REVIEW ON FAST COMPLEX MULTIPLICATION ALGORITHMS AND IMPLEMENTATION

[1] L . Malathi, [2] Dr. A. Bharathi, [3] Dr. A. N. Jayanthi
[1] Assistant Professor, Sri Ramakrishna Institute of Technology, Department of ECE, India
[2] Professor, Bannari Amman Institute of Technology, Department of IT, India
[3] Associate Professor, Sri Ramakrishna Institute of Technology, Department of ECE, India

Abstract
Multipliers are key components of many high performance systems such as FIR filters, microprocessor, digital signal processor, etc. The performance of the system is determined based on the performance of the multiplier. Complex number operations are playing a major role in many digital signal processing based applications. The overall performance of the multiplication is based on repeated addition and subtraction. There is the possibility to improve the performance of multiplier is only by concentrate on the factors of delay, adders, subtractions and thereby speed and accuracy will be increased. While involved in complex number multiplication
Index Terms: RCBNS, Algorithms, FFT

I. INTRODUCTION

Most of the algorithms related to efficient multiplies are reviewed. Fast Fourier transforms are widely used for applications in engineering, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805. In 1994, Gilbert Strang [9] [10] [11] described the FFT as "the most important numerical algorithm of our lifetime", and it was included in Top 10 Algorithms of 20th Century by the IEEE journal Computing in Science & Engineering [5].

The best-known FFT algorithms depend upon the factorization of N, but there are FFTs with O(N log N) complexity for all N, even for prime N. Many FFT algorithms only depend on the fact that $e^{-2\pi i/N}$ is an N-th primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.

FFT-related algorithms: Cooley–Tukey FFT algorithm, Prime-factor FFT algorithm, Bruun's FFT algorithm, Rader's FFT algorithm, Bluestein's FFT algorithm, Goertzel algorithm – Computes individual terms of discrete Fourier transform are may be permitted in complex multiplication which are discussed in the following.

II. REVIEW ON METHODS

A. Divide and Conquer Method

The divide-and-conquer paradigm is often used to find the optimal solution of a problem. Its basic idea is to decompose a given problem into two or more similar, but simpler, sub problems, to solve them in turn, and to compose their solutions to solve the given problem. Problems of sufficient simplicity are solved directly.

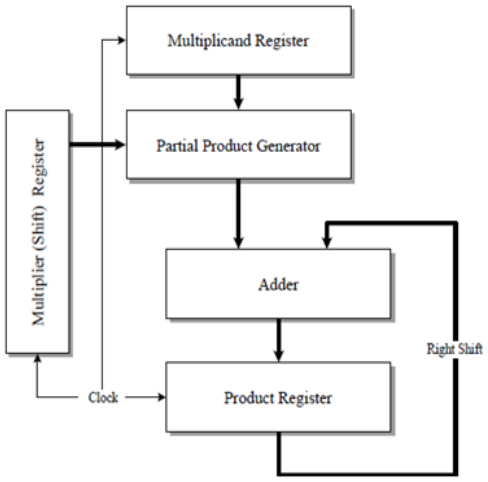

Fig. 1. Basic Iterative Multiplier

An important application of divide and conquer is in optimization, where if the search space is reduced by a constant factor at each step, the overall algorithm has the same asymptotic complexity as the pruning step, with the constant depending on the reduced factor.

B. Redundant Complex Binary Number System

A Redundant Complex Binary Number System (RCBNS) consists of both real and imaginary-radix number systems that form a redundant integer digit set. This system is formed by using complex radix of (-1+j) and a digit set of α = 3, where α assumes a value of -3, -2, -1, 0, 1, 2, 3. The arithmetic operations of complex numbers with this system treat the real and imaginary parts as one unit. The carry-free addition has the advantage of Redundancy in number representation in the arithmetic operations.

Results of the arithmetic operations are in the RCBNS form. The two methods for conversion from the RCBNS form to the standard binary number form have been presented. In this paper the RCBNS reduces the number of steps required to perform complex number arithmetic operations, thus enhancing the speed.

III. REVIEW ON ALGORITHMS

A. Cooley–Tukey Algorithm

The Cooley–Tukey algorithm, named after J. W. Cooley and John Tukey, is the most common fast Fourier transform (FFT) algorithm. It re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size $N = N_1N_2$ in terms of N_1 smaller DFTs of sizes N_2, recursively, to reduce the computation time to O(N log N) for highly composite N (smooth numbers). Because of the algorithm's importance, specific variants and implementation styles have become known by their own names, as described below.

Because of the Cooley–Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT. For example, Rader's or Bluestein's algorithm can be used to handle large prime factors that cannot be decomposed by Cooley–Tukey or the prime-factor algorithm can be exploited for greater efficiency in separating out relatively prime factors.

The algorithm, along with its recursive application, was invented by Carl Friedrich Gauss. Cooley and Tukey independently rediscovered and popularized it 160 years later.

A radix-2 decimation-in-time (DIT) FFT is the simplest and most common form of the Cooley–Tukey algorithm, although highly optimized Cooley–Tukey implementations typically use other forms of the algorithm as described below. Radix-2 DIT divides a DFT of size N into two interleaved DFTs (hence the name "radix-2") of size $N/2$ with each recursive stage.

The discrete Fourier transform (DFT) is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk},$$

The radix-2 DIT algorithm rearranges the DFT of the function x_n into two parts: a sum over the even-numbered indices n=2m and a sum over the odd-numbered indices n=2m+1:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of even-indexed part of } x_n} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of odd-indexed part of } x_n} = E_k + e^{-\frac{2\pi i}{N}k} O_k.$$

$$
\begin{aligned}
X_{k+\frac{N}{2}} &= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} m\left(k+\frac{N}{2}\right)} + e^{-\frac{2\pi i}{N}\left(k+\frac{N}{2}\right)} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} m\left(k+\frac{N}{2}\right)} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk} e^{-2\pi mi} + e^{-\frac{2\pi i}{N} k} e^{-\pi i} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk} e^{-2\pi mi} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk} - e^{-\frac{2\pi i}{N} k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk} \\
&= E_k - e^{-\frac{2\pi i}{N} k} O_k
\end{aligned}
$$

B. Prime-Factor Algorithm (PFA)

The prime-factor algorithm (PFA), also called the Good–Thomas algorithm (1958/1963), is a fast Fourier transform (FFT) algorithm that re-expresses the discrete Fourier transform (DFT) of a size $N = N_1N_2$ as a two-dimensional $N_1 \times N_2$ DFT, but *only* for the case where N_1 and N_2 are relatively prime. These smaller transforms of size N_1 and N_2 can then be evaluated by applying PFA recursively or by using some other FFT algorithm.

PFA should not be confused with the mixed-radix generalization of the popular Cooley–Tukey algorithm, which also subdivides a DFT of size $N = N_1N_2$ into smaller transforms of size N_1 and N_2. The latter algorithm can use *any* factors (not necessarily relatively prime), but it has the disadvantage that it also requires extra multiplications by roots of unity called twiddle factors, in addition to the smaller transforms. On the other hand, PFA has the disadvantages that it only works for relatively prime factors (e.g. it is useless for power-of-two sizes) and that it requires a more complicated re-indexing of the data based on the Chinese remainder theorem (CRT). However, that PFA can be combined with mixed-radix Cooley–Tukey, with the former factorizing N into relatively prime components and the latter handling repeated factors.

Recall that the DFT is defined by the formula:

$$
X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1.
$$

The PFA involves a re-indexing of the input and output arrays, which when substituted into the DFT formula transforms it into two nested DFTs (a two-dimensional DFT).

(i) Re-indexing

Suppose that $N = N_1N_2$, where N_1 and N_2 are relatively prime. In this case, we can define a re-indexing of the input n and output k by:

$$
\begin{aligned}
n &= n_1N_2 + n_2N_1 \mod N, \\
k &= k_1N_2^{-1}N_2 + k_2N_1^{-1}N_1 \mod N,
\end{aligned}
$$

(ii) DFT re-expression

The above re-indexing is then substituted into the formula for the DFT, and in particular into the product nk in the exponent. Because $e^{2\pi i} = 1$, this exponent is evaluated modulo N: any $N_1N_2 = N$ cross term in the nk product can be set to zero. (Similarly, X_k and x_n are implicitly periodic in N, so their subscripts are evaluated modulo N.) The remaining terms give:

$$
X_{k_1N_2^{-1}N_2 + k_2N_1^{-1}N_1} = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} x_{n_1N_2+n_2N_1} e^{-\frac{2\pi i}{N_2} n_2k_2} \right) e^{-\frac{2\pi i}{N_1} n_1k_1}
$$

The inner and outer sums are simply DFTs of size N_2 and N_1, respectively.

(Here, N_1^{-1} N_1 is unity when evaluated modulo N_2 in the inner sum's exponent, and vice versa for the outer sum's exponent.)

C. BRUUN'S ALGORITHM

Bruun's algorithm is a fast Fourier transform (FFT) algorithm based on an unusual recursive polynomial-factorization approach, proposed for powers of two by G. Bruun in 1978 and generalized to arbitrary even composite sizes by H. Murakami in 1996. Because its operations involve only real coefficients until the last computation stage, it was initially proposed as a way to efficiently compute the discrete Fourier transform (DFT) of real data. Bruun's algorithm has not seen widespread use, however, as approaches based on the ordinary Cooley–Tukey FFT algorithm have been successfully adapted to real data with at least as much efficiency. Furthermore, there is evidence that Bruun's algorithm may be intrinsically less accurate than Cooley–Tukey in the face of finite numerical precision (Storn, 1993).

(i) Brunn factorization

The basic Bruun algorithm for powers of two $N=2^n$ factorizes $z^{2n}-1$ recursively via the rules:

$$z^{2M}-1=(z^M-1)(z^M+1)$$
$$z^{4M}+az^{2M}+1=(z^{2M}+\sqrt{2-a}z^M+1)(z^{2M}-\sqrt{2-a}z^M+1)$$

where a is a real constant with $|a| \le 2$. If $a=2\cos(\phi), \phi \in (0,\pi)$, then

$\sqrt{2+a}=2\cos\frac{\phi}{2}$ and $\sqrt{2-a}=2\cos(\frac{\pi}{2}-\frac{\phi}{2})$

$$p_{s,0}(z)=p(z) \mod \left(z^{2^{n-s}}-1\right)$$

and

$$p_{s,m}(z)=p(z) \mod \left(z^{2^{n-s}}-2\cos\left(\frac{m}{2^s}\pi\right)z^{2^{n-1-s}}+1\right) \quad m=1,2,\ldots,2^s-1$$

$$X_k=p(e^{2\pi i\frac{k}{2^n}})$$

D. Rader's Algorithm

Rader's algorithm (1968), named for Charles M. Rader of MIT Lincoln Laboratory, is a fast Fourier transform (FFT) algorithm that computes the discrete Fourier transform (DFT) of prime sizes by re-expressing the DFT as a cyclic convolution (the other algorithm for FFTs of prime sizes, Bluestein's algorithm, also works by rewriting the DFT as a convolution).

Since Rader's algorithm only depends upon the periodicity of the DFT kernel, it is directly applicable to any other transform (of prime order) with a similar property, such as a number-theoretic transform or the discrete Hartley transform.

The algorithm can be modified to gain a factor of two savings for the case of DFTs of real data, using a slightly modified re-indexing/permutation to obtain two half-size cyclic convolutions of real data;[2] an alternative adaptation for DFTs of real data uses the discrete Hartley transform.[3]

rewrite the DFT using these new indices p and q as:

E. Bluestein's FFT algorithm

Beginning with the DFT

$$X(k)=\sum_{n=0}^{N-1}x(n)W^{-kn}, \quad k=0,1,2,\ldots,N-1,$$

where $W \triangleq \exp(j2\pi/N)$

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1}x(n)W^{-kn}W^{\frac{1}{2}(n^2+k^2)}W^{-\frac{1}{2}(n^2+k^2)} \\ &= W^{-\frac{1}{2}k^2}\sum_{n=0}^{N-1}\left[x(n)W^{-\frac{1}{2}n^2}\right]W^{\frac{1}{2}(k-n)^2} \\ &= W^{-\frac{1}{2}k^2}(x_q * w_q)_k, \end{aligned}$$

$$X_0=\sum_{n=0}^{N-1}x_n,$$

$$X_{g^{-p}}=x_0+\sum_{q=0}^{N-2}x_{g^q}e^{-\frac{2\pi i}{N}g^{-(p-q)}} \qquad p=0,\ldots,N-2.$$

F. Goertzel Algorithm

The Goertzel algorithm [7] is a technique in digital signal processing (DSP) that provides a means for efficient evaluation of individual terms of the discrete Fourier transform (DFT), thus making it useful in certain practical applications, such as recognition of DTMF tones produced by the buttons pushed on a telephone keypad. The algorithm was first described by Gerald Goertzel in 1958.[1]

Like the DFT, the Goertzel algorithm analyses one selectable frequency component from a discrete signal.[2][3][4] Unlike direct DFT calculations, the Goertzel algorithm applies a single real-valued coefficient at each iteration, using real-valued arithmetic for real-valued input sequences. For covering a full spectrum, the Goertzel algorithm has a higher order of complexity than fast Fourier transform (FFT) algorithms, but for computing a small number of selected frequency components, it is more numerically efficient. The simple structure of the Goertzel algorithm makes it well suited to small processors and embedded applications, though not limited to these.

(i) DFT computations

For the important case of computing a DFT term, the following special restrictions are applied.

The frequencies chosen for the Goertzel analysis are restricted to the special form

$$\omega_0=2\pi\frac{k}{N}.$$

The index number indicating the "frequency bin" of the DFT is selected from the set of index numbers

$$k \in \{0,1,2,\ldots,N-1\}$$

Making these substitutions into equation (6) and observing that the term $e^{+j2\pi k} = 1$

$$y[N] = \sum_{n=0}^{N} x[n] e^{-j2\pi \frac{nk}{N}}$$

$$\begin{aligned} y[N] &= s[N] - e^{-j2\pi \frac{k}{N}} s[N-1] \\ &= (2\cos(\omega_0) s[N-1] - s[N-2]) - e^{-j2\pi \frac{k}{N}} s[N-1] \\ &= e^{j2\pi \frac{k}{N}} s[N-1] - s[N-2]. \end{aligned}$$

G. Gauss Algorithm

Re + j Im = (W + j X) (Y + j Z)
= (WY-XZ) + j (XY + WZ)

Multiply	W	X
Y	WY	jXY
jZ	jWZ	-XZ

As per Gauss's algorithm *[1][2][3]* for complex number multiplication gives two separate equations to calculate real and imaginary part of the final result. From eqn. the real part of the output can be given by (WY - XZ), and the imaginary part of the result can be computed using (XY + WZ). Thus four separate multiplications and are required to produce the real as well as imaginary part numbers.

H. Complex Multiplication Algorithm

This algorithm requires three multiplications and five additions or subtractions .In 1963, Peter Ungar suggested setting *m* to *i* to obtain a similar reduction in the complex multiplication algorithm. To multiply $(a + b\,i) \cdot (c + d\,i)$, follow these steps:

1. Compute $b \cdot d$, call the result F
2. Compute $a \cdot c$, call the result G
3. Compute $(a + b) \cdot (c + d)$, call the result H
4. The imaginary part of the result is $K = H - F - G = a \cdot d + b \cdot c$
5. The real part of the result is $G - F = a \cdot c - b \cdot d$

I. Karatsuba Multiplication

The system which needs to multiply numbers in the range of several thousand digits, such as computer algebra systems and bignum libraries, long multiplication is too slow. These systems may employ Karatsuba multiplication, which was discovered in 1960 (published in 1962). The heart of Karatsuba's method lies in the observation that two-digit multiplication can be done with only three rather than the four multiplications classically required. This is an example of what is now called a *divide and conquer algorithm*. Suppose we want to multiply two 2-digit base-*m* numbers: $x_1\,m + x_2$ and $y_1\,m + y_2$:

1. Compute $x_1 \cdot y_1$, call the result F
2. Compute $x_2 \cdot y_2$, call the result G
3. Compute $(x_1 + x_2) \cdot (y_1 + y_2)$, call the result H
4. Compute $H - F - G$, call the result K; this number is equal to $x_1 \cdot y_2 + x_2 \cdot y_1$
5. Compute $F \cdot m^2 + K \cdot m + G$.

To compute these three products of *m*-digit numbers, we can employ the same trick again, effectively using recursion. Once the numbers are computed, we need to add them together (steps 4 and 5), which takes about *n* operations.

Karatsuba multiplication has a time complexity of $O(n^{\log_2 3}) \approx O(n^{1.585})$, making this method significantly faster than long multiplication. Because of the overhead of recursion, Karatsuba's multiplication is slower than long multiplication for small values of *n*; typical implementations therefore switch to long multiplication if *n* is below some threshold.

Karatsuba's algorithm is the first known algorithm for multiplication that is asymptotically faster than long multiplication,[16] and can thus be viewed as the starting point for the theory of fast multiplications.

J. Rader's Algorithm

Rader's algorithm (1968),[1] named for Charles M. Rader of MIT Lincoln Laboratory, is a fast Fourier transform (FFT) algorithm that computes the discrete Fourier transform (DFT) of prime sizes by re-expressing the DFT as a cyclic convolution (the other algorithm for FFTs

of prime sizes, Bluestein's algorithm, also works by rewriting the DFT as a convolution).

Since Rader's algorithm only depends upon the periodicity of the DFT kernel, it is directly applicable to any other transform (of prime order) with a similar property, such as a number-theoretic transform or the discrete Hartley transform.

The algorithm can be modified to gain a factor of two savings for the case of DFTs of real data, using a slightly modified re-indexing/permutation to obtain two half-size cyclic convolutions of real data;[2] an alternative adaptation for DFTs of real data uses thediscrete Hartley transform.[3]

rewrite the DFT using these new

$$X_0 = \sum_{n=0}^{N-1} x_n,$$

$$X_{g^{-p}} = x_0 + \sum_{q=0}^{N-2} x_{g^q} e^{-\frac{2\pi i}{N} g^{-(p-q)}} \qquad p = 0, \ldots, N-2.$$

indices *p* and *q* as:

IV. CONCLUSION

In FFT, complex multiplication is play an important role. Hence, if suitable algorithm included then thereby the parameters like power, delay and area can be reduced in order to design a speed systems which can be applied in various applications like digital communication, DSP applications and wherever there is a need of complex multiplication.

REFERENCES

[1] H. S. Dhillon, et al, “A Reduced Bit Multiplication Algorithm for Digital Arithmetic,” International Journal of Computational and Mathematical Sciences, 2008, pp 64-69.

[2] Man Yan Kong, J.M. Pierre, and Dhamin Al-Khalili, “Efficient FPGA Implimentation of Complex Multipliers Using the Logarithmic Number System,” 2008 IEEE. Pp 3154-3157.

[3] Langlois Rizalafande Che Ismail and Razaidi Hussin, “High Performance Complex Number Multiplier Using Booth-Wallace Algorithm,”ICSE2006 Proc. 2006, Kuala Lumpur, Malaysia, pp 786-790.

[4] *Frigo, Matteo; Johnson, Steven G. (2005).* "The Design and Implementation of FFTW3" (PDF). Proceedings of the IEEE. *93(2):216–231.* CiteSeerX 10.1.1.66.3097.doi*:*10.1109/jproc.2004.840301.

[5] *Dongarra, Jack; Sullivan, Francis (January 2000).* "Guest Editors Introduction to the top 10 algorithms", *Computing in Science Engineering. 2 (1):22–23.* doi*:*10.1109/MCISE.2000.814652.ISSN 1521-9615.

[6] Matteo Frigo and Steven G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE 93 (2), 216–231 (2005).

[7] Goertzel, G. (January 1958), "An Algorithm for the Evaluation of Finite Trigonometric Series", American Mathematical Monthly, 65(1):34–35, doi:10.2307/2310304, JSTOR 2310304

[8] H. Zaini, Florida Inst. of Technol., Melbourne, FL, USA, R.G. Deshmukh, Florida Inst. of Technol., Melbourne, FL, USA “Complex number representation in RCBNS form for arithmetic operations and conversion of the result into standard binary form”, IEEE SoutheastCon, 2003. Proceedings. 03 March 2004, Print ISBN: 0-7803-7856-3, DOI: 10.1109/SECON.2003.1268439, Publisher: IEEE, INSPEC Accession Number: 8211949

[9] *Heideman, Michael T.; Johnson, Don H.;* Burrus, Charles Sidney *(1984).* "Gauss and the history of the fast Fourier transform" (PDF). *IEEE ASSP Magazine. 1 (4): 14–21.*doi*:*10.1109/MASSP.1984.1162257.

[10] Strang, Gilbert *(May–June 1994). "Wavelets".* American Scientist.*82 (3): 250–255.* JSTOR 29775194.

[11] *Kent, Ray D.; Read, Charles (2002). Acoustic Analysis of Speech.*ISBN 0-7693-0112-6. ISBN 978-0-7693-0112-9.