{a} = (a_1 t, a_2 t) : t = 0, 1, \ldots, p^n - 1\}. \tag{2.35}$$

The following theorem gives a set of p^n point radial lines that cover $Z/p^n \times Z/p^n$ [22].

Theorem 2.2.2 *The set of* $p^n + p^{n-1}$ *lines*

$$L((m, 1)), \qquad 0 \leq m < p^n$$

$$L((1, ps)), \qquad 0 \leq s < p^{n-1}$$

cover $Z/p^n \times Z/p^n$.

Proof. For any $\underline{a} = (a_1, a_2) \in Z/p^n \times Z/p^n$, let α and β be the maximums such that $p^\alpha | a_1$ and $p^\beta | a_2$. The following describe all points.

$\alpha \geq \beta$ **and** $a_2 \neq 0$: Then a_2/p^β is invertible modulo p^n and the point $\underline{a}$ may be written

$$\underline{a} = (a_1, a_2) = p^\beta \left(\frac{a_1}{p^\beta}, \frac{a_2}{p^\beta}\right) = a_2 \left(\frac{a_1}{p^\beta}(\frac{a_2}{p^\beta})^{-1}, 1\right)$$

All such $\underline{a}$ will be contained in the union of the p^n lines $L((m, 1))$.

$\alpha < \beta$ **and** $a_1 \neq 0$: Then a_1/p^α is invertible modulo p^n and the point $\underline{a}$ may be written

$$\underline{a} = (a_1, a_2) = p^\alpha \left(\frac{a_1}{p^\alpha}, \frac{a_2}{p^\alpha}\right) = a_1 \left(1, (\frac{a_1}{p^\alpha})^{-1} \frac{a_2}{p^\alpha}\right)$$

where

$$\left(\frac{a_1}{p^\alpha}\right)^{-1} \left(\frac{a_2}{p^\alpha}\right) = p \left(\frac{a_1}{p^\alpha}\right)^{-1} \left(\frac{a_2}{p^{\alpha+1}}\right) = ps$$

since

$$\left(\frac{a_1}{p^\alpha}\right)^{-1} \left(\frac{a_2}{p^{\alpha+1}}\right) = cp^{n-1} + s, \qquad 0 \leq s < p^{n-1}$$

All such $\underline{a}$ will be contained in the union of the p^{n-1} lines $L((1, ps))$.

$a_1 = 0$ **or** $a_2 = 0$: By definition the point $\underline{a}$ lies on the line $L((0,1))$ if $a_1 = 0$ or the line $L((1,0))$ if $a_2 = 0$.

□

Below, the redundancy of the covering set of lines of theorem 2.2.2 is considered. Since there are p^{2n} points in the $p^n \times p^n$ 2D grid, and the covering set contains $p^n + p^{n-1}$ lines, each of p^n points, there are p^{2n-1} redundant points in the set. The following three theorems fully describe the redundancy between any two lines of the covering set of theorem 2.2.2.

Theorem 2.2.3 *The redundancy between any two lines of the set* $L((m,1))$ *is*

$$L((j,1)) \cap L((k,1)) = L((p^{n-\alpha}(j_{\bmod p^\alpha}), p^{n-\alpha})), \quad \overset{\max}{\alpha}: j \equiv k \bmod p^\alpha$$

$$0 \leq j,k < p^n$$

Proof. By definition the points on the lines $L((j,1))$ and $L((k,1))$ are equal only when $tj = tk$ modulo p^n. If $t = 0$ the point $(0,0)$ is common. If $t \neq 0$ and $p \not| t$ then t is invertible modulo p^n and common points exist only for $j = k$. Otherwise there is some β such that t/p^β is invertible modulo p^n, and common points exist if $j = k$ modulo $p^{n-\beta}$.

Let $\alpha = n - \beta$, then the points common to the lines $L((j,1))$ and $L((k,1))$ can be written $c_1(p^{n-\alpha}(k_{\bmod p^\alpha}), p^{n-\alpha})$. The subgroup defined by these points is maximal for maximum α satisfying $j = k$ modulo p^α.

□

Theorem 2.2.4 *The redundancy between any two lines of the set* $L((1,sp))$ *is*

$$L((1,pj)) \cap L((1,pk)) = L((p^{n-\alpha-1}, p^{n-\alpha}(j_{\bmod p^\alpha}))), \quad \overset{\max}{\alpha}: j \equiv k \bmod p^\alpha$$

$$0 \leq j,k < p^{n-1}$$

Proof. By definition the points on the lines $L((1,pj))$ and $L((1,pk))$ are equal only when $ptj = ptk$ modulo p^n. As was the case for the lines $L((m,1))$, if $t = 0$ the point $(0,0)$ is common. If $p \not| t$ then the only common points exist for $j = k$. Otherwise

there is some β such that t/p^{β} is invertible modulo p^n, and common points exist for $j = k$ modulo $p^{n-\beta-1}$.

Let $\alpha = n - \beta - 1$, then the points common to the lines $L((1, pj))$ and $L((1, pk))$ can be written $c_1(p^{n-\alpha-1}, p^{n-\alpha}(k_{\bmod p^{\alpha}}))$. The subgroup defined by these points is maximal for maximum α satisfying $j = k$ modulo p^{α}.

□

Theorem 2.2.5 *The redundancy between lines from the set $L((m, 1))$ and lines from the set $L((1, sp))$ is*

$$L((j, 1)) \cap L((1, pk)) = (0, 0)$$

$$0 \leq j < p^n, \text{ and } 0 \leq k < p^{n-1}$$

Proof. By definition the points on these lines are given by (tj, t) and (t, tpk). For these points to be equal both $tj = t$ and $t = tpk$ must hold modulo p^n. Consider $t = tpk$. Since for any t there exists a maximum β such that $p^{\beta}|t$, it is implied that $pk \equiv 1 \mod p^{n-\beta}$ and therefore $k = p^{-1}$ modulo $p^{n-\beta}$. Since p^{-1} does not exist modulo p^v the proof is completed.

□

The $p^n \times p^n$ 2D DFT can be computed by restricting it to lines covering the $p^n \times p^n$ output grid. The output points on the lines $L((m, 1))$ are evaluated by

$$V(mt, t) = \sum_{k_1=0}^{p^n-1} \sum_{k_2=0}^{p^n-1} x(k_1, k_2) \omega_{p^n}^{(k_1 m + k_2)t}. \tag{2.36}$$

Let

$$\begin{aligned} i &= k_1 \\ d &= k_1 m + k_2 \\ k_2 &= d - mi \end{aligned} \tag{2.37}$$

Substituting (2.37) into (2.36), the 2D DFT restricted to the line of output $L((m, 1))$ is written:

$$V(mt, t) = \sum_{i=0}^{p^n-1} \sum_{d=0}^{p^n-1} x(i, d - mi) \omega_{p^n}^{dt}. \tag{2.38}$$

In a similar manner, the points on the line $L((1, sp))$ are given by

$$V(t, spt) = \sum_{d=0}^{p^n-1} \omega_{p^n}^{dt} \sum_{i=0}^{p^n-1} x(d - 2si, i). \tag{2.39}$$

Rearranging terms and identifying

$$a_d^{(m,1)} = \sum_{i=0}^{p^n-1} x(i, d - mi) \tag{2.40}$$

in equation (2.38), and

$$a_d^{(1,sp)} = \sum_{i=0}^{p^n-1} x(d - spi, i) \tag{2.41}$$

in equation (2.39), the $p^n \times p^n$ 2D DFT is seen to be computed by

$$V(mt, t) = \sum_{d=0}^{p^n-1} \omega_{p^n}^{dt} a_d^{(m,1)}, \quad 0 \le m < p^n \tag{2.42}$$

$$V(t, spt) = \sum_{d=0}^{p^n-1} \omega_{p^n}^{dt} a_d^{(1,0)}, \quad 0 \le s < p^{n-1} \tag{2.43}$$

$$0 \le t < p^n.$$

Equations (2.42) and (2.43) evaluate the $p^n \times p^n$ 2D Fourier transform by

$$\left(\frac{p+1}{p}\right) p^n$$

one dimensional Fourier transforms on data derived from the input by the preadditions of (2.40) and (2.41).

Example

The following is an example of the prime case of the algorithm.

Consider the computation of the 3 x 3 2D DFT

$$V(n_1, n_2) = \sum_{k_1=0}^{1} \sum_{k_2=0}^{2} x(k_1, k_2) \omega^{n_1 k_1} \omega^{n_2 k_2} \tag{2.44}$$

$$n_1, n_2 = 0, 1, 2$$

by the line algorithm. The output array $V(n_1, n_2)$ will be produced along four lines. These are shown by the arrays below. The label above each array defines the line

of output represented by the array. The boxed elements in each array identify the points contained in that line.

$$
\begin{array}{cc}
L((0,1)) & L((1,1)) \\
\begin{pmatrix} \boxed{v_{0,0}} & \boxed{v_{0,1}} & \boxed{v_{0,2}} \\ v_{1,0} & v_{1,1} & v_{1,2} \\ v_{2,0} & v_{2,1} & v_{2,2} \end{pmatrix} &
\begin{pmatrix} \boxed{v_{0,0}} & v_{0,1} & v_{0,2} \\ v_{1,0} & \boxed{v_{1,1}} & v_{1,2} \\ v_{2,0} & v_{2,1} & \boxed{v_{2,2}} \end{pmatrix} \\
L((2,1)) & L((1,0)) \\
\begin{pmatrix} \boxed{v_{0,0}} & v_{0,1} & v_{0,2} \\ v_{1,0} & v_{1,1} & \boxed{v_{1,2}} \\ v_{2,0} & \boxed{v_{2,1}} & v_{2,2} \end{pmatrix} &
\begin{pmatrix} \boxed{v_{0,0}} & v_{0,1} & v_{0,2} \\ \boxed{v_{1,0}} & v_{1,1} & v_{1,2} \\ \boxed{v_{2,0}} & v_{2,1} & v_{2,2} \end{pmatrix}
\end{array}
\tag{2.45}
$$

Each of the lines of output listed above is computed by performing a 3−point DFT on data derived from the input by additions. The preadditions required to generate line $L((x,y))$ of the output are given by the reduction operation $a_d^{(x,y)}$. These are listed below.

$$
\begin{array}{lcl}
 & a_d^{(0,1)} & \\
a_0^{(0,1)} & = & x_{0,0} + x_{1,0} + x_{2,0} \\
a_1^{(0,1)} & = & x_{0,1} + x_{1,1} + x_{2,1} \\
a_2^{(0,1)} & = & x_{0,2} + x_{1,2} + x_{2,2}
\end{array}
\qquad
\begin{array}{lcl}
 & a_d^{(1,1)} & \\
a_0^{(1,1)} & = & x_{0,0} + x_{1,2} + x_{2,1} \\
a_1^{(1,1)} & = & x_{0,1} + x_{1,0} + x_{2,2} \\
a_2^{(1,1)} & = & x_{0,2} + x_{1,1} + x_{2,0}
\end{array}
$$

$$
\begin{array}{lcl}
 & a_d^{(2,1)} & \\
a_0^{(2,1)} & = & x_{0,0} + x_{1,1} + x_{2,2} \\
a_1^{(2,1)} & = & x_{0,1} + x_{1,2} + x_{2,0} \\
a_2^{(2,1)} & = & x_{0,2} + x_{1,0} + x_{2,1}
\end{array}
\qquad
\begin{array}{lcl}
 & a_d^{(1,0)} & \\
a_0^{(1,0)} & = & x_{0,0} + x_{0,1} + x_{0,2} \\
a_1^{(1,0)} & = & x_{1,0} + x_{1,1} + x_{1,2} \\
a_2^{(1,0)} & = & x_{2,0} + x_{2,1} + x_{2,2}
\end{array}
$$

The lines of output shown in the arrays of (2.45) are produced by performing a 3−point DFT on each $a_d^{(x,y)}$ given above.

2.2.2 A Relationship Between the Nussbaumer–Quandalle Polynomial Algorithm and the Line Algorithm

This section shows that the Nussbaumer–Quandalle polynomial transform algorithm can be reduced to the line algorithm. A constructive approach is adopted to identify corresponding entities of the polynomial algorithm and the parametric line algorithm. Only the prime case of the algorithm is considered. The Nussbaumer-Quandalle algorithm depends on the following polynomial representation of the $P \times P$ 2D DFT, where (2.48) gives the 2D DFT.

$$X_{n_1}(z) = \sum_{n_2=0}^{P-1} x_{n_1,n_2} z^{n_2} \tag{2.46}$$

$$V_{k_1}(z) = \sum_{n_1=0}^{P-1} X_{n_1}(z)\omega^{n_1 k_1} \bmod (z^P - 1) \tag{2.47}$$

$$V_{k_1,k_2} = V_{k_1}(z) \bmod (z - \omega^{k_2}) \tag{2.48}$$

$$0 \leq k_1, k_2 < P$$

The first step of the algorithm requires the computation of the 2D DFT at $k_2 = 0$. Rearrange (2.47) by powers of z, write

$$V_{k_1}(z) = \sum_{i=0}^{P-1} b_i(k_1) z^i \tag{2.49}$$

where

$$b_i(k_1) = \sum_{d=0}^{P-1} x_{d,i}\omega^{dk_1} \tag{2.50}$$

which by the modulo operation of (2.48), gives V_{k_1,k_2} evaluated at $k_2 = 0$ as

$$V_{k_1,0} = \sum_{d=0}^{P-1} \omega^{dk_1} \sum_{i=0}^{P-1} x_{d,i} \tag{2.51}$$

Expression (2.51) is identically to the restriction of the 2D DFT to the line of output $L((1,0))$. This is the computation of $a_d^{(1,0)}$ given by equation (2.32) followed a 1D DFT of $P-$points on $a_d^{(1,0)}$.

The next stage of the algorithm computes the output for points where $k_2 \neq 0$. This stage is derived in two steps. The first reduces the order of the computation

by exploiting the fact that $(z^P - 1)$ can be written as the product of two cyclotomic polynomials for prime P

$$(z^P - 1) = (z - 1)P(z) \tag{2.52}$$

$$P(z) = \prod_{GCD(k_2,P)=1} (z - \omega^{k_2}) = \prod_{k_2=1}^{P-1} (z - \omega^{k_2})$$

Since $(z - \omega^{k_2})$ is a factor of $P(z)$ which is in turn a factor of $(z^P - 1)$, equation (2.48) can be rewritten

$$V_{k_1,k_2} = \Big(\big(V_{k_1} \bmod (z^P - 1)\big) \bmod P(z)\Big) \bmod (z - \omega^{k_2}) \tag{2.53}$$

which reduces equation (2.46) to

$$X^1_{n_1}(z) = X_{n_1}(z) \bmod P(z) = \sum_{n_2=0}^{P-2} (x_{n_1,n_2} - x_{n_1,P-1}) z^{n_2} \tag{2.54}$$

and equations (2.47) and (2.48) can be written

$$V^1_{k_1}(z) = \sum_{n_1=0}^{P-1} X^1_{n_1}(z)\omega^{n_1 k_1} \bmod P(z) \tag{2.55}$$

$$V_{k_1,k_2} = V^1_{k_1}(z) \bmod (z - \omega^{k_2}) \tag{2.56}$$

$$1 \le k_2 < P$$

The second step of this stage applies the permutation $k_1 k_2$ to equations (2.55) and (2.56) giving

$$V^1_{k_1 k_2}(z) = \sum_{n_1=0}^{P-1} X^1_{n_1}(z)\omega^{n_1 k_1 k_2} \bmod P(z) \tag{2.57}$$

$$V_{k_1 k_2,k_2} = V^1_{k_1 k_2}(z) \bmod (z - \omega^{k_2}) \tag{2.58}$$

The effect of the modulo operation of equation (2.58) is to replace z with ω^{k_2}. Thus equation (2.57) may be written

$$V^1_{k_1 k_2}(z) = \sum_{n_1=0}^{P-1} X^1_{n_1}(z) z^{n_1 k_1} \bmod P(z) \tag{2.59}$$

$$V_{k_1 k_2,k_2} = V^1_{k_1 k_2}(z) \bmod (z - \omega^{k_2}) \tag{2.60}$$

Equations (2.59) and (2.60) complete the algorithm. To show that these equations matchup to the line algorithm consider equation (2.59) written as

$$V^1_{k_1 k_2}(z) = \sum_{n_1=0}^{P-1} \sum_{n_2=0}^{P-2} b_{(n_1,n_2)} z^{(n_2+n_1 k_1)} \bmod P(z) \tag{2.61}$$

where

$$b_{(n_1,n_2)} = (x_{n_1,n_2} - x_{n_1,P-1}) \tag{2.62}$$

The coefficients of this polynomial are derived by adding the b_{n_1,n_2} associated with like powers of z. Rearranging equation (2.61) in this way yields

$$V^1_{k_1 k_2}(z) = \sum_{d=0}^{P-2} \sum_{i=0}^{P-1} b_{(i,(d-ik_1)_{mod\,P})} z^d \tag{2.63}$$

where the identifications

$$\begin{aligned} d &= n_2 + n_1 k_1 \\ i &= n_1 \end{aligned} \tag{2.64}$$

have been made. Performing the modulo operation of (2.60) gives

$$V_{k_1 k_2, k_2} = \sum_{d=0}^{P-2} \omega^{dk_2} \sum_{i=0}^{P-1} b_{(i,d-ik_1)} \tag{2.65}$$

The innermost summation matches up with $a_d^{(m,1)}$ given by equation (2.31) where m plays the role of k_1 and t plays the role of k_2, and the DFT stage is restricted to the dataset $b_{(n_1,n_2)}$.

2.3 Parallel Algorithms

This section reviews parallel algorithms for the computation of multidimensional discrete Fourier transform (MD DFT) that have appeared in the literature. The algorithms are categorized by their target architectures. These include algorithms for array and multiprocessor architectures, algorithms specialized to mesh, tree, heirarchical memory, shared memory, and ensemble architectures, and algorithms intended for special purpose and dedicated VLSI machines.

2.3.1 Array Processors

Array Processors [30] denote synchronous k-dimensional arrays of $P = Q^k$ processing elements (PE). These machines operate in a SIMD (single-instruction-multiple-data) [19] mode. In this mode the P processing elements are issued instructions by a single control unit. Each PE has a local memory, and the PEs communicate with each other over an interconnect network (IN) that connects neighbors in each dimension. Communications overhead plays a significant role in execution time, and is measured in terms of routing operations. These are defined as moving 1 datum from every PE to its neighbor in one of the dimensions.

Jesshope [33] examines 1D and 2D FFT algorithms on array processors. The performance of fine grain and course grain row-column and Cooley-Tukey FFT implementations are considered in terms of arithmetic operations and routing operations. In this context, course grain computations refer to those where complete FFTs are computed at each PE using local memory, and fine grain computations are those that distribute the computation across the PEs. Jesshope determines that the arithmetic count is similar for both granularities, but that the course grain computations achieve superior performance because they require many fewer routing operations. For this computational model he shows that the minimum number of routing operations occurs for course grain computations on hypercube INs, and increases exponentially with decreasing connectivity (number of dimensions).

Given this mode of processing, an $n \times n$ 2D DFT algorithm can be specified by applying the commutation theorem (2.1.1) to the 2D DFT definition of equation

(2.18) to obtain

$$F_{n,n} = P_{n^2,n}(I_n \otimes F_n)P_{n^2,n}(I_n \otimes F_n). \tag{2.66}$$

Each of the computation stages $(I_n \otimes F_n)$ describes n independent F_n, while the permutation $P_{n^2,n}$ defines the interprocessor communication required.

Jamieson *et al* [32] also consider 2D FFT implementations on SIMD array processors. They examine INs that support $cube_c$ and $shift_i$ interconnect functions [55]. $Cube_c$ specifies the interconnection between PEs whose addresses differ only in the c^{th} bit position, while $shift_i$ connects PE_x to $PE_{(x+i)modN}$. Using this model, the computation of the 2D FFT of $N \times N$ data on an N PE machine requires $N-1$ interconnect functions. This compares favorably with the number of arithmetic operations $O(Nlog_2N)$, but is expensive in hardware. INs capable of supporting one or both of these functions include: data manipulator[18], omega[37], Staran flip[7],generalized cube[56], and ADM[57]. The data manipulator requires log_2N stages of N switches, while the others require log_2N stages of $N/2$ switches. This represents a considerable cost and is frequently the most complex subsystem of the processor.

In [70] Zapata *et al.* optimize MD FFT for SIMD hypercubes. They develop schemes for minimizing the routing requirement of the data exchange stages of FFT algorithms. The algorithm was simulated on the NCUBE [40] and similar to Jesshope they reported that courser grained algorithms outperformed finer grained ones owing to their reduced routing complexity.

2.3.2 Multiprocessors

Multiprocessors are single computers composed of a collection of processing elements (PE) together with some means of interprocessor communication [30]. These machines operate in a MIMD (multiple-instruction-stream multiple-data-stream) [19] mode. Each processor has its own control, memory, and arithmetic units. Interprocessor communications are supported either by an interconnect network (IN) or a shared memory. Those machines which rely on INs for communication have been termed *loosely coupled* multiprocessors, while those that use shared memory

have been termed *tightly coupled.* This nomenclature is derived from the degree of interaction expected between processors in these systems.

Multiprocessors exist which operate in single-program multiple-data-stream SPMD mode [24]. This is a sliced procedure where the same program is executed simultaneously at all the PEs on possibly different datasets. The constraint this imposes is that all PEs begin processing at the same point and return from processing to the same

point. During processing, program flow is free to follow different paths based on the dataset.

2.3.3 Tree Processors

In [24] the concept of the balanced parallel algorithm is applied to the computation of the 2D DFT on tree machines. These are machines where each PE is connected to its parent, a right child and a left child. The processing mode is taken to be SPMD. The entire tree is assumed to be connected to a communications channel through the root. Identical PE's are distinguished by some position dependent labeling scheme ie. breadthfirst numbering. The communications functions required are broadcast and report. In broadcast mode the channel sends data to all the PEs in the tree. In report mode a PE distinguished by ID number sends data to the channel.

A balanced algorithm is one where the communication and processing time are equal. Such algorithms can achieve 100% machine utilization using task level pipelining. The algorithm implemented is the standard row-column 2D DFT realized by the following steps: The rows of input are distributed evenly among the PEs. In parallel the PEs perform 1D FFT(N) on those rows. A global transpose is performed. In parallel the PEs perform 1D FFT(N) on the columns. The output is uploaded by column from the PEs. Extensive interprocessor communications are not supported by this model; only host-PE communications are provided. The simplicity of the communications skeleton allows high speed communications at low cost. A drawback of this scheme is that $4N^2$ host-PE data moves are required. This quantity limits system performance and cannot be improved by increasing the number of

processors.

2.3.4 Hierarchical and Shared Memory Machines

Norton and Silberger [43] consider tightly coupled shared memory architectures for parallel 2D FFT computation. Their memory model is a three level hierarchy: private cache and local memory for each processor, and shared global memory for all the processors. They compare the performance of two algorithms that compute the 2D FFT on these architectures. Both algorithms compute the 1D FFT on the rows in parallel, then on the columns in parallel. The difference between the methods is which memory is used for computation. The first method computes with the rows and columns in place in shared memory. The second copies the row/column from global to local memory, transforms in local memory, then copies back. Both algorithms use the shared memory to realize a matrix transpose. Their comparison shows that if the global shared memory is even 10% slower than the local memory, the copy algorithm is faster. Since the copy algorithm uses the shared memory only as an IN, the shared memory must be within 10% of the speed of local memory or an IN would be a more efficient solution. Because of the difficulty involved in realizing shared memory, these machine will be considered equivalent to IN multiprocessors for the purpose of 2D FFT computation.

2.3.5 Ensemble Processors

Milutinovic [39] examines 1D DFT computation by direct matrix-vector multiplication on loosely coupled multiprocessors. They show that for problems where only a subset of output points are required this method is preferable to parallel CT FFT methods. The primary reason for this is the absence of interprocessor communication in direct DFT computation. This simplicity reduces both machine and design cost. However, the extension of this method to 2D $N \times N$ DFT requires N^2 processors each of which would have to compute or have stored N^2 constants. If L processors were available, the computation would require time for $\lceil N^2/L \rceil N^2$ multiply-accumulates if the PEs stored $\lceil N^2/L \rceil$ constant vectors of length N^2.

Bergland [9] introduces an ensemble processor algorithm for parallel FFT computation. His original algorithm maps the 1D DFT to a 2D DFT using the Good–Thomas algorithm [23], and therefore constrains the size of the dimensions to be relatively prime. However, this method may be applied to any two-dimensional DFT. The machine model can be taken as a multiprocessor together with an IN which supports broadcast and report communication functions. On an N PE machine the $N \times N$ 2D DFT can be computed as

$$F_{N,N}\underline{x} = (I_N \otimes F_N)(F_N \otimes I_N)\underline{x}$$

Where one of the FFTs of the second stage $(I_N \otimes F_N)$ is assigned to each of the PEs. Given this assignment, the i^{th} PE must perform the matrix operation

$$\left(I_N\, \omega^i I_N\, \omega^{2i} I_N \cdots \omega^{(N-1)i} I_N\right) \underline{x}.$$

on the input. Thus PE_i computes one complex multiply-accumulate operation for each point input to PE_i. This requires eight arithmetic operations (4 multiplies and 4 adds) for each complex point input to the machine. If a PE requires time a to complete one arithmetic operation, and the system requires time c to complete a single word of a communication operation, then if $a = c$ the algorithm requires the time for the data input and output, plus the time for $O(N^2)$ arithmetic operations, as well as a memory of size $O(N^2)$.

2.3.6 Special Purpose Machines and VLSI implementations

Special purpose architectures for MD FFT implementations also reflect the requirement that extensive interprocessor communications are required. In [15] a real-time $N \times N$ 2D DFT engine is proposed. The machine implements the row-column algorithm, and relies on a ram array transposer (RAMAT) to realize the transpose stage between the row and column FFTs. The RAMAT is a memory that is organized as a matrix and can be accessed by row or column address.

In [21] an AT^2 optimal circuit [62] is introduced for $N \times N$ 2D DFT computation. The circuit uses N processors each performing 1D DFT(N) and a rotator of area

$O(N^4)$ to perform the transpose. This circuit is generalized by [14] to permit greater area-time tradeoffs. Hornick [28] considers the $P \times P$ mesh of trees circuit which has P^4 processing elements for $N \times N$ 2D DFT computation. By applying the vector-radix algorithm [26] he attains a family of circuits that are AT^2 optimal.

2.3.7 Conclusion

The parallel algorithms for MD DFT computation considered in this section require either extensive interprocessor communications or excessive computation. The majority of algorithms that have been studied rely on the row-column or multidimensional Cooley-Tukey algorithms. These algorithms intermingle data exchange stages with computation stages. Because of this they require extensive interprocessor communications when implemented on distributed machines. The algorithms that have been studied that do not require interprocessor communication are direct matrix-vector multiplication implementations. These require excessive numbers of complex multiplications.

The remainder of this work analyzes and describes multiprocessor algorithms that use reduced transform methods to compute the DFT. These methods are shown to eliminate interprocessor communications at the cost of an increased number of additions. The additions can be performed concurrently with the data download.

Chapter 3

A Tensor Product Formulation for Multidimensional Cooley-Tukey Algorithms

The tensor product formulation of one dimensional Cooley-Tukey FFT algorithms was reviewed in chapter 2. This chapter presents a generalization of that formulation to multidimensional cases. The central feature of the generalization is that CT decimation permutes the indexing sets by coset decomposition; the decompositions induced by these algorithms are shown to depend on dual coset decompositions of the input and output index groups. The formulation presented here differs from other multidimensional tensor product formulations of the CT FFT in that it is intrinsically multidimensional. The required entities are defined in terms of the additive properties of the underlying indexing set and require no correction matrices.

As in the one dimensional case, the multidimensional Cooley-Tukey algorithm is built up from three basic stages. These are (1) stages of data movement (permutation), (2) stages of multiplication (twiddle factors), and (3) stages of lower order multidimensional DFT. In order to procede with the development of the formulation it is necessary to define the coset decompostion of a group [11]. Following this the basic entities of the formulation will be introduced.

Definition. Let $< G, + >$ be any finite group, and $< H, + >$ be any subgroup of $< G, + >$. The following steps define a constructive procedure for obtaining the coset decomposition of G with respect to H.

1. Create an array whose first row contains the elements of the subgroup H.

2. Select an element from G that has not appeared in any previous row and apply it on the left to the elements of H using the group operation to create the next row. Continue this step until all elements have been exhausted. Each row of the resulting array is a coset. The first element of each row is termed the *coset leader.*

Example: Consider the ring of integers Z/N. The coset decomposition of Z/N with respect to the subgroup sZ/N, where $N = rs$, is shown below. Then $Z/N = \{0, 1, \ldots, N-1\}$ and $sZ/N = \{0, s, \ldots, (r-1)s\}$. The coset decompostion of Z/N with respect to sZ/N is given by the array

$$\begin{array}{ccccc} 0 & s & 2s & \cdots & (r-1)s \\ 1 & s+1 & 2s+1 & \cdots & (r-1)s+1 \\ 2 & s+2 & 2s+2 & \cdots & (r-1)s+2 \\ \vdots & \vdots & \vdots & & \vdots \\ s-1 & 2s-1 & 3s-1 & \cdots & n-1 \end{array}$$

Observe that row 0 is the subgroup sZ/N, and that the row i is coset i, with coset leader i.

□

3.1 Coset Permutation

The coset permutation is the multidimensional analog of the stride permutation (section 2.1.1) for Cooley-Tukey type FFTs. The coset permutation is denoted $P_{\underline{n},\underline{s}}$, where $\underline{n} = (n_1, \ldots, n_d)$ and $\underline{s} = (s_1, \ldots, s_d)$. It is defined in terms of the coset decomposition of the underlying indexing set $Z/n_1 \times Z/n_2 \times \cdots \times Z/n_d$ with respect to the subgroup $s_1 Z/n_1 \times s_2 Z/n_2 \times \cdots \times s_d Z/n_d$. The following notation will be used throughout:

1. $Z/\underline{n}$ denotes $Z/n_1 \times Z/n_2 \times \cdots \times Z/n_d$.

2. $\underline{s}Z/\underline{n}$ denotes $s_1 Z/n_1 \times s_2 Z/n_2 \times \cdots \times s_d Z/n_d$.

If $\underline{x}$ is the vector formed by writting down the elements of the $d-$dimensional array $X_{\underline{n}}$ lexicographically by dimension, then the coset permutation matrix $P_{\underline{n},\underline{s}}$ is fully defined by

$$\underline{y} = P_{\underline{n},\underline{s}}\underline{x}, \qquad n_i = s_i r_i \tag{3.1}$$

where $\underline{y}$ is the vector formed by writting down the elements of $X_{\underline{n}}$ lexicographically by cosets of the decomposition $Z/\underline{n}$ with respect to $\underline{s}Z/\underline{n}$.

Consider the two dimensional array $X(n_1, n_2)$, where $n_1 = r_1 s_1$ and $n_2 = r_2 s_2$. The underlying indexing set is the ring $Z/n_1 \times Z/n_2$. The (i,j) coset of $Z/n_1 \times Z/n_2$ with respect to the subgroup $s_1 Z/n_1 \times s_2 Z/n_2$ is

$$\begin{pmatrix} i,j & i,s_2+j, & \cdots & i,n_2-s_2+j \\ s_1+i,j & s_1+i,s_2+j & \cdots & s_1+i,n_2-s_2+j \\ \vdots & \vdots & & \vdots \\ n_1-s_1+i,j & n_1-s_1+i,s_2+j & \cdots & n_1-s_1+i,n_2-s_2+j \end{pmatrix}. \tag{3.2}$$

The coset permutation of $\underline{x}$ is formed by reordering the elements of $\underline{x}$ lexicographically by coset, and is given by:

$$\underline{y} = \begin{pmatrix} x_{0,0} \\ x_{0,s_2} \\ \vdots \\ x_{0,n_2-s_2} \\ \vdots \\ x_{n_1-1,s_2-1} \\ x_{n_1-1,2s_2-1} \\ \vdots \\ x_{n_1-1,n_2-1} \end{pmatrix} = P_{(n_1,n_2)(s_1,s_2)} \begin{pmatrix} x_{0,0} \\ \vdots \\ x_{0,n_2-1} \\ x_{1,0} \\ \vdots \\ x_{1,n_2-1} \\ \vdots \\ x_{n_1-1,0} \\ \vdots \\ x_{n_1-1,n_2-1} \end{pmatrix} \tag{3.3}$$

The following example illustrates this idea.

Example: Consider the permutation $\underline{y} = P_{(4,4)(2,2)}\underline{x}$. The coset decomposition of $Z/4 \times Z/4$ with respect to $2Z/4 \times 2Z/4$ is given below. Each coset is shown as a matrix with its coset leader in parenthesis above it.

$$\begin{matrix} (0,0) & (0,1) \\ \begin{pmatrix} 0,0 & 0,2 \\ 2,0 & 2,2 \end{pmatrix} & \begin{pmatrix} 0,1 & 0,3 \\ 2,1 & 2,3 \end{pmatrix} \\ (1,0) & (1,1) \\ \begin{pmatrix} 1,0 & 1,2 \\ 3,0 & 3,2 \end{pmatrix} & \begin{pmatrix} 1,1 & 1,3 \\ 3,1 & 3,3 \end{pmatrix} \end{matrix}$$

The permuted vector formed by reading the cosets of indexes lexicographically is

$$\underline{y} = \begin{pmatrix} x_{0,0} \\ x_{0,2} \\ x_{2,0} \\ x_{2,2} \\ x_{0,1} \\ x_{0,3} \\ x_{2,1} \\ x_{2,3} \\ x_{1,0} \\ x_{1,2} \\ x_{3,0} \\ x_{3,2} \\ x_{1,1} \\ x_{1,3} \\ x_{3,1} \\ x_{3,3} \end{pmatrix} = P_{(4,4)(2,2)} \begin{pmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \\ x_{0,3} \\ x_{1,0} \\ x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{2,0} \\ x_{2,1} \\ x_{2,2} \\ x_{2,3} \\ x_{3,0} \\ x_{3,1} \\ x_{3,2} \\ x_{3,3} \end{pmatrix}. \tag{3.4}$$

□

In the one dimensional case, where $n = rs$, the underlying indexing set is the group Z/n and the subgroup used for the coset decomposition is sZ/n. Direct application of the definition of the coset permutation results in the stride permutation $P_{n,s}$.

Example: Consider the coset decomposition of Z/n with respect to sZ/n shown in the previous section. The permuted vector formed by reordering the elements of $\underline{x}$

lexicographically by coset is

$$\underline{y} = \begin{pmatrix} x_0 \\ x_s \\ \vdots \\ x_{(r-1)s} \\ \vdots \\ x_{s-1} \\ x_{2s-1} \\ \vdots \\ x_{n-1} \end{pmatrix} = P_{n,s} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} \tag{3.5}$$

which is exactly the stride by s permutation $P_{n,s}$ of $\underline{x}$.

□

The vector corresponding to the coset permutation of a $d-$dimensional array $X_{\underline{n}}$ can also be defined by the mapping

$$\Psi_1 : Z/n \longmapsto Z/\underline{n},$$

where $n = n_1 \cdots n_d$ and $n_k = s_k r_k$. Then every $j \in Z/n$ can be written uniquely as

$$\begin{aligned} j &= \frac{n}{s_1}b_1 + \frac{n}{s_1 s_2}b_2 + \cdots + \frac{n}{s_1 \cdots s_d}b_d \\ &\quad + \frac{r}{r_1}a_1 + \frac{r}{r_1 r_2}a_2 + \cdots + \frac{r}{r_1 \cdots r_d}a_d \\ &= \sum_{k=1}^{d} \left[b_k \frac{n}{\prod_{i=1}^{k} s_i} + a_k \frac{r}{\prod_{i=1}^{k} r_i} \right], \\ &\quad 0 \le a_k < r_k \quad \text{and} \quad 0 \le b_k < s_k \end{aligned} \tag{3.6}$$

and every $\alpha_k \in Z/n_k$ can be written uniquely as

$$\alpha_k = s_k a_k + b_k.$$

Let Ψ_1 be the mapping defined by

$$\begin{aligned} \Psi_1(j) &= \Psi_1 \left(\sum_{k=1}^{d} \left[b_k \frac{n}{\prod_{i=1}^{k} s_i} + a_k \frac{r}{\prod_{i=1}^{k} r_i} \right] \right) \\ &= (s_1 a_1 + b_1, s_2 a_2 + b_2, \ldots, s_d a_d + b_d). \end{aligned} \tag{3.7}$$

Then

$$y(j) = X(\Psi_1(j)), \qquad j = 0, 1, \ldots, n-1 \tag{3.8}$$

defines the sequence formed by reading the cosets of $X_{\underline{n}}$ lexicographically.

The coset permutation of a vector representing a multidimensional array can be specified by a permutation of Z/n. First, let Ψ_2 be the mapping $\Psi_2 : Z/\underline{n} \longmapsto Z/n$ which for all $\underline{\alpha}$ in $Z/\underline{n}$ is given by

$$\Psi_2(\underline{\alpha}) = \Psi_2(\alpha_1, \ldots, \alpha_d) = (j) = \sum_{k=1}^{d} \alpha_k \left(n / \prod_{i=1}^{k} n_i \right). \tag{3.9}$$

Then the $d-$dimensional array $X_{\underline{n}}$ can be represented by the sequence $\underline{x}$

$$X(\underline{\alpha}) = x\left(\Psi_2(\underline{\alpha})\right)$$

$$X(\alpha_1, \ldots, \alpha_d) = x\left(\sum_{k=1}^{d} \alpha_k \left(n / \prod_{i=1}^{k} n_i \right) \right) = x(j) \tag{3.10}$$

which is the vector formed by reading the elements of $X(n_1, \ldots, n_d)$ lexicographically by dimension.

Let Ψ be the mapping

$$\Psi : Z/n \longmapsto Z/n$$
$$\Psi = \Psi_2(\Psi_1)$$

which for all $j \in Z/n$ is given by

$$\Psi(j) = \Psi_2(\Psi_1(j)) \tag{3.11}$$

$$\Psi\left(\sum_{k=1}^{d} \left[b_k \frac{n}{\prod_{i=1}^{k} s_i} + a_k \frac{r}{\prod_{i=1}^{k} r_i} \right] \right) = \sum_{k=1}^{d} (s_k a_k + b_k) \left(\frac{n}{\prod_{i=1}^{k} n_i} \right)$$

then

$$y(j) = x\left(\Psi(j)\right), \qquad j = 0, 1, \ldots, n-1 \tag{3.12}$$

defines the coset permutation $P_{\underline{n},\underline{s}}$ of a vector $\underline{x}$.

The permutation matrix $P_{\underline{n},\underline{s}}$ can be defined by the permutation $\Psi : Z/n \longmapsto Z/n$ which takes $j \in Z/n$ to $\Psi(j) \in Z/n$. The matrix $P_{\underline{n},\underline{s}}$ is written

$$P_{\underline{n},\underline{x}} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-1} \\ \vdots & \vdots & & \vdots \\ p_{n-1,0} & p_{n-1,1} & \cdots & p_{n-1,n-1} \end{bmatrix} \tag{3.13}$$

where

$$p_{j,k} = \begin{cases} 1, & \mathrm{k} = \Psi(j) \\ 0, & else \end{cases} \qquad \text{where} \quad 0 \leq j,k < n-1$$

Since every $k \in Z/n$ can be written uniquely as

$$k = \sum_{l=1}^{d} \left(n / \prod_{i=1}^{l} n_i \right) (s_l a_l + b_l) \tag{3.14}$$

and similarly every $j \in Z/n$ can be written uniquely as

$$j = \sum_{l=1}^{d} \left[b_l \left(n / \prod_{i=1}^{l} s_i \right) + a_l \left(r / \prod_{i=1}^{l} r_i \right) \right]. \tag{3.15}$$

Then the inverse coset permutation of a vector representing a $d-$dimensional array can be defined by the permutation

$$\Psi^{-1} : Z/n \longmapsto Z/n$$

$$\Psi^{-1}(k) = j \tag{3.16}$$

$$\Psi^{-1} \left(\sum_{l=1}^{d} \left(n / \prod_{i=1}^{l} n_i \right) (s_l a_l + b_l) \right) = \sum_{l=1}^{d} \left[b_l \left(n / \prod_{i=1}^{l} s_i \right) + a_l \left(r / \prod_{i=1}^{l} r_i \right) \right]$$

and

$$z(k) = y(\Psi^{-1}(k)), \qquad k = 0, 1, \ldots, n-1 \tag{3.17}$$

defines the inverse coset permutation of a vector $\underline{y}$.

The inverse coset permutation matrix $P^{-1}_{\underline{n},\underline{s}}$ is given by

$$P^{-1}_{\underline{n},\underline{s}} = P^{t}_{\underline{n},\underline{s}}. \tag{3.18}$$

Note that unlike the inverse stride permutation where $P^{-1}_{n,s} = P_{n,r}$, in general the inverse coset permutation $P^{-1}_{\underline{n},\underline{s}} \neq P_{\underline{n},\underline{r}}$.

3.2 Multidimensional Twiddle Factors

The diagonal matrix $D_{n,s}$ of order s is defined by

$$D_{n,s} = diag[1, \omega_n, \ldots, \omega_n^{s-1}] \tag{3.19}$$

The multidimensional twiddle factor matrix $T_{\underline{n},\underline{s}}$ is defined as the direct sum of the tensor product of diagonal matrices

$$T_{\underline{n},\underline{s}} = \sum_{j_1=0}^{s_1-1} \oplus \sum_{j_2=0}^{s_2-1} \oplus \cdots \sum_{j_d=0}^{s_d-1} \oplus (D_{n_1,r_1}^{j_1} \otimes D_{n_2,r_2}^{j_2} \otimes \cdots \otimes D_{n_d,r_d}^{j_d}) \tag{3.20}$$

Restricted to two dimensions this becomes

$$T_{(n_1,n_2),(s_1,s_2)} = \sum_{j_1=0}^{s_1-1} \oplus \sum_{j_2=0}^{s_2-1} \oplus (D_{n_1,r_1}^{j_1} \otimes D_{n_2,r_2}^{j_2}) \tag{3.21}$$

3.3 MD FFT Tensor Formulation

A tensor product formulation for Cooley-Tukey type multidimensional FFT algorithms is introduced in this section. The presentation begins with the development of the formulation for the two-dimensional case. Following this the reduction of the formulation to the one-dimensional case is given, and is shown to be identical to the one-dimensional formulation given in [51]. Finally, the general multidimensional tensor product fomulation is presented.

The main result of this section is that the $n_1 \times n_2$ 2D DFT can be evaluated by the procedure listed below, and that this interpretation of the evaluation leads to a tensor product of matrices formulation for CT type MD FFT algorithms.

1. Computation of $r_1 \times r_2$ 2D DFT. Each of these is performed on a coset of the input with repect to the subgroup $s_1 Z/n_1 \times s_2 Z/n_2$.

2. Twiddle multiplications. The (a_1, a_2) element of the (u_1, u_2) coset transformed is multiplied by $\omega_{n_1}^{a_1 u_1} \omega_{n_2}^{a_2 u_2}$.

3. Computation of $s_1 \times s_2$ 2D DFT. Each of these produces a coset of output with respect to the dual of the input subgroup. The output subgroup is $r_1 Z/n_1 \times r_2 Z/n_2$.

Consider an $n_1 \times n_2$ 2D DFT where $n_1 = r_1 s_1$ and $n_2 = r_2 s_2$. The Cooley-Tukey decomposition of this transform is given by equation (2.24). Letting

$$X_1(u_1, u_2, p_1, p_2) = x(s_1 p_1 + u_1, s_2 p_2 + u_2) \tag{3.22}$$

and using (3.22), the innermost summation terms of (2.24) can be written

$$Y_1(u_1, u_2, a_1, a_2) = \sum_{p_1=0}^{r_1-1} \sum_{p_2=0}^{r_2-1} X_1(u_1, u_2, a_1, a_2) \omega_{r_1}^{a_1 p_1} \omega_{r_2}^{a_2 p_2}. \tag{3.23}$$

To write equation (3.23) as a matrix-vector product let $\underline{x}_1$ and $\underline{y}_1$ denote the vectors formed by reading the elements of arrays X_1 and Y_1 lexicographically by dimension. Then equation (3.23) can be represented by

$$\underline{y}_1 = \begin{pmatrix} F_{r_1,r_2} & & & \\ & F_{r_1,r_2} & & \\ & & \ddots & \\ & & & F_{r_1,r_2} \end{pmatrix} \underline{x}_1. \tag{3.24}$$

Refering back to the definition of the tensor product of matrices given by (2.2), equation (3.24) may be written

$$\underline{y}_1 = (I_{s_1 s_2} \otimes F_{r_1,r_2}) \underline{x}_1. \tag{3.25}$$

Each segment of $r_1 r_2$ points of the vector $\underline{x}_1$ is a coset of $Z/n_1 \times Z/n_2$ with respect to $s_1 Z/n_1 \times s_2 Z/n_2$. By equations (3.11) and (3.12) $\underline{x}_1$ is seen to be given simply by the coset permutation of $\underline{x}$

$$\underline{x}_1 = P_{(n_1,n_2),(s_1,s_2)} \underline{x} \tag{3.26}$$

where $\underline{x}$ is the vector formed by reading the elements of the input array X lexicographically by dimension. Substituting (3.26) into (3.25) gives the first stage of the computation as

$$\underline{y}_1 = (I_{s_1 s_2} \otimes F_{r_1,r_2}) P_{(n_1,n_2),(s_1,s_2)} \underline{x}. \tag{3.27}$$

The next stage of the computation of (2.24) is the evaluation of the twiddle multiplications. These are given by

$$Y_2(u_1, u_2, a_1, a_2) = \omega_{N_1}^{a_1 u_1} \omega_{N_2}^{a_2 u_2} Y_1(u_1, u_2, a_1, a_2). \tag{3.28}$$

Letting $\underline{y}_2$ be the vector representing array Y_2, a matrix representation of equation (3.28) is given by

(3.29)

$$\underline{y}_2 = \begin{pmatrix} (I_{r_1} \otimes I_{r_2}) & & & & & & \\ & \ddots & & & & & \\ & & (I_{r_1} \otimes D_{n_2,r_2}^{s_2-1}) & & & & \\ & & & (D_{n_1,r_1} \otimes I_{r_2}) & & & \\ & & & & \ddots & & \\ & & & & & (D_{n_1,r_1} \otimes D_{n_2,r_2}^{s_2-1}) & \\ & & & & & & \ddots \\ & & & & & & (D_{n_1,r_1} \otimes D_{n_2,r_2}^{s_2-1}) \end{pmatrix} \underline{y}_1.$$

The diagonal matrix in this expression is the twiddle factor matrix defined by equation (3.21). This allows (3.29) to be rewritten as

$$\underline{y}_2 = T_{(n_1,n_2)(s_1,s_2)}\underline{y}_1. \tag{3.30}$$

Letting

$$Y'(b_1, b_2, a_1, a_2) = Y(a_1 + r_1 b_1, a_2 + r_2 b_2) \tag{3.31}$$

the last two summations of equation (2.24) are written

$$Y'(b_1, b_2, a_1, a_2) = \sum_{u_1=0}^{s_1-1} \sum_{u_2=0}^{s_2-1} \omega_{s_1}^{u_1 b_1} \omega_{s_2}^{u_2 b_2} Y_2(u_1, u_2, a_1, a_2). \tag{3.32}$$

If $\underline{y}'$ and $\underline{y}_2$ denote the vectors representing the arrays Y' and Y_2, equation (3.32) can be written in matrix form as

$$\underline{y}' = (F_{s_1 s_2} \otimes I_{r_1 r_2})\underline{y}_2. \tag{3.33}$$

Observe in (3.31) and (3.32) that the output of each $F_{s_1 s_2}$ is the (a_1, a_2) coset of the output array Y with respect to the subgroup $r_1 Z/n_1 \times r_2 Z/n_2$. The vector $\underline{y}'$

of (3.33) is seen to contain the elements of each output coset seperated by $r = r_1 r_2$ points. So the vector

$$\underline{y}'' = P_{n,r}\underline{y}' \tag{3.34}$$

must contain contiguous cosets of the output. By definition this vector is the coset permutation of the output vector, and may be written

$$\underline{y}'' = P_{\underline{n},\underline{r}}\underline{y}, \tag{3.35}$$

where $\underline{y}$ is the vector formed by reading the output array Y lexicographically by dimension. Conversely, $\underline{y}$ is given by

$$\underline{y} = P^{-1}_{(n_1,n_2),(r_1,r_2)}\underline{y}''. \tag{3.36}$$

When these results are combined the tensor product formulation is seen to be

$$\underline{y} = Q_{(n_1,n_2)(r_1,r_2)}(F_{s_1,s_2} \otimes I_{r_1 r_2})T_{(n_1 n_2)(s_1,s_2)}(I_{s_1 s_2} \otimes F_{r_1,r_2})P_{(n_1,n_2)(s_1,s_2)}\underline{x} \tag{3.37}$$

$$Q_{(n_1,n_2)(r_1,r_2)} = P^{-1}_{(n_1,n_2)(r_1,r_2)}P_{n,r_1 r_2}.$$

The formulation reduces to

$$\underline{y} = (F_s \otimes I_r)T_{n,s}(I_s \otimes F_r)P_{n,s}\underline{x} \tag{3.38}$$

in the one dimensional case since

$$Q_{n,r} = P^{-1}_{n,r}P_{n,r} = I_n.$$

The formulation of (3.37) extends to general multidimensional cases naturally for DFT which have a composite number of points in each dimension. In general

$$F_{\underline{n}} = (F_{n_1} \otimes \cdots \otimes F_{n_d}) \tag{3.39}$$

where

$$\begin{array}{lll} \underline{n} = (n_1, n_2, \ldots, n_d) & n = n_1 \cdots n_d & \\ \underline{r} = (r_1, r_2, \ldots, r_d) & r = r_1 \cdots r_d & n_i = r_i s_i \\ \underline{s} = (s_1, s_2, \ldots, s_d) & s = s_1 \cdots s_d & i = 1, 2, \ldots d \end{array} \tag{3.40}$$

can be formulated as

$$F_{\underline{n}} = Q_{\underline{n},\underline{r}}(F_{\underline{s}} \otimes I_r)T_{\underline{n},\underline{s}}(I_s \otimes F_{\underline{r}})P_{\underline{n},\underline{s}} \tag{3.41}$$

$$Q_{\underline{n},\underline{r}} = P^{-1}_{\underline{n},\underline{r}} P_{n,r}.$$

If $F_{\underline{N}}$ denotes the $d-$dimensional DFT of $N-$points in each dimension, and $N = 2^n$, equation (3.41) can be applied iteratively to obtain the radix-$(2 \times \cdots \times 2)$ MD factorization. This is given by

$$\begin{aligned} F_{\underline{2}^n} &= Q_{\underline{2}^n,\underline{2}^{n-1}}(F_{\underline{2}} \otimes I_{(2^{n-1})^d})T_{\underline{2}^n,\underline{2}} \qquad (3.42)\\ &\quad \cdots (I_{(2_{n-k})^d} \otimes Q_{\underline{2}^k,\underline{2}^{k-1}})(I_{(2_{n-k})^d} \otimes F_{\underline{2}} \otimes I_{(2^{k-1})^d})(I_{(2_{n-k})^d} \otimes T_{\underline{2}^k,\underline{2}})\\ &\quad \cdots (I_{(2^{n-1})^d} \otimes F_{\underline{2}})Q\\ &= \left(\prod_{k=1}^{n}(I_{(2_{n-k})^d} \otimes Q_{\underline{2}^k,\underline{2}^{k-1}})(I_{(2_{n-k})^d} \otimes F_{\underline{2}} \otimes I_{(2^{k-1})^d})(I_{(2_{n-k})^d} \otimes T_{\underline{2}^k,\underline{2}})\right) Q. \end{aligned}$$

$$\begin{aligned} Q &= (I_{(2^{n-2})^d} \otimes P_{\underline{2}^2,\underline{2}}) \cdots (I_{(2^k)^d} \otimes P_{\underline{2}^{n-k},\underline{2}}) \cdots P_{\underline{2}^n,\underline{2}}\\ &= \prod_{k=1}^{n-2}(I_{(2^k)^d} \otimes P_{\underline{2}^{n-k},\underline{2}}) \end{aligned}$$

where $\underline{\alpha}$ denotes the vector $\underline{\alpha} = (\alpha, \ldots, \alpha)$.

In the introduction of this chapter, reference was made to the dual nature of the input and output decompositions. These decompositions are duals in the sense that they are coset permutations defined by dual subgroups of the $d-$dimensional index set $Z/\underline{n}$. To make this explicit use the inner product defined as

$$\underline{\alpha} \cdot \underline{\beta} = \alpha_1\beta_1 + \cdots + \alpha_d\beta_d, \qquad \underline{\alpha}, \underline{\beta} \in Z/\underline{n}. \tag{3.43}$$

The subgroup defining the input partition was shown to be $\underline{s}Z/\underline{n}$ for $n_i = s_i r_i$, and

$$\underline{\alpha} = (s_1\alpha_1, \ldots, s_d\alpha_d), \quad \underline{\alpha} \in \underline{s}Z/\underline{n}. \tag{3.44}$$

The output is produced on the partition defined by the subgroup $\underline{r}Z/\underline{n}$, where

$$\underline{\beta} = (r_1\beta_1, \ldots, r_d\beta_d), \quad \underline{\beta} \in \underline{r}Z/\underline{n}. \tag{3.45}$$

Then, with respect to the definition of (3.43), the inner product of all

$$\underline{\alpha} \in \underline{s}Z/\underline{n}$$

and

$$\underline{\beta} \in \underline{r}Z/\underline{n}$$

is

$$\underline{\alpha} \cdot \underline{\beta} = \alpha_1\beta_1 s_1 r_1 + \cdots + \alpha_d\beta_d s_d r_d = 0, \tag{3.46}$$

establishing the duality.

3.4 Conclusion

This chapter has presented a new tensor product formulation for multidimensional Cooley-Tukey type FFT algorithms. The factorizations of the DFT induced by these algorithms are shown to depend on dual coset decompositions of the input and output index groups.

This fact is used to define the entities of the tensor product formulation and results in a natural representation of the algorithms. Once a decomposition of the DFT into stages of lower ordered DFTs is specified using the tensor product, it may be manipulated using matrix algebra to achieve the design goals of an application.

Chapter 4

A General MD DFT Reduction Algorithm

This chapter contributes a new reduced transform algorithm (RTA) for the computation of the MD DFT. The algorithm computes a $d-$dimensional DFT by a set of independent $k-$dimensional DFT; it is a reduction algorithm in the sense that it has lowered the dimension of the Fourier transforms being computed. The $k-$dimensional DFT are performed on data derived from the input data using only additions, and produce $k-$dimensional hyperplanes of output.

The algorithm is derived by restricting the $d-$dimensional DFT to a collection of subgroups of its output indexing set. In order to describe these subgroups in a natural manner, the notion of the discrete $k-$dimensional hyperplane is employed. In terms of these hyperplanes the development of the algorithm is undertaken in two parts. The first part defines a minimal set of $k-$dimensional hyperplanes that cover the $d-$dimensional array. The second part restricts the $d-$dimensional DFT to each of the $k-$dimensional hyperplanes of the covering set. The restrictions are shown to evaluate as independent $k-$dimensional DFT.

The algorithm is computationaly similar to the reduced transform algorithms of Auslander *etal.* [5], Nussbaumer and Quandalle [45,44], and Gertner and Tolimieri [20,22]. Like those algorithms it computes the multidimensional DFT by a preaddition stage followed by a stage of independent DFT. The algorithm introduced in this chapter computes the MD DFT on $k-$dimensional hyperplanes of the output array. It is a generalization and extension of the line algorithm of [20,22].

The motivation for this algorithm is the generation of a parallel algorithm for MD DFT computation that scales to the degree of parallelism of the target architecture. In chapter 5 a new parallel algorithm based on the algorithm presented in this chapter is introduced. Target architectures for that algorithm include broadcast mode multiprocessors. On such machines, the granularity of the computation is a $k-$dimensional DFT, and the degree of parallelism of the algorithm is the number of $k-$dimensional hyperplanes required to complete a covering set.

The remainder of this chapter is organized as follows: First, the definitions pertinent to the generation of $k-$dimensional hyperplanes are stated. Then, a set of covering $k-$dimensional hyperplanes is derived for the cases: (1) where the array is of equal and prime size in at least $d-k+1$ dimensions, and (2) where the array is of equal and power of prime (including 2) size in at least $d-k+1$ dimensions. For each case, the restriction of the $d-$dimensional DFT to the covering set of $k-$dimensional hyperplanes is given. Both cases are shown to reduce the computation to a set of independent DFT performed on data derived from the input data using only additions. An appendix is provided that enumerates the 3D and 4D cases of the algorithm.

4.1 Definitions

This section presents the definitions pertinent to the derivation of the algorithm. In essence, the algorithm restricts the computation of the $d-$dimensional DFT to a collection of subgroups of the output array. These subgroups are defined on the the underlying indexing set of the $d-$dimensional DFT.

In general, the index set of the $N_1 \times \cdots \times N_d$ $d-$dimensional DFT is associated with the ring

$$\mathcal{R} = Z/N_1 \times \cdots \times Z/N_d,$$

where Z/N_i is the ring of integers modulo N_i. This set defines the region of support of the function [17]. A point in the index set is taken to be the vector $\underline{u} \in \mathcal{R}$ defined by the $d-$tuple

$$\underline{u} = (u_1, \ldots, u_d), \quad u_i \in Z/N_i.$$

The value of the DFT restricted to a point $\underline{u}$ in the output array is evaluated in the natural way as

$$V(u_1,\ldots,u_d) = \sum_{a_1=0}^{N_1-1} \cdots \sum_{a_d=0}^{N_d-1} x(a_1,\ldots,a_d)\omega_{N_1}^{a_1u_1}\cdots\omega_{N_d}^{a_du_d}$$

for a fixed $\underline{u}$.

In section 2.2.1 the notion of a discrete line in a two-dimensional array was given. Recall that the line through a point $\underline{a} \in Z/N \times Z/N$ was defined by the subgroup

$$L(\underline{a}) = \{s\underline{a} = (sa_1, sa_2) : s \in Z/N\}. \tag{4.1}$$

The line was said to be of full order if $|L(\underline{a})| = N$.

This definition of a line extends naturally from the case where the line lies in a two-dimensional grid to the case where the line lies in a $d-$dimensional array. The line through a point $\underline{a} \in (Z/N)^d$ [22] is defined by the subgroup

$$L(\underline{a}) = \{s\underline{a} = (sa_1,\ldots,sa_d) : s \in Z/N\}. \tag{4.2}$$

By definition the line $L(\underline{a})$ is additively generated by $\underline{a}$; $L((\underline{a}))$ contains all linear combinations of the point $\underline{a} \in (Z/N)^d$. We say that $L(\underline{a})$ defines the additive closure of $\underline{a} \in (Z/N)^d$.

In a manner similar to the way $L(\underline{a})$ defines the closure of a point $\underline{a} \in (Z/N)^d$, we may define the closure of a set of points $\underline{a}_1,\ldots,\underline{a}_k \in (Z/N)^d$. The closure of $\underline{a}_1,\ldots,\underline{a}_k \in (Z/N)^d$ is given by the subgroup

$$H(\underline{a}_1,\ldots,\underline{a}_k) = \{s_1\underline{a}_1 + \cdots + s_k\underline{a}_k : s_i \in Z/N\}. \tag{4.3}$$

$H(\underline{a}_1,\ldots,\underline{a}_k)$ contains all linear combinations of the points in the set $\{\underline{a}_1,\ldots,\underline{a}_k\}$.

The idea of dimension can be associated with the subgroups generated in this way. The dimension of a subgroup of $(Z/N)^d$ is taken to be the number of linearly independent points $a_i \in (Z/N)^d$ required to generate it by the definition of (4.3). In this manner, subgroups generated by the additive closure of 1, 2, and 3 linearly independent points will be called lines, planes, and cubes respectively. Similarly, a subgroup whose dimension is $d-1$ is called a hyperplane. If the dimension of a subgroup is $k < d$ it is called a $k-$dimensional hyperplane.

The requirement of (4.3) that the array be of equal size in each dimension is unnecessarily restrictive for the development of the algorithm. It will be seen that it is only necessary for the array to be of equal size in $d-k+1$ dimensions.

Consider a $d-$dimensional array whose region of support is given by $\mathcal{R} = Z/N_1 \times \cdots \times Z/N_d$. Assume that $d-k+1$ dimensions are of equal size, and for simplicity of presentation allow those dimensions to be contiguous. Write $N_k = \cdots = N_d$. The definition adopted for the presentation of the algorithm uses the subgroup $H(\underline{a})$ formed by the closure of the vector $\underline{a} \in R$ with the $k-1$ standard basis vectors

$$\delta_i(j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

The subgroup

$$H(\underline{a}) = \{s_1\underline{\delta}_1 + \cdots + s_{k-1}\underline{\delta}_{k-1} + s_k\underline{a}_k : s_i \in Z/N_i\} \tag{4.5}$$

will define the $k-$dimensional discrete hyperplane. For the purpose of the development of the algorithm we will only be concerned with $k-$dimensional hyperplanes of full order. That is $(N_1 \cdots N_k)$ points. As an example, consider the 4D array $(Z/N)^4$. The hyperplane generated by the point $\underline{a} = (0,0,1,1)$ is given by

$$H((0,0,1,1)) = \{(s_1, s_2, s_3, s_3) : s_i \in Z/N\},$$

and $H((0,0,1,1))$ contains N^3 points. Similarly, the hyperplane generated by the point $\underline{a} = (0,0,x,y)$ is given by

$$H((0,0,x,y)) = \{(s_1, s_2, s_3x, s_3y) : s_i \in Z/N\}.$$

A set of $k-$dimensional hyperplanes is said to *cover* the $d-$dimensional array $\mathcal{R}$, if every point in $\mathcal{R}$ is in at least one $k-$dimensional hyperplane of the set. A set of covering $k-$dimensional hyperplanes is *minimal* if there is no smaller set of $k-$dimensional hyperplanes that also covers $\mathcal{R}$.

4.2 Prime Case

This section derives the prime case of the algorithm. For this case the $d-$dimensional DFT is defined over the region $\mathcal{R} = Z/N_1 \times \cdots \times Z/N_d$ where $\mathcal{R}$ is of equal

and prime size in $d-k+1$ dimensions. That is,

$$\begin{aligned} \mathcal{R} &= Z/N_1 \times \cdots \times Z/N_d = \mathcal{A} \times \mathcal{B}, \\ \mathcal{A} &= Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P, \\ \mathcal{B} &= (Z/P)^{d-k}, \text{ and } P \text{ is a prime.} \end{aligned} \tag{4.6}$$

The algorithm is derived in two parts. The first part is the specification of a minimal set of $k-$dimensional hyperplanes that cover $\mathcal{R}$. This part follows immediately below. The second part of the derivation requires the $d-$dimensional DFT to be restricted to each of the $k-$dimensional hyperplanes of the covering set. The resulting algorithm is shown to be computed by a set of independent $k-$dimensional DFTs. This case of the algorithm is specified by the procedure of equations (4.20) and (4.21).

Covering Hyperplanes

The following theorem gives a minimal set of $|\mathcal{A}|$ point $k-$dimensional hyperplanes that cover R. Let $\underline{\theta}(j)$ denote the vector of j zeros, ie. $\underline{\theta}(3) = (0,0,0)$.

Theorem 4.2.1 *The set of $k-$dimensional hyperplanes*

$$\begin{aligned} H_0 &= H((\underline{\theta}_{(k-1)}, 1, m_1, \ldots, m_{d-k})) \qquad m_i \in Z/P \\ H_1 &= H((\underline{\theta}_{(k-1)}, 0, 1, m_2, \ldots, m_{d-k})) \\ &\vdots \\ H_{d-k} &= H((\underline{\theta}_{(k-1)}, 0, 0, \ldots, 0, 1)) \end{aligned}$$

covers the $d-$dimensional array $\mathcal{R} = \mathcal{A} \times \mathcal{B}$, where $\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P$, $\mathcal{B} = (Z/P)^{d-k}$, and P is a prime. There are

$$\frac{P^{d-k+1}-1}{P-1}$$

$k-$dimensional hyperplanes in the set.

Proof. Every $\underline{a} = (a_1, \ldots, a_d) \in \mathcal{R}$ can be written

$$\underline{a} = \underline{a}^1 + \cdots + \underline{a}^{k-1} + \underline{a}',$$

where $\underline{a}^i$ is the vector whose i^{th} component is a_i and is zero elsewhere, and $\underline{a}'$ is the vector $\underline{a}' = (\underline{\theta}_{(k-1)}, a_k, \ldots, a_d)$.

By definition of $H(\underline{a})$, $\underline{a}^i$ is in every hyperplane of the covering set for $i = 1, \ldots, k-1$. Then by the closure property of the hyperplanes, covering is shown if every $\underline{a}'$ is in at least one hyperplane of the covering set. The following conditions include all cases of $\underline{a}'$.

If $a_k \neq 0$ then a_k is invertible modulo P and $\underline{a}'$ may be rewritten

$$\underline{a}' = a_k(\underline{\theta}_{(k-1)}, 1, a_k^{-1}a_{k+1}, \ldots, a_k^{-1}a_d),$$

and $\underline{a}'$ is seen to be contained in the union of the hyperplanes of the set H_0 of the theorem. By closure, the point $\underline{a} = \underline{a}^1 + \cdots + \underline{a}^{k-1} + \underline{a}'$ must also be contained in the union of the hyperplanes of the set H_0. There are P^{d-k} $k-$dimensional hyperplanes in H_0.

Repetition of this step until all the remaining $d-k$ components of $\underline{a}'$ are exhausted completes the proof. The α^{th} set of hyperplanes, H_α, generated contains $P^{d-k-\alpha}$ $k-$dimensional hyperplanes. In all there are

$$\frac{P^{d-k+1}-1}{P-1}$$

$k-$dimensional hyperplanes in the covering set.

□

The $k-$dimensional hyperplanes of the covering set of theorem 4.2.1 contain redundant points. The number of points in $\mathcal{R}$ is

$$|\mathcal{R}| = N_1 \cdots N_{k-1} \cdot P^{d-k+1}.$$

The number of points in each $k-$dimensional hyperplane is

$$|\mathcal{A}| = N_1 \cdots N_{k-1} \cdot P,$$

and the number of redundant points is

$$|\mathcal{A}| \cdot \left(\frac{P^{d-k}-1}{P-1}\right).$$

The locations of the redundant points are given by the $(k-1)$ dimensional hyperplane

$$H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1}) = \{(s_1, \ldots, s_{k-1}, \underline{\theta}(d-k+1)) : s_i \in Z/N_i\} \tag{4.7}$$

where $H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1})$ is defined by equation (4.3). The elements of (4.7) are common to every hyperplane of theorem 4.2.1, and are redundant

$$P \cdot \left(\frac{P^{d-k} - 1}{P - 1} \right)$$

times, accounting for all redundant points.

DFT Restriction

The $d-$dimensional DFT can be computed by restricting it to the $k-$dimensional hyperplanes of theorem 4.2.1. Below, the restriction of the MD DFT to those hyperplanes is derived.

The $d-$dimensional DFT over a region $\mathcal{R}$ is given by

$$V(\underline{u}) = \sum_{\underline{a} \in \mathcal{R}} x(\underline{a}) \omega_{N_1}^{a_1 u_1} \cdots \omega_{N_d}^{a_d u_d}, \qquad \underline{u} \in \mathcal{R}. \tag{4.8}$$

The prime case of the algorithm requires $d-k+1$ dimensions to be of equal and prime size. For simplicity of presentation these are assumed to be contiguous, and write

$$\begin{aligned} &\mathcal{R} = \mathcal{A} \times \mathcal{B}, \\ &\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times P, \\ &\mathcal{B} = (Z/P)^{d-k}, \text{ and } P \text{ is a prime.} \end{aligned}$$

The P^{d-k} $k-$dimensional hyperplanes of H_0 given by theorem 4.2.1 can be defined by the homomorphism

$$\Phi_0 : \mathcal{A} \longmapsto \mathcal{R}. \tag{4.9}$$

For all $\underline{s} \in \mathcal{A}$, Φ_0 is given by

$$\Phi_0(\underline{s}) = (s_1, \ldots, s_{k-1}, s_k, s_k m_1, \ldots, s_k m_{d-k}). \tag{4.10}$$

The restriction of the $d-$dimensional DFT (4.8) to the $k-$dimensional hyperplanes of H_0 is evaluated by replacing $\underline{u}$ with $\Phi_0(\underline{s})$ of (4.10) to obtain

$$V(\Phi_0(\underline{s})) = \sum_{\underline{a} \in \mathcal{R}} x(\underline{a}) \omega_{N_1}^{s_1 a_1} \cdots \omega_{N_d}^{s_k m_{d-k} a_d}, \tag{4.11}$$

$$\underline{s} \in \mathcal{A}.$$

The inner product

$$\Phi_0(\underline{s}) \cdot \underline{a} = s_1 a_1 + \cdots + s_{k-1} a_{k-1} + s_k(a_k + a_{k+1} m_1 + \cdots + a_d m_{d-k}) \tag{4.12}$$

implies

$$\Phi_0(\underline{s}) \cdot \underline{a} = s_1 d_1 + \cdots + s_k d_k = \underline{s} \cdot \underline{d} \tag{4.13}$$

$$\underline{d} \in \mathcal{A}.$$

In (4.13) the identifications

$$\begin{aligned} d_j &= a_j, \qquad j = 1, \ldots, k-1 \\ i_{j-k} &= a_j, \qquad j = k+1, \ldots, d \\ d_k &= a_k + m_1 a_{k+1} + \cdots + m_{d-k} a_d \end{aligned} \tag{4.14}$$

allow equation (4.11), the $d-$dimensional DFT restricted to the $k-$dimensional hyperplanes of H_0, to be written

$$V(\Phi_0(\underline{s})) = \sum_{\underline{d} \in \mathcal{A}} \sum_{\underline{i} \in \mathcal{B}} x(\underline{d}_{(k-1)}, d_k - \underline{m} \cdot \underline{i}, \underline{i}) \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \tag{4.15}$$

$$\underline{d} \in \mathcal{A}.$$

Where

$$a_k = d_k - \underline{m} \cdot \underline{i}, \tag{4.16}$$

and

$$\underline{d}_{(k-c)} = (d_1, \ldots, d_{k-c}). \tag{4.17}$$

Equation (4.15) can be rewritten as a reduction stage, followed by a DFT stage. The reduction operation is given by

$$a_{\underline{d}}^{H_0} = \sum_{\underline{i} \in \mathcal{B}} x\left(\underline{d}_{(k-1)}, d_k - \underline{m} \cdot \underline{i}, \underline{i}\right), \quad \underline{d} \in \mathcal{A} \tag{4.18}$$

and the DFT stage is given by

$$V(\Phi_0(\underline{s})) = \sum_{\underline{d} \in \mathcal{A}} a_{\underline{d}}^{H_0} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \tag{4.19}$$

$$\underline{s} \in \mathcal{A}, \quad \underline{m} \in \mathcal{B}.$$

Equations (4.18) and (4.19) give the values of the $d-$dimensional DFT on the $k-$dimensional hyperplanes of H_0. There is one such hyperplane for every $\underline{m} \in \mathcal{B}$.

In a similar manner, the restriction of the $d-$dimensional DFT to each of the remaining $k-$dimensional hyperplanes of the covering set of theorem 4.2.1 is computed by a reduction operation $a_{\underline{d}}^{H_\alpha}$, followed by a $k-$dimensional DFT of those points.

The complete algorithm is stated below for the case where the region of support, $\mathcal{R}$, is of prime size in at least $d-k+1$ dimensions. Then $\mathcal{R}$ is given as

$$\begin{aligned}
&\mathcal{R} = \mathcal{A} \times \mathcal{B}, \\
&\mathcal{A} = N_1 \times \cdots \times N_{k-1} \times P, \\
&\mathcal{B} = (Z/P)^{d-k}, \text{ and } P \text{ is a prime.}
\end{aligned}$$

For this case, the algorithm is specified by the following two step procedure:

Step 1 Reduction stage. This stage requires the evaluation of the summations

$$\begin{aligned}
a_{\underline{d}}^{H_0} &= \sum_{\underline{i} \in \mathcal{B}} x(\underline{d}_{(k-1)},\ d_k - \sum_{l=1}^{d-k} m_l i_l,\ \underline{i}), \qquad (4.20) \\
&\vdots \\
a_{\underline{d}}^{H_j} &= \sum_{\underline{i} \in \mathcal{B}} x(\underline{d}_{(k-1)},\ \underline{i}_{(j)},\ d_k - \sum_{l=j+1}^{d-k} m_l i_l,\ i_{j+1}, \ldots, i_{d-k}), \\
&\vdots \\
a_{\underline{d}}^{H_{d-k}} &= \sum_{\underline{i} \in \mathcal{B}} x(\underline{d}_{(k-1)},\ \underline{i},\ d_k)
\end{aligned}$$

$$\underline{d} \in \mathcal{A}, \quad m_i \in Z/N_i$$

Step 2 DFT stage. This stage requires the evaluation of the

$k-$dimensional DFTs

$$V(\Phi_0(\underline{s})) = \sum_{\underline{d} \in \mathcal{A}} a_{\underline{d}}^{H_0} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \qquad (4.21)$$

$$\vdots$$

$$
\begin{aligned}
V(\Phi_j(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{H_j}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \\
&\vdots \\
V(\Phi_{d-k}(\underline{s})) &= \sum_{\underline{d}\in\mathcal{A}} a_{\underline{d}}^{H_{d-k}}\omega_{N_1}^{s_1 d_1}\cdots\omega_{N_k}^{s_k d_k}, \\
&\underline{s}\in\mathcal{A}.
\end{aligned}
$$

In general Φ_j is given by

$$
\begin{gathered}
\Phi_j(\underline{s}) = (s_1,\ldots,s_{k-1},\underline{\theta}(j),s_k,s_k m_{j+1},\ldots,s_k m_{d-k}), \\
m_i \in Z/N_i.
\end{gathered}
$$

The $d-$dimensional DFT over $\mathcal{R} = \mathcal{A}\times\mathcal{B}$ is seen to be computed by

$$
\frac{P^{d-k+1}-1}{P-1}
$$

independent $k-$dimensional DFT over $\mathcal{A}$. The $k-$dimensional DFT are evaluated on data derived from the input data using only additions.

4.3 Power of Prime Case

This section derives the power of prime case of the algorithm. Like the prime case, the power of prime case is developed in two parts. The first part is the specification of a covering set of $k-$dimensional hyperplanes. The second part is the restriction of the $d-$dimensional to each of the hyperplanes of the covering set. The derivation of the algorithm follows below. The algorithm is stated by the procedure of equations (4.35) and (4.36).

Covering Hyperplanes

Consider the $d-$dimensional DFT defined over the region

$$
\mathcal{R} = Z/N_1\times\cdots\times Z/N_d
$$

where $\mathcal{R}$ is of equal and power of prime size in at least $d-k+1$ dimensions. For ease of presentation these dimensions are taken to be contiguous. That is,

$$\begin{aligned} &\mathcal{R} = \mathcal{A} \times \mathcal{B}, \\ &\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P^n, \\ &\mathcal{B} = (Z/P^n)^{d-k}, \text{ and } P \text{ is any prime including 2.} \end{aligned} \tag{4.22}$$

The following theorem gives a minimal set of $|\mathcal{A}|$ point $k-$dimensional hyperplanes that cover $\mathcal{R}$. Let $\underline{\theta}_{(j)}$ denote the vector of j zeros, ie. $\underline{\theta}_{(3)} = (0,0,0)$.

Theorem 4.3.1 *The set of $k-$dimensional hyperplanes*

$$\begin{aligned} H_0 &= H((\underline{\theta}_{(k-1)}, 1, m_1, \ldots, m_{d-k})), && m_i \in Z/P^n \\ H_1 &= H((\underline{\theta}_{(k-1)}, r_1 P, 1, m_2, \ldots, m_{d-k})), && r_i \in Z/P^{n-1} \\ &\vdots \\ H_{d-k} &= H((\underline{\theta}_{(k-1)}, Pr_1, Pr_2, \ldots, Pr_{d-k}, 1)) \end{aligned}$$

covers the $d-$dimensional array $\mathcal{R} = \mathcal{A} \times \mathcal{B}$, where $\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P^n$, $\mathcal{B} = (Z/P^n)^{d-k}$, and P is any prime including 2. There are

$$\left(\frac{P^n}{P}\right)^{d-k} \cdot \left(\frac{P^{d-k+1}-1}{P-1}\right)$$

$k-$dimensional hyperplanes in the set.

Proof. Every $\underline{a} = (a_1, \ldots, a_d) \in \mathcal{R}$ can be written

$$\underline{a} = \underline{a}^1 + \cdots + \underline{a}^{k-1} + \underline{a}',$$

where $\underline{a}^i$ is the vector whose i^{th} component is a_i and is zero elsewhere, and $\underline{a}'$ is the vector $\underline{a}' = (\underline{\theta}_{(k-1)}, a_k, \ldots, a_d)$.

By definition of $H(\underline{a})$, $\underline{a}^i$ is in every hyperplane of the covering set for $i = 1, \ldots, k-1$. Then by the closure property of hyperplanes, covering is shown if every $\underline{a}'$ is in at least one hyperplane of the set.

For any $\underline{a}'$ let $\alpha_k, \ldots, \alpha_d$ be maximums such that $P^{\alpha_i} | a_i$, $i = k, \ldots, d$. Then every point $\underline{a}'$ is described by the following for some j:

$$\begin{aligned} &\alpha_i < \alpha_j, \quad i = k, \ldots, j-1 \\ &\alpha_i \leq \alpha_j, \quad i = j+1, \ldots, d \end{aligned}$$

and $a_j \neq 0$ then a_j/P^{α_j} is invertible modulo P^n and $\underline{a}'$ may be written

$$\underline{a}' = a_j \cdot \left(\underline{\theta}_{(k-1)}, \left(\frac{a_k}{P^{\alpha_j}}\right)\left(\frac{a_j}{P^{\alpha_j}}\right)^{-1}, \ldots, 1, \ldots, \left(\frac{a_d}{P^{\alpha_j}}\right)\left(\frac{a_j}{P^{\alpha_j}}\right)^{-1} \right).$$

Such $\underline{a}'$ are contained in the union of the hyperplanes of the set H_j given in the theorem. Then by closure $\underline{a}$ is also in the union of the hyperplanes of the set H_j. Furthermore, since $\alpha_i < \alpha_j$, $i = k, \ldots, j-1$ we may write

$$\left(\frac{a_k}{P^{\alpha_j}}\right)\left(\frac{a_j}{P^{\alpha_j}}\right)^{-1} = P\left(\frac{a_k}{P^{\alpha_j+1}}\right)\left(\frac{a_j}{P^{\alpha_j}}\right)^{-1} = P \cdot r.$$

Then $\underline{a}'$ may be written

$$\underline{a}' = a_j(\underline{\theta}_{(k-1)}, r_1 P, \ldots, r_{j-1}P, 1, m_{j+1}, \ldots, m_{d-k}).$$

There are $(P^n)^{d-k}/P^j$ $k-$dimensional hyperplanes in the set H_j. Application of this procedure for each component $j = k, \ldots, d$ of $\underline{a}'$ completes the proof. In all there are

$$\left(\frac{P^n}{P}\right)^{d-k}\left(\frac{P^{d-k+1}-1}{P-1}\right)$$

$k-$dimensional hyperplanes in the covering set.

□

The covering set of theorem 4.3.1 contains redundant points. Altogether there are

$$|\mathcal{A}|\left(\frac{P^n}{P}\right)^{d-k}\left(\frac{P^{d-k}-1}{P-1}\right)$$

such points. The following theorems fully describe the redundancy between any two hyperplanes of the covering set of theorem 4.3.1. Throughout the sequel, the notation $\underline{x}_{\mathrm{mod}\, y}$ is taken to be

$$\underline{x}_{\mathrm{mod}\, y} = x_{1\,\mathrm{mod}\, y}, \ldots, x_{n\,\mathrm{mod}\, y}$$

where $\underline{x}$ is the vector $\underline{x} = x_1 \ldots, x_n$ and y is a scalar.

Theorem 4.3.2 *The redundancy between any two $k-$dimensional hyperplanes of the set H_0 is given by*

$$H((\underline{\theta}_{(k-1)}, 1, \underline{m})) \cap H((\underline{\theta}_{(k-1)}, 1, \underline{j})) =$$
$$H((\underline{\theta}_{(k-1)}, P^{n-\alpha}, (\underline{m}_{\mathrm{mod}\, P^\alpha})P^{n-\alpha})), \quad m_i, j_i \in Z/P^n.$$

Where α is the maximum such that

$$m_i \equiv j_i \bmod P^\alpha$$

holds for all $i = 1, \ldots, d-k$. The intersection contains $(N_1 \cdots N_{k-1})P^\alpha$ points.

Proof. By definition, the points on the hyperplanes

$$H((\underline{\theta}_{(k-1)}, 1, \underline{m})), \quad \text{and}$$
$$H((\underline{\theta}_{(k-1)}, 1, \underline{j}))$$

are equal only when $tm_i = tj_i$ modulo P^n for all $i = 1, \ldots d-k$. If $t = 0$ the points of the $(k-1)$ dimensional hyperplane $H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1}, \underline{\theta}_{(d-k+1)})$ are common. If $t \neq 0$ and $p \not| t$ then t is invertible modulo P^n and common points exist only when $m_i = j_i$ for all $i = 1, \ldots, d-k$. Otherwise there is some β such that t/P^β is invertible modulo P^n, and common points exist when $m_i = j_i$ modulo $P^{n-\beta}$ for all $i = 1, \ldots, d-k$.

For such points let $\alpha = n - \beta$, then the points common to the $k-$dimensional hyperplanes $H((\underline{\theta}_{(k-1)}, 1, \underline{m}))$ and $H((\underline{\theta}_{(k-1)}, 1, \underline{j}))$ can be written

$$(s_1, \ldots, s_{k-1}, cP^{n-\alpha}, c(m_{1 \bmod P^\alpha})P^{n-\alpha}, \ldots, c(m_{d-k \bmod P^\alpha})P^{n-\alpha}).$$

The subgroup defined by these points is maximal for the maximum α satisfying $m_i = j_i$ modulo P^α for all $i = 1, \ldots, d-k$.

□

Theorem 4.3.3 *The redundancy between any two $k-$dimensional hyperplanes of the set H_l, $l = 1, \ldots, d-k-1$, is given by*

$$H((\underline{\theta}_{(k-1)}, \underline{r}, 1, \underline{m})) \cap H((\underline{\theta}_{(k-1)}, \underline{q}, 1, \underline{j})) =$$
$$H((\underline{\theta}_{(k-1)}, P^{n-\alpha+1}(\underline{r}_{\bmod P^\alpha}), P^{n-\alpha}, P^{n-\alpha}(\underline{m}_{\bmod P^\alpha})))$$

where

$$r_i, q_i \in Z/P^{n-1}, \quad i = 1, \ldots, l$$
$$m_i, j_i \in Z/P^n, \quad i = l+1, \ldots, d-k$$

and α is the maximum such that

$$r_i \equiv q_i \bmod P^\alpha$$

and

$$m_h \equiv j_h \bmod P^\alpha$$

hold for all $i = 1,\ldots,l$ and $h = l+1,\ldots,d-k$. The intersection contains $(N_1 \cdots N_{k-1})P^\alpha$ points.

Proof.

Similar to theorem 4.3.2

□

Theorem 4.3.4 *The redundancy between any two $k-$dimensional hyperplanes of the set H_{d-k} is given by*

$$H((\underline{\theta}_{(k-1)}, P\underline{r}, 1)) \cap H((\underline{\theta}_{(k-1)}, P\underline{q}, 1)) = H((\underline{\theta}_{(k-1)}, P^{n-\alpha}(\underline{r}_{\bmod P^\alpha}), P^{n-\alpha-1}, P^{n-\alpha}(\underline{m}_{\bmod P^\alpha})))$$

where

$$r_i, q_i \in Z/P^{n-1}$$

and α is the maximum such that

$$r_i \equiv q_i \bmod P^\alpha$$

holds for all $i = 1,\ldots,d-k$. The intersection contains $(N_1 \cdots N_{k-1})P^\alpha$ points.

Proof. By definition, the points on the hyperplanes

$$H((\underline{\theta}_{(k-1)}, P\underline{r}, 1)), \quad \text{and}$$
$$H((\underline{\theta}_{(k-1)}, P\underline{q}, 1))$$

are equal only when $tPr_i = tPq_i$ modulo P^n for all $i = 1,\ldots d-k$. If $t = 0$ the points of the $(k-1)$ dimensional hyperplane $H(\underline{\delta}_1,\ldots,\underline{\delta}_{k-1},\underline{\theta}_{(d-k+1)})$ are common. If $t \neq 0$ and $p \not| t$ then t is invertible modulo P^n and common points exist only if $r_i = q_i$,

$i = 1, \ldots, d-k$. Otherwise there is some β such that t/P^{β} is invertible modulo P^n, and common points exist if $r_i = q_i$ modulo $P^{n-\beta-1}$ for all $i = 1, \ldots, d-k$.

Let $\alpha = n-\beta-1$, then the points common to the hyperplanes $H((\underline{\theta}_{(k-1)}, P\underline{r}, 1))$ and $H((\underline{\theta}_{(k-1)}, P\underline{q}, 1))$ can be written

$$(s_1, \ldots, s_{k-1}, cP^{n-\alpha}(r_{1 \bmod P^{\alpha}}), \ldots, cP^{n-\alpha}(r_{d-k \bmod P^{\alpha}}), cP^{n-\alpha-1}).$$

The subgroup defined by these points is maximal for the maximum α satisfying $r_i = q_i$ modulo P^{α} for all $i = 1, \ldots, d-k$.

□

Theorem 4.3.5 *The redundancy between any two $k-$dimensional hyperplanes H_x and H_y, where $x \neq y$ is given by*

$$H((\underline{\theta}_{(k-1)}, \underline{r}, 1, \underline{m})) \cap H((\underline{\theta}_{(k-1)}, \underline{q}, 1, \underline{j})) = H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1}, \underline{\theta}_{(d-k+1)}).$$

The intersection contains $(N_1 \cdots N_{k-1})$ points.

Proof. Assume $x > y$, then if a point is common to both hyperplanes $tPr_{y+1} = t$ modulo P^n must hold. Since for every t there exists a maximum β such that $P^{\beta}|t$, it is implied that $Pr_{y+1} \equiv 1 \bmod P^{n-\beta}$ and therefore $r_{y+1} = P^{-1}$ modulo $P^{n-\beta}$. Since P^{-1} does not exist modulo p^v the proof is completed. The only common points exist for $t = 0$, and are contained in the $(k-1)$ dimensional hyperplane $H(\underline{\delta}_1, \ldots, \underline{\delta}_{k-1}, \underline{\theta}_{(d-k+1)})$.

□

DFT Restriction

In a manner similar to the prime case described in the previous section, the $d-$dimensional DFT can be computed by restricting it to the covering set of $k-$dimensional hyperplanes given by theorem 4.3.1. This restriction is detailed below.

Recall the definition of the $d-$dimensional DFT over a region $\mathcal{R}$

$$V(\underline{u}) = \sum_{\underline{a} \in \mathcal{R}} x(\underline{a}) \omega_{N_1}^{a_1 u_1} \cdots \omega_{N_d}^{a_d u_d} \tag{4.23}$$

$$\underline{u} \in \mathcal{R}.$$

The power of prime case of the algorithm requires $d - k + 1$ dimensions to be of equal and power of prime size (including 2). For simplicity of presentation these are assumed to be contiguous, and we can write

$$\mathcal{R} = \mathcal{A} \times \mathcal{B},$$
$$\mathcal{A} = Z/N_1 \times \cdots \times Z/N_{k-1} \times Z/P^n,$$
$$\mathcal{B} = (Z/P^n)^{d-k}, \text{ and } P \text{ is any prime (including 2).}$$

The restriction of the DFT to the $(P^n)^{d-k}$ $k-$dimensional hyperplanes of H_0 is identical to that of the prime case derived in the previous section. To illustrate the derivation of the other terms, the H_1 term of theorem 4.3.1 is considered below.

The $(1/P)(P^n)^{d-k}$ $k-$dimensional hyperplanes of H_1 given by theorem 4.3.1 can be defined by the homomorphism

$$\Omega_1 : \mathcal{A} \longmapsto \mathcal{R}. \tag{4.24}$$

For all $\underline{s} \in \mathcal{A}$, Ω_1 is given by

$$\Omega_1(\underline{s}) = (s_1, \ldots, s_{k-1}, s_k P r_1, s_k, s_k m_2, \ldots, s_k m_{d-k}). \tag{4.25}$$

The restriction of the $d-$dimensional DFT (4.23) to the $k-$dimensional hyperplanes of H_1 is evaluated by replacing $\underline{u}$ with $\Omega_1(\underline{s})$ of (4.25) to obtain

$$V(\Omega_1(\underline{s})) = \sum_{\underline{a} \in \mathcal{R}} x(\underline{a}) \omega_{N_1}^{s_1 a_1} \cdots \omega_{N_d}^{s_k m_{d-k} a_d}, \tag{4.26}$$

$$, \quad \underline{s} \in \mathcal{A}.$$

The inner product of (4.26)

$$\begin{aligned} \Omega_1(\underline{s}) \cdot \underline{a} \;\; = \;\; & s_1 a_1 + \cdots + s_{k-1} a_{k-1} \\ & + s_k (a_k r_1 P + a_{k+1} + a_{k+2} m_2 + \cdots + a_d m_{d-k}) \end{aligned} \tag{4.27}$$

implies

$$\Omega_1(\underline{s}) \cdot \underline{a} = s_1 d_1 + \cdots + s_k d_k = \underline{s} \cdot \underline{d} \tag{4.28}$$

$$\underline{d} \in \mathcal{A}.$$

In (4.28) the identifications

$$\begin{aligned} d_j &= a_j, \qquad j = 1, \ldots, k-1 \\ i_1 &= a_k, \\ i_{j-k} &= a_j, \qquad j = k+2, \ldots, d \\ d_k &= a_k r_1 P + a_{k+1} + a_{k+2} m_2 + \cdots + m_{d-k} a_d \end{aligned} \tag{4.29}$$

may be made, allowing equation (4.26) to be written

$$\begin{aligned} V(\Omega_1(\underline{s})) \;=\; & \sum_{\underline{d} \in A} \sum_{\underline{i} \in B} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k} \\ & x(\underline{d}_{(k-1)}, i_1, d_k - P r_1 i_1 - \sum_{l=2}^{d-k} m_l i_l, i_2, \ldots, i_{d-k}), \\ & \underline{d} \in A. \end{aligned} \tag{4.30}$$

Where

$$a_{k+1} = d_k - i_1 r_1 P - \sum_{l=2}^{d-k} m_l i_l, \tag{4.31}$$

and

$$\underline{d}_{(k-c)} = (d_1, \ldots, d_{k-c}). \tag{4.32}$$

Equation (4.30) can be rewritten as a reduction stage, followed by a DFT stage. The reduction operation is given by

$$a_{\underline{d}}^{H_1} = \sum_{\underline{i} \in B} x \left(\underline{d}_{(k-1)}, i_1, d_k - i_1 r_1 P - \sum_{l=2}^{d-k} i_l m_l, i_2, \ldots, i_{d-k} \right), \quad \underline{d} \in \mathcal{A} \tag{4.33}$$

and the DFT stage is given by

$$\begin{aligned} V(\Omega_1(\underline{s})) = & \sum_{\underline{d} \in \mathcal{A}} a_{\underline{d}}^{H_1} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \\ & \underline{s} \in \mathcal{A}, \quad \underline{m} \in \mathcal{B}. \end{aligned} \tag{4.34}$$

Equations (4.33) and (4.34) give the values of the $d-$dimensional DFT on the $k-$dimensional hyperplanes of H_1.

In a similar manner, the restriction of the $d-$dimensional DFT to each of the remaining $k-$dimensional hyperplanes of the covering set of theorem 4.3.1 is computed by a reduction operation $a_{\underline{d}}^{H_\alpha}$, followed by a $k-$dimensional DFT of those points.

The complete algorithm is stated below for the case where the region of support, $\mathcal{R}$, is of power of prime size in at least $d-k+1$ dimensions. Let $\mathcal{R}$ be given by

$$\mathcal{R} = \mathcal{A} \times \mathcal{B},$$
$$\mathcal{A} = N_1 \times \cdots \times N_{k-1} \times P^n,$$
$$\mathcal{B} = (Z/P^n)^{d-k}, \text{ and } P \text{ is a prime (including 2).}$$

For this case, the algorithm is specified by the following two step procedure:

Step 1 Reduction stage. This stage requires the evaluation of the summations

$$a_{\underline{d}}^{H_1} = \sum_{\underline{i} \in \mathcal{B}} x\left(\underline{d}_{(k-1)},\ d_k - \sum_{l=1}^{d-k} m_l i_l,\ \underline{i}\right), \tag{4.35}$$

$$\vdots$$

$$a_{\underline{d}}^{H_j} = \sum_{\underline{i} \in \mathcal{B}} x\left(\underline{d}_{(k-1)},\ \underline{i}_{(j)},\ d_k - \sum_{l=1}^{j} i_l r_l P - \sum_{l=j+1}^{d-k} m_l i_l,\ i_{j+1}, \ldots, i_{d-k}\right),$$

$$\vdots$$

$$a_{\underline{d}}^{H_{d-k}} = \sum_{\underline{i} \in \mathcal{B}} x\left(\underline{d}_{(k-1)},\ \underline{i},\ d_k - \sum_{l=1}^{d-k} i_l r_l P\right)$$

$$\underline{d} \in \mathcal{A}, \quad m_i \in Z/N_i$$

Step 2 DFT stage. This stage requires the evaluation of the $k-$dimensional DFT

$$V(\Omega_0(\underline{s})) = \sum_{\underline{d} \in \mathcal{A}} a_{\underline{d}}^{H_0} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k}, \tag{4.36}$$

$$\vdots$$

$$V(\Omega_j(\underline{s})) = \sum_{\underline{d} \in \mathcal{A}} a_{\underline{d}}^{H_j} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k},$$

$$\vdots$$

$$V(\Omega_{d-k}(\underline{s})) = \sum_{\underline{d} \in \mathcal{A}} a_{\underline{d}}^{H_{d-k}} \omega_{N_1}^{s_1 d_1} \cdots \omega_{N_k}^{s_k d_k},$$

$$\underline{s} \in \mathcal{A}.$$

The $d-$dimensional DFT over $\mathcal{R} = \mathcal{A} \times \mathcal{B}$ is seen to be computed by

$$\left(\frac{P^n}{P}\right) \frac{P^{d-k+1}-1}{P-1}$$

independent $k-$dimensional DFT over $\mathcal{A}$. The $k-$dimensional DFT are evaluated on data derived from the input data using only additions.

4.4 Appendix

This section enumerates the 3D and 4D cases of the algorithm.

- 3D $\longrightarrow$ 2D, Prime Case.

  $N \times P \times P$ 3D DFT computed by $N \times P$ 2D DFTs,

  where P is prime and N is any number.

  1. Covering planes

$$H_0 = H((0,1,m)), \quad m = 0,1,\ldots,P-1$$
$$H_1 = H((0,0,1)).$$

  2. Reduction stage

$$a^{H_0}_{(d_1,d_2)} = \sum_{i=0}^{P-1} x(d_1,\, d_2 - im,\, i), \qquad m = 0,1,\ldots,P-1$$
$$a^{H_1}_{(d_1,d_2)} = \sum_{i=0}^{P-1} x(d_1,\, i,\, d_2).$$

  3. $N \times P$ 2D DFT stage

$$V(s_1,\, s_2,\, s_2 m) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{H_0}_{(d_1,d_2)}\omega_N^{d_1 s_1}\omega_P^{d_2 s_2}$$
$$V(s_1,\, 0,\, s_2) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{H_1}_{(d_1,d_2)}\omega_N^{d_1 s_1}\omega_P^{d_2 s_2}$$

$$s_1 = 0,1,\ldots,N-1,$$
$$s_2 = 0,1,\ldots,P-1.$$

The computation of the $N \times P \times P$ 3D DFT by this method requires $P+1$ reduction operations and a like number of $N \times P$ 2D DFT. Note that [63] is a source of $O(P \log P)$ prime size FFT algorithms.

To compute a 3D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P \times P \times P$ 3D DFT computed as $P^2 + P + 1$ reduction operations and a like number of P point 1D DFT.

- 3D $\longrightarrow$ 2D , Power of Prime Case.

  $N \times P^n \times P^n$ 3D DFT computed by $N \times P^n$ 2D DFTs,

  where P is prime (including 2) and N is any number.

  1. Covering planes

$$H_0 = H((0,1,m)), \quad m = 0,1,\ldots,P^n - 1$$
$$H_1 = H((0,rP,1)), \quad r = 0,1,\ldots,P^{n-1} - 1.$$

  2. Reduction stage

$$a^{H_0}_{(d_1,d_2)} = \sum_{i=0}^{P^n-1} x(d_1, d_2 - im, i), \qquad m = 0,1,\ldots,P^n - 1$$
$$a^{H_1}_{(d_1,d_2)} = \sum_{i=0}^{P^n-1} x(d_1, i, d_2 - rPi), \qquad r = 0,1,\ldots,P^{n-1} - 1$$

  3. $N \times P^n$ 2D DFT stage

$$V(s_1, s_2, s_2m) = \sum_{d_1=0}^{N-1} \sum_{d_2=0}^{P^n-1} a^{H_0}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_{P^n}^{d_2 s_2}$$
$$V(s_1, rPs_2, s_2) = \sum_{d_1=0}^{N-1} \sum_{d_2=0}^{P^n-1} a^{H_1}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_{P^n}^{d_2 s_2}$$
$$s_1 = 0,1,\ldots,N-1 \text{ and } s_2 = 0,1,\ldots,P^n - 1$$

The computation of the $N \times P^n \times P^n$ 3D DFT by this method requires $P^n + P^{n-1}$ reduction operations and a like number of 2D $N \times P^n$ DFT.

To compute a 3D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P^n \times P^n \times P^n$ DFT computed by $(P^n)^2(1 + 1/P + 1/P^2)$ reduction operations and a like number of P^n point 1D DFT.

- 4D $\longrightarrow$ 3D , Prime Case.

  $N_1 \times N_2 \times P \times P$ 4D DFT computed by $N_1 \times N_2 \times P$ 3D DFTs,

  where P is prime and N_1, N_2 are any numbers.

  1. Covering cubes

$$H_0 = H((0,0,1,m)), \quad m = 0,1,\ldots,P-1$$
$$H_1 = H((0,0,0,1)).$$

  2. Reduction stage

$$a^{H_0}_{(d_1,d_2,d_3)} = \sum_{i=0}^{P-1} x(d_1,\, d_2,\, d_3 - im,\, i), \qquad m = 0,1,\ldots,P-1$$
$$a^{H_1}_{(d_1,d_2,d_3)} = \sum_{i=0}^{P-1} x(d_1,\, d_2,\, i,\, d_3).$$

  3. $N \times N \times P$ 3D DFT stage

$$V(s_1,\, s_2,\, s_3,\, s_3 m) = \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P-1} a^{H_0}_{(d_1,d_2,d_3)}\omega_{N_1}^{d_1 s_1}\omega_{N_2}^{d_2 s_2}\omega_P^{d_3 s_3}$$
$$V(s_1,\, s_2,\, 0,\, s_3) = \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P-1} a^{H_1}_{(d_1,d_2,d_3)}\omega_{N_1}^{d_1 s_1}\omega_{N_2}^{d_2 s_2}\omega_P^{d_3 s_3}$$

$$s_1 = 0,1,\ldots,N_1-1, \quad s_2 = 0,1,\ldots,N_2-1,$$
$$s_3 = 0,1,\ldots,P-1$$

The computation of the $N_1 \times N_2 \times P \times P$ 4D DFT by this method requires $P+1$ reduction operations and a like number of 3D $N_1 \times N_2 \times P$ DFT. Note that [63] is a source of $O(P \log P)$ prime size FFT algorithms.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P \times P \times P \times P$ DFT computed as $P^3 + P^2 + P + 1$ reduction operations and a like number of P point 1D DFT.

- 4D $\longrightarrow$ 3D , Power of Prime Case.

  $N_1 \times N_2 \times P^n \times P^n$ 4D DFT computed by $N_1 \times N_2 \times P^n$ 3D DFT, where P is any prime (including 2) and N_1, N_2 are any numbers.

  1. Covering cubes

$$H_0 = H((0,0,1,m)), \quad m = 0,1,\ldots,P-1$$
$$H_1 = H((0,0,0,1)).$$

  2. Reduction stage

$$a^{H_0}_{(d_1,d_2,d_3)} = \sum_{i=0}^{P^n-1} x(d_1,\, d_2,\, d_3 - im,\, i), \qquad m = 0,1,\ldots,P^n-1$$
$$a^{H_1}_{(d_1,d_2,d_3)} = \sum_{i=0}^{P^n-1} x(d_1,\, d_2,\, i,\, d_3 - rPi), \qquad r = 0,1,\ldots,P^{n-1}-1$$

  3. $N_1 \times N_2 \times P^n$ 3D DFT stage

$$V(s_1,\, s_2,\, s_3,\, s_3 m) = \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P^n-1} a^{H_0}_{(d_1,d_2,d_3)}\omega_{N_1}^{d_1 s_1}\omega_{N_2}^{d_2 s_2}\omega_{P^n}^{d_3 s_3}$$
$$V(s_1,\, s_2,\, rPs_3,\, s_3) = \sum_{d_1=0}^{N_1-1}\sum_{d_2=0}^{N_2-1}\sum_{d_3=0}^{P^n-1} a^{H_1}_{(d_1,d_2,d_3)}\omega_{N_1}^{d_1 s_1}\omega_{N_2}^{d_2 s_2}\omega_{P^n}^{d_3 s_3}$$

$$s_1 = 0,1,\ldots,N_1-1,$$
$$s_2 = 0,1,\ldots,N_2-1,$$
$$s_3 = 0,1,\ldots,P^n-1$$

The computation of the $N_1 \times N_2 \times P^n \times P^n$ 4D DFT by this method requires $P^n + P^{n-1}$ reduction operations and a like number of 3D $N_1 \times N_2 \times P^n$ DFT.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P^n \times P^n \times P^n \times P^n$ DFT computed as $(P^n)^3(1 + 1/P + 1/P^2 + 1/P^3)$ reduction operations and a like number of P point 1D DFT.

- 4D $\longrightarrow$ 2D , Prime Case.

  $N \times P \times P \times P$ 4D DFT computed by $N \times P$ 2D DFTs,

  where P is prime and N is any number.

  1. Covering planes

$$H_0 = H((0,1,m_1,m_2)), \quad m_i = 0,1,\ldots,P-1$$
$$H_1 = H((0,0,1,m_2))$$
$$H_2 = H((0,0,0,1))$$

  2. Reduction stage

$$a^{H_0}_{(d_1,d_2)} = \sum_{i_1=0}^{P-1}\sum_{i_2=0}^{P-1} x(d_1,\, d_2 - m_1 i_1 - m_2 i_2,\, i_1,\, i_2), \qquad m_l = 0,1,\ldots,P-1$$
$$a^{H_1}_{(d_1,d_2)} = \sum_{i_1=0}^{P-1}\sum_{i_2=0}^{P-1} x(d_1,\, i_1,\, d_2 - m_2 i_2,\, i_2)$$
$$a^{H_2}_{(d_1,d_2)} = \sum_{i_1=0}^{P-1}\sum_{i_2=0}^{P-1} x(d_1,\, i_1,\, i_2,\, d_2)$$

  3. 2D $N \times P$ DFT stage

$$V(s_1,\, s_2,\, s_2 m_1,\, s_2 m_2) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{H_0}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_P^{d_2 s_2}$$
$$V(s_1,\, 0,\, s_2,\, s_2 m_2) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{H_1}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_P^{d_2 s_2}$$
$$V(s_1,\, 0,\, 0,\, s_2) = \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P-1} a^{H_2}_{(d_1,d_2)} \omega_N^{d_1 s_1} \omega_P^{d_2 s_2}$$

$$s_1 = 0,1,\ldots,N-1,$$
$$s_2 = 0,1,\ldots,P$$

The computation of the $N \times P \times P \times P$ 4D DFT by this method requires $P^2 + P + 1$ reduction operations and a like number of 2D $N \times P$ DFT. Note that [63] is a source of $O(P \log P)$ prime size FFT algorithms.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P \times P \times P \times P$ DFT computed by $P^3 + P^2 + P + 1$ reduction operations and a like number of P point 1D DFT.

- 4D $\longrightarrow$ 2D , Power of Prime Case.

  $N \times P^n \times P^n \times P^n$ MD DFT computed by $N \times P^n$ 2D DFTs,

  where P is any prime (including 2) and N is any number.

  1. Covering planes

$$
\begin{aligned}
H_0 &= H((0,1,m_1,m_2)), \quad m_i = 0,1,\ldots,P^n-1 \\
H_1 &= H((0,r_1P,1,m_2)) \quad r_i = 0,1,\ldots,P^{n-1}-1 \\
H_2 &= H((0,r_1P,r_2P,1))
\end{aligned}
$$

  2. Reduction stage

$$
\begin{aligned}
a^{H_0}_{(d_1,d_2)} &= \sum_{i_1=0}^{P^n-1}\sum_{i_2=0}^{P^n-1} x(d_1,\, d_2 - m_1i_1 - m_2i_2,\, i_1,\, i_2), \qquad m_l = 0,1,\ldots,P^n-1 \\
a^{H_1}_{(d_1,d_2)} &= \sum_{i_1=0}^{P^n-1}\sum_{i_2=0}^{P^n-1} x(d_1,\, i_1,\, d_2 - r_1Pi_1 - m_2i_2,\, i_2) \\
a^{H_2}_{(d_1,d_2)} &= \sum_{i_1=0}^{P^n-1}\sum_{i_2=0}^{P^n-1} x(d_1,\, i_1,\, i_2,\, d_2 - r_1Pi_1 - r_2Pi_2)
\end{aligned}
$$

  3. 2D $N \times P^n$ DFT stage

$$
\begin{aligned}
V(s_1,\, s_2,\, s_2m_1,\, s_2m_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{H_0}_{(d_1,d_2)}\omega_N^{d_1s_1}\omega_{P^n}^{d_2s_2} \\
V(s_1,\, r_1Ps_2,\, s_2,\, s_2m_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{H_1}_{(d_1,d_2)}\omega_N^{d_1s_1}\omega_{P^n}^{d_2s_2} \\
V(s_1,\, r_1Ps_2,\, r_2Ps_2,\, s_2) &= \sum_{d_1=0}^{N-1}\sum_{d_2=0}^{P^n-1} a^{H_2}_{(d_1,d_2)}\omega_N^{d_1s_1}\omega_{P^n}^{d_2s_2}
\end{aligned}
$$

$$
\begin{aligned}
s_1 &= 0,1,\ldots,N-1, \\
s_2 &= 0,1,\ldots,P^n
\end{aligned}
$$

The computation of the $N \times P^n \times P^n \times P^n$ 4D DFT by this method requires $(P^n)^2(1+1/P+1/P^2)$ reduction operations and a like number of 2D $N \times P^n$ DFT.

To compute a 4D DFT by the line approach each dimension would have to be of equal size. The corresponding case is a $P^n \times P^n \times P^n \times P^n$ DFT computed by $(P^n)^3(1/P^2+1/P+1)$ reduction operations and a like number of P^n point 1D DFT.

Chapter 5

A Parallel Algorithm for MD DFT Computation with No Interprocessor Communication

This chapter introduces a new algorithm for the parallel computation of the multidimensional discrete Fourier transform. Most algorithms for this application attempt to minimize the communication cost of row-column or multidimensional Cooley-Tukey algorithms for a specific computational structure. The main features of the algorithm proposed in this chapter are that it eliminates interprocessor communication on broadcast mode multiprocessors, and maps to machines with any number of processors. The algorithm depends on a machine model that allows concurrent processing and communication, and supports the communication functions broadcast and report.

The methodology proposed in this chapter is based on the $k-$dimensional hyperplane algorithm introduced in chapter 4. The central feature of that algorithm with respect to parallel processing is that it does not intersperse communication stages with processing stages. Two basic applications of the hyperplane algorithm are considered.

The first is a direct mapping of the hyperplane algorithm to the multiprocessor. It is shown that such a mapping can be made when the number of processors is equal to the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array ($k < d$). For this case the $d-$dimensional DFT is required to be of equal and prime, or power of prime, size in $d - k + 1$ (or more) dimensions. The degree

of parallelism of the algorithm is the number of $k-$dimensional hyperplanes in a covering set, and the granularity of the algorithm is a single $k-$dimensional DFT. Under a given performance metric, the speedup of the algorithm is shown to be the ratio of a $d-$dimensional DFT to a $k-$dimensional DFT.

The major benefit of the direct mapping of the hyperplane algorithm is that it eliminates the requirement for interprocessor communication for efficient MD DFT computation. The major limitation of this method are the restrictions it imposes on the machine size. There are only $d-1$ possible mappings of the algorithm, and the degree of parallelism changes exponentially between mappings.

This limitation is addressed by an alternative mapping of the hyperplane algorithm. The variant developed is for those cases where the degree of parallelism of the machine is not matched by the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array. For these cases, the multidimensional Cooley-Tukey (MD CT) algorithm and the hyperplane algorithm are combined. The role of the MD CT algorithm is to factor the $d-$dimensional DFT into stages of lower order $d-$dimensional DFT. The composite algorithm that results has a degree of parallelism equal to the number of $k-$dimensional hyperplanes required to cover one of the resulting lower order arrays. This composite method allows great flexibilty in matching the degree of parallelism of the algorithm to the size of the target processor.

The remainder of the chapter is organized as follows: First, a machine model is presented for a generic broadcast mode multiprocessor. For this machine model a set of performance metrics are introduced and defined. In this context a multiprocessor mapping of the row-column algorithm is given for use as a comparative measure. Then the direct mapping of the hyperplane algorithm to the multiprocessor machine model is stated. Detailed cases of the algorithm are presented for the 2D prime, 2D power of prime (including 2), and 3D prime cases. A design example of the 2D prime case is given for a binary tree implementation. The implementation is benchmarked against the row-column method on the same machine. Finally, the composite algorithm is introduced. The development of the algorithm is motivated

by the need for greater flexibilty in the degree of parallelism of the implementation than is attainable by the direct hyperplane method. The composite algorithm results from the marriage of the multidimensional Cooley-Tukey algorithm and the hyperplane algorithm. The general form of the algorithm is stated, and examples are given.

5.1 A Multiprocessor Hyperplane Algorithm

This section develops a multiprocessor algorithm for MD DFT computation. The algorithm presented is a direct mapping of the hyperplane algorithm of chapter 4 to the parallel machine. The definition of the parallel algorithm is stated in terms of a generic multiprocessor. A model for the multiprocessor is given, and a set of performance measures is specified for that model. A mapping of the row-column algorithm to that machine model is given for later use as a comparitive measure of performance. The section closes with a general statement of the direct mapping of the hyperplane algorithm to the machine model, and detailed examples of the algorithm for the 2D prime, 2D power of prime, and 3D prime cases. The next section provides a design example of the multiprocessor hyperplane algorithm. In that example the generic processor used for the description of the algorithm in this section is replaced by the binary tree computer. The implementation described was implemented on the AT&T BT100 binary tree multiprocessor [3].

5.1.1 Machine Model

This section presents a broadcast mode multiprocessor machine model and gives the mapping of the $k-$dimensional hyperplane algorithm to it. The machine is considered to be a collection of processing elements (PEs) together with an interconnection network for interprocessor communication. The machine is externally connected to a host by a single I/O channel. All data enters the machine through the I/O channel. The communication functions that are supported must include broadcast and report.

1. *Broadcast*: This function downloads data from the I/O channel to all processing elements.

2. *Report*: This function allows a distinguished PE to upload data to the I/O channel.

Definitions and Performance Measures

The terms: *degree of parallelism*, *granularity*, and *speedup* are associated with the description and performance of these machines. The degree of parallelism describes the number of processing elements (PE) composing a machine, or the number of independent processes of a task. The *granularity* of a task refers to the size and type of computation performed at each PE. A *scalable* algorithm is one that can be mapped to machines of varying degrees of parallelism. The *Speedup* of a task is a measure of the multiprocessor execution time to the single processor execution time. This term is developed below.

Every task can be thought of as consisiting of parts which can be computed in parallel, and parts which can not be computed in parallel [27]. Given this, the single processor execution time T of a task can be written

$$T = T_{ser} + T_{par}, \tag{5.1}$$

where T_{ser} denotes the execution time of those sections of the task which are not amenable to parallel computation, and T_{par} is the execution time of those sections which could be computed in parallel. When parallelism is introduced to the computation of the task, the time improvement can occur only in T_{par}. In the limit, maximum speedup cannot exceed T_{ser}. This is essentially a statement of Amdahl's law [4] which says that the speedup can not exceed that obtained if the parallel section of the task is executed in zero time.

For the purpose of this work, the term Su will denote speedup. The speedup, Su, of a task is taken to be the ratio of the single processor execution time to the multiprocessor execution time for those parts of the code that can be parallelized.

In addition to these measures the time performance of an algorithm is further described in terms of its: *execution time* T_{exec}, *pipeline time* T_{pipe}, and *delay time* T_{delay}. The execution time T_{exec} of a process is measured from the first input to the last output. It includes the computation time, communication time, and overhead time. Pipeline time T_{pipe} is the average time for producing complete computations. Delay time T_{delay} is the delay from first input to last output in a pipelined system.

Speedup is defined in the previous section, and is taken to be the ratio of a single processor's time performance to the multiprocessor's time performance on the parallelized portion of the task.

Comparitive Measures

Through out this chapter a broadcast mode multiprocessor mapping of the row-column algorithm is used for comparitive purposes. That algorithm is stated in figure 5.1.1.

---

1. Broadcast the N rows of x uniformly among the PEs; each PE is assigned its own set of rows.
2. In parallel, compute the N independent 1D FFT(N) associated with the rows.
3. Globally transpose the intermediate results in the machine.
4. In parallel, compute the N independent 1D FFT(N) associated with the columns.
5. Upload results. The transformed data is stored column-wise in the machine. A transpose must be achieved at some point of the upload process if data is to be returned in row major form.

Figure 5.1: A multiprocessor implemetation of the row-column algorithm.

By the algorithm of figure 5.1.1, a 2D DFT task is seen to require alternating stages of communication and computation. The communication stages include the download of input data, the upload of transformed data, and the exchange of intermediate results between processors. The computation stages are both 1D DFT of N points. The serial portion of the algorithm (those sections that can not be computed

in parallel) are the data download and data upload. The parallel portions of the algorithm are the 1D DFTs. The determination of whether or not the global transpose can be performed in parallel depends on the interconnection network, and is not considered at this time. Assuming granularity of an entire DFT, the computation of a DFT can not proceed until the entire data set to be transformed is available at the PE. Given this, the computational speedup of an N processor implementation of the algorithm is

$$Speedup = \frac{2 \cdot N \cdot FFT(N)}{2 \cdot FFT(N)} = N$$

The overall time performance of the algorithm is limited by the N^2 word data download and data upload operations. The time cost of the transpose has not been considered here.

5.1.2 Algorithm Definition

Below, the mapping of the $k-$dimensional hyperplane algorithm of chapter 4 to the broadcast mode multiprocessor described above is given for prime and power of prime cases. The hyperplane algorithm is computed by a reduction stage followed by a stage of $k-$dimensional DFTs. The prime case of the algorithm requires

$$\frac{P^{d-k+1} - 1}{P - 1}$$

reduction operations $a_{\underline{d}}^{H_i}$ given by expression (4.20) of section 4.2. The power of prime case has more redundancy, and requires

$$\left(\frac{P^n}{P}\right)^{d-k} \cdot \left(\frac{P^{d-k+1} - 1}{P - 1}\right)$$

reduction operations $a_{\underline{d}}^{H_i}$ given by expression (4.35) of section 4.3. Throughout the sequel, let M denote the number of reduction operations of the algorithm (*ie.* number of covering $k-$dimensional hyperplanes). For both cases the $k-$dimensional DFTs are computed on data derived from the input using only additions. They produce data on hyperplanes H_i of the output array. These hyperplanes are defined by theorem 4.2.1 or 4.3.1 for the prime and power of prime cases respectively. The necessary conditions for implementing the hyperplane algorithm on a multiprocessor with no interprocessor communications are given by the following theorem and proof.

Theorem 5.1.1 *The optimal degree of parallelism for minimizing interprocessor communication is the number M of $k-$dimensional hyperplanes required to cover the output array.*

Proof. A k-dimensional discrete Fourier transform is performed on the $k-$dimensional array produced by each reduction operation. Hence, no interprocessor communication is required from input to output if the granularity of the reductions $a_{\underline{d}}^{H_i}$ is no finer than that to put all $\underline{d} \in Z/N_1 \times \cdots \times Z/N_k$ points of an $a_{\underline{d}}^{H_i}$ in a single processor. Any finer degree would necessitate interprocessor communication between the stage in which the summations of the reduction are computed and the stage where the $k-$dimensional discrete Fourier transforms are computed.

□

Consider also that the reduction stage has no data interdependencies, and can be computed simultaneously with the data download. Exploiting this concurrency places a lower limit on the partitioning of the computation. Parallel machines externally connected to an I/O channel have completion times bound by the N^2 word download and N^2 word upload. Regardless of the speed of addition, or the number of PEs, the download time of N^2 words limits the reduction stage completion time. Assuming granularity no finer than a complete reduction operation, an L PE machine reaches the limit if add time is $\lceil M/L \rceil$ faster than communication time.

All cases of the multiprocessor hyperplane algorithm are defined by the same basic structure. That structure is outlined below.

---

1. Assign the reduction operations $a_{\underline{d}}^{H}$ uniformly among the processing elements (PE).

2. During the download (broadcast) of the input data to the machine, each PE forms the $a_{\underline{d}}^{H_i}$ terms assigned to it. For every $a_{\underline{d}}^{H_i}$ term assigned to a PE only 1 addition is performed for each word broadcast to the PE. When the input download is complete, the reduction stage of the computation is also complete.

3. Each PE performs a single $k-$dimensional DFT to finish the computation.

4. Upload results and remove redundant data.

Figure 5.2 : Structure of the multiprocessor hyperplane algorithm

---

Using the hyperplane algorithm, a MD DFT task requires stages of communication and stages of computation. The communication stages include only the data download and data upload operations. The computation stages are the DFT and data reduction operations. The communication and computation stages are the same as the serial and parallel segments of the algorithm. That is, the serial portions of the algorithm are the data download and the data upload, and the parallel segments of the algorithm are the DFT and reduction operations. Unlike the computation of the DFT, the reduction operations contain no data interdependencies, and can be computed in parallel with the download operation. On a machine with M processors, the computational speedup of the algorithm relative to a single processor computing the row-column algorithm is

$$Speedup = \frac{d}{k}(N)^{d-k}.$$

This assumes the additions of the reduction stage are computed simultaneously with the data download and incur no time penalty of their own. In the remaninder of this section the algorithm is stated in some detail for the 2D prime, 2D power of prime, and the 3D prime cases. The pertinent equations are (2.31) thru (2.34) and (2.40) thru (2.43) for the 2D prime and power of prime cases respectively, the derivations for the 3D case are given in section 4.2. In the next section a design example of the 2D prime case on a tree machine is given.

Prime Case

The 2D prime case of the algorithm is developed below. For this case the hyperplane algorithm reduces to the line algorithm. In section 2.2, the prime case of the line algorithm was shown to be evaluated by the following two step procedure.

1. Reduction stage. Compute the $P+1$ summations:

$$a_d^{(m,1)} = \sum_{i=0}^{P-1} x(i, d-mi)$$

$$a_d^{(1,0)} = \sum_{i=0}^{P-1} x(d, i)$$

2. DFT stage. Compute the $P+1$ one-dimensional P−point DFTs:

$$V(mt,t) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(m,1)}, \qquad m = 0,1,\ldots,P-1$$

$$V(t,0) = \sum_{d=0}^{P-1} \omega_P^{dt} a_d^{(1,0)}, \qquad t = 0,1,\ldots,P-1$$

The summation terms $a_d^{(m,1)}$ reduce the input data to the vectors that must be transformed in order to obtain the output along the line $L((m,1))$. The summation term $a_d^{(1,0)}$ gives the vector that must be transformed to obtain the output along the line $L((1,0))$. The computation of the $P \times P$ line algorithm on a broadcast mode multiprocessor is realized by the procedure of figure 5.1.2.

1. Assign the computation of the reduction operations $a^{(x,y)}$ evenly among M or fewer PEs.

2. Broadcast the rows of input data to the PEs and simultaneously compute the reductions $a^{(x,y)}$ at the PEs. The reduction operations are computed as follows: When row i is received, the PE assigned $a^{(1,0)}$ sums the elements of the row and places that sum in position i of $a^{(1,0)}$. In a similar manner, the PE assigned $a^{(m,1)}$ rotates row i mi positions to the right and sums it componentwise to the other received rows. In this fashion $a_d^{(m,1)}$ will exist at position d of the first row received. The rotation can be achieved by an address offset and modulo P address arithmetic.

   Note that the partial result due to row i is accumulated as row i is received. In this way each PE requires only $O(N)$ storage for each $a^{(x,y)}$ it is assigned.

3. In parallel, each PE computes a P–point 1D DFT for every reduction $a^{(x,y)}$ assigned to it.

4. Upload data to host. There is some redundancy in the data, and the data is permuted among the PEs.

   Redundancy: For this case, the redundancy is trivially that output point $V(0,0)$ is common to every PE.

   Permutation: The permutation is that every PE contain a line of output data as specified by theorem 2.2.1. The row 0 and column 0 are in the PEs which computed the $a^{(0,1)}$ and $a^{(1,0)}$ terms, and are not permuted. Element l of the vector produced by the PE that computed $a^{(m,1)}$ is the column l and row $(N - ml) \bmod P$ element of the 2D DFT.

Figure 5.2: 2D $P \times P$ multiprocessor mapping of the line algorithm.

To obtain a measure of the performance of the algorithm, allow a comparison of a multiprocessor with $P+1$ PEs running the parallelized line algorithm to a single processor computing the row column algorithm. Let the computational burden of the reduction stage be taken simultaneously with the data download. Each reduction operation for the 2D $P \times P$ case requires $O(P^2)$ additions. Then the speedup of the parallel line algorithm on $P+1$ processors relative to the row-column method on a single processor is

$$Speedup = \frac{2 \cdot P \cdot FFT(P)}{FFT(P)} = 2P.$$

The algorithm achieves a $2P$ speedup with $P+1$ processors rather than the expected P speedup because the number of 1D DFT has been reduced from $2P$ to $P+1$. The justification for not including the time cost of the additions of the reduction stage is that these additions are concurrent with the data download operation, which is common to both algorithms and can not be parallelized.

The performance improvement of the algorithm depends entirely on the computation of the reduction operations in parallel with the data download. Consider that a single processor computing just the reduction stage of the line algorithm would require $O(P^3)$ additions. Below, the mapping of the 2D power of prime and 3D prime cases of the algorithm are considered. Then a design example implementing the 2D prime case of the algorithm on a tree computer is shown. This algorithm is compared to the row-column algorithm on the same machine.

Power of Prime Case

This section gives the power of prime case of the algorithm specialized for $2^n \times 2^n$ 2D DFT. The main differences between the prime and power of prime cases are the computation of the reduction stage, and the removal of redundant data. The power of prime case of the line algorithm is restated below.

1. Reduction stage. Compute the $2^n + 2^{n-1}$ summations:

$$a_d^{(m,1)} = \sum_{i=0}^{2^n-1} x(i, d - mi)$$

$$a_d^{(1,2s)} = \sum_{i=0}^{2^n-1} x(d - 2si, i)$$

2. DFT stage. Compute $2^n + 2^{n-1}$ one-dimensional 2^n–point DFTs:

$$V(mt, t) = \sum_{d=0}^{2^n-1} \omega^{dt} a_d^{(m,1)}, \quad m = 0, 1, \ldots, 2^n - 1$$

$$V(t, 2st) = \sum_{d=0}^{2^n-1} \omega^{dt} a_d^{(1,0)}, \quad s = 0, 1, \ldots, 2^{n-1} - 1$$

$$t = 0, 1, \ldots, 2^n - 1$$

The computation of the $a^{(m,1)}$ terms is the same as the prime case. That is, when row i is recieved, the PE assigned $a^{(m,1)}$ rotates it mi positions to the right, and adds component-wise to the previously received rows. This compresses the data so that the storage requirement for this stage is only that of a single row.

The computation of the $a^{(1,2s)}$ terms is similar. The PE assigned $a^{(1,2s)}$ rotates column i $2si$ positions to the right, and adds component-wise to the other columns. The problem this introduces is that the data is downloaded in row major order. The following procedure computes the $a^{(1,2s)}$ terms given row-major data.

Let α be the maximum such that $2^\alpha | 2s$, and let $x(i, j)$ denote element j of row i. As row i is received, the PE sums all elements of the row which are $2^{n-\alpha}$ positions apart. There will be $2^{n-\alpha}$ such sums each of 2^α elements. These can be expressed as

$$c_{i,l} = \sum_{k=0}^{2^\alpha-1} x(i, l + 2^{n-\alpha}k), \quad l = 0, 1, \ldots, 2^{n-\alpha} - 1.$$

The l^{th} of these $2^{n-\alpha}$ sums of the elements of row i are accumulated with the $(i + 2sl) \bmod 2^n$ element of $a^{(1,2s)}$. Essentially, the $c_{i,l}$ terms of any $a^{(1,2s)}$ are computed by striding through a received row by $2^{n-\alpha}$ and forming an accumulated sum of those points. The following example illustrates this procedure.

Example: Consider a 4×4 2D DFT. The reduction operations $a^{(1,2s)}$ are required for $s = 0, 1$. Using the procedure outlined above, the reductions are formed from downloaded rows in the following manner.

For $s = 0$, the maximum α such that $2^\alpha | 0$ is n. The PE assigned this term must add elements of each row that are 1 apart; all elements of each row received are summed together. The term $a_d^{(1,0)}$ is the accumulated sum of row d.

For $s = 1$, the required reduction operation is $a^{(1,2s)}$. The maximum α such that $2^\alpha | 2s$ is $\alpha = 1$. As row i is received, the PE sums elements which are 2 positions apart. There will be 2 such sets formed for each row. These are:

$$\begin{aligned} c_{0,0} &= x_{0,0} + x_{0,2} & \qquad c_{0,1} &= x_{0,1} + x_{0,3} \\ c_{1,0} &= x_{1,0} + x_{1,2} & \qquad c_{1,1} &= x_{1,1} + x_{1,3} \\ c_{2,0} &= x_{2,0} + x_{2,2} & \qquad c_{2,1} &= x_{2,1} + x_{2,3} \\ c_{3,0} &= x_{3,0} + x_{3,2} & \qquad c_{3,1} &= x_{3,1} + x_{3,3}. \end{aligned}$$

After row i is received and the $c_{i,l}$ are formed, the elements of $a_d^{(1,2)}$ are given as:

$$\begin{aligned} a_0^{(1,2)} &= c_{0,0} + c_{1,2} \\ a_1^{(1,2)} &= c_{0,1} + c_{1,1} \\ a_2^{(1,2)} &= c_{1,0} + c_{0,2} \\ a_3^{(1,2)} &= c_{1,1} + c_{0,3}. \end{aligned}$$

This finishes the reduction stage of the algorithm. The 1D DFT are performed on points

$$a_d^{(x,y)}, \quad d = 0, \ldots, P-1$$

to complete the computation.

□

As in the prime case, when the computation is complete there are some output points that appear in more than one processor. The redundant points are identified by theorems 2.2.3 - 2.2.5. The procedure of figure 5.3 allows the data to be uploaded with the redundant points removed. In the procedure, $L((x,y))$ denotes the line of output computed by the PE that was assigned the reduction operation $a^{(x,y)}$.

---

1. Upload all points of the line $L((0,1))$.
2. For each $\alpha = 0, 1, \ldots, n-1$, define $j = 2^{\alpha}, \ldots, 2^{\alpha+1} - 1$ and upload all points on the lines $L((j,1))$ except multiples of $2^{n-\alpha}$.
3. Upload all points of the line $L((1,0))$ except $(0,0)$.
4. For each $\alpha = 0, 1, \ldots, n-2$, define $k = 2^{\alpha}, \ldots, 2^{\alpha+1} - 1$ and upload all points on the lines $L((1,2k))$ except multiples of $2^{n-\alpha-1}$.

Figure 5.3: $2^n \times 2^n$ data upload procedure.

Example: Consider a 4×4 2D DFT. The array below shows the effect of the upload strategy of figure 5.3 on the set of covering lines. The leftmost column of the array enumerates the lines $L((x,y))$ that cover the output array. All elements of the vector $L((x,y))$ are listed in the corresponding row of the array. The elements that are not uploaded are boxed.

$$
\begin{array}{llllll}
L((0,1)) & V_{0,0} & V_{0,1} & V_{0,2} & V_{0,3} \\
L((1,1)) & \boxed{V_{0,0}} & V_{1,1} & V_{2,2} & V_{3,3} \\
L((2,1)) & \boxed{V_{0,0}} & V_{2,1} & \boxed{V_{0,2}} & V_{2,3} \\
L((3,1)) & \boxed{V_{0,0}} & V_{3,1} & \boxed{V_{2,2}} & V_{1,3} \\
L((1,0)) & \boxed{V_{0,0}} & V_{1,0} & V_{2,0} & V_{3,0} \\
L((2,0)) & \boxed{V_{0,0}} & V_{1,2} & \boxed{V_{2,2}} & V_{3,2}
\end{array}
$$

Observe that all the elements of any vector (line) not uploaded from a PE are multiples of 2 apart.

□

Note that line algorithm required 6 1D DFT to compute the DFT of the example. This compares to the 8 1D DFT required by the row-column algorithm. Given that the additions of the reduction operations are performed concurrently with the input data download, a 6 PE machine could compute this DFT in the time required for the data upload and download, plus the computation time for one 1D DFT. This represents a speedup in the DFT computation of 8 relative to a single processor executing the row-column method.

If the allowed granularity is no smaller than a 1D DFT the maximum degree of parallelism of the row-column method is only 4, even though 8 1D DFT are required. This is due to the fact that the 1D DFT of the second stage (*ie.* column) are dependent on the first stage. On a 4 PE machine, the row-column method would require time for the data upload and download, time for 2 1D DFT, and time for

a global data transpose that requires every PE to exchange data with every other PE. If we assume that the global transpose requires zero time, then the speedup of this method relative to a single processor computing the row-column method is 4.

The parallelized row-column method for $N \times N$ 2D DFT, $N = 2^n$, gives a linear speedup limited by N (disregarding transpose time). The parallel line algorithm attains speedup of $2N$ given $3N/2$ PEs. This "super linear" performance is due to the reduction of the number of 1D DFT from $2N$ to $3N/2$. The major limitation of this method is that it does not scale to machines whose degree of parallelism is lower than $3N/2$. This constraint will be eliminated in later sections. Before proceeding to that work the 3D prime case of the algorithm is discussed. This section closes with a detailed implementation of the 2D prime case of line algorithm on a tree machine. This implementation is compared to a realization of the row-column algorithm on the same machine.

3D Case

The general mapping of the 3D prime case of the hyperplane algorithm is presented below. The primary motivation for the algorithm is that it exchanges increased computation time (larger granularity) for a decreased degree of parallelism relative to the parallel line method (k=1).

The example covered in this section is the 3D $P \times P \times P$ DFT, where P is a prime number. If the line algorithm were applied (case of $k = 1$), a number of lines equal to $P^2 + P + 1$ would be required to cover the output array, and therefore a like number of processors is needed. To reduce the degree of parallelism of the computation, take $k = 2$ and compute planes of the output. For this case the degree of parallelism is $P + 1$ (number of planes in a covering set) and the granularity is a single $P \times P$ 2D DFT.

In the previous section the convention that data was input to the machine in row-major order was adopted. In order to remain consistent with this convention it is necessary to consider a different set of covering planes that those given in theorem

4.2.1. Allow the covering planes to be given by

$$\begin{aligned} H((0,m,1)) &= \{(s_1, s_2 m, s_2) : s_1, s_2 \in Z/P\}, \\ H((0,1,0)) &= \{(s_1, s_2, 0) : s_1, s_2 \in Z/P\}, \\ & m = 0, 1, \ldots, P-1. \end{aligned} \tag{5.2}$$

Various data flows can be obtained by permuting the order of the components of the hyperplanes of theorems 4.2.1 and 4.3.1. In order for the sets to maintain their covering property each hyperplane must be permuted in the same way.

For the hyperplanes of (5.2) the algorithm is given by the following procedure:

1. Reduction stage. Compute the $P+1$ summations:

$$\begin{aligned} a_{d_1,d_2}^{H((0,m,1))} &= \sum_{i=0}^{P-1} x(d_1, i, d_2 - mi) \\ a_{d_1,d_2}^{H((0,1,0))} &= \sum_{i=0}^{P-1} x(d_1, d_2, i) \end{aligned}$$

2. DFT stage. Compute the $P+1$ 2D DFT:

$$\begin{aligned} V(s_1, s_2 m, s_2) &= \sum_{d_1=0}^{P-1} \sum_{d_2=0}^{P-1} a_{d_1,d_2}^{H((0,m,1))} \omega^{s_1 d_1} \omega^{s_2 d_2}, \\ V(s_1, s_2, 0) &= \sum_{d_1=0}^{P-1} \sum_{d_2=0}^{P-1} a_{d_1,d_2}^{H((0,1,0))} \omega^{s_1 d_1} \omega^{s_2 d_2}, \end{aligned}$$

$$m = 0, 1, \ldots, P-1 \quad \text{and} \quad s_1, s_2 = 0, 1, \ldots, P-1.$$

The computation of the 3D prime algorithm on a multiprocessor is essentially the same as the 2D case given in figure 5.1.2. They key differences between them are the evaluation of the reduction operations and the removal of the redundancy in the data upload. These are specified below.

Assign the $P+1$ reduction operations a^H to the PEs. During the download operation the rows of input data are broadcast to the machine. Let row (r_1, r_2) denote the P input points $x_{(r_1,r_2,0)}, \ldots, x_{(r_1,r_2,p-1)}$ of the input array. The PE assigned the computation of

$$a_{d_1,d_2}^{H((0,1,0))}$$

forms an accumulated sum of the elements of row (r_1, r_2) when it is received. That sum is placed in location (r_1, r_2) of $a^{H((0,1,0))}$. The computation of this term requires one addition for each word input to the PE, and is completed when the last input enters the PE.

The PEs assigned the computation of the $a^{H((0,m,1))}$ terms compute as follows: When row (r_1, r_2) of the input is received it is rotated mr_2 positions to the right and summed componentwise with the elements of row r_1 of $a^{H((0,m,1))}$. The rotation is achieved by address offset and modulo N address arithmetic. Similar to the 1D case, the summation operation compresses the data. Therefore the reduction stage requires the PE to have storage for only P^2 points for each reduction operation assigned to it.

After the reduction stage is finished, the 2D DFTs are performed on the a^H in each PE. This completes the computation. The output is distributed among the PEs as hyperplanes. There are output points that were computed in more than one PE. It is desireable that these be removed before the results are uploaded. The redundant points are all on the the line $L((1,0,0))$ of the output array. In order to eliminate these points during the data upload, only one PE is required to upload the points on the line $L((1,0))$ in its 2D output array. That is, only one PE uploads the transformed points $a^H_{0,0}, a^H_{1,0}, \ldots, a^H_{P-1,0}$.

For the 3D prime case the multiprocessor hyperplane algorithm requires a degree of parallelism equal to $P+1$ and the granularity of a single 2D $P \times P$ DFT. Therefore, the computational speedup relative to a single processor computing the row-column method is

$$Speedup = \frac{3 \cdot P^2 \cdot FFT(P)}{2 \cdot P \cdot FFT(P)} = \frac{3P}{2}.$$

This does not account for the data upload and download or the additions of the reduction operations that are performed simultaneously with the download.

For comparative purposes, consider a P processor machine computing the row-column like procedure of figure 5.1.2. The modified row-column algorithm of figure 5.1.2 requires communication and computation stages. The communication stages include the upload and download of data points, and the exchange on intermediate

1. Broadcast two dimensions of data to each PE.
2. Compute 2D $P \times P$ DFT at each PE.
3. Globally transpose the data in the machine.
4. Compute 1D DFT on the remaining dimension.
5. Upload the data.

Figure 5.4: A modified row-column 3D algorithm.

results (global transpose). The computation stages are 1D and 2D DFT. If the 2D DFT at each PE are computed by the row-column method, then this algorithm is seen to require the time to compute $3P$ 1D DFT. Its computational speedup relative to a single processor computing the row-column method is

$$Speedup = \frac{3 \cdot P^2 \cdot FFT(P)}{3 \cdot P \cdot FFT(P)} = P.$$

The P processor implementation of the row-column like procedure of figure 5.1.2 has a speedup of P, but the $P + 1$ processor implementation of the hyperplane algorithm has a speedup of $3P/2$. The hyperplane algorithm achieves a 50% improvement over the row-column method at the cost of only a single processor. The advantage of the hyperplane method is due to its overall reduced number of DFTs relative to the row-column method.

5.2 Design Example

This section presents a design example using the multiprocessor hyperplane algorithm. In the previous section the algorithm was defined relative to a generic multiprocessor machine model. In this section, the binary tree machine is selected to give a specific case study of an implemetation of the algorithm. The tree architecture is choosen because it provides an easily scaled low cost parallel architecture that supports all the communication functions required. This section reviews the binary tree architecture and maps the hyperplane algorithm to it for the 2D prime case. For this case the hyperplane algorithm and the line algorithm are identical.

5.2.1 The Binary Tree Computer

Consider a machine where each PE is connected to its parent, a right child and a left child. Each PE also has a CPU and local memory. The entire tree is assumed to be connected to a communication channel through the root. Identical PE's are distinguished by some position dependent labeling scheme ie. breadthfirst numbering.

Processing at the PEs is done in single program multiple data SPMD mode. In [24] this is described as a sliced procedure where the same program is executed simultaneously at all the PEs on possibly different datasets. The constraint this imposes is that the PEs must begin processing at the same point and return from processing to the same point. During processing, program flow is free to follow different paths based on the dataset.

The communication functions required are broadcast and report. In broadcast mode the channel sends data to all the PEs in the tree. In report mode a PE distinguished by ID number sends data to the channel.

Two enhancements of this basic architecture are also considered. The first allows the PEs of the tree to perform concurrent communication and processing. The second also allows the PEs of the tree to perform concurrent communication and processing, and extends the model to support task level pipelining as well.

The model described above is the target machine for the $k-$dimensional hyper-

plane implementation of this section. The implementation is for the 2D prime case for $k = 1$. For this case, the $k-$dimensional hyperplane algorithm reduces to the line algorithm. The pertinent formulas defining the algorithm were restated in the last section. For this implementation let the tree have $M = P + 1$ PEs numbered 1 to M. The M reductions are assigned to the PEs in the natural order. That is, PE_m is assigned the computation of the reduction term $a^{(m,1)}$ for $m = 0, ..., P - 1$, and PE_P is assigned the computation of the reduction term $a^{(1,0)}$. The algorithm can be scaled for $K < M$ PEs by assigning $L = \lceil M/K \rceil$ or fewer reduction operations to each PE. Given the above, the code fragment of figure5.2.1 computes the parallel line algorithm on the tree. Note that this code is similar to that realized for the AT&T BT100 binary tree computer.

Throughout this chapter a tree machine mapping of the row-column algorithm [24] is used for comparative purposes. This algorithm was stated in figure 5.1.1. It requires two stages of P 1D DFT seperated by a global transpose. The transpose requires interprocessor communication that are not supported directly by the tree architecture. The communication requirement of the transpose is achieved using only host-PE communication.

```
                    /* Reduction Stage */
For (k = 0 to number of rows −1);
    Broadcast row k to all PEs;
    If (k ≠ 0) then
      Do_parallel: For (i = 0 to P);
                          compute partial reduction a^(x,y);
      End;
End;


                    /* 1D DFT Stage */
Do_parallel: For (i = 0 to P);
                    perform 1D FFT(P); /*  a_d^(x,y) → v_d */
End;


                    /* Data Upload Stage */
For k = 0 to P

    If (k = 0) then
      For (j = 0) to (P − 1)
          V[0, j] = Report(PE_k, v_j);

    If (k = P) then
      For (j = 1) to (P − 1)
          V[j, 0] = Report(PE_k, v_j);

    If (0 < k < P) then
      For (j = 1) to (P − 1)
          V[jk, j] = Report(PE_k, v_j);
End;
```

Figure 5.5: Tree machine mapping of the multiprocessor line algorithm. 2D prime case.

5.2.2 Performance Evaluations

In this section the performance of the multiprocessor hyperplane algorithm is analyzed for the 2D square prime case. Throughout this section, the row-column method is used for comparison. The machine capabilities that are analyzed are listed below.

1. Tree architectures with no concurrency between communication and processing. Such machines exploit the computational parallelism of the M independent reductions and M independent 1D DFT.

2. Tree architectures capable of concurrent communication and processing. These exploit both the parallelism of the reduction and DFT stages, as well as the parallelism between the download and reduction stage.

3. Trees capable of concurrent communication and processing, as well as task level pipelining. These exploit the more general parallelism between independent 2D DFT tasks.

These three systems are evaluated with respect to the performance measures: execution time T_{exec}, pipeline time T_{pipe}, delay time T_{delay}, and speedup Su, which were defined earlier in this chapter. Unless otherwise specified all expressions are for complex data.

For $N \times N$ data, the execution time of the multiprocessor line algorithm on a tree with K PEs and no concurrency between communication and processing is:

$$\begin{aligned} T_{exec} \;=\; & \text{download time} + \text{reduction stage time} \\ & + \text{FFT(N) stage time} + \text{Upload time} \\ & + \text{Overhead time.} \end{aligned} \tag{5.3}$$

To quantify this expression let $L = \lceil M/K \rceil$ be the number of reduction operations assigned each PE. Let A and C denote word addition and communication time, $T(F_N)$ be the time to compute a 1D FFT(N), and assume the overhead time is negligible. Furthermore, let the host perform the upload permutation as defined in

section 2 with no time penalty. Then the execution time of (5.3) becomes:

$$T_{exec} = 2CN^2 + ALN^2 + LT(F_N). \tag{5.4}$$

As the number of PEs falls below M, N^2 additions must be performed for each additional reduction operation assigned to a PE. This is a large number of additions, but as the following analysis will show the line algorithm still compares favorably to the row-column algorithm because of that method's interprocessor communication requirement. The row-column method of given in figure (5.1.1) has execution time:

$$\begin{aligned} T_{exec} \;=\; & \text{download time} + \text{FFT(N) stage time} \\ & + \text{transpose time} + \text{FFT(N) stage time} \\ & + \text{Upload time} + \text{Overhead time.} \end{aligned} \tag{5.5}$$

Performing the transpose using only host-PE communication and letting $L' = \lceil N/K \rceil$, the execution time becomes

$$T_{exec} = 4CN^2 + 2L'T(F_N). \tag{5.6}$$

The tradeoff between these methods is the reduction stage of the line algorithm for the transpose stage and one stage of FFT(N) of the row-column algorithm. This trade is LN^2 additions for $2N^2$ communication operations plus the operations required to compute L' FFT(N). In terms of time this is ALN^2 versus $2CN^2 + L'T(F_N)$. For trees with M PEs equal to the number of reduction operations, the tradeoff is time $2CN^2 + T(F_N)$ versus AN^2. So for M PEs, the parallel reduction algorithm is faster than the row-column algorithm whenever addition time is less than twice communication time (assume $T(F_N)$ is minimal). Currently there are DSP chips available [2] that perform floating point additions above $10MFlops$ on 32 bit words. The communication channel must average $80MBytes$ for the row-column algorithm to breakeven with the parallel reduction algorithm if such PEs are used.

Another feature of the parallel line (and $k-$dimensional hyperplane) algorithm is its ability to exploit real input data for performance improvement. The reduction stage performs its additions on data that is only real, but the row-column algorithm

must transpose complex data. This doubles the tradeoff to $4CN^2$ versus AN^2, and a communication channel capable of averaging $160MBytes$ is required for the row-column algorithm to break even. Furthermore, the DFT stage of the parallel line method is on real data, while only one of the DFT stages of the row-column method is on real data.

If the tree supports concurrent communication and processing then the execution time for the parallel line algorithm improves to

$$T_{exec} = max(ALN^2, CN^2) + LT(F_N) + CN^2 \tag{5.7}$$

which for M PEs is

$$T_{exec} = max(AN^2, CN^2) + T(F_N) + CN^2. \tag{5.8}$$

The source of the improvement is that the summations of the reduction stage are computed concurrently with the input data download. This approaches the $2CN^2$ time limit of machines with a single I/O channel, particularly if N is a power of 2 (however this requires an increase of the degree of parallelism to 3N/2). The space-time diagram below shows the allocation of the machines resources during this computation. The vertical axis depicts the I/O channel and a processing element. The horizontal axis depicts time.

	$\max(AN^2, CN^2)$	$T(F_N)$	CN^2
I/O	*Download*		*Upload*
PE_i	*Reduction*	$FFT(N)$	

The row-column method cannot benefit from this architectural enhancement since it requires all the data to be present at the PEs before an FFT stage can begin. A space-time diagram for this computation is

	CN^2	$T(F_N)$	$2CN^2$	$T(F_N)$	CN^2
I/O	*Download*		*Transpose*		*Upload*
PE_i		$FFT(N)$		$FFT(N)$	

If task level pipelining is permitted, the row-column algorithm can perform the communication stages of one 2D FFT simultaneously with another's computation

stages. The execution, pipeline, and delay times for this two task algorithm are

$$T_{exec} = 8CN^2, \quad T_{pipe} = 4CN^2, \quad T_{delay} = 7CN^2.$$

Here, execution time measures the pipeline period of the two-task algorithm, and it is assumed that $LT(F_N) \leq 2CN^2$. A space-time diagram for this computation is

	CN^2	CN^2	$2CN^2$	$2CN^2$	CN^2	CN^2
I/O	$Down\,1$	$Down\,2$	$Trnsp\,1$	$Trnsp\,2$	$Upload\,1$	$Upload\,2$
PE_i		$FFT(N)\,1$ (row)	$FFT(N)\,2$ (row)	$FFT(N)\,1$ (col)	$FFT(N)\,2$ (col)	

The parallel line algorithm improves with task level pipeling by performing a 2D DFT's reduction stage in parallel with its data download, and its 1D DFT stage in parallel with another 2D DFT's data upload. The execution, pipeline, and delay times for this two-task algorithm are

$$T_{exec} = T_{delay} = \max(2CN^2, 2ALN^2) + \max(2LT(F_N), 2CN^2)$$

$$T_{pipe} = \max(CN^2, ALN^2) + \max(LT(F_N), CN^2).$$

A space time-diagram for this computation on an M PE machine is

	$T_{\max}$	$T(F_N)$	$T_{\max}$	CN^2	$T_{\max}$	CN^2	
I/O	$Down\,1$		$Down\,2$	$Up\,1$	$Down\,3$	$Up\,2$	$\cdots$
PE_i	$Red\,1$	$FFT(N)\,1$	$Red\,2$	$FFT(N)\,2$	$Red\,3$	$FFT\,3$	

where $T_{\max} = \max(AN^2, CN^2)$. If $ALN^2 \leq CN^2$ and $LT(F_N) \leq CN^2$ a result is available at every $2CN^2$ interval. This is the time performance limit for single I/O channel machines. Figure 5.6 gives a summary of the parallel line algorithm's performance on an M PE tree.

1. On machines with no concurrency between communication and processing, the execution time can be

$$T_{exec} = 2CN^2 + AN^2 + T(F_N).$$

2. On machines where concurrency exists between communication and processing the execution time can be

$$T_{exec} = \max(AN^2, CN^2) + T(F_N) + CN^2.$$

3. On machines with concurrent communication and processing, and task level pipelining, the execution, delay, and pipeline times can be

$$T_{exec} = T_{delay} = \max(2CN^2, 2ALN^2) + \max(2LT(F_N), 2CN^2)$$

$$T_{pipe} = \max(CN^2, ALN^2) + \max(LT(F_N), CN^2).$$

Figure 5.6: Performance summary. Tree implementation of the prime case of the 2D line algorithm.

Conclusion

This section demonstrated a multiprocessor algorithm for MD DFT computation. The algorithm is based on a direct mapping of the hyperplane algorithm to the multiprocessor architecture. The 2D case of the hyperplane algorithm was studied extensively, and is identical to the line algorithm. The algorithm is shown to naturally partition the $d-$dimensional DFT into M independent computations, M equal to the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array.

For multiprocessor architectures a mapping is given that requires no interprocessor communication, and allows the M independent computations to occur concurrently with the input download. On single I/O channel machines that are capable of exploiting the full degree of parallelism of the algorithm, execution times as low as the time to compute a single one dimensional FFT on N points, plus the time to upload and download the data are attainable. If task level pipelining is used, average times equal to the single channel I/O limit occur, but single task completion times are longer. Furthermore, if the input data is real, the only stage of the algorithm that inputs complex data is the data upload.

The primary benefit of the algorithm was seen to be that it requires no interprocessor communication. The primary limitation of the algorithm is that it does not scale flexibly to the degree of parallelism of the target processor. This issue is addressed in the next section.

5.3 A Composite Algorithm

This section combines the hyperplane and multidimensional Cooley-Tukey algorithms to produce a highly scalable broadcast mode multiprocessor algorithm. The main features of the algorithm are that it requires no interprocessor communication, and that it maps to machines of varying degrees of parallelism.

5.3.1 Overview

In the previous sections of this chapter the $k-$dimensional hyperplane algorithm was mapped to a broadcast mode multiprocessor machine model. The parallel algorithm that resulted requires no interprocessor communication. However, in order to exploit that benefit, the number of processors in the machine has to equal the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional output array. For many cases this can be a large number. Consider that the power of prime case (including 2) of the algorithm requires the degree of parallelism of the machine to be

$$\left(\frac{P^n}{P}\right)^{d-k}\left(\frac{P^{d-k+1}-1}{P-1}\right).$$

The granularity of the corresponding computation is a single
$k-$dimensional DFT of size $(P^n)^k$.

For a given problem size the only means available for adjusting the degree of parallelism is to alter k. The effect this has on the algorithm is to change the dimension of the hyperplanes used to cover the output array. An increase in k increases the granularity of the computation performed at each PE by increasing the dimension of the required DFTs. This increase in granularity is accompanied by an assoicated decrease in the degree of parallelism required by the algorithm.

Using this method, the tradeoff between granularity and degree of parallelism is limited to powers of P^n. That is, for an increase in dimension of 1, the size of the computation increases P^n times and the degree of parallelism decreases roughly P^n times. At the limit $k = d - 1$, this method allows a minimum degree of parallelism

of

$$(P^n)\left(\frac{P+1}{P}\right).$$

The corresponding granularity of the computation performed at each PE is a $(P^n)^{d-1}$ DFT. For the $N = 2^n$ case, the minimum degree of parallelism is $3N/2$, and the corresponding granularity is a single $d-1$ dimension DFT of size $(N)^{d-1}$.

The primary limitations that result from manipulating only the dimensionality of the problem are: (1) that the degree of parallelism can only be scaled by powers of P^n, and (2) that there are only $d-1$ possible mappings of the algorithm.

One method for obtaining increased control over the degree of parallelism of the algorithm is to modify the order (size) of the computation in each dimension. This can be done using multidimensional Cooley-Tukey (MD CT) / vector-radix techniques. The feature of the MD CT algorithms central to this work is that they can factor a $d-$dimensional DFT into stages of lower order $d-$dimensional DFT.

In the next section an algorithm that marries the hyperplane algorithm to the MD CT algorithm is introduced. The resulting composite algorithm can be said to use the MD CT algorithm to reduce the order of the computations performed in each dimension, and the hyperplane algorithm to reduce the number of dimensions. The composite algorithm requires no interprocessor communication, and can be scaled to match the degree of parallelism of the target machine.

5.3.2 Derivation

The composite algorithm will be developed as follows: First the structure of the MD CT factorization is reviewed. The mechanism used to describe the structure is the tensor product of matrices. A tensor product formulation of the MD CT factorization is given in section 3.3. Then a nesting of the hyperplane algorithm within the MD CT factorization is given that permits the MD DFT to be computed with a reduced degree of parallelism and no interprocessor communication.

MD CT Components

The computational structure of the MD CT factorization is investigated below. The matrix formulation of this factorization (section 3.3) is

$$F_{\underline{n}} = Q_{\underline{n},\underline{r}} \overbrace{(F_{\underline{s}} \otimes I_r)}^{Stage\,2} T_{\underline{n},\underline{s}} \overbrace{(I_s \otimes F_{\underline{r}})}^{Stage\,1} P_{\underline{n},\underline{s}} \tag{5.9}$$

$$Q_{\underline{n},\underline{r}} = P_{\underline{n},\underline{r}}^{-1} P_{n,r},$$

where $F_{\underline{n}}$ is defined as the matrix form of the $n_1 \times \cdots \times n_d$ $d-$dimensional DFT, and

$$\begin{array}{rclcrcl} \underline{n} & = & (n_1, n_2, \ldots, n_d) & & n & = & n_1 \cdots n_d \\ \underline{r} & = & (r_1, r_2, \ldots, r_d) & , & r & = & r_1 \cdots r_d \\ \underline{s} & = & (s_1, s_2, \ldots, s_d) & & s & = & s_1 \cdots s_d \end{array} \quad , \text{ and } \quad \begin{array}{c} n_i = r_i s_i \\ i = 1, 2, \ldots d. \end{array}$$

The algorithm developed in this section will apply the hyperplane algorithm to the computation of the DFTs of the first stage of equation (5.9), and more conventional methods to the DFTs of the second stage. Given this, the first issue to consider is the dataflow between the DFT stages.

The first stage of lower order MD DFTs is

$$\underline{y}_1 = (I_s \otimes F_{\underline{r}}) P_{\underline{n},\underline{s}}\, \underline{x} = \begin{pmatrix} F_{\underline{r}} & & & \\ & F_{\underline{r}} & & \\ & & \ddots & \\ & & & F_{\underline{r}} \end{pmatrix} P_{\underline{n},\underline{s}}\, \underline{x}. \tag{5.10}$$

This stage is evaluated by computing a number s of independent $d-$dimensional DFT $F_{\underline{r}}$. The results of this computation are stored lexicographically in $\underline{y}_1$. The next computation applies the twiddle multiplications to the vector $\underline{y}_1$. While these have no bearing on the current development, they are given for completeness as

$$\underline{z} = T_{\underline{n},\underline{s}}\, \underline{y}_1,$$

where $T_{\underline{n},\underline{s}}$ was defined in section 3.2. The second stage of lower order DFTs is described by

$$\underline{y}_3 = (F_{\underline{s}} \otimes I_r)\, \underline{z}. \tag{5.11}$$

Equation (5.11) can be rewriten using the commutation property of stride permutations (theorem 2.1.1) as

$$\underline{y}_3 = P_{n,s}(I_r \otimes F_{\underline{s}})P_{n,r}\,\underline{z} = P_{n,s}\begin{pmatrix} F_{\underline{s}} & & & \\ & F_{\underline{s}} & & \\ & & \ddots & \\ & & & F_{\underline{s}} \end{pmatrix} P_{n,r}\,\underline{z}. \tag{5.12}$$

Expressed in this way, the computation of stage two is seen to be performed by a number r of independent $d-$dimensional DFT $F_{\underline{s}}$. In order to determine which elements of $\underline{z}$ (stage one) are being transformed by any $F_{\underline{s}}$, we may neglect the twiddle multiplications and expand (5.12) to

$$\underline{y}_3 = P_{n,s}\overbrace{\begin{pmatrix} F_{\underline{s}} & & & \\ & F_{\underline{s}} & & \\ & & \ddots & \\ & & & F_{\underline{s}} \end{pmatrix}}^{Stage\,2} P_{n,r}\overbrace{\begin{pmatrix} F_{\underline{r}} \}\ r-points & & & \\ & F_{\underline{r}} & & \\ & & \ddots & \\ & & & F_{\underline{r}} \end{pmatrix}}^{Stage\,1} P_{\underline{n},\underline{s}}\,\underline{x}. \tag{5.13}$$

Consider the output of any stage one DFT. Let $\underline{x}_j$ and $\underline{z}_j$ be the input and output vectors respectively of the j^{th} DFT of stage one. That is, the j^{th} $F_{\underline{r}}$ of the block diagonal matrix of stage one. Then $\underline{z}_j$ is given by

$$\underline{z}_j = \begin{pmatrix} z_j(0) \\ z_j(1) \\ \vdots \\ z_j(r-1) \end{pmatrix} = F_{\underline{r}}\,\underline{x}_j. \tag{5.14}$$

The stage one output vector $\underline{z}$ is formed by concatenating the $\underline{z}_j$ lexicographically to obtain

$$\underline{z} = \begin{pmatrix} \underline{z}_0 \\ \underline{z}_1 \\ \vdots \\ \underline{z}_{s-1} \end{pmatrix} = (I_s \otimes F_{\underline{r}})P_{\underline{n},\underline{s}}\,\underline{x}. \tag{5.15}$$

The input vector to the second stage of DFT is formed by applying the stride by r permutation to the stage one output $\underline{z}$.

Recall from section 2.1.1 that the action of the stride by r permutation $P_{n,r}$ on a vector $\underline{z}$ is to reorder it by multiples of r. As an example of this consider the vector $\underline{z}$ of length $n = rs$ acted on by $P_{n,r}$

$$P_{n,r}\,\underline{z} = (z_0, z_r, \ldots, z_{(s-1)r}, z_1, z_{r+1}, \ldots, z_{(s-1)r+1}, \ldots, z_{n-1})^t.$$

Denote the input vector to stage two by $\underline{z}'$. Represented in terms of the stage one output, $\underline{z}'$ can be written:

$$\underline{z}' = P_{n,r}\,\underline{z} = \begin{pmatrix} z_0(0) \\ z_1(0) \\ \vdots \\ z_{s-1}(0) \\ \vdots \\ z_0(r-1) \\ z_1(r-1) \\ \vdots \\ z_{s-1}(r-1) \end{pmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} s - points \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \tag{5.16}$$

where $\underline{z}_j(i)$ is defined in equation (5.14) as the i^{th} output of the j^{th} DFT of stage one. Let $\underline{z}'_i$ be the segment of $\underline{z}'$ that is input to the i^{th} DFT of stage two. The inputs to the first DFT of stage two are the first s elements of $\underline{z}'$ in equation (5.16). These elements will be denoted $\underline{z}'_0$. In terms of the inputs to the stage two DFT, $\underline{z}'$ is written

$$\underline{z}' = \begin{pmatrix} \underline{z}'_0 \\ \underline{z}'_1 \\ \vdots \\ \underline{z}'_{r-1} \end{pmatrix}. \tag{5.17}$$

$\underline{z}'_i$ is the i^{th} block of s elements of the vector $\underline{z}'$, and is given by

$$\underline{z}_i = \begin{pmatrix} z_0(i) \\ z_1(i) \\ \vdots \\ z_{s-1}(i) \end{pmatrix}. \tag{5.18}$$

The inputs to the i^{th} DFT ($F_{\underline{s}}$) of stage two are given by $\underline{z}'_i$. Equation (5.18) shows that these points are formed by collecting output i from each of the s stage one DFT $F_{\underline{r}}$.

In order to illustrate this point further, consider the 2D $n \times n$ DFT where $n = rs$. A block diagram for this case is shown in figure 5.3.2.

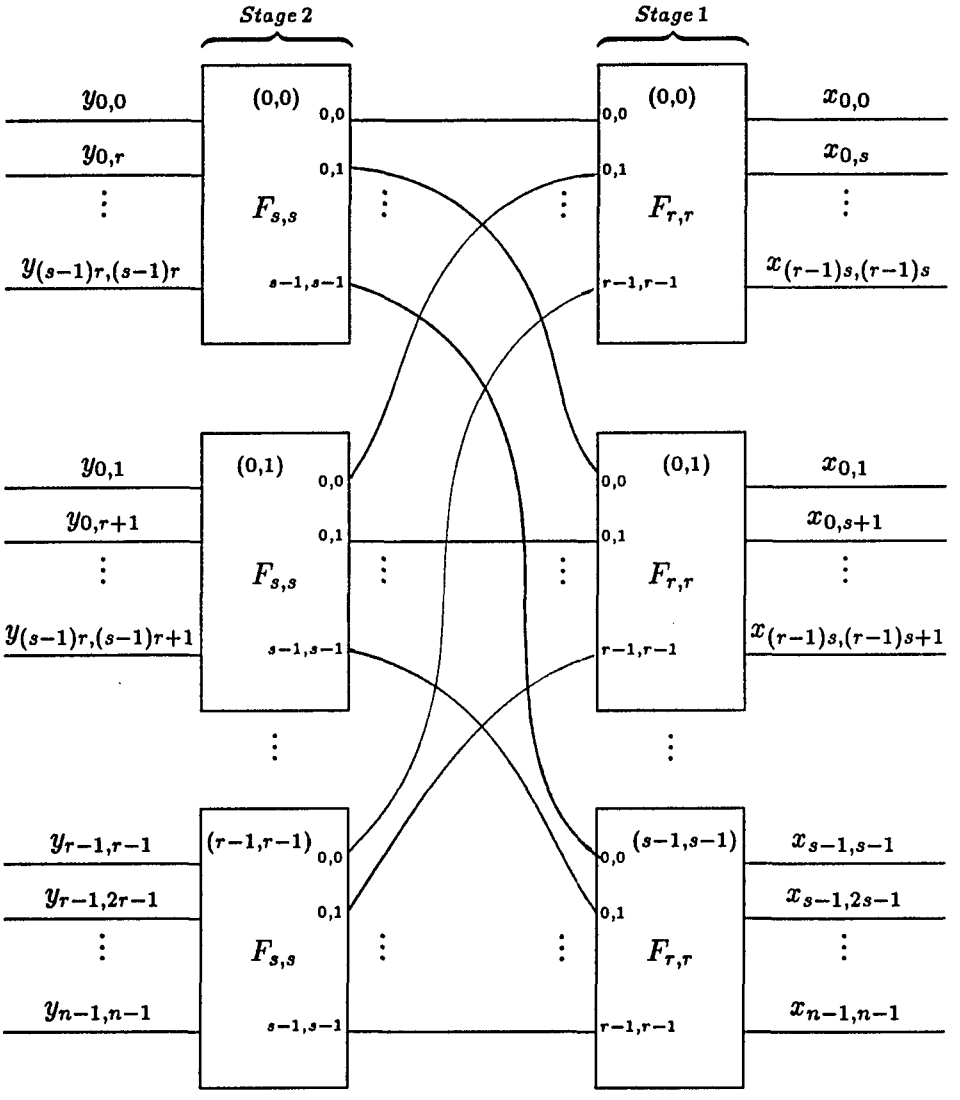


Figure 5.7: Flow diagram of an $n \times n$ 2D FFT factorization for $n = rs$.

The diagram describes the relationship between the first and second DFT stages of a MD CT factorization for $n = rs$. In the diagram, each DFT of a stage is

identified by a rectangle with an index (x, y) afixed to it. DFT i of stage two refers to the DFT whose index (x, y) is the i^{th} taken lexicographically. The diagram shows graphically that input j to DFT i of stage two is from output i of DFT j of stage one.

For an alternative perspective consider the three step procedure given in section 3.3 for the evaluation of the factorization of (5.9). Using this procedure, the data flow between stages one and two can is defined in terms of the decomposition of the input and output index sets. The procedure is restated below.

1. Computation of $d-$dimensional DFTs $F_{\underline{r}}$. Each of these is performed on a coset of the input with respect to the subgroup $\underline{s}Z/\underline{n}$.

2. Twiddle multiplications. The element $\underline{a}$ of the input coset $\underline{u}$ that was transformed in step (1) is multiplied by $\omega_{n_1}^{a_1 u_1} \cdots \omega_{n_d}^{a_d u_d}$.

3. Computation of $d-$dimensional DFTs $F_{\underline{s}}$. Each of these produces a coset of output with respect to the subgroup $\underline{r}Z/\underline{n}$. The input and output subgroups are dual.

This procedure relates directly to the diagram of figure 5.3.2. The DFTs of stage one of the figure correspond to the $F_{\underline{r}}$ of step one of the procedure, and the DFTs of stage two of the figure correspond to step three. Step two of the procedure defines the twiddle multiplications, which are not our immediate concern. The diagram shows the input points to each DFT of stage one. The DFT labeled $(0, 0)$ is seen to be computed on the points whose indexes are given by the set

$$\{(x, y) : (x, y) \in sZ/n \times sZ/n\}.$$

Similarly, the DFT labeled $(0, 1)$ is computed on the points whose indexes are in the set

$$\{(x, y) + (0, 1) : (x, y) \in sZ/n \times sZ/n\}.$$

By the definitions given in chapter 3, the input data for the DFT labeled $(0, 0)$ are the points whose indexes are the subgroup $\underline{s}Z/\underline{n}$ of $Z/\underline{n}$. The input data to the DFT

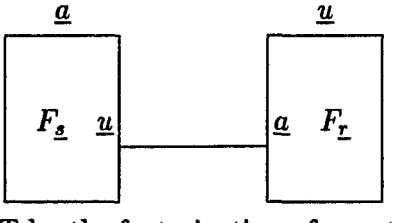
labeled $(0,1)$ are the points whose indexes are the $(0,1)$ coset of the decomposition of $Z/\underline{n}$ with respect to the subgroup $\underline{s}Z/\underline{n}$. In general the input to the DFT labeled (x,y) is coset (x,y) of the decomposition. The cosets of the decomposition of $Z/\underline{n}$ with respect to $\underline{s}Z/\underline{n}$ can be enumerated by $\underline{u} \in Z/s \times Z/s$. In this way $F_{\underline{r}}$ number $\underline{u}$ is said to transform coset $\underline{u}$ of the input array.

The second stage of DFT can be described in a similar way. The difference between the stages is that the DFTs of stage two each produce a coset of the output array with respect to the subgroup $rZ/n \times rZ/n$. The cosets in this decomposition are enumerated by $\underline{a} \in Z/r \times Z/r$. In this way $F_{\underline{s}}$ number $\underline{a}$ produces the coset $\underline{a}$ of the output array. These are the output points whose indexes are given by

$$\{(x,y) + (a_1,a_2) : (x,y) \in rZ/n \times rZ/n\}.$$

Above, the input and output data flow has been described in terms of the decomposition of the input and output index sets. Below, the data flow between DFT stages is described in terms of those same decompositions.

The output of each $F_{\underline{r}}$ of stage one is associated with the index set $\underline{a} \in Z/r \times Z/r$. These correspond to the indexes marked on the left column of the stage one DFT in the diagram. Similarly, the input to each DFT of stage two is associated with the index set $\underline{u} \in Z/s \times Z/s$. These correspond to the indexes marked on the right column of the DFT of stage two. Using these associations, the data flow between the DFT stages can be described in terms of the decompositions of the input and output index sets. Recall from the discussion above that an $F_{\underline{s}}$ of stage two is labeled by $\underline{a} \in Z/\underline{r}$, and an $F_{\underline{r}}$ is labeled by $\underline{u} \in Z/\underline{s}$. Then the $F_{\underline{s}}$ associated with coset $\underline{a}$ of the output array takes its input $\underline{u}$ from the output $\underline{a}$ of the $F_{\underline{r}}$ associated with input coset $\underline{u}$. This relation is depicted graphically below.



Evaluating the MD DFT by the factorization of equation (5.9) requires computing DFT stages composed of $F_{\underline{r}}$ and $F_{\underline{s}}$. The inputs to the i^{th} $F_{\underline{s}}$ were shown to

be the collection of i^{th} outputs of every $F_{\underline{r}}$. Let that $F_{\underline{s}}$ be assigned to a PE in a multiprocessor. The PE can evaluate that $F_{\underline{s}}$ if the i^{th} point from every stage one $F_{\underline{r}}$ is available at the PE. Below, an assignment of the hyperplane algorithm to the DFTs of stage one is given that satisfies this criterion.

Hyperplane Components

The multiprocessor hyperplane algorithm developed in the previous sections of this chapter will be applied to the computation of the DFTs of the first stage of the MD CT factorization. These are

$$\underline{y}_1 = (I_s \otimes F_{\underline{r}})P_{\underline{n},\underline{s}}\,\underline{x} = \overbrace{\begin{pmatrix} F_{\underline{r}} & & & \\ & F_{\underline{r}} & & \\ & & \ddots & \\ & & & F_{\underline{r}} \end{pmatrix}}^{Stage\,1} P_{\underline{n},\underline{s}}\,\underline{x}. \tag{5.19}$$

The algorithm depends on the computation of the $F_{\underline{r}}$ of stage one by the multiprocessor hyperplane method. Before proceeding with the development of the algorithm, first consider the computation of a single $F_{\underline{r}}$ by that method.

Assume that $F_{\underline{r}}$ is of the same size in $d-k+1$ dimensions, and let $r_k = \cdots = r_d$. Then the indexing set of $F_{\underline{r}}$ may be written

$$\begin{aligned} \mathcal{R} &= Z/\underline{r} = Z/r_1 \times \cdots \times Z/r_d = \mathcal{A} \times \mathcal{B}, \\ \mathcal{A} &= Z/r_1 \times \cdots \times Z/r_{k-1} \times Z/r_k, \\ \mathcal{B} &= (Z/r_k)^{d-k}. \end{aligned}$$

The number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array $\mathcal{R} = Z/\underline{r}$ is denoted $M(\underline{r})$. Let the number of PEs in the machine be $M(\underline{r})$, and recall that on such a machine the DFT $F_{\underline{r}}$ may be computed by the procedure listed below.

1. Assign one hyperplane of the covering set to each PE.

2. Broadcast the inputs to the machine, and simultaneously compute the reduction operations.

3. Compute a single $k-$dimensional DFT over $\mathcal{A}$.

Figure: Computation of $F_{\underline{r}}$ at a PE.

After this procedure is complete each PE has its own $k-$dimensional hyperplane of the outputs of the $F_{\underline{r}}$. For each of the remaining $F_{\underline{r}}$ of stage one, apply the same assignment of hyperplanes to PEs that was used for the first $F_{\underline{r}}$. These DFTs are computed in the same manner as the first.

When the computation of this stage is complete, every PE has one $k-$dimensional hyperplane from each $F_{\underline{r}}$ of stage one. Since each PE was assigned the same hyperplane in every $F_{\underline{r}}$, the PE that has point $\underline{a}$ from the first $F_{\underline{r}}$ has point $\underline{a}$ from every $F_{\underline{r}}$ of stage one. In the previous section these were shown to be the points needed to compute coset $\underline{a}$ of the output partition. The PE generates coset $\underline{a}$ of the output partition by applying the appropriate twiddle multiplications to the stage one outputs, then computing a single $d-$dimensional DFT $F_{\underline{s}}$.

Below an example is given to illustrate this procedure. The example is for a 9×9 2D DFT computed on a four PE broadcast mode multiprocessor. Following the example a general statement of the algorithm is given.

Example: Consider the problem of computing a 9×9 2D DFT on a 4 processor machine. A direct mapping of the line algorithm ($k = 1$ case of the hyperplane algorithm) for this computation would require 12 processors. However, the composite algorithm using a 3×3 factorization requires only 4 processors. This algorithm will be developed below.

The computation depends on the evaluation of the 3×3 2D DFT $F_{3,3}$, by the line algorithm. That computation is

$$\underline{z} = F_{3,3}\,\underline{x},$$

and is described by the example at the end of section 2.2. The output array of the 3×3 2D DFT is given by four lines. These are:

$$\begin{aligned} L((0,1)) &= \{z_{0,0}\,,\, z_{0,1}\,,\, z_{0,2}\} \\ L((1,1)) &= \{z_{0,0}\,,\, z_{1,1}\,,\, z_{2,2}\} \\ L((2,1)) &= \{z_{0,0}\,,\, z_{2,1}\,,\, z_{1,2}\} \\ L((1,0)) &= \{z_{0,0}\,,\, z_{1,0}\,,\, z_{2,0}\} \end{aligned}$$

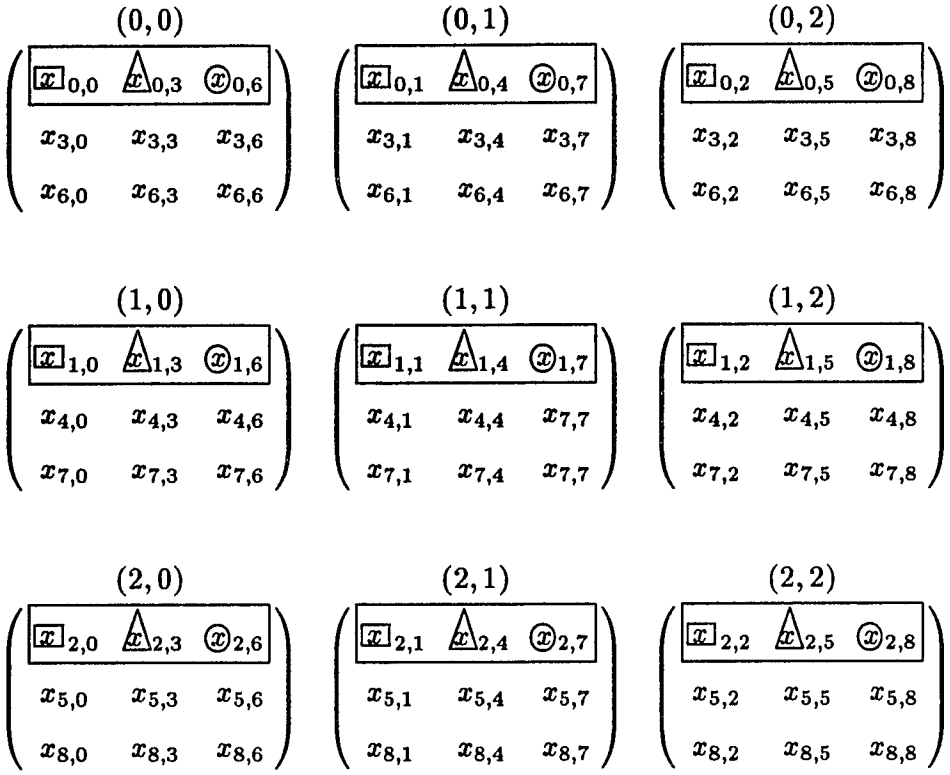
Each of these lines is assigned to a different processor. Let the output on the lines $L((m,1))$, $m = 0,1,2$ be computed by PE_m, and the output on the line $L((1,0))$ be computed by PE_3. As the inputs are broadcast to the machine, each PE computes the reduction operation $a^{(x,y)}$ associated with its assigned line of the output. After the broadcast is complete, each PE performs a 3−point DFT to produce its set of output points. Each of the 3×3 2D DFT of the first DFT stage of the factorization are computed in this way.

The 9×9 2D DFT can be written

$$F_{(9,9)}\,\underline{x} = P^{-1}_{(9,9)(3,3)} \overbrace{\left(I_9 \otimes F_{(3,3)}\right)}^{Stage\,2} P_{81,9}\, T_{(9,9)(3,3)} \overbrace{\left(I_9 \otimes F_{(3,3)}\right)}^{Stage\,1} P_{(9,9)(3,3)}\,\underline{x}$$

The first stage of DFT of this factorization requires the computation of nine 2D 3×3 DFT. The inputs to these DFT are listed in the arrays below. Each of the

arrays is labeled above by the index of its first element in parenthesis.



This partition of the input elements results from decimating both dimensions of the array simultaneously. Each array of the partition is a coset of $Z/9 \times Z/9$ with respect to the subgroup $3Z/9 \times 3Z/9$. The label above each array is its coset leader. Throughout this example, the arrays will be defined and refered to as cosets, and distinguished by their coset leaders.

Each of the nine required 3 × 3 2D DFT of stage one are computed by the multiprocessor line algorithm as described above. The lines are taken over each of the cosets. The assignment of lines to PEs is the same for each of the cosets. The PE assigned line $L((0,1))$ in coset $(0,0)$ is assigned line $L((0,1))$ in every coset.

For simplicity of presentation, allow the input data to be broadcast to the PEs by coset. Then the reduction operations are performed simultaneously with the data download. Each PE performs one addition for each word it inputs. As all the points of a coset are received, the associated reduction operation is completed. When the download is finished the PE performs nine independent 3−point 1D DFT to complete stage one.

Consider the computation of the line $L((0,1))$ in each coset of the input partition. For each coset downloaded the PE computes the required reduction operation. When the download is complete, the PE performs the 1D DFT. At this time the PE contains all the stage one outputs corresponding to the first row of each coset shown above (enlosed in rectangles). The approriate twiddle multiplications are applied at this time.

The second stage of DFTs requires computing nine 3×3 2D DFT. To obtain all output points by the method described in this section, each PE must compute three 3×3 2D DFT. For the PE assigned line $L((0,1))$ of each stage one coset, one 2D DFT is computed for each element in the line. That is, the first 2D DFT is performed on all points enclosed in squares, the second is computed on all points enclosed in triangles, and the third 2D DFT is computed on all points enclosed in circles. After this stage of 2D DFT is complete, the entire computation is distributed by output coset throughout the machine. This permutation is similar to the permutation that results from a typical application of the vector-radix algorithm.

□

The factorization of the 9×9 2D DFT given in the example above contains 18 $F_{3,3}$ 2D DFT. If each of these were evaluated by the row-column method there would be 108 FFTs of 3−points each. Assume the reduction operations of the line algorithm are performed concurrently with the input download and the communication time is not considered, then the computational speedup of the composite algorithm of the example relative to a single processor computing the 3×3 factorization is

$$Speedup = \frac{108\, FFT(3)}{27\, FFT(3)} = 4,$$

which is the ideal linear speedup, and requires no interprocessor communication.

The procedure listed below describes a best fit strategy for matching the degree of parallelism of the algorithm to the number of processors in the target machine.

1. The computation is an $n_1 \times \cdots \times n_d$ $d-$dimensional DFT $F_{\underline{n}}$, where $\underline{n} = (n_1, \ldots, n_d)$.

2. The initial degrees of parallelism attainable by the direct mapping of the algo-

rithm are given by $M_k(\underline{n})$, the number of $k-$dimensional hyperplanes required to cover the $d-$dimensional array. These exist for $k = 1, \ldots, d-1$ and are constrained by the condition that $\underline{n}$ be of the same size in at least $d-k+1$ dimensions. The algorithm has only been defined for the cases where $\underline{n}$ is of prime or power of prime (including 2) size in those dimensions.

3. Lower degrees of parallelism are attained by factoring $F_{\underline{n}}$ into stages of $F_{\underline{s}}$ and $F_{\underline{r}}$, where it is required that

$$\begin{array}{rcl} \underline{r} & = & (r_1, \ldots, r_d) \\ \underline{s} & = & (s_1, \ldots, s_d) \end{array} \text{, and } \begin{array}{c} n_i = r_i s_i \\ i = 1, \ldots, d \end{array}$$

4. The degrees of parallelism possible for a given factorization of $F_{\underline{n}}$ into $F_{\underline{s}}$ and $F_{\underline{r}}$ are given by $M_k(\underline{r})$ for $k = 1, \ldots, d-1$. $M_k(\underline{r})$ is the number of $k-$dimensional hyperplanes required to cover $Z/\underline{r}$. The constraints on $\underline{r}$ are identical to those given for $\underline{n}$ in step (2).

5. Different degrees of parallelism result for various factorizations of $F_{\underline{n}}$ into $F_{\underline{s}}$ and $F_{\underline{r}}$ and the dimensionality k of the covering hyperplanes. The selection of best fit can be based on which gives the closest match to the degree of parallelism of the machine, as well as which gives the best speedup.

The multiprocessor computation of $F_{\underline{n}}$ was described in previous sections of this chapter. The procedure that follows describes the multiprocessor computation of $F_{\underline{n}}$ factored into stages of $F_{\underline{s}}$ and $F_{\underline{r}}$.

1. Associate one $k-$dimensional hyperplane of $Z/\underline{r}$ with each PE. The required hyperplanes are given by theorems 4.2.1 and 4.3.1. A hyperplane represents a subset of the output of a single $F_{\underline{r}}$.

2. The input is partitioned into cosets with respect to the subgroup $\underline{s}Z/\underline{n}$. These are enumerated by $\underline{u} \in Z/\underline{s}$. Each corresponds to an array $Z/\underline{r}$. Assume for simplicity of presentation that the input has been ordered by coset.

3. Broadcast the input to the machine. As coset $\underline{u}$ is input, each PE computes the reduction operation a^H corresponding to the hyperplane assignment of step

(1). For each coset, this step is identical to the multiprocessor computation of a single $F_{\underline{r}}$. When this step is complete, s independent reduction operations a^H have been computed by each PE.

4. Compute a number s of $k-$dimensional DFT at each PE. Each $k-$dimensional DFT is computed on the output of a reduction operation from step (3). Each produces the hyperplane of the coset associated with the PE in step (1).

5. Apply the twiddle multiplications.

6. Compute independent DFTs $F_{\underline{s}}$ at each PE. There are s $k-$dimensional hyperplanes in each PE. Each hyperplane has $|\mathcal{A}|$ points. $|\mathcal{A}|$ DFTs are performed. For $\underline{x} \in \mathcal{A}$, a DFT $F_{\underline{s}}$ is performed on the collection of all points with index $\underline{x}$ from each hyperplane in the PE. After this step the computation is complete.

7. Data upload. The output is distributed among the PEs according to the coset decomposition of $Z/\underline{n}$ with respect to the subgroup $\underline{r}Z/\underline{n}$. This is a normal vector-radix [26] permutation. The cosets are enumerated by $\underline{a} \in Z/\underline{r}$. If $\underline{a} \in Z/\underline{r}$ is contained in the $k-$dimensional hyperplane associated with the PE in step (1), then output coset $\underline{a}$ is also contained in the PE. Cosets of output are redundant exactly the same way that points are redundant in the computation of a single $F_{\underline{r}}$. When it is desireable, the redundancy can be eliminated in an analougous manner.

5.4 Conclusion

This chapter presented a new algorithm for the multiprocessor computation of the MD DFT. The algorithm is based on the hyperplane algorithm introduced in chapter 4, and is intended for multiprocessor machines connected to a single I/O channel. The target processor must support the communication functions broadcast and report. The algorithm is refered to as the multiprocessor hyperplane algorithm.

Implementations of the algorithm are considered for two basic situations. The first is when the degree of parallelism of the target processor matches one of $M_k(\underline{n})$, $k = 1, \ldots, d-1$. Where $M_k(\underline{n})$ is the number of $k-$dimensional hyperplanes required to cover $Z/\underline{n}$. For these cases the algorithm is shown to require the time to compute one $k-$dimensional DFT plus the time to input and output the data. Computationally, the speedup of the algorithm is the ratio of a $d-$dimensional DFT to a single $k-$dimensional DFT.

The second case of the algorithm applies when the size of the machine does not match $M_k(\underline{n})$. That is, the degree of parallelism of the algorithm is not compatible with the size of the machine. For this case the hyperplane algorithm is married to the multidimensional Cooley-Tukey algorithm (MD CT). The role of the MD CT algorithm in the composite algorithm is to factor $F_{\underline{n}}$ into stages of $F_{\underline{s}}$ and $F_{\underline{r}}$, where each n_i is the composite $n_i = r_i s_i$. For this case the degree of parallelism is one of $M_k(\underline{r})$, for $k = 1, \ldots, d-1$. In this manner the degree of parallelism of the computation can be modified to match the size of the target processor. Like the direct method, the composite algorithm requires no interprocessor communication.

Chapter 6

Conclusions and Future Research

One of the major issues of parallel processing is the design of algorithms that minimize interprocessor communication. This research has addressed that issue from the perspective of the parallel computation of the multidimensional discrete Fourier transform.

The majority of algorithms for the parallel computation of the multidimensional DFT are based on row-column and Cooley-Tukey type methods. These algorithms have complex data flow and computational structure. To facilitate the design of such algorithms a tensor product formulation for MD DFT has been proposed in this work. The formulation is based on the fact that these algorithms decompose the input and output groups with respect to dual subgroups. Using the tensor formulation, MD CT type algorithms can be created and modified by applying algebraic manipulations to the formulation. In future research, this work can be extended to other multidimensional computations. These include the multidimensional cosine transform, linear convolution, and Hadamard transforms.

This dissertation has also developed a new reduced transform algorithm. The algorithm computes a $d-$dimensional DFT by restricting it to a collection of subgroups of its output indexing set. These subgroups are defined in terms of $k-$dimensional hyperplanes ($k < d$). The result of restricting the $d-$dimensional DFT to the $k-$dimensional hyperplanes is shown to be a set of independent $k-$dimensional DFT. Each $k-$dimensional DFT is performed on data derived from the input

by additions, and produces a $k-$dimensional hyperplane of the output array. A direction for future research could be the application of this method to other digital signal processing computations. A likely candidate for such an approach is the multidimensional discrete cosine transform.

A common characteristic of row-column and CT based parallel algorithms is that they intersperse stages of computation with stages of global data exchange. The central feature of most implementations of these algorithms is the manner in which interprocessor communication is minimized.

This dissertation has examined reduced transform algorithms as an alternative means of computation. Some reduced transform algorithms were shown to have structures that do not intersperse communication and computation stages. They are computable by a preaddition stage followed by a stage of independent lower order DFTs. The hyperplane algorithm described above falls into this category. These algorithms can be mapped to broadcast mode multiprocessors, and compute the MD DFT with no interprocessor communication.

The primary objective of this research has been the development of a broadcast mode multiprocessor algorithm for MD DFT computation. The main features of that algorithm are that it requires no interprocessor communication, and that it maps flexibly to the degree of parallelism of the target processor.

Two variants of the algorithm are developed. The first is a direct mapping of the hyperplane algorithm to the multiprocessor. This method is shown to produce a parallel algorithm that requires no interprocessor communication, but imposes limiting restrictions on the degree of parallelism of the target processor. In order to obtain a multiprocessor algorithm with a flexible degree of parallelism, the hyperplane algorithm is married to the multidimensional Cooley-Tukey algorithm. The resulting composite algorithm requires no interprocessor communications and has a flexible degree of parallelism.

Expressions for estimating the performance of the algorithm have been obtained as a function of the size of the $d-$dimensional DFT, the bandwidth C of the communications channel, the time A for an addition, the time $T(FFT)$ for a single

processing element to perform a k-dimensional DFT, and the degree of parallelism of the machine. For single I/O channel machines capable of exploiting the full degree of parallelism of the algorithm, computation times as low as the input/output limit are attainable.

An immediate direction for future work based on this research is the development of codes to realize the algorithm on an assortment of machines. These can be used for performance benchmarking to determine practical applications where the algorithm provides a feasible solution for MD DFT computation. Another research direction is the development of like minded algorithms for the computation of other DSP functions targeted for the same class of distributed processors.

Bibliography

[1] M. An, I. Gertner, M. Rofheart, and R. Tolimieri. *Discrete Fast Fourier Transforms, A Tutorial.* Advances in Applied Physics, To Appear July 1991.

[2] *AT&T WE DSP32 and DSP32C Reference Manual.* AT&T, Whippany, NJ, 1989.

[3] *AT&T DSP Parallel Processor BT-100.* AT&T, Whippany, NJ, 1988.

[4] G. Amdahl. *The validity of the single processor approach to achieving large scale computing capabilities.* AFIPS Conference Proceedings, Joint Computing Conference, Spring, 1967. pp. 483–485.

[5] L. Auslander, E. Feig, and S. Winograd. *New algorithms for the multidimensional discrete Fourier Transform.* IEEE Transactions Acoustics, Speech, Signal Processing, ASSP-31(2), April 1983, pp. 388–403.

[6] D. H. Bailey. *FFTs in external and hierarchical memory.* The Journal of Supercomputing. Vol. 4, 1990, pp. 23–25.

[7] K. E. Batcher. *The flip network in Staran.* Int. Conf. Par. Proc., Aug. 1976, pp. 65–71.

[8] K. E. Batcher. *Design of a massively parallel processor.* IEEE Transactions on Computers, Sept 1980, pp. 836–840.

[9] G. D. Bergland. *A parallel implementation of the fast Fourier transform algorithm.* IEEE Transactions on Computers, C-21(4), April 1972, pp. 366-370.

[10] R. E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Mass., 1987.

[11] R. E. Blahut. *Theory and Practice of Error Correcting Codes.* Addison-Wesley, Reading, Mass., 1983.

[12] *BBN Butterfly Parallel Processing Computer.* Bolt Bernak and Newman Inc., Cambridge Ma, 1986.

[13] Sarah A. Browning. *The Tree Machine: A Highly Concurrent Computing Environment.* PhD thesis, California Institute of Technology, Pasedena CA, January 1980.

[14] C. Chakrabarti and T. JáJá. *A family of optimal architectures for multidimensional transforms.* Proceedings of the 26th Annual Conference on Communications, Control, and Computing, Monticello ,IL., Sept. 1988, pp. 1015–1024.

[15] N. U. Chowdary and W. Steenart. *A high speed two dimensional FFT processor.* ICASSP, San Diego Ca, 1984, pp. 4.11.1–4.11.4.

[16] J. W. Cooley and J. W. Tukey. *An algorithm for the machine calculation of complex Fourier series.* Math. Comput., Vol. 19, Apr 1965, pp. 297–301.

[17] D. E. Dudgeon and R. M. Mersereau. *Multidimensional Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, N.J., 1984.

[18] T. Feng. *Data manipulating functions in parallel processors and their implementations.* IEEE Transactions Computers, C-23, Mar 1974, pp. 309–318.

[19] M. J. Flynn. *Very high speed computing systems.* Proc. IEEE, 54(12), Dec. 1966, pp. 1901–1909.

[20] Izidor Gertner. *A new efficient algorithm to compute the two-dimensional discrete Fourier transform.* IEEE Transactions Acoustics, Speech, Signal Processing, ASSP-36(7), July 1988, pp. 1036–1050.

[21] I. Gertner and M. Shamash. *VLSI architectures for multidimensional Fourier transform processing.* IEEE Transactions on Computers, C-36(11), Nov. 1987, pp. 1265–1274.

[22] I. Gertner and R. Tolimieri. *Fast algorithms to compute multidimensional discrete Fourier transform.* SPIE Real-Time Signal Processing Proceedings, San Diego Ca., August 1989, pp. 132–146.

[23] I. J. Good. *The relationship between two fast Fourier transforms.* IEEE Transactions on Computers, C-20, March 1971, pp. 310–317.

[24] A. L. Gorin, L. Auslander, and A. Silberger. *Balanced computation of 2D transforms on a tree machine.* To Appear in Applied Mathematics Letters.

[25] John A. Granata. *The Design of Discrete Fourier Transform and Convolution Algorithms for Risc Architectures.* PhD thesis, City University of New York, September 1990.

[26] D. B. Harris, J. H. McClellan, D. S. Chan, and H. W. Schuessler. *Vector radix fast Fourier transform.* IEEE ICASSP, Hartford, Conn. May 1977, pp. 548–551.

[27] R. W. Hockney and C. R. Jesshope. *Parallel Computers.* IOP Publishing, Bristol, England. 1989.

[28] Wayne Scott Hornick. *The Mesh of Trees Architecture for Parallel Computation.* PhD thesis, University of Illinois at Urbana-Champaign, January 1989.

[29] E. A. Hoyer, W. R. Berry. *An algorithm for the two dimensional Fourier transform.* IEEE ICASSP, Hartford, Conn. May 1977, pp. 552–555.

[30] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984.

[31] *iPSC/2 and iPSC/860 User's Guide.* Intel Corporation, Beaverton Oregon, Order number 311532-006, June 1990.

[32] L. H. Jamieson, P. T. Mueller, and H. J. Siegel. *FFT algorithms for SIMD processing.* J. Par. & Dist. Comp., Vol. 3, 1986, pp. 48–71.

[33] C. R. Jesshope. *The implementation of fast radix 2 transforms on array processors.* IEEE Transactions on Computers, C-29(1), Jan 1980, pp. 20–27.

[34] J. Johnson, R. Johnson, D. Rodriguez, R. Tolimieri. *A methodology for designing modifying and implementing Fourier transform algorithms on various architectures.* IEEE Transactions on Circuits and Systems, To appear 1990.

[35] H. T. Kung. *Why systolic architectures.* Computer, January 1982, pp. 271-292.

[36] S. Y. Kung. *VLSI Array Processors.* IEEE ASSP Magazine, July 1985, pp. 4–22.

[37] D. H. Lawrie. *Access and alignment of data in an array processor.* IEEE Transactions on Computers, C-24, Dec 1975, pp. 1145–1155.

[38] R. M. Mersereau, T. C. Speake. *A unified treatment of Cooley-Tukey algorithms for the evaluation of the multidimensional DFT.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-29(5), Oct. 1981, pp. 1011–1018.

[39] V. Milutinovic, A. B. Fortes, and L. H. Jamieson. *A multiprocessor architecture for real-time computation of a class of DFT algorithms.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-34, Oct. 1986,pp. 1301–1309.

[40] *NCUBE Programming Handbook.* NCUBE Corporation, Beaverton Oregon, 1987.

[41] *NCUBE 6400 Processor Handbook.* NCUBE Corporation, Beaverton Oregon, 1989.

[42] L. M. Napolitano Jr. and G. R. Redinbo. *A comment on the efficiency of "A new efficient algorithm to compute the discrete two-dimensional Fourier transform".* IEEE Transactions Acoustics, Speech, Signal Processing. To Appear 1991.

[43] A. Norton and A. Silberger. *Parallelization and performance analysis of the Cooley-Tukey FFT algorithm on shared-memory architectures.* IEEE Transactions on Computers, C-36(5), May 1987, pp. 581–591.

[44] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, New York, 1982 (Second Edition).

[45] H. Nussbaumer, P. Quandalle. *Fast computation of discrete Fourier transforms using polynomial transforms.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-27, April 1979, pp. 169–181.

[46] A. Oppenheim. *Applications of Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1978.

[47] M. C. Pease *An adaption of the fast Fourier transform for parallel processing.* Journal of the ACM, Vol. 15, April 1968, pp. 252–264.

[48] M. Porat. *Position-Spatial Frequency Relationship and the Visual Relationship.* PhD thesis, Technion-Israel Institute of Technology, 1987.

[49] F. Preparata and J. Vuillemin. *The Cube-Connected-Cycles: a versatile network for parallel computation.* Communications of the ACM, Vol. 4(5), May 1981, pp. 300-309.

[50] G. E. Rivard. *Direct fast Fourier transform of bivariate functions,* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-25, 1977, pp. 300-309.

[51] Domingo Rodriguez. *On Tensor Products Formulations of Additive Fast Fourier Transform Algorithms and Their Implementations.* PhD thesis, City University of New York, January 1988.

[52] M. Rofheart and I. Gertner. it A parallel algorithm for 2D DFT computation with no interprocessor communication. IEEE Transactions on Parallel & Distributed Systems, Vol. 1(3), July 1990, pp. 377–382.

[53] M. Rofheart, I. Gertner, and R. Tolimieri. *A parallel reduced transform algorithm to compute $P^N \times P^N$ 2D DFT with no interprocessor communications.* SIAM Conference on Parallel Processing for Scientific Computing, Houston Texas, March 1991. D. C. Sorenson, ed.

[54] M. Rofheart, J. Granata, and M. Conner. *The Tensor Product as a Tool for Designing Efficient Fourier Transform Algorithms for Digital Signal Processors.* In Revision.

[55] H. J. Seigel. *Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks.* IEEE Transactions on Computers, Vol. C-26, Feb 1977, pp. 153–161.

[56] H. J. Seigel and R. J. McMillen. *The multistage cube: a versatile interconnection network.* Computer, Vol. 14, Dec. 1981, pp. 65–76.

[57] H. J. Seigel and R. J. McMillen. *Using the augmented data manipulator network in PASM.* Computer, Vol. 14, Feb. 1981, pp. 25–33.

[58] C. L. Seitz. *Concurrent VLSI architectures.* IEEE Transactions on Computers, Vol. C-33(12), Dec. 1984, pp. 1247–1264.

[59] C. L. Seitz. The Cosmic Cube. Communications of the ACM, Vol. 28, Jan. 1985, pp. 22–33.

[60] Moshe Shamash. *VLSI Architectures for Multidimensional Digital Signal Processing.* PhD thesis, Technion-Israel Institute of Technology, November 1989.

[61] C. D. Thompson. *Fourier transforms in VLSI.* IEEE Transactions on Computers, Vol. C-32, Dec. 1983, pp. 1047–1057.

[62] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Carnegie Mellon University. August 1980.

[63] Richard Tolimieri, Myoung An, and Chao Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, New York, 1989.

[64] Dwen-Ren Tsai and Michael Vulis. *Computing discrete Fourier transform on a rectangular data array.* IEEE Transactions Acoustics, Speech, Signal Processing. Vol. ASSP-38(2), Feb. 1990, pp. 271–276.

[65] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, Rockville, MD, 1984.

[66] B. D. Van Veen, K. M. Buckley. *Beamforming: A versatile approach to spatial filtering.* IEEE ASSP Magazine, April 1988, pp. 4–24.

[67] Michael Vulis. *The weighted redundancy transform.* IEEE Transactions Acoustics, Speech, Signal Processing. ASSP-37(11), Nov. 1989, pp. 1687–1692.

[68] D. S. Wise. *Compact layouts of Banyan/FFT networks.* VLSI Systems and Computations, Computer Science Press, Rockville Md., 1981. pp. 186–195.

[69] Hong Ren Wu and Frank John Paoloni. *The structure of vector radix fast Fourier transforms.* IEEE Transactions Acoustics, Speech, Signal Processing. Vol. ASSP-37(9), Sept. 1989, pp. 1415–1424.

[70] E. L. Zapata, F. F. Rivera, I. Benavides, J. M. Carazo, R. Peskin. *Multidimensional fast Fourier transform into SIMD hypercubes.* Proc. IEE, Vol. 137(4), July, 1990, pp. 253–260.