Defence Science Journal, Vol 46, No. 4, October 1996, pp 253-255

Improved Method of Generating Bit Reversed Numbers for Calculating Fast Fourier Transform

T. Suresh

National Institute of Oceanography, Goa-403 004

ABSTRACT

Fast Fourier Transform (FFT) is an important tool required for signal processing in defence applications. This paper reports an improved method for generating bit reversed numbers needed in calculating FFT using radix-2. The refined algorithm takes advantage of some features of the bit reversed numbers, using intermediate array for storage and improved procedure for calculating base values required when generating bit reversed numbers.

1. INTRODUCTION

Fast Fourier Transform (FFT) is an ubiquitous tool required for processing signals in defence applications, such as radars, doppler frequency measurements, moving target indicators, sonars, underwater communications, image reconstructions and restorations, digital fitters and others. One of the noticeable features observed with 'in-place' FFT calculation is that with input data placed in a natural sequence, the output obtained for each data point from the calculation is in bit reversed position. Thus, if input data are in natural order (x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7)), the output of the FFT calculation will have data at bit reversed positions (x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7)). It is often found difficult to calculate FFT with input and output in natural sequence. Thus, it is necessary to either load input data in natural order and then reorder the output, or place the input data at bit reversed positions before the calculation to obtain the output in a natural sequence.

It is essential to have a fast permutation algorithm for reordering. This could be done either by placing each data directly at bit reversed position in the array or by reordering the data available in normal sequence to bit reversed positions. The latter method is usually adopted[1-3]. These algorithms do not actually generate bit reversed numbers to place data at bit reversed positions, but only make use of efficient methods of swapping the data from the array for placing them at bit reversed positions.

An improved method of generating bit reversed numbers is presented here, based on an earlier algorithm by Suresh[4] (Hereafter referred to as basic algorithm. Method is given in Appendix I). The modified algorithm generates a continuous stream of N bit reversed numbers for any given index n, i.e., $N = 2^n$.

These bit reversed numbers can be used as indices of the data array for placing the data at bit reversed positions.

2. METHOD

The features observed in the bit reversed numbers generated using basic algorithm are (Table 1):

(a) Base values, which are the first values in the block of four numbers, are observed to be in bit reversed sequence. These are bit reversed numbers obtained with an index value of $n-2$. Thus there are 2^{n-2} base values.

(b) The bit reversed numbers are divided into two halves. First half has even values while the next half contains odd values. These odd values are the incremental of the first half even values.

Received 30 October 1995, revised 13 May 1996

The following refined method is therefore adopted taking into consideration the above observations.

Step 1. With the index n-2, calculate the first half 2^{n-3} of bit reversed numbers using basic algorithm. These are the first half of base values. Store them in an array. base (j), $j = 0, \ldots 2^{n-3}-1$

Step 2. Calculate 2^{n-1} bit reversed numbers using above base values as given in the basic algorithm.

Step 3. Increment the base values in the array. base (j) = base (j) + 1, for $j = 0, .. 2^{n-3}-1$

Step 4. Calculate next half 2^{n-1} bit reversed numbers using the incremented base values from the array.

Table 1. Bit reversed numbers for index n = 5

Base = R0	R1	R2	R3
0	16	8	24
4	20	12	
2	18	10	
6	22	14	
1	17	9	
5	21	13	
3	19	11	
7	23	15	

Base values are given in column 1. Numbers are given in blocks. Each block contains four numbers in a sequence.

RESULTS

Using this improved algorithm, the execution speed has been found to be above six times faster than the basic algorithm. The performance of this modified algorithm as compared to the algorithm given in[3], is given in Table 2. These average execution speeds have been obtained with programs in C on PC 486/25 MHz. These are the average values obtained after 10,000 execution cycles. It is seen that the speed has been improved by using intermediate array and a modified method of generating base values.

Table 2. Execution speed of bit reversal algorithms obtained on PC/486 - 25 MHz using C language

n	NUM (ms)	TS (ms)
6	0.4	0.1
7	0.7	0.2
8	0.9	0.6
9	1.8	1.1
10	3.2	2.5

TS : Algorithm given by the author, NUM : Algorithm[3]

Modified algorithms presented here have shown to perform better. Though this method makes use of intermediate array of size 2^{n-3}, its speed of calculation is found to be above 25 per cent faster than the commonly used swapping methods for such permutations in the calculation of FFT.

ACKNOWLEDGMENTS

The author is thankful to Dr. Ehrlich Desa, Director, National Institute of Oceanography, Goa, for all the encouragement and support. Thanks are also due to Dr. Elgar Desa, for his guidance, discussions and valuable suggestion.

REFERENCES

Ahmed, N. & Rao, K.R. Orthogonal transforms for digital signal processing. Springer-Verlag, New York, 1986, p. 263.

2. Rabiner, L.R. & Gold, B. Theory and application of digital signal processing. Prentice Hall, New Jersey, 1975.

3. William, H. Press, *et al.* Numerical recipes in fortran, Cambridge University Press, New York, 1992, p. 50f.

4. Suresh, T. Generating bit reversed numbers for calculating fast fourier transform. *Computer & Geosciences*, 1995, 21(2), 349-52.

Appendix I

Method of generating sequence of bit reversed numbers for a given number of bits, *n*.

Step 1. Define Constants

Let C_1, C_2, C_3 be the constants, whose values are given as

$C_2 = 2^{n-2}$

$C_1 = C_2 + C_2$

$C_3 = C_1 + C_2$

Step 2. Generate bit reversed numbers

Numbers are calculated in blocks. Each block contains four numbers given in a sequence as R_k, R_{k+1}, R_{k+2}, R_{k+3}.

$R_k = b$

$R_{k+1} = b + C_1$

$R_{k+2} = b + C_2$

$R_{k+3} = b + C_3$

where the base value b is calculated as

Begin

$i = n$

$by = bx = 0$

do while by # bx

$i = i - 1$

$by = bx$

$t = 2^i$

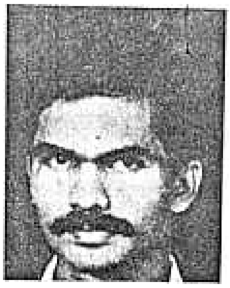
$bx = \text{mod}(R_{k+3}, t)$

End do

$b = by + t$

End

Contributor



Mr T Suresh obtained his MTech (Electronics) from Cochin University. Presently, he is working as Scientist at the National Institute of Oceanography, Goa. The areas of his interest are computers, marine optics and signal processing.