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 9207098

New algorithms for convolutions and FFTs

Liu, Hongyi, Ph.D.

City University of New York, 1991

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

A

NEW ALGORITHMS FOR CONVOLUTIONS AND FFTs

HONGYI LIU

A dissertation submitted to the Graduate Faculty in Computer Science 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 Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

6/18/91
Date

Richard Palmer
Chair of Examining Committee

6/18/91
Date

T C Wendelken
Executive Officer

James W. Cooley

Michael Conner

Michael Anshel

Octavio Betancourt

Supervisory Committee

The City University of New York

Abstract

NEW ALGORITHMS FOR CONVOLUTIONS AND FFTs

Hongyi Liu

Adviser: Professor Richard Tolimieri

The FFT algorithm with prime size is a key to develop the general FFT algorithms. The basic tool of computing the prime size FFT is the use of the cyclic convolution algorithm. Unfortunately, in many cases, the traditional algorithms for cyclic convolution are not very effcient and the computational structures with these algorithms are not flexible for the parallel implementation on different architectures.

By introducing the new concept *half-cyclic convolution*, the cyclic convolution can be treated as a special case of the new concept. Some *half-cyclic convolution* algorithms with general case and some special cases have been designed. The new algorithms for cyclic convolution with different purposes then can be developed, which can be very efficient and well-structured for both sequential and parallel processing. In particular, the prime case FFT algorithms can be improved a lot by using these new cyclic convolution algorithms. In addition, the algorithms for multi-dimensional FFT and some special computations such as symmetrized FFT can also be improved or redesigned. Furthermore, most of the algorithms for convolution and FFT could be benifited from this development.

Contents

CHAPTER 1

INTRODUCTION

1.1 Introduction

The computation of Discrete Fourier Transform (DFT) [1,2] has become more and more important in the application world [3], applied to such things as weather forecasting, image processing and so on. All of them require that the computation be as fast as possible, which could be carried out by the development in both hardware and software. In hardware, fast computers and parallel computers with various architectures should be built up; in software, new algorithms for DFT should be developed, which, usually, may deal with the Fast Fourier Transform (FFT) [3,4]. Two basic efforts can be made on the algorithm design. The first effort is to design fast algorithms for sequential execution, usually, this is concerned with small size. As result, a few efficient libraries for different purposes could be established, which, as sequential subroutines, can also be used in parallel environment. The second effort is to develop parallel algorithms. These, in general, will deal with large size. The algorithms should have very good parallel features so that they can be efficiently implemented on the machines with different architectures. The books [6] by R. Tolimieri and the book [5] by R. Blahut cover most of the algorithms developed so far. Some of the details can also be found in [7,8,9].

The algorithms for prime size are very important for any dimensional FFTs. The non-prime size algorithms can be built up by multiplicative algorithms using the prime case algorithms as building blocks. Any development for the prime case can benefit the non-prime case also. In this thesis, we focus on the prime case FFTs. As we know, there is a very close relation between the computations for the FFT and for cyclic convolution, which is further evidenced in this thesis.

1.2 FFT And Cyclic Convolution

The n-point cyclic convolution [10] is defined as

$$\underline{y} = C\underline{x}, \tag{1}$$

where $\underline{x}$ and $\underline{y}$ are the input and output vectors of order n, and the computational matrix C is circulant [11],

$$C = \begin{pmatrix} a_0 & a_{n-1} & \cdots & a_1 \\ a_1 & a_0 & \cdots & a_2 \\ \vdots & & & \vdots \\ a_{n-1} & a_{n-2} & \cdots & a_0 \end{pmatrix}. \tag{2}$$

One of the relationships between the cyclic convolution and DFT is described by the the Convolution theorem [6],

$$C = F(n)DF(n)^{-1}, \tag{3}$$

where $F(n)$ is the n-point Fourier transform and

$$D = F(n)^{-1}CF(n) \tag{4}$$

is a diagonal matrix. Which means that the n-point cyclic convolution can be carried out by n-point Fourier transform. On the other hand, the p-point Fourier transform can be designed by using p'-point cyclic convolution algorithm, where p is a prime number and $p' = p - 1$.

By the knowledge of linear algebra [12], for a prime number p, a generator z of Z/p can be found such that the indices of input and output can be reordered to

$$0, 1, z, z^2, \cdots, z^{p-2}. \tag{5}$$

Then, by the Rader's algorithm [13],

$$F(p) = P_\pi^{-1} F_\pi P_\pi, \tag{6}$$

where P_π is the permutation matrix corresponding to the ordering (5). And

$$F_\pi = (1 \oplus C(p))A(p), \tag{7}$$

where

$$A(p) = \begin{pmatrix} 1 & 1_{p'}^t \\ -1_{p'} & I_{p'} \end{pmatrix}. \tag{8}$$

$1_{p'}$ means a $p' - tuple$ matrix of all 1s and $I_{p'}$ is $p' \times p'$ identity matrix. And

$$C(p) = \left[v^{z^{j+k}}\right]_{0 \le j,k < p-1}; \quad v = -exp(2\pi i/p) \tag{9}$$

is a $p' \times p'$ skew-circulant matrix and is called the Winograd core.

A circulant matrix C can be changed into a skew-circulant matrix H by

$$H = CP_c = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} \\ a_1 & a_2 & \cdots & a_{n-2} \\ \vdots & & & \vdots \\ a_{n-1} & a_0 & \cdots & a_{n-2} \end{pmatrix}, \tag{10}$$

where P_c is the permutation matrix

$$P_c = \begin{pmatrix} 1 & & & \\ & \ddots & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix}, \tag{11}$$

and also,

$$P_c = P_c^{-1}; \qquad P_c^2 = I_n. \tag{12}$$

Obviously, an algorithm for computing either C or H can be applied to each other. Since

$$F(n)^{-1} P_c = F(n), \tag{13}$$

the variant form of the Convolution theorem in (3) with skew-circulant matrix becomes

$$H = F(n) D F(n), \tag{14}$$

where

$$D = F(n)^{-1} H F(n)^{-1}. \tag{15}$$

In this thesis, we introduce some new algorithms for prime case 2-dimensional and symmetrized FFTs. It turns out that since the most of computational matrices are also skew-circulant, the algorithms have to be designed based on the convolution algorithms.

As we know, the Winograd cyclic convolution algorithm [6] is very efficient for small size. But when the size grows the Winograd algorithm quickly becomes more complicated, the entries of the pre-addition and the post-addition matrices are no longer small integers, and the number of additions increases rapidly. For large size, either a multiplicative algorithm or the Convolution theorem has to be used.

Now consider the Agarwal-Cooley algorithm [14], a multiplicative algorithm for cyclic convolution. For $n = s \cdot r$, where s and r are relative prime, by the Chinese Remainder Theorem (CRT), a permutation P_e can be found such that the matrix

$$C_e = P_e^{-1} C P_e^{-1} = \begin{pmatrix} h_0 & h_{s-1} & \cdots & h_1 \\ h_1 & h_0 & \cdots & h_2 \\ \vdots & & & \vdots \\ h_{s-1} & h_0 & \cdots & h_{s-2} \end{pmatrix} \tag{16}$$

becomes a $s \times s$ block circulant matrix and each h_i, $0 \leq i < s$, is a $r \times r$ circulant matrix. Then we can carry out the computation by nesting s-point cyclic convolution inside r-point cyclic convolution.

To use this algorithm on the skew-circulant matrix $C(p)$ we have to use the permutation (10) in the corresponding place. In addition, $C(p)$ is complex, and after the permutations, each h_i is also complex. As we know, for any size except 2 and 3, the Winograd algorithm is not efficient than the Convolution theorem on complex matrices. In general, for the purpose of computing the Winograd core $C(p)$, the Agarwal-Cooley algorithm can't take the advantage of the Winograd algorithm, and the Convolution theorem in (3) with sizes s and r has to be used. We see that the formula (3) uses both $F(n)$ and $F(n)^{-1}$, which is not so convenient. To change this by formula (13), more permutations would be involved. Even if the algorithm provides parallel structure, the data flow can't be well-organized. In this way, the algorithms for cyclic convolution and prime case FFT seem to be dead-locked. It follows that by developing new algorithm for cyclic convolution, new algorithm for prime FFT can be easily obtained and the non-prime case FFT can also be benefited. The Convolution theorem, finally, feeds the gains back to the cyclic convolution.

The following properties of $C(p)$ of the Winograd core $C(p)$ are helpful, where p is a prime number.

(i). Because

$$\sum_{i=0}^{p'} v^i = 0,$$

then

$$\sum_{i=0}^{p'-1} a_i = \sum_{i=0}^{p'-1} v^{z^i} = -1; \qquad v = -exp(2\pi i/p). \tag{17}$$

(ii). The size $p' = p - 1$ is an even number except where $p = 2$. Then p' can be written as the form

$$p' = 2^k \cdot r; \qquad k > 0, \tag{18}$$

where r is an odd number.

(iii). Because

$$z^{\frac{p'}{2}} = -1,$$

then the entry

$$a_{i+\frac{p'}{2}} = v^{z^{i+\frac{p'}{2}}} = v^{-z^i} = a_i^*, \tag{19}$$

which we will call the *Conjugate property* throughout this text.

The property (i) has been used to obtain the algorithm (7), and later it is used further to reduce the computation of $A(p)$ in (8). The size property suggests that the multiplicative algorithm is one of the keys to compute the $C(p)$. What the conjugate property interests us is that $a_i + a_i^*$ is a real number and $a_i - a_i^*$ is a pure imaginary. Unfortunately, by the Agarwal-Cooley algorithm, the conjugate property cannot be utilized because of the permutations.

We mentioned that the computational matrices in the new algorithms for prime 2-dimensional FFT and symmetrized FFT are also skew-circulant. In addition, we have proved that these skew-circulant matrices at least keep all the three properties of the Winograd core $C(p)$ in the above. Because we will mostly deal with the skew-circulant matrix in this thesis, whenever we mention the cyclic convolution we mean that the computational matrix is skew-circulant unless otherwise indicated.

1.3 Cyclic Convolution And Non-Cyclic Convolution

To use the conjugate property of $C(p)$, a basic idea is to block-diagonalize $C(p)$ by 2-point cyclic convolution algorithm without any permutation. $C(p)$ can be written as the block form

$$C(p) = \begin{pmatrix} h_0 & h_1 \\ h_1 & h_0 \end{pmatrix}. \tag{20}$$

$C(p)$ is 2×2 block skew-circulant, but each block h_i is not circulant. It

has the form

$$H^{\diamond} = \begin{pmatrix} b_0 & b_1 & \cdots & b_{r-1} \\ b_1 & b_2 & \cdots & b'_0 \\ \vdots & & & \vdots \\ b_{r-1} & b'_0 & \cdots & b'_{r-2} \end{pmatrix}, \tag{21}$$

which seems to be a computational matrix in between cyclic convolution and linear convolution. We will call $H^{\diamond}$ in (21) *Half-circulant matrix* and the corresponding computation defined by $H^{\diamond}$ will be called *Half-cyclic convolution* throughout this thesis.

More general, for $p' = s \cdot r$, where s is an even number, if we block-diagonalize $C(p)$ by s-point Winograd algorithm, the resulting blocks in the block-diagonal matrix are closed in the group of half-circulant matrix under the addition. Sometimes, There is a relationship between all of b_i and b'_i,

$$b'_i = \Delta b_i; \qquad \Delta \neq 0. \tag{22}$$

The matrix $H^{\diamond}$ now has the form

$$H^{\Delta} = \begin{pmatrix} b_0 & b_1 & \cdots & b_{r-1} \\ b_1 & b_2 & \cdots & \Delta b_0 \\ \vdots & & & \vdots \\ b_{r-1} & \Delta b_0 & \cdots & \Delta b_{r-2} \end{pmatrix}. \tag{23}$$

The matrix H^{Δ} in (23) is called Δ-*Half-circulant.* In (22), if $\Delta = 0$, then H^{Δ} becomes the matrix

$$H^{\circ} = \begin{pmatrix} b_0 & b_1 & \cdots & b_{r-1} \\ b_1 & b_2 & \cdots & 0 \\ \vdots & & & \vdots \\ b_{r-1} & 0 & \cdots & 0 \end{pmatrix}, \tag{24}$$

which is called *Zero-half-circulant.*

Obviously, the set of circulant matrices is only a subset of Δ-half-circulant matrices with $\Delta = 1$ and is also a subgroup of half-circulant matrices. However, the set of Δ-half-circulant matrices is a subset but not a subgroup of half-circulant matrices because the result under the addition could produce a matrix with $\Delta = 0$, which is not Δ-half-circulant by our definition. The set of zero-half-circulant matrices, is another subgroup of half-circulant matrices. For example, in (20), $h_0 + h_1$ is also skew-circulant, but $h_0 - h_1$ would be Δ-half-circulant,

where $\Delta = -1$. To develop new multiplicative algorithm without permutation, the algorithm for half-cyclic convolution must be designed. According to these new concepts, the algorithms for cyclic convolution should be included in the algorithms for half-cyclic convolution. Some algorithms for convolutions can be found in the references [15] to [21], but none of them uses the concepts we defined here.

1.4 About This Dissertation

In this thesis, we will use the words *Real-type* and *Complex-type* to distinguish the algorithms for convolutions. If the computational matrx is real, all the computation steps by the *Real-type* algorithm are in the real field, as with the Winograd algorithm; while by the *Complex-type* algorithm some computation steps may be changed into the complex field. For example, applying the Convolution theorem to a real circulant matrix H, the resulting diagonal matrix may be complex. However, this does not mean that the real-type algorithm is always efficient for computing the real matrix.

In the real-type algorithm, sometimes the pre-addition or the post-addition matrix contains the elements with the values of either 0 or ± 1. For convenience, we call this kind of matrix *Uni-valued* in this thesis. The Uni-valued matrix is easy to implement. In our new algorithms, most of the pre-addition and post-addition matrices are Uni-valued.

Sometimes, we design or apply an algorithm on a matrix in *block format*, which implies that each block is treated as a single element by the algorithm. The definitions about the general matrix can also be applied to the matrix in block format, for examples, *Block-circulant matrix*, *Block-half-circulant matrix* and so on. But we have to notice that a matrix with certain type may have a different type in block format, so does in the other way. For example, a block-circulant matrix itself may be not circulant. In this work, one of the techniques we frequently used is the transformation between the matrices and the transformation between the forms of a matrix. In general, all of the algorithms on a certain type matrix can also be applied to the matrx in block format with the same type. But, there are some algorithms on the matrix with special block format, which may not be applied to the general matrix. By developing algorithms on various type matrices or on different forms of block matrix, many problems in cyclic convolution and FFT can be solved. The terminologies *Block-diagonalize* or *Partially-diagonalize* are

usually used in such a situation. The resulting diagonal matrix is called *Block-diagonal matrix.*

This dissertation is organized into 13 chapters including this introduction. A brief summary for each chapter is presented in the following.

In chapter 2 we briefly describe some mathematics and notations used in this topic, such as Abstract algebra, special matrices and their operations. Some development and a few of new concepts are included.

In chapter 3, an important algorithm for half-cyclic convolution is discussed and some interesting properties of the algorithm are proved. The algorithm is very efficient and well-constructed. In chapter 4, two algorithms for computing zero-half-circulant matrix and another algorithm for half-cyclic convolution are also presented.

In chapter 5, similar to the Convolution theorem, an important relation between Δ-half-circulant matrix and circulant matrix has been found. By this theorem, a Δ-half-circulant matrix can be computed based on cyclic convolution algorithm, sometimes without extra cost. The multiplicative algorithms for Δ-half-circulant matrix are presented by combining the theorem with the algorithms for both cyclic and half-cyclic convolutions. In addition, another theorem on the block half-circulant matrix with special shift form is introduced.

In chapter 6 we present three new algorithms for the prime case cyclic convolution. One of them is built up based on half-cyclic convolution algorithm and 2-point cyclic convolution algorithm. The other two, called $(p\pm 1)$-*algorithm*, are built up by the algorithm for zero-half-circulant matrix and the p'-point cyclic convolution algorithm.

Some new multiplicative algorithms for cyclic convolution are developed in chapter 7 and chapter 8. First, a new algorithm for relative prime cyclic convolution is introduced, which is designed based on the theorem in chapter 6. The result is similar to the Agarwal-Cooley algorithm except that the permutation in this algorithm can be implemented in parallel also. For the non-relative prime case, applying both Convolution theorem and Δ-theorem, a complex-type algorithm is created without any permutation, therefore, it is good for parallel processing. Chapter 8 concerns with the designing of the real-type algorithms. In addition to the relative prime case, the algorithms for the cases of 2^k and $2^k \cdot r$ are developed, which are identical to the size of $C(p)$. The 2^k-*algorithm* is the same efficient as the Winograd algorithm but is well-structured, and the $2^k \cdot r$-*algorithm* is even faster than the relative prime

case algorithm. The conjugate property of $C(p)$ can be used by the two algorithms and no permutation is necessary. The improvement of performance can be obtained straightforwardly in two ways, one is the algorithm itself and the other is the use of the conjugate property of $C(p)$.

Based on the new algorithms for cyclic convolution, the new algorithms for prime case 1-dimensional FFT have been designed in chapter 9, for both sequential and parallel processing. A more efficient library can be established. The performance can be improved from 20% up to 50% than the traditional algorithms.

In chapter 10, new prime case algorithms for 2-dimensional FFT are illustrated. The computational matrices are all the same as the 1-dimensional prime case. In addition, from the point view of parallel processing, these algorithms provide more flexibility based on the new algorithms for cyclic convolution.

In chapters 11 and 12, the prime case algorithms for 2-dimensional symmetrized FFTs with 90°-rotation and 120°-rotation are discussed. The algorithms are designed based on Richard Tolimieri's works and some development has also been made. It turns out that the two kinds of symmetries have the same computational structure, which is also similar to the 1-dimensional prime case FFT. The gains can be obtained from both data symmetry and the new algorithms.

Finally, in chapter 13, some techniques used to implement these algorithms are mentioned. A procedure of minimizing the number of additions for Uni-valued matrix is briefly described. A few libraries will be briefly introduced and some future work will be proposed, but, which is not covered by this dissertation.

CHAPTER 2

SOME MATHEMATICS AND NOTATIONS USED IN THIS THESIS

2.1 Abstract Algebra

Some objects of abstract algebra have been used in this thesis. For example, the knowledge of *group, ring* and *field*; some special rings such as the *ring Z/n of integers mod n, the quotient polynomial ring* $F(x)/f(x)$ and the special *finite field* Z/p. In addition, the *Chinese Remainder theorem* (CRT) and the transformations between the algebraic structures such as *isomorphism* are also important tools. We do not describe them in detail. The details and some of the notations used in this thesis can be found in Tolimieri's book [6] or Blahut's book [5], some of them can also be found in [12]. As an exception, a theorem about the ring Z/p is proved here.

Theorem 2.1 For p a prime, denote by $U(p)$ the non-zero elements in the ring Z/p. If $a, b \in U(p)$, then for any positive integer m,

$$(a \pm b)^{p^m} = a \pm b. \tag{1}$$

Proof By the Theorem 2.24 of chapter 2 in [5], we have that

$$(a \pm b)^{p^m} = a^{p^m} \pm b^{p^m}. \tag{2}$$

Let z be a generator of Z/p. Because $a \neq 0$, then $a = z^k$ for some k, $0 \leq k < p-1$. Now we have

$$a^p = a \cdot a^{p-1} = a \cdot z^{k(p-1)} = a. \tag{3}$$

Furthermore,

$$a^{p^m} = (a^p)^{p^{m-1}} = a^{p^{m-1}} = a; \qquad b^{p^m} = b. \tag{4}$$

Replacing the result in (4) into (2), the theorem is proved.

This theorem will be used in chapter 10 to find a generator of a field. In addition, some special ring-structures will be introduced in chapter 9 for direct use.

2.2 Matrices

The matrix language has been frequently used in this text. Sometimes, a new algorithm could be created by using the operations and transformations between matrices. In addition to the general operations such as addition and multiplication, some special operations on matrices will be defined. These operations are very useful for representing the characters of matrices and the relations between matrices. Also, the definitions and the notations for some special matrices will be given. Because the tensor product and the permutation matrix as well as the circulant matrix are more important, they will be discussed in separate sections.

(i). A Few Operations

If A is a $n \times s$ matrix and B is a $n \times r$ matrix, the operation $A \vdash B$ is defined as

$$A \vdash B = (A \quad B), \tag{5}$$

which is a $n \times (s + r)$ matrix.

If A is a $s \times n$ matrix and B is a $r \times n$ matrix, the operation $A \perp B$ is defined as

$$A \perp B = \begin{pmatrix} A \\ B \end{pmatrix}, \tag{6}$$

the result is a $(s + r) \times n$ matrix.

The direct sum of matrices A and B is a block diagonal matrix

$$A \oplus B = \begin{pmatrix} A & \\ & B \end{pmatrix}, \tag{7}$$

and the operation

$$A \hat{\oplus} B = \begin{pmatrix} & A \\ B & \end{pmatrix}. \tag{8}$$

It is not difficult to prove that the following properties will hold for the operations $\vdash$ and $\perp$,

$$A(B \vdash C) = AB \vdash AC; \tag{9}$$

$$(A \perp B)C = AC \perp BC; \tag{10}$$

If A and C are $n \times s$ matrices, B and D are $n \times r$ matrices,

$$(A \vdash B) + (C \vdash D) = (A + C) \vdash (B + D); \tag{11}$$

If A and C are $s \times n$ matrices, B and D are $r \times n$ matrices,

$$(A \perp B) + (C \perp D) = (A + C) \perp (B + D). \tag{12}$$

A $m \times n$ matrix A can be written as

$$A = \mathop{\perp}_{i=0}^{m-1} \mathop{\vdash}_{j=0}^{n-1} a_{i,j} = \mathop{\vdash}_{j=0}^{n-1} \mathop{\perp}_{i=0}^{m-1} a_{i,j}. \tag{13}$$

The addition

$$A+B = \mathop{\perp}_{i=0}^{m-1} \mathop{\vdash}_{j=0}^{n-1} (a_{i,j}+b_{i,j}) = \mathop{\vdash}_{j=0}^{n-1} \mathop{\perp}_{i=0}^{m-1} (a_{i,j}+b_{i,j}), \tag{14}$$

and the multiplication of a $m \times s$ matrix A and a $s \times n$ matrix B, by the formulas (9) and (10)

$$AB = \mathop{\vdash}_{j=0}^{n-1} \mathop{\perp}_{i=0}^{m-1} \sum_{k=0}^{s-1} (a_{i,k} b_{k,j}). \tag{15}$$

Let $A = \vdash_{k=0}^{s-1} a_k$ and $B = \perp_{k=0}^{s-1} b_k$, which is the special case of (15) with $i = j = 1$, then we have that

$$AB = (\mathop{\vdash}_{k=0}^{s-1} a_k) \cdot (\mathop{\perp}_{k=0}^{s-1} b_k) = \sum_{k=0}^{s-1} a_k b_k. \tag{16}$$

In fact, formula (15) can be derived directly by the definition for multiplication with formula (13). Here we can see the use of these new operations.

(ii). Some Special Matrices And Their Notations

A n-tuple matrix of all 1s is denoted by 1_n,

$$1_n = \mathop{\perp}_{i=0}^{n-1} 1 = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}; \tag{17}$$

A n-tuple matrix of all 0s except that the first element is 1 is denoted by α_n,

$$\alpha_n = 1 \perp (\overset{n-2}{\underset{i=0}{\perp}} 0) = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}; \tag{18}$$

A $n \times n$ identity matrix is denoted by I_n,

$$I_n = \bigoplus_{i=0}^{n-1} 1; \tag{19}$$

The matrix of all 1s is denoted by $I(n)$,

$$I(n) = \overset{n-1}{\underset{i=0}{\perp}} \overset{n-1}{\underset{j=0}{\vdash}} 1; \tag{20}$$

The matrix

$$\widehat{I}_n = \begin{pmatrix} & & & 1 \\ & & 1 & \\ & \iddots & & \\ 1 & & & \end{pmatrix} = \widehat{\oplus}_{i=0}^{n-1} 1 \tag{21}$$

is a special skew-circulant matrix and has the following property

$$\widehat{I}_n^2 = I_n. \tag{22}$$

The transpose of a matrix A is denoted by A^t. Sometimes, we may use the vector formed by the first column (or the first row) to represent a skew-circulant matrix, the indices of the elements use 1-dimensional notation, i.e., a_i. In the above, all definitions, operations and properties dealing with single element are also true for block format if we replace the element by a corresponding block.

2.3 Tensor Products

If A is a $m \times n$ matrix with the entries $a_{i,j}$ and B is a $s \times r$ matrix, the tensor product [6,22] is defined by

$$A \otimes B = \begin{pmatrix} a_{0,0}B & a_{0,1}B & \cdots & a_{0,n-1}B \\ a_{1,0}B & a_{1,1}B & \cdots & a_{1,n-1}B \\ \vdots & & & \vdots \\ a_{m-1,0}B & a_{m-1,1}B & \cdots & a_{m-1,n-1}B \end{pmatrix}, \tag{23}$$

which is a $(ms)\times(nr)$ matrix. By (13), the matrix in (23) can be written as

$$A\otimes B=\mathop{\perp}_{i=0}^{m-1}\mathop{\vdash}_{j=0}^{n-1}a_{i,j}B=\mathop{\vdash}_{j=0}^{n-1}\mathop{\perp}_{i=0}^{m-1}a_{i,j}B. \tag{24}$$

The tensor product is *bilinear* in the following sense,

$$(A+B)\otimes C=A\otimes C+B\otimes C; \tag{25}$$

$$A\otimes(B+C)=A\otimes B+A\otimes C. \tag{26}$$

Also, by the definitions for the operations in the above, it is easy to prove that

$$(A\vdash B)\otimes C=(A\otimes C)\vdash(B\otimes C); \tag{27}$$

$$(A\perp B)\otimes C=(A\otimes C)\perp(B\otimes C); \tag{28}$$

$$(A\oplus B)\otimes C=(A\otimes C)\oplus(B\otimes C); \tag{29}$$

$$(A\hat{\oplus}B)\otimes C=(A\otimes C)\hat{\oplus}(B\otimes C). \tag{30}$$

Theorem 2.2 The associative law holds for tensor product,

$$(A\otimes B)\otimes C=A\otimes(B\otimes C). \tag{31}$$

It can be easily proved by the new definitions with some of the above properties.

Proof Using the formula (24) as the definition for $A\otimes B$ and then applying the properties (28) and (27), the proving procedure is listed below,

$$\begin{aligned}(A\otimes B)\otimes C&=(\mathop{\perp}_{i=0}^{m-1}\mathop{\vdash}_{j=0}^{n-1}a_{i,j}B)\otimes C\\&=\mathop{\perp}_{i=0}^{m-1}(\mathop{\vdash}_{j=0}^{n-1}a_{i,j}B)\otimes C\\&=\mathop{\perp}_{i=0}^{m-1}\mathop{\vdash}_{j=0}^{n-1}a_{i,j}(B\otimes C)\\&=A\otimes(B\otimes C).\end{aligned} \tag{32}$$

But it is not commutative. In general, $A \otimes B \neq B \otimes A$.

Theorem 2.3 If A and C are $m \times m$ matrices, and B and D are $n \times n$ matrices, then

$$(A \otimes B)(C \otimes D) = AC \otimes BD. \tag{33}$$

Proof We again use the new way to prove this. Using the definition (24) for both $A \otimes B$ and $C \otimes D$, doing the block multiplication based on formula (10), we have

$$\begin{aligned}
(A \otimes B)(C \otimes D) &= (\mathop{\perp}_{i=0}^{m-1} \mathop{\vdash}_{j=0}^{m-1} a_{i,j}B)(\mathop{\vdash}_{l=0}^{m-1} \mathop{\perp}_{k=0}^{m-1} c_{k,l}D) \\
&= \mathop{\perp}_{i=0}^{m-1} (\mathop{\vdash}_{j=0}^{m-1} a_{i,j}B)(\mathop{\vdash}_{l=0}^{m-1} \mathop{\perp}_{k=0}^{m-1} c_{k,l}D) \\
&= \mathop{\perp}_{i=0}^{m-1} \mathop{\vdash}_{l=0}^{m-1} (\mathop{\vdash}_{j=0}^{m-1} a_{i,j}B)(\mathop{\perp}_{k=0}^{m-1} c_{k,l}D) \\
&= \mathop{\perp}_{i=0}^{m-1} \mathop{\vdash}_{l=0}^{m-1} \sum_{j=0}^{m-1} (a_{i,j}c_{j,l}BD) \\
&= \mathop{\perp}_{i=0}^{m-1} \mathop{\vdash}_{l=0}^{m-1} (\sum_{j=0}^{m-1} a_{i,j}c_{j,l})(BD) \\
&= AC \otimes BD.
\end{aligned} \tag{34}$$

In fact, the property (33) holds for non-square matrices too.

Theorem 2.4 If A_0, B_0, C_0 and D_0 are $m \times s, s \times n, m_1 \times r$ and $r \times n_1$ matrices respectively, then

$$(A_0 \otimes B_0)(C_0 \otimes D_0) = A_0C_0 \otimes B_0D_0. \tag{35}$$

Proof It can be easily proved by replacing the size m with s or n at the corresponding places in the procedure (34). Therefore, the theorem (33) now is only a special case of (35).

A special application of (35) is that

$$(A \otimes B) = (A \otimes I_s)(I_n \otimes B) = (I_m \otimes B)(A \otimes I_r), \tag{36}$$

where A is a $m \times n$ matrix and B is a $s \times r$ matrix.

Theorem 2.5 If both A and B have inverses, then $A \otimes B$ has inverse, and

$$(A \otimes B)^{-1} = (A^{-1} \otimes B^{-1}). \tag{37}$$

Proof It can be proved by directly using theorem 2.3,

$$(A \otimes B)(A^{-1} \otimes B^{-1}) = (I_m \otimes I_n) = I_{mn}. \tag{38}$$

Now we can see that some special matrices in section 1. can be represented by using the tensor product. For example,

$$I(n) = 1_n \otimes 1_n^t = 1_n^t \otimes 1_n,$$

and if $n = sr$, then

$$1_n = 1_s \otimes 1_r; \quad I_n = I_s \otimes I_r; \quad \widehat{I}_n = \widehat{I}_s \otimes \widehat{I}_r; \quad \alpha_n = \alpha_s \otimes \alpha_r.$$

2.4 Permutations

A permutation functions as an isomorphism mapping between matrices, which can be used to change the computation structure. The matrix of implementing a permutation is called *permutation matrix*. The operations and transformations on permutation matrices may help us to reduce computation and may result in well-organized data flow for different purpose. Obviously, a permutation matrix is a square matrix and there is only one non-zero constant 1 in each column and in each row in a permutation matrix. In this section, we only introduce some properties and some special permutation matrices. Some new ideas are also included.

(i) Two Basic Properties

For two permutation matrices P_1 and P_2, a direct computation shows that the product $P = P_1 P_2$ is still permutation matrix.

The inverse of a permutation matrix P exists, and

$$P^{-1} = P^t, \tag{39}$$

which is also a permutation matrix and can be obtained by straightforward computation.

(ii). Circulant Shift Matrices

The name *Circulant shift matrix* means that it is a circulant permutation matrix. We will simply call it *Shift matrix*. The n-point shift matrix is defined as

$$S_n = \begin{pmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{pmatrix} = 1 \widehat{\oplus} I_{n-1}. \tag{40}$$

For a vector $A = \perp_{i=0}^{n-1} a_i$, the action of S_n on A is

$$S_n A = S_n(\mathop{\perp}_{i=0}^{n-1} a_i) = \mathop{\perp}_{i=0}^{n-1} a_{i+1}, \tag{41}$$

where $i+1$ takes modulo n. Obviously, the action of S_n^k on A is the result of replacing $i+1$ by $i+k$ in (41). Then we have

$$S_n^n = I_n. \tag{42}$$

The shift matrix is very useful for describing the relations between matrices and changing the structures of computation.

(iii). Stride Permutations

The *Stride permutation* is one of the powerful tools in representing parallel structures. The stride permutation itself can be implemented by either parallel or vector processing. Suppose that $n = rs$, the *n-point stride s permutation matrix* $P(n,s)$ can be defined as

$$P(n,s) = \mathop{\perp}_{i=0}^{r-1} (I_s \otimes \alpha_r^t) S_n^{-i} = \mathop{\vdash}_{i=0}^{r-1} S_n^i (I_s \otimes \alpha_r). \tag{43}$$

The action of $P(n,s)$ on the matrix $A = \mathop{\perp}_{i=0}^{n-1} a_i$ can be obtained by direct computation,

$$P(n,s)A = P(n,s)(\mathop{\perp}_{i=0}^{n-1} a_i) = \mathop{\perp}_{j=0}^{s-1} \mathop{\perp}_{k=0}^{m-1} a_{j+ks}. \tag{44}$$

An important property of the stride permutation is described as follows. The proving procedure can be found in the reference [6] by different method.

Theorem 2.6 If $n = srt$, then

$$P(n,s)P(n,r) = P(n,sr) = P(n,r)P(n,s). \tag{45}$$

If $t = 1$, a direct corollary can be derived. Because

$$P(n,s)P(n,r) = P(n,n) = I_n,$$

then

$$P(n,s) = P(n,r)^{-1}. \tag{46}$$

The stride permutation is defined on a factor of n, now we will extend the definition to the relative prime case. Suppose that $(n,s) = 1$, the permutation matrix of *n-point Stride s modulo n* can be defined as

$$P(n,s/n) = \mathop{\perp}_{i=0}^{n-1}(\alpha_n^t S_n^{-si}) = \mathop{\vdash}_{i=0}^{n-1}(S_n^{is^{-1}}\alpha_n). \tag{47}$$

The action of $P(n,s/n)$ on the matrix $A = \perp_{i=0}^{n-1} a_i$ can be obtained by direct computation,

$$P(n,s/n)A = P(n,s)(\mathop{\perp}_{i=0}^{n-1} a_i) = \mathop{\perp}_{j=0}^{n-1} a_{js}, \tag{48}$$

where js takes modulo n. Suppose that $(n,sr) = 1$, the property (45) holds for this case too.

Theorem 2.7 If $n = srt$, then

$$P(n,s/n)P(n,r/r) = P(n,sr/n). \tag{49}$$

Proof By (48), we have

$$\begin{aligned} P(n,r/n)P(n,s/n)A &= P(n,r/n)(\mathop{\perp}_{i=0}^{n-1} a_{js}) \\ &= \mathop{\perp}_{i=0}^{n-1} a_{jsr} \\ &= P(n,sr/n)A, \end{aligned} \tag{50}$$

which proved the theorem. A direct corollary can also be derived by (49). Because

$$P(n,s/n)P(n,s^{-1}/n) = P(n,1) = I_n,$$

we have that

$$P(n, s/n) = P(n, s^{-1}/n)^{-1}. \tag{51}$$

Theorem 2.8 If $n = sr$ and A, B are $s \times s$ and $r \times r$ matrices respectively, then

$$(A \otimes B) = P(n, s)(B \otimes A)P(n, r). \tag{52}$$

The proof can be found in [6]. Now we extend the above theorem to the case of non-square matrices.

Theorem 2.9 Suppose that A and B are $s_0 \times s_1$ and $r_0 \times r_1$ matrices respectively, and $n_0 = s_0 r_0$, $n_1 = s_1 r_1$, then

$$(A \otimes B) = P(n_0, s_0)(B \otimes A)P(n_1, r_1). \tag{53}$$

Proof Let $n = s_1 r_0$. By direct computation, we have that

$$(A \otimes I_{r_0}) = P(n_0, s_0)(I_{r_0} \otimes A)P(n, s_1) \tag{54}$$

and

$$(I_{s_1} \otimes B) = P(n, r_0)(B \otimes I_{s_1})P(n_1, r_1). \tag{55}$$

Then, by (36) and (45) we have

$$\begin{aligned}(A \otimes B) &= (A \otimes I_{r_0})(I_{s_1} \otimes B) \\ &= P(n_0, s_0)(I_{r_0} \otimes A)P(n, s_1)P(n, r_0)(B \otimes I_{s_1}) \\ &= P(n_0, s_0)(I_{r_0} \otimes A)(B \otimes I_{s_1})P(n_1, r_1) \\ &= P(n_0, s_0)(B \otimes A)P(n_1, r_1).\end{aligned} \tag{56}$$

Theorem 2.10 The transpose of the tensor product of matrices A and B

$$(A \otimes B)^t = (A^t \otimes B^t). \tag{57}$$

Proof Using the notations in theorem 9, by some properties we have proved in the above, we have

$$\begin{aligned}(A \otimes B)^t &= ((A \otimes I_{r_0})(I_{s_1} \otimes B))^t \\ &= (I_{s_1} \otimes B)^t (A \otimes I_{r_0})^t \\ &= (I_{s_1} \otimes B^t)(P(n_0, s_0)(I_{r_0} \otimes A)P(n_1, r_1))^t \\ &= (I_{s_1} \otimes B^t)(P(n_1, r_1)^t (I_{r_0} \otimes A)^t P(n_0, s_0)^t \\ &= (I_{s_1} \otimes B^t)P(n_1, s_1)(I_{r_0} \otimes A^t)P(n_0, r_0) \\ &= (I_{s_1} \otimes B^t)(A^t \otimes I_{r_0}) \\ &= (A^t \otimes B^t).\end{aligned} \tag{58}$$

By using the above properties and notations, some permutation matrices can be clearly represented for different purposes. For example the permutation P_c in (6) of chapter 1,

$$P_c = P(n,(n-1)/n) = 1 \oplus \widehat{I}_{n-1}. \tag{59}$$

In fact, the matrx

$$S_n^k \widehat{I}_n = \widehat{I}_k \oplus \widehat{I}_{n-k} \tag{60}$$

is also a special kind of permutation matrix.

2.5 Circulant Matrix And Skew-Circulant Matrix

Theorem 2.11 For two circulant matrices C_a and C_b, the product C_aC_b is also circulant.

Proof By the Convolution theorem,

$$C_a = F(n)D_aF(n)^{-1}; \quad C_b = F(n)D_bF(n)^{-1}. \tag{61}$$

Then

$$C_aC_b = F(n)D_aD_bF(n)^{-1} = F(n)DF(n)^{-1}, \tag{62}$$

where D is also a diagonal matrix, which implies that C_aC_b is also circulant.

Some corollaries can be derived from this theorem.

If H_a and H_b are both skew-circulant matrices, then H_aH_b is circulant. Because by the relation (11) in chapter 1, then we have

$$H_aH_b = H_aP_c^2H_b = (H_aP_c)(P_cH_b), \tag{63}$$

where both H_aP_c and P_cH_b are circulant matrices, so is H_aH_b.

If H_a is skew-circulant and C_b is circulant, then H_aC_b is skew-circulant. Because

$$H_aC_b = P_c^2H_aC_b = P_c((P_cH_a)C_b), \tag{64}$$

where $P_c H_a$ is circulant, so is $(P_c H_a)C_b$. Then $H_a C_b$ is skew-circulant. The result is also true for $C_b H_a$.

Because the shift matrix S^k is circulant, for a skew-circulant matrix H of the same size, $S^k H$ is also skew-circulant. Then we have

$$S^k H = (S^k H)^t = H^t (S^k)^t = H S^{-k}. \tag{65}$$

Because $\widehat{I}_n$ is skew-circulant, for a circulant matrix C of size n, $\widehat{I}_n C$ is also skew-circulant. Then

$$S^k C = S^k \widehat{I}_n \widehat{I}_n C = \widehat{I}_n S^{-k} (\widehat{I}_n C) = \widehat{I}_n (\widehat{I}_n C) S^k = C S^k. \tag{66}$$

In addition, for a symmetric matrix T, we have that

$$T^t = T. \tag{67}$$

If T can be factorized by

$$T = CGM, \tag{68}$$

then

$$T = T^t = M^t G^t C^t, \tag{69}$$

which is another factorization of T if $M^t \neq C$ and $C^t \neq M$. We know that the DFT matrix is symmetric. According to the definitions for skew-circulant and half-circulant matrix, all of them are symmetric, but the circulant matrix is not. Whenever we have a factorization for a symmetric matrix, another factorization can also be obtained by (69). In addition, if we prove some property on the factor matrix C (or M), the same property must be true for the matrix M^t (or C^t), which simplifies the proving procedure.

2.6 The Measurement of Performance

In the algorithm design and implementation, we often need to measure the performance or make comparisons between algorithms, which, usually, is quite complicated [23]. A traditional method is to use the arithmetic count, i.e., the numbers of multiplications and additions required by the algorithm. They are denoted by $M(n)$ and $A(n)$ for n-point algorithm, respectively. The arithmetic count can basically measure the performance for an algorithm with the implementation in sequential and in high level language. But, it may not give an accurate

measurement for parallel purpose. In this topic, we assume that the input data are always complex numbers. For the real-type algorithm, we usually suppose that both input the computation matrix are real. If the input is complex the arithmetic count can be simply doubled. For the complex-type algorithm, we may face the problem with the trade off between the complex operations and the real operations. A complex addition is implemented by two real additions, but a complex multiplication can be implemented in two ways. One is using 4 real multiplications and 2 real additions, the other is using 3 real multiplications and 3 real additions. We adopted the first approach in this text unless indicated.

Sometimes, we also need to make a trade off between the additions and multiplications, however, it is machine dependant. To do so, we take a parameter R in our evaluation,

$$R = \frac{t_m}{t_a}, \tag{70}$$

where t_m and t_a are the execution times of a real multiplication and a real addition on a given machine. The values of R for some machines are given in the last chapter. Now the performance for a n-point algorithm can be measured by

$$E(n) = A(n) + RM(n). \tag{71}$$

The comparisons can be made based on the values of $E(n)$.

The parameter for complex operation can be obtained by

$$R_c = \frac{2t_m + 4t_a}{2t_a} = R + 2. \tag{72}$$

Usually, the number of multiplications can be easily figured out from the expression of an algorithm, but, sometimes the number of additions is not obvoius, especially, the minimum number. In the last chapter, a procedure to find the minimum number of additions for Uni-valued matrix will be described, Most of our addition matrices are Uni-valued matrices. Denote by $A_m[B]$ the minimum number of additions for computing the addition matrix B and suppose that for any single addition matrix B the value of $A_m[B]$ is known, now the difficulty comes from the following fact, the pre-addition matrix or post-addition matrix of an algorithm, sometimes, is represented by several addition matrices with certain operations, such as addition, multiplication and especially, the

tensor product. For example, if $C = C_0C_1$, from the view of mathematics,

$$A_m[C] = A_m[C_0C_1], \tag{73}$$

but, it may be not convenient to implement C_0C_1 in one step, instead, two separate steps could be used. Denote by $A_i[C_0C_1]$ the number of additions required by this implementation. Then

$$A_i[C_0C_1] = A_m[C_0] + A_m[C_1]. \tag{74}$$

In general,

$$A_m[C_0C_1] \leq A_i[C_0C_1]. \tag{75}$$

If C is represented by the operations of a few matrices in the final version of an algorithm, we will adopt the formula (74) for our measurement, unless indicated.

If $C = C_0 + C_1$ and C is a $m \times n$ matrix, then

$$\begin{aligned} A_i[C] &= A_i[C_0 + C_1] \\ &= A_m[C_0] + A_m[C_1] + m. \end{aligned} \tag{76}$$

If $C = C_s \otimes C_r$, where C_s is a $s \times m_s$ matrix and C_r is a $r \times m_r$ matrix, the measurement for this case is a little complicated. In practical, by the experience in the implementation, the tensor product of $C_s \otimes C_r$ is either implemented by $(C_s \otimes I_r)(I_{m_s} \otimes C_r)$ or by $(I_s \otimes C_r)(C_s \otimes I_{m_r})$, which may result in different performance. Because

$$\begin{aligned} &A_i[(C_s \otimes I_r)(I_{m_s} \otimes C_r)] \\ &= A_m[C_s \otimes I_r] + A_m[I_{m_s} \otimes C_r] \\ &= rA_m[C_s] + m_sA_m[C_r], \end{aligned} \tag{77}$$

and

$$A_i[(I_s \otimes C_r)(C_s \otimes I_{m_r})] = m_rA_m[C_s] + sA_m[C_r], \tag{78}$$

in general,

$$rA_m[C_s] + m_sA_m[C_r] \neq m_rA_m[C_s] + sA_m[C_r]. \tag{79}$$

This is why the different orders of factorization may result in different implementations, therefore, the different performance. From the view of mathematics, to obtain the minimum number of additions for C by (77) and (78), we need to prove that

$$A_m[C] = MIN\{rA_m[C_s] + m_sA_m[C_r], m_rA_m[C_s] + sA_m[C_r]\}. \tag{80}$$

We did not give a proof for (80), but we have assumed that it is true in our evaluation.

CHAPTER 3

THE WINOGRAD-LIKE ALGORITHM FOR HALF-CYCLIC CONVOLUTION

3.1 Introduction

In chapter 1, we have introduced the new concepts *Half-cyclic convolution* and *Half-circulant matrix.* We will use the symbol $\diamond$ in denoting half-circulant matrix and the matrices in the corresponding factorizations as well as the arithmetic count in measurement. The n-point half-circulant matrix $H_n^\diamond$ is defined as

$$H_n^\diamond = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \\ a_1 & a_2 & \cdots & a_{n-1} & a_0' \\ \vdots & & & & \vdots \\ a_{n-2} & a_{n-1} & \cdots & a_{n-4}' & a_{n-3}' \\ a_{n-1} & a_0' & \cdots & a_{n-3}' & a_{n-2}' \end{pmatrix}, \tag{1}$$

and the computation

$$\underline{y} = H^\diamond \underline{x} \tag{2}$$

is called the *Half-cyclic convolution.* Where $\underline{x}$ and $\underline{y}$ represent the input and output vectors.

As we see, any submatrix of $H_n^\diamond$ is also half-circulant, and a submatrix of a skew-circulant matrix may be also half-circulant. The set of half-circulant matrices forms an additive group, which is closed under addition. In this sense, the large size algorithm for half-cyclic convolution can be build up by small size half-cyclic convolution algorithms, but the large size algorithm for cyclic convolution cannot be built up by only small size cyclic convolution algorithms, it has to combine the half-cyclic convolution algorithms. Therefore, the half-cyclic convolution is not only practical, but important. Unfortunately, so far there is no any general algorithm for half-cyclic convolution. In this chapter we will design a basic algorithm for half-cyclic convolution, which is called the *Winograd-like algorithm* in this text because it is similar to the *Winograd algorithm.* Another algorithm for half-cyclic convolution and some special techniques will be discussed in the following chapters.

3.2 The Base case of $n = 2$

When the size $n = 2$,

$$H_2^\diamond = \begin{pmatrix} a_0 & a_1 \\ a_1 & a_0' \end{pmatrix}, \tag{3}$$

the direct computation of $H_2^\diamond$ will use 4 multiplications and 2 additions. Now we describe a new method to compute $H_2^\diamond$. Let

$$H_2' = \begin{pmatrix} -a_1 & a_1 \\ a_1 & -a_1 \end{pmatrix} = a_1 \cdot \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4}$$

and set

$$H_2'' = H_2^\diamond - H_2' = \begin{pmatrix} a_0 + a_1 & 0 \\ 0 & a_0' + a_1 \end{pmatrix},$$

which is a diagonal matrix. Then

$$H_2^\diamond = H_2' + H_2'', \tag{5}$$

and it can be factorized as

$$H_2^\diamond = C_2^\diamond G_2 M_2^\diamond, \tag{6}$$

where

$$\begin{aligned} C_2^\diamond &= \begin{pmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}; \qquad M_2^\diamond = \begin{pmatrix} 1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}; \\ G_2 &= \begin{pmatrix} a_1 & & \\ & a_0 + a_1 & \\ & & a_0' + a_1 \end{pmatrix}. \end{aligned} \tag{7}$$

Now we see that the computation of $H_2^\diamond$ will use 3 multiplications and 3 additions, which is 1 multiplication less and 1 addition more than the direct computation and is 1 multiplication more and 1 addition less than the 2-point Winograd cyclic convolution algorithm. Notice that the matrix H_2' in (4) can also be set as

$$H_2' = \begin{pmatrix} a_1 & a_1 \\ a_1 & a_1 \end{pmatrix} \tag{8}$$

and

$$H_2'' = \begin{pmatrix} a_0 - a_1 & 0 \\ 0 & a_0' - a_1 \end{pmatrix}. \tag{9}$$

The corresponding matrices in the factorization (6) becomes

$$\begin{aligned} C_2^\diamond &= \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}; \qquad M_2^\diamond = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}; \\ G_2 &= \begin{pmatrix} a_1 & & \\ & a_0 - a_1 & \\ & & a_0' - a_1 \end{pmatrix}. \end{aligned} \tag{10}$$

The algorithm based on the factorization (6) is called *2-point Negative algorithm* and the second is called *2-point Positive algorithm.* The two ways have the same performance for the general case. In some applications, we may use either of them optionally. For example, if the matrix

$$H_2^\diamond = \begin{pmatrix} a_0 & a_1 \\ a_1 & -a_1 \end{pmatrix},$$

then, by the negative algorithm $H_2^\diamond$ can be computed by only 2 multiplications and 2 additions instead of 3 multiplications and 3 additions by the positive algorithm.

3.3 The Winograd-like Algorithm

Now we are ready to construct an arbitrary point algorithm based on the 2-point algorithm. Because the negative algorithm has some interesting properties, we would use this method in the following discussion. A recursive procedure of building the arbitrary n-point algorithm is described as follows.

(i). For the base case of $n = 2$, as we discussed,

$$H_2^\diamond = H_2' + H_2'' = C_2^\diamond G_2^\diamond M_2^\diamond, \tag{11}$$

where, by writing in regular form,

$$H_2' = C_2' G_2' M_2' \tag{12}$$

and H_2'' is a diagonal matrix of size 2. Then,

$$C_2^\diamond = (\, C_2' \quad I_2 \,) = C_2' \vdash I_2; \tag{13}$$

$$M_2^\diamond = \begin{pmatrix} M_2' \\ I_2 \end{pmatrix} = M_2' \perp I_2; \tag{14}$$

$$G_2 = G_2' \oplus H_2''. \tag{15}$$

(ii). If $n > 2$, the following two rules can be alternatively used during the recursion.

a). If $n = p$, where p is a prime, let $p' = p - 1$, we see that the matrix $H_p^\diamond$ has the form

$$H_p^\diamond = \begin{pmatrix} a_0 & a_1 & \cdots & a_{p'} \\ a_1 & & & \\ \vdots & & H_{p'}^\diamond & \\ a_{p'} & & & \end{pmatrix}, \tag{16}$$

where $H_{p'}^\diamond$ is a half-circulant matrix of order p'. Denote by $\underline{a}$ the vector formed by the first column of $H_p^\diamond$ excluding the element a_0, $\underline{a} = \perp_{i=1}^{p'} a_i$. By (16), $H_p^\diamond$ can be written as

$$H_p^\diamond = \begin{pmatrix} 0 & \\ & H_{p'}^\diamond \end{pmatrix} + \begin{pmatrix} a_0 & \underline{a}^t \\ \underline{a} & 0(p') \end{pmatrix}. \tag{17}$$

Suppose that $H_{p'}^\diamond$ can be factorized by the same forms from (11) to (15),

$$H_{p'}^\diamond = H_{p'}' + H_{p'}'', \tag{18}$$

where

$$H_{p'}' = C_{p'}' G_{p'}' M_{p'}' \tag{19}$$

and $H_{p'}''$ is the diagonal matrix,

$$H_{p'}'' = \bigoplus_{i=1}^{p'} g_i'. \tag{20}$$

Then $H_p^\diamond$ in (17) becomes

$$H_p^\diamond = \begin{pmatrix} 0 & \\ & H_{p'}' \end{pmatrix} + \begin{pmatrix} a_0 & \underline{a}^t \\ \underline{a} & H_{p'}'' \end{pmatrix}. \tag{21}$$

Let

$$h = \begin{pmatrix} a_0 & \underline{a}^t \\ \underline{a} & H_{p'}'' \end{pmatrix} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{p'} \\ a_1 & g_1' & & \\ \vdots & & \ddots & \\ a_{p'} & & & g_{p'}' \end{pmatrix}. \tag{22}$$

We see that h consists of p' 2-point half-circulant matrices. Set

$$a = \sum_{i=0}^{p'} a_i; \qquad h_0 = \begin{pmatrix} a & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}, \tag{23}$$

and for $0 < i < p$, set

$$h_1 = \begin{pmatrix} -a_1 & a_1 & \cdots & 0 \\ a_1 & g_1' & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix};$$

$$\vdots \tag{24}$$

$$h_i = \begin{pmatrix} -a_i & \cdots & a_i & \cdots \\ \vdots & & \vdots & \\ a_i & \cdots & g_i' & \\ \vdots & & & \end{pmatrix}.$$

Then, we have

$$h = \sum_{i=0}^{p'} h_i. \tag{25}$$

Observe that each h_i, $0 < i < p$, is a special 2-point half-circulant matrix. By the negative algorithm,

$$h_i = h_i' + h_i''; \qquad 0 < i < p, \tag{26}$$

where h_i' is formed from h_i by replacing g_i' with $-a_i$, and h_i'' is the diagonal matrix with the only non-zero element $a_i + g_i'$ in the i-th position. Set

$$h'' = h_0 + \sum_{i=1}^{p'} h_i'' = a \oplus (\bigoplus_{i=1}^{p'} (a_i + g_i')), \tag{27}$$

which is a diagonal matrix, and also set

$$h' = \sum_{i=1}^{p'} h_i' = \begin{pmatrix} a_0 - a & \underline{a}^t \\ \underline{a} & diag.(-\underline{a}) \end{pmatrix}, \tag{28}$$

where $diag.(-\underline{a})$ is the diagonal format of $-\underline{a}$. Then, h' can be factorized as

$$h' = C'G'M', \tag{29}$$

where

$$\begin{aligned} C' &= -1^t_{p'} \perp I_{p'}; \\ G' &= diag.(\underline{a}); \\ M' &= 1_{p'} \vdash -I_{p'}. \end{aligned} \tag{30}$$

By (25), we have

$$h = h' + h''. \tag{31}$$

Now, set

$$H''_p = h'' \tag{32}$$

and

$$H'_p = (0 \oplus H'_{p'}) + h' = C'_p G'_p M'_p. \tag{33}$$

Then, by (19) and (29), we have that

$$\begin{aligned} C'_p &= (0_{m'_{p'}} \perp C'_{p'}) \vdash C'; \\ M'_p &= (0_{m'_{p'}} \vdash M'_{p'}) \perp M'; \\ G'_p &= G'_{p'} \oplus G'; \qquad m'_{p'} = M^\circ(p') - p'. \end{aligned} \tag{34}$$

Finally, by (21) and from (31) to (33), we obtained

$$H^\circ_p = H'_p + H''_p, \tag{35}$$

which has the same regular forms as the base case.

Compared with p'-point algorithm, the extra computation for p-point algorithm comes from the computation for the matrix h, which uses p multiplications and $2p'$ additions, plus p' additions for the final sum, we have

$$M^\circ(p) = M^\circ(p') + p; \tag{36}$$

$$A^\circ(p) = A^\circ(p') + 3(p-1). \tag{37}$$

b). If $n = s \cdot r$, then, break H°_n into $s \times s$ blocks,

$$H^\circ_n = \begin{pmatrix} h_0 & h_1 & \cdots & h_{s-1} \\ h_1 & h_2 & \cdots & h'_0 \\ \vdots & & & \vdots \\ h_{s-1} & h'_0 & \cdots & h'_{s-2} \end{pmatrix}, \tag{38}$$

which is a $s \times s$ block half-circulant matrix having $r \times r$ half-circulant blocks. First block-diagonalize H°_n by s-point algorithm,

$$H^\circ_n = (C^\circ_s \otimes I_r) G' (M^\circ_s \otimes I_r), \tag{39}$$

where

$$G' = \bigoplus_{i=0}^{m_s-1} G'_i; \qquad m_s = M^\circ(s), \tag{40}$$

is a block diagonal matrix with size m_s. Each G'_i is also a half-circulant matrix of size r and can be further diagonalized by r-point algorithm,

$$G'_i = C^\circ_r G_i M^\circ_r. \tag{41}$$

Then,

$$G' = (I_{m_s} \otimes C^\circ_r) G (I_{m_s} \otimes M^\circ_r), \tag{42}$$

where

$$G = \bigoplus_{i=0}^{m_s-1} G_i. \tag{43}$$

Put (42) into (39), we have the result

$$H^\circ_n = (C^\circ_s \otimes C^\circ_r) G (M^\circ_s \otimes M^\circ_r). \tag{44}$$

The arithmetic count can be measured by

$$M^\circ(n) = M^\circ(s) \cdot M^\circ(r); \tag{45}$$

$$A^\circ(n) = r \cdot A^\circ(s) + M^\circ(s) \cdot A^\circ(r). \tag{46}$$

However, to prove that the factorization of H°_n has the same form as the base case, we have to make some modification to formula (44). The purpose of doing this is to meet the requirements by the recursive rules and for the later proving. But it is not necessary to do so in the practical applications.

According to the recursive assumptions, we have that

$$C^\circ_s = C'_s \vdash I_s; \qquad C^\circ_r = C'_r \vdash I_r, \tag{47}$$

then,

$$C^\circ_s \otimes C^\circ_r = (C'_s \vdash I_s) \otimes C'_r = (C'_s \otimes C^\circ_r) \vdash (I_s \otimes C^\circ_r). \tag{48}$$

Observe that

$$\begin{aligned} I_s \otimes C^\circ_r &= P(n,r)(C^\circ_r \otimes I_s) P(sm_r, m_r) \\ &= P(n,r)((C'_r \otimes I_s) \vdash (I_r \otimes I_s)) P(sm_r, m_r) \\ &= ((P(n,r)(C'_r \otimes I_s)) \vdash P(n,r)) P(sm_r, m_r) \\ &= C' P(sm_r, m_r), \end{aligned} \tag{49}$$

where

$$m_r = M^\circ(r), \qquad m'_r = m_r - r,$$

and C'_r is $r \times m'_r$ matrix. Similarly, we have

$$\begin{aligned} I_s \otimes M_r^\circ &= P(sm_r, s)(((M'_r \otimes I_s)P(n,s)) \perp P(n,s)) \\ &= P(sm_r, s)M'. \end{aligned} \tag{50}$$

Now set

$$G(1) = \bigoplus_{i=0}^{m'_s-1} G_i; \qquad G(2) = \bigoplus_{i=m'_s}^{m_s-1} G_i. \tag{51}$$

By (43), we have $G = G(1) \oplus G(2)$. Set

$$\begin{aligned} H(1) &= (C'_s \otimes C_r^\circ)G(1)(M'_s \otimes M_r^\circ); \\ H(2) &= (I_s \otimes C_r^\circ)G(2)(I_s \otimes M_r^\circ). \end{aligned} \tag{52}$$

By (49) and (50),

$$H(2) = C'P(sm_r, m_r)G(2)P(sm_r, s)M' = C'G'M', \tag{53}$$

where

$$G' = P(sm_r, m_r)G(2)P(sm_r, s)$$

is also a diagonal matrix. Again break G' into two parts,

$$G' = G'(1) \oplus G'(2), \tag{54}$$

where $G'(1)$ has the size sm'_r and $G'(2)$ has the size n. Then, by (49), (50) and (54), $H(2)$ in (53) can be rewritten as the sum

$$\begin{aligned} H(2) &= P(n,r)(C'_r \otimes I_s)G'(1)(M'_r \otimes I_s)P(n,s) \\ &\quad + P(n,r)G'(2)P(n,s) \\ &= h' + h'', \end{aligned} \tag{55}$$

where h'' is also a diagonal matrix with the size n. Finally, set

$$H'_n = H(1) + h'; \qquad H''_n = h''. \tag{56}$$

We have that

$$H_n^\circ = H'_n + H''_n = C_n^\circ G_n M_n^\circ, \tag{57}$$

where

$$
\begin{aligned}
&C_n^\circ = (C_s' \otimes C_r^\circ) \vdash (P(n,r)(C_r' \otimes I_s) \vdash I_n = C_n' \vdash I_n; \\
&M_n^\circ = (M_s' \otimes M_r^\circ) \perp (M_r' \otimes I_s)P(n,s) \perp I_n = M_n' \perp I_n; \\
&G_n = G(1) \oplus G'(1) \oplus H_n'' = G_n' \oplus H''.
\end{aligned} \tag{58}
$$

The recursive procedure now is completed. Any point algorithm can be constructed by the procedure based only on the half-cyclic convolution algorithm. The algorithm can be built up by three ways, based only on the 2-point negative algorithm or only on the 2-point positive algorithm, or mixing them during the construction, which are called *Negative, Positive or Mixed Winograd-like algorithm*, respectively.

3.4 Some Properties of The Winograd-like Algorithm

The Winograd-like algorithm, by our informal definition, is real-type algorithm. It has some more interesting properties than the Winograd algorithm. For example, all the addition matrices are Uni-valued, which is very easy for implementation. Also, every element in the diagonal matrix is a sum of some elements in H_n° without any other coefficients. In particular, for negative algorithm, the sums consist of only additions, which is easy for pre-computation. In the following, we will discuss some properties of negative algorithm, which are useful in the later discussion. Finally, the performance of the Winograd-like algorithm will be analyzed.

(i) Some Properties of The Negative Algorithm

Property 1. For the n-point algorithm, where the post-addition matrix $C_n^\circ = C_n' \vdash I_n$ and the pre-addition matrix $M_n^\circ = M_n' \perp I_n$, denote by $c_{i,j}$ and $m_{l,k}$ the entries of the matrix C_n' and the matrix M_n', respectively. Then,

$$
\sum_{i=0}^{n-1} c_{i,j} = 0; \quad \sum_{k=0}^{n-1} m_{l,k} = 0; \qquad 0 \le j, l < M^\circ(n) - n. \tag{59}
$$

Proof We only give a proof for C_n', the same procedure can be applied to M_n'. Obviously, it is true for $n = 2$. If n is a prime number

p, by (34) we have

$$C'_p = (0_{m'_{p'}} \perp C'_{p'}) \vdash C'; \qquad m'_{p'} = M^\circ(p') - p'.$$

Suppose that the property (59) holds for $C'_{p'}$, then

$$\sum_{i=0}^{p'} c_{i,j} = 0; \qquad 0 \le j < m_{p'} = M^\circ(p'). \tag{60}$$

By (30), we have

$$C' = -1^t_{p'} \perp I_{p'}.$$

It is very clear that

$$\sum_{i=0}^{p'} c_{i,j} = 0; \qquad m_{p'} \le j < M^\circ(n). \tag{61}$$

Putting (60) and (61) together, we proved the property for prime case.

If $n = sr$, by (58)

$$C'_n = (C'_s \otimes C^\diamond_r) \vdash (P(n,r)(C'_r \otimes I_s).$$

Suppose that the property holds for both C'_s and C'_r. It is easy to prove that for any matrix A, if the property holds for matrix B then the property holds for the matrices $A \otimes B$ and $B \otimes A$, too. Also, the permutation matrix $P(n,r)$ does not change the property. Therefore, the matrix C'_n satisfies the property (59).

Property 2. For the n-point algorithm, where $H^\diamond_n = H'_n + H''_n$, denote by $b_{i',j'}$ the elements in $H^\diamond_n$, and by g_i the elements in the diagonal matrix H''_n. Then

$$g_i = \sum_{j'=0}^{n-1} b_{i',j'}; \qquad 0 \le i' < n. \tag{62}$$

Proof The base case of $n = 2$ is obvious. If n is a prime number p, by (32) and (27), we have

$$H''_p = a \oplus (\bigoplus_{i=1}^{p'} (a_i + g'_i)).$$

Because

$$H''_{p'} = \bigoplus_{i=1}^{p'} g'_i,$$

suppose that the property holds for $H''_{p'}$, then we have

$$g_i = a_i + g'_i = b_{i,0} + \sum_{j=1}^{p'} b_{i,j} = \sum_{j=0}^{p'} b_{i,j}; \qquad 0 < i < p. \tag{63}$$

By (23), $a = \sum_{i=0}^{p'} a_i$ also satisfies the property.

If $n = sr$, redenote by $h_{k,l}$ the block entries of H°_n in (38). Suppose that the property holds for s-point algorithm, so does for s-point block format. Then the blocks G'_i in (41), where $m'_s \le i < m_s$, can be written as

$$G'_i = \sum_{l=0}^{s-1} h_{k,l}; \qquad 0 \le k < m_s, i = k + m'_s, \tag{64}$$

which was factorized by r-point algorithm. Denote by $g_{i,j}$ the j-th element in the corresponding diagonal matrix G_i in (41), where $0 \le j < m_r$. By the property for r-point algorithm, it is easy to find that

$$g_{i,j} = \sum_{j'=0}^{n-1} b_{i',j'}, \tag{65}$$

where

$$m'_s \le i < m_s; \quad m'_r \le j < m_r; \quad i' = (i - m'_s)r + (j - m'_r).$$

By the settings of (51), (54) and the permutation in (53),

$$G' = P(sm_r, m_r)G(2)P(sm_r, s) = G'(1) + G'(2),$$

where

$$G(2) = \bigoplus_{i=m'_s}^{m_s-1} G_i.$$

Observe that

$$G'(2) = \bigoplus_{j=m'_r}^{m_r-1} \Big(\bigoplus_{i=m'_s}^{m_s-1} g_{i,j} \Big), \tag{66}$$

finally, by (55) and (56),

$$\begin{aligned} H_n'' &= P(n,r)G'(2)P(n,s) \\ &= \bigoplus_{i=m_s'}^{m_s-1} \Big(\bigoplus_{j=m_r'}^{m_r-1} g_{i,j} \Big) \\ &= \bigoplus_{i'=0}^{n-1} l\Big(\sum_{j'=0}^{n-1} b_{i',j'} \Big). \end{aligned} \tag{67}$$

The proof for the property (62) now is completed.

The property 1. is similar to the property of the DFT matrix. It may be used to reduce the computation for matrix $A(p)$ in prime case FFT algorithm, which has been mentioned in chapter 1. The property 2., sometimes, may give some special values which can also reduce the computation, or by the block format, may give some special matrices such as circulant matrix. We will see some applications of them.

(ii). The Performance of The Winograd-Like Algorithm

In the construction procedure, we have measured the performance for each particular case by the means of arithmetic count. However, if we think the procedure carefully, we may find something ambiguous. For example, if n is a non-prime odd number, the algorithm can be built up by either rule; if $n = sr = s_1 r_1$, the algorithm can also be built up by either using the factorization $n = rs$ or using $n = s_1 r_1$. We will discuss these problems.

For the first question, we have proved that only when $n = p^2$ the two rules are equivalent, where p is a prime. Otherwise the second rule is always better than the first one for non-prime size. We are not going to write the proof here because it is a little tedious. For the second question, we will prove that all factorizations are equivalent in the sense of performance measured by equations (45) and (46). We begin with the proving for a property about the arithmetic count for this algorithm.

Theorem 3.1 The number of additions $A^\circ(n)$ and the number of multiplications $M^\circ(n)$ required by the n-point Winograd-like algorithm satisfies

$$A^\circ(n) = 3(M^\circ(n) - n). \tag{68}$$

Proof For $n = 2$, we have that

$$A^\circ(2) = 3 = 3(M^\circ(2) - 2). \tag{69}$$

If $n = p$, suppose that (69) holds for p'-point algorithm. Then by the formula (37) and (38), we have

$$A^\circ(p) = A^\circ(p') + 3p' = 3M^\circ(p') = 3(M^\circ(p) - p). \tag{70}$$

If $n = sr$, suppose that the property holds for both s-point and r-point algorithms. By (45) and (46),

$$\begin{aligned} A^\circ(n) &= r \cdot A^\circ(s) + M^\circ(s) \cdot A^\circ(r) \\ &= 3rM^\circ(s) - 3rs + 3M^\circ(s)M^\circ(r) - 3rM^\circ(s) \\ &= 3(M^\circ(n) - n). \end{aligned} \tag{71}$$

Now consider the different factorizations of n. Denote by $M^\circ_{(s,r)}(n)$ and $A^\circ_{(s,r)}(n)$ the arithmetic count corresponding to the factorization $n = sr$. Then, by (45) we have

$$M^\circ_{(r,s)}(n) = M^\circ(r)M^\circ(s) = M^\circ_{(s,r)}(n). \tag{72}$$

Now by the property (68),

$$A^\circ_{(r,s)}(n) = A^\circ_{(s,r)}(n). \tag{73}$$

For the factorization $n = s_1 r_1$, because s can not be relative prime to both s_1 and r_1, otherwise, s is relative prime to n, there must exist a number k such that

$$\begin{aligned} s_1 &= ks \quad (or \quad s_1 = kr); \\ kr_1 &= r \quad (or \quad kr_1 = s). \end{aligned} \tag{74}$$

Again by (45), we have

$$\begin{aligned} M^\circ_{(s_1,r_1)}(n) &= M^\circ(ks)M^\circ(r_1) \\ &= M^\circ(s)M^\circ(k)M^\circ(r_1) \\ &= M^\circ(s)M^\circ(kr_1) = M^\circ(s)M^\circ(r). \end{aligned} \tag{75}$$

So,

$$A^\circ_{(s_1,r_1)}(n) = A^\circ_{(s,r)}(n). \tag{76}$$

Therefore, for any factorization of n the arithmetic count is unique and can be obtained by (45) and (46). This property will give us a flexible way of designing non-prime size algorithms.

The recursive procedure for non-prime case can be extended to any factorization of n with more than two factors. In general, by the theorem in Number theory, n can be factorized by

$$n = \prod p_i^{k_i}, \tag{77}$$

where p_i is a prime number. Then

$$M^\circ(n) = \prod (M^\circ(p_i))^{k_i}. \tag{78}$$

The property (68) shows a relation between $A^\circ(n)$ and $M^\circ(n)$,

$$\frac{A^\circ(n)}{M^\circ(n)} \leq 3, \tag{79}$$

which is much close to the values of the parameter R on some machines. A better load balance could be achieved.

3.5 The Special Case of $n = 2^k$

When $n = 2^k$, if we build up the 2^k-point algorithm by recursively using 2-point algorithm, then for the factorization

$$H_n = C_n^\circ G_n^\circ M_n^\circ, \tag{80}$$

we have

$$\begin{aligned} C_n^\circ &= \prod_{i=1}^{k} (I_{t_i} \otimes C_2^\circ \otimes I_{2^{k-i}}); \\ M_n^\circ &= \prod_{i=k}^{1} (I_{t_i} \otimes M_2^\circ \otimes I_{2^{k-i}}), \end{aligned} \tag{81}$$

where

$$t_i = 3^{i-1}. \tag{82}$$

Proof We will use the inductive method and only prove the equation for C_n°, the proof for M_n° can be made in the same way. When $k = 1$, it is obviously true. Suppose that for $n = 2^{k-1}$, we have

$$C_{2^{k-1}}^\circ = \prod_{i=1}^{k-1} (I_{t_i} \otimes C_2^\circ \otimes I_{2^{k-i-1}}); \qquad t_i = 3^{i-1}. \tag{83}$$

Then when $n = 2^k$, by the construction rule, we have

$$C_{2^k}^{\circ} = (C_2^{\circ} \otimes I_{2^{k-1}})(I_3 \otimes C_{2^{k-1}}^{\circ}). \tag{84}$$

Because

$$\begin{aligned}(C_2^{\circ} \otimes I_{2^{k-1}})(I_3 \otimes C_{2^{k-1}}^{\circ}) &= I_3 \otimes \prod_{i=1}^{k-1}(I_{t_i} \otimes C_2^{\circ} \otimes I_{2^{k-i-1}}) \\ &= \prod_{i=1}^{k-1} I_3 \otimes (I_{t_i} \otimes C_2^{\circ} \otimes I_{2^{k-i-1}}) \\ &= \prod_{i=1}^{k-1}(I_{3t_i} \otimes C_2^{\circ} \otimes I_{2^{k-i-1}}) \\ &= \prod_{i=1}^{k-1}(I_{t_{i+1}} \otimes C_2^{\circ} \otimes I_{2^{k-i-1}}),\end{aligned} \tag{85}$$

placing (85) into (84), we obtained that

$$\begin{aligned}C_{2^k}^{\circ} &= (C_2^{\circ} \otimes I_{2^{k-1}}) \cdot \prod_{i=1}^{k-1}(I_{t_{i+1}} \otimes C_2^{\circ} \otimes I_{2^{k-i-1}}) \\ &= \prod_{i=1}^{k}(I_{t_i} \otimes C_2^{\circ} \otimes I_{2^{k-i}}).\end{aligned} \tag{86}$$

The formula (81) provides a way to build up the 2^k-point algorithm for sequential purpose. The result will be used in the later chapters also. For parallel or vector processing, we can choose any factorization of n. For nay factorization, by (78), the arithmetic count would be

$$M^{\circ}(2^k) = M^{\circ}(2)^k = 3^k, \tag{87}$$

and by the Theorem 3.1,

$$A^{\circ}(2^k) = 3(3^k - 2^k) = 3^{k+1} - 3 \cdot 2^k. \tag{88}$$

3.6 Summary

In the following table we list the arithmetic counts for both Winograd cyclic convolution algorithm and the Winograd-like half-cyclic convolution algorithm with the sample points from 2 to 9. Notice that

when the size is prime (2,3,5,7) the Winograd-like algorithm uses a few more multiplications than the Winograd cyclic convolution algorithm, but uses less additions with at least the same number as increased in multiplication.

In addition, the algorithm has very good features for parallel or vector processing. By the construction rule for non-prime case, there is no any permutation involved, and the factorization of n is not necessarily relative prime, which provides a flexibility for parallel or vector processing. For prime size, the algorithm is constructed also based on non-prime case algorithm. As we proved all the factorizations will result in the same performance in sequential manner, which can also benefit the implementation by parallel or vector processing.

Table 3-1

Size	Winograd-cyclic		Half-cyclic	
k	$M(k)$	$A(k)$	$M^{\circ}(k)$	$A^{\circ}(k)$
2	2	4	3	3
3	4	11	6	9
4	5	15	9	15
5	10	31	14	27
6	8	34	18	36
7	16	70	25	54
8	14	46	27	57
9	19	74	36	81

CHAPTER 4

THE ALGORITHMS FOR ZERO-HALF-CIRCULANT MATRIX AND ANOTHER ALGORITHM FOR HALF-CYCLIC CONVOLUTION

4.1 Introduction

The *n-point Zero-half-circulant matrix* is defined as

$$H_n^\circ = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \\ a_1 & a_2 & \cdots & a_{n-1} & 0 \\ \vdots & & & & \vdots \\ a_{n-2} & a_{n-1} & \cdots & 0 & 0 \\ a_{n-1} & 0 & \cdots & 0 & 0 \end{pmatrix}, \tag{1}$$

which is a special case of half-circulant matrix as we mentioned. But, we see that the matrix H_n° is more like a half of the computational matrix in Linear convolution. Therefore, the zero-half-circulant matrix has very close relation with linear convolution and half-cyclic convolution or even cyclic convolution.

Consider the base case of $n = 2$,

$$H_2^\circ = \begin{pmatrix} a_0 & a_1 \\ a_1 & 0 \end{pmatrix}, \tag{2}$$

which can be computed by 3 multiplications and only 1 addition instead of 3 additions for general half-cyclic convolution. Therefore, it is necessary to use separate algorithm or technique to do the computation on zero-half-circulant matrix. There are many ways to reduce the computation for this case. In this chapter, two algorithms on zero-half-circulant matrix have been designed. Then, an algorithm for half-cyclic convolution, which is based on the cyclic convolution and the computation on zero-half-circulant matrix, will be introduced. We will use the symbol o in the corresponding notations for zero-half-circulant matrix.

4.2 The Binary-Recursion Algorithm

In the definition (1), we see that the non-zero submatrices of a zero-half-circulant matrix are either zero-half-circulant or half-circulant

matrices in the general form. The idea of this algorithm is to apply half-cyclic convolution algorithm on the maximum half-circulant submatrix, which is about half size of n. The zero-half-circulant submatrices then can be computed on recursion. The algorithm is described as follows.

(i). If n is an even number, where $n > 2$, set $k = \frac{n}{2}$, H_n° can be written in the block form

$$H_n^\circ = \begin{pmatrix} H_k^\diamond & H_k^\circ \\ H_k^\circ & 0(k) \end{pmatrix}, \tag{3}$$

where $H_k^\diamond$ is a k-point half-circulant matrix and H_k° is a k-point zero-half-circulant matrix. If

$$H_k^\diamond = C_k^\diamond G_k^\diamond M_k^\diamond; \qquad H_k^\circ = C_k^\circ G_k^\circ M_k^\circ, \tag{4}$$

then

$$H_n^\circ = C_n^\circ G_n^\circ M_n^\circ, \tag{5}$$

where

$$\begin{aligned} C_n^\circ &= (C_k^\diamond \vdash C_k^\circ) \oplus C_k^\circ; \\ G_n^\circ &= G_k^\diamond \oplus G_k^\circ \oplus G_k^\circ; \\ M_n^\circ &= (M_k^\diamond \oplus M_k^\circ) \perp (M_k^\circ \vdash 0_{m_k^\circ} \otimes 0_k^t). \end{aligned} \tag{6}$$

The number of multiplications

$$M^\circ(n) = M^\circ(k) + 2M^\circ(k). \tag{7}$$

There are k more additions needed for the summation of $C_k^\diamond$ and C_k° in C_n°. But, because there some common additions in both C_n° and M_n°,

$$A^\circ(n) \leq A^\circ(k) + 2A^\circ(k) + k, \tag{8}$$

where $M^\circ(k)$ and $A^\circ(k)$ are the arithmetic count for k-point half-cyclic convolution algorithm.

However, when k is an odd number, there is a better way. Set $k_0 = k + 1$ and $k_1 = n - k_0$. Instead of picking up $H_k^\diamond$ as the maximum half-circulant submatrix, we may pick $H_{k_0}^\diamond$ which is one more row and one more column added to $H_k^\diamond$. But, $H_{k_0}^\diamond$ will contain one zero element at the lower-right corner. Now the matrix H_n° in (3) can be written as

$$H_n^\circ = \begin{pmatrix} H_{k_0}^\diamond & (h^\circ)^t \\ h^\circ & 0(k_1) \end{pmatrix}, \tag{9}$$

where

$$h^\circ = (\, h^\circ_{k_1} \quad 0\,) = h^\circ_{k_1} \vdash (0_{k_1} \otimes 0^t_2) \tag{10}$$

has the size $k_1 \times k_0$ and $h^\circ_{k_1}$ is a k_1-point zero-half-circulant matrix. Set

$$\begin{aligned} C^\circ &= C_{k_1} \perp (0_2 \otimes 0^t_{k_1}); \\ M^\circ &= M_{k_1} \vdash (0_{k_1} \otimes 0^t_2). \end{aligned} \tag{11}$$

The matrices in (6) now become

$$\begin{aligned} C^\circ_n &= (C^\circ_{k_0} \vdash C^\circ) \oplus C^\circ_{k_1}; \\ G^\circ_n &= G^\circ_{k_0} \oplus G^\circ_{k_1} \oplus G^\circ_{k_1}; \\ M^\circ_n &= (M^\circ_{k_0} \oplus M^\circ_{k_1}) \perp (M^\circ \vdash 0_{m^\circ_{k_1}} \otimes 0_{k_1}). \end{aligned} \tag{12}$$

The arithmetic count

$$M^\circ(n) = M^\circ(k_0) + 2M^\circ(k_1); \tag{13}$$

$$A^\circ(n) \le A^\circ(k_0) + 2A^\circ(k_1) + k_1. \tag{14}$$

By this modification, the result could be better. For example, $n = 6$. By the first way, we need 16 multiplications and 18 additions, while by second way only 15 multiplications and 19 additions are needed.

(ii). If n is odd, then set $k_0 = \frac{n+1}{2}$ and $k_1 = n - k_0$. The matrix H°_n can be written as the same form as in (9) except that the submatrix h° consists of $h^\circ_{k_1}$ and one column 0_{k_1},

$$h^\circ = h^\circ_{k_1} \vdash 0_{k_1}. \tag{15}$$

The corresponding factor matrices can be derived from (12). The arithmetic count is the same as (13) and (14).

The n-point algorithm can be built up based on k-point or k_0-point algorithm for both half-circulant and zero-half-circulant matrices. Because k or k_0 is about half size of n, the algorithm is called *Binary-recursion algorithm.*

4.3 The Base-2-Recursion Algorithm

Instead of breaking H°_n into the blocks of about half size, we can divide it into the blocks of size 2. Then, H°_n seems to be a block zero-half-circulant matrix of about half size. We call this algorithm *Base-2-recursion algorithm*, which is described below.

(i). If n is an even number, then set $k = \frac{n}{2}$. Writing H_n° as

$$H_n^\circ = \begin{pmatrix} h_0 & h_1 & \cdots & h_{k-1} \\ h_1 & h_2 & \cdots & 0(2) \\ \vdots & & & \vdots \\ h_{k-1} & 0(2) & \cdots & 0(2) \end{pmatrix}, \tag{16}$$

where each h_i is a 2-point half-circulant matrix except that h_{k-1} is also zero-half-circulant. We will treat h_{k-1} as general. By the positive 2-point Winograd-like algorithm,

$$h_i = h_i' + h_i'', \tag{17}$$

where

$$\begin{aligned} h_i' &= a_{2i+1} \otimes I(2); \qquad 0 \le i < k; \\ h_i'' &= (a_i - a_{2i+1}) \oplus (a_{2i+2} - a_{2i+1}). \end{aligned} \tag{18}$$

Notice that when $i = k-1$, we have that $a_{2i+2} = 0$ in h_{k-1}''. Set

$$\begin{aligned} H_n(1) &= \begin{pmatrix} h_0' & h_1' & \cdots & h_{k-1}' \\ h_1' & h_2' & \cdots & 0(2) \\ \vdots & & & \vdots \\ h_{k-1}' & 0(2) & \cdots & 0(2) \end{pmatrix}; \\ H_n(2) &= \begin{pmatrix} h_0'' & h_1'' & \cdots & h_{k-1}'' \\ h_1'' & h_2'' & \cdots & 0(2) \\ \vdots & & & \vdots \\ h_{k-1}'' & 0(2) & \cdots & 0(2) \end{pmatrix}. \end{aligned} \tag{19}$$

Then

$$H_n^\circ = H_n(1) + H_n(2). \tag{20}$$

By (18) and the properties for tensor product, $H_n(1)$ can be written as

$$H_n(1) = H_k^\circ(0) \otimes I(2), \tag{21}$$

where

$$H_k^\circ(0) = \begin{pmatrix} a_1 & a_3 & \cdots & a_{2k-1} \\ a_3 & a_5 & \cdots & 0 \\ \vdots & & & \vdots \\ a_{2k-1} & 0 & \cdots & 0 \end{pmatrix}. \tag{22}$$

Suppose that $H_k^\circ(0)$ is factorized by

$$H_k^\circ(0) = C_k^\circ G_0^\circ M_k^\circ. \tag{23}$$

Then

$$\begin{aligned} H_n(1) &= (C_k^\circ G_0^\circ M_k^\circ) \otimes (1_2 \otimes 1_2^t) \\ &= (C_k^\circ \otimes 1_2) G_0^\circ (M_k^\circ \otimes 1_2^t). \end{aligned} \tag{24}$$

Now look at $H_n(2)$. Denote by g_i' and g_i'' the two elements in each h_i'' in formula (18) and set two zero-half-circulant matrices

$$\begin{aligned} H_k^\circ(1) &= \begin{pmatrix} g_0' & g_1' & \cdots & g_{k-1}' \\ g_1' & g_2' & \cdots & 0 \\ \vdots & & & \vdots \\ g_{k-1}' & 0 & \cdots & 0 \end{pmatrix}; \\ H_k^\circ(2) &= \begin{pmatrix} g_0'' & g_1'' & \cdots & g_{k-1}'' \\ g_1'' & g_2'' & \cdots & 0 \\ \vdots & & & \vdots \\ g_{k-1}'' & 0 & \cdots & 0 \end{pmatrix}. \end{aligned} \tag{25}$$

It is easy to find that

$$H_n(2) = P(n,k)(H_k^\circ(1) \oplus H_k^\circ(2))P(n,2). \tag{26}$$

Suppose that $H_k^\circ(1)$ and $H_k^\circ(2)$ are factorized by

$$H_k^\circ(1) = C_k^\circ G_1^\circ M_k^\circ; \qquad H_k^\circ(2) = C_k^\circ G_2^\circ M_k^\circ. \tag{27}$$

Then the factorization for $H_n(2)$ is

$$\begin{aligned} H_n(2) &= P(n,k)(I_2 \otimes C_k^\circ)(G_1^\circ \oplus G_2^\circ)(I_2 \otimes M_k^\circ)P(n,2) \\ &= (C_k^\circ \otimes I_2)P(n,2)(G_1^\circ \oplus G_2^\circ)P(n,k)(M_k^\circ \otimes I_2). \end{aligned} \tag{28}$$

Finally, by (20) and (24) with (28), we have

$$H_n^\circ = C_n^\circ G_n^\circ M_n^\circ, \tag{29}$$

where

$$\begin{aligned} C_n^\circ &= (C_k^\circ \otimes 1_2) \vdash (C_k^\circ \otimes I_2); \\ M_n^\circ &= (M_k^\circ \otimes 1_2^t) \perp (M_k^\circ \otimes I_2); \\ G_n^\circ &= G_0^\circ \oplus P(n,2)(G_1^\circ \oplus G_2^\circ)P(n,k). \end{aligned} \tag{30}$$

The arithmetic count would be

$$M^\circ(n) = 3M^\circ(k); \tag{31}$$

$$A^\circ(n) \le 3A^\circ(k) + k + n, \tag{32}$$

where the k additions are needed by the tensor product $M_k^\circ \otimes 1_2^t$, and the n additions are for implementing the sum $H_n(1) + H_n(2)$ in C_n°.

(ii). If n is an odd number, set $k = \frac{n-1}{2}$ and $k_0 = n - k = k + 1$. The matrix H_n° can be written as

$$H_n^\circ = \begin{pmatrix} H_{n-1}^\circ & \\ & 0 \end{pmatrix} + \widehat{T}, \tag{33}$$

where H_{n-1}° is a $(n-1)$-point zero-half-circulant matrix and

$$\widehat{T} = a_{n-1}\widehat{I}_{n-1}. \tag{34}$$

Because $n-1$ is even, by the method used in (20) for H_{n-1}°, set

$$\begin{aligned} H_n(1) &= H_{n-1}(1) \oplus 0; \\ H_n(2) &= (H_{n-1}(2) \oplus 0) + \widehat{T}. \end{aligned} \tag{35}$$

We see that $H_n(2)$ is similar to $H_{n-1}(2)$ except that now n is odd. Because $(n,2) = 1$ and $2 \cdot k_0 \equiv 1 \bmod n$, by the n-point stride 2 mod n permutation, we have that

$$\begin{aligned} H_n(2) &= P(n, k_0/n)(H_k^\circ(1) \oplus H_{k_0}^\circ(2))P(n, 2/n) \\ &= P(n, k_0/n)(C_k^\circ \oplus C_{k_0}^\circ)(G_1^\circ \oplus G_2^\circ)(M_k^\circ \oplus M_{k_0})P(n, 2/n). \end{aligned} \tag{36}$$

Finally, we have

$$\begin{aligned} C_n^\circ &= ((C_k^\circ \otimes 1_2) \perp 0_{m_k^\circ}^t) \vdash P(n, k_0/n)(C_k^\circ \oplus C_{k_0}^\circ); \\ M_n^\circ &= ((M_k^\circ \otimes 1_2^t) \vdash 0_{m_k^\circ}) \perp (M_k^\circ \oplus M_{k_0})P(n, 2/n); \\ G_n^\circ &= G_0^\circ \oplus G_1^\circ \oplus G_2^\circ, \end{aligned} \tag{37}$$

where $m_k^\circ = M^\circ(k)$. The arithmetic count

$$M^\circ(n) = 2M^\circ(k) + M^\circ(k_0); \tag{38}$$

$$A^{\circ}(n) \leq 2A^{\circ}(k) + A^{\circ}(k_0) + 3k, \tag{39}$$

where k additions are needed by $M_k^{\circ} \otimes 1_2^t$ and $2k = n-1$ additions for implementing $H_n(1) + H_n(2)$ in C_n°.

4.4 The Cyclic-Based Algorithm For Half-Cyclic Convolution

In the Winograd-like algorithm, the computation is based on the half-circulant matrix. By the definition for half-circulant matrix in (1) of chapter 3, if we replace the elements a_i' by a_i, the half-circulant matrix becomes skew-circulant. We may consider to write the n-point half-circulant matrix H_n° as the sum of a n-point skew-circulant matrix H_n and one or more zero-half-circulant matrices. There are many ways to do this because the skew-circulant matrix H_n can be formed by either replacing a_i' by a_i or replacing a_i by a_i' for each i, $0 \leq i < n-1$. Obviously, the best way is to have the minimum replacements. Then the computation of H_n° can be carried out based on the algorithm for cyclic convolution and the algorithm on zero-half-circulant matrix. We will call this algorithm *Cyclic-based algorithm*, which is described in the following.

(i). If n is even, set $k = \frac{n}{2}$, otherwise set $k = \frac{n-1}{2}$. Let $k_0 = n-k-1$. For $0 \leq i < k$, replace a_i by a_i', for $k \leq i < n-1$, replace a_i' by a_i. It is easy to prove that by this way the number of replacements is minimum. Now the half-circulant matrix H_n° becomes the skew-circulant matrix

$$H_n = \begin{pmatrix} a_0' & \cdots & a_{k-1}' & a_k & \cdots & a_{n-1} \\ a_1' & \cdots & a_k & a_{k+1} & \cdots & a_0' \\ \vdots & & & & & \vdots \\ a_{n-1} & a_0' & \cdots & \cdots & \cdots & a_{n-2} \end{pmatrix}. \tag{40}$$

Set

$$H_n' = H_n^{\circ} - H_n. \tag{41}$$

The matrix H_n' has the form

$$H_n' = H_k^{\circ} \oplus 0 \oplus \widehat{H}_{k_0}^{\circ}, \tag{42}$$

where

$$H_k^{\circ} = \begin{pmatrix} b_0 & b_1 & \cdots & b_{k-1} \\ b_1 & b_2 & \cdots & 0 \\ \vdots & & & \vdots \\ b_{k-1} & 0 & \cdots & 0 \end{pmatrix}; \tag{43}$$

$$b_i = a_i - a_i', \qquad 0 \leq i < k,$$

is a k-point zero-half-circulant matrix, and

$$\widehat{H}^{\circ}_{k_0} = \begin{pmatrix} 0 & \cdots & 0 & b_k \\ 0 & \cdots & b_k & b_{k+1} \\ \vdots & & & \vdots \\ b_k & \cdots & b_{n-3} & b_{n-2} \end{pmatrix}; \qquad (44)$$
$$b_i = a'_i - a_i, \qquad k \le i < n-1,$$

which, by our definition, is not zero-half-circulant. But, observe that

$$\widehat{H}^{\circ}_{k_0} = \widehat{I}_{k_0} H^{\circ}_{k_0} \widehat{I}_{k_0}, \qquad (45)$$

where $H^{\circ}_{k_0}$ is a k_0-point zero-half-circulant matrix. Then, by the algorithm we introduced in the preceding sections, if

$$H^{\circ}_{k} = C^{\circ}_{k} G^{\circ}_{k} M^{\circ}_{k}; \qquad H^{\circ}_{k_0} = C^{\circ}_{k_0} G^{\circ}_{k_0} M^{\circ}_{k_0}, \qquad (46)$$

we have a factorization for H'_n,

$$H'_n = C^{\circ} G^{\circ} M^{\circ}, \qquad (47)$$

where

$$\begin{aligned} C^{\circ} &= (C^{\circ}_{k} \perp 0_{m^{\circ}_{k}}) \oplus \widehat{I}_{k_0} C^{\circ}_{k_0}; \\ M^{\circ} &= (M^{\circ}_{k} \vdash 0^{t}_{m^{\circ}_{k}}) \oplus M^{\circ}_{k_0} \widehat{I}_{k_0}; \\ G^{\circ} &= G^{\circ}_{k} \oplus G^{\circ}_{k_0}. \end{aligned} \qquad (48)$$

Especially, when n is odd, $k_0 = k$. If the skew-circulant matrix H_n is factorized by

$$H_n = C_n G_n M_n, \qquad (49)$$

then, we have the factorization for H°_n,

$$H^{\circ}_n = C^{\circ}_n G^{\circ}_n M^{\circ}_n, \qquad (50)$$

where

$$\begin{aligned} C^{\circ}_n &= C_n \vdash C^{\circ}; \\ M^{\circ}_n &= M_n \perp M^{\circ}; \\ G^{\circ}_n &= G_n \oplus G^{\circ}. \end{aligned} \qquad (51)$$

The arithmetic count

$$M^{\circ}(n) = M(n) + M^{\circ}(k) + M^{\circ}(k_0); \qquad (52)$$

$$A^{\circ}(n) \leq A(n) + A^{\circ}(k) + A^{\circ}(k_0) + n - 1. \tag{53}$$

4.5 Summary

In table 4-1, we list the arithmetic counts from 2 to 9 for both binary-recursion and base-2-recursion algorithms. When the size is even, the binary-recursion algorithm is better, while the base-2-recursion algorithm is better if the size is odd. This is because in base-2-recursion algorithm with even size, we treated the 2-point zero-half-circulant matrix h_{k-1} in formula (16) as half-circulant matrix. We may combine the two algorithms into one recursive procedure, when n is even, use binary-recursion while n is odd the base-2-recursion can be used. The arithmetic count in table 4-1 could be improved by doing so. The algorithm on zero-half-circulant matrix will uses about the same number of multiplications and about half number of additions than the Winograd-like algorithm with the same point.

Table 4-1

Size	Binary-recursion		Base-2-recursion	
k	$M^{\circ}(k)$	$A^{\circ}(k)$	$M^{\circ}(k)$	$A^{\circ}(k)$
2	3	1	3	1
3	5	4	5	4
4	9	7	9	7
5	12	13	11	12
6	15	19	15	21
7	19	24	19	24
8	27	31	27	33
9	32	43	29	38

The arithmetic counts from 2 to 9 for Cyclic-based algorithm has been listed in table 4-2. The expressions within the parenthesis means some common additions have been saved. The algorithm is not efficient than the Winograd-like algorithm. If the algorithm for cyclic convolution are improved the Cyclic-based algorithm can be improved also. It may be useful for some special applications.

Table 4-2

Size	Additive-half-cyclic	
k	$M^{\circ}(k)$	$A^{\circ}(k)$
2	3	5
3	6	13
4	9	19
5	16	37
6	16	42 (44-2)
7	26	80 (84-4)
8	28	62 (64-2)
9	37	93 (96-3)

CHAPTER 5

THE ALGORITHMS FOR Δ-HALF-CYCLIC CONVOLUTION AND THE ALGORITHMS ON THE MATRIX IN BLOCK FORM

5.1 Introduction

In this chapter, we will introduce the *Δ-Half-circulant matrix* and the corresponding algorithms with it. Using the symbol Δ for this case, the *n-point Δ-Half-circulant matrix* H_n^{Δ}, can be written as

$$H_n^{\Delta} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \\ a_1 & a_2 & \cdots & \Delta \cdot a_{n-1} & \Delta \cdot a_0 \\ \vdots & & & & \vdots \\ a_{n-2} & a_{n-1} & \cdots & \Delta \cdot a_{n-4} & \Delta \cdot a_{n-3} \\ a_{n-1} & \Delta \cdot a_0 & \cdots & \Delta \cdot a_{n-3} & \Delta \cdot a_{n-2} \end{pmatrix}, \quad \Delta \neq 0 \tag{1}$$

and the corresponding computation

$$\underline{y} = H^{\Delta}\underline{x} \tag{2}$$

is called the *Δ-Half-cyclic convolution*, where $\underline{x}$ and $\underline{y}$ are the input and output vectors.

In chapter 1, we have mentioned that the computation for cyclic convolution may contain the computation for the half-cyclic convolution including the Δ-half-cyclic convolution. For example, to compute the Winograd core $C(p)$, we may first block-diagonalize $C(p)$ by 2-point algorithm,

$$C(p) = (F(2) \otimes I_r)(D_0 \oplus D_1)(F(2) \otimes I_r); \qquad r = \frac{p-1}{2}, \tag{3}$$

where D_0 and D_1 are $r \times r$ blocks. As we know, the block D_0 is also skew-circulant, but D_1 is not. In fact, D_1 has the form

$$D_1 = \begin{pmatrix} a_0 & a_1 & \cdot & a_{r-1} \\ a_1 & a_2 & \cdots & -a_0 \\ \vdots & & & \vdots \\ a_{r-1} & -a_0 & \cdots & -a_{r-2} \end{pmatrix}, \tag{4}$$

which, according to our definition, is a Δ-half-circulant matrix with $\Delta = -1$. By the conjugate property of $C(p)$, D_0 is a real matrix and D_1 contains only pure imaginary entries. It is reasonable to develop the algorithm for Δ-half-cyclic convolution.

Certainly, the Δ-half-cyclic convolution can be carried out by the half-cyclic convolution algorithm, such as the Winograd-like algorithm. However, an important relation between Δ-half-circulant matrix and skew-circulant matrix has been found such that the computation for Δ-half-cyclic convolution can be carried out based on the algorithm for cyclic convolution, which is called Δ-*Theorem* in this text. Sometimes, the computation may have no extra cost.

If the size $n = s \cdot r$, by applying the Δ-theorem to the block form of a n-point Δ-half-circulant matrix, the computation can also be carried out by nesting the r-point half-cyclic convolution algorithm inside the s-point cyclic convolution algorithm, which we call the *Mixed-multiplicative algorithm.* In certain cases, it is much better than using the Δ-theorem on n-point directly.

There is another special case concerning with the block form of half-circulant matrix, which we have not mentioned yet. Consider the n-point half-circulant matrix with $s \times s$ block form in formula (38) of chapter 3. Instead of Δ, the blocks h'_i and h_i may have the relation

$$h'_i = S \cdot h_i, \tag{5}$$

where S is a constant matrix. Especially, if S is a circulant shift matrix, the matrix with this form is called $(s \times r)$-*point Shifted-Block half-circulant matrix.* The interesting thing is that the shifted-block half-circulant matrix can be changed into block circulant matrix if each of h_i is circulant, which is called the *Block-shift theorem.*

So far, the block forms of matrices have been frequently used in many places. We will make a brief discussion on the properties of the block forms with certain types and on the relations between the algorithms with the block forms.

5.2 Δ-Theorem

Theorem 5.1 For a n-point Δ-half-circulant matrix H_n^{Δ}, there exists a diagonal matrix T such that the matrix H_n generated by the products

$$H_n = T^{-1} H_n^{\Delta} T^{-1} \tag{6}$$

is a n-point skew-circulant matrix.

Proof Set

$$T = \bigoplus_{i=0}^{n-1} \delta_i; \qquad 0 \le i < n. \tag{7}$$

Denote by $H[i,j]$ the entries of the corresponding matrix H. By the definition (1), the entries of H_n^{Δ} can be written as

$$H_n^{\Delta}[i,j] = \Delta^{\lfloor \frac{i+j}{n} \rfloor} a_{i+j}, \tag{8}$$

where $i+j$ takes modulo n for the index of a_{i+j},

$$\lfloor \frac{i+j}{n} \rfloor = \begin{cases} 0 & \text{if} \quad i+j < n; \\ 1 & \text{if} \quad i+j \ge n, \end{cases}$$

and the entries of T^{-1},

$$T^{-1}[i,j] = \begin{cases} 0 & \text{if} \quad i \ne j; \\ \delta_i^{-1} & \text{if} \quad i = j. \end{cases}$$

Then, by direct computation,

$$\begin{aligned} (H_n^{\Delta} T^{-1})[i,j] &= \sum_{k=0}^{n-1} H_n^{\Delta}[i,k] \cdot T^{-1}[k,j] \\ &= \Delta^{\lfloor \frac{i+j}{n} \rfloor} \cdot \delta_i^{-1} \cdot a_{i+j}, \end{aligned}$$

$$\begin{aligned} H_n[i,j] &= \sum_{k=0}^{n-1} T^{-1}[i,k] \cdot (H_n^{\Delta} T^{-1})[k,j] \\ &= \Delta^{\lfloor \frac{i+j}{n} \rfloor} \cdot (\delta_i \delta_j)^{-1} \cdot a_{i+j}. \end{aligned} \tag{9}$$

Now, set

$$\delta_1 = \delta; \qquad \delta_i = \delta^i, \tag{10}$$

and fix

$$i + j \equiv k \bmod n, \tag{11}$$

then, the result in (9) with the entries of $i+j=k$ would be

$$H_n[i,j] = \begin{cases} \delta^{-k} a_k & \text{if } 0 \le i+j < n; \\ (\Delta \cdot \delta^{-n}) \cdot \delta^{-k} a_k & \text{if } n \le i+j < 2n-1. \end{cases} \tag{12}$$

The condition for H_n to be a skew-circulant matrix is that

$$\Delta \cdot \delta^{-n} = 1. \tag{13}$$

Then

$$\delta = \Delta^{\frac{1}{n}}; \qquad T = \bigoplus_{i=0}^{n-1} \delta^i, \tag{14}$$

is one of the solutions. The theorem is proved. In general, there are at least n solutions. And

$$\delta_i = b \cdot \delta^i; \qquad b \neq 0, \tag{15}$$

will also make H_n skew-circulant. The relation established by (6) is called Δ-*Theorem.*

By this theorem, the n-point Δ-half-cyclic convolution can be carried out by the n-point cyclic convolution plus multiplying two diagonal matrices,

$$H_n^{\Delta} = T H_n T. \tag{16}$$

In general, the computation with two Ts will need $2(n-1)$ multiplications. The arithmetic count can be roughly described by

$$\begin{aligned} M^{\Delta}(n) &= M(n) + 2(n-1); \\ A^{\Delta}(n) &= A(n). \end{aligned} \tag{17}$$

But this count may be modified based on the following different situations.

(i). H_n^{Δ} is complex and T is complex;

(ii). H_n^{Δ} is real and T contains real or pure imaginary entries;

(iii). H_n^{Δ} is complex and T contains real or pure imaginary entries;

(iv). H_n^{Δ} is real but T is complex.

For the situations from (i) to (iii), the Δ-theorem behaves like real-type algorithm, but in the situation (iv), the matrix H_n by (6) will become complex. In this case, we may prefer using the half-cyclic convolution algorithms or developing new algorithms.

Because the solution of T is not unique, by selecting the solution of δ carefully, the computation could be reduced. We prefer that the value of δ is either real or pure imaginary. The more special case is that

$\Delta = -1$. By (14), if n is an odd number then T consists of only ± 1 as its diagonal elements. The computation with T has no cost at all. We will take an example.

Example 1. Take $n = 3$ and $\Delta = -1$. Then by (14) and (7), we would take

$$\delta = -1; \qquad T = 1 \oplus -1 \oplus 1,$$

and by (6),

$$H_3 = \begin{pmatrix} a_0 & -a_1 & a_2 \\ -a_1 & a_2 & a_0 \\ a_2 & a_0 & -a_1 \end{pmatrix}$$

is a real skew-circulant matrix. Notice that in such case,

$$T^{-1} = T. \tag{18}$$

However, if n is an even number and H_n^{Δ} is real, then the computation falls into the situation (iv). We will give further discussion for this case in the following sections.

5.3 The Multiplicative Algorithms For Δ-Half-Cyclic Convolution

If $n = s \cdot r$, the n-point Δ-half-circulant matrix H_n^{Δ} can be written in the block form

$$H_n^{\Delta} = \begin{pmatrix} h_0 & h_1 & \cdots & h_{s-1} \\ h_1 & h_2 & \cdots & \Delta h_0 \\ \vdots & & & \vdots \\ h_{s-1} & \Delta h_0 & \cdots & \Delta h_{s-2} \end{pmatrix}, \tag{19}$$

which is a $s \times s$ block Δ-half-circulant matrix, but each block h_i is $r \times r$ half-circulant matrix. Applying the Δ-theorem to the block form in (19) with the size s, we have

$$H_n^{\Delta} = (T \otimes I_r) H'_{s,r} (T \otimes I_r), \tag{20}$$

where

$$T = \bigoplus_{i=0}^{s-1} \delta_i; \qquad \delta^s = \Delta, \tag{21}$$

and $H'_{s,r}$ is the block circulant matrix

$$H'_{s,r} = (T^{-1} \otimes I_r) H_n^{\Delta} (T^{-1} \otimes I_r)$$
$$= \begin{pmatrix} h_0 & \delta^{-1}h_1 & \cdot & \delta^{-(s-1)}h_{s-1} \\ \delta^{-1}h_1 & \delta^{-2}h_2 & \cdots & h_0 \\ \vdots & & & \vdots \\ \delta^{-(s-1)}h_{s-1} & h_0 & \cdots & \delta^{-(s-2)}h_{s-2} \end{pmatrix}, \tag{22}$$

where $\delta^{-i}h_i$ is also half-circulant. But notice that $H'_{s,r}$ is different from H_n in (6), it may be not skew-circulant. Block-diagonalize $H'_{s,r}$ by s-point cyclic convolution algorithm,

$$H'_{s,r} = (C_s \otimes I_r) G^{\circ} (M_s \otimes I_r), \tag{23}$$

where G° is a block diagonal matrix

$$G^{\circ} = \bigoplus_{i=0}^{m_s-1} G_i^{\circ}; \qquad m_s = M(s), \tag{24}$$

and each G_i° is a $r \times r$ half-circulant matrix. Using r-point half-cyclic convolution algorithm for each G_i°,

$$G_i^{\circ} = C_r^{\circ} G_i M_r^{\circ}, \tag{25}$$

then,

$$G^{\circ} = (I_{m_s} \otimes C_r^{\circ}) G (I_{m_s} \otimes M_r^{\circ}), \tag{26}$$

where

$$G = \bigoplus_{i=0}^{m_s-1} G_i. \tag{27}$$

Replacing (26) into (23), we have that

$$H'_{s,r} = (C_s \otimes C_r^{\circ}) G (M_s \otimes M_r^{\circ}). \tag{28}$$

Finally, by (20), we obtained

$$H_n^{\Delta} = (T \otimes I_r)(C_s \otimes C_r^{\circ}) G (M_s \otimes M_r^{\circ})(T \otimes I_r). \tag{29}$$

The arithmetic count in this case would be

$$\begin{aligned} M_{s,r}^{\Delta}(n) &= M(s)M^{\circ}(r) + 2(n-s); \\ A_{s,r}^{\Delta}(n) &= rA(s) + M(s)A^{\circ}(r). \end{aligned} \tag{30}$$

Here we apply the Δ-theorem with the block size s instead of n, the solution of δ may be much better. Because this algorithm combines both cyclic convolution and half-cyclic convolution algorithms, we call it *Mixed-multiplicative algorithm* for Δ-half-cyclic convolution, which is different from the method that applies the Δ-theorem first, then uses the multiplicative algorithm based on cyclic convolution.

Notice that the algorithm provides parallel structure. For the purpose of parallel or vector processing, we may prefer the flexibility. We may set $n = r \cdot s$, or $n = s_0 \cdot r_0$. In this case, the performance could be different from the factorization of $n = s \cdot r$. In general, the best way is to have the maximum size for cyclic convolution.

In addition, the algorithm has another version. If we make the stride permutations $P(n,r)$ and $P(n,s)$ on both sides of H_n^Δ in (1), then the matrix H_n^Δ will becomes a $r \times r$ block half-circulant matrix,

$$\begin{aligned} H_{r,s}^{\Delta'} &= P(n,r) H_n^\Delta P(n,s) \\ &= \begin{pmatrix} h_0^\Delta & h_1^\Delta & \cdots & h_{r-1} \\ h_1^\Delta & h_2^\Delta & \cdots & h_0^{\Delta'} \\ \vdots & & & \vdots \\ h_{r-1}^\Delta & h_0^{\Delta'} & \cdots & h_{r-2}^{\Delta'} \end{pmatrix}, \end{aligned} \tag{31}$$

where each h_i^Δ and $h_i^{\Delta'}$ is a $s \times s$ Δ-half-circulant matrix. First, use the r-point half-cyclic convolution algorithm on this block form,

$$H_{r,s}^{\Delta'} = (C_r^\circ \otimes I_s) G^\Delta (M_r^\circ \otimes I_s), \tag{32}$$

where

$$G^\Delta = \bigoplus_{i=0}^{m_r^\circ - 1} G_i^\Delta; \qquad m_r^\circ = M^\circ(r), \tag{33}$$

and each G_i^Δ is also Δ-half-circulant and keeps the same Δ as H_n^Δ. Second, apply the Δ-theorem to each G_i^Δ,

$$G_i^\Delta = T G_i' T, \tag{34}$$

where T is the same as in (21), and each G_i' is a $s \times s$ skew-circulant matrix. Now, by s-point cyclic convolution algorithm,

$$G_i' = C_s G_i M_s, \tag{35}$$

then, we have

$$G^{\Delta} = (I_{m_r^\circ} \otimes T \cdot C_s) G (I_{m_r^\circ} \otimes M_s \cdot T), \tag{36}$$

where

$$G = \bigoplus_{i=0}^{m_r^\circ - 1} G_i. \tag{37}$$

Finally, we obtained

$$H_n^{\Delta} = P(n,s)(I_r \otimes T)(C_r^\circ \otimes C_s) G (M_r^\circ \otimes M_s)(I_r \otimes T) P(n,r). \tag{38}$$

By this way, the arithmetic count becomes

$$\begin{aligned} M_{r,s}^{\Delta'} &= M^\circ(r) M(s) + 2(n-r); \\ A_{r,s}^{\Delta'} &= s A^\circ(r) + M^\circ(r) A(s). \end{aligned} \tag{39}$$

Notice that this way is different from the factorization of $n = r \cdot s$ without permutation, where H_n^{Δ} is $r \times r$ block skew-circulant matrix. Because the number of multiplications for half-cyclic convolution is more than that for cyclic convolution, usually, this way is not better than the first approach.

5.4 The Special Case of $\Delta = -1$

When $\Delta = -1$, the solution of δ will depend on the size n. If n is an odd number, the Δ-theorem can be directly used, and the performance is the same as the n-point cyclic convolution. but if n is an even number the situation is a little complicated. We will focus on this case in this section.

An even number n can be written as

$$n = s \cdot 2^k = s \cdot r; \qquad r = 2^k, 0 < k, \tag{40}$$

where s is an odd number.

(i). If $s = 1$, then $n = 2^k$. Because there is no real solution for δ, by the Δ-theorem the computation for H_n^{Δ} should be in complex field. If H_n^{Δ} is complex, usually, this way is better. But, if H_n^{Δ} is real, we have to make a comparison between the 2^k-point half-cyclic convolution

algorithm on real matrix and the Δ-theorem with the 2^k-point cyclic convolution algorithm on complex matrix. By the Convolution theorem, the 2^k-point cyclic convolution can be carried out by two 2^k-point FFTs plus the multiplications with a 2^k-point diagonal matrix. The arithmetic count of 2^k-point Cooley-Tukey algorithm [5] is given by

$$\begin{aligned} M_f(2^k) &= 2^{k+1}(k-3)+8; \\ A_f(2^k) &= 3\cdot 2^k(k-1)+4, \end{aligned} \tag{41}$$

which is calculated by real operations. The diagonal matrix, translated into real count, will need

$$\begin{aligned} M_d(2^k) &= 2^{k+2}; \\ A_d(2^k) &= 2^{k+1}. \end{aligned} \tag{42}$$

Then, the arithmetic count for 2^k-point complex cyclic convolution is the sum of (42) and (41) multiplied by 2,

$$\begin{aligned} M_c(2^k) &= 2^{k+2}(k-2)+16; \\ A_c(2^k) &= 2^{k+1}(3k-2)+8. \end{aligned} \tag{43}$$

For computing H_n^{Δ}, the extra multiplications with two diagonal matrices are needed. Because there are two elements 1 and i in T, the arithmetic count by real operations is

$$\begin{aligned} M_t(2^k) &= 2^{k+3}-16; \\ A_t(2^k) &= 2^{k+2}-8. \end{aligned} \tag{44}$$

Plus (43) and (44) together, the arithmetic count in total for computing H_n^{Δ} is

$$\begin{aligned} M^{\Delta}(2^k) &= k\cdot 2^{k+2}; \\ A^{\Delta}(2^k) &= 3k\cdot 2^{k+1}. \end{aligned} \tag{45}$$

Now, if we compute H_n^{Δ} by negative algorithm, then by (79) and (80) in chapter 3,

$$\begin{aligned} M^{\circ}(2^k) &= 3^k; \\ A^{\circ}(2^k) &= 3(3^k-2^k). \end{aligned} \tag{46}$$

If we take the parameter $R=1$, then,

$$\begin{aligned} &E^{\circ}(2^k)-E^{\Delta}(2^k) \\ &= (M^{\circ}(2^k)+A^{\circ}(2^k))-(M^{\Delta}(2^k)+a^{\Delta}(2^k)) \\ &= 4\cdot 3^k-2^k(10k+3). \end{aligned} \tag{47}$$

It is easy to figure out that the condition for (47) to be negative is that $k > 7$, which implies that when $k \leq 7$, the negative algorithm is better. If we take $R = 2$, the answer is the same. The range is enough to build a library for sequential purpose.

(ii). If $s > 1$, then take $\delta = -1$. No computation is necessary for the matrix T. The factorization (29) now can be rewritten as

$$H_n^{\Delta} = (T \cdot C_s \otimes C_r^{\circ}) G^{\circ} (M_s \cdot T \otimes M_r^{\circ}). \tag{48}$$

The arithmetic count for (48) is

$$\begin{aligned} M_{s,r}^{\Delta}(n) &= M(s)M^{\circ}(r); \\ A_{s,r}^{\Delta}(n) &= rA(s) + M(s)A^{\circ}(r). \end{aligned} \tag{49}$$

If the matrix H_n^{Δ} is real, the advantage of this algorithm is obvious.

If s is not prime, say $s = s_0 \cdot s_1$, where s_0 and s_1 must be odd numbers too. Set $r_0 = s_1 \cdot 2^k$, the factorization (40) can be changed by $n = s_0 \cdot r_0$. As we mentioned, we would keep the maximum size for cyclic convolution, certainly, the factorization (40) is the best choice.

We can also do the computation by the second approach described in the preceding section. Then, deleting $2(n - r)$ multiplications from (39), the arithmetic count now is

$$\begin{aligned} M_{r,s}^{\Delta'}(n) &= M^{\circ}(r)M(s) = M_{s,r}^{\Delta}; \\ A_{r,s}^{\Delta'}(n) &= sA^{\circ}(r) + M^{\circ}(r)A(s). \end{aligned} \tag{50}$$

We see that the number of multiplications is the same as the first approach, but the number of additions may be different. We can choose either of the two approaches by making a comparison between the formulas (49) and (50). Usually, the first approach, which uses the cyclic convolution first, is better than the second.

5.5 The Block-Shift Theorem

A $(s \times r)$-*point Shifted-block half-circulant matrix*, denoted by $H_{s,r}^{\flat}$, is defined as

$$H_{s,r}^{\flat} = \begin{pmatrix} h_0 & h_1 & \cdots & h_{s-1} \\ h_1 & h_2 & \cdots & S_r^k h_0 \\ \vdots & & & \vdots \\ h_{s-1} & S_r^k h_0 & \cdots & S_r^k h_{s-2} \end{pmatrix}, \tag{51}$$

where each h_i, in general, is a $r \times r$ half-circulant matrix, and S_r is the $r \times r$ shift circulant matrix, $S_r^k = (S_r)^k$. What we are interested in is that each h_i is circulant or skew-circulant. In such cases, the matrix $H_{s,r}^b$ can be changed into $s \times s$ block circulant by a theorem similar to the Δ-Theorem.

Theorem 5.2 For a $(s \times r)$-point shifted-block half-circulant matrix $H_{s,r}^b$, if each block h_i inside is a skew-circulant matrix and if there exists an integer e such that

$$e \cdot s \equiv k \bmod r, \tag{52}$$

then, there exist a block-diagonal matrix T_s,

$$T_s = \bigoplus_{i=0}^{s-1} S_r^{i \cdot e}, \tag{53}$$

such that the matrix

$$H_{s,r} = T_s^{-1} H_{s,r}^b T_s \tag{54}$$

is a $s \times s$ block skew-circulant matrix having $r \times r$ skew-circulant blocks.

Proof. It can be proved by a similar procedure to the proof for Δ-theorem. Here, we treat each $r \times r$ block as a single element. Set

$$T_s = \bigoplus_{i=0}^{s-1} S_r^{i \cdot k_0}. \tag{55}$$

Denote by $H[i,j]$ the block entries of the corresponding matrix in $s \times s$ block form. Then the entries of $H_{s,r}^b$ can be written as

$$H_{s,r}^b[i,j] = S_r^{k \cdot b} \cdot h_{i+j}, \tag{56}$$

where $i+j$ takes modulo s for the index of h_{i+j}, and

$$b = \begin{cases} 0 & \text{if} \quad i+j < s; \\ 1 & \text{if} \quad i+j \geq s. \end{cases} \tag{57}$$

Then, by direct computation,

$$H_{s,r}[i,j] = S_r^{-i \cdot k_0 + k \cdot b} \cdot h_{i+j} S_r^{j \cdot k_0}. \tag{58}$$

Because h_{i+j} is skew-circulant, by the property (65) in chapter 2, we have

$$h_{i+j}S_r^{j\cdot k_0} = S_r^{-j\cdot k_0}\cdot h_{i+j}, \tag{59}$$

the result in (58) now be $H_{s,r}[i,j] = S_r^{-(i+j)k_0+k\cdot b}\cdot h_{i+j}$.

$$H_{s,r}[i,j] = S_r^{-(i+j)k_0+k\cdot b}\cdot h_{i+j}. \tag{60}$$

Now, fix

$$i+j \equiv k' \bmod s, \tag{61}$$

then, the result in (60) with the entries of $i+j=k'$ would be

$$H_{s,r}[i,j] = \begin{cases} S_r^{-k'\cdot k_0}h_{k'} & \text{if } 0 \le i+j < s; \\ S_r^{-s\cdot k_0+k}\cdot S_r^{-k'\cdot k_0}\cdot h_{k'} & \text{if } s \le i+j < 2s-1. \end{cases} \tag{62}$$

The condition for $H_{s,r}$ to be a $s\times s$ block skew-circulant matrix is that

$$S_r^{-s\cdot k_0+k} = I_r, \tag{63}$$

which implies that

$$s\cdot k_0 \equiv k \bmod r, \tag{64}$$

where k_0 is the required integer e by the theorem.

For example, if s and r are relative prime, there must exists an integer k_0 which satisfies (64). The special case is that $k = s\cdot k_0$. Notice that the solution may be not unique in some cases. In the later chapters, we will see some applications of this theorem.

Instead of skew-circulant matrix, if each block h_i of $H^b_{s,r}$ in (51) is a general circulant matrix, a corollary of the theorem can be directly derived. Because for circulant matrix h_i, we have that

$$h_iS_r^j = S_r^jh_i. \tag{65}$$

If we replace the matrix T_s by T_s^{-1} and follow the proving procedure in the above, then the matrix

$$H_{s,r} = T^{-1}H^b_{s,r}T^{-1} \tag{66}$$

is a $s\times s$ block skew-circulant matrix having $r\times r$ circulant blocks.

5.6 The Block Form And Its Algorithm - A Summary

So far, the block form of matrices have been frequently used in many places. The multiplicative algorithm can be designed based on the types and the properties of the matrix in block format and the blocks inside.

If $n = sr$, a matrix can be broken into $s \times s$ block matrix having $r \times r$ blocks as we did many times. For half-circulant matrix, the type of the block form and the type of the blocks are all half-circulant, which are identical. For Δ-half-circulant matrix, as we see, the block form is Δ-half-circulant but the blocks inside are half-circulant. For Skew-circulant matrix, as we introduced in the first chapter, it has skew-circulant block form and the blocks have half-circulant type. Recall that in the mixed-multiplicative algorithm for Δ-half-cyclic convolution in the section 3 of this chapter, there is a matrix $H'_{s,r}$ in (20), which is also a block skew-circulant matrix with half-circulant blocks, but itself is not skew-circulant. Therefore, the skew-circulant matrix can be treated as a special case of $H'_{s,r}$. It reminds us that the circulant convolution can be carried out based on the algorithm for $H'_{s,r}$ that we discussed in this chapter.

The form of a matrix can be changed by using the stride permutation as we did by the second approach for mixed-multiplicative algorithm in this chapter. In general, the block form and the blocks will interchange their types. The computation procedure is also changed and the performance could be different. In addition, after the permutations, the resulting matrix may have a special form. For example, after the stride permutations, a skew-circulant matrix becomes a shifted-block half-circulant matrix with skew-circulant blocks, which may be further changed by the Block-shift theorem. We will make further discussion on it in the later chapters.

We have made a lot of discussions on the new concepts of convolutions and the corresponding algorithms as well as the relations between them. Now, we are ready to design the new algorithms for cyclic convolution and FFTs.

CHAPTER 6

NEW ALGORITHMS FOR THE PRIME CASE CYCLIC CONVOLUTION

6.1 Introduction

So far, there are two basic algorithms to do the cyclic convolution when the size is prime. As we already mentioned in the first chapter, when size is small the Winograd algorithm is very efficient for the real matrix, while the size is large or the matrix is complex we have to use the Convolution Theorem. If the circulant matrix is real, by the Convolution theorem the diagonal matrix will contain one real element and $p-1$ complex entries, for complex input the performance can be measured by

$$\begin{aligned} M(p) &= 2M_f(p) + 4p - 2; \\ A(p) &= 2A_f(p) + 2p - 2, \end{aligned} \tag{1}$$

where $M_f(p)$ and $A_f(p)$ are the arithmetic count by real operations for $F(p)$, but based on the complex input. If the circulant matrix is complex, there would be no big difference.

In addition, both algorithms can not provide any parallel structures. In this chapter, we will describe three new real-type algorithms for the prime case cyclic convolution. The first is constructed by $(\frac{p-1}{2})$-point Negative algorithm for half-cyclic convolution and 2-point cyclic convolution algorithm, which is called *Negative-based algorithm.* The other two are built up by the algorithm on zero-half-circulant matrix and the $(p \pm 1)$-point cyclic convolution algorithm. We call them $(p-1)$-*Algorithm* or $(p+1)$-*Algorithm* respectively. We continue using the skew-circulant matrix in the cyclic convolution.

6.2 The Negative-Based Algorithm

Recall the method we used to build the prime case Negative algorithm for half-cyclic convolution, a p-point skew-circulant matrix can be written as

$$H_p = \begin{pmatrix} a_0 & \underline{a}^t \\ \underline{a} & H^{\circ}_{p'} \end{pmatrix} = \begin{pmatrix} 0 & \\ & H^{\circ}_{p'} \end{pmatrix} + \begin{pmatrix} a_0 & \underline{a}^t \\ \underline{a} & 0(p') \end{pmatrix}, \tag{2}$$

where $\underline{a}$ is the vector formed by the first column of H_p excluding the first element a_0 and $H^{\diamond}_{p'}$ is a p'-point half-circulant matrix, $p' = p - 1$.

Set $s = \frac{p'}{2}$ and write $H^{\diamond}_{p'}$ in $s \times 2$ block format,

$$H^{\diamond}_{p'} = \begin{pmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,s-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,s-1} \\ h_{s-1,0} & h_{s-1,1} & \cdots & h_{s-1,s-1} \end{pmatrix}, \tag{3}$$

where each $h_{i,j}$ is 2×2 half-circulant matrix, and

$$h_{i,j} = \begin{pmatrix} a_{2(i+j+1)} & a_{2(i+j+1)+1} \\ a_{2(i+j+1)+1} & a_{2(i+j+2)} \end{pmatrix}; \qquad 0 \le i, j < s. \tag{4}$$

Here the index of a takes modulo p. We use this notation for convenience in the later proving.

Instead of using p'-point negative algorithm, we block-diagonalize $H^{\diamond}_{p'}$ by the s-point Negative algorithm. By (39) and (40) in chapter 3,

$$H^{\diamond}_{p'} = (C^{\diamond}_s \otimes I_2) G'(M^{\diamond}_s \otimes I_2), \tag{5}$$

where

$$\begin{aligned} (C^{\diamond}_s \otimes I_2) &= ((C'_s \vdash I_s) \otimes I_2) = (C'_s \otimes I_2) \vdash I_{2s}; \\ (M^{\diamond}_s \otimes I_2) &= ((M'_s \perp I_s) \otimes I_2) = (M'_s \otimes I_2) \perp I_{2s}, \end{aligned} \tag{6}$$

and use the new notation $G^{\diamond}_k$ instead of G'_i in (40) of chapter 1,

$$G' = \bigoplus_{k=0}^{m_s-1} G^{\diamond}_k = \Big(\bigoplus_{k=0}^{m'_s-1} G^{\diamond}_k\Big) + \Big(\bigoplus_{k=m'_s}^{m_s-1} G^{\diamond}_k\Big) = G^{\diamond}(1) \oplus G^{\diamond}(2);$$

$$m_s = M^{\diamond}(s), \qquad m'_s = m_s - s. \tag{7}$$

Each $G^{\diamond}_k$ is a 2-point half-circulant matrix. Applying 2-point Negative algorithm to each block in $G^{\diamond}(1)$, then we have

$$G^{\diamond}(1) = (I_s \otimes C^{\diamond}_2) G'(1)(I_s \otimes M^{\diamond}_2). \tag{8}$$

Now set

$$\begin{aligned} H'_{p'} &= (C'_s \otimes I_2) G^{\diamond}(1)(M'_s \otimes I_2) \\ &= (C'_s \otimes C^{\diamond}_2) G'(1)(M'_s \otimes M^{\diamond}_2). \end{aligned} \tag{9}$$

Then

$$H^{\diamond}_{p'} = H'_{p'} + G^{\diamond}(2), \tag{10}$$

where $G^{\circ}(2)$ is a $s \times s$ block diagonal matrix having 2×2 half-circulant blocks. By the property 2 in (62) of chapter 3, each block G_k° in $G^{\circ}(2)$ can be obtained by

$$G_{i+m_s'}^{\circ} = \sum_{j=0}^{s-1} h_{i,j} = \begin{pmatrix} b_i & c_i \\ c_i & b_i' \end{pmatrix}; \qquad 0 \le j < s, \tag{11}$$

where the elements b_i, b_i' and c_i can be obtained by (4),

$$\begin{aligned} b_i &= \sum_{j=0}^{s-1} a_{2(i+j+1)}; \\ c_i &= \sum_{j=0}^{s-1} a_{2(i+j+1)+1}; \\ b_i' &= \sum_{j=0}^{s-1} a_{2(i+j+2)}. \end{aligned} \tag{12}$$

Now set

$$\begin{aligned} h &= \begin{pmatrix} a_0 & \underline{a}^t \\ \underline{a} & 0(p') \end{pmatrix} + \begin{pmatrix} 0 & \\ & G^{\circ}(2) \end{pmatrix} \\ &= \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{p'-1} & a_{p'} \\ a_1 & b_0 & c_0 & \cdots & 0 & 0 \\ a_2 & c_0 & b_0' & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{p'-1} & 0 & 0 & \cdots & b_{s-1} & c_{s-1} \\ a_{p'} & 0 & 0 & \cdots & c_{s-1} & b_{s-1}' \end{pmatrix}. \end{aligned} \tag{13}$$

We see that h is similar to the form (22) of chapter 3, where we built up the prime case negative algorithm, which consists of p' 2-point half-circulant matrices. Here, $\underline{a}$, $\underline{a}^t$ and the diagonal elements of h also form p' 2-point half-circulant matrices. We can follow the procedure from (22) to (31) of chapter 3. Set

$$a = \sum_{l=0}^{p'} a_l; \qquad h' = \begin{pmatrix} a_0 - a & \underline{a}^t \\ \underline{a} & diag(-\underline{a}) \end{pmatrix}, \tag{14}$$

where $diag(-\underline{a})$ is $-\underline{a}$ in diagonal form. And set

$$\begin{aligned} G_i &= \begin{pmatrix} a_{2i+1} + b_i & c_i \\ c_i & a_{2i+2} + b_i' \end{pmatrix}; \\ G(2) &= \bigoplus_{i=0}^{s-1} G_i. \end{aligned} \tag{15}$$

Then we see that

$$h = h' + (0 \oplus G(2)), \tag{16}$$

where h' can be factorized by

$$h' = (-1_{p'} \perp I_{p'}) diag(\underline{a})(1_{p'} \vdash -I_{p'}). \tag{17}$$

Now we prove that each block G_i in $G(2)$ is a 2-point circulant matrix. From (15), for the block G_i to be circulant, we need to prove that

$$a_{2i+1} + b_i = a_{2i+2} + b'_i. \tag{18}$$

Because $p = 2s + 1$, by (12), we have

$$\begin{aligned} a_{2i+1} + b_i &= a_{2(i+s+1)} + \sum_{j=0}^{s-1} a_{2(i+j+1)} \\ &= \sum_{j=0}^{s} a_{2(i+j+1)}; \\ a_{2i+2} + b'_i &= a_{2(i-1+2)} + \sum_{j=0}^{s-1} a_{2(i+j+2)} \\ &= \sum_{j'=-1}^{s-1} a_{2(i+j+2)} \\ &= \sum_{j'=0}^{s} a_{2(i+j'+1)}, \end{aligned} \tag{19}$$

which proves the equation (18). Then, $G(2)$ can be computed by

$$G(2) = (I_s \otimes C_2) G'(2) (I_s \otimes M_2). \tag{20}$$

From (13)and (16), the skew-circulant matrix H_p in (2) now becomes

$$H_p = (0 \oplus H'_{p'})) + h = (0 \oplus H'_{p'}) + h' + (a \oplus G(2)). \tag{21}$$

Relating to the corresponding factorizations of (9), (17) and (20) for each submatrix in (21), set

$$\begin{aligned} C_p &= (0^t_{m'_s} \perp (C'_s \otimes C^\circ_2)) \vdash (-1_{p'} \perp I_{p'}) \vdash (1 \oplus (I_s \otimes C_2)); \\ M_p &= (0_{m'_s} \vdash (M'_s \otimes M^\circ_2)) \perp (1_{p'} \vdash -I_{p'}) \perp (1 \oplus (I_s \otimes M_2)); \\ G_p &= G^\circ(1) \oplus (diag(\underline{a})) \oplus a \oplus G'(2). \end{aligned}$$

(22)

Then

$$H_p = C_p G_p M_p. \tag{23}$$

In fact, we just replaced s 2-point half-cyclic convolutions by s 2-point cyclic convolutions in the p-point Negative algorithm. Then, s multiplications are decreased but s additions are increased to the count for p-point half-cyclic convolution. We have

$$\begin{aligned} M(p) &= M^{\circ}(p) - s; \\ A(p) &= A^{\circ}(p) + s. \end{aligned} \tag{24}$$

If we take $R = 1$, then the two algorithms have no different in the performance. The relationship between $A(p)$ and $M(p)$ can be found as follows,

$$\frac{A(p)}{M(p)} = \frac{3M^{\circ}(p) - 3p + s}{M(p) - s} < 3; \qquad p > 2. \tag{25}$$

Because the number of additions is increased and the number of multiplications is decreased than the Negative algorithm, we have another bound that

$$A(p) > 3(M(p) - p); \qquad p > 2. \tag{26}$$

By the winograd algorithm, we will see that the number of additions is even more and the number of multiplications is even less than this algorithm, the unequation (26) holds for general.

We call this algorithm *Negative-based algorithm.* In Table 6-1, we list the arithmetic counts of some sample points for both Winograd and Negative-based algorithm.

Table 6-1

Size	Winograd		Winograd-like	
k	$M(p)$	$A(p)$	$M(p)$	$A(p)$
3	4	11	5	10
5	10	31	12	29
7	16	70	22	57
11			48	131
13			61	168
17			90	251

6.3 The $(p-1)$-Algorithm

Sometimes, for a prime number p, the p'-point cyclic convolution algorithm may be very efficient. Recall the Cyclic-based algorithm for half-cyclic convolution that we introduced in chapter 4, where the algorithm is designed based on circulant matrix. Then, the idea and the techniques can be used here also.

Again, write H_p as the same form as (2) in the preceding section. But for the half-circulant matrix $H^{\diamond}_{p'}$, we will use the Cyclic-based algorithm. But notice that $H^{\diamond}_{p'}$ has special form than the general case. If we denote its entries by c_i and c'_i as usual for half-circulant matrix,

$$H^{\diamond}_{p'} = \begin{pmatrix} c_0 & c_1 & \cdots & c_{p'-1} \\ c_1 & c_2 & \cdots & c'_0 \\ \vdots & & & \vdots \\ c_{p'-1} & c'_0 & \cdots & c_{p'-2} \end{pmatrix}, \tag{27}$$

then, we can find that

$$c_i = a_{i+2}; \qquad c'_i = a_{i+1} = c_{i-1}; \qquad 0 \le i < p', \tag{28}$$

where the index of a takes modulo p.

Now we will follow the procedure from (40) to (51) of chapter 4. Because p' is even, set $k = \frac{p'}{2}$, $k_0 = k-1$. After the replacing procedure, $H^{\diamond}_{p'}$ can be written as the sum

$$H^{\diamond}_{p'} = H_{p'} + H'_{p'}, \tag{29}$$

where $H_{p'}$ is p'-point skew-circulant matrix. The vector formed by the elements in the first column of it, denoted by $\underline{c}$,

$$\begin{aligned} \underline{c}^t &= (c'_0, \cdots, c'_{k-1}, c_k, \cdots, c_{p'-1}) \\ &= (a_1, \cdots, a_k, a_{k+2}, \cdots, a_0). \end{aligned} \tag{30}$$

The element a_{k+1} is not in the matrix. Observe that the first k elements of both vectors $\underline{c}$ and $\underline{a}$ in (2) are the same. Set

$$\underline{b}^t = \underline{a}^t - \underline{c}^t = (0, \cdots, 0, b_k, \cdots, b_{p'-1}), \tag{31}$$

where

$$b_i = \begin{cases} 0 & \text{if} \quad 0 \le i < k; \\ a_{i+1} - a_{i+2} & \text{if} \quad k \le i \le p'. \end{cases} \tag{32}$$

Now, set

$$H_p(1) = \begin{pmatrix} 0 & \underline{c}^t \\ \underline{c} & H_{p'} \end{pmatrix}; \qquad H_p(2) = \begin{pmatrix} a_0 & \underline{b}^t \\ \underline{b} & H'_{p'} \end{pmatrix}. \tag{33}$$

Then, the skew-circulant matrix H_p in (2) can be rewritten as

$$H_p = H_p(1) + H_p(2). \tag{34}$$

We first take a look at $H_p(1)$. Set the matrix

$$A = (\alpha_{p'} \quad I_{p'}) = \alpha_{p'} \vdash I_{p'}. \tag{35}$$

Because $\underline{c}^t$ is the same as the first row of $H_{p'}$ and $\underline{c}$ is the same as the first column of $H_{p'}$, then

$$H_p(1) = A^t H_{p'} A. \tag{36}$$

Suppose that

$$H_{p'} = C_{p'} G_{p'} M_{p'}. \tag{37}$$

Then

$$H_p(1) = (A^t C_{p'}) G_{p'} (M_{p'} A), \tag{38}$$

which needs only one more addition than the p'-point cyclic convolution.

Now look at the matrix $H_p(2)$. Set

$$H'_p(2) = S_p^{-1} H_p(2) S_p. \tag{39}$$

By (33),

$$H'_p(2) = \begin{pmatrix} H'_{p'} & \underline{b} \\ \underline{b}^t & a_0 \end{pmatrix}. \tag{40}$$

The matrix $H'_{p'}$ was described from (42) to (45) in chapter 4,

$$H'_{p'} = H_k^{\circ} \oplus 0 \oplus \widehat{H}_{k_0}^{\circ}. \tag{41}$$

Because the first k elements of $\underline{b}$ are zeros, for convenience, we denote by $\underline{b}'$ the vector formed by the last k non-zero elements of $\underline{b}$. From (31), (32) and (27),

$$\begin{aligned} &(\underline{b}')^t = (b_k, \cdots, b_{p'-1}); \\ &\quad b_i = a_{i+1} - a_{i+2} = c'_i - c_i; \qquad k \le i < p'. \end{aligned} \tag{42}$$

Then, set

$$\widehat{H}^{\circ}_{k+1} = \begin{pmatrix} (0 \oplus \widehat{H}^{\circ}_{k_0}) & \underline{b}' \\ (\underline{b}')^t & a_0 \end{pmatrix}, \tag{43}$$

which, by the formula (44) of chapter 4, has the form

$$\widehat{H}^{\circ}_{k+1} = \begin{pmatrix} 0 & \cdots & 0 & b_{k+1} \\ 0 & \cdots & b_{k+1} & b_{k+2} \\ \vdots & & \cdot & \vdots \\ b_{k+1} & \cdots & b_{p-2} & a_0 \end{pmatrix}; \tag{44}$$

$$b_i = c'_i - c_i, \quad k \le i < p'.$$

We see that it is also a variant form of a $(k+1)$-point zero-half-circulant matrix. By setting

$$\widehat{H}^{\circ}_{k+1} = \widehat{I}_{k+1} H^{\circ}_{k+1} \widehat{I}_{k+1}, \tag{45}$$

we have that H°_{k+1} is a $(k+1)$-point zero-half-circulant matrix. Now by (39) and (41), we have that

$$H_p(2) = S_p H'_{p'}(2) S_p^{-1} = S_p (H^{\circ}_k \oplus H^{\circ}_{k+1}) S_p^{-1}. \tag{46}$$

By the k-point and the $(k+1)$-point algorithms on zero-half-circulant matrix, if

$$H^{\circ}_k = C^{\circ}_k G^{\circ}_k M^{\circ}_k; \qquad H^{\circ}_{k+1} = C^{\circ}_{k+1} G^{\circ}_{k+1} M^{\circ}_{k+1}, \tag{47}$$

then

$$\begin{aligned} H_p(2) &= S_p (C^{\circ}_k \oplus \widehat{I}_{k+1} C^{\circ}_{k+1})(G^{\circ}_k \oplus G^{\circ}_{k+1})(M^{\circ}_k \oplus M^{\circ}_{k+1} \widehat{I}_{k+1}) S_p^{-1} \\ &= (S_{k+1} \oplus \widehat{I}_k)(C^{\circ}_k \oplus C^{c}_{k+1})(G^{\circ}_k \oplus G^{\circ}_{k+1})(M^{\circ}_k \oplus M^{\circ}_{k+1})(S_{k+1}^{-1} \oplus \widehat{I}_k). \end{aligned} \tag{48}$$

Finally, by (34), (48) and the factorization (38) for $H_p(1)$, we have that

$$H_p = C_p G_p M_p, \tag{49}$$

where

$$\begin{aligned} C_p &= (A^t C_{p'}) \vdash ((S_{k+1} \oplus \widehat{I}_k)(C^{\circ}_k \oplus C^{\circ}_{k+1})); \\ M_p &= (M_{p'} A) \perp ((M^{\circ}_k \oplus M^{\circ}_{k+1})(S_{k+1}^{-1} \oplus \widehat{I}_k)); \\ G_p &= G_{p'} \oplus G^{\circ}_k \oplus G^{\circ}_{k+1}. \end{aligned} \tag{50}$$

The arithmetic count can be given by

$$M(p) = M(p') + M^{\circ}(k) + M^{\circ}(k+1); \tag{51}$$

$$A(p) \leq A(p') + A^{\circ}(k) + A^{\circ}(k+1) + p + 1. \tag{52}$$

6.4 The $(p+1)$-Algorithm

Instead of $(p-1)$, we may change H_p into $(p+1)$-point skew-circulant matrix if the $(p+1)$-point cyclic convolution algorithm is efficient. First, set

$$k = \frac{p'}{2}; \qquad p' = p-1; \qquad p_0 = p+1, \tag{53}$$

and define the vector of length p_0 by setting its elements

$$c_i = \begin{cases} a_{i-1} & \text{if} \quad 0 \leq i \leq k+1; \\ a_{i-2} & \text{if } k+1 < i < p_0 \end{cases} . \tag{54}$$

Then, the vector

$$\underline{c}^t = (a_{p'}, a_0, \cdots, a_k, a_k, \cdots, a_{p'-1}). \tag{55}$$

Notice that the two elements

$$c_{k+1} = c_{k+2} = a_k. \tag{56}$$

Build up the p_0-point skew-circulant matrix H_{p_0} by using $\underline{c}$ as its first column and write it as

$$H_{p_0} = \begin{pmatrix} a_{p'} & (\underline{c}')^t \\ \underline{c}' & H_p^{\diamond} \end{pmatrix}, \tag{57}$$

where the vector $\underline{c}'$ is formed by deleting the first element $a_{p'}$ from the vector $\underline{c}$, and $H_p^{\diamond}$ is the p-point half-circulant matrix,

$$H_p^{\diamond} = \begin{pmatrix} c_2 & c_3 & \cdots & c_0 \\ c_3 & c_4 & \cdots & c_1 \\ \vdots & & & \vdots \\ c_0 & c_1 & \cdots & c_{p'-1} \end{pmatrix}. \tag{58}$$

Now set

$$H'_{p_0} = \begin{pmatrix} a_{p'} & (\underline{c}')^t \\ \underline{c}' & H_p \end{pmatrix};$$
$$H''_{p_0} = H'_{p_0} - H_{p_0} = 0 \oplus (H_p - H_p^\circ). \tag{59}$$

Then, we have that

$$H'_{p_0} = H_{p_0} + H''_{p_0}. \tag{60}$$

If we set the matrix

$$B = 0_p \perp I_p, \tag{61}$$

then by (59) we can find that

$$H_p = B^t H'_{p_0} B. \tag{62}$$

Now look at the matrix H''_{p_0}. Denote the submatrix $H_p - H_p^\circ$ by H'_p. By (54) and (58), we have that

$$H'_p = H_k^\circ \oplus 0 \oplus 0 \oplus \widehat{H}_{k-1}^\circ, \tag{63}$$

where H_k° is the k-point zero-half-circulant matrix

$$H_k^\circ = \begin{pmatrix} b_0 & b_1 & \cdots & b_{k-1} \\ b_1 & b_2 & \cdots & 0 \\ \vdots & & & \vdots \\ b_{k-1} & 0 & \cdots & 0 \end{pmatrix};$$
$$b_i = a_i - c_{i+2} = a_i - a_{i+1}, \quad 0 \le i < k, \tag{64}$$

and $\widehat{H}_{k-1}^\circ$ is the variant $(k-1)$-point zero-half-circulant matrix

$$\widehat{H}_{k-1}^\circ = \begin{pmatrix} 0 & \cdots & 0 & b_{k+1} \\ 0 & \cdots & b_{k+1} & b_{k+2} \\ \vdots & & & \vdots \\ b_{k+1} & b_{k+2} & \cdots & b_{p'-2} \end{pmatrix};$$
$$b_i = a_{i+1} - a_i, \quad k \le i < p' - 1. \tag{65}$$

Then, the matrix H''_{p_0} in (59) can be written as

$$H''_{p_0} = 0 \oplus H_k^\circ \oplus 0 \oplus 0 \oplus \widehat{H}_{k-1}^\circ, \tag{66}$$

and can be factorized by

$$H''_{p_0} = C'_{p_0} G'_{p_0} M'_{p_0}, \tag{67}$$

where

$$
\begin{aligned}
C'_{p_0} &= (0^t_{m^\circ_k} \perp C^\circ_k) \oplus ((0_2 \otimes 0^t_{m^\circ_{k-1}}) \perp \widehat{I}_{k-1} C^\circ_{k-1}); \\
M'_{p_0} &= (0_{m^\circ_k} \vdash M^\circ_k) \oplus ((0_2 \otimes 0_{m^\circ_{k-1}}) \vdash M^\circ_{k-1} \widehat{I}_{k-1}); \\
G'_{p_0} &= G^\circ_k \oplus G^\circ_{k-1}; \quad m^\circ_k = M^\circ(k), \quad m^\circ_{k-1} = M^\circ(k-1).
\end{aligned} \tag{68}
$$

Finally, if the skew-circulant matrix H_{p_0} is factorized by

$$
H_{p_0} = C_{p_0} G_{p_0} M_{p_0}, \tag{69}
$$

then by (62), we obtained

$$
H_p = B^t (H_{p_0} + H''_{p_0}) B = C_p G_p M_p, \tag{70}
$$

where

$$
\begin{aligned}
C_p &= B^t (C_{p_0} \vdash C'_{p_0}) \\
&= B^t C_{p_0} \vdash (C^\circ_k \oplus ((0_2 \otimes 0^t_{m^\circ_{k-1}}) \perp \widehat{I}_{k-1} C^\circ_{k-1})); \\
M_p &= (M_{p_0} \perp M'_{p_0}) B \\
&= M_{p_0} B \perp (M^\circ_k \oplus ((0_2 \otimes 0_{m^\circ_{k-1}}) \vdash M^\circ_{k-1} \widehat{I}_{k-1})); \\
G_p &= G_{p_0} \oplus G'_{p_0} \\
&= G_{p_0} \oplus G^\circ_k \oplus G^\circ_{k-1}.
\end{aligned} \tag{71}
$$

The number of multiplications

$$
M(p) = M_{p_0} + M^\circ(k) + M^\circ(k-1). \tag{72}
$$

Because the first column of M_{p_0} and the first row of C_{p_0} are deleted, the number of additions

$$
A(p) < A(p_0) + A^\circ(k) + A^\circ(k-1) + 2k - 1. \tag{73}
$$

6.5 Summary

In this chapter, we have introduced three new algorithms for prime case cyclic convolution. The Negative-based algorithm can be applied to any size, and is not too difficult to build up. The $(p \pm 1)$-algorithm will depend on the size $(p \pm 1)$. If the $(p \pm 1)$-point cyclic convolution algorithm is very efficient such as the power's of 2, the $(p \pm 1)$-algorithm

is efficient. In the sense of performance, the Negative-based algorithm even covers a larger range than the Winograd algorithm. Also, because there is no big difference between the Negative-based algorithm for cyclic convolution and the Negative algorithm for half-cyclic convolution, we may use the later instead so that we can obtain a parallel structure in part for certain environment.

In Table 6-2, we list the arithmetic counts for $(p \pm 1)$-algorithm with some prime sizes. Compared with the arithmetic counts in Table 6-1 by the Winograd algorithm and the Negative-based algorithm, we see that the performance are better than the Negative-based algorithm and even better than the Winograd algorithm when the size is a little large. Because these two algorithms depend on the $(p \pm 1)$-point cyclic convolution algorithm, like the Cyclic-based algorithm for half-cyclic convolution, whenever the $(p \pm 1)$-point cyclic convolution algorithm is improved, so is the $(p \pm 1)$-algorithm for p-point. In Table 6-2, we use the traditional method for cyclic convolution. In the later chapters, we will develop new algorithms for cyclic convolution which is faster than the traditional way, the performance measured in Table 6-2 could be further improved.

Table 6-2

Size	$p \pm 1$	$(p-1)$-algorithm		$(p+1)$-Algorithm	
p	p_0 or p'	$M(p)$	$A(p)$	$M(p)$	$A(p)$
3	p-1=2	6	9		
5	p-1=4	13	26		
7	p+1=8			22	52
11	p+1=12			46	
13	p-1=12	54	148		
17	p-1=16	97	221		

The reason that we develop these real-type algorithms for prime case cyclic convolution comes from the following fact, if the computational matrix H_p is real and the size p is neither 'too small' nor 'too large', both Winograd algorithm and the Convolution theorem are not so efficient. By the Convolution theorem, we need two $F(p)$s plus the computation with a diagonal matrix of length p which is complex. Each $F(p)$, by the Rader's algorithm, consists of at least a p'-point cyclic convolution. By the Negative-based algorithm, the computation is based

on p'-point half-cyclic convolution plus the computation with a diagonal matrix of also length p but real. Roughly speaking, if the p'-point half-cyclic convolution algorithm is faster than twice the p'-point cyclic convolution algorithm on the Winograd core $C(p)$, the p-point Negative-based algorithm must be better than the Convolution theorem, so must the $(p \pm 1)$-algorithms for certain points. The Table 6-3 lists the arithmetic counts for both Convolution theorem and the real-type algorithms, where p from 3 to 17. The real-type algorithms here mean that we may use different algorithms for different size p. For $k \leq 7$ we use the Winograd algorithms for both $F(p)$ and convolution, and for $K > 7$, we use the reduced form of the Rader factorization for $F(p)$s and the Negative-based algorithm for prime case cyclic convolution. By the Convolution theorem, there is a real number in the diagonal matrix, which is also taken into the consideration. The performance were obtained by setting R with values 1, 2 and 3 respectively, where the parameter R is defined in the section 6 of chapter 2. We see that the performance can be improved about 30% on the average. In fact, the real-type algorithms are efficient for most of the prime sizes under 100.

Table 6-3

Size	Convolution Theorem		Real Type Algorithm		The Improvement (%)		
p	$M(p)$	$A(p)$	$M(p)$	$A(p)$	R=1	R=2	R=3
3	18	28	8	22	35	41	44
5	38	76	20	62	28	33	36
7	58	156	32	140	20	25	28
11	198	412	96	262	41	44	45
13	206	456	122	336	31	33	35
17	270	688	180	502	29	10	15
				Average (%)	28	24	34

CHAPTER 7

NEW ALGORITHMS FOR NON-PRIME CASE CYCLIC CONVOLUTION I – THE COMPLEX TYPE

7.1 Introduction

The basic tool of computing the cyclic convolution is the Convolution theorem, which we have introduced in chapter 1. When the size is large and the computational matrix is complex the Convolution theorem is efficient. But, when the size is very large, we may prefer parallel or vector processing. So far, the only multiplicative algorithm for cyclic convolution is the Agarwal-Cooley algorithm [14] for the relative prime case. If $n = s \cdot r$, for a n-point circulant matrix H_n, a permutation P can be found such that the matrix

$$H_{s,r} = PH_nP^{-1} = H_s \otimes H_r \tag{1}$$

is a block-circulant matrix of size s with circulant blocks of size r. Which turns 1-dimensional n-point cyclic convolution into 2-dimensional cyclic convolution. Applying the Convolution theorem to both circulant matrices H_s and H_r, then

$$H_n = P^{-1}(F(s) \otimes F(r))D(F(r)^{-1} \otimes F(s)^{-1})P. \tag{2}$$

Notice that here the circulant matrix H_n is not skew-circulant. If we apply this algorithm to the skew-circulant matrix, more permutations should be added.

Another way of obtaining a parallel structure is the use of the multiplicative algorithm for FFTs. By the Good-Thomas Prime Factor Algorithm (PFA) [24,25],

$$F(n) = (F(s) \otimes I_r)T(I_s \otimes F(r))P_0, \tag{3}$$

where P_0 is a permutation matrix and T is a diagonal matrix or twiddle factor. If s and r are relative prime, one form of the Good-Thomas PFA is given by the factorization

$$F(n) = Q_1(F(s) \otimes F(r))Q_2, \tag{4}$$

where Q_1 and Q_2 are permutation matrices. An obvious advantage of (4) is that the multiplications required in the twiddle factor stage of (2) are no longer necessary. Now, to compute a skew-circulant matrix H_n by using (3) or (4) instead $F(n)$ in the Convolution theorem, we have

$$H_n = (F(s) \otimes I_r)T(I_s \otimes F(r))P_0 D(F(s) \otimes I_r)T(I_s \otimes F(r))P_0, \tag{5}$$

or if s and r are relative prime,

$$H = Q_1(F(s) \otimes F(r))Q_2 D Q_1(F(s) \otimes F(r))Q_2. \tag{6}$$

We can see from (5) and (6) that the computation of H_n has some parallel features which are expressed by tensor products. However there are so many permutations involved in different stages of the computation, which will be much costly for parallel or vector processing. In fact, the formula (6) is a variant form of the Agarwal-Cooley algorithm in (2).

In this chapter, we will introduce a few new algorithms for non-prime case. For the relative prime case, an algorithm is designed based on the *Block-shift theory* described in chapter 5, which is called *Shifted-block algorithm.* This algorithm, in natural, is the same as the Agarwal-Cooley algorithm but working with the skew-circulant matrix, and the permutation can be implemented in parallel way. Another general algorithm has been designed based on both Convolution theorem and Δ-theorem, which we call the *Block-convolution algorithm.* By this algorithm, the computation structure is similar to the formula (5) but no permutation at all. When s and r are relative prime the result is the same as using the Block-shift theorem. We also use skew-circulant matrix in the discussion.

7.2 The Shifted-Block Algorithm

Given a n-point skew-circulant matrix H_n, if $n = s \cdot r$, by adding the stride permutations $P(n,s)$ and $P(n,r)$ to both sides of H_n, then the matrix

$$H^{\flat}_{s,r} = P(n,s)H_n P(n,r) = \begin{pmatrix} h_0 & h_1 & \cdots & h_{s-1} \\ h_1 & h_2 & \cdots & h'_0 \\ \vdots & & & \vdots \\ h_{s-1} & h'_0 & \cdots & h'_{s-2} \end{pmatrix} \tag{7}$$

is a s-point block half-circulant matrix. But each block h_i and h_i' here is a r-point skew-circulant matrix,

$$h_i = \begin{pmatrix} a_i & a_{s+i} & \cdots & a_{(r-1)s+i} \\ a_{s+i} & a_{2s+i} & \cdots & a_i \\ \vdots & & & \vdots \\ a_{(r-1)s+i} & a_i & \cdots & a_{(r-2)s+i} \end{pmatrix};$$
$$h_i' = \begin{pmatrix} a_{s+i} & a_{2s+i} & \cdots & a_i \\ a_{2s+i} & a_{3s+i} & \cdots & a_{s+i} \\ \vdots & & & \vdots \\ a_i & a_{s+i} & \cdots & a_{(r-1)s+i} \end{pmatrix}. \tag{8}$$

It is not difficult to find out the relationship between h_i and h_i',

$$h_i' = S_r^{-1} h_i, \tag{9}$$

which implies that if we replace each h_i' by $S_r^{-1}h_i$ in (7) the matrix $H_{s,r}^b$ becomes a $s \times r$-point shifted-block half-circulant matrix, which we introduced in chapter 5. Because each block h_i is skew-circulant, by the Block-shift theorem that we have proved in chapter 5, there may exist a block-diagonal matrix T_s such that the matrix

$$H_{s,r} = T_s^{-1} H_{s,r}^b T_s \tag{10}$$

is a s-point block skew-circulant matrix having r-point skew-circulant blocks. The condition is that there must exist an integer e such that

$$e \cdot s \equiv -1 \bmod r. \tag{11}$$

It is easy to find that if s and r are not relative prime there is no solution for e. If s and r are relative prime, by the Chinese Remainder theorem, there exists an integer e_1, called idempotent, such that

$$e_1 \equiv 0 \bmod s; \qquad e_1 \equiv 1 \bmod r. \tag{12}$$

Set

$$e = -\frac{e_1}{s}, \tag{13}$$

we obtained a solution. By the formula (53) of chapter 5, we have

$$T_s = \bigoplus_{i=0}^{s-1} S_r^{i \cdot e}. \tag{14}$$

Set

$$P = P(n,r)T_s,$$

then

$$T_s^{-1}P(n,s) = P^{-1}.$$

By (7) and (10), we have that

$$\begin{aligned} H_n &= P(n,r)H_{s,r}^{\flat}P(n,s) \\ &= P(n,r)T_sH_{s,r}T_s^{-1}P(n,s) \\ &= PH_{s,r}P^{-1}. \end{aligned} \tag{15}$$

Example 1. Take $n = 6 = 2 \times 3$. Then

$$H_{2,3}^{\flat} = P(6,2)H_6P(6,3) = \begin{pmatrix} h_0 & h_1 \\ h_1 & h_0' \end{pmatrix},$$

where

$$h_0 = \begin{pmatrix} a_0 & a_2 & a_4 \\ a_2 & a_4 & a_0 \\ a_4 & a_0 & a_2 \end{pmatrix}; \quad h_0' = \begin{pmatrix} a_2 & a_4 & a_0 \\ a_4 & a_0 & a_2 \\ a_0 & a_2 & a_4 \end{pmatrix} = S_3^{-1}h_0.$$

Because the integer e may be not unique, we would keep that $0 < e < r$ in the practical application. The result is really similar to the Agarwal-Cooley algorithm except that the matrix H_n here is skew-circulant and the permutation is different. The permutation in (15) consists of two steps, one is shift operation and the other is stride permutation. Both of them can be implemented easily. In fact, we can put them into one step.

In fact, the permutation $P(n,r)T_s$ in (15) has the similar structure to the CRT mapping used by Temperton in [26] for two-dimensional FFTs, where Temperton called it $Self-sorting$ and $In-place$. We can find that the permutation in our algorithm can be implemented by the same way.

Example 2. Consider the size $n = 42 = 7 \times 6$ and take the idempotent $e_1 = 7$, then $e = 7/7 = 1$. The permutation $P^{-1} = T_s^{-1}P(n,s)$ maps the 1-dimensional data into the 2-dimensional array

$$\begin{pmatrix} 0 & 7 & 14 & 21 & 28 & 35 \\ 36 & 1 & 8 & 15 & 22 & 29 \\ 30 & 37 & 2 & 9 & 16 & 23 \\ 24 & 31 & 38 & 3 & 10 & 17 \\ 18 & 25 & 32 & 39 & 4 & 11 \\ 12 & 19 & 26 & 33 & 40 & 5 \\ 6 & 13 & 20 & 27 & 34 & 41 \end{pmatrix}$$

which is the same as the mapping implemented by Temperton in [26].

More interesting, the permutation in (15), when combined with the permutation p_π in the Rader's algorithm for the prime case FFT, can also be implemented in parallel. To distinguish this algorithm from the Agarwal-Cooley algorithm, it is called *Shifted-block algorithm.*

7.3 The Block-Convolution Algorithm

In this section, we will consider the general case by using the Convolution theorem, where s and r are not necessary to be relative prime. Again, denote by h_i the r-point block entries of the skew-circulant matrix H_n. Then each half-circulant block h_i can be written as

$$h_i = \begin{pmatrix} b_{i,0} & b_{i,1} & \cdots & b_{i,r-1} \\ b_{i,1} & b_{i,2} & \cdots & b'_{i,0} \\ \vdots & & & \vdots \\ b_{i,r-1} & b'_{i,0} & \cdots & b'_{i,r-2} \end{pmatrix}; \qquad 0 \le i < s, \tag{16}$$

where

$$\begin{aligned} b_{i,j} &= a_{ir+j}; \qquad 0 \le j < r, \\ b'_{i,j} &= a_{(i+1)r+j}; \quad 0 < j < r-1. \end{aligned} \tag{17}$$

Applying the Convolution theorem to this s-point block skew-circulant form, we have

$$H_n = (F(s) \otimes I_r) D_b (F(s) \otimes I_r), \tag{18}$$

where

$$D_b = (F(s)^{-1} \otimes I_r) H_n (F(s)^{-1} \otimes I_r) = \bigoplus_{i=0}^{s-1} D_i^{\Delta} \tag{19}$$

is a block diagonal matrix, each block D_i^{Δ} is a r-point half-circulant matrix because each block h_i in H_n is half-circulant. Now, we prove that each D_i^{Δ} is also Δ-half-circulant. By direct computation on (19), we have that

$$D_i^{\Delta} = \frac{1}{r} \sum_{k=0}^{s-1} v^{-ik} \cdot h_k;$$

$$v = exp(-2\pi i/s), \qquad 0 \le i < r. \tag{20}$$

Denote by $c_{i,j}$ and $c'_{i,j}$ the entries for the corresponding half-circulant matrix D_i^{Δ}, as we did for each h_i. By (16), (17) and (20), then

$$\begin{aligned} c_{i,j} &= \frac{1}{s}\sum_{k=0}^{s-1} v^{-ik} b_{k,j} = \frac{1}{s}\sum_{k=0}^{s-1} v^{-ik} a_{kr+j}; \quad 0 \le j < r, \\ c'_{i,j} &= \frac{1}{s}\sum_{k=0}^{s-1} v^{-ik} b'_{k,j} = \frac{1}{s}\sum_{k=0}^{s-1} v^{-ik} a_{(k+1)r+j}; \quad 0 \le j < r-1. \end{aligned} \tag{21}$$

Set $k' = k+1$ and replace k by k' for the equation of $c'_{i,j}$ in (21), then

$$\begin{aligned} c'_{i,j} &= \frac{1}{s}\sum_{k'=1}^{s} v^{-i(k'-1)} a_{k'r+j} \\ &= \frac{v^i}{s}\left(\sum_{k'=1}^{s-1} v^{-ik'} a_{k'r+j} + v^{-is} a_{sr+j}\right) \\ &= \frac{v^i}{s}\sum_{k'=0}^{s-1} v^{-ik'} a_{k'r+j} \\ &= v^i c_{i,j}. \end{aligned} \tag{22}$$

Then, each D_i^{Δ} is a Δ-half-circulant matrix, where

$$\Delta_i = v^i. \tag{23}$$

By the Δ-theorem introduced in chapter 5, for each D_i^{Δ}, there is a diagonal matrix T_i such that

$$D_i^{\Delta} = T_i D'_i T_i, \tag{24}$$

where

$$D'_i = T_i^{-1} D_i^{\Delta} T_i^{-1} \tag{25}$$

is a skew-circulant matrix and T_i is a diagonal matrix,

$$T_i = \bigoplus_{j=0}^{r-1} \delta_i^j; \qquad \delta_i^r = \Delta_i. \tag{26}$$

If we set

$$\delta = \delta_1 = v^{\frac{1}{r}}, \tag{27}$$

then it is easy to find that

$$\delta_i = \delta^i; \qquad T_i = T_1^i = \bigoplus_{j=0}^{r-1} \delta^{ij}. \tag{28}$$

We see that when $i = 0$, then $T_0 = I_s$ and $D_0^{\Delta} = D_0'$ is always skew-circulant. Now, applying the Convolution theorem to each r-point skew-circulant block D_i', then

$$\begin{aligned} D_i' &= F(r) D_i F(r); \\ D_i &= F(r)^{-1} D_i' F(r)^{-1}. \end{aligned} \tag{29}$$

Combining (24) and (29), the block-diagonal matrix D_b in (19) can be factorized by

$$\begin{aligned} D_b &= \bigoplus_{i=0}^{s-1} (T_i F(r) D_i F(r) T_i) \\ &= T(I_r \otimes F(r)) D (I_r \otimes F(r)) T, \end{aligned} \tag{30}$$

where

$$T = \bigoplus_{i=0}^{s-1} T_i; \qquad D = \bigoplus_{i=0}^{s-1} D_i. \tag{31}$$

Putting (30) into (18), finally we obtained the factorization

$$H_n = (F(s) \otimes I_r) T (I_s \otimes F(r)) D (I_s \otimes F(r)) T (F(s) \otimes I_r). \tag{32}$$

The result is similar to the factorization (5), but there is no permutation at all in the factorization (32). So the factorization has very good features for parallel or vector processing.

If s and r are relative prime, we can modify the factorization (32) so that the result is the same as (15) where the Block-shift theorem is used. Again, take an idempotent e_1 of s and r,

$$e_1 \equiv 1 \bmod s; \qquad e_1 \equiv 0 \bmod r; \qquad e = e_1/r. \tag{33}$$

Then, by the equation (26),

$$\delta^r = \delta_1^r = v = v^{e_1} = v^{e \cdot r}.$$

Instead of the solution $\delta = v^{\frac{1}{r}}$ in (27), we can choose that

$$\delta = v^{\frac{e \cdot r}{r}} = v^e. \tag{34}$$

Set

$$T' = P(n,r)TP(n,s) \tag{35}$$

and observe that by (31) and (28) T' can be written as

$$T' = \bigoplus_{j=0}^{r-1} T'_j, \tag{36}$$

where

$$T'_1 = \bigoplus_{i=0}^{s-1} \delta^i = \bigoplus_{i=0}^{s-1} v^{ie}, \tag{37}$$

and

$$T'_j = (T'_1)^j. \tag{38}$$

Define the diagonal matrix t by

$$t = \bigoplus_{i=0}^{s-1} v^i. \tag{39}$$

Then the matrices T'_i can be written as

$$T'_1 = t^e; \qquad T'_i = t^{ei}. \tag{40}$$

Now look at the product $F(s)t$, it easy to prove that

$$F(s)t = S_s^{-1}F(s), \tag{41}$$

where S_s is the circulant shift matrix of order s. Then

$$t = F(s)^{-1}S_s^{-1}F(s); \qquad t^k = F(s)^{-1}S_s^{-k}F(s). \tag{42}$$

Which implies that

$$F(s)T'_j = F(s)t^{ej} = S_s^{-ej}F(s). \tag{43}$$

Set

$$T_r = \bigoplus_{j=0}^{r-1} S_s^{-je}. \tag{44}$$

Then we have that

$$\begin{aligned}(I_r \otimes F(s))T' &= \bigoplus_{j=0}^{r-1} F(s)T'_j \\ &= \bigoplus_{j=0}^{r-1} S_s^{-je} F(s) \\ &= T_s(I_s \otimes F(s)).\end{aligned} \tag{45}$$

By the Theorem 2.8 in chapter 2,

$$\begin{aligned}(F(s) \otimes I_r) &= P(n,s)(I_r \otimes F(s))P(n,r); \\ (I_s \otimes F(r)) &= P(n,s)(F(r) \otimes I_s)P(n,r).\end{aligned} \tag{46}$$

Based on the above procedure, we can obtain the following result,

$$\begin{aligned}&(F(s) \otimes I_r)T(I_s \otimes F(r)) \\ &= P(n,s)(I_r \otimes F(s))T'(F(r) \otimes I_s)P(n,r) \\ &= P(n,s)T_r(I_r \otimes F(s))(F(r) \otimes I_s)P(n,r) \\ &= P(n,s)T_r(F(r) \otimes F(s))P(n,r).\end{aligned} \tag{47}$$

Similarly,

$$(I_s \otimes F(r))T(F(s) \otimes I_r) = P(n,s)(F(r) \otimes F(s))T_r^{-1}P(n,s). \tag{48}$$

Setting

$$D' = P(n,r)DP(n,s); \qquad P = P(n,s)T_r, \tag{49}$$

where D' is also a diagonal matrix. Then the factorization in (32) now becomes

$$H_n = P(F(s) \otimes F(r))D'(F(s) \otimes F(r))P^{-1}, \tag{50}$$

which is the same as the result in (15) except that here we used the different order for the factorization.

7.4 Summary

Because most of the cyclic convolution involved in the FFT computation would be on the skew-circulant matrix, for the case that s and r are relative prime, the Block-shift theorem is more convenient than the Agarwal-Cooley algorithm. The permutation is very easy to be implemented as we discussed. Notice that the relative prime case algorithms can be used to build up the real-type algorithms also, which we will discuss in the next chapter.

In addition, when s and r are relative prime, by using the Convolution theorem, either one of the factorizations (50), or (32) which is for general case, can be used. The advantage of (50) is that it uses the permutation P instead of the computation T in (32). Sometimes, for the purpose of parallel or vector processing, we may prefer the factorization (32) which has no permutation at all. Also, recall that in the discussion for the Δ-theorem in chapter 5, the computation of T may cost nothing for some special cases, such as $\delta = \pm 1, \pm i$.

For the purpose of sequential processing, it is also possible to reduce the computation of T even if s and r are not relative prime. We will take an example.

Example 3. let $n = 24$ and chose $s = 6$ and $r = 4$. By (27), one of the values of δ is

$$\delta = v^{\frac{1}{6}}; \qquad v = exp(-2\pi i/4).$$

We see that 12 out of total 24 entries in T are ± 1 or $\pm i$. But if we take

$$\delta = v^{\frac{3}{2}},$$

which is also a solution because

$$\delta^6 = v^9 = v.$$

Now 18 entries in T become ± 1 or $\pm i$, which reduce the computation.

In general, to find an optional solution for

$$\delta^r = v; \qquad v = exp(-2\pi i/s), \tag{51}$$

we can use the following procedure. Denote the greatest common divider of s and r by

$$g = gcd(s, r). \tag{52}$$

If $g = s$ or $g = r$, we can not make any further improvement. Suppose

$$g < MIN(s, r), \tag{53}$$

then s and r can be written as

$$s = gs_0; \qquad r = gr_0. \tag{54}$$

Now r_0 and s are relative prime. Take an idempotent of s_0 and r,

$$e_1 \equiv 0 \bmod r_0; \qquad e_1 \equiv 1 \bmod s, \tag{55}$$

we have that

$$\delta^r = v = v^{e_1}; \qquad \delta = v^{\frac{e_1}{gr_0}} = v^{\frac{e}{g}}; \qquad e = e_1/r_0. \tag{56}$$

We see that if $g = r$ then $r_0 = 1$, formula (56) becomes (27). Also, if $g = 1$, which means that s and r are relative prime, then (56) becomes (34) as we discussed for relative prime case. Therefore the formula (56) is the optional solution for general case.

CHAPTER 8

NEW ALGORITHMS FOR NON-PRIME CASE CYCLIC CONVOLUTION II – THE REAL TYPE

8.1 Introduction

In the preceding chapter, we used the Convolution theorem to carry out the non-prime case cyclic convolution, we call it Complex-type algorithm. In this chapter, we will focus on the Real-type algorithms for the same case. As one of the real-type algorithm, the Winograd algorithm can solve some special cases such as 2's power. But it is only efficient within very limited range, and is complicated to be implemented when the size is a little large.

If $n = s \cdot r$, where s and r are relative prime, the n-point real-type algorithm can be build up based on s-point and r-point real-type algorithms by using the Agarwal-Cooley algorithm or the Shifted-block algorithm. But, permutations must be used, which will destroy the conjugate property of the Winograd core $C(p)$. For some special cases, such as $n = 2 \cdot r$ where r is an odd number, an algorithm has been designed with the same performance as the relative prime case algorithms but without permutation. Which combines the cyclic convolution algorithm and the Δ-theorem. Also, a general algorithm, where s and r are not necessarily relative prime, will be described, which uses both cyclic and half-cyclic convolution algorithms. When $n = 2^k$, the 2's power case, a 2^k-*Algorithm* is developed with at least the same performance as the Winograd algorithm. Finally, another algorithm will be introduced for the general case of $n = 2^k \cdot r$, which is even better than the relative prime case algorithms.

8.2 The Relative Prime Case Algorithm

If s and r are relative prime, by the Shifted-block theorem (or the Agarwal-Cooley algorithm), the n-point real-type algorithm on skew-circulant matrix (or circulant matrix) can be built up based on the s-point and r-point real-type algorithms,

$$H_n = P(C_s \otimes C_r)G(M_s \otimes M_r)P^{-1}, \tag{1}$$

where C_s, C_r and M_s, M_r are the post and pre addition matrices for s-point and r-point real-type algorithms, respectively. The arithmetic count

$$M_{s,r}(n) = M(s)M(r); \tag{2}$$

$$A_{s,r}(n) = rA(s) + M(s)A(r). \tag{3}$$

As we already discussed, for different factorization of n the performance may be different for implementations. Therefore, to obtain the 'best' result by these algorithms, we have to find the 'best' factorization of n so that the equations (2) and (3) are minimum. It means that

$$M(n) = MIN\{M_{s,r}(n) : n = s \cdot r, \quad (s,r) = 1\}; \tag{4}$$

$$A(n) = MIN\{A_{s,r}(n) : n = s \cdot r, \quad (s,r) = 1\}. \tag{5}$$

Obviously, if $n = r \cdot s$, then

$$M_{r,s}(n) = M(r)M(s) = M_{s,r}(n). \tag{6}$$

Otherwise, if $n = s_0 \cdot r_0$ and $(s_0, r_0) = 1$, then there must exist a number k such that either $s_0 = ks$ and $r = kr_0$ (or $s = ks_0$ and $r_0 = kr$), where k must be relative prime to both s (or s_0) and r_0 (or r), otherwise, s and r are not relative prime. If s_0-point and r-point algorithms are also built up by the relative prime case algorithms corresponding to the factorization $s_0 = k \cdot s$ and $r = kr_0$, then by (4) we have

$$M_{s_0,r_0}(n) = M(s)M(k)M(r_0) = M(s)M(r). \tag{7}$$

This proving procedure, in fact, is inductive. Then the result in (7) implies that the result in (4) is

$$M(n) = M(s)M(r). \tag{8}$$

In chapter 6, the unequation (26) for prime case holds for this case too. Suppose that both s and r are prime numbers, by (26) of chapter 6, we have

$$A(s) < 3(M(s) - s); \qquad A(r) < 3(M(r) - r). \tag{9}$$

Then by (3), we have

$$A_{s,r}(n) < 3r(M(s) - s) + 3M(s)(M(r) - r) = 3(M(n) - n). \tag{10}$$

It is easy to see that for any factorization of n the result is the same as (10).

The 'best' result can be obtained only depending on the formula (5), i.e, the number of additions required. Later we will find that the 'best' result obtained by the relative prime case algorithms may be not the best.

8.3 The Algorithm For The General Non-Prime Case

If $n = s \cdot r$, recall that the n-point skew-circulant matrix H_n can be written as the s-point block skew-circulant form with r-point half-circulant blocks. Block-diagonalize H_n by s-point real-type algorithm,

$$H_n = (C_s \otimes I_r)G^{\diamond}(M_s \otimes I_r), \tag{11}$$

where

$$G^{\diamond} = \bigoplus_{i=0}^{m_s-1} G_i^{\diamond}; \qquad m_s = M(s), \tag{12}$$

and each $G_i^{\diamond}$ is a r-point half-circulant matrix. Applying r-point half-cyclic convolution algorithm to each $G_i^{\diamond}$, then

$$G_i^{\diamond} = C_r^{\diamond} G_i M_r^{\diamond}. \tag{13}$$

Putting (13) into (12) and then into (11), finally we have the result

$$H_n = (C_s \otimes C_r^{\diamond})G(M_s \otimes M_r^{\diamond}), \tag{14}$$

where

$$G = \bigoplus_{i=0}^{m_s-1} G_i. \tag{15}$$

The arithmetic count now is

$$\begin{aligned} M_{s,r}(n) &= M(s)M^{\diamond}(r); \\ A_{s,r}(n) &= rA(s) + M(s)a^{\diamond}(r). \end{aligned} \tag{16}$$

If we take $n = r \cdot s$ in this case, unfortunately, in general,

$$M_{r,s} = M(r)M^{\circ}(s) \neq M_{s,r}. \tag{17}$$

Also, if we make the permutations $P(n,r)$ and $P(n,s)$ on both sides of H_n as we did in the discussion for the Block-shift theorem, the matrix $H^{\flat}_{r,s} = P(n,r)H_nP(n,s)$ is a r-point block skew-circulant matrix having s-point half-circulant blocks. Then, we can compute $H_{r,s}$ by applying the r-point half-cyclic convolution algorithm first, then using the s-point cyclic convolution algorithm on each block. By this way, the arithmetic count becomes

$$M^{\circ}_{r,s}(n) = M^{\circ}(r)M(s) = M_{s,r}(n); \tag{18}$$

$$A^{\circ}_{r,s}(n) = sA^{\circ}(r) + M^{\circ}(r)A(s). \tag{19}$$

We see that the number of multiplications are the same as the factorization of $n = s \cdot r$ by the first approach, but the number of additions is different. Because in general,

$$M^{\circ}(k) > M(k) \geq k. \tag{20}$$

Usually, $A^{\circ}_{r,s}(n)$ in (19) is greater than $A_{s,r}(n)$ in (16).

We are not going to make further discussion on the performance of this algorithm, because the advantage of this algorithm is its parallel structure. If H_n is real, we should make the comparison between this algorithm and the complex-type algorithm with the same parallel structure that we discussed in the preceding chapter. In many cases, this real-type algorithm is better than the complex-type algorithm. We will not discuss it in further detail.

Recall the Mixed-multiplicative algorithm for Δ-half-cyclic convolution that we introduced in chapter 5, where the n-point Δ-half-circulant matrix H^{Δ}_n can be changed into the matrix $H'_{s,r}$ which is a s-point block skew-circulant matrix with r-point half-circulant blocks. The procedure (from (20) to (28) of chapter 5) of computing $H'_{s,r}$, in fact, is general. The skew-circulant matrix H_n with the $s \times r$ block form is only a special case of $H'_{s,r}$ which is not skew-circulant on n-point. But the algorithm is the same on the block format. Because H_n is skew-circulant, some blocks G°_i of G° in (12) could be skew-circulant or Δ-half-circulant. The performance can be improved. The following algorithms will use this fact.

8.4 The Case of $n = 2^k$

For $n = 2^k$, set $s = 2$ and $r = 2^{k-1}$. Block-diagonalize H_n by 2-point cyclic convolution algorithm, we have that

$$H_n = (C_2 \otimes I_{2^{k-1}})(G_0 \oplus G_1)(M_2 \otimes I_{2^{k-1}}), \tag{21}$$

where G_0 is a 2^{k-1}-point skew-circulant matrix and G_1 is a 2^{k-1}-point Δ-half-circulant matrix with $\Delta = -1$. In section 5.4 of chapter 5, we have discussed the algorithms for the special case of $\Delta = -1$. Suppose that we use the Negative algorithm to compute G_1, then

$$G_1 = C^{\diamond}_{2^{k-1}} G^{\diamond} M^{\diamond}_{2^{k-1}}. \tag{22}$$

If G_0 is computed by 2^{k-1}-point cyclic convolution algorithm, i.e.,

$$G_0 = C_{2^{k-1}} G M_{2^{k-1}}, \tag{23}$$

then

$$H_n = C_n G_n M_n, \tag{24}$$

where

$$\begin{aligned} C_n &= (C_2 \otimes I_{2^{k-1}})(C_{2^{k-1}} \oplus C^{\diamond}_{2^{k-1}}); \\ M_n &= (M_{2^{k-1}} \oplus M^{\diamond}_{2^{k-1}})(M_2 \otimes I_{2^{k-1}}); \\ G_n &= G \oplus G^{\diamond}. \end{aligned} \tag{25}$$

Actually, the procedure here is recursive. The 2^{k-1}-point algorithm for cyclic convolution should be built up by 2-point algorithm also. Now we are going to prove the following properties of the 2^k-algorithm.

(i). The pre-addition and the post-addition matrices can be written in the product form,

$$\begin{aligned} C_{2^k} &= \prod_{i=1}^{k} ((C_2 \oplus (I_{t_i} \otimes C^{\diamond}_2)) \otimes I_{2^{k-i}}); \\ M_{2^k} &= \prod_{i=k}^{1} ((M_2 \oplus (I_{t_i} \otimes M^{\diamond}_2)) \otimes I_{2^{k-i}}), \end{aligned} \tag{26}$$

where $C^{\diamond}_2$ and $M^{\diamond}_2$ are the post and pre addition matrices of the 2-point Negative algorithm, and

$$t_i = \frac{1}{2}(3^{i-1} - 1). \tag{27}$$

Proof We will only prove the equation for C_n. When $k = 1$, then $t_1 = 0$ and $2^{k-1} = 1$, we see that (26) is true for C_2. Suppose that for $n = 2^{k-1}$ we have that

$$C_{2^{k-1}} = \prod_{i=1}^{k-1}((C_2 \oplus (I_{t_i} \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}});$$

$$t_i = \frac{1}{2}(3^{i-1} - 1). \tag{28}$$

Then for $n = 2^k$, by the 2^{k-1}-point negative algorithm in (81) of chapter 3, we have that

$$C_{2^{k-1}}^\diamond = \prod_{i=1}^{k-1}(I_{t_i'} \otimes C_2^\diamond \otimes I_{2^{k-i-1}});$$

$$t_i' = 3^{i-1}. \tag{29}$$

Because

$$t_i + t_i' = \frac{1}{2}(3^{i-1} - 1) + 3^{i-1} = \frac{1}{2}(3^i - 1) = t_{i+1}, \tag{30}$$

then

$$\begin{aligned}
&C_{2^{k-1}} \oplus C_{2^{k-1}}^\diamond \\
&= \left(\prod_{i=1}^{k-1}((C_2 \oplus (I_{t_i} \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}})\right) \oplus \left(\prod_{i=1}^{k-1}(I_{t_i'} \otimes C_2^\diamond \otimes I_{2^{k-i-1}})\right) \\
&= \prod_{i=1}^{k-1}\left(((C_2 \oplus (I_{t_i} \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}}) \oplus ((I_{t_i'} \otimes C_2^\diamond) \otimes I_{2^{k-i-1}})\right) \\
&= \prod_{i=1}^{k-1}\left((C_2 \oplus ((I_{t_i} \oplus I_{t_i'}) \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}}\right) \\
&= \prod_{i=1}^{k-1}((C_2 \oplus (I_{t_{i+1}} \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}}).
\end{aligned} \tag{31}$$

Putting the result (31) into (25), we can obtain that

$$\begin{aligned}
C_{2^k} &= (C_2 \otimes I_{2^{k-1}}) \cdot \prod_{i=1}^{k-1}((C_2 \oplus (I_{t_{i+1}} \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}}) \\
&= \prod_{i=1}^{k}((C_2 \oplus (I_{t_{i+1}} \otimes C_2^\diamond)) \otimes I_{2^{k-i-1}}).
\end{aligned} \tag{32}$$

The computation consists of n contiguous steps, and each step contains some forms of tensor products which can be implemented either by parallel ways or by loops in sequential.

(ii). If we denote by $c_{i,j}$ and $m_{i,j}$ the entries of the post-addition matrix C_n and the pre-addition matrix M_n, respectively, then

$$\begin{aligned}
\sum_{i=0}^{2^k-1} c_{i,j} &= \begin{cases} 2^k & \text{if } j=0, \\ 0 & \text{if } 0<j<M(2^k); \end{cases} \\
\sum_{j=0}^{2^k-1} m_{i,j} &= \begin{cases} 2^k & \text{if } i=0, \\ 0 & \text{if } 0<i<M(2^k). \end{cases}
\end{aligned} \tag{33}$$

Proof When $k=1$, (33) is true. Suppose that when $n=2^{k-1}$ the formula (33) is true. Then when $n=2^k$, by direct computation on (25) we have

$$\begin{aligned}
C_n &= (C_2 \otimes I_{2^{k-1}})(C_{2^{k-1}} \oplus C^{\diamond}_{2^{k-1}}) \\
&= \begin{pmatrix} C_{2^{k-1}} & C^{\diamond}_{2^{k-1}} \\ C_{2^{k-1}} & -C_{2^{k-1}} \end{pmatrix}.
\end{aligned} \tag{34}$$

The answer can be found straightforwardly. So does the answer for M_n.

(iii). The first element of G_n in (25), say, g_0, has the following format

$$g_0 = b \cdot \sum_{i=0}^{2^k-1} a_i; \qquad b = \frac{1}{2^k}, \tag{35}$$

and g_0 is the only element in G which has the format (35) with any value of b.

Proof When $k=1$, we have that

$$g_0 = \frac{1}{2}(a_0 + a_1). \tag{36}$$

Suppose that $n=2^{k-1}$, the property is true. Then when $n=2^k$, the matrix H_n can be written as the 2×2 block form,

$$H_n = \begin{pmatrix} h_0 & h_1 \\ h_1 & h_0 \end{pmatrix}. \tag{37}$$

By the property for 2-point algorithm on the block form, we have that

$$G_0 = \frac{1}{2}(h_0 + h_1). \tag{38}$$

Then by the assumption for $n = 2^{k-1}$ on G_0, the first element g_0 in G of (23) is the only one that satisfies the property. And by (25), g_0 is also the first element of G_n.

(iv). The arithmetic count can be calculated by the following functions of k,

$$M(2^k) = \frac{1}{2}(3^k + 1); \tag{39}$$

$$A(2^k) = \frac{1}{2}(2^{k+1} + 3^{k+1} - 5). \tag{40}$$

Proof When $k = 1$ we see that $M(2) = 2$ and $A(2) = 4$, which is true. Suppose that (39) and (40) are true for $n = 2^{k-1}$. Then for $n = 2^k$, By (25), the arithmetic count can be represented by the recursive functions,

$$M(2^k) = M(2^{k-1}) + M^\diamond(2^{k-1}); \tag{41}$$

$$A(2^k) = 2^{k-1}A(2) + A(2^{k-1}) + A^\diamond(2^{k-1}). \tag{42}$$

By the Negative algorithm, we have that

$$\begin{aligned} M^\diamond(2^{k-1}) &= 3^{k-1}; \\ A^\diamond(2^{k-1}) &= 3^k - 3 \cdot 2^{k-1}. \end{aligned} \tag{43}$$

Then

$$\begin{aligned} M(2^k) &= \frac{1}{2}(3^{k-1} + 1) + 3^{k-1} = \frac{1}{2}(3^k + 1); \\ A(2^k) &= 4 \cdot 2^{k-1} + \frac{1}{2}(2^k + 3^k - 5) + 3^k - 3 \cdot 2^{k-1} \\ &= \frac{1}{2}(2^{k+1} + 3^{k+1} - 5). \end{aligned} \tag{44}$$

In the above discussion, we took the factorization of $s = 2$ and $r = 2^{k-1}$. We may also set $s = 2^{k_1}$ and $r = 2^{k_2}$ instead, where $k = k_1 + k_2$. Thus,

$$H_n = (C_{2^{k_1}} \otimes I_{2^{k_2}}) G_b (M_{2^{k_1}} \otimes I_{2^{k_2}}). \tag{45}$$

For the purpose of parallel processing, we may use the general algorithm described in the section 3. But if we still want the optional result, we can use the above method to compute each 2^{k_2}-point block in G_b of (45), in another words, use cyclic convolution algorithm for the skew-circulant

matrix instead of half-cyclic convolution algorithm. By the property (iii) in the above, the first block G_0 in G_b

$$G_0 = \frac{1}{2^{k_1}} \sum_{i=0}^{2^{k_1}-1} h_i; \qquad G_b = G_0 \oplus G^\circ, \tag{46}$$

is the only skew-circulant matrix. Applying this 2^k-algorithm to G_0 and the negative algorithm to the rest blocks of G_b, we have that

$$\begin{aligned} C_n &= (C_{2^{k_1}} \otimes I_{2^{k_2}})(C_{2^{k_2}} \oplus (I_{m'} \otimes C^\circ_{2^{k_2}})); \\ M_n &= (M_{2^{k_2}} \oplus (I_{m'} \otimes M^\circ_{2^{k_2}}))(C_{2^{k_1}} \otimes I_{2^{k_2}}); \\ m' &= M(2^{k_1}) - 1. \end{aligned} \tag{47}$$

Now we prove that the performance by this way is the same as the measurement in (39) and (40).

Proof By (47), the arithmetic count

$$\begin{aligned} M(2^k) &= M(2^{k_2}) + (M(2^{k_1}) - 1)M^\circ(2^{k_2}) \\ &= \frac{1}{2}(3^{k_2} + 1) + \frac{1}{2}(3^{k_1} - 1) \cdot 3^{k_2} \\ &= \frac{1}{2}(3^k + 1) \end{aligned} \tag{48}$$

and

$$\begin{aligned} A(2^k) &= 2^{k_2} A(2^{k_1}) + A(2^{k_2}) + (M(2^{k_1}) - 1)A^\circ(2^{k_2}) \\ &= 2^{k_2-1}(2^{k_1+1} + 3^{k_1+1} - 5) + \frac{1}{2}(2^{k_2+1} + 3^{k_2+1} - 5) \\ &\quad + \frac{1}{2}(3^{k_1} - 1)(3^{k_2+1} - 3 \cdot 2^{k_2}) \\ &= \frac{1}{2}(2^{k+1} + 3^{k+1} - 5). \end{aligned} \tag{49}$$

Therefore, for the purpose of sequential implementation, any factorization of n has the same performance measured by (39) and (40). The relationship between $A(n)$ and $M(n)$ can also be figured out,

$$A(2^k) = 3M(2^k) + (2^k - 4), \tag{50}$$

and we can also prove that

$$2 \leq \frac{A(2^k)}{M(2^k)} < 4; \qquad 0 < k < \infty. \tag{51}$$

Because

$$A(2^k) - 3(M(2^k) - 2^k) = 2^{k+2} - 4 > 0,$$

so the unequation

$$A(2^k) > 3(M(2^k) - 2^k) \tag{52}$$

holds for this case too.

The algorithm is called 2^k-*Algorithm* in this text. In the table 8-1 we have listed the arithmetic counts for both Winograd algorithm and the new 2^k-algorithm, where k from 1 to 5. We can see that the new algorithm has the same performance as the Winograd algorithm (2, 4, 8) so far we have known.

Table 8-1

Size	Winograd-cyclic		New-cyclic	
$n = 2^k$ k	$M(2^k)$	$A(2^k)$	$M(2^k)$	$A(2^k)$
$2 = 2^1$	2	4	2	4
$4 = 2^2$	5	15	5	15
$8 = 2^3$	14	46	14	46
$16 = 2^4$			41	135
$32 = 2^5$			122	394

8.5 The Case of $n = 2 \cdot r$, Where r Is An Odd Number

For this case, block-diagonalize H_n by 2-point algorithm as we did in (21). Where the block G_0 is r-point skew-circulant matrix, but G_1 is r-point Δ-half-circulant matrix with $\Delta = -1$. Because r is odd number, by the Δ-theorem on this special case we discussed in chapter 5, we have that

$$G_1 = T_r C_r G^{\Delta} M_r T_r; \qquad T_r = \bigoplus_{i=0}^{r-1} (-1)^i. \tag{53}$$

Then

$$\begin{aligned} H_n =& (C_2 \otimes I_r)(I_r \oplus T_r)(I_2 \otimes C_r)G \\ & (I_2 \otimes M_r)(I_r \oplus T_r)(M_2 \otimes I_r). \end{aligned} \tag{54}$$

Because $(2, r) = 1$, the relative prime case algorithms can be used for this case too. The performance would be the same. But there is no permutation in the factorization (54). Simply, we call it $2 \cdot r$-*Algorithm.*

8.6 The Case of $n = 2^k \cdot r$, Where r Is An Odd Number

Now we are ready to build up the $2^k \cdot r$-*Algorithm* based on the preceding discussions. Again, block-diagonalizing H_n by 2-point algorithm as we did in (21) and setting $n' = 2^{k-1} \cdot r$, now the block G_0 in (21) is a n'-point skew-circulant matrix, which can be computed by the following recursive procedure. The block G_1 is a n'-point Δ-half-circulant matrix, where $\Delta = -1$. Set $s = 2^{k-1}$ and use the factorization $n' = r \cdot s$. Then G_1 can be written in $r \times r$ block Δ-half-circulant format. By the Mixed-multiplicative algorithm for this special case, which we introduced in section 4 of chapter 5, G_1 can be factorized by

$$G_1 = (T_r \otimes I_s) G^b_{r,s} (T_r \otimes I_s); \qquad T_r = \bigoplus_{i=0}^{r-1} (-1)^i, \tag{55}$$

where $G^b_{r,s}$ is r-point block skew-circulant matrix having s-point half-circulant blocks. Then

$$G^b_{r,s} = (C_r \otimes C^\circ_s) G'_1 (M_r \otimes M^\circ_s). \tag{56}$$

Placing (56) into (55), we have that

$$\begin{aligned} G_1 &= (T_r \otimes I_s)(C_r \otimes C^\circ_s) G'_1 (M_r \otimes M^\circ_s)(T_r \otimes I_s) \\ &= (T_r C_r \otimes C^\circ_s) G'_1 (M_r T_r \otimes M^\circ_s). \end{aligned} \tag{57}$$

Suppose that

$$G_0 = C_{n'} G'_0 M_{n'}. \tag{58}$$

Then by (21), we obtain the factorization

$$H_n = C_n G_n M_n,$$

where

$$\begin{aligned} C_n &= (C_2 \otimes I_{n'})(C_{n'} \oplus (T_r C_r \otimes C^\circ_s)); \\ M_n &= (M_{n'} \oplus (M_r T_r \otimes M^\circ_s))(M_2 \otimes I_{n'}); \\ G_n &= G'_0 \oplus G'_1. \end{aligned} \tag{59}$$

The arithmetic count can be obtained by the following recursive functions,

$$M(2^k r) = M(2^{k-1} r) + M^{\Delta}(2^{k-1} r); \tag{60}$$

$$A(2^k r) = 2^{k-1} r A(2) + A(2^{k-1} r) + A^{\Delta}(2^{k-1} r), \tag{61}$$

where $M^{\Delta}(2^{k-1}r)$ and $A^{\Delta}(2^{k-1}r)$ are the arithmetic count for $2^{k-1}r$-point Δ-cyclic convolution algorithm with the special case of $\Delta = -1$. The equations (60) and (61) can be computed by the following functions,

$$M(2^k \cdot r) = \frac{1}{2}(3^k + 1)M(r) = M(2^k)M(r); \tag{62}$$

$$A(2^k \cdot r) = \frac{3}{2}(3^k - 2^{k+1} + 1)M(r) + 2^k A(r) + 4r(2^k - 1). \tag{63}$$

Proof When $k = 1$, by the $2r$-algorithm, we have

$$\begin{aligned} M(2r) &= 2M(r); \\ A(2r) &= 2A(r) + 4r, \end{aligned} \tag{64}$$

which satisfies (62) and (63). Suppose that both (62) and (63) are true for $n = 2^{k-1} \cdot r$. Then when $n = 2^k \cdot r$, by (49) of chapter 5 and (87), (88) of chapter 3, we have that

$$\begin{aligned} M^{\Delta}(2^{k-1}r) &= M(r)M^{\circ}(2^{k-1}) = 3^{k-1}M(r); \\ A^{\Delta}(2^{k-1}r) &= 2^{k-1}A(r) + M(r)A^{\circ}(2^{k-1}) \\ &= 2^{k-1}A(r) + (3^k - 3 \cdot 2^{k-1})M(r). \end{aligned} \tag{65}$$

Then by (60) and (61) with our assumption,

$$\begin{aligned} M(2^k r) &= \frac{1}{2}(3^{k-1} + 1)M(r) + 3^{k-1}M(r) \\ &= \frac{1}{2}(3^k + 1)M(r) \\ &= M(2^k)M(r); \\ A(2^k r) &= 4r2^{k-1} + (\frac{3}{2}(3^{k-1} - 2^k + 1)M(r) + 2^{k-1}A(r) + \\ &\quad 4r(2^{k-1} - 1)) + (2^{k-1}A(r) + (3^k - 3 \cdot 2^{k-1})M(r)) \\ &= \frac{3}{2}(3^k - 2^{k+1} + 1)M(r) + 2^k A(r) + 4r(2^k - 1). \end{aligned} \tag{66}$$

The unequation

$$A(2^k r) > 3(M(2^k r) - 2^k r) \tag{67}$$

holds for this case too. Because

$$A(r) > 3(M(r) - r).$$

Then by (63),

$$\begin{aligned} A(2^k r) > \frac{3}{2}(3^k + 1)M(r) - 3r2^k + 4r(2^k - 1) = \\ 3(M(2^k r) - 2^k r) + 4r(2^k - 1). \end{aligned} \tag{68}$$

We call this algorithm $2^k \cdot r$-*Algorithm.* Now we can see that the 2^k-algorithm is only one of the special cases where $r = 1$, and the $2r$-algorithm is another special case of $k = 1$. Placing these values into the above functions in turn, we can obtain the arithmetic count for those special cases respectively.

Now we are going to prove that the $2^k \cdot r$-algorithm is better than the 'best' result obtained by the relative prime case algorithms. From the equation (62), we found that the number of multiplications is the same as the result derived by the relative prime case algorithms. But the number of additions will be saved a lot.

Proof By the function (5) in the discussion for the relative prime case algorithms, we have to make comparisons for all the relative prime factorizations of n. First, suppose that r is prime. The only two relative prime factorizations of n is $(2^k, r)$ and $(r, 2^k)$. To avoid confusing with the notations, we use $M(n)$ and $A(n)$ to stand for the arithmetic count by the $2^k \cdot r$-algorithm and use the notations with the index for the corresponding factorization by the relative prime case algorithms. The arithmetic count for 2^k case can be found in (39) and (40), we will not specify in each place where we use. For the factorization $(2^k, r)$, we have

$$\begin{aligned} & A_{2^k,r}(n) - A(n) \\ &= rA(2^k) + M(2^k)A(r) - \frac{3}{2}(3^k - 2^{k+1} + 1)M(r) \\ &\quad - 2^k A(r) - 4r(2^k - 1) \\ &= \frac{1}{2}(3^k - 2^{k+1} + 1)(A(r) - 3(M(r) - r)) \\ &\geq 0. \end{aligned} \tag{69}$$

Because $A(r) > 3(M(r) - r)$, which has been proved in (26) of chapter 6, and observe that $(3^k - 2^{k+1} + 1) > 0$ except that when $k = 1$ the result is 0. Then we proved that

$$A(n) \leq A_{2^k,r}(n). \tag{70}$$

Now consider the factorization $(r, 2^k)$,

$$\begin{aligned} & A_{r,2^k}(n) - A(n) \\ &= 2^k A(r) + M(r)A(2^k) - \frac{3}{2}(3^k - 2^{k+1} + 1)M(r) \\ &\quad - 2^k A(r) - 4r(2^k - 1) \\ &= 4(2^k - 1)(M(r) - r) \\ &> 0, \end{aligned} \tag{71}$$

because for any $r > 1$ we have that $M(r) - r > 0$.

If r is not prime, say, $r = r_0 r_1$, where r_0 and r_1 are relative prime. Obviously, both r_0 and r_1 are odd numbers. Again suppose that both r_0 and r_1 are prime numbers. Set $s_1 = 2^k r_0$. Then, two new factorizations of n would be (s_1, r_1) and (r_1, s_1), the expressions of both $A_{s_1,r_1}(n)$ and $A_{r_1,s_1}(n)$, by the relative prime case algorithm, will contain the positive items $A(s_1)$, which is also non-prime case, as we proved in the above,

$$A(s_1) \leq MIN\{A_{2^k,r_0}(s_1), A_{r_0,2^k}(s_1)\}, \tag{72}$$

which implies that if the s_1-point algorithm is built up by the relative prime case algorithms, then

$$A(n) \leq MIN\{A_{s_1,r_1}(n), A_{r_1,s_1}(n)\}. \tag{73}$$

By the inductive method, the result is true for general. Then we can conclude that

$$A(n) \leq MIN\{A_{s,r}(n) : n = sr, \quad (s,r) = 1\}. \tag{74}$$

We complete the whole proof.

There is a final problem. In the discussion for the computation of G_1 in (55), we used the first approach of the Mixed-multiplicative algorithm. Recall that there is the second approach we described in chapter 5, which makes the stride permutation on G_1 first. Now we prove that the first

approach is better than the second. We already proved that the number of multiplications are equal for both approaches. The difference of the numbers of the additions by the second approach and the first approach,

$$\begin{aligned} & A^{\Delta}_{2^k,r}(n) - A^{\Delta}_{r,2^k}(n) \\ &= (rA^{\circ}(2^k) + M^{\circ}(2^k)A(r)) - (2^kA(r) + M(r)A^{\circ}(2^k)) \\ &= (3^k - 2^k)(A(r) - 3(M(r) - r)) \\ &> 0, \end{aligned} \tag{75}$$

which implies that the first approach is better, so is the $2^k \cdot r$-algorithm we just introduced.

The table 8-2 lists the results for some sample points. Corresponding to the 'best' results obtained by the relative prime case algorithm, we see that when $k > 1$ a lot of additions can be saved by this new algorithm, which is more than 11% on average. Also, because there is no permutation at the beginning stage of the computation (in fact, a stride permutation implied by the tensor product), the conjugate property of the Winograd core $C(P)$ can be completely used.

Table 8-2

Size	Relative Prime Case			$2^k \cdot r$ case		
n	$s \times r$	M(n)	A(n)	$2 \times n'$	M(n)	A(n)
6	2×3	8	34	$2 \cdot 3$	8	34
10	2×5	20	82	$2 \cdot 5$	20	82
12	4×3	20	100	$2 \cdot 6$	20	92
14	2×7	32	168	$2 \cdot 3$	32	168
20	4×5	50	230	$2 \cdot 10$	50	212
24	3×8	56	272	$2 \cdot 12$	56	244
28	4×7	80	455	$2 \cdot 14$	80	412
40	8×5	140	664	$2 \cdot 20$	140	566
48	3×16	164	716	$2 \cdot 24$	164	656
56	7×8	224	1296	$2 \cdot 28$	224	1042

8.7 The Comparison Between Two Type Algorithms

We have introduced both complex-type and real-type algorithms for cyclic convolution of non-prime size. For a given n there are many

ways to build the n-point algorithm. The analysis on the performance will also depend on at least the following two factors, the environment of implementation and the field of the computational matrix. We already assumed that all the input is complex. In this section, we will make a comparison between the two types of algorithms briefly.

(i). For Parallel Processing.

Because the Descrete Fourier transform is square matrix the data flow in the complex algorithm is well-organized. In the real type algorithm, the pre-addition matrix and the post-addition matrix are not the same, we need more subroutines if we want to do so. The analysis on the performance for parallel or vector processing is quite complicated. In general, when the size is very large, the complex-type algorithm is efficient even if the computational matrix is real. The large size is also good for parallel or vector processing. The new algorithms we introduced in the preceding chapter also provide a flexibility for this purpose. Some real-type algorithms, such as the algorithm discussed in section 3 of this chapter, has parallel feature too. However, as we will prove, when the size is very large the algorithm is not efficient than the complex-type algorithm even if the computational matrix is real. For small size, the benefit from the parallel processing is trivial. A disadvantage of the complex-type algorithm is that the multiplications are distributed in different steps while the real type algorithm collect all the multiplications in only one step which is better for long vector processing.

(ii). For Sequential Processing

If the matrix H_n is complex, in general, the complex-type algorithm is efficient except for $n \leq 3$. We will focus on the real matrix H_n. Denote by $M_f(n)$ and $A_f(n)$ the real arithmetic count. To compute the n-point cyclic convolution on real H_n, the Convolution theorem will use two $F(n)$s plus the multiplication with a n-point complex diagonal matrix. Take into the consideration that there are two real elements in the diagonal matrix, then the arithmetic count in real operations with the complex input, denoted by $M_c(n)$ and $A_c(n)$, is

$$\begin{aligned} M_c(n) &= 2 \cdot M_f(n) + 4n - 4; \\ A_c(n) &= 2 \cdot A_f(n) + 2n - 4. \end{aligned} \tag{76}$$

By the real type algorithm we only simply double the count for real

input, i.e.,

$$\begin{aligned} M_r(n) &= 2 \cdot M(n); \\ A_r(n) &= 2 \cdot A(n). \end{aligned} \tag{77}$$

The performance can also be represented by one parameter, that is,

$$E(n) = A(n) + RM(n).$$

The improvement can be measured by

$$E_n = \frac{E_c(n) - E_r(n)}{E_c(n)}, \tag{78}$$

where $E_c(n)$ and $E_r(n)$ stand for the performance of complex-type and real-type algorithms respectively. We will discuss several cases in a little detail as follows.

a). The Case of $n = 2^k$.

By the Cooley-Tukey radix-two algorithm,

$$\begin{aligned} M_f(2^k) &= 2^{k+1}(k-3) + 8; \\ A_f(2^k) &= 3 \cdot 2^k(k-1) + 4. \end{aligned} \tag{79}$$

Then,

$$\begin{aligned} M_r(2^k) &= 2^{k+2}(k-3) + 16 + 2^{k+2} = 2^{k+2}(k-2) + 16; \\ A_r(2^k) &= 3 \cdot 2^{k+1}(k-1) + 8 + 2^{k+1} = 2^{k+1}(3k-2) + 8. \end{aligned} \tag{80}$$

By the 2^k algorithm, double (39) and (40), we have

$$\begin{aligned} M_r(2^k) &= 3^k + 1; \\ A_r(2^k) &= 2^{k+1} + 3^{k+1} - 5. \end{aligned} \tag{81}$$

Then

$$\begin{aligned} E_c(2^k) &= (2^{k+1}(3k-2) + 8) + R(2^{k+2}(k-2) + 16); \\ E_r(2^k) &= (2^{k+1} + 3^{k+1} - 5) + R(3^k + 1). \end{aligned} \tag{82}$$

Assume that $R = 1$, we have

$$E_{2^k} = E_r(2^k) - E_c(2^k) = 2 \cdot 3^k - 2^k(5k-7) - 14. \tag{83}$$

To determine the value of k so that $E_r(2^k) \leq E_c(2^k)$, we have to find the solution to the unequation

$$2 \cdot 3^k - 2^k(5k-7) - 14 \leq 0. \tag{84}$$

The answer is $k \leq 6$. We have checked that the result is the same for $R \leq 3$ which is the general value on most machines. The table 8-3 gives the real arithmetic counts for both complex-type and real-type algorithms with the values of k up to 6. When $k \leq 4$ the Winograd small FFT algorithms are used. The improvement E_n corresponds to each of $R = 1, 2, 3$. The improvement is about 20% on average.

Table 8-3

Size	Convolution theorem		2^k case algorithm		E_n (%)		
$n = 2^k$	$M_c(n)$	$A_c(n)$	$M_r(n)$	$A_r(n)$	R=1	R=2	R=3
4	12	36	10	30	17	17	17
8	36	116	28	92	21	21	22
16	100	324	82	270	17	17	17
32	396	836	244	788	16	22	25
64	1036	2052	730	2310	2	8	13
				Average E_n (%)	15	17	19

b). The Case of $n = 2^k \cdot r$

There is no general conclusion for this case. The table 8-4 lists the results for some sample points. The $F(n)$ is generated by the factorization of $2^k \times r$. The improvement E_n is about 21% on average.

Table 8-4

Size	Convolution Theorem		$2^k \cdot r$ Algorithm		E_n (%)		
$n = 2^k r$	$M_c(n)$	$A_c(n)$	$M_r(n)$	$A_r(n)$	R=1	R=2	R=3
6	36	80	16	68	28	34	38
10	76	192	40	164	24	29	32
12	76	212	40	184	22	28	31
14	116	368	64	336	17	23	26
20	156	468	100	424	16	20	23
24	180	548	112	488	18	22	24
28	236	852	160	824	10	14	17
40	356	1140	280	1132	6	9	11
48	436	1364	328	1312	9	12	14
	Average	E_n	(%)		17	21	24

From the above tables, including Table 6-3 of chapter 6 for prime case, we can see that the improvement is more than 20% on the average. In particular, the sample points we used cover most of the important points for the library, which can be used to build the large size algorithms. In the later chapters we will find the applications of these new algorithms, especially for computing the Winograd core $C(p)$ and the similar matrices in 2-dimensional FFTs and symmetrized FFTs.

CHAPTER 9

NEW ALGORITHMS FOR ONE DIMENSIONAL PRIME CASE FFT

9.1 Introduction

As building blocks, the algorithms for prime size are very important. The basic idea is the Rader's algorithm, which we already introduced from (5) to (9) in chapter 1,

$$F(p) = P_\pi^{-1} F_\pi P_\pi = P_\pi^{-1}(1 \oplus C(p))A(p)P_\pi, \tag{1}$$

where the Winograd core

$$C(p) = \left[v^{z^{l+k}}\right]_{0 \le l,k < p'}; \quad v = -exp(2\pi i/p), \tag{2}$$

is p'-point skew-circulant matrix, $p' = p - 1$. And

$$A(p) = \begin{pmatrix} 1 & 1_{p'}^t \\ -1_{p'} & I_{p'} \end{pmatrix}. \tag{3}$$

The cyclic convolution algorithm of computing $C(p)$ is the key to solve the $F(p)$. As we discussed, the matrix $C(p)$ is complex but has the conjugate property. By the traditional real-type algorithms, the conjugate property can not be fully used. In general, $C(p)$ is computed by Convolution theorem [27]. A variant form of the Rader's algorithm [13] has been derived to reduce the number of additions of $A(p)$ by using the Convolution theorem. Let

$$\begin{aligned} C(p) &= F(p')D_{p'}F(p'); \\ B(p) &= \begin{pmatrix} 1 & \alpha_{p'}^t \\ -p' \cdot \alpha_{p'} & I_{p'} \end{pmatrix}. \end{aligned} \tag{4}$$

Then,

$$F(p) = P_\pi^{-1} FDB(p)FP_\pi, \tag{5}$$

where

$$F = (1 \oplus F(p'); \qquad D = (1 \oplus D_{p'}). \tag{6}$$

Now the matrix $B(p)$ needs only 2 additions plus 1 rational multiplication which is less than $2p'$ additions for computing $A(p)$.

We have discussed the cyclic convolution algorithms for many different cases in the preceding chapters. The traditional algorithms can neither provide good features for parallel processing nor provide good performance for sequential processing in many cases. First, look at the Winograd core $C(p)$, it has the following properties which may be useful.

(i). For the element a_i in $C(p)$, we have that

$$\sum_{i=0}^{p'} a_i = -1, \tag{7}$$

which implies that the sum in each column or each row is -1.

(ii). The conjugate property which we already mentioned,

$$a_{i+\frac{p'}{2}} = a_i^*. \tag{8}$$

(iii). The size of $C(p)$ can be written as

$$p' = 2^k \cdot r, \tag{9}$$

where r is an odd number except $p = 2$.

The first property has been used in deriving the variant form (5). Unfortunately, the other two properties have not been used sufficiently. By the new algorithms we developed for cyclic convolution in the previous chapters, the Winograd core $C(p)$ can be computed efficiently for both sequential and parallel processing. The new algorithms for prime case FFT then can be designed.

9.2 Computing $C(p)$ By Complex-Type Algorithms

Let $p' = s \cdot r$. By the formula (32) in Chapter 7,

$$\begin{aligned} C(p) = (F(s) \otimes I_r)T(I_s \otimes F(r))D_{p'} \\ (I_s \otimes F(r))T(F(s) \otimes I_r), \end{aligned} \tag{10}$$

where $D_{p'}$ and T are diagonal matrices,

$$\begin{aligned} D_{p'} = (F^{-1}(s) \otimes I_r)T^{-1}(I_s \otimes F^{-1}(r))C(p) \\ (I_s \otimes F^{-1}(r))T^{-1}(F^{-1}(s) \otimes I_r) \end{aligned} \tag{11}$$

and

$$T = \bigoplus_{i=0}^{s-1} T_i; \qquad T_i = \bigoplus_{j=0}^{r-1} \delta^{ij}. \tag{12}$$

The value of δ can be obtained by finding a solution for the equation

$$\delta^r = w; \qquad w = exp(-2\pi i/s). \tag{13}$$

The procedure of finding an optional solution is described from (51) to (56) of chapter 7. Simply, we can take

$$\delta = \omega; \qquad \omega = exp(-2\pi i/p'). \tag{14}$$

Now, a new form of the Rader's algorithm is

$$F(p) = P_\pi^{-1} F_{s,r} D F_{s,r} P_\pi, \tag{15}$$

where

$$\begin{aligned} F_{s,r} &= (1 \oplus (F(s) \otimes I_r) T (I_s \otimes F(r)))\,; \\ D &= 1 \oplus D_{p'}. \end{aligned} \tag{16}$$

If s and r are relative prime, as we discussed in chapter 7, $C(p)$ can be factorized as

$$C(p) = P(F(s) \otimes F(r)) D'_{p'} (F(s) \otimes F(r)) P^{-1}, \tag{17}$$

where $P = P(p', r)T_s$ is a permutation matrix and

$$D'_{p'} = (F^{-1}(s) \otimes F^{-1}(r)) P^{-1} C(p) P (F^{-1}(s) \otimes F^{-1}(r)). \tag{18}$$

Then, the corresponding factorization of $F(p)$ is

$$F(p) = P_0 F'_{s,r} D F'_{s,r} P_0, \tag{19}$$

where P_0 is the permutation matrix,

$$P_0 = P_\pi^{-1}(1 \oplus P) = 1 \oplus P_\pi^{-1} P, \tag{20}$$

and

$$\begin{aligned} F(s, r' &= (1 \oplus P(F(s) \otimes F(r)))\,; \\ D &= 1 \oplus D'_{p'}. \end{aligned} \tag{21}$$

Now formulas (15) and (19) contain parallel feature. The reduced form can also be obtained by these algorithms. Now we will prove that

$$B(p) = F_{s,r} A(p) F_{s,r}^{-1}, \tag{22}$$

and for the case that s and r are relative prime,

$$B(p) = F'_{s,r} A(p) (F'_{s,r})^{-1}. \tag{23}$$

The equation (23) can be treated as a special case of the equation (22), where $T = I_{p'}$. We only make the proof on (22).

Proof The item

$$\begin{aligned} &1 \oplus ((F(s) \otimes I_r) T (I_s \otimes F(r)))^{-1} = \\ &\quad (1 \oplus (I_s \otimes F(r))^{-1})(1 \oplus T^{-1})(1 \oplus (F(s) \otimes I_r)^{-1}). \end{aligned} \tag{24}$$

We will make the proof step by step. First,

$$\begin{aligned} A_0 &= A(p)(1 \oplus (I_s \otimes F(r))^{-1}) \\ &= \begin{pmatrix} 1 & 1_s^t \otimes \alpha_r^t \\ -1_{p'} & (I_s \otimes F(r))^{-1} \end{pmatrix}. \end{aligned} \tag{25}$$

Because T consists of s diagonal blocks of size r and the first element of each block is 1, we have

$$(1_s^t \otimes \alpha_r^t) T^{-1} = (1_s^t \otimes \alpha_r^t). \tag{26}$$

Then

$$A_1 = A_0(1 \oplus T^{-1}) = \begin{pmatrix} 1 & 1_s^t \otimes \alpha_r^t \\ -1_{p'} & (I_s \otimes F(r))^{-1} T^{-1} \end{pmatrix}. \tag{27}$$

Because

$$(1_s^t \otimes \alpha_r^t)(F(s) \otimes I_r)^{-1} = \alpha_s^t \otimes \alpha_r^t = \alpha_{p'}, \tag{28}$$

then,

$$\begin{aligned} A_2 &= A_1(1 \oplus (F(s) \otimes I_r))^{-1}) \\ &= \begin{pmatrix} 1 & \alpha_{p'}^t \\ -1_{p'} & (I_r \otimes F(r))^{-1} T^{-1} (F(s) \otimes I_r)^{-1} \end{pmatrix}. \end{aligned} \tag{29}$$

The following steps now become easier,

$$B_2 = (1 \oplus (I_s \otimes F(r)))A_2 = \begin{pmatrix} 1 & \alpha_{p'}^t \\ -s \cdot 1_s \otimes \alpha_s & T^{-1}(F(s) \otimes I_r)^{-1} \end{pmatrix}, \tag{30}$$

and

$$B_1 = (1 \oplus T)B_2 = \begin{pmatrix} 1 & \alpha_{p'}^t \\ -s \cdot 1_s \otimes \alpha_s & (F(s) \otimes I_r)^{-1} \end{pmatrix}. \tag{31}$$

Finally,

$$B_0 = (1 \oplus (F(s) \otimes I_r))B_1 = \begin{pmatrix} 1 & \alpha_{p'}^t \\ -p' \cdot \alpha_{p'} & I_{p'} \end{pmatrix} = B(p). \tag{32}$$

We complete the proof.

The formulas (15) and (19) now become

$$F(p) = P_\pi^{-1} F_{s,r} DB(p) F_{s,r} P_\pi, \tag{33}$$

and

$$F(p) = P_0 F'_{s,r} DB(p) F'_{s,r} P_0^{-1}, \tag{34}$$

respectively.

The permutation P_π seems to be more simple than the permutation P_0 in (20) because P_0 is a product of three permutations. In fact, P_0 still can be implemented in simple and parallel way. We take the input permutation P_0^{-1} as an example.

$$P_0^{-1} = 1 \oplus T_s P(p', s) P_\pi. \tag{35}$$

The matrix of input indices after $P(p', s)P_\pi$ would be

$$X = \begin{pmatrix} 1 & z^s & \cdots & z^{(r-1)s} \\ z & z^{s+1} & \cdots & z^{(r-1)s+1} \\ \vdots & & \ddots & \vdots \\ z^{s-1} & z^{2s-1} & \cdots & z^{sr-1} \end{pmatrix}. \tag{36}$$

We see that the element in row i can be obtained from the corresponding element in row i by

$$X[i, j] = X[i-1, j] \times z. \tag{37}$$

Because

$$T_s = I_r \oplus S^e \oplus \cdots \oplus S^{(s-1)e}$$

is a block diagonal matrix with shift blocks each of which corresponds to a row of X, it shifts the i-th row of X by $i \cdot e$ positions to the right. It is not difficult to find that these shift operations are circulant. Denote by X' the index matrix after the shift operation T_s, then

$$X'[i,j] = X[i,j] \times z^{-es}. \tag{38}$$

Combining (37) and (38) together, we have that

$$X'[i,j] = X[i-1,j] \times z^{1-es}, \tag{39}$$

which means that if the elements in any row are pre-defined the elements in the rest rows can be obtained in order (take modulo s) by (39). Therefore, The permutation P_0 and P_0^{-1} can be implemented in parallel, which is identical to the computation structure in (19).

9.3 Computing $C(p)$ By Real-Type Algorithms

The size property of $C(p)$ reminds us that the multiplicative algorithm can be used to solve $C(p)$. In traditional way, the 'best' one would be the relative prime case algorithm. For $p' = 2^k \cdot r$, if $r > 1$ there must exist a pair that $p' = s_0 r_0$ and $(s_0, r_0) = 1$, then $C(p)$ can be computed either by the Agarwal-Cooley algorithm or by the Block-shift theorem. Unfortunately, both of them can not use the conjugate property because of the permutation. For example, take $p = 7$, then $p' = 6 = 2 \times 3$. Then, the vector $\underline{a}^t$ formed by the first row of the matrix after the permutation on $C(p)$ would be, by the Agarwal-Cooley algorithm,

$$\underline{a}^t = (a_0, a_4, a_2; a_3, a_1, a_5),$$

and by the Block-shift theorem,

$$\underline{a}^t = (a_0, a_2, a_4; a_3, a_5, a_1).$$

Even if there is a pair satisfying the conjugate property, but for the whole matrix the conjugate property no longer exists. The computation has to be taken on complex field, the real-type algorithms have no more advantages. Now we will discuss how the new algorithms can work perfectly.

First, we will prove that the conjugate property holds for the block format, too, if the block size is also even. Break $C(p)$ into $s \times s$ blocks, where s is an even number, and denote the blocks by h_i. As we know that each h_i is a half-circulant matrix, denoting its entries by $c_{i,j}$ and $c'_{i,j}$ as we used to do, then we have

$$\begin{aligned} c_{i,j} &= a_{ir+j}; \qquad & c'_{i,j} &= a_{(i+1)r+j}; \\ 0 \le i < s, & \qquad & 0 \le j < s-1. \end{aligned} \tag{40}$$

Then for the elements in the block $h_{i+\frac{s}{2}}$, we have

$$\begin{aligned} c_{i+\frac{s}{2},j} &= a_{ir+j+\frac{p'}{2}} = a^*_{ir+j} = c^*_{i,j}; \\ c'_{i+\frac{s}{2},j} &= a_{(i+1)r+j+\frac{p'}{2}} = a^*_{(i+1)+j} = (c'_{i,j})^*. \end{aligned} \tag{41}$$

So we have that

$$h_{i+\frac{s}{2}} = h_i^*; \qquad 0 \le i < s. \tag{42}$$

By the new $2^k \cdot r$-algorithm including the special cases of $k = 1$ and $r = 1$, the first step is breaking $C(p)$ into 2×2 block format, then applying the 2-point algorithm,

$$C(p) = (C_2 \otimes I_{\frac{p'}{2}})(G_0 \oplus G_1)(M_2 \otimes I_{\frac{p'}{2}}), \tag{43}$$

where, by the conjugate property with block form,

$$\begin{aligned} G_0 &= \frac{1}{2}(h_0 + h_1) = \frac{1}{2}(h_0 + h_0^*) = \left[cos(z^{l+k}\theta)\right]; \\ G_1 &= \frac{1}{2}(h_0 - h_1) = \frac{1}{2}(h_0 - h_0^*) = i \cdot \left[sin(z^{l+k}\theta)\right]; \\ & 0 \le l, k < \frac{p'}{2}, \end{aligned} \tag{44}$$

where G_0 is a real skew-circulant matrix and G_1 is a Δ-half-circulant matrix with $\Delta = -1$, but has all pure imaginary entries. Which can be found by

$$\begin{aligned} cos(z^{j+\frac{p'}{2}}\theta) &= cos(-z^j\theta) = cos(z^j\theta); \\ sin(z^{j+\frac{p'}{2}}\theta) &= sin(-z^j\theta) = -sin(z^j\theta). \end{aligned} \tag{45}$$

G_0 and G_1 now can be computed by corresponding real-type algorithms. This result can be extended to any factorization of $p' = s.r_0$, where s

is an even number. Which means that if we block-diagonalize $C(p)$ by s-point (also $2^k \cdot r$ case) algorithm, the blocks in block-diagonal matrix should be either real or pure imaginary. Because after the first step, the computation will continue on either real or pure imaginary matrices.

If $C(p)$ can be factorized by one of the cases of the $2^k \cdot r$-algorithm as

$$C(p) = C_{p'} G_{p'} M_{p'}, \tag{46}$$

then

$$F(p) = P_\pi^{-1} C_f G_f M_f A(p) P_\pi, \tag{47}$$

where

$$\begin{aligned} C_f &= (1 \oplus C_{p'}); \\ G_f &= (1 \oplus G_{p'}); \\ M_f &= (1 \oplus M_{p'}). \end{aligned} \tag{48}$$

If the property (33) of chapter 8 holds for the pre-addition matrix $M_{p'}$, then the computation of $A(p)$ can be reduced in the same way as (34) by the real-type algorithm. Set

$$B'(p) = \begin{pmatrix} 1 & \alpha^t_{m_{p'}} \cdot S^{-i} \\ -p' \cdot S^i \cdot \alpha_{m_{p'}} & I_{m_{p'}} \end{pmatrix};$$

$$m_{p'} = M(p'), \tag{49}$$

where S is the $m_{p'} \times m_{p'}$ circulant shift matrix. If the position of the row having the only non-zero sum of its elements is i, i.e., the i-th row, then, by direct computation it easy to prove that

$$(1 \oplus M_{p'}) A(p) = B'(p)(1 \oplus M_{p'}). \tag{50}$$

Then, the similar variant form to (5) can be obtained for the real-type algorithms, too,

$$F(p) = P_\pi^{-1} C_f G_f B'(p) M_f P_\pi. \tag{51}$$

The arithmetic count, denoted by $M_f(p)$ and $A_f(p)$, now can be measured by

$$\begin{aligned} M_f(p) &= M(p') + 1; \\ A_f(p) &= A(p') + 2. \end{aligned} \tag{52}$$

The one more multiplication comes from the matrix $B'(p)$. In fact, this multiplication with an integer can be saved in the implementation. If the input is complex, the count should be doubled. But if the input is

real the number of additions can be further saved from (52). Because the last step in the post-addition matrix $C_{p'}$ is $C_2 \otimes I_{\frac{p'}{2}}$, which does the additions between the real numbers and pure imaginaries, therefore, no computation at all. The total number of additions now is

$$A_{f_r}(p) = A(p') + 2 - \frac{p'}{2}. \tag{53}$$

We now give a further discussion on each case.

(i). The Case of $p' = 2^k$

By the property (33) for 2^k-algorithm, the parameter i in (49) is 1. Then the matrix $B'(p)$ is the same as the matrix $B(p)$ in (4) except that the size is different. Let $p' = 2^k$. The arithmetic count for this case is

$$\begin{aligned} M_f(p) &= \frac{1}{2}(3^k + 3); \\ A_f(p) &= \frac{1}{2}(2^{k+1} + 3^{k+1} - 1). \end{aligned} \tag{54}$$

(ii). The Case of $p' = 2r$

By the $2r$-algorithm, we have that

$$M_{p'} = (M_r \oplus M_r T_r)(M_2 \otimes I_r). \tag{55}$$

Set

$$A(r) = \begin{pmatrix} 1 & 1_r^t \\ -2 \cdot 1_r & I_r \end{pmatrix}. \tag{56}$$

Then

$$\begin{aligned} &(1 \oplus M_{2r})A(p) \\ &= (1 \oplus M_r \oplus M_r T)(1 \oplus (M_2 \otimes I_r))A(p) \\ &= (1 \oplus M_r \oplus M_r T)(A(r) \oplus I_r)(1 \oplus (M_2 \otimes I_r)) \\ &= ((1 \oplus M_r)A(r) \oplus M_r T)(1 \oplus (M_2 \otimes I_r)). \end{aligned} \tag{57}$$

Now the problem is that we have to assume that the property (33) of chapter 8 holds for the r-point algorithm. If r is prime, by the Negative-based algorithm introduced in chapter 6, we know that there are $\frac{r-1}{2}$ rows which do not satisfy the property. It implies that the computation of $A(r)$ can be reduced to $\frac{r-1}{2}$ integer multiplications and $2r - 1$

additions. But we found that by the Winograd algorithm the property holds for all the sample points so far we have known, but the row having the non-zero sum may be not in the first position. For convenience, we assume that the property is true for all the prime case.

If r is not prime, say, $r = r_0 r_1$, we have proved that the property holds for r-point algorithm if, a). r_0 and r_1 are relative prime; b). the property holds for both r_0 and r_1 point algorithms; and c). the r-point algorithm is constructed by the relative prime case algorithm. By the inductive procedure, the property will hold for the general r, where r is an odd number.

Based on the assumption and the above discussion, we have that

$$(1 \oplus M_r)A(r) = A'(r)(1 \oplus M_r), \tag{58}$$

where

$$A'(r) = \begin{pmatrix} 1 & \alpha_r^t \\ -2r \cdot \alpha_r & I_r \end{pmatrix}. \tag{59}$$

Now the formula (57) becomes

$$(1 \oplus M_{p'}) = (A'(r) \oplus I_r)(1 \oplus M_r \oplus M_r T)(M_2 \otimes I_r). \tag{60}$$

The arithmetic count for this case is

$$\begin{aligned} M_f(p) &= 2M(r) + 1; \\ A_f(p) &= 2A(r) + 4r + 2. \end{aligned} \tag{61}$$

(iii). The Case of $p' = 2^k \cdot r$

Based on the discussion for $2r$ case, we can prove the $2^k r$ case by using the inductive method. Set $s = 2^{k-1} r$ and

$$A(s) = \begin{pmatrix} 1 & 1_s^t \\ -2 \cdot 1_s & I_s \end{pmatrix}. \tag{62}$$

Suppose that the property holds for s-point algorithm. Which implies that

$$(1 \oplus M_s)A(s) = B(s)(1 \oplus M_s), \tag{63}$$

where

$$B(s) = \begin{pmatrix} 1 & \alpha_s^t \\ -2s \cdot \alpha_s & I_s \end{pmatrix}. \tag{64}$$

Thus,

$$\begin{aligned}
&(1 \oplus M_{p'})A(p) \\
&= (1 \oplus M_s \oplus M_s^{\Delta})(1 \oplus (M_2 \otimes I_s))A(p) \\
&= (1 \oplus M_s \oplus M_s^{\Delta})(1 \oplus A(s) \oplus I_s)(1 \oplus (M_2 \otimes I_s)) \\
&= ((1 \oplus M_s)A(s) \oplus M_s^{\Delta})(1 \oplus (M_2 \otimes I_s)) \\
&= (B(s) \oplus I_s)(1 \oplus M_{p'}) \\
&= B'(p)(1 \oplus M_{p'}).
\end{aligned} \tag{65}$$

The arithmetic count for this general case is

$$M_f(p) = \frac{1}{2}(3^k + 1)M(r) + 1; \tag{66}$$

$$A_f(p) = \frac{3}{2}(3^k - 2^{k+1} + 1)M(r) + 2^k A(r) + 4r(2^k - 1) + 2. \tag{67}$$

9.4 Summary

The above two approaches give us more choices than the traditional way for the computation of $F(p)$. For the large size, the complex-type algorithm can be used for efficient and parallel processing, while the size is not too large the real-type algorithm with different cases can be applied. As we analyzed in the preceding chapter, the performance can be improved even more than 20% percent. The gains come from two parts, one is that the new algorithms are better than all the existing algorithms at least for some cases, another is the perfect use of the conjugate property of $C(p)$. Because the performance of the prime case FFT algorithms depends on the p'-point cyclic convolution algorithm, the tables 8-3 and 8-4 can also be used as reference for prime case FFT algorithms.

By the multiplicative algorithms for FFTs, the non-prime case algorithms are built up by the prime case algorithms, the gains then can be transferred to all of these non-prime case algorithms. In addition, for multi-dimensional FFTs and symmetrized FFTs with the prime size, it will be found that the computational matrix likes the Winograd core $C(p)$, and at least keeps all the properties of $C(p)$. The computation procedures and the algorithms are much alike.

CHAPTER 10

NEW ALGORITHMS FOR TWO DIMENSIONAL PRIME CASE FFT

10.1 Introduction

The 2-dimensional $n \times n$ finite Fourier Transform [28] is defined by

$$B_{k,l} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} v^{ik+jl} A_{i,j};$$

$$v = exp(2\pi i/n), \qquad 0 \leq k, l < n, \tag{1}$$

where A is input data and B is output data.

In general, to compute the 2-dimensional Fourier Transform $F(n \times n)$ we would first change the 2-dimensional input and output data into one dimension and then use 1-dimensional FFT algorithms. The basic algorithm is the Nested transform algorithm or called Row-column algorithm. By which one can compute the 2-dimensional Fourier transform by computing a one-dimensional Fourier transform along every column, then along every row, or the other way around. Using the tensor product formulation,

$$F(n \times n) = F(n) \otimes F(n). \tag{2}$$

Obviously, the arithmetic count for computing $F(n \times n)$ is

$$\begin{aligned} M(n \times n) &= 2nM_f(n); \\ A(n \times n) &= 2nA_f(n). \end{aligned} \tag{3}$$

This algorithm has good parallel feature, but no flexibility. In this chapter, we will describe some new algorithms which do not depend on the one-dimensional FFT but cyclic convolution algorithm. We will focus on the prime case $F(p \times p)$. By using algebraic transformation on the indexing set, for some prime numbers the resulting computation structure is completely the same as 1-dimensional prime case. First, we will introduce some algebraic structures of the indexing set for 2-dimensional array, then the algorithms for the cases of $p \equiv 3 \bmod 4$ and $p \equiv 2 \bmod 3$ will be discussed. The new algorithms are more efficient in some cases.

10.2 The Algebraic Structure Of The Indexing Set

Professor Tolimieri has represented the algebraic structures of the indexing set for 2-dimensional array in several paper [31]. Based on these algebraic structures, the computation structure of $F(n \times n)$ can be easily changed for different purpose. We will give a brief description for some of them.

The data indexing set $Z/n \times Z/n$ has a natural ring-structure as the direct product of the ring Z/n with itself. Denoting a typical point by $a = (a_0, a_1)$ we have

$$a + b = (a_0 + b_0, a_1 + b_1); \tag{4}$$

$$ab = (a_0 b_0, a_1 b_1). \tag{5}$$

A second ring-structure can be defined by first forming the quotient polynomial ring

$$Z/n\,[x]\,/(x^2 + 1) \tag{6}$$

consisting of all polynomials $a(x) = a_0 + a_1 x$ where addition and multiplication are taken mod $(x^2 + 1)$. Identifying the point $a = (a_0, a_1)$ with the polynomial $a(x) = a_0 + a_1 x$ we have a second ring-structure in the indexing set

$$a * b = (a_0 b_0 - a_1 b_1, a_0 b_1 + a_1 b_0). \tag{7}$$

Another ring-structure can be also defined by a similar way as the second,

$$Z/n\,[x]\,/(x^2 + x + 1), \tag{8}$$

here the operations are taken mod $(x^2 + x + 1)$ and the multiplication now is

$$a \circ b = (a_0 b_0 - a_1 b_1, a_0 b_1 + a_1 b_0 - a_1 b_1). \tag{9}$$

Denote by $L(Z/n \times Z/n)$ the set of all complex-valued functions F on $Z/n \times Z/n$. We can view F as a 2-dimensional array of data

$$F = [f(j,k)]\,, \qquad 0 \le j,k < n. \tag{10}$$

The 2-dimensional $n \times n$ finite Fourier Transform can be defined by

$$F(f)(b) = \sum_{a \in Z/n \times Z/n} f(a) v^{<a,b>}; \quad v = exp(2\pi i/n),$$

$$b \in Z/n \times Z/n, \quad < a, b >= a_0 b_0 + a_1 b_1. \tag{11}$$

If we set

$$\phi : Z/n \times Z/n \longrightarrow Z/n$$

and define

$$G(f)(b) = \sum_{a \in Z/n \times Z/n} f(a) v^{\phi(a \cdot b)}; \qquad b \in Z/n \times Z/n. \tag{12}$$

Then the computation of (11) can be replaced by (12) which is related to a certain ring-structure with the multiplication $\cdot$. For example, set

$$\phi(a) = a_0; \qquad a = (a_0, a_1), \tag{13}$$

and take the second or the third ring-structure. Then

$$\phi(a * b) = \phi(a \circ b) = a_0 b_0 - a_1 b_1. \tag{14}$$

A direct computation shows that

$$G(f)(b) = F(f)(b_0, -b_1); \qquad b \in Z/n \times Z/n, \tag{15}$$

which implies that $G(f)$ and $F(f)$ are related by the output permutation

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{16}$$

We may use these different ring-structures and define different mapping ϕ, then the computation of $G(f)$ would be different, so does the output permutation. Many different algorithms can be obtained by using algebraic transformation.

10.3 The Prime Size $p \equiv 3 \bmod 4$

Denote by $R(p)$ the indexing set $Z/p \times Z/p$ and by $U_2(p)$ the non-zero elements in $R(p)$. In this case, the order of the set $U(p)$ of non-zero elements in Z/p

$$p' = p - 1 \equiv 2 \bmod 4,$$

there is no $u \in U(p)$ such that $u^2 = -1$. Therefore, $x^2 + 1$ is irreducible over Z/p and $Z/p[x]/(x^2 + 1)$ is the field $GF(p^2)$, and the set $U_2(p)$ is a cyclic group under multiplication. Set

$$a^{m+1} = a * a^m; \qquad a \in R(p), \quad m \geq 1. \tag{17}$$

Then, a generator w of $U_2(p)$ can be found and input and output can be ordered by

$$0; 1, w, \cdots; w^{p^2-2}. \tag{18}$$

Denote the matrix of the mapping G introduced in (15) relative to (18) by G, again. Then

$$G = \begin{pmatrix} 1 & 1_{p''}^t \\ 1_{p''} & C_2(p) \end{pmatrix}; \qquad p'' = p^2 - 1, \tag{19}$$

where

$$C_2(p) = \begin{bmatrix} & v^{\phi(w^{j+k})} \end{bmatrix}, \qquad 0 \le j, k < p^2 - 1, \tag{20}$$

is a skew-circulant matrix.

We see now that G has the same structure as one dimensional prime case and $C_2(p)$ keeps the same properties as $C(p)$. First,

$$\sum_{i=0}^{p-2} v^{\phi(w^i)} = -1, \tag{21}$$

then G can be rewritten as

$$G = (1 \oplus C_2(p))A_2(p), \tag{22}$$

where

$$A_2(p) = \begin{pmatrix} 1 & 1_{p''}^t \\ -1_{p''} & I_{p''} \end{pmatrix}. \tag{23}$$

Let $p = 4n + 3$, then $p'' = 8(2n^2 + 3n + 1)$, the size of $C_2(p)$ can also be represented as

$$p'' = 2^k \cdot r; \qquad k \ge 3.$$

Finally, because $\phi(-a) = -\phi(a)$ the conjugate property holds for $C_2(p)$ too. Denote by P_w the permutation matrix generating the ordering sequence in (18), then

$$F(p \times p) = P_b P_w (1 \oplus C_2(p)) A_2(p) P_w^{-1}, \tag{24}$$

where P_b is the output permutation described in (16). Here we treat input and output data as 1-dimensional. $F(p \times p)$ is completely computed by 1-dimensional prime case FFT algorithms. The computation

of $A_2(p)$ can also be replaced by $B_2(p)$ which is similar to $B(p)$ in (4) of chapter 9,

$$B_2(p) = \begin{pmatrix} 1 & \alpha_{p''}^t \\ -p'' \cdot \alpha_{p''} & I_{p''} \end{pmatrix}. \tag{25}$$

One way to find the generator w is described as follows. For a generator z of Z/p, a generator w of $R(p)$ can be found such that

$$w^k = z; \qquad k = p''/p' = (p+1). \tag{26}$$

Set

$$i = (0,1), \tag{27}$$

then

$$i^4 = 1. \tag{28}$$

Let

$$w = (a,b) = a + ib; \qquad a, b \in U(p). \tag{29}$$

By the Theorem 2.1 of chapter 2, we have that

$$\begin{aligned} w^k &= (a+ib)^p(a+ib) \\ &= (a^p + (ib)^p)(a+ib) \\ &= (a-ib)(a+ib) \\ &= a^2 + b^2 \end{aligned} \tag{30}$$

Set

$$a^2 + b^2 = z. \tag{31}$$

Under the addition in Z/p except 0 and z, there must have $p'-1$ different pairs (α, β) such that $\alpha + \beta = z$. Because the number of elements with even power of z is one more than the number of elements with odd power of z, there must exist a pair such that

$$\alpha = z^{2k_1}; \qquad \beta = z^{2k_2}. \tag{32}$$

Set

$$a = \pm z^{k_1}; \qquad b = \pm z^{k_2}, \tag{33}$$

or

$$a = \pm z^{k_2}; \qquad b = \pm z^{k_1}.$$

Then, one value of w can be found from these solutions by satisfying

$$(a+ib)^i \neq z^j; \qquad 1 \leq i \leq p. \tag{34}$$

Example 1. Take $p = 3$ and $z = 2$. The pair $(1,1)$ satisfies $1+1=2$ and $1=1^2$. Set $a=b=1$, which also satisfies the condition (34), then $w=(1,1)$ is a generator of $U_2(3)$. The table 10-1 listed one of w's for each p from 3 to 59.

Table 10 - 1

p:	3	7	11	19	23	31	43	47	59
z:	2	3	2	2	5	3	3	5	2
w:	(1,1)	(1,3)	(2,3)	(2,6)	(1,2)	(1,8)	(5,8)	(1,2)	(2,23)

By the way we find a generator, the input and output permutations can be implemented very efficiently. Let the ordered set

$$A_0 = 1, w, \cdots, w^{k-1}. \tag{35}$$

Then the ordering (18) can be rewritten as

$$A_0, zA_0, \cdots z^{p'-1}A_0. \tag{36}$$

Suppose that the 2-dimensional array $D(p',k)$ holds input data, and for $w^j = (z^{j_1}, z^{j_2})$, define two 1-dimensional arrays $W_1(k)$ and $W_2(k)$ by

$$W_1(j) = j_1; \qquad W_2(j) = j_2, \tag{37}$$

except that $w^{k/2} = (0, -z^{1/2})$ and $w^k = (z, 0)$. Let another array Z holds the powers of z by $Z(i) = z^i$. To avoid the modulo operation on p', double length can be defined for Z. the following program implements the input permutation.

```
      INTEGER   W1(0:K-1),W2(0:K-1),Z(0:2p'-1),i,j,k0
      ...
      k0 = k/2
      DO 10 i = 0,p'-1
        D(i,k0) = f(0,Z(i+W2(k0))
        D(i,k) = f(Z(i+W1(k)),0)
        DO 20 j = 1,k0-1
          D(i,j) = f(Z(i+W1(j),Z(i+W2(j))
20      CONTINUE
        DO 10 j = k0+1,k-1
          D(i,j) = f(Z(i+W1(j),Z(i+W2(j))
10    CONTINUE
```

We see that only additions for index addressing are needed. Also, set

$$c_0 = [v^{\phi(w^{i+j})}]_{0 \le i,j < k}, \tag{38}$$

the computation matrix $C_2(p)$ in (20) now can be rewritten as

$$C_2(p) = C(p) \otimes c_0, \tag{39}$$

where $C(p)$ is the Winograd core.

Example 2. Now we continue the example of $p = 3$ to illustrate the whole algorithm. Take $z = 2$ and $w = (1, 1)$, the corresponding ordering and mapping for A_0 are listed below,

input	$(1,0)$	$(1,1)$	$(0,2)$	$(1,2)$
A_0	w^0	w^1	w^2	w^3
$\phi(w^i)$	1	1	0	1

We have that

$$C_2(3) = \begin{pmatrix} v & v & 1 & v & v^2 & v^2 & 1 & v^2 \\ v & 1 & v & v^2 & v^2 & 1 & v^2 & v \\ 1 & v & v^2 & v^2 & 1 & v^2 & v & v \\ v & v^2 & v^2 & 1 & v^2 & v & v & 1 \\ v^2 & v^2 & 1 & v^2 & v & v & 1 & v \\ v^2 & 1 & v^2 & v & v & 1 & v & v^2 \\ 1 & v^2 & v & v & 1 & v & v^2 & v^2 \\ v^2 & v & v & 1 & v & v^2 & v^2 & 1 \end{pmatrix}.$$

If we directly use 8-point algorithm to compute $C_2(3)$, it would need 14 multiplications and 46 additions (in fact, some of them may be trivial operations). Instead, we use our 2^k algorithm for $C_2(3)$. First, partially diagonalizing $C_2(3)$ by 2-point algorithm,

$$C_2(3) = (C_2 \otimes I_4) D (M_2 \otimes I_4),$$

where $D = D_0 \oplus D_1$, and

$$D_0 = cos(\theta) \begin{pmatrix} 1 & 1 & -2 & 1 \\ 1 & -2 & 1 & 1 \\ -2 & 1 & 1 & 1 \\ 1 & 1 & 1 & -2 \end{pmatrix};$$

$$D_1 = i \cdot sin(\theta) \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 \\ 0 & 1 & -1 & -1 \\ 1 & -1 & -1 & 0 \end{pmatrix},$$

where

$$\theta = 2\pi/3; \qquad cos(\theta) = -\frac{1}{2}.$$

We see that D_0 and D_1 can be directly computed without any algorithm,

$$(1 \oplus D_0 \oplus D_1)(M_2 \otimes I_4)A_2(3) = (D_0' \oplus D_1)(M_2 \otimes I_4),$$

where

$$D_0' = \begin{pmatrix} 1 & 1_4^t \\ 1_4 & D_0 \end{pmatrix}.$$

Now the computation of D_0' needs 14 additions and 4 rational multiplications and the computation of D_1 needs 8 additions and 4 multiplications. The total count for computing $F(3\times 3)$ would be 8 multiplications and 38 additions. It is better than the Nested algorithm which uses 12 multiplications and 36 additions.

10.4 The Prime Size $p \equiv 2 \bmod 3$

In the case of $p \equiv 2 \bmod 3$, the order of $U(p)$ is

$$p' = p - 1 \equiv 1 \bmod 3,$$

there is no $u \in U_(p)$, where $u \neq 1$, such that $u^3 = 1$. Then, $u^3 - 1 \neq 0$ implies that $u^2 + u + 1 \neq 0$. Therefore, $x^2 + x + 1$ is irreducible over Z/p and $Z/p[x]/(x^2 + x + 1)$ is the field $GF(p^2)$. Set

$$a \circ a = a^2; \qquad a^{m+1} = a \circ a^m; \qquad a \in R(p). \tag{40}$$

The computation structure for this case is the same as the case of $p \equiv 3 \bmod 4$.

The generator γ can be found by a similar way to finding w in the preceding section. Set

$$\rho = (0, 1). \tag{41}$$

We see that

$$\rho^3 = 1. \tag{42}$$

Let

$$\gamma = (a, b) = a + b\rho; \qquad a, b \in U(p). \tag{43}$$

Again by the Theorem 2.1 of chapter 2, we have that

$$\begin{aligned} \gamma^k &= (a + b\rho)^p(a + b\rho) \\ &= (a^p + b^p\rho^p)(a + b\rho) \\ &= (a + b\rho^2)(a + b\rho) \\ &= a^2 + b^2 - ab \end{aligned} \tag{44}$$

Set

$$\alpha = a^2 + b^2; \qquad \beta = -ab, \tag{45}$$

and take

$$\alpha + \beta = z; \qquad \alpha, \beta \in U(p). \tag{46}$$

Then

$$(a+b)^2 = \alpha - 2\beta. \tag{47}$$

If

$$\alpha - 2\beta = a_0^2; \qquad a_0 \in Z/p, \tag{48}$$

the condition that the set of equations

$$\begin{cases} a + b = \pm z^{k_1} \\ ab = -\beta \end{cases} \tag{49}$$

have solution is

$$\alpha + 2\beta = b_0^2; \qquad b_0 \in Z/p. \tag{50}$$

Then, the solutions would be

$$a = \pm\frac{1}{2}(a_0 \pm b_0); \qquad b = \pm\frac{1}{2}(a_0 \mp b_0). \tag{51}$$

To be a generator, the condition

$$(a + b\rho)^i \neq z^j; \qquad 1 \leq i \leq p, \tag{52}$$

should also be satisfied.

Example 3. Take $p = 5$ and $z = 3$. The pair $(2, 1)$ satisfies

$$\alpha + \beta = 2 + 1 = 3;$$
$$\alpha - 2\beta = 2 - 2 = 0;$$
$$\alpha + 2\beta = 2 + 2 = 2^2.$$

By checking the condition (52), one of w is $(1, -1)$. The table 10-2 listed one of the values of γ for each such p from 5 to 59.

Table 10 - 2

p:	5	11	17	23	29	41	47	53	59
z:	3	2	3	5	2	7	5	2	32
γ:	$(1,4)$	$(3,-1)$	$(3,7)$	$(4,6)$	$(5,6)$	$(3,1)$	$(2,8)$	$(6,26)$	$(2,19)$

The procedure of implementing the input and output permutation is all the same as the case of $p \equiv 3 \bmod 4$, except that k_0 is either $\frac{k}{3}$ or $\frac{2k}{3}$. The computation matrix $C_2(p)$ can also be written as the same form as (39).

Example 4. Now we continue the example of $p = 5$. Take $z = 3$ and $\gamma = (1, -1)$, the ordering and mapping for A_0 are listed below,

input	$(1,0)$	$(1,4)$	$(0,2)$	$(2,4)$	$(1,1)$	$(2,1)$
A_0	γ^0	γ^1	γ^2	γ^3	γ^4	γ^5
$\phi(\gamma^i)$	1	1	0	2	1	2

$C_2(5)$ is 24×24 skew-circulant matrix. The $2^k \cdot r$-algorithm can be used to compute $C_2(5)$. In fact, if we do the computation one step by one step, the computation can be further saved. For convenience, we write $C_2(5)$ in block format of 2×2,

$$C_2(5) = \begin{pmatrix} c_0 & c_1 \\ c_1 & c_0 \end{pmatrix},$$

where $c_1 = c_0^*$. Block-diagonalizing $C_2(5)$ by 2-point algorithm,

$$C_2(5) = (C_2 \otimes I_{12})(D_0 \oplus D_1)(M_2 \otimes I_{12}).$$

We will only discuss the computation of D_0 to see how the computation can be saved. Let $a = cos(\theta)$ and $b = cos(2\theta)$, then

$$D_0 = \begin{pmatrix}
a & a & 1 & b & a & b & b & b & 1 & a & b & a \\
a & 1 & b & a & b & b & b & 1 & a & b & a & a \\
1 & b & a & b & b & b & 1 & a & b & a & a & a \\
b & a & b & b & b & 1 & a & b & a & a & a & 1 \\
a & b & b & b & 1 & a & b & a & a & a & 1 & b \\
b & b & b & 1 & a & b & a & a & a & 1 & b & a \\
b & b & 1 & a & b & a & a & a & 1 & b & a & b \\
b & 1 & a & b & a & a & a & 1 & b & a & b & b \\
1 & a & b & a & a & a & 1 & b & a & b & b & b \\
a & b & a & a & a & 1 & b & a & b & b & b & 1 \\
b & a & a & a & 1 & b & a & b & b & b & 1 & a \\
a & a & a & 1 & b & a & b & b & b & 1 & a & b
\end{pmatrix}.$$

Continuing to block-diagonalize D_0 by 2-point algorithm, because $cos\theta + cos(2\theta) = -\frac{1}{2}$, the two blocks

$$D_0' = -\frac{1}{4}\begin{pmatrix}
1 & 1 & -4 & 1 & 1 & 1 \\
1 & -4 & 1 & 1 & 1 & 1 \\
-4 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & -4 \\
1 & 1 & 1 & 1 & -4 & 1 \\
1 & 1 & 1 & -4 & 1 & 1
\end{pmatrix};$$

$$D_0'' = \frac{1}{2}(cos\theta - cos(2\theta)) \begin{pmatrix} 1 & 1 & 0 & -1 & 1 & -1 \\ 1 & 0 & -1 & 1 & -1 & -1 \\ 0 & -1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 0 \\ 1 & -1 & -1 & -1 & 0 & 1 \\ -1 & -1 & -1 & 0 & 1 & -1 \end{pmatrix}.$$

D_0' can be rewritten as

$$D_0' = -\frac{1}{4} \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

which can be computed by only 16 additions and 6 multiplications by $-\frac{1}{4}$. D_0'' needs 6 multiplications and 18 additions. Plus the additions for 2-point algorithm, the total are 12 multiplications and 58 additions. Directly using 12-point $2^k \cdot r$-algorithm we need 20 multiplications and 92 additions. A lot of savings can be made by this way.

10.5 Summary

From the above discussion, we can see that the flexibility for parallel processing can be obtained by changing 2-dimensional FFT into 1-dimensional FFT. By the Nested transform algorithm the degree of parallelism is completely determined by size p, while the new algorithms are more flexible. Notice that for some p both algorithms can be used, only the case of $p \equiv 1 \bmod 3$ can not be covered.

By these algorithms, we see that the computational matrix $C_2(p)$ is the same as the Winograd core $C(p)$ except that the $C_2(p)$ has the large size $p^2 - 1$, and $C_2(p)$ keeps all the properties of $C(p)$. The new algorithms for computing $F(p)$ can be completely used here. In particular, from the examples 2 and 4, we can find that by the $2^k r$-algorithm in chapter 8 for cyclic convolution, the two blocks D_0 and D_1 have very special format, the computation can be further reduced. For some points, the new algorithms may be more efficient than the Nested transform algorithm. But, when the size is a little large the size of $C_2(p)$ increases very fast, the real-type cyclic convolution algorithm may be no long efficient. In general, the complex-type multiplicative algorithms can be used for parallel processing, which we already discussed in chapter 7.

Because $i^4 = 1$ and $\rho^3 = 1$ correspond to 90° and 120° symme-

tries respectively, the computation structures can be applied to the 2-dimensional 90° and 120° symmetrized FFTs.

CHAPTER 11

SYMMETRIZED FFT ALGORITHMS WITH 90° ROTATION

11.1 Introduction

In many application fields, the input and output data for the computation of DFTs admit some kind of symmetry [29,30]. Since the data is redundant, it seems reasonable that the 2-dimensional $n \times n$ Fourier transform of this data can make use of this redundancy to reduce the computation. However, by traditional way, the Nested transform or the Cooley-Tukey algorithms operate on the complete n^2 data points. Except in certain cases where data redundancy can be interpreted as redundancy along a coordinate axis these algorithms can not be directly applied. Several important works by Richard Tolimieri have made use of this type redundancy.

In this chapter we will only discuss the prime case which admits 90° rotational symmetry based on Tolimieri's work [31]. Some development has been made in the discussion. The cyclic convolution also plays an important role in these computations.

Suppose that F denotes input data and is invariant under 90°-rotation. Then

$$f(a_0, a_1) = f(-a_1, a_0); \qquad a_0, a_1 \in Z/n. \tag{1}$$

Setting $i = (0, 1)$ and corresponding to the ring-structure

$$Z/n[x]/(x^2 + 1),$$

we have

$$i * a = (-a_1, a_0); \qquad i^4 = 1,$$
$$a = (a_0, a_1) \in Z/n \times Z/n, \tag{2}$$

which corresponds to 90°-rotation. Then

$$f(i * a) = f(a), \tag{3}$$

and also

$$f(-a) = f((-i) * a) = f(a). \tag{4}$$

From (12) of chapter 10, direct computation shows

$$G(f)(b) = G(f)(i * b) = G(f)(-b) = G(f)((-i) * b);$$

$$b \in Z/n \times Z/n, \tag{5}$$

which implies the output data keep the same symmetry.

Example 1. Take $p = 2$. Because $i*(1,1) = (1,1)$ and $i*(1,0) = (0,1)$, we order the input by $(0,0),(1,1)$ and $(1,0)$. The computation matrix now is

$$\begin{pmatrix} 1 & 1 & 2 \\ 1 & 1 & -2 \\ 1 & -1 & 0 \end{pmatrix},$$

which can be computed by 5 additions.

We will discuss two basic cases of $p \equiv 3 \bmod 4$ and $p \equiv 1 \bmod 4$ in the following sections.

11.2 The Case of $p \equiv 3 \bmod 4$

In this case we have proved that $Z/p[x]/(x^2+1)$ is the field $GF(p^2)$. A generator of the set $U_2(p)$ of non-zero elements in $R(p)$ can be obtained by

$$w^r = i; \qquad r = (p^2 - 1)/4. \tag{6}$$

By the 90° symmetry in (3), f is completely determined by its values in the set

$$0; 1, w, \cdots, w^{r-1}, \tag{7}$$

and $C_2(p)$ in the computational matrix G can be rewritten as block format

$$C_2(p) = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 & c_0 \\ c_2 & c_3 & c_0 & c_1 \\ c_3 & c_0 & c_1 & c_2 \end{pmatrix}, \tag{8}$$

where $c_2 = c_0^*$ and $c_3 = c_1^*$ by the conjugate property. Now setting

$$X = \begin{pmatrix} 1 & & & \\ & I_r & & \\ & & I_r & \\ & & & I_r \\ & & & I_r \end{pmatrix} = 1 \oplus (1_4 \otimes I_r), \tag{9}$$

we have

$$\underline{f} = X \begin{pmatrix} f(0) \\ f(1) \\ f(w) \\ \vdots \\ f(w^{r-1}) \end{pmatrix}, \tag{10}$$

where $\underline{f}$ denotes input data linearly ordered by (7).

Output data $g = G(f)$ is also invariant under 90°-rotation, which reduces the computation to

$$\begin{pmatrix} g(0) \\ 4g(1) \\ 4g(w) \\ \vdots \\ 4g(w^{r-1}) \end{pmatrix} = X^t G X \begin{pmatrix} f(0) \\ f(1) \\ f(w) \\ \vdots \\ f(w^{r-1}) \end{pmatrix}. \tag{11}$$

Direct computation shows that

$$X^t G X = \begin{pmatrix} 1 & 4 \cdot 1_r^t \\ 4 \cdot 1_r & 4 \cdot H_0 \end{pmatrix}, \tag{12}$$

where

$$H_0 = c_0 + c_1 + c_2 + c_3 \tag{13}$$

is a real skew-circulant matrix of order r. Set

$$T = 1 \oplus \frac{1}{4} \cdot I_r \tag{14}$$

and

$$\underline{g} = Tg = G_4(p)\underline{f}. \tag{15}$$

Then

$$G_4(p) = TX^t G X = \begin{pmatrix} 1 & 4 \cdot 1_r^t \\ 1_r & H_0 \end{pmatrix}. \tag{16}$$

Now the computation matrix $G_4(p)$ is similar to the 1-dimensional prime FFT. The H_0 is skew-circulant and the size has the format $2^k \cdot r$ too. Because H_0 is already real, any real-type algorithm can be applied directly. A variant form of $G_4(p)$ can also be obtained by the similar way we used before. Let

$$A' = \begin{pmatrix} 1 & 4 \cdot 1_r^t \\ -1_r & I_r \end{pmatrix}. \tag{17}$$

Because the property (21) of chapter 6 holds for H_0 too, then

$$G_4(p) = (1 \oplus H_0)A'. \tag{18}$$

If we compute H_0 by corresponding cyclic convolution algorithm, as we already proved,

$$G_4(p) = (1 \oplus C_r)DB'(1 \oplus M_r) \tag{19}$$

where

$$B' = \begin{pmatrix} 1 & 4 \cdot \alpha_r^t \\ -r \cdot 1_r & I_r \end{pmatrix}, \tag{20}$$

which needs one more rational multiplication by 4 than the matrix $B(p)$ in 1-dimensional case.

By the way we find a generator w of $U_2(p)$ in the preceding chapter, we have that

$$w^{k_0} = w^{\frac{p+1}{2}} = -z^{\frac{1}{2}}i,$$

then by (6),

$$w^r = w^{\frac{p^2-1}{4}} = (w^{k_0})^{\frac{p'}{2}} = (-z^{\frac{1}{2}}i)^{\frac{p'}{2}} = \pm i, \tag{21}$$

which does not change the symmetry. Therefore, the way can also be applied here. The procedures of implementing the input and output permutations have no big difference.

When the size p is small, by the approach we used for $C_2(p)$ in the preceding chapter, the computation for H_0 can be further saved.

Example 2. Take $p = 3$. Recall this example we used in the last chapter for 2-dimensional FFT without symmetry. We have that

$$G_4(3) = \begin{pmatrix} 1 & 4 & 4 \\ 1 & 1 & -2 \\ 1 & -2 & 1 \end{pmatrix},$$

which can be directly computed by using total of 9 additions.

11.3 The Case of $p \equiv 1 \bmod 4$

In this case, there exists $u \in U(p)$ such that

$$u^2 \equiv -1 \bmod p, \tag{22}$$

and the polynomial x^2+1 is not irreducible over Z/p. In fact,

$$x^2+1=(x-u)(x+u), \tag{23}$$

and $Z/p[x]/(x^2+1)$ is not a field. Denote by $R(p)$ the indexing set $Z/p\times Z/p$ with this ring-structure. By the Chinese Remainder theorem, we have a ring-isomorphism ψ

$$R(p)\cong Z/p\times Z/p \qquad (direct \quad product), \tag{24}$$

given by

$$\psi(a)=(a_0+a_1u,a_0-a_1u), \qquad a=(a_0,a_1)\in R(p). \tag{25}$$

The operation

$$\begin{aligned}&\psi(a*b)=\psi(a)\psi(b)=\\&((a_0b_0-a_1b_1)+(a_0b_1+a_1b_0)u,(a_0b_0-a_1b_1)-(a_0b_1+a_1b_0)u).\end{aligned} \tag{26}$$

By the definition for 2-dimensional DFT,

$$<\psi(a),\psi(b)>=2(a_0b_0-a_1b_1), \tag{27}$$

set

$$\phi(a)=2a_0; \qquad a=(a_0,a_1)\in R(p). \tag{28}$$

Then

$$<\psi(a),\psi(b)>=\phi(a*b), \tag{29}$$

and the output

$$G(f)(b)=F(f)(2b_0,-2b_1), \tag{30}$$

the output indexing permutation

$$B=\begin{pmatrix}2&0\\0&-2\end{pmatrix}. \tag{31}$$

Denote by D_1 and D_2 the subsets of $R(p)$ which under ψ correspond to $(0)\times U(p)$ and $U(p)\times(0)$, respectively. And denote the unit group of $R(p)$ by $U_2(p)$. Under ψ, we have the group-isomorphism

$$U_2(p)\cong U(p)\times U(p). \tag{32}$$

Then, a partition of $R(p)$ is give by the subsets

$$0; D_2, D_1; U_2(p). \tag{33}$$

The group $U(p)$ is cyclic, it can be ordered by the powers of a generator z of $U(p)$,

$$1, z, \cdots, z^{p-2}; \tag{34}$$

then D_1 and D_2 can be ordered related to this ordering for $(0) \times U(p)$ and $U(p) \times (0)$ under ψ, respectively. We will order $U_2(p)$ consistent with the 90^0-rotation by the following way. Set

$$\alpha_1 = (z, z^{-1}); \qquad \alpha_2 = (1, z). \tag{35}$$

Then, by direct product every element in $U(p) \times U(p)$ can be written uniquely as

$$\alpha_1^j \alpha_2^k, \qquad 0 \le j, k < p-1.$$

Set

$$\begin{aligned} A_1 &= \{\alpha_1^j\colon \quad 0 \le j < p-1\}; \\ A_2 &= \{\alpha_2^k\colon \quad 0 \le k < p-1\}. \end{aligned} \tag{36}$$

We order $U(p) \times U(p)$ by

$$A_2, \alpha_1 A_2, \cdots, \alpha_1^{p-2} A_2, \tag{37}$$

where A_2 is ordered by the powers of α_2, and order $U_2(p)$ relative to this ordering under ψ^{-1}.

Finally, we order $R(p)$ by (33). Relative to this ordering in input and output, and taking the operation and the function ϕ in $R(p)$, or by the equivalent operation (29) in $Z/p \times Z/p$, the matrix of G is

$$\begin{pmatrix} 1 & 1_{p'}^t & 1_{p'}^t & 1_{p'^2}^t \\ 1_{p'} & C(p) & I(p') & C(p) \otimes 1_{p'}^t \\ 1_{p'} & I(p') & C(p) & (1_{p'}^t \otimes C(p))S \\ 1_{p'^2} & C(p) \otimes 1_{p'} & S^{-1}(1_{p'} \otimes C(p)) & S^{-1}(C(p) \otimes C(p))S \end{pmatrix}, \tag{38}$$

where $p' = p-1$, $C(p)$ is the skew-circulant Winograd core

$$C(p) = \left[v^{z^{i+j}}\right]; \qquad 0 \le j, k < p', \quad v = exp(2\pi i/p), \tag{39}$$

and S is the matrix direct sum

$$S = I_{p'} \oplus S_{p'} \oplus \cdots \oplus S_{p'}^{p-2}, \tag{40}$$

with $S_{p'}$ the $p' \times p'$ cyclic shift matrix.

Now we consider the symmetry of the input data. Let

$$r = p'/4, \tag{41}$$

a generator z in $U(p)$ can be found such that $z^r = u$. Then

$$\alpha_1^r = (z^r, z^{-r}) = (u, -u) = \psi(i). \tag{42}$$

The action of α_1^r on $(0) \times U(p)$:

$$\alpha_1^r(0, z^j) = (0, z^{j-r}); \qquad 0 \leq j < p-1. \tag{43}$$

It follows that on $(0) \times U(p)$

$$f(0, z^j) = f(0, z^{j+r}) = f(0, z^{j+2r}) = f(0, z^{j+3r}). \tag{44}$$

Denoting by $\underline{f}_1$ the r-dimensional vector

$$\begin{pmatrix} f(0,1) \\ f(0,z) \\ \vdots \\ f(0,z^{r-1}) \end{pmatrix}, \tag{45}$$

we have

$$\begin{pmatrix} f(0,1) \\ f(0,z) \\ \vdots \\ f(0,z^{p-2}) \end{pmatrix} = (1_4 \otimes I_r)\underline{f}_1. \tag{46}$$

Related to $U(p) \times (0)$ and $U(p) \times U(p)$, denoting by $\underline{f}_2$ and $\underline{f}_3$ in the same way, respectively. Then the vector formed by the linear ordering of $R(p)$ with the indexing under ψ

$$\underline{f} = X \begin{pmatrix} f(0) \\ \underline{f}_1 \\ \underline{f}_2 \\ \underline{f}_3 \end{pmatrix}, \tag{47}$$

where

$$X = 1 \oplus (1_4 \otimes I_r) \oplus (1_4 \otimes I_r) \oplus (1_4 \otimes I_{rp'}), \tag{48}$$

and $\underline{f}$ is the vector formed from F by the linear ordering of $R(p)$. In particular, F is completely determined by its values on the points

$$0; (1,0), \cdots, (z^{r-1}, 0); (0,1), \cdots, (0, z^{r-1}); A_2, \cdots, \alpha_1^{r-1} A_2. \tag{49}$$

The output data $g = G(f)$ also can be written as in (47), and the computation can be reduced by

$$X^t G X \underline{f}. \tag{50}$$

A scaler multiplication by $\frac{1}{4}$ for each output in g except 0 is necessary. Set

$$T = (1 \oplus \frac{1}{4} \cdot I_{p^2-1}) \tag{51}$$

and

$$\underline{g} = Tg = TX^t G X \underline{f} = G_4(p)\underline{f}. \tag{52}$$

We now describe the matrix $G_4(p)$. Set

$$\$ = (1 \oplus I_{p'} \oplus I_{p'} \oplus S). \tag{53}$$

Then the matrix G in (38) can be rewritten as

$$G = \$^{-1} 1 G_1 \$, \tag{54}$$

where G_1 the matrix removing all the items S and S^{-1} from G. Now set

$$\begin{aligned} S_0 &= I_{p'} \oplus S_{p'} \oplus \cdots \oplus S_{p'}^{r-1}; \\ S_1 &= (I_r \otimes I_{p'}) \oplus (I_r \otimes S_{p'}^r) \oplus (I_r \otimes S_{p'}^{2r}) \oplus (I_r \otimes S_{p'}^{3r}), \end{aligned} \tag{55}$$

and observe that S can be written

$$S = S_1(I_4 \otimes S_0). \tag{56}$$

Set

$$\begin{aligned} \$_0 &= (1 \oplus I_r \oplus I_r \oplus S_0); \\ \$_1 &= (1 \oplus I_{p'} \oplus I_{p'} \oplus S_1). \end{aligned} \tag{57}$$

Then

$$X^t\mathcal{S}^{-1} = X^t(1 \oplus I_{p'} \oplus I_{p'} \oplus (I_4 \otimes S_0^{-1}))\mathcal{S}_1^{-1} = \mathcal{S}_0^{-1}X^t\mathcal{S}_1^{-1} \tag{58}$$

and

$$G_4(p) = T\mathcal{S}_0^{-1}X^t\mathcal{S}_1^{-1}G_1\mathcal{S}_1X\mathcal{S}_0. \tag{59}$$

Write the Winograd core $C(p)$ in block format,

$$C(p) = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 & c_0 \\ c_2 & c_3 & c_0 & c_1 \\ c_3 & c_0 & c_1 & c_2 \end{pmatrix}, \tag{60}$$

and set

$$H_0 = c_0 + c_1 + c_2 + c_3. \tag{61}$$

By direct computation,

$$TX^t\mathcal{S}_1^{-1}G_1\mathcal{S}_1X = \begin{pmatrix} 1 & 4\cdot 1_r^t & 4\cdot 1_r^t & 4\cdot 1_{rp'}^t \\ 1_r & H_0 & 4\cdot I(r) & H_0 \otimes 1_{p'}^t \\ 1_r & 4\cdot I(r) & H_0 & 1_{p'}^t \otimes H_0 \\ 1_{rp'} & H_0 \otimes 1_{p'} & 1_{p'} \otimes H_0 & J_0 \end{pmatrix}, \tag{62}$$

where

$$\begin{aligned} J_0 = {} & (c_0 \otimes C(p)) + (c_1 \otimes C(p)S_{p'}^r) + \\ & (c_2 \otimes C(p)S_{p'}^{2r}) + (c_3 \otimes C(p)S_{p'}^{3r}). \end{aligned} \tag{63}$$

The matrix (62) can be decomposed into the action of a pre-addition matrix followed by a 'multiplications' matrix

$$TX^t\mathcal{S}_1^{-1}G_1\mathcal{S}_1X = H'A', \tag{64}$$

a direct computation shows that

$$H' = 1 \oplus H_0 \oplus H_0 \oplus J_0 \tag{65}$$

and

$$A' = \begin{pmatrix} 1 & 4\cdot 1_r^t & 4\cdot 1_r^t & 4\cdot 1_{rp'}^t \\ -1_r & I_r & -4\cdot I(r) & I_r \otimes 1_{p'}^t \\ -1_r & -4\cdot I(r) & I_r & 1_{p'}^t \otimes I_r \\ 1_{rp'} & -I_r \otimes 1_{p'} & -1_{p'} \otimes I_r & I_{rp'} \end{pmatrix}. \tag{66}$$

Now we consider how to compute H'. By the property of 90° rotation, all the entries are real in H'. H_0 is skew-circulant, but J_0 is not. Because any matrix having the form $a \cdot C(p)S_{p'}^k$ is also skew-circulant, where a is any constant number. From (63), we see that J_0 is $r \times r$ block half-circulant having $p' \times p'$ skew-circulant blocks,

$$J_0 = \begin{pmatrix} h_0 & h_1 & \cdots & h_{r-1} \\ h_1 & h_2 & \cdots & h_0' \\ \vdots & & & \vdots \\ h_{r-1} & h_0' & \cdots & h_{r-2}' \end{pmatrix}, \tag{67}$$

where h_k and h_k' are all skew-circulant matrix of order p'. Now we try to figure out the relation between h_k and h_k'. If we represent the entries of $C(p)$ by v^{z^i}, from (63),

$$\begin{aligned} h_k &= v^{z^k}C(p) + v^{z^{k+r}}C(p)S_{p'}^r + v^{z^{k+2r}}C(p)S_{p'}^{2r} + v^{z^{k+3r}}C(p)S_{p'}^{3r}; \\ h_k' &= v^{z^{k+r}}C(p) + v^{z^{k+2r}}C(p)S_{p'}^r + v^{z^{k+3r}}C(p)S_{p'}^{2r} + v^{z^k}C(p)S_{p'}^{3r}, \end{aligned} \tag{68}$$

then we have that

$$h_k' = h_k S_{p'}^{3r} = S_{p'}^{-r} h_k. \tag{69}$$

By the Block-shift theorem we discussed in chapter 5, the matrix J_0 can be changed into block skew-circulant if there is a solution δ_s satisfying

$$\delta_s^r S_{p'}^{-r} = I_{p'}. \tag{70}$$

Obviously, one of the solutions is

$$\delta_s = S_{p'}. \tag{71}$$

Then,

$$J_0 = T_s J_0' T_s^{-1}, \tag{72}$$

where

$$T_s = (I_{p'} \oplus S_{p'} \oplus S_{p'}^2 \oplus \cdots \oplus S_{p'}^{r-1}) = S_0, \tag{73}$$

and J_0' is $r \times r$ block skew-circulant having $p' \times p'$ skew-circulant blocks. Denote by the blocks in J_0' by h_i'', $0 \le i < r-1$. By (68) and (69), we have

$$h_k'' = h_k S_{p'}^k = \sum_{j=0}^{3} v^{z^{k+jr}} C(p) S_{p'}^{k+jr}. \tag{74}$$

Set

$$H = 1 \oplus H_0 \oplus H_0 \oplus J_0', \tag{75}$$

then

$$H' = \mathcal{S}_0 H \mathcal{S}_0^{-1}. \tag{76}$$

The matrix $G_4(p)$ in (59) now can be rewritten as

$$G_4(p) = H\mathcal{S}_0^{-1} A' \mathcal{S}_0 = HA, \tag{77}$$

where

$$\begin{aligned} A &= \mathcal{S}_0^{-1} A' \mathcal{S}_0 \\ &= \begin{pmatrix} 1 & 4 \cdot 1_r^t & 4 \cdot 1_r^t & 4 \cdot 1_{rp'}^t \\ -1_r & I_r & -4 \cdot I(r) & I_r \otimes 1_{p'}^t \\ -1_r & -4 \cdot I(r) & I_r & (1_{p'}^t \otimes I_r) S_0 \\ 1_{rp'} & -I_r \otimes 1_{p'} & -S_0^{-1}(1_{p'} \otimes I_r) & I_{rp'} \end{pmatrix}. \end{aligned} \tag{78}$$

Like the 1-dimensional case, the computation of A can also be replaced by a matrix B. Set

$$F = (1 \oplus F(r) \oplus F(r) \oplus (F(r) \otimes F(p'))). \tag{79}$$

Then

$$G_4(p) = FDFA, \tag{80}$$

where $D = F^{-1} H F^{-1}$ is the diagonal matrix. Let

$$B = FAF^{-1}. \tag{81}$$

Direct computation shows that

$$B = \begin{pmatrix} 1 & 4 \cdot \alpha_r^t & 4 \cdot \alpha_r^t & 4 \cdot \alpha_{rp'}^t \\ -r \cdot \alpha_r & I_r & -p'(\alpha_r \otimes \alpha_r^t) & I_r \otimes \alpha_{p'}^t \\ -r \cdot \alpha_r & -p'(\alpha_r \otimes \alpha_r^t) & I_r & \alpha_r \otimes \alpha_{p'}^t \\ rp' \cdot \alpha_{rp'} & -p'(I_r \otimes \alpha_{p'}) & -p'(\alpha_{p'} \otimes \alpha_r^t) & I_{rp'} \end{pmatrix}, \tag{82}$$

which can be computed by $(r+5)$ rational multiplications and $(2r+6)$ additions. The formula (80) now becomes

$$G_4(p) = FDBF. \tag{83}$$

Because H is real, we may prefer to use real-type algorithm for better performance. To compute J_0', we may permute it into $p' \otimes p'$ block format if it is better. Also, If the pre-addition matrices of r-point algorithm and p'-point algorithm have the property (33) in chapter 8, as we discussed in chapter 9, a similar variant form as (83) can be obtained also.

The input and output permutation for this case is a little complicated, because it is the combination of the ordering (37) and the function ψ in (25). Some effort has been made to reduce this overhead. Let two 1-dimensional arrays $D_1(r)$, $D_2(r)$ and one 2-dimensional array $D(r, p')$ hold input data corresponding to the indexing set of D_1, D_2 and $\alpha_1^i A_2$ respectively. Suppose that

$$D_1(i) = f(a_1, b_1); \qquad D_2(i) = f(a_2, b_2); \qquad D(i,j) = f(a, b).$$

Then, we have

$$a_1 = \frac{1}{2}z^i, \qquad b_1 = -\frac{1}{2}u^{-1}z^i;$$

$$a_2 = \frac{1}{2}z^i, \qquad b_2 = \frac{1}{2}u^{-1}z^i;$$

$$a = \frac{1}{2}(z^i + z^{j-i}), \qquad b = \frac{1}{2}u^{-1}(z^i - z^{j-i}).$$

Suppose that

$$\frac{1}{2} = z^{k_0}; \qquad \frac{1}{2}u^{-1} = z^{k_1}; \qquad -\frac{1}{2}u^{-1} = z^{k_2}.$$

Now the two permutations have been taken into one. The following program of implementing this permutation can reduce the computation.

```
INTEGER      Z(0 : 2p' - 1), i, j, i_0, j_0, i_1, j_1, i_2, j_2
...
DO 10 i = 0, r - 1
  i_0 = k_0 + i
  i_2 = k_0 - i
  j_0 = k_1 + i
  j_2 = k_1 - i
  D_1(i) = f(Z(i_0), Z(i + k_2))
  D_2(i) = f(Z(i_0), Z(j_0))
  DO 10 j = 0, p' - 1
    i_1 = Z(i_0) + Z(j + i_2)
```

```
      IF (i1 .GE. p) i1 = i1 - p
      j1 = Z(j0) - Z(j + j2)
      IF (j1 .LT. 0) j1 = j + p
      D(i,j) = f(i1,j1)
10    CONTINUE
```

Only additions and IF statements are used instead of MODulo function. The IF operation is much faster than the MODulo function.

Example 3. Take $p = 5$ and use $z = 2$, the ordering (49) and the corresponding input and output are listed below,

	D_2	D_1		A_2		
input	(3,4)	(3,1)	(1,0)	(4,1)	(0,3)	(2,2)
ψ	[1,0]	[0,1]	[1,1]	[1,2]	[1,4]	[1,3]
input	(3,1)	(3,4)	(1,0)	(4,4)	(0,2)	(2,3)

Because $r = 1$, then $H_0 = -1$, $S_0 = I_4$ and

$$J_0' = J_0 = \begin{pmatrix} 2(1+cos(2\theta) & 4cos(2\theta) & 2(1+cos\theta) & 4cos\theta \\ 4cos(2\theta) & 2(1+cos\theta) & 4cos\theta & 2(1+cos(2\theta) \\ 2(1+cos\theta) & 4cos\theta & 2(1+cos(2\theta) & 4cos(2\theta) \\ 4cos\theta & 2(1+cos(2\theta) & 4cos(2\theta) & 2(1+cos\theta) \end{pmatrix},$$

which is a real skew-circulant matrix. By a 4-point real-type algorithm,

$$J_0 = C_4 D M_4.$$

Now look at the pre-addition matrix A,

$$A = \begin{pmatrix} 1 & 4 & 4 & 4 \cdot 1_4^t \\ -1 & 1 & -4 & 1_4^t \\ -1 & -4 & 1 & 1_4^t \\ 1_4 & -1_4 & -1_4 & I_4 \end{pmatrix}.$$

In this case, a matrix B can be found such that

$$B \begin{pmatrix} 1 & & & \\ & -1 & & \\ & & -1 & \\ & & & M_4 \end{pmatrix} = (I_3 \oplus M_4)A,$$

where

$$B = \begin{pmatrix} 1 & 4 & 4 & 4\alpha_5^t \\ 1 & -1 & 4 & -\alpha_5^t \\ 1 & 4 & -1 & -\alpha_5^t \\ 4\alpha_5 & -4\alpha_5 & -4\alpha_5 & I_5 \end{pmatrix}.$$

Here we take the computation of $H_0 = -1$ into A. Notice that M_4 is 5×4 matrix. The computation of B needs 4 multiplications by 4 and 8 additions. The final version of $G_4(5)$ would be

$$G_4(5) = (I_3 \oplus C_4)(I_3 \oplus D)B(I_3 \oplus M_4).$$

The total arithmetic count is 9 multiplications and 23 additions. Compared with the 5-point 2-dimensional FFT without the symmetry, by the row-column algorithm 50 multiplications and 170 additions are needed. About 80% of performance can be saved by using this data redundancy.

11.4 Summary

The use of the data redundancy makes the computation reduced a lot. For the symmetry with 90°-rotation, usually, the computation is more than 5 times faster than the computation without the symmetry. There may have many ways to use this data redundancy. By the algorithms we discussed in this chapter, for the case of $p \equiv 3 \bmod 4$, we can find that, like the 1-dimensional prime case FFT algorithm, the computational matrix H_0 is also skew-circulant and keeps all the properties of the Winograd core $C(p)$, but here H_0 is real matrix. And, the matrix H_0 has the same special form as the block D_0 in the example 2 of chapter 10. By using the $2^k r$-algorithm on H_0, the computation can be further reduced. For the case of $p \equiv 1 \bmod 4$, the computational matrix J_0' is r-point block skew-circulant matrix having p'-point skew-circulant blocks, the multiplicative algorithm can be used without any permutation, but notice that here r and p' are not necessarily relative prime. Because the matrix J_0' is also real, the real-type algorithm will play an important role, too. By the new algorithms we developed in chapter 7 and chapter 8, the symmetrized FFT can be computed more efficiently.

CHAPTER 12

SYMMETRIZED FFT ALGORITHMS WITH 120° ROTATION

12.1 Introduction

Suppose that F denotes input data and is invariant under 120°-rotation. Then

$$f(a_0, a_1) = f(-a_1, a_0 - a_1); \qquad a_0, a_1 \in Z/n. \tag{1}$$

Setting $\rho = (0, 1)$ and corresponding to the ring-structure $Z/n[x]/(x^2 + x + 1)$, we have

$$\rho \circ a = (-a_1, a_0 - a_1);$$

$$\rho^3 = 1, \qquad a = (a_0, a_1) \in Z/n \times Z/n, \tag{2}$$

which corresponds to 120°-rotation. Then

$$f(\rho \circ a) = f(a), \tag{3}$$

and also

$$f(\rho^2 \circ a) = f(\rho \circ a) = f(a). \tag{4}$$

From (12) of chapter 10, direct computation shows

$$G(f)[b] = G(f)[(b_1, -b_0 - b_1]) = G(f)[(-b_0 - b_1, b_0)];$$

$$b = (b_0, b_1) \in Z/n \times Z/n, \tag{5}$$

which, unfortunately, is different from input symmetry. We begin with the discussion of the relationship between two symmetries.

Corresponding to the ring-structure $Z/n[x]/(x^2 - x + 1)$, the multiplication is defined as

$$(a_0, a_1) \bullet (b_0, b_1) = (a_0 b_0 - a_1 b_1, a_0 b_1 + a_1 b_0 + a_1 b_1). \tag{6}$$

Set $\rho_0 = (0, 1)$, then

$$\rho_0^3 = -1; \qquad (-\rho_0)^3 = 1 \tag{7}$$

under the multiplication $\bullet$. We have

$$(-\rho_0) \bullet (b_0, b_1) = (b_1, -b_0 - b_1), \tag{8}$$

which corresponds to the output symmetry in (5), it now can be rewritten as

$$G(f)[b] = G(f)[-\rho_0 \bullet b] = G(f)[\rho_0^2 \bullet b]. \tag{9}$$

An isomorphism between the two ring-structures $Z/n[x]/(x^2 + x + 1)$ and $Z/n[x]/(x^2 - x + 1)$ can be established by setting

$$\psi_0(a) = (a_0, -a_1); \qquad a = (a_0, a_1) \in Z/n \times Z/n, \tag{10}$$

because

$$\psi_0(a \circ b) = \psi_0(a) \bullet \psi_0(b). \tag{11}$$

If we set

$$\phi(a) = a_0, \qquad a = (a_0, a_1) \in Z/n \times Z/n, \tag{12}$$

then

$$\phi(a \circ b) = \phi(a \bullet b) = a_0 b_0 - a_1 b_1, \tag{13}$$

which implies that under either ring-structure the output permutation is equivalent to the function ψ_0. Because

$$\psi_0(\rho) = -\rho_0, \tag{14}$$

then the output symmetry is identical with the input symmetry under ψ_0.

Example 1. Take $p = 3$. Because $\rho \circ (1,2) = (1,2)$, $\rho \circ (2,1) = (2,1)$ and $(1,1)^2 = \rho$, we order the input by

$$(0,0),(1,2),(2,1);(1,0),(1,1);(0,1),(2,0);(2,2),(0,2).$$

Then, the input data is determined by the values in the set

$$(0,0),(1,2),(2,1);(1,0),(1,1);$$

the corresponding output sequence under ψ_0 is

$$(0,0),(1,1),(2,2);(1,0),(1,2).$$

Applying the input and output symmetries, the computation matrix is

$$\begin{pmatrix} 1 & 1_2^t & 3\cdot 1_2^t \\ 1_2 & I(2) & 3c_0 \\ 1^2 & c_0 & 0 \end{pmatrix}; \qquad c_0 = \begin{pmatrix} v & v^2 \\ v^2 & v \end{pmatrix},$$

which can be computed by 12 additions and 5 multiplications.

We will discuss two basic cases of $p \equiv 2 \bmod 3$ and $p \equiv 1 \bmod 3$ in the following sections.

12.2 The Case of $p \equiv 2 \bmod 3$

In this case we have proved that $Z/p[x]/(x^2+x+1)$ is the field $GF(p^2)$. A generator γ of the set $U_2(p)$ of non-zero elements in $R(p)$ can be obtained by

$$\gamma^r = \rho; \qquad r = (p^2-1)/3. \tag{15}$$

By the 120° symmetry in (3), f is completely determined by its values in the set

$$0; 1, \gamma, \cdots, \gamma^{r-1}, \tag{16}$$

and $C_2(p)$ in the comp[illegible] $\backslash f(\gamma^{r-1}) /$ format

and $C_2(p)$ in the co[illegible]where f denotes input data linearly ordered by (16). format

No

$$C_2(p) = \begin{pmatrix} c_0 & c_1 & c_2 \\ c_1 & c_2 & c_0 \\ c_2 & c_0 & c_1 \end{pmatrix}. \tag{17}$$

$$X = \begin{pmatrix} & I_r & \\ & I_r & \\ & I_r & \end{pmatrix} = 1 \oplus (1_3 \otimes I_r), \tag{18}$$

we have

$$\underline{f} = X \begin{pmatrix} f(0) \\ f(1) \\ f(\gamma) \\ \vdots \\ f(\gamma^{r-1}) \end{pmatrix}, \tag{19}$$

where $\underline{f}$ denotes input data linearly ordered by (16).

Output data $g = G(f)$, as we discussed, is identical with 120°-rotation under ψ_0, which reduces the computation to

$$\begin{pmatrix} g(0) \\ 3g(1) \\ 3g(\gamma) \\ \vdots \\ 3g(\gamma^{r-1}) \end{pmatrix} = X^t G X \begin{pmatrix} f(0) \\ f(1) \\ f(\gamma) \\ \vdots \\ f(\gamma^{r-1}) \end{pmatrix}. \tag{20}$$

Direct computation shows that

$$X^t G X = \begin{pmatrix} 1 & 3 \cdot 1_r^t \\ 3 \cdot 1_r & 3 \cdot H_0 \end{pmatrix}, \tag{21}$$

where

$$H_0 = c_0 + c_1 + c_2 \tag{22}$$

is a skew-circulant matrix of order r. Because

$$\gamma^{\frac{3r}{2}} = -1, \tag{23}$$

then

$$\gamma^{\frac{r}{2}+i} = -\gamma^{2r+i}; \qquad \gamma^{\frac{3r}{2}+i} = -\gamma^{i}; \qquad \gamma^{\frac{5r}{2}+i} = -\gamma^{r+i}. \tag{24}$$

Denote by a_i and $a_{\frac{r}{2}+i}$ the i-th element and $(\frac{r}{2}+i)$-th element in the same row of H_0, it is easy to find that

$$a_{\frac{r}{2}+i} = a_i^*, \tag{25}$$

which implies that H_0 keeps the conjugate property.

Now set

$$T = 1 \oplus \frac{1}{3} \cdot I_r \tag{26}$$

and

$$\underline{g} = Tg = G_3(p)\underline{f}. \tag{27}$$

Then

$$G_3(p) = TX^t G X = \begin{pmatrix} 1 & 3 \cdot 1_r^t \\ 1_r & H_0 \end{pmatrix}. \tag{28}$$

Now the computation matrix $G_3(p)$ is really similar to the matrix $G_4(p)$ in the preceding chapter. The H_0 is skew-circulant and the size

has the format $2^k \cdot r$ too. The variant forms of $G_3(p)$ can also be obtained by

$$G_3(p) = (1 \oplus H_0)A', \tag{29}$$

where

$$A' = \begin{pmatrix} 1 & 3 \cdot 1_r^t \\ -1_r & I_r \end{pmatrix}, \tag{30}$$

and also, by the corresponding r-point cyclic convolution algorithm,

$$G_3(p) = (1 \oplus C_r)DB'(1 \oplus M_r), \tag{31}$$

where

$$B' = \begin{pmatrix} 1 & 3 \cdot \alpha_r^t \\ -r \cdot 1_r & I_r \end{pmatrix}. \tag{32}$$

By the way we find a generator γ in the preceding chapter, we have that

$$\gamma^r = (z^{p'})^{\frac{1}{3}} = 1^{\frac{1}{3}} \tag{33}$$

the value of which is either ρ or ρ^2, the input symmetry is unchanged. The output permutations would have to add the mapping ψ_0. Instead, we may set

$$\gamma_0 = \psi_0(\gamma) \tag{34}$$

which is the generator of the output sequence under the ring-structure $Z/p[x]/(x^2 - x + 1)$, and is identical with the ordering (16) under ψ_0. The implementing procedure would be the same as for the input except that the pre-stored data for W_2 is different.

Example 2. Take $p = 5$. Recall this example in the section 4 of chapter 10, and by the conjugate property (24), we have that

$$H_0 = \begin{pmatrix} c & c^* \\ c^* & c \end{pmatrix},$$

where

$$c = \begin{pmatrix} 1+v+v^4 & 2v+v^3 & 1+v^2+v^3 & 2v^2+v \\ 2v+v^3 & 1+v^2+v^3 & 2v^2+v & 1+v+v^4 \\ 1+v^2+v^3 & 2v^2+v & 1+v+v^4 & 2v^4+v^2 \\ 2v^2+v & 1+v+v^4 & 2v^4+v^2 & 1+v^2+v3 \end{pmatrix}$$

which is a half-circulant matrix.

Directly applying the 8-point Winograd algorithm on H_0, because some multiplications and additions can be saved, 41 additions and 13

multiplications are used, which is less than 46 additions and 14 multiplications for general case. But, notice that the elements $1+v+v^4$ and $1+v^2+v^3$ are real numbers, which implies that their conjugates in c^* will be the same as these two elements. If the 2^k-algorithm is used we only need 39 additions and 13 multiplications, which is even better.

Denote by D_0 and D_1 the two blocks in the block-diagonal matrix. By 2-point algorithm, then

$$D_0 = \frac{1}{2}(c + c^*); \qquad D_1 = \frac{1}{2}(c - c^*),$$

where D_0 is a real skew-circulant matrix. Our interest now is in D_1,

$$D_1 = i \cdot \begin{pmatrix} 0 & a & 0 & b \\ a & 0 & b & 0 \\ 0 & b & 0 & -a \\ b & 0 & -a & 0 \end{pmatrix};$$

$$a = 2sin\theta - sin(2\theta), \qquad b = 2sin(2\theta) + sin\theta,$$

which consists of two half-circulant matrices. By 2-point negative algorithm, only 6 multiplications and 6 additions are needed, which makes a lot of savings. The form of D_1 is in general, we now prove that it is one of the properties of H_0.

First, denote by a_i the i-th element in the first row of H_0. Then

$$a_i = v^{\phi(\gamma^i)} + v^{\phi(\gamma^{r+i})} + v^{\phi(\gamma^{2r+i})}. \tag{35}$$

If $\gamma^i = b\rho$, then

$$\gamma^{r+i} = b\rho^2; \qquad \gamma^{2r+i} = b. \tag{36}$$

Because

$$\phi(b\rho) = 0; \qquad \phi(b\rho^2) = -b = -\phi(b), \tag{37}$$

obviously, a_i is a real number. Now set

$$k = \frac{p+1}{3}. \tag{38}$$

Then

$$\gamma^{r+k+i} = (b\rho) \cdot \gamma^{\frac{(p+1)p}{3}} = (b\rho) \cdot z^{\frac{p}{3}} = (b\rho) \cdot z^{\frac{1}{3}}. \tag{39}$$

by the way we find the generator γ,

$$(z^{\frac{1}{3}}\rho)^3 = z; \qquad or \qquad (z^{\frac{1}{3}}\rho^2)^3 = z, \tag{40}$$

the value of (39) is either $a\rho^2$ or a, $a = bz^{\frac{1}{3}}$. Then

$$\gamma^{2r+k+i} = a \quad (or \quad a\rho), \tag{41}$$

which implies that the element a_{k+i} is also real. Because

$$\gamma^{p+1} = \gamma^{3k}, \tag{42}$$

we have the following result,

$$a_{jk} = a, \tag{43}$$

where a is a real number. This property can be used to reduce the computation of H_0.

12.3 The Case of $p \equiv 1 \bmod 3$

In this case, there exists $u \in U(p)$ such that

$$u^3 \equiv 1 \bmod p, \tag{44}$$

and the polynomial $x^2 + x + 1$ is not irreducible over Z/p. In fact

$$x^2 + x + 1 = (x - u)(x - u^2), \tag{45}$$

and $Z/p\,[x]\,/(x^2 + x + 1)$ is not a field. Denote by $R(p)$ the indexing set $Z/p \times Z/p$ with this ring-structure. By the Chinese Remainder theorem, we have a ring-isomorphism ψ

$$R(p) \cong Z/p \times Z/p \qquad (direct \quad product), \tag{46}$$

given by

$$\psi(a) = (a_0 + a_1 u, a_0 + a_1 u^2); \qquad a = (a_0, a_1) \in R(p). \tag{47}$$

Because $u + u^2 = -1$, The operation

$$\begin{aligned} &\psi(a \circ b) = \psi(a)\psi(b) = (a', b'); \\ &\quad a' = ((a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0 - a_1 b_1)u; \\ &\quad b' = (a_0 b_0 - a_1 b_1) + (a_0 b_1 + a_1 b_0 - a_1 b_1)u^2). \end{aligned} \tag{48}$$

By the definition for 2-dimensional DFT,

$$< \psi(a), \psi(b) >= a_0(2b_0 - b_1) - a_1(b0 + b_1), \tag{49}$$

which shows that

$$G(f)(b) = F(f)(2b_0 - b_1, -b_0 - b_1). \tag{50}$$

The output indexing permutation matrix would be

$$\begin{pmatrix} 2 & -1 \\ -1 & -1 \end{pmatrix}. \tag{51}$$

Now we can use the same way as we did for the case of $p \equiv 1 \bmod 4$ in the preceding chapter. Partition $R(p)$ into the subsets

$$0; D_2; D_1; U_2(p), \tag{52}$$

corresponding to the subsets 0, $U(p) \times (0)$, $(0) \times U(p)$ and $U(p) \times U(p)$ under ψ respectively. And also set

$$\alpha_1 = (z, z^{-1}); \qquad \alpha_2 = (1, z). \tag{53}$$

Order $R(p)$ as we did in the preceding chapter, by the operation (49), the matrix of G is the same as (38) of the chapter 11. Let

$$r = \frac{p-1}{3}, \tag{54}$$

a generator z in $U(p)$ can be found such that $z^r = u$. Then

$$\alpha_1^r = (z^r, z^{-r}) = (u, u^2) = \psi(\rho). \tag{55}$$

The input symmetry now is

$$X = 1 \oplus (1_3 \otimes I_r) \oplus (1_3 \otimes I_r) \oplus (1_3 \otimes I_{rp'}). \tag{56}$$

The computation matrix after symmetries would be

$$G_3(p) = TX^tGX; \qquad T = (1 \oplus \frac{1}{3} \cdot I_{p^2-1}). \tag{57}$$

Write the Winograd core $C(p)$ in block format

$$C(p) = \begin{pmatrix} c_0 & c_1 & c_2 \\ c_1 & c_2 & c_0 \\ c_2 & c_0 & c_1 \end{pmatrix}, \tag{58}$$

and set

$$H_0 = c_0 + c_1 + c_2. \tag{59}$$

Then

$$G_3(p) = \$_0^{-1} \begin{pmatrix} 1 & 4 \cdot 1_r^t & 4 \cdot 1_r^t & 4 \cdot 1_{rp'}^t \\ 1_r & H_0 & 4 \cdot I(r) & H_0 \otimes 1_{p'}^t \\ 1_r & 4 \cdot I(r) & H_0 & 1_{p'}^t \otimes H_0 \\ 1_{rp'} & H_0 \otimes 1_{p'} & 1_{p'} \otimes H_0 & J_0 \end{pmatrix} \$_0, \tag{60}$$

where

$$J_0 = (C_0 \otimes C(p)) + (C_1 \otimes C(p)S_{p'}^r) + (C_2 \otimes C(p)S_{p'}^{2r}). \tag{61}$$

The variant form of $G_3(p)$ is

$$G_3(p) = \$_0^{-1} H' A' \$_0, \tag{62}$$

where

$$H' = 1 \oplus H_0 \oplus H_0 \oplus J_0 \tag{63}$$

and

$$A' = \begin{pmatrix} 1 & 3 \cdot 1_r^t & 3 \cdot 1_r^t & 3 \cdot 1_{rp'}^t \\ -1_r & I_r & -3 \cdot I(r) & I_r \otimes 1_{p'}^t \\ -1_r & -3 \cdot I(r) & I_r & 1_{p'}^t \otimes I_r \\ 1_{rp'} & -I_r \otimes 1_{p'} & -1_{p'} \otimes I_r & I_{rp'} \end{pmatrix}. \tag{64}$$

Also, as we proved

$$G_3(p) = HA, \tag{65}$$

where

$$A = \$_0^{-1} A' \$_0 = \begin{pmatrix} 1 & 3 \cdot 1_r^t & 3 \cdot 1_r^t & 3 \cdot 1_{rp'}^t \\ -1_r & I_r & -3 \cdot I(r) & I_r \otimes 1_{p'}^t \\ -1_r & -3 \cdot I(r) & I_r & (1_{p'}^t \otimes I_r)S_0 \\ 1_{rp'} & -I_r \otimes 1_{p'} & -S_0^{-1}(1_{p'} \otimes I_r) & I_{rp'} \end{pmatrix} \tag{66}$$

and

$$H = 1 \oplus H_0 \oplus H_0 \oplus J_0'. \tag{67}$$

In this case, H_0 is not real. But by (24) that we have proved, H_0 keeps the conjugate property. Now we prove that the conjugate property holds for J_0' too. Denote by h_i'' the blocks in J_0'. By (74) of chapter 11,

$$h_i'' = \sum_{j=0}^{2} v^{z^{i+jr}} C(p) S_{p'}^{i+jr}. \tag{68}$$

Because of the following facts

$$z^{k+\frac{r}{2}} = -z^k; \tag{69}$$

$$C(p)S_{p'}^{k+\frac{r}{2}} = \left(C(p)S_{p'}^{k}\right)^* = C(p)^* S_{p'}^{k}; \tag{70}$$

$$a^* b^* = (ab)^*, \tag{71}$$

then

$$v^{z^{(i+jr)+\frac{r}{2}}} C(p) S_{p'}^{(i+jr)+\frac{r}{2}} = (v^{z^{i+jr}} C(p) S_{p'}^{i+jr})^*. \tag{72}$$

Also because

$$a^* + b^* = (a+b)^*, \tag{73}$$

we have that

$$h_{i+\frac{r}{2}} = \sum_{j=0}^{2} v^{z^{(i+jr)+\frac{r}{2}}} C(p) S_{p'}^{(i+jr)+\frac{r}{2}} = h_i^*, \tag{74}$$

which implies that the conjugate property holds for J_0'. We see that the 2^k and $2^k \cdot r$-algorithms also play a very important role in this case. By the corresponding algorithm, the computation of A in (66) can also be replaced by a matrix B as we did in chapter 11.

The input and output permutation for this case is a little complicated, too. For input data, Let

$$\begin{aligned} D_1(i) &= f(a_1, b_1); \\ D_2(i) &= f(a_2, b_2); \\ D(i,j) &= f(a,b). \end{aligned} \tag{75}$$

Then under ψ^{-1}, we have

$$\begin{aligned} a_1 &= \frac{1-u}{3} z^i, \qquad b_1 = \frac{u^2-u}{3} z^i; \\ a_2 &= \frac{1-u^2}{3} z^i, \qquad b_2 = \frac{u-u^2}{3} z^i = -b_1; \\ a &= \frac{1-u}{3} z^i + \frac{1-u^2}{3} z^{j-i}, \qquad b = \frac{u^2-u}{3}(z^i - z^{j-i}). \end{aligned} \tag{76}$$

Suppose that

$$\frac{1-u}{3} = z^{k_0}; \qquad \frac{u^2-u}{3} = z^{k_1}; \qquad \frac{1-u^2}{3} = z^{k_2}.$$

Then the program in chapter 11. can be modified to implement this permutation.

The output permutation would be the combination of the above permutation and the permutation (51). Denote the corresponding output indices by (a_i', b_i'). The action of the permutation (51) after the above mapping will give the following results,

$$\begin{aligned} a_1' &= 2a_1 - b_1 = z^i, \qquad & b_1' &= -a_1 - b_1 = uz^i; \\ a_2' &= 2a_2 - b_2 = z^i, \qquad & b_2' &= -a_2 - b_2 = u^2 z^i; \\ a' &= 2a - b = z^i + z^{j-i}, \qquad & b' &= -a - b = uz^i + u^2 z^{j-i}, \end{aligned} \tag{77}$$

which is even simpler than the input permutation.

<u>Example</u> 3. Taking $p = 7$ and the generator $z = 3$, the ordering (52) with the symmetry and the corresponding indices of input and output are listed below.

	D_2				D_1	
input	(2,3)	(6,2)			(6,4)	(4,5)
ψ	[1,0]	[3,0]			[0,1]	[0,3]
output	(1,2)	(3,6)			(1,4)	(3,5)
			A_2			
input	(1,0)	(6,1)	(0,4)	(3,6)	(5,5)	(4,2)
ψ	[1,1]	[1,3]	[1,2]	[1,6]	[1,4]	[1,5]
output	(2,6)	(4,0)	(3,3)	(0,5)	(5,4)	(6,1)
			$\alpha_1 A_2$			
	(1,1)	(5,6)	(3,0)	(4,3)	(0,5)	(2,4)
	[3,5]	[3,1]	[3,3]	[3,2]	[3,6]	[3,4]
	(1,5)	(4,3)	(6,4)	(5,0)	(2,2)	(0,1)

Because $r = 2$, then

$$H_0 = \begin{pmatrix} v + v^2 + v^4 & v^3 + v^6 + v^5 \\ v^3 + v^6 + v^5 & v + v^2 + v^4 \end{pmatrix},$$

which is a 2-point skew-circulant matrix with conjugate property. 4 additions and 2 multiplications are needed for the computation of H_0. J_0' can be written as the block form,

$$J_0' = \begin{pmatrix} c & c^* \\ c^* & c \end{pmatrix}.$$

Using $2^k \cdot r$-algorithm, at the first step, we have

$$D_0 = \frac{1}{2}(c + c^*); \qquad D_1 = \frac{1}{2}(c - c^*),$$

where D_0 is also skew-circulant. Again by the 6-point $2^k \cdot r$-algorithm, D_0 can be computed by 8 multiplications and 34 additions. Now look at D_1,

$$D_1 = i \cdot \begin{pmatrix} a_0 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & a_1 & 0 & a_2 & 0 & -a_0 \\ a_1 & 0 & a_2 & 0 & -a_0 & 0 \\ 0 & a_2 & 0 & -a_0 & 0 & -a_1 \\ a_2 & 0 & -a_0 & 0 & -a_1 & 0 \\ 0 & -a_0 & 0 & -a_1 & 0 & -a_2 \end{pmatrix},$$

where

$$a_0 = sin(2\theta) - 2sin\theta;$$
$$a_1 = sin\theta + 2sin(3\theta);$$
$$a_2 = -(sin(3\theta) + 2sin(2\theta)).$$

By δ-algorithm, D_1 can be computed by two 3-point cyclic convolution algorithms, which need 8 multiplications and 22 additions. The whole computation of J_0' will use 16 multiplications and 80 additions. It is even better than the general 12-point $2^k \cdot r$-algorithm which needs 20 multiplications and 92 additions. The computation of $G_3(7)$ now can be computed by (65) or by its variant form.

The matrix D_1 has the same form as the matrix D_1 in the example of $p = 5$ in the preceding section. Now we prove that a similar property to (43) holds for J_0' too. Denote by a_k the k-th element in the first row of J_0'. Then for some i, k',

$$k = ip' + k'. \tag{78}$$

By the formula (68) for the blocks h_i'', we have that

$$a_k = v^{z^i} \cdot v^{z^{-i+k'}} + v^{z^{i+r}} \cdot v^{z^{-i-r+k'}} + v^{z^{i+2r}} \cdot v^{z^{-i-2r+k'}}. \tag{79}$$

There must be a pair i, k', such that

$$z^i + z^{-i+k'} = 0, \tag{80}$$

then,

$$k' = 2i + \frac{3r}{2}. \tag{81}$$

By $z^{\frac{3r}{2}} = -1$, we have that

$$\begin{aligned}
&(z^{i+r} + z^{-i-r+k'}) + (z^{i+2r} + z^{-i-2r+k'}) \\
&= z^{i+r} + z^{i+\frac{r}{2}} + z^{i+2r} + z^{i-\frac{r}{2}} \\
&= z^{i+r} - z^{i+2r} + z^{i+2r} - z^{i+r} \\
&= 0,
\end{aligned} \tag{82}$$

which implies that a_k is real. Let $k_1 = k + r$, then

$$\begin{aligned}
z^{i+2r} + z^{-i-2r+k_1} &= z^{i+2r} + z^{i-r+\frac{3r}{2}} \\
&= z^{i-r} - z^{i-r} \\
&= 0,
\end{aligned} \tag{83}$$

which by (82) implies that a_{k_1} is real. Because

$$z^r + z^{-r+\frac{r}{2}} = z^r - z^r = 0, \tag{84}$$

i.e., $a_{\frac{r}{2}}$ is real, the following result is then obtained,

$$a_{\frac{2j+1}{2}r} = a; \qquad 0 \le j < p'. \tag{85}$$

The use of this property can also reduce the computation of J_0'.

12.4 Summary

The use of the data redundancy with 120° symmetry also makes the computation reduced a lot. We see that the algorithms discussed in this chapter are almost the same as the symmetrized algorithms with 90°-rotation introduced in the preceding chapter. But here the computational matrix H_0 for the case of $p \equiv 2 \bmod 3$ and the computational matrix J_0' for the case of $p \equiv 1 \bmod 3$ are not real, but also keep the conjugate property. In addition, the elements of the matrices H_0 and J_0' here have very interesting property that we have proved in (43) and (85). The conclusion is that, by the use of $2^k r$-algorithm or the half-cyclic convolution algorithm, the computation can be further reduced.

CHAPTER 13

THE IMPLEMENTATIONS OF THE ALGORITHMS

13.1 Introduction

To implement an algorithm on a given machine seems to be a simple work. But, there are also many problems involved from the representation of an algorithm to its implementation. For example, for a given algorithm, we have to decide what kind of machine is better for its implementation; on the other hand, if the machine is fixed, we may have to chose an algorithm which may have the best performance on the machine if the algorithm is not unique. An good algorithm for sequential execution may have no parallel feature and a parallel algorithm may be not good in sequential. Certainly, the architecture and some parameters of machine will affect the performance of the implementation of an algorithm. In such sense, we can say that the performance of an algorithm is machine dependant.

In the previous chapters, we used the arithmetic count to evaluate the performance of an algorithm. Sometimes, a parameter R is used,

$$E = A + RM, \tag{1}$$

where A and M are the numbers of additions and multiplications required by the algorithm, and R is the ratio between the operation speeds of multiplication and addition on a given machine. The value of R can be obtained either from the manual of the machine or by testing. The values of R for a few machines are listed below,

$VAX/780$	$MICRO-VAXII$	$CRAY-XMP$	$ASPEN$
3.5	2.3	1.6	1.0

(2)

For example, the Winograd 7-point cyclic convolution algorithm uses 16 multiplications and 70 additions, while our Winograd-like 7-point cyclic convolution algorithm uses 22 multiplications and 54 additions. On VAX/780, the Winograd algorithm is better, but on the rest of machines the new algorithm is better.

For parallel machine, the architectures are too complicated to describe by some formulas. In addition to the parameter R, the degree of

the parallelism and the type of the parallel processing such as SIMD, MIMD and etc., will also affect the implementation.

The measurement by arithmetic count based on addition, in fact, corresponds to the programming in high level language. If we use assembly language, the performance can be further measured by including the times of the memory accessing. The size of cache memory and the number of general registers would be the two main parameters which affect the memory accessing. A programmer can deal with neither the physical segmentation nor the use of registers in high level language. In assembly, some efforts can be made to decrease the number of memory accessing.

To obtain the arithmetic count for an algorithm, sometimes, we have to find the minimum number of additions for a given matrix such as pre or post addition matrices. A procedure of doing this will be briefly described in the later section. But another procedure that tries to figure out the minimum number of memory accessing for assembly language program may not be introduced in detail because it is too complicated. In addition, some programming techniques are very helpful for our implementations, we will mention a few of them. Finally, as results, a new library for small sizes 1-dimensional has been established, and the similar libraries for non-cyclic and cyclic convolutions, 2-dimensional and symmetrized FFTs will be considered. The timing results and some conclusions on MICRO-VAXII and IBM 3090 will be reported.

13.2 A Procedure of Finding The Minimum Number of Additions For Uni-Valued Matrix

For a given matrix, even an Uni-valued matrix, i.e., all entries are either 0 and ± 1, to find the minimum number of additions required, if by hands, we can not guarantee that the result is minimum even if the size is not too large. For example, the following matrix

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

may be considered to use 9 additions. In fact, the minimum number is 8, not 9. Denote by a_i and x_i the input and output respectively. First

compute

$$a_0 = x_1 + x_2 + x_3 + x_4,$$

co
and then set

$$a = a_0 + x_4.$$

an
Then, the rest 4 outputs can be obtained by
$a = a_0 + x_4.$

Then, the rest 4 outputs can be obtained by

$$a_i = a - x_i.$$

There is no general algorithm that can find the minimum number of additions for a given matrix, even if the matrix is Uni-valued. Instead, searching techniques can be used for this purpose. Because most of the pre and post addition matrices in our algorithms are Uni-valued, we only consider this case.

Using the above notation for input and output, if the size of the computational matrix is $m \times n$, then a_i can be defined as a vector

$$a_i = \{b_j x_j : b_j \in 0, \pm 1; 0 \leq j < n\}; \qquad 0 \leq j < m. \tag{3}$$

and then, the computation matrix can be defined as a set of such vectors,

$$A = \{a_i : \qquad 0 \leq i < m\}, \tag{4}$$

where n is the size of A. An upper bound of generating a_i can be set by

$$L(a_i) = O(a_i) - 1, \tag{5}$$

where $O(a_i)$ is the number of non-zero elements in a_i. And

$$L(A) = \sum_{i=0}^{n-1} L(a_i). \tag{6}$$

Obviously, for $L(a_i) = 0$, which implies that a_i contains only one non-zero element, we can delete it from A. Denote by $M(A)$ the minimum number of steps to compute A. Then

$$M(A) = M(A - B), \tag{7}$$

where

$$B = \{a_k : L(a_k) = 0\}. \tag{8}$$

If an element $\pm x_i$ is only contained in a vector a_k, then set

$$a'_k = a_k \mp x_i, \tag{9}$$

and we have

$$M(A) = M(A \cup \{a'_k\} - \{a_k\}) + 1; \qquad L(a_k) > 0. \tag{10}$$

After repeating the above two procedures, all the elements will have that $L(a_i) > 0$. Notice that the set A has been changed. For convenience, we keep this notation. If two elements,

$$a_i = \pm a_j, \tag{11}$$

we call them 2-linearly dependant. Then one of them can be deleted, which implies that

$$M(A) = M(A - \{a_i\}). \tag{12}$$

Now partition A into A_0 and A_1, where

$$A_0 = \{a_i : L(a_i) = 1\} \tag{13}$$

is called indecomposable matrix. Here we mean that it is not necessary to decompose a_i, because

$$M(\{a_i\}) = L(a_i) = 1. \tag{14}$$

If $A_1 = \emptyset$, the empty set, then we have

$$M(A) = M(A_0) = \sum L(a_j) = O(A_0); \qquad a_j \in A_0, \tag{15}$$

where $O(A_0)$ is the number of elements in A_0, because $L(a_j) = 1$ for all $a_j \in A_0$.

If A_1 is not empty, then we can pick up any element a_i from A_1 and decompose it by setting

$$a_i = a'_j + a'_k, \tag{16}$$

and

$$A' = A \cup \{a'_j, a'_k\} - \{a_i\}. \tag{17}$$

Notice that a'_j and a'_k are not unique, we have to try all the possibilities. Denote by $M_i(A)$ the minimum number of steps to compute A with first picking up a_i, we have that

$$M_i(A) = MIN\{M(A')\} + 1, \tag{18}$$

where MIN means the function Minimum. However, if a_i with other two elements in A satisfy

$$a_i = \pm a_j \pm a_k; \qquad a_i \in A_1; \quad a_j, a_k \in A, \tag{19}$$

which we call 3-linearly dependant, then by (17),

$$A' = A - \{a_i\}, \tag{20}$$

which is the minimum set for all possibilities. Then we have

$$M_i(A) = M(A'). \tag{21}$$

Therefore, by checking the 3-linearly dependant property before decomposition the searching procedure can be reduced.

Repeating all the above procedures on A', $M_i(A)$ can be obtained by (18) for each i. Finally we have

$$M(A) = MIN\{M_i(A)\}. \tag{22}$$

The searching is implemented by recursive procedure with backtracking and is written in structural PASCAL. Because the convergence of the procedure is very slow, some other techniques are also used, such as establishing evaluation functions for A' in different stages, modifying the constraint bound $L(A)$ during the procedure and the efforts to avoid unnecessary repeating. The procedure will give both the minimum number and the corresponding computation steps in pseudo-language. We have applied this procedure in building the small libraries and in some other implementations. For example, the 9-point Winograd FFT algorithm by Richard E. Blahut in the Appendix B of [5] uses 44 additions, but by using our program, a result with 36 additions was found, which is the minimum.

The disadvantage of this program is too slow for large size. It will take hours even days for the size over 16. But in our new algorithms,

the pre or post addition matrices with large size are usually represented by tensor products of small matrices. For example,

$$C_n = C_s \otimes C_r, \tag{23}$$

where $n = s \cdot r$. If C_s is $s \times m_s$ matrix and C_r is $r \times m_r$ matrix, then as we stated in chapter 8,

$$M(C_n) = MIN\{r \cdot M(C_s) + m_s \cdot M(C_r), s \cdot M(C_r) + m_r \cdot M(C_s)\}, \tag{24}$$

which implies that $M(C_n)$ can be obtained by applying the procedure on the small size matrices C_s and C_r.

13.3 Some Other Programming Efforts

(i). The Implementations of the permutations

In high level language, One of the difficulties to implement these algorithms could be the permutation if there is. Some permutations are very simple such as Stride permutation. In many cases, the permutations have to be implemented by certain functions, which need some computations including multiplication, addition and in particular, Modulo function. Sometimes the IF structure is also needed. In fact, these computations should not be ignored from the measurement if they are really time wasted. To reduce these computations, Two polices have been used here, 1). replacing higher level computation such as multiplication and Modulo operation by lower level operation such as addition and IF operation; 2). replacing mathematical operation by corresponding addressing mode. A basic technique to do so is the trade-off between the space and time. For example, to implement the indices z^i, where z is a generator for some prime p, an array $Z(i)$ can be defined as we did in many examples before. Again, if the indexing function is $z^i \cdot z^j$, a modulo operation on $(i + j, p)$ is needed. In this case, we may define $Z(i)$ with double length. For the permutation in parallel algorithms, the policies would be a little different. Recall that we did for the permutation $T_s^{-1}P(n, s)$ in the section 2. of chapter 7, We may prefer that the permutation has parallel feature too.

(ii). The Efforts on Minimizing The Number of Memory Accessing

In assembly, two basic efforts can be made to reduce the number of memory accessing. One is to segment program physically as identical to the logical segmentation as possible, so that the page faults can be reduced. The cache memory size of the machine is a relevant parameter for this purpose. The other effort is to well-organize the computation sequence so that the memory accessing can be reduced. However the minimizations for both items are very complicated. We have made a procedure to minimize the number of memory accessing for the second purpose.

For a given computation structure, the implementation sequence is not unique. To find a computation sequence which uses the minimum number of memory accessing, the searching techniques has to be used again. The relevant parameter is the number of the general-purpose registers in the machine. Because the procedure is more complicated and less significant than the above procedure, we are not going to give it further discussion here. By applying this procedure on some samples, more than 30% of memory accessing can be saved, which may result in about 10% improvement of performance than the corresponding programs written in FORTRAN. The procedure may take time. too.

(iii). A Special Method to Multiply a Power of 2

Another advantage of using assembly is that we may utilize some special characters of the machine to reduce the computation. A typical example is the multiplication with the powers of 2. In general, a floating-point number a is represented by two parts,

$$a = a_m * 2^{a_e}; \qquad 0 \leq a_m < 1; \tag{25}$$

where a_m is called mantissa and a_e is exponent. Then

$$a * 2^i = a_m * 2^{a_e+i}, \tag{26}$$

where i is an integer. The floating multiplication now is changed into integer addition. To do this we have to know how the data is represented in binary on the given machine. On VAX/780 and MICRO VAXII this integer addition can be carried out only on the first two bytes of the floating data, which is about 15 times fast than the floating multiplication. Further, the floating-point multiplication

$$a * (2^i \pm 1) = a * 2^i \pm a \tag{27}$$

can be carried out by one 2-byte integer addition and one floating-point addition, which may be also faster than one floating-point multiplication on some machine.

In our algorithms, many multiplications consist of constants 2^k or $2^k \pm 1$. For example, one half multiplications required by 3-point 1-dimensional FFT and 2-dimensional FFT by new algorithm contain $-\frac{1}{2}$. Using this approach about 20% of improvement can be obtained.

13.4 The Libraries and Some Sample Programs

By combining the new algorithms with the existing algorithms, some libraries can be established on any machine. We are going to implement a few of them on both MICRO VAX II and IBM 3090. Because this work is beyond the scope of this dissertation, we will only give a brief introduction for these implementations.

(i). The Library For Half-Cyclic Convolution

This library will use the negative algorithm. Only the points which are necessary to build the library for cyclic convolution are sampled.

(ii). The Library For Cyclic Convolution

This library will contain the two types algorithms, the real-type and the complex-type algorithms for both real input and complex input. The samples in the library will cover the need by the 1-dimensional FFT library.

(iii). The Library For 1-Dimensional FFT

Using the new algorithms for cyclic convolution and some implementation techniques in the above, a new library for 1-dimensional FFT is going to be established. This would be one of the important results of the whole work. The range is from 2-point to 61-point. First, the prime case algorithms will be written, which then can be used to build up the multiplicative algorithm.

(iv). Some Other Programs

In addition to the libraries in the above, some other sample programs for 2-dimensional FFT and symmetrized algorithms will also be written for sequential execution.

All of the above projects are seperated from this dissertation and the results will be reported in the later papers.

REFERENCES

[1] S. Winograd, "*On Computing the Discrete Fourier Transform*", *Math Comp., vol. 32, num. 141, pp175-199*, 1978

[2] S. Winograd, "*On Computing the Discrete Fourier Transform*", *Proc. Nat. Acad. Sci. USA., vol.73, no.4, pp1005-1006*, 1976

[3] M. T. Heideman, D. H. Johnson and C. S. Burrus, "*Gauss and the History of the Fast Fourier Transform*", *IEEE ASSP*, October, 1984

[4] W. T. Cochran, et al., "*What is the Fast Fourier Transform?*" *IEEE Trans. Aud. and Elec., vol.15, num.2, pp45-55*, 1967

[5] R. Blahut, "*Fast Algorithms for Digital Signal Processing*", *Addison Wersley Publishing Company*, Reading, MA, 1985

[6] R. Tolimieri, M. An and C. Lu, "*Algorithms for Discrete Fourier Transform and Convolution*", *Springer-Verlag Publishing Company*, 1989

[7] J. Cooley and J. Tukey, "*An Algorithm for the Machine Calculation of the Complex Fourier Series*", *Math. Comp., vol.19, pp297-301*, 1965

[8] H. Johnson and S. Burrus, "*The Design of DFT Algorithms*", *PH.D thesis, Department of Electrical and Computer Engineering, Rice University*, 1982

[9] C. Lu, "*Fast Fourier Transform Algorithms for Special N's and the Implementation on VAX*", *Ph.D thesis, E.E. Department, City College of the City University of New York*, 1988

[10] H. Nussbaumer, "*Fast Polynomial Transform Algorithms for Digital Convolution*", *IEEE ASSP vol.28, num.2, pp205-215*, 1980

[11] P. Davis, "*Circulant Matrices*", *John Wiley & Sons*, 1979

[12] G. Birkoff and S. Maclane, "*A Survey of Modern Algebra*", *Macmillam Publishing Company*, 1953

[13] C. Rader, "*Discrete Fourier Transforms When the Number of Data Samples is Prime*", *IEEE Proc., vol.56, pp1107-1108*, 1968

[14] R. Agarwal and J. Cooley, "*New Algorithms for Digital Convolution*", *IEEE ASSP, vol.25, num.5, pp392-410*, 1975

[15] T. Stockham, "*High Speed Convolution and Correlation*", *Spring Joint Comput. Conf., AFIPS Conf. Proc., 28, pp229-233*, 1966

[16] C. Rader, "*Discrete Convolution via Mersenne Transform,*" *IEEE Trans. on Comp., vol.21, num.12, pp1269-1273,* 1972

[17] R. Agarwal and C. Burrus, "*Number Theoretic Transforms to Implement Fast Digital Convolution*", *IEEE ASSP-75, vol.63, num.4, pp550-560,* 1975

[18] S. Lee and H. Lu, "*Fast Convolution Using Generalized Format/ Mersenne Number Transforms*", *IEEE ICASSP 88, pp1910-1913,* 1988

[19] L. Auslander and A. Silberger, "*On the Use of Scratchpad in the Construction of Convolution Algorithms*", *Research Rpt. RC10554, IBM Watson Research Center,* 1984

[20] L. Auslander, J. W. Cooley and A. J. Silberger, "*Number Stability of Fast Convolution Algorithms for Digital Filtering*", *VLSI Signal Processing, IEEE Press., pp172-213,* 1984

[21] M. Heideman, "*Applications of Multiplicative Complexity Theory to Convolution and Discrete Fourier Transform*", *Ph.D thesis, Department of Electrical and Computer Engineering, Rice University,* 1986

[22] D. Rodriguez, "*On Tensor Product Formulations of Additive Fast Fourier Transform Algorithms and Their Implementations*", *Ph.D thesis, E.E. Department, City College of the City University of New York,* 1987

[23] S. Winograd, "*Arithmetic Complexity of Computations*", *CBMS-NSF Conference Series in Appl. Math., SIAM 33,* 1980

[24] I. J. Good, *"The Interaction Algorithm and Practical Fourier Analysis", J. Royal Statist. Soc. Ser. B 20, addendum 22, pp372-375,* 1960

[25] L. H. Thomas, *"Using a Computer to Solve Problems in Physics", App. of Digital Computers, Gin and Co.,* Boston, MA, 1963

[26] C. Temperton, "*A Note on Prime Factor FFT Algorithms*', *Jour. of Comp. Phy., vol.52, pp198-204,* 1983

[27] D. Kolba and T. Parks, "*A Prime Factor FFT Algorithm Using High Speed Convolution*", *IEEE ASSP, vol.25, num.4, pp281-294,* 1977

[28] L. Auslander, E. Feig and S. Winograd, "*New Algorithm for the Multidimensional Discrete Fourier Transform*", *IEEE ASSP-31, pp 388-403*, 1983

[29] Myoung An and L. Auslander, "*Fourier Transforms That Respect Crystallographic Symmetries*", *IBM Journal of Research and Development, 2(31):pp213-223*, 1987

[30] L. F. Ten Eyck, "*Crystallographic Fast Fourier Transform*", *Acta Cryst., A29:pp183-199*, 1973

[31] R. Tolimieri and G. Bricogne, "*Symmetrized FFT Algorithms*", *IMA Proc. on Signal Processing, Springer-Verlag*, 1990

[32] Hui Cheng, "*Vector Pipelining, Chaining, and Speed on the IBM 3090 and Cray X-MP*", *Computer, pp31-46*, September 1989