Sensors & Transducers, Vol. 154, Issue 7, July 2013, pp. 260-265

www.sensorsportal.com

Sensors & Transducers

http://www.sensorsportal.com

An Algorithm for Computing the Radix-2^n Fast Fourier Transform

*** Junyuan ZHANG, Zhenhua LI**

China University of Geosciences,
Lumo Road No.388, Wuhan City, Hubei Province, 430074, P. R. China
* Tel.: 13971619762
* E-mail: 616736194@qq.com

Received: 28 April 2013 /Accepted: 19 July 2013 /Published: 31 July 2013

Abstract: In digital signal processing, the Fast Fourier Transform (FFT) is a kind of high efficient method to calculate the discrete Fourier transform (DFT). It cuts the discrete signal sequence which the length is N for different radix sequences to operate using the way of handing back and partition. Currently, the radix-2 FFT algorithm is a popular approach to do the transform work. However, its computation is still big. This paper seeks for a more efficient algorithm to better reduce computational complexity and it starts the study from the radix-2 and the radix-4 fast Fourier transform, then explores more efficient and faster radix-8 FFT algorithm and finally extends to radix any power of 2. Experiments evidence that the radix-8 FFT algorithm outperform the radix-2 in all in circumstances, therefore prove the feasibility and efficiency of the radix-2^n.

Keywords: Signal processing, Fourier transform, Discrete Fourier transform, Fast Fourier transform, Radix-2^n.

1. Introduction

Digital signal processing is using a computer or dedicated processing equipment to transform the signal into a number or a symbol sequence, through the method of numerical calculation for signal collection, transformation, integration, valuations and recognition and other processing, so as to achieve the purpose of extracting information and easy to application [1]. It has been more and more widely used in communications, voice, images, radar, earthquake forecast, sonar, remote sensing, TV, biological medicine, space technology, automatic control, artificial intelligence, Earth and nuclear physics and many other fields [2].

1.1. Fourier Transform

Fourier transform is the most widely used signal processing method [3]. According to the different types of an original signal, the Fourier transform can be divided into four categories [4], as seen in the Table 1. Only the discrete Fourier transform (DFT) can be processed on a computer. The Fast Fourier Transform (FFT) is presented, showing a strong function of the discrete Fourier transform.

1.2. Discrete Fourier Transform

Select one cycle of a discrete Fourier series [5], there can be the discrete Fourier transform formula (1). Each point here is complex valued data. There are real part and imaginary part. The Fourier transform is composed of these two parts [6], because the complex number form can shorten transform expression. The FFT is based on the complex number form [7], so almost all the description of the Fourier transform is a complex number form.

Table 1. The Fourier Transform classification.

1.	Non-periodic continuous signal	Fourier Transform
2.	Periodic continuous signal	Fourier Series
3.	Non-periodic discrete signal	Discrete Time Fourier Transform
4.	Periodic discrete signal	Discrete Fourier Transform

$$X(k)=DFT[x(n)]=\sum_{n=0}^{N-1}x(n)e^{-j\frac{2\pi}{N}kn}=\sum_{n=0}^{N-1}x(n)W_N^{kn},0\le k\le N-1. \quad (1)$$

The coefficient has the following properties (2).

$$W_N^{k(N-n)}=W_N^{(N-k)n}=W_N^{-kn},W_N^{\frac{N}{2}}=-1,W_N^{k+\frac{N}{2}}=-W_N^k. \quad (2)$$

Using these characteristics, some items in the DFT operation can be combined and the long sequence of the DFT can be divided into several short sequences of DFT, thus greatly reducing the number of operations of the DFT. Computational amount of the DFT is proportional with N^2 [8]. N is smaller, the DFT computational amount is smaller. The FFT algorithm is based on this basic idea.

1.3. Fast Fourier Transform

The FFT calculation can be divided into three steps. An N-point time domain is firstly divided into N time-domain signals of one point [9]. Then calculate each one point signal to get its frequency domain. After this, the N frequency domains are combined in a certain order to get the required spectrum.

First step, the signal segmentation is in accordance with the law of Fig. 1. (Take a 16-points signal as an example).

As seen in the Fig. 1, it is according to bit reverse order to segment [10]. The data comparison in the sample of before bit inversion and after is shown in Table 2.

Second step, calculate each one point spectrum. Here the spectrum of a time domain point value is its own. At this time, the point is not a time-domain signal, but a frequency-domain signal.

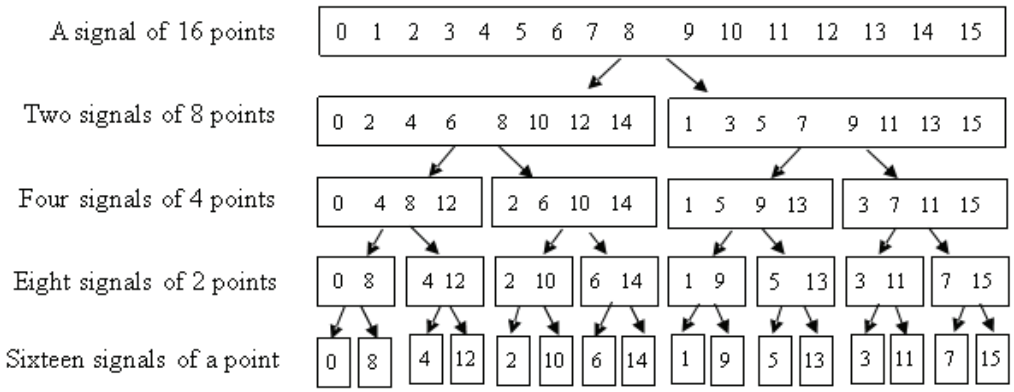

Fig. 1. The FFT signal segmentation schematic diagram.

Table 2. The data comparison before and after.

Normal order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit reversal	0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15

Third step, combining these N frequency domain signals in the opposite order of the time domain segmentation. Combine two signals of a point into a signal of two points, and then combine two signals of 2 points into a signal of 4 points, until the end.

Now take a look at the amount of computing DFT. Here x(n) is a complex sequence as said before. Therefore, the calculation of the value of X(k) needs N times complex multiplications and N-1 times complex additions. As analyzed above, the N-point DFT calculation amount is N^2, where the N-point FFT operation amount here is $N\log_2 N$ [11]. FFT speed can be quicker, such as the use of radix-4. It calculates every four points instead of every two points to improve the speed. The calculated amount of the radix-4 FFT is $N\log_4 N$. The following analysis of the study is based on radix-8 as well as even faster FFT.

2. Radix-2 FFT and Radix-4 FFT Algorithm

2.1. Radix-2 FFT

Radix-2 means that there are 2^M points [12]. Setting the input sequence length is N which is equal to 2^M. M is a positive integer. If N doesn't comply with this condition, artificially plus zero to 2^M. This sequence x(n) (n=0,1,…,N-1) is divided into two groups according to the parity, forming two sub-sequences, and bring them two into the DFT formula [13, 14]. Use the characteristics of the coefficient W_N^{kn} to simplify the formula, and then get (3) and (4).

$$X(k)=X_1(k)+W_N^k X_2(k), k=0,1,\ldots,\frac{N}{2}-1. \quad (3)$$

$$X(k+\frac{N}{2})=X_1(k)-W_N^k X_2(k), k=0,1,\ldots,\frac{N}{2}-1. \quad (4)$$

So all the value of X(k) in the interval from 0 to N-1 can be gotten as long as the value of X1(k) and X2(k) corresponding to each integer k in the range of 0 to N/2-1 can be known. The operation above can use the butterfly flow diagram to indicate [15], as seen in Fig. 2.

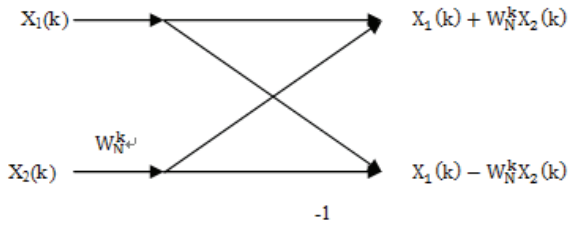

Fig. 2. The Radix-2 FFT butterfly flow diagram.

On the left side, X1(k) and X2(k) are input, the right side is output. When the N/2 is still an even number, each of the sequence with N/2 points can be further divided into two subsequences each with N/4 points. Continue like this until each subsequence has only one point. Dividing for one time, there is a level of operations [16].

2.2. Radix-4 FFT

Similar to the radix-2 FFT algorithm, the radix-4 algorithm is dividing k into four equal portions, 4k, 4k+1, 4k+2 and 4k+3. Take them into the DFT formula (1) and simplify the equation with the formula (2). Assuming there are 4^M points, M is a positive integer. In the same way, there are four formulas (5)-(8).

$$X(4k)=\sum_{n=0}^{\frac{N}{4}-1}[x(n)+x(n+\frac{N}{4})+x(n+\frac{N}{2})+x(n+\frac{3N}{4})]W_N^0 W_{N/4}^{kn} \quad (5)$$

$$X(4k+1)=\sum_{n=0}^{\frac{N}{4}-1}[x(n)-jx(n+\frac{N}{4})-x(n+\frac{N}{2})+jx(n+\frac{3N}{4})]W_N^n W_{N/4}^{kn} \quad (6)$$

$$X(4k+2)=\sum_{n=0}^{\frac{N}{4}-1}[x(n)-x(n+\frac{N}{4})+x(n+\frac{N}{2})-x(n+\frac{3N}{4})]W_N^{2n} W_{N/4}^{kn} \quad (7)$$

$$X(4k+3)=\sum_{n=0}^{\frac{N}{4}-1}[x(n)+jx(n+\frac{N}{4})-x(n+\frac{N}{2})-jx(n+\frac{3N}{4})]W_N^{3n} W_{N/4}^{kn} \quad (8)$$

And the Fig. 3 is the radix-4 butterfly. The left side is time-domain signal, and the right side is the frequency-domain signal.

3. Radix-2^n FFT Algorithm

3.1. Radix-8 FFT Algorithm

Radix-8 means that there are 8^M points. Setting the input sequence length is N which is equal to 8^M. M is a positive integer. Similar to the radix-2 FFT algorithm and the radix-4 FFT algorithm, the radix-8 algorithm is dividing k into eight equal portions, 8k, 8k+1, 8k+2, 8k+3, 8k+4, 8k+5, 8k+6 and 8k+7 (k=0,1,…,(N/8)-1). This sequence x(n) (n=0,1,…, N-1) is divided into eight groups, forming eight sub-sequences. Take them into the DFT formula, as seen in (9).

$$\begin{aligned}X(8k+p)&=\sum_{m=0}^{7}\sum_{n=0}^{(N/8)-1}x(n+\frac{mN}{8})W_N^{(8k+p)(\frac{mN}{8}+n)}\\&=\sum_{n=0}^{(N/8)-1}[x(n)+x(n+\frac{N}{8})W_8^p+x(n+\frac{2N}{8})W_8^{2p}\\&\quad+x(n+\frac{3N}{8})W_8^{3p}+x(n+\frac{4N}{8})W_8^{4p}+x(n+\frac{5N}{8})W_8^{5p}\\&\quad+x(n+\frac{6N}{8})W_8^{6p}+x(n+\frac{7N}{8})W_8^{7p}]W_N^{np}W_{N/8}^{nk}.\end{aligned}$$

$$k=0,\ldots(N/8)-1; p=0,\ldots 7. \quad (9)$$

And the coefficient has characteristics as shown in (10).

$$W_N^{kN/4}=(-j)^k, W_N^{kN/2}=(-1)^k, W_N^{3kN/4}=(j)^k \quad (10)$$

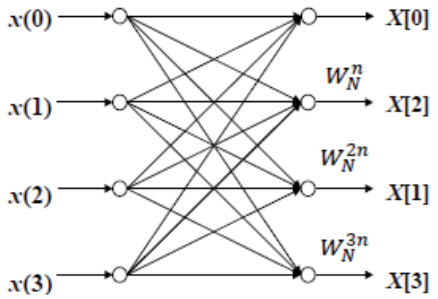

Fig. 3. The Radix-4 FFT butterfly.

Since these properties, the rdix-2 DFT formula and the radix-4 DFT formula can be simplified. But the coefficients of $W_N^{kN/8}$, $W_N^{3kN/8}$, $W_N^{5kN/8}$ and $W_N^{7kN/8}$ are completely complex valued data. The formulas like the radix-2 FFT and the radix-4 FFT are not easy to get from simplification. Therefore, thinking from another perspective, the definition of $W_N^{kN/8}$ in formula (11) is used.

$$W_N^{kn} = e^{-j\frac{2\pi nk}{N}} \quad (11)$$

The Euler's formula is shown in formula (12).

$$e^{-j\frac{2\pi nk}{N}} = \cos(\frac{2\pi nk}{N}) - j\sin(\frac{2\pi nk}{N}) \quad (12)$$

So there is a formula as shown in (13).

$$W_N^{kn} = \cos(\frac{2\pi nk}{N}) - j\sin(\frac{2\pi nk}{N}) \quad (13)$$

3.2. The Implementation of the Algorithm

Using the definition of the coefficient, take the equation (13) into the formula of the radix-8 FFT. The radix-8 FFT algorithm can be implemented by complex multiplication, addition and subtraction. The pseudo-code of the radix-8 FFT algorithm is shown as follows.

```
void FFT(complex* v)
{
    Copy node
    Initialize the next node to 0
    Set W the complex number
    Express the Eq.13
    Calculate X1+WX2,X1-WX2
}
```

The rest is similar to the radix-2 FFT algorithm. It cuts the discrete signal sequence which the length is N for eight sequences to operate using the way of handing back and division. Continue dividing until each subsequence has only two points.

Each division is a level with a set of operations. N points of DFT can be divided for M times. And every level of operation has $N/2^m$ groups. Thus, two loops can be used.

In these loops there is a function to implement each set of butterfly operations. As the input of the butterfly is the natural order and the output is bit-inverted, there is a sorting process in the function. After the completion of each set of butterfly, the output is put to the position where the next level input is, and this is equivalent to rearrange each level. The pseudo-code of the butterfly function of the radix-8 FFT algorithm is shown as follows.

```
void FftIteration(int j, int N, int R, int Ns,
complex* data0, complex*data1)
{
    Define the array subscript parameters
    Calculate rotating factor
    FFT function
    Rearrange array
}
```

3.3. The FFT Algorithm Promotion

The implementation of the radix-8 FFT algorithm uses the definition of the coefficient factor and the Euler's formula which all can be common used. The DFT formula is also universal. So it doesn't need to simplify the formula like the radix-2 FFT algorithm and the radix-4 FFT algorithm. The following sentence of the template is to implement the function of radix-2^n FFT algorithm.

```
template <int R>
```

Keep the framework of the FFT algorithm and replace 8 in the radix-8 FFT algorithm with R(R=2^n). Because the basic idea doesn't change, the only one needs to be changed is the radix. Then, a series of fast Fourier transform of radix 2^n can be realized, such as radix-16 FFT, radix-32 FFT and so on. Thus, the FFT algorithm module can be implemented and promoted.

4. Experimental Design

4.1. Theoretical Analysis

Assume that an N-point signal sequence needs to be transformed through the radix-2^n FFT algorithm. When the N is large enough, the bigger the radix number 2^n is, the amount of the computation of the FFT algorithm is smaller, and run faster. Now analyze the running speed in theory. The operation speed can be estimated from the amount of computation. The calculated amount of different radix FFT algorithm is shown in the Table 3.

Of course, the amount of computation drawn here are theoretical values, roughly calculating through the mathematical way so as to estimate the speed of the running time of the algorithm processing. Through the actual operation, the results will be test below.

Table 3. The calculated amount of different radix FFT.

	Complex multiplication times
radix-2FFT	$\frac{N}{2}\log_2 N$
radix-4FFT	$\frac{3N}{4}\log_4 N = \frac{3N}{8}\log_2 N$
radix-8FFT	$\frac{7N}{8}\log_8 N = \frac{7N}{24}\log_2 N$

4.2. Experimental Design

The hardware test environment of the design is on the Intel® Core™ i3550CPU at 3.2 GHz computer through the Microsoft Visual Studio 2008 software environmental platform.

In order to test the performance of the algorithm, adding a function of time in the main function module to test the run time. Use the function of time coming with the system, clock (), to get the system time before and after implementation of the FFT algorithm. The value after subtraction of these two is the system time of each calculation. Then divide the system time per second, CLOCKS_PER_SEC, to get the time in seconds for each running.

Now choose the appropriate input data to run the FFT algorithm. As the FFT algorithm is aiming at implement large amounts of data quickly, here a lot of data are also needed to have a test. However, manually entering large amounts of data to test takes too much time and doesn't conform to the actual situation. Generally read data in the file or input data automatically by the system. Here in order to facilitate the testing, only emphasizes the correctness of the algorithm and time performance, it specifies the input data directly through the cycle. The input is a sequence of complex numbers. Through the structures define the input data of the algorithm. Here specify that the real part is the odd from 1 and the imaginary part is the natural number from 1.

The experiment is designed to test the high-speed performance of the radix-8 FFT algorithm. Here just take the radix-2 FFT algorithm as a reference for easy setting N. Because the N must be the power of 2 and the power of 8.The N starts from 8 to be tested here. By a large number of data testing, the running speed comparison between the radix-2 FFT algorithm and the radix-8 FFT algorithm is shown in Table 4.

Table 4. The running speed of radix-2and radix-8 FFT.

N	2^3=8	2^6=64	2^9=512	2^{12}=4096	2^{15}=32768	2^{18}=262144	2^{21}=2097152
Radix-2FFT	0s	0s	0.007s	0.033s	0.353s	2.944s	25.486s
Radix-8FFT	0s	0s	0.005s	0.021s	0.220s	2.029s	19.668s
Speed difference	0s	0s	0.002s	0.012s	0.133s	0.915s	5.818s

It is easy to see that the radix-8 FFT takes less time when the N is the same. That is to say, the radix-8 is faster than the radix-2 FFT algorithm.

When the N is very small, such as 8 or 64 points, the running time can't be recorded by the computer. Because the time is too short even approximate to zero.

The bigger the N is, the speed difference can be more obvious, and it is easier to see the good performance of the radix-8.

In order to see the speed difference between the radix-2 FFT and the radix-8 FFT much more clearly, the line chart is better, as shown in Fig. 4.

5. Conclusions and Future Work

1) When the N is large enough, the bigger the 2n is, the calculated amount of the FFT algorithm is smaller. Starting the study from the radix-2 and the radix-4 fast Fourier transform, implement them and put forward the more efficient and faster radix-8 FFT algorithm and finally extend to radix any power of 2 FFT algorithm, such as radix-16, radix-32. Through the experiments, the radix-8 FFT algorithm proves its faster and more efficient performance than the radix-2 from the fact.

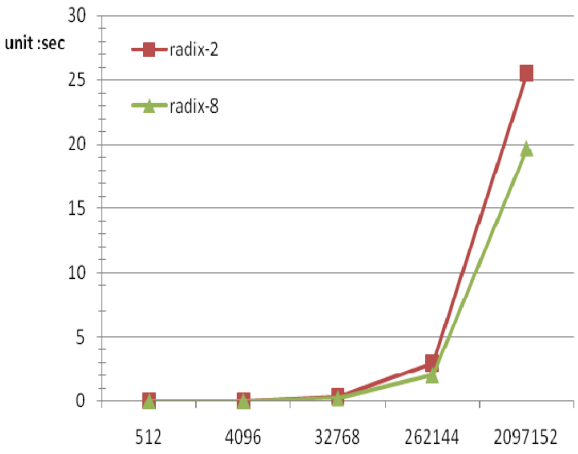

Fig. 4. The running speed of the radix-2 FFT algorithm and the radix-8 FFT algorithm.

2) The amount of data that the radix-2n FFT can transform is just the power of 2n. If the amount of data doesn't comply with this condition, artificially add the number of zero. So according to the actual situation, consider a decision to choose n as the most suitable number.

3) The further work will run this algorithm on the GPU. The CPU is not fast enough and strong enough to run even more data. It takes a long time when the N is reach two million as shown in the Table 4.The GPU is a faster device that can popularize this algorithm.

References

[1]. Mitra Sanjit K., Digital Signal Processing - A Computer-Based Approach, 2nd ed. [S.l.], *McGraw-Hill Companies Inc,* 2001.

[2]. Chi Lu, Jinfeng Hu, Qingsheng Ding, Based on DSP High Precision Radar Signal Acquisition and FFT Implementation, *Technological Development of Enterprise (Academic Edition)*, Vol. 29, No. 6, 2010, pp. 20-21, (in Chinese).

[3]. Ronald Newbold Bracewell, Fourier Transform and Its Applications, *McGraw Hill,* 1986.

[4]. Steven W. Smith, Ph. D. The Scientist and Engineer's Guide to Digital Signal Processing. http://www.dspguide.com/pdfbook.htm, 2013-03-28.

[5]. Serbs Ahmet, Durak-Ata Lutfiye, The discrete fractional Fourier transform based on the DFT matrix, *Signal Processing*, Vol. 91, No. 3, 2011, pp. 571-581.

[6]. Hongxia Liu, Liang Yang, Jin Huang, Shitan Huang, Variable length FFT parallel rotating factor efficient generation algorithm and realization, *Journal of Xi'an University of Electronic Technology,* Vol. 36, No. 3, 2009.

[7]. B. Boashash, Time-Frequency Signal Analysis and Processing–A Comprehensive Reference, *Elsevier Science,* Oxford, 2003.

[8]. Burrus and Parks, DFT/FFT and Convolution Algorithms, *John Wiley & Sons*, New York, 1985.

[9]. C. V. Loan, Computational Frameworks for the Fast Fourier Transform, *Society for Industrial Mathematics,* 1992.

[10]. Takahashi, Daisuke, An implementation of parallel 2-D FFT using Intel AVX instructions on multi-core processors, *Lecture Notes in Computer Science*, Vol. 7440 LNCS, Part 2, 2012, pp. 197-205.

[11]. Baxley Robert J, Zhou G. Tong, Computational complexity analysis of FFT pruning-A Markov modeling approach, in *Proceedings of the IEEE 12th Digital Signal Processing Workshop*, Vol. 1 and 2, 2006, pp. 535-539.

[12]. Pierre Duhamel, Algorithms meeting the lower bounds on the multiplicative complexity of length-2n DFTs and their connection with practical algorithms, *IEEE Trans Acoust. Speech. Sig. Proc,* Vol. 38, 1990, pp. 1504-1511.

[13]. Tang, Song-Nien, Liao, Chi-Hsiang, Chang, Tsin-Yuan, An area- and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems, *IEEE Journal of Solid-State Circuits,* Vol. 47, No. 6, 2012, pp. 1419-1435.

[14]. Lobeiras, J., Amor, M., Doallo, R., FFT implementation on a streaming architecture, in *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed, and Network-Based Processing (PDP'11)* , 2011, pp. 119-126.

[15]. Jiyang Yu, Yang Li, Dan Huang, A Method on Vector Base 2×2 Two-dimensional FFT Efficient Structure(In Chinese), *Journal of Beijing University of Science and Technology,* Vol. 31, No. 8, 2011, pp. 962-965, 1004.

[16]. N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith and J. Manferdelli, High Performance Discrete Fourier Transforms on Graphics Processors, in *Proceedings of the Supercomputing*, November 2008, pp. 1–12.