# ON PROGRAMS FOR PRIME LENGTH FFTS AND CIRCULAR CONVOLUTION

*C. Sidney Burrus and Ivan W. Selesnick*

Department of Electrical and Computer Engineering - MS 366
Rice University, Houston, TX 77251-1892
(*selesi@rice.edu*)

## ABSTRACT

We describe a set of programs for circular convolution and prime length FFTs that are short, possess great structure, share many computational procedures, and cover a large variety of lengths. The programs make clear the structure of the algorithms and clearly enumerate independent computational branches that can be performed in parallel. Moreover, each of these independent operations is made up of a sequence of sub-operations which can be implemented as vector/parallel operations. This is in contrast with previously existing programs for prime length FFTs: they consist of straight line code, no code is shared between them, and they can not be easily adapted for vector/parallel implementations. We have also developed a program that automatically generates these programs for prime length FFTs.

## 1. INTRODUCTION

Because algorithms for prime lengths are building blocks for composite length FFTs (the maximum length and the variety of lengths of a PFA or WFTA algorithm depend upon the availability of prime length modules) prime length FFTs 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 [4, 5, 14, 16]. These programs rely upon Rader's observation [10] that a prime $p$ length DFT can be found by performing a $p-1$ length circular convolution. Since then, Winograd had designed algorithms for small convolutions attaining the minimum number of multiplications [16]. 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. We have previously reported the design of programs for prime lengths greater than 31 [11] but those programs required more additions than necessary and were very long. Like the previously existing ones, they consisted of a long list of instructions, did not take advantage of the attainable common computational structures and were not amenable to vector/parallel implementations.

We describe a set of programs for circular convolution and prime length FFTs that 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 [3, 15].

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 variety of prime length FFT algorithms.

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 [9]. As Stasinski notes [13], this yields algorithms possessing greater structure, simpler programs and doesn't generally require more computation. We wish to note also, that Jones [6] has advocated the the use of the Agarwal-Cooley algorithm for prime length FFTs.

## 2. PRELIMINARIES

It is useful to use the matrix formulation of convolution:

$$Y(s) = \langle H(s)X(s)\rangle_{M(s)} \iff y = \left(\sum_{k=0}^{n-1} h_k C_M^k\right) x \quad (1)$$

where $C_M$ is the companion matrix of $M(s)$. In the case of circular convolution, $M(s) = s^n - 1$ and $C_{s^n-1}$ is the circular shift matrix denoted by $S_n$.

Similarity transformations can be used to interpret the action of some convolution algorithms [12]. The Winograd algorithm can be described by noting the similarity,

$$S_n \sim \begin{bmatrix} C_{\Phi_1} & & \\ & \ddots & \\ & & C_{\Phi_n} \end{bmatrix} = \bigoplus_{d|n} C_{\Phi_d} \quad (2)$$

where $\Phi_d(s)$ is the $d^{th}$ cyclotomic polynomial. In this case, each block represents a 'cyclotomic convolution'.

Similarly, the split nesting algorithm [9] combines the structures of the Winograd and Agarwal-Cooley algorithms [1], and can be described by noting the similarity,

$$S_n \sim \bigoplus_{d|n} \Psi(d). \quad (3)$$

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

This work has been supported by DARPA under an NDSEG fellowship.

**Example 1:** When $n = 45$, eq (3) becomes

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

Here, a multidimensional cyclotomic convolution, represented by $\Psi(d)$, replaces each cyclotomic convolution in Winograd's algorithm (represented by $C_{\Phi_d}$ in (2)).

## 2.1. Prime Factor Permutations

When $n = n_1 \cdots n_k$ and $n_1, \ldots, n_k$ are pairwise relatively prime, one can convert a one dimensional $n$ point circular convolution to a $k$ dimensional one: $S_n = P^t(S_{n_1} \otimes \cdots \otimes S_{n_k})P$ where $P = P_{n_1,\ldots,n_k}$ is the prime factor permutation.

## 2.2. Reduction Operations

To obtain the block diagonal form of eq (3), let $\underline{1}_p$ be a column vector of $p$ 1's and $G_p$ be the $(p-1) \times p$ matrix:

$$G_p = \begin{bmatrix} 1 & & & -1 \\ & 1 & & -1 \\ & & \ddots & \vdots \\ & & 1 & -1 \end{bmatrix} \tag{4}$$

Then

$$R\left(S_{p_1^{e_1}} \otimes \cdots \otimes S_{p_k^{e_k}}\right) R^{-1} = \bigoplus_{d|n} \Psi(d) \tag{5}$$

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

$$R_{p_1^{e_1},\ldots,p_k^{e_k}} = \prod_{i=k}^{1} Q(m_i, p_i^{e_i}, n_i) \tag{6}$$

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

$$Q(a, p^e, c) = \prod_{j=0}^{e-1} \begin{bmatrix} I_a \otimes \underline{1}_p^t \otimes I_{cp^j} & \\ I_a \otimes G_p \otimes I_{cp^j} & \\ & I_{ac(p^e - p^{j+1})} \end{bmatrix}. \tag{7}$$

The number of additions incurred by $R$ is given by $2nk - 2n \sum_{i=1}^{k} \frac{1}{p_i^{e_i}}$ where $n = p_1^{e_1} \ldots p_k^{e_k}$.

**Example 2:** $RP\,(S_{45})\,P^t R^{-1}$ equals the block diagonal matrix in example 1 where $P = P_{9,5}$ and $R = R_{9,5}$.

These are some of the computational procedures common among the prime length FFTs programs.

## 3. BILINEAR FORMS FOR CONVOLUTION

Many convolution algorithms can be written as $y = F\{Eh * Dx\}$ where $*$ denotes point-wise multiplication [2]. For convenience, if $y = F\{Eh * Dx\}$ computes the $n$ point linear convolution of $h$ and $x$ then we say "$(D, E, F)$ describes a bilinear form for $n$ point linear convolution." For example, $(D, D, F)$ describes a 2 point linear convolution where

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} 1 & 0 & 0 \\ -1 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Similarly, we can write a bilinear form for cyclotomic convolution. Let $X(s)$ and $H(s)$ be polynomials of degree $\phi(d) - 1$ where $\phi(\cdot)$ is the Euler totient function. Let $e_k$ be the $k^{th}$ standard basis vector. If $A$, $B$ and $C$ satisfy $(C_{\Phi_d})^k = C diag(Be_k)A$ for $0 \le k \le \phi(d) - 1$, then the coefficients of $Y(s) = \langle X(s)H(s)\rangle_{\Phi_d(s)}$ are given by $y = C\{Bh * Ax\}$. As above, if $y = C\{Bh * Ax\}$ computes the $d$-cyclotomic convolution, then we say "$(A, B, C)$ describes a bilinear form for $\Phi_d(s)$ convolution."

But since $\langle X(s)H(s)\rangle_{\Phi_d(s)}$ can be found by reducing $X(s)H(s)$, a cyclotomic convolution algorithm can be derived by following a linear convolution algorithm by a reduction: If $G$ is the appropriate reduction matrix and if $(A, B, F)$ describes a bilinear form for a $\phi(d)$ point *linear* convolution, then $(A, B, GF)$ describes a bilinear form for $\Phi_d(s)$ convolution. That is, $y = GF\{Bh * Ax\}$ computes the coefficients of $\langle X(s)H(s)\rangle_{\Phi_d(s)}$.

## 3.1. Circular Convolution

Consider $p^e$ point circular convolution. Since

$$S_{p^e} = R_{p^e}^{-1}\left(\oplus_{i=0}^{e} C_{\Phi_{p^i}}\right) R_{p^e},$$

the circular convolution is decomposed into disjoint $\Phi_{p^i}(s)$ convolutions. If $(A_{p^i}, B_{p^i}, C_{p^i})$ describes a bilinear form for $\Phi_{p^i}(s)$ convolution and if

$$A = 1 \oplus A_p \oplus \cdots \oplus A_{p^e} \qquad B = 1 \oplus B_p \oplus \cdots \oplus B_{p^e} \tag{8}$$

$$C = 1 \oplus C_p \oplus \cdots \oplus C_{p^e} \tag{9}$$

then $(AR_{p^e}, BR_{p^e}, R_{p^e}^{-1}C)$ describes a bilinear form for $p^e$ point circular convolution. In particular, if $(D_d, E_d, F_d)$ describes a bilinear form for $d$ point linear convolution, then $A_{p^i}$, $B_{p^i}$ and $C_{p^i}$ can be taken to be $D_{\phi(p^i)}$, $E_{\phi(p^i)}$ and $G_{p^i}F_{\phi(p^i)}$ where $G_{p^i}$ represents the appropriate reduction. Specifically, $G_{p^i}$ has the following form

$$G_{p^i} = \begin{bmatrix} I_{(p-1)p^{i-1}} & -\underline{1}_{p-1} \otimes I_{p^{i-1}} & \begin{bmatrix} I_{(p-2)p^{i-1}-1} \\ 0_{p^{i-1}+1,(p-2)p^{i-1}-1} \end{bmatrix} \end{bmatrix}$$

if $p \ge 3$, while $G_{2^i} = \begin{bmatrix} I_{2^{i-1}} & \begin{bmatrix} -I_{2^{i-1}-1} \\ 0_{1,2^{i-1}-1} \end{bmatrix} \end{bmatrix}$.

## 3.2. The Split Nesting Algorithm

Let $n = p_1^{e_1} \cdots p_k^{e_k}$. To obtain a bilinear form for $n$ point circular convolution, we can combine bilinear forms for smaller convolutions [9]. If $(A_{p_j^i}, B_{p_j^i}, C_{p_j^i})$ describes a bilinear form for $\Phi_{p_j^i}(s)$ convolution and if

$$A = \oplus_{d|n} A_d \qquad B = \oplus_{d|n} B_d \qquad C = \oplus_{d|n} C_d \tag{10}$$

with

$$A_d = \otimes_{p|d, p\in\mathcal{P}} A_{H_d(p)} \qquad B_d = \otimes_{p|d, p\in\mathcal{P}} B_{H_d(p)} \tag{11}$$

$$C_d = \otimes_{p|d, p\in\mathcal{P}} C_{H_d(p)} \tag{12}$$

then $(ARP, BRP, P^t R^{-1} C)$ describes a bilinear form for $n$ point circular convolution. That is,

$$y = P^t R^{-1} C\{BRPh * ARPx\}. \tag{13}$$

**Example 3:** A 45 point circular convolution algorithm:

$$y = P^t R^{-1} C \{BRPh * ARPx\} \tag{14}$$

where $P = P_{9,5}$, $R = R_{9,5}$,

$$\begin{aligned} A &= 1 \oplus A_3 \oplus A_9 \oplus A_5 \oplus (A_3 \otimes A_5) \oplus (A_9 \otimes A_5) \\ B &= 1 \oplus B_3 \oplus B_9 \oplus B_5 \oplus (B_3 \otimes B_5) \oplus (B_9 \otimes B_5) \\ C &= 1 \oplus C_3 \oplus C_9 \oplus C_5 \oplus (C_3 \otimes C_5) \oplus (C_9 \otimes C_5) \end{aligned}$$

and where $(A_{p_j^i}, B_{p_j^i}, C_{p_j^i})$ describes a bilinear form for $\Phi_{p_j^i}(s)$ convolution.

Note that $RP$ block diagonalizes $S_n$ and each diagonal block represents a multidimensional cyclotomic convolution. Correspondingly, $A$, $B$ and $C$ are block diagonals.

## 3.3. The Matrix Exchange Property

When $h$ is known and fixed, $BRPh$ can be pre-computed and stored. But $P^t R^{-1} C$ is more complicated than $BRP$ so it is advantageous to absorb the work of $P^t R^{-1} C$ instead of $BRP$ into the multiplicative constants. Let $J$ be the reversal matrix. Applying the matrix exchange property [4] to eq (13) one gets

$$\boxed{y = JP^t R^t B^t \left\{C^t R^{-t} PJh * ARPx\right\}. \qquad (15)}$$

## 3.4. Implementing Kronecker Products

In the algorithm above we encountered expressions of the form $A_1 \otimes A_2 \otimes \cdots \otimes A_n$. To calculate the product $(\otimes_i A_i)\, x$ it is computationally advantageous to factor $\otimes_i A_i$ into terms of the form $I \otimes A_i \otimes I$ [1]. For the Kronecker product $\otimes_{i=1}^n A_i$ there are $n!$ possible different ways in which to order the operations $A_i$. To find the best factorization of $\otimes_i A_i$ it is necessary only to compute the ratios $(rows_i - cols_i)/adds_i$ and to order them in an non-decreasing order [1].

## 3.5. Vector/Parallel Interpretation

The command $I \otimes A \otimes I$ where $\otimes$ is the Kronecker (or Tensor) product can be interpreted as a vector/parallel command [3, 15]. In these references, the implementation of these commands is discussed in detail and it was found that the Tensor product is "an extremely useful tool for matching algorithms to computer architectures [3]."

The expression $I \otimes A$ can easily be seen to represent a parallel command. Since each diagonal block represents exactly the same operation, this form is amenable to implementation on a computer with a parallel architectural configuration. The expression $A \otimes I$ can be similarly seen to represent a vector command, see [3].

In [3] it is suggested that it might be practical to develop tensor product compilers. The FFT programs described here will be well suited for such compilers.

## 4. PROGRAMS FOR CIRCULAR CONVOLUTION

In writing a program that computes the circular convolution of $h$ and $x$ using the bilinear form (15) we have written subprograms that carry out the action of $P$, $P^t$, $R$, $R^t$, $A$ and $B^t$. We are assuming, as is usually done, that $h$ is fixed and known so that $u = C^t R^{-t} PJh$ can be pre-computed and stored. To compute these constants $u$ we need additional subprograms to carry out $C^t$ and $R^{-t}$ but the efficiency with which we compute $u$ is unimportant since this is done beforehand and $u$ is stored.

We noted above that bilinear forms for linear convolution, $(D_d, E_d, F_d)$, can be used to construct forms for cyclotomic convolutions. Specifically, to obtain a bilinear form $(A_{p^i}, B_{p^i}, C_{p^i})$ for $\Phi_{p^i}(s)$ convolution we can take $A_{p^i} = D_{\phi(p^i)}$, $B_{p^i} = E_{\phi(p^i)}$ and $C_{p^i} = G_{p^i} F_{\phi(p^i)}$. When we use bilinear forms for linear convolution obtained by nesting [2] one can take, for example, $D_4 = D_2 \otimes D_2$ and $D_6 = D_2 \otimes D_3$. It should also be noted that in many convolution algorithms $D_d = E_d$, so that only $D_d$ and $D_d^t$ are needed.

It is possible to make further improvements to the operation counts. Specifically, algorithms for prime power cyclotomic convolution based on the polynomial transform, although more complicated, will give improvements for the longer lengths [8, 9]. These improvements can be included in the code generating program we have developed.

## 5. PROGRAMS FOR PRIME LENGTH FFTS

Using the circular convolution algorithms described above, we can easily design algorithms for prime length FFTs. The only modifications involve the Rader's permutation [10] and the calculation of the DC term.

The multiplicative constants, the input and output permutation are each stored as vectors. These vectors are then passed to the prime length FFT program, which consists of the appropriate function calls, see the example program below, included for exposition. In previous prime length FFT modules, the input and output permutations can be completely absorbed in to the computational instructions. This is possible because they are written as straight line code. If desired, the code generating program described here can be modified so that it also produces straight line code.

Table 1 lists the arithmetic operations incurred by the FFT programs we have generated. The number of additions and multiplications are the same as previously existing programs for prime lengths up to and including 13. For $p = 17$ a program with 70 multiplications and 314 additions has been written, and for $p = 19$ a program with 76 multiplications and 372 additions has been written [5]. Thus for the length $p = 17$, the program we have generated requires fewer total arithmetic operations, while for $p = 19$, the new program uses more. There are several table of operation counts in [7]. For each, the algorithms described here use fewer additions and fewer multiplications. The focus of [7], however, is the implementation on various architectures and the advantage that can be gained from matching algorithms with architectures. Although we have not executed the programs described in this paper on these computers, they are, as mentioned above, written to be easily adapted to parallel/vector computers.

Some algorithms for prime length FFTs in [7, 13] for which operation counts are given assume that a row-column method can be used for convolution. Unfortunately, however, the convolution of two sequences can not be found in general by forming two arrays, by convolving their rows, and by then convolving their columns.

Table 1: Operation counts for prime length FFTs

| P | M | A | P | M | A | P | M | A |
|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 12 | 41 | 280 | 1140 | 241 | 3280 | 13020 |
| 5 | 10 | 34 | 43 | 256 | 1440 | 271 | 3760 | 18152 |
| 7 | 16 | 72 | 61 | 400 | 1908 | 281 | 4480 | 19036 |
| 11 | 40 | 168 | 71 | 640 | 3112 | 337 | 5248 | 22268 |
| 13 | 40 | 188 | 73 | 532 | 2504 | 379 | 6016 | 32880 |
| 17 | 82 | 274 | 109 | 940 | 5096 | 421 | 6400 | 29412 |
| 19 | 76 | 404 | 113 | 1312 | 5516 | 433 | 7708 | 32864 |
| 29 | 160 | 836 | 127 | 1216 | 6760 | 541 | 9400 | 43020 |
| 31 | 160 | 776 | 181 | 1900 | 8936 | 631 | 12160 | 56056 |
| 37 | 190 | 990 | 211 | 2560 | 12368 | 757 | 15040 | 76292 |

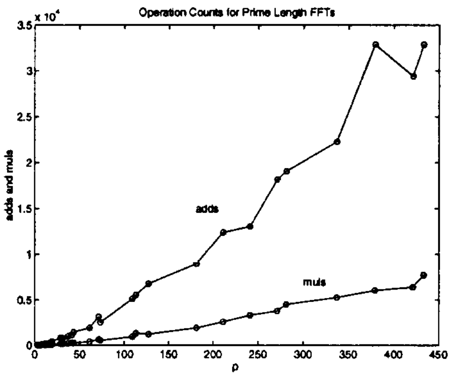

Figure 1: Plot of additions and multiplications incurred by prime length FFTs.

## 6. CONCLUSION

We have found that by exploiting the structure of the split nesting algorithm we have been able to write a program that automatically generates compact readable code for convolution and prime length FFT programs. By recognizing, also, that the algorithms for different lengths share many of the same computational structures, the code we generate is made up of calls to a relatively small set of functions. Accordingly, the subroutines can be designed to specifically suit a given architecture.

A full length version of this paper is available on the World Wide Web at http://jazz.rice.edu and from SPIB at http://bellona.cs.rice.edu/spib.html.

## 7. REFERENCES

[1] R. C. Agarwal and J. W. Cooley. New algorithms for digital convolution. *IEEE Trans. Acoust., Speech, Signal Proc.*, 25(5):392–410, October 1977.

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

[3] J. Granata, M. Conner, and R. Tolimieri. The Tensor product: A mathematical programming language for FFTs and other fast DSP operations. *IEEE Signal Processing Magazine*, 9(1):40–48, January 1992.

[4] H. W. Johnson and C. S. Burrus. On the structure of efficient DFT algorithms. *IEEE Trans. Acoust., Speech, Signal Proc.*, 33(1):248–254, February 1985.

[5] H. W. Johnson and S. Burrus. Large DFT Modules: 11, 13, 17, 19, 25. Technical Report 8105, Rice University, 1981.

[6] K. J. Jones. Prime number DFT computation via parallel circular convolvers. *IEE Proceedings, part F*, 137(3):205–212. June 1990.

[7] C. Lu, J. W. Cooley, and R. Tolimieri. FFT algorithms for prime transform sizes and their implementations of VAX, IBM3090VF, and IBM RS/6000. *IEEE Trans. Acoust., Speech, Signal Proc.*, 41(2):638–648, February 1993.

[8] H. J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Trans. Acoust., Speech, Signal Proc.*, 28(2):205–215, April 1980.

[9] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, 1982.

[10] C. M. Rader. Discrete Fourier transform when the number of data samples is prime. *Proc. IEEE,* 56(6):1107–1108, June 1968.

[11] I. W. Selesnick and C. S. Burrus. Automating the design of prime length FFT programs. In *Proc. of ISCAS*, 1992.

[12] I. W. Selesnick and C. S. Burrus. Extending Winograd's small convolution algorithm to longer lengths. In *Proc. of ISCAS*, 1994.

[13] R. Stasinski. Easy generation of small-n discrete Fourier transform algorithms. *IEE Proceedings, part G*, 133(3):133–139, June 1986.

[14] C. Temperton. A new set of minimum-add small-n rotated DFT modules. *J. of Comput. Physics*, 75:190–198, 1988.

[15] R. Tolimieri, M. An, and C. Lu. *Algorithms for Discrete Fourier Transform and Convolution.* Springer-Verlag, 1989.

[16] S. Winograd. *Arithmetic Complexity of Computations.* SIAM, 1980.

## A. A 31 POINT FFT PROGRAM

```
function y = fft31(x,u,ip,op)
% y : the 31 point DFT of x
% u : a vector of precomputed constants
% ip,op : input, ouput permutation
x = x(ip);                          % input permutation
x(2:31) = KRED([2,3,5],[1,1,1],x(2:31)); % reduction
y(1) = x(1)+x(2);                   % DC term calculation
% -------------- block : 1 -------------------------
y(2) = x(2)*u(1);
% -------------- block : 2 -------------------------
y(3) = x(3)*u(2);
% -------------- block : 3 -------------------------
v = ID2I(1,1,x(4:5));         % I(1) kron D2 kron I(1)
v = v.*u(3:5);
y(4:5) = ID2tI(1,1,v);        % I(1) kron D2' kron I(1)
% -------------- block : 6 = 2 * 3 -----------------
v = ID2I(1,1,x(6:7));         % I(1) kron D2 kron I(1)
v = v.*u(6:8);
y(6:7) = ID2tI(1,1,v);        % I(1) kron D2' kron I(1)
% -------------- block : 5 -------------------------
v = ID2I(1,2,x(8:11));        % I(1) kron D2 kron I(2)
v = ID2I(3,1,v);              % I(3) kron D2 kron I(1)
v = v.*u(9:17);
v = ID2tI(1,3,v);             % I(1) kron D2' kron I(3)
y(8:11) = ID2tI(2,1,v);       % I(2) kron D2' kron I(1)
% -------------- block : 10 = 2 * 5 ----------------
v = ID2I(1,2,x(12:15));       % I(1) kron D2 kron I(2)
v = ID2I(3,1,v);              % I(3) kron D2 kron I(1)
v = v.*u(18:26);
v = ID2tI(1,3,v);             % I(1) kron D2' kron I(3)
y(12:15) = ID2tI(2,1,v);      % I(2) kron D2' kron I(1)
% -------------- block : 15 = 3 * 5 ----------------
v = ID2I(1,4,x(16:23));       % I(1) kron D2 kron I(4)
v = ID2I(3,2,v);              % I(3) kron D2 kron I(2)
v = ID2I(9,1,v);              % I(9) kron D2 kron I(1)
v = v.*u(27:53);
v = ID2tI(1,9,v);             % I(1) kron D2' kron I(9)
v = ID2tI(2,3,v);             % I(2) kron D2' kron I(3)
y(16:23) = ID2tI(4,1,v);      % I(4) kron D2' kron I(1)
% -------------- block : 30 = 2 * 3 * 5 ------------
v = ID2I(1,4,x(24:31));       % I(1) kron D2 kron I(4)
v = ID2I(3,2,v);              % I(3) kron D2 kron I(2)
v = ID2I(9,1,v);              % I(9) kron D2 kron I(1)
v = v.*u(54:80);
v = ID2tI(1,9,v);             % I(1) kron D2' kron I(9)
v = ID2tI(2,3,v);             % I(2) kron D2' kron I(3)
y(24:31) = ID2tI(4,1,v);      % I(4) kron D2' kron I(1)
% -------------------------------------------------
y(2) = y(1)+y(2);                   % DC term calculation
y(2:31) = tKRED([2,3,5],[1,1,1],y(2:31)); % reduction
y = y(op);                          % output permutation
```