Automatic Generation of Prime Length FFT Programs

Collection Editor:
C. Sidney Burrus

Automatic Generation of Prime Length FFT Programs

Collection Editor:
C. Sidney Burrus

Authors:
C. Sidney Burrus
Ivan Selesnick

Online:
< http://cnx.org/content/col10596/1.4/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: September 9, 2009
PDF generated: October 26, 2012
For copyright and attribution information for the modules contained in this collection, see p. 75.

Table of Contents

1 Introduction 1
2 Preliminaries 3
3 Bilinear Forms for Circular Convolution 13
4 A Bilinear Form for the DFT 19
5 Implementing Kronecker Products Efficiently 23
6 Programs for Circular Convolution 29
7 Programs for Prime Length FFTs 33
8 Conclusion 37
9 Appendix: Bilinear Forms for Linear Convolution 39
10 Appendix: A 45 Point Circular Convolution Program 45
11 Appendix: A 31 Point FFT Program 47
12 Appendix: Matlab Functions For Circular Convolution and Prime Length FFTs 53
13 Appendix: A Matlab Program for Generating Prime Length FFT Programs 57
Bibliography 70
Index 74
Attributions 75

Chapter 1

Introduction[1]

1.1 Introduction

The development of algorithms for the fast computation of the Discrete Fourier Transform in the last 30 years originated with the radix 2 Cooley-Tukey FFT and the theory and variety of FFTs has grown significantly since then. Most of the work has focused on FFTs whose sizes are composite, for the algorithms depend on the ability to factor the length of the data sequence, so that the transform can be found by taking the transform of smaller lengths. For this reason, algorithms for prime length transforms are building blocks for many composite length FFTs - the maximum length and the variety of lengths of a PFA or WFTA algorithm depend upon the availability of prime length FFT modules. As such, prime length Fast Fourier Transforms are a special, important and difficult case.

Fast algorithms designed for specific short prime lengths have been developed and have been written as straight line code [9], [13]. These dedicated programs rely upon an observation made in Rader's paper [24] in which he shows that a prime p length DFT can be found by performing a $p-1$ length circular convolution. Since the publication of that paper, Winograd had developed a theory of multiplicative complexity for transforms and designed algorithms for convolution that attain the minimum number of multiplications [38]. Although Winograd's algorithms are very efficient for small prime lengths, for longer lengths they require a large number of additions and the algorithms become very cumbersome to design. This has prevented the design of useful prime length FFT programs for lengths greater than 31. Although we have previously reported the design of programs for prime lengths greater than 31 [27] those programs required more additions than necessary and were long. Like the previously existing ones, they simply consisted of a long list of instructions and did not take advantage of the attainable common structures.

In this paper we describe a set of programs for circular convolution and prime length FFTs that are are short, possess great structure, share many computational procedures, and cover a large variety of lengths. Because the underlying convolution is decomposed into a set of disjoint operations they can be performed in parallel and this parallelism is made clear in the programs. Moreover, each of these independent operations is made up of a sequence of sub-operations of the form $I \otimes A \otimes I$ where $\otimes$ denotes the Kronecker product. These operations can be implemented as vector/parallel operations [34]. Previous programs for prime length FFTs do not have these features: they consist of straight line code and are not amenable to vector/parallel implementations.

We have also developed a program that automatically generates these programs for circular convolution and prime length DFTs. This code generating program requires information only about a set of modules for computing cyclotomic convolutions. We compute these non-circular convolutions by computing a linear convolution and reducing the result. Furthermore, because these linear convolution algorithms can be built from smaller ones, the only modules needed are ones for the linear convolution of prime length sequences. It turns out that with linear convolution algorithms for only the lengths 2 and 3, we can generate a wide

[1] This content is available online at <http://cnx.org/content/m18131/1.5/>.

variety of prime length FFT algorithms. In addition, the code we generate is made up of calls to a relatively small set of functions. Accordingly, the subroutines can be designed and optimized to specifically suit a given architecture.

The programs we describe use Rader's conversion of a prime point DFT into a circular convolution, but this convolution we compute using the split nesting algorithm [20]. As Stasinski notes [31], this yields algorithms possessing greater structure and simpler programs and doesn't generally require more computation.

1.1.1 On the Row-Column Method

In computing the DFT of an $n = n_1 n_2$ point sequence where n_1 and n_2 are relatively prime, a row-column method can be employed. That is, if an $n_1 \times n_2$ array is appropriately formed from the n point sequence, then its DFT can be computed by computing the DFT of the rows and by then computing the DFT of the columns. The separability of the DFT makes this possible. It should be mentioned, however, that in at least two papers [31], [15] it is mistakenly assumed that the row-column method can also be applied to convolution. Unfortunately, the convolution of two sequences can not be found by forming two arrays, by convolving their rows, and by then convolving their columns. This misunderstanding about the separability of convolution also appears in [3] where the author incorrectly writes a diagonal matrix of a bilinear form as a Kronecker product. If it were a Kronecker product, then there would indeed exist a row-column method for convolution.

Earlier reports on this work were published in the conference proceedings [27], [28], [29] and a fairly complete report was published in the IEEE Transaction on Signal Processing [30]. Some parts of this approach appear in the Connexions book, Fast Fourier Transforms[2]. This work is built on and an extension of that in [29] which is also in the Connexions Technical Report[3].

[2] *Fast Fourier Transforms* <http://cnx.org/content/col10550/latest//latest/>

[3] *Large DFT Modules: 11, 13, 16, 17, 19, and 25. Revised ECE Technical Report 8105* <http://cnx.org/content/col10569/latest//latest/>

Chapter 2

Preliminaries[1]

2.1 Preliminaries

Because we compute prime point DFTs by converting them in to circular convolutions, most of this and the next section is devoted to an explanation of the split nesting convolution algorithm. In this section we introduce the various operations needed to carry out the split nesting algorithm. In particular, we describe the prime factor permutation that is used to convert a one-dimensional circular convolution into a multi-dimensional one. We also discuss the reduction operations needed when the Chinese Remainder Theorem for polynomials is used in the computation of convolution. The reduction operations needed for the split nesting algorithm are particularly well organized. We give an explicit matrix description of the reduction operations and give a program that implements the action of these reduction operations.

The presentation relies upon the notions of similarity transformations, companion matrices and Kronecker products. With them, we describe the split nesting algorithm in a manner that brings out its structure. We find that when companion matrices are used to describe convolution, the reduction operations block diagonalizes the circular shift matrix.

The companion matrix of a monic polynomial, $M(s) = m_0 + m_1 s + \cdots + m_{n-1}s^{n-1} + s^n$ is given by

$$C_M = \begin{bmatrix} & & -m_0 1 \\ 1 & & -m_1 \\ & \ddots & \vdots \\ & 1 & -m_{n-1} \end{bmatrix}. \tag{2.1}$$

Its usefulness in the following discussion comes from the following relation which permits a matrix formulation of convolution. Let

$$\begin{aligned} X(s) &= x_0 + x_1 s + \cdots x_{n-1}s^{n-1} \\ H(s) &= h_0 + h_1 s + \cdots h_{n-1}s^{n-1} \\ Y(s) &= y_0 + y_1 s + \cdots y_{n-1}s^{n-1} \\ M(s) &= m_0 + m_1 s + \cdots m_{n-1}s^{n-1} + s^n \end{aligned} \tag{2.2}$$

Then

$$Y(s) = < H(s) X(s) >_{M(s)} \quad \Leftrightarrow \quad y = \left(\sum_{k=0}^{n-1} h_k C_M^k \right) x \tag{2.3}$$

[1] This content is available online at <http://cnx.org/content/m18132/1.5/>.

where $y = (y_0, \cdots, y_{n-1})^t$, $x = (x_0, \cdots, x_{n-1})^t$, and C_M is the companion matrix of $M(s)$. In (2.3), we say y is the convolution of x and h with respect to $M(s)$. In the case of circular convolution, $M(s) = s^n - 1$ and C_{s^n-1} is the circular shift matrix denoted by S_n,

$$S_n = \begin{bmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix} \tag{2.4}$$

Notice that any circulant matrix can be written as $\sum_k h_k S_n^k$.

Similarity transformations can be used to interpret the action of some convolution algorithms. If $C_M = T^{-1}AT$ for some matrix T (C_M and A are similar, denoted $C_M \sim A$), then (2.3) becomes

$$y = T^{-1}\left(\sum_{k=0}^{n-1} h_k A^k\right) Tx. \tag{2.5}$$

That is, by employing the similarity transformation given by T in this way, the action of S_n^k is replaced by that of A^k. Many circular convolution algorithms can be understood, in part, by understanding the manipulations made to S_n and the resulting new matrix A. If the transformation T is to be useful, it must satisfy two requirements: (1) Tx must be simple to compute, and (2) A must have some advantageous structure. For example, by the convolution property of the DFT, the DFT matrix F diagonalizes S_n,

$$S_n = F^{-1}\begin{bmatrix} w^0 & & & \\ & w^1 & & \\ & & \ddots & \\ & & & w^{n-1} \end{bmatrix} F \tag{2.6}$$

so that it diagonalizes every circulant matrix. In this case, Tx can be computed by using an FFT and the structure of A is the simplest possible. So the two above mentioned conditions are met.

The Winograd Structure can be described in this manner also. Suppose $M(s)$ can be factored as $M(s) = M_1(s) M_2(s)$ where M_1 and M_2 have no common roots, then $C_M \sim (C_{M_1} \oplus C_{M_2})$ where $\oplus$ denotes the matrix direct sum. Using this similarity and recalling (2.3), the original convolution is decomposed into disjoint convolutions. This is, in fact, a statement of the Chinese Remainder Theorem for polynomials expressed in matrix notation. In the case of circular convolution, $s^n - 1 = \prod_{d|n} \Phi_d(s)$, so that S_n can be transformed to a block diagonal matrix,

$$S_n \sim \begin{bmatrix} C_{\Phi_1} & & & \\ & C_{\Phi_d} & & \\ & & \ddots & \\ & & & C_{\Phi_n} \end{bmatrix} = \left(\bigoplus_{d|n} C_{\Phi_d}\right) \tag{2.7}$$

where $\Phi_d(s)$ is the d^{th} cyclotomic polynomial. In this case, each block represents a convolution with respect to a cyclotomic polynomial, or a 'cyclotomic convolution'. Winograd's approach carries out these cyclotomic convolutions using the Toom-Cook algorithm. Note that for each divisor, d, of n there is a corresponding block on the diagonal of size $\phi(d)$, for the degree of $\Phi_d(s)$ is $\phi(d)$ where $\phi(\cdot)$ is the Euler totient function. This method is good for short lengths, but as n increases the cyclotomic convolutions become cumbersome, for as the number of distinct prime divisors of d increases, the operation described by $\sum_k h_k (C_{\Phi_d})^k$ becomes more difficult to implement.

The Agarwal-Cooley Algorithm utilizes the fact that

$$S_n = P^t \left(S_{n_1} \otimes S_{n_2} \right) P \tag{2.8}$$

where $n = n_1 n_2$, $(n_1, n_2) = 1$ and P is an appropriate permutation [1]. This converts the one dimensional circular convolution of length n to a two dimensional one of length n_1 along one dimension and length n_2 along the second. Then an n_1-point and an n_2-point circular convolution algorithm can be combined to obtain an n-point algorithm. In polynomial notation, the mapping accomplished by this permutation P can be informally indicated by

$$Y(s) = \; < X(s) H(s) >_{s^n - 1} \overset{P}{\Leftrightarrow} Y(s,t) = \; < X(s,t) H(s,t) >_{s^{n_1}-1, t^{n_2}-1}. \tag{2.9}$$

It should be noted that (2.8) implies that a circulant matrix of size $n_1 n_2$ can be written as a block circulant matrix with circulant blocks.

The Split-Nesting algorithm [21] combines the structures of the Winograd and Agarwal-Cooley methods, so that S_n is transformed to a block diagonal matrix as in (2.7),

$$S_n \sim \bigoplus_{d|n} \Psi(d). \tag{2.10}$$

Here $\Psi(d) = \otimes_{p|d, p \in \mathcal{P}} C_{\Phi_{H_d(p)}}$ where $H_d(p)$ is the highest power of p dividing d, and $\mathcal{P}$ is the set of primes.

Example 2.1

$$S_{45} \sim \begin{bmatrix} 1 & & & & & & & \\ & C_{\Phi_3} & & & & & & \\ & & C_{\Phi_9} & & & & & \\ & & & C_{\Phi_5} & & & & \\ & & & & C_{\Phi_3} \otimes C_{\Phi_5} & & & \\ & & & & & & & \\ & & & & & & C_{\Phi_9} \otimes C_{\Phi_5} & \end{bmatrix} \tag{2.11}$$

In this structure a multidimensional cyclotomic convolution, represented by $\Psi(d)$, replaces each cyclotomic convolution in Winograd's algorithm (represented by C_{Φ_d} in (2.7). Indeed, if the product of $b_1, \cdots, b_k$ is d and they are pairwise relatively prime, then $C_{\Phi_d} \sim C_{\Phi_{b_1}} \otimes \cdots \otimes C_{\Phi_{b_k}}$. This gives a method for combining cyclotomic convolutions to compute a longer circular convolution. It is like the Agarwal-Cooley method but requires fewer additions [21].

2.2 Prime Factor Permutations

One can obtain $S_{n_1} \otimes S_{n_2}$ from $S_{n_1 n_2}$ when $(n_1, n_2) = 1$, for in this case, S_n is similar to $S_{n_1} \otimes S_{n_2}$, $n = n_1 n_2$. Moreover, they are related by a permutation. This permutation is that of the prime factor FFT algorithms and is employed in nesting algorithms for circular convolution [1], [18]. The permutation is described by Zalcstein [40], among others, and it is his description we draw on in the following.

Let $n = n_1 n_2$ where $(n_1, n_2) = 1$. Define e_k, $(0 \leq k \leq n-1)$, to be the standard basis vector, $(0, \cdots, 0, 1, 0, \cdots, 0)^t$, where the 1 is in the k^{th} position. Then, the circular shift matrix, S_n, can be described by

$$S_n e_k = e_{\; < k+1 >_n}. \tag{2.12}$$

Note that, by inspection,

$$\left(S_{n_2} \otimes S_{n_1} \right) e_{a + n_1 b} = e_{\; < a+1 >_{n_1} + n_1 \; < b+1 >_{n_2}} \tag{2.13}$$

where $0 \leq a \leq n_1 - 1$ and $0 \leq b \leq n_2 - 1$. Because n_1 and n_2 are relatively prime a permutation matrix P can be defined by

$$Pe_k = e_{< k>_{n_1} + n_1 < k>_{n_2}}. \tag{2.14}$$

With this P,

$$\begin{aligned} PS_n e_k &= Pe_{< k+1>_n} \\ &= e_{< < k+1>_n >_{n_1} + n_1 < < k+1>_n >_{n_2}} \\ &= e_{< k+1>_{n_1} + n_1 < k+1>_{n_2}} \end{aligned} \tag{2.15}$$

and

$$\begin{aligned} (S_{n_2} \otimes S_{n_1})\, Pe_k &= (S_{n_2} \otimes S_{n_1})\, e_{< k>_{n_1} + n_1 < k>_{n_2}} \\ &= e_{< k+1>_{n_1} + n_1 < k+1>_{n_2}}. \end{aligned} \tag{2.16}$$

Since $PS_n e_k = (S_{n_2} \otimes S_{n_1})\, Pe_k$ and $P^{-1} = P^t$, one gets, in the multi-factor case, the following.

Lemma 2.1:
If $n = n_1 \cdots n_k$ and $n_1, ..., n_k$ are pairwise relatively prime, then $S_n = P^t (S_{n_k} \otimes \cdots \otimes S_{n_1}) P$ where P is the permutation matrix given by $Pe_k = e_{< k>_{n_1} + n_1 < k>_{n_2} + \cdots + n_1 \cdots n_{k-1} < k>_{n_k}}$.

This useful permutation will be denoted here as $P_{n_k, \cdots, n_1}$. If $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ then this permutation yields the matrix, $S_{p_1^{e_1}} \otimes \cdots \otimes S_{p_k^{e_k}}$. This product can be written simply as $\bigotimes_{i=1}^{k} S_{p_i^{e_i}}$, so that one has $S_n = P^t_{n_1, \cdots, n_k} \left(\bigotimes_{i=1}^{k} S_{p_i^{e_i}} \right) P_{n_1, \cdots, n_k}$.

It is quite simple to show that

$$P_{a,b,c} = (I_a \otimes P_{b,c})\, P_{a,bc} = (P_{a,b} \otimes I_c)\, P_{ab,c}. \tag{2.17}$$

In general, one has

$$P_{n_1, \cdots, n_k} = \prod_{i=2}^{k} \left(P_{n_1 \cdots n_{i-1}, n_i} \otimes I_{n_{i+1} \cdots n_k} \right). \tag{2.18}$$

A Matlab function for $P_{a,b} \otimes I_s$ is `pfp2I()` in one of the appendices. This program is a direct implementation of the definition. In a paper by Templeton [32], another method for implementing $P_{a,b}$, without 'if' statements, is given. That method requires some precalculations, however. A function for $P_{n_1, \cdots, n_k}$ is `pfp()`. It uses (2.18) and calls `pfp2I()` with the appropriate arguments.

2.3 Reduction Operations

The Chinese Remainder Theorem for polynomials can be used to decompose a convolution of two sequences (the polynomial product of two polynomials evaluated modulo a third polynomial) into smaller convolutions (smaller polynomial products) [39]. The Winograd n point circular convolution algorithm requires that polynomials are reduced modulo the cyclotomic polynomial factors of $s^n - 1$, $\Phi_d(s)$ for each d dividing n.

When n has several prime divisors the reduction operations become quite complicated and writing a program to implement them is difficult. However, when n is a prime power, the reduction operations are very structured and can be done in a straightforward manner. Therefore, by converting a one-dimensional convolution to a multi-dimensional one, in which the length is a prime power along each dimension, the split nesting algorithm avoids the need for complicated reductions operations. This is one advantage the split nesting algorithm has over the Winograd algorithm.

By applying the reduction operations appropriately to the circular shift matrix, we are able to obtain a block diagonal form, just as in the Winograd convolution algorithm. However, in the split nesting algorithm, each diagonal block represents multi-dimensional cyclotomic convolution rather than a one-dimensional one. By forming multi-dimensional convolutions out of one-dimensional ones, it is possible to combine algorithms for smaller convolutions (see the next section). This is a second advantage split nesting algorithm has over the Winograd algorithm. The split nesting algorithm, however, generally uses more than the minimum number of multiplications.

Below we give an explicit matrix description of the required reduction operations, give a program that implements them, and give a formula for the number of additions required. (No multiplications are needed.)

First, consider $n = p$, a prime. Let

$$X(s) = x_0 + x_1 s + \cdots + x_{p-1} s^{p-1} \tag{2.19}$$

and recall $s^p - 1 = (s-1)\left(s^{p-1} + s^{p-2} + \cdots + s + 1\right) = \Phi_1(s)\,\Phi_p(s)$. The residue $< X(s) >_{\Phi_1(s)}$ is found by summing the coefficients of $X(s)$. The residue $< X(s) >_{\Phi_p(s)}$ is given by $\sum_{k=0}^{p-2} (x_k - x_{p-1})\, s^k$. Define R_p to be the matrix that reduces $X(s)$ modulo $\Phi_1(s)$ and $\Phi_p(s)$, such that if $X_0(s) = < X(s) >_{\Phi_1(s)}$ and $X_1(s) = < X(s) >_{\Phi_p(s)}$ then

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = R_p X \tag{2.20}$$

where X, X_0 and X_1 are vectors formed from the coefficients of $X(s)$, $X_0(s)$ and $X_1(s)$. That is,

$$R_p = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & & & & -1 \\ & 1 & & & -1 \\ & & 1 & & -1 \\ & & & 1 & -1 \end{bmatrix} \tag{2.21}$$

so that $R_p = \begin{bmatrix} 1_{-1} \\ G_p \end{bmatrix}$ where G_p is the $\Phi_p(s)$ reduction matrix of size $(p-1) \times p$. Similarly, let $X(s) = x_0 + x_1 s + \cdots + x_{p^{e-1}} s^{p^{e-1}}$ and define R_{p^e} to be the matrix that reduces $X(s)$ modulo $\Phi_1(s)$, $\Phi_p(s)$, ..., $\Phi_{p^e}(s)$ such that

$$\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_e \end{bmatrix} = R_{p^e} X, \tag{2.22}$$

where as above, X, X_0, ..., X_e are the coefficients of $X(s)$, $< X(s) >_{\Phi_1(s)}$, ..., $< X(s) >_{\Phi_{p^e}(s)}$.

It turns out that R_{p^e} can be written in terms of R_p. Consider the reduction of $X(s) = x_0 + \cdots + x_8 s^8$ by $\Phi_1(s) = s - 1$, $\Phi_3(s) = s^2 + s + 1$, and $\Phi_9(s) = s^6 + s^3 + 1$. This is most efficiently performed by reducing $X(s)$ in two steps. That is, calculate $X'(s) = < X(s) >_{s^3-1}$ and $X_2(s) = < X(s) >_{s^6+s^3+1}$. Then calculate $X_0(s) = < X'(s) >_{s-1}$ and $X_1(s) = < X'(s) >_{s^2+s+1}$. In matrix notation this becomes

$$\begin{bmatrix} X' \\ X_2 \end{bmatrix} = \begin{bmatrix} I_3 & I_3 & I_3 \\ I_3 & & -I_3 \\ & I_3 & -I_3 \end{bmatrix} X \quad \text{and} \quad \begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & & -1 \\ & 1 & -1 \end{bmatrix} X'. \tag{2.23}$$

Together these become

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} R_3 & & \\ & I_3 & \\ & & I_3 \end{bmatrix} \begin{bmatrix} I_3 & I_3 & I_3 \\ I_3 & & -I_3 \\ & I_3 & -I_3 \end{bmatrix} X \tag{2.24}$$

or

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = (R_3 \oplus I_6)(R_3 \otimes I_3) X \tag{2.25}$$

so that $R_9 = (R_3 \oplus I_6)(R_3 \otimes I_3)$ where $\oplus$ denotes the matrix direct sum. Similarly, one finds that $R_{27} = (R_3 \oplus I_{24})((R_3 \otimes I_3) \oplus I_{18})(R_3 \otimes I_9)$. In general, one has the following.

Lemma 2.2:
R_{p^e} is a $p^e \times p^e$ matrix given by $R_{p^e} = \prod_{k=0}^{e-1} \left(\left(R_p \otimes I_{p^k} \right) \oplus I_{p^e - p^{k+1}} \right)$ and can be implemented with $2(p^e - 1)$ additions.

The following formula gives the decomposition of a circular convolution into disjoint non-circular convolutions when the number of points is a prime power.

$$\begin{aligned} R_{p^e} S_{p^e} R_{p^e}^{-1} &= \begin{bmatrix} 1 & & & \\ & C_{\Phi_p} & & \\ & & \ddots & \\ & & & C_{\Phi_{p^e}} \end{bmatrix} \\ &= \bigoplus_{i=0}^{e} C_{\Phi_{p^i}} \end{aligned} \tag{2.26}$$

Example 2.2

$$R_9 S_9 R_9^{-1} = \begin{bmatrix} 1 & & \\ & C_{\Phi_3} & \\ & & C_{\Phi_9} \end{bmatrix} \tag{2.27}$$

It turns out that when n is not a prime power, the reduction of polynomials modulo the cyclotomic polynomial $\Phi_n(s)$ becomes complicated, and with an increasing number of prime factors, the complication increases. Recall, however, that a circular convolution of length $p_1^{e_1} \cdots p_k^{e_k}$ can be converted (by an appropriate permutation) into a k dimensional circular convolution of length $p_i^{e_i}$ along dimension i. By employing this one-dimensional to multi-dimensional mapping technique, one can avoid having to perform polynomial reductions modulo $\Phi_n(s)$ for non-prime-power n.

The prime factor permutation discussed previously is the permutation that converts a one-dimensional circular convolution into a multi-dimensional one. Again, we can use the Kronecker product to represent this. In this case, the reduction operations are applied to each matrix in the following way:

$$T \left(S_{p_1^{e_1}} \otimes \cdots \otimes S_{p_k^{e_k}} \right) T^{-1} = \left(\oplus_{i=0}^{e_1} C_{\Phi_{p_1^i}} \right) \otimes \cdots \otimes \left(\oplus_{i=0}^{e_k} C_{\Phi_{p_k^i}} \right) \tag{2.28}$$

where

$$T = R_{p_1^{e_1}} \otimes \cdots \otimes R_{p_k^{e_k}} \tag{2.29}$$

Example 2.3

$$T\left(S_9 \otimes S_5\right) T^{-1} = \begin{bmatrix} 1 & & \\ & C_{\Phi_3} & \\ & & C_{\Phi_9} \end{bmatrix} \otimes \begin{bmatrix} 1 & \\ & C_{\Phi_5} \end{bmatrix} \tag{2.30}$$

where $T = R_9 \otimes R_5$.

The matrix $R_{p_1^{e_1}} \otimes \cdots \otimes R_{p_k^{e_k}}$ can be factored using a property of the Kronecker product. Notice that $(A \otimes B) = (A \otimes I)(I \otimes B)$, and $(A \otimes B \otimes C) = (A \otimes I)(I \otimes B \otimes I)(I \otimes C)$ (with appropriate dimensions) so that one gets

$$\bigotimes_{i=1}^{k} R_{p_i^{e_i}} = \prod_{i=1}^{k} \left(I_{m_i} \otimes R_{p_i^{e_i}} \otimes I_{n_i} \right), \tag{2.31}$$

where $m_i = \prod_{j=1}^{i-1} p_j^{e_j}$, $n_i = \prod_{j=i+1}^{k} p_j^{e_j}$ and where the empty product is taken to be 1. This factorization shows that T can be implemented basically by implementing copies of R_{p^e}. There are many variations on this factorization as explained in [35]. That the various factorization can be interpreted as vector or parallel implementations is also explained in [35].

Example 2.4

$$R_9 \otimes R_5 = (R_9 \otimes I_5)(I_9 \otimes R_5) \tag{2.32}$$

and

$$R_9 \otimes R_{25} \otimes R_7 = (R_9 \otimes I_{175})(I_9 \otimes R_{25} \otimes I_7)(I_{225} \otimes R_7) \tag{2.33}$$

When this factored form of $\otimes R_{n_i}$ or any of the variations alluded to above, is used, the number of additions incurred is given by

$$\begin{aligned} \sum_{i=1}^{k} \frac{N}{p_i^{e_i}} \mathcal{A}\left(R_{p_i^{e_i}}\right) &= \sum_{i=1}^{k} \frac{N}{p_i^{e_i}} 2\left(p_i^{e_i} - 1\right) \\ &= 2N \sum_{i=1}^{k} 1 - \frac{1}{p_i^{e_i}} \\ &= 2N\left(k - \sum_{i=1}^{k} \frac{1}{p_i^{e_i}}\right) \end{aligned} \tag{2.34}$$

where $N = p_1^{e_1} \cdots p_k^{e_k}$.

Although the use of operations of the form $R_{p_1^{e_1}} \otimes \cdots \otimes R_{p_k^{e_k}}$ is simple, it does not exactly separate the circular convolution into smaller disjoint convolutions. In other words, its use does not give rise in (2.28) and (2.30) to block diagonal matrices whose diagonal blocks are the form $\otimes_i C_{\Phi_{p_i}}$. However, by reorganizing the arrangement of the operations we can obtain the block diagonal form we seek.

First, suppose A, B and C are matrices of sizes $a \times a$, $b \times b$ and $c \times c$ respectively. If

$$TBT^{-1} = \begin{bmatrix} B_1 & \\ & B_2 \end{bmatrix} \tag{2.35}$$

where B_1 and B_2 are matrices of sizes $b_1 \times b_1$ and $b_2 \times b_2$, then

$$Q\left(A \otimes B \otimes C\right) Q^{-1} = \begin{bmatrix} A \otimes B_1 \otimes C & \\ & A \otimes B_2 \otimes C \end{bmatrix} \tag{2.36}$$

where

$$Q = \begin{bmatrix} I_a \otimes T(1:b_1,:) \otimes I_c \\ I_a \otimes T(b_1+1:b,:) \otimes I_c \end{bmatrix}. \tag{2.37}$$

Here $T(1:b_1,:)$ denotes the first b_1 rows and all the columns of T and similarly for $T(b_1+1:b,:)$. Note that

$$\begin{bmatrix} A \otimes B_1 \otimes C & \\ & A \otimes B_2 \otimes C \end{bmatrix} \neq A \otimes \begin{bmatrix} B_1 & \\ & B_2 \end{bmatrix} \otimes C. \tag{2.38}$$

That these two expressions are not equal explains why the arrangement of operations must be reorganized in order to obtain the desired block diagonal form. The appropriate reorganization is described by the expression in (2.37). Therefore, we must modify the transformation of (2.28) appropriately. It should be noted that this reorganization of operations does not change their computational cost. It is still given by (2.34).

For example, we can use this observation and the expression in (2.37) to arrive at the following similarity transformation:

$$Q(S_{p_1} \otimes S_{p_2})Q^{-1} = \begin{bmatrix} 1 & & & \\ & C_{\Phi_{p_1}} & & \\ & & C_{\Phi_{p_2}} & \\ & & & C_{\Phi_{p_1}} \otimes C_{\Phi_{p_2}} \end{bmatrix} \tag{2.39}$$

where

$$Q = \begin{bmatrix} I_{p_1} \otimes 1_{-\mathrm{p}_2}^t \\ I_{p_1} \otimes G_{p_2} \end{bmatrix} (R_{p_1} \otimes I_{p_2}) \tag{2.40}$$

$1_{-\mathrm{p}}$ is a column vector of p 1's

$$1_{-\mathrm{p}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^t \tag{2.41}$$

and G_p is the $(p-1) \times p$ matrix:

$$G_p = \begin{bmatrix} 1 & & & -1 \\ & 1 & & -1 \\ & & \ddots & \vdots \\ & & 1 & -1 \end{bmatrix} = \begin{bmatrix} I_{p-1} - \underline{1}_{p-1} \end{bmatrix}. \tag{2.42}$$

In general we have

$$R\left(S_{p_1^{e_1}} \otimes \cdots \otimes S_{p_k^{e_k}}\right) R^{-1} = \underset{d|n}{\oplus} \Psi(d) \tag{2.43}$$

where $R = R_{p_1^{e_1},\cdots,p_k^{e_k}}$ is given by

$$R_{p_1^{e_1},\cdots,p_k^{e_k}} = \prod_{i=k}^{1} Q(m_i, p_i^{e_i}, n_i) \tag{2.44}$$

with $m_i = \prod_{j=1}^{i-1} p_j^{e_j}$, $n_i = \prod_{j=i+1}^{k} p_j^{e_j}$ and

$$Q(a, p^e, c) = \prod_{j=0}^{e-1} \begin{bmatrix} I_a \otimes 1_{-p}^t \otimes I_{cp^j} & \\ I_a \otimes G_p \otimes I_{cp^j} & \\ & I_{ac(p^e - p^{j+1})} \end{bmatrix}. \tag{2.45}$$

$1_{-\mathrm{p}}$ and G_p are as given in (2.41) and (2.42).

Example 2.5

$$R(S_9 \otimes S_5) R^{-1} = \begin{bmatrix} 1 & & & & & & \\ & C_{\Phi_3} & & & & & \\ & & C_{\Phi_9} & & & & \\ & & & C_{\Phi_5} & & & \\ & & & & C_{\Phi_3} \otimes C_{\Phi_5} & & \\ & & & & & C_{\Phi_9} \otimes C_{\Phi_5} \end{bmatrix} \tag{2.46}$$

where

$$\begin{aligned} R &= R_{9,5} \\ &= Q(9,5,1) Q(1,9,5) \end{aligned} \tag{2.47}$$

and R can be implemented with 152 additions.

Notice the distinction between this example and example "Reduction Operations" (Section 2.3: Reduction Operations). In example "Reduction Operations" (Section 2.3: Reduction Operations) we obtained from $S_9 \otimes S_5$ a Kronecker product of two block diagonal matrices, but here we obtained a block diagonal matrix whose diagonal blocks are the Kronecker product of cyclotomic companion matrices. Each block in (2.46) represents a multi-dimensional cyclotomic convolution.

A Matlab program that carries out the operation $R_{p_1^{e_1}, \cdots, p_k^{e_k}}$ in (2.43) is `KRED()` .

```
    function x = KRED(P,E,K,x)
% x = KRED(P,E,K,x);
% P : P = [P(1),...,P(K)];
% E : E = [E(K),...,E(K)];
for i = 1:K
   a = prod(P(1:i-1).^E(1:i-1));
   c = prod(P(i+1:K).^E(i+1:K));
   p = P(i);
   e = E(i);
   for j = e-1:-1:0
      x(1:a*c*(p^(j+1))) = RED(p,a,c*(p^j),x(1:a*c*(p^(j+1))));
   end
end
```

It calls the Matlab program `RED()` .

```
    function y = RED(p,a,c,x)
% y = RED(p,a,c,x);
y = zeros(a*c*p,1);
for i = 0:c:(a-1)*c
   for j = 0:c-1
      y(i+j+1) = x(i*p+j+1);
```