Radix-3 FFT에 관한 고찰

정혜승*

Study of Radix-3 FFT

Haeseung Jung*

Abstract

Fast Fourier Transform is the fast implementation of Discrete Fourier Transform, which deletes periodic operation of DFT. According to the definition, radix-2 FFT can be implemented by recursive call which divides the input signal points into 2 signal points. Because of its time-consuming stack-copy operation, this recursive method is very slow. To overcome this drawback, butterfly operation with signal rearrangement was devised. Based on the ideas of signal rearrangement and butterfly operation, this paper applies the signal rearrangement method to the Radix-3 FFT and checks the validity of this method.

초 록

고속푸리에변환(Fast Fourier Transform)은 이산푸리에변환(Discrete Fourier Transform)의 주기적으로 반복되는 연산을 생략하여 그 속도를 향상시킨 연산방법이다. Radix-2 FFT는 그 정의에 따라 함수 재귀호출에 의해 구현될 수 있는데 이 방법은 스택복사 과정의 시간소모 때문에 고속동작이 어렵게 된다. 이를 극복하기 위해 신호점을 연산순서에 맞게 미리 재배열하고 배열된 신호점을 나비연산하는 방법으로 고속연산을 구현할 수 있다. 이 논문은 신호점 재배열 방법에 의한 Radix-2 FFT의 고속연산에 착안하여 Radix-3 FFT에 신호점 재배열 방식을 적용해 보고 그 타당성에 관해 고찰하였다.

키워드 : 푸리에변환(Fourier Transform), 고속푸리에변환(Fast Fourier Transform), Radix-2, Radix-3, Twiddle Factor, Bit-Reversing

1. 서 론

고속푸리에변환(Fast Fourier Transform)은 이산푸리에변환(Discrete Fourier Transform)의 주기적으로 반복되는 연산을 생략하여 그 속도를 향상시킨 연산방법이다. 고속푸리에변환은 그 정의에 따라 신호점을 나누고 연산하는 재귀호출에 의해 구현될 수 있는데, 이 방법은 재귀호출에 의해 발생하는 스택복사 과정의 시간소모 때문에 고속동작이 어렵게 된다. 이를 극복하기 위해 신

접수일(2009년12월21일), 수정일(1차 : 2010년 5월 14일, 2차 : 2010년 6월 11일, 게재 확정일 : 2010년 7월 1일)

* 전자팀/hsjung@kari.re.kr

호점을 연산순서에 맞게 미리 재배열하고 배열된 신호점을 나비연산하는 방법으로 고속연산을 수행할 수 있게 된다. 신호점 재배열 방법은 비트역전에 의해 구현될 수 있는데, 이는 신호점을 2의 거듭제곱 수로 나누는 경우에만 해당된다. 이 논문은 Radix-2 고속푸리에변환 방법에 관해 살펴보고, 재귀호출에 의한 Radix-3 고속푸리에변환법 및 신호점 재배열 방법에 관해 고찰한다.

2. Radix-2 FFT

2.1 Radix-2 FFT의 정의

이산푸리에변환의 정의식은 아래와 같다.

$$X[k]=\sum_{n=0}^{N-1}x[n]e^{-j\frac{2\pi}{N}kn} \tag{1}$$

식 (1)에서 $e^{-j2\pi kn/N}$ 는 복소평면에서 kn값이 증가함에 따라 N단계에 걸쳐 시계방향으로 회전하게 되는데, 이 특징에 기인해 이를 회전자(twiddle factor)라 부르고, W_N^{kn}으로 쓴다.

Radix-2 FFT는 식 (1)을 짝수 번째 수열과 홀수 번째 수열로 나누어 연산의 수를 줄인 것이다.

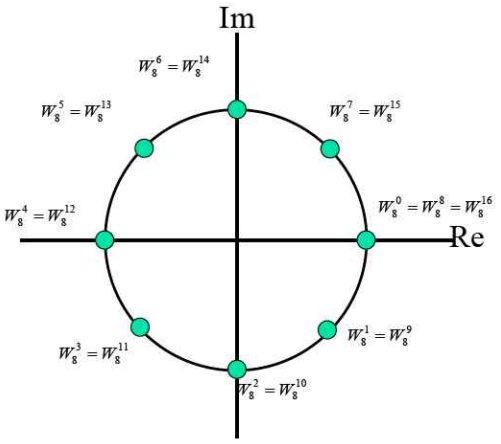

그림 1. 복소평면에서 N=8인 회전자의 주기성

$$\begin{aligned}X[k]&=\sum_{n=0}^{N-1}x[n]W_N^{kn}\\&=\sum_{n:even}x[n]W_N^{kn}+\sum_{n:odd}x[n]W_N^{kn}\\&=\sum_{m=0}^{(N/2)-1}x[2m]W_N^{2mk}+\sum_{m=0}^{(N/2)-1}x[2m+1]W_N^{(2m+1)k}\\&=\sum_{m=0}^{(N/2)-1}x[2m]W_N^{2mk}+W_N^k\sum_{m=0}^{(N/2)-1}x[2m+1]W_N^{2mk}\end{aligned} \tag{2}$$

식 (2)에서, W_N^2는

$$W_N^2=e^{-j\frac{2\pi}{N}\cdot 2}=e^{-j\frac{2\pi}{N/2}}=W_{N/2}^1 \tag{3}$$

이므로 $X[k]$는 아래와 같이 쓸 수 있다.

$$\begin{aligned}X[k]=&\sum_{m=0}^{(N/2)-1}x[2m]W_{N/2}^{mk}\\&+W_N^k\sum_{n:odd}^{(N/2)-1}x[2m+1]W_{N/2}^{mk}\end{aligned} \tag{4}$$

식 (4)를 살펴보면, 이산 푸리에 변환은 결국 짝수 번째 수열의 이산 푸리에 변환과 홀수 번째 수열의 이산 푸리에 변환으로 각각 나누어짐을 알 수 있다. 식 (4)의 두 항을,

$$A[k]=\sum_{m=0}^{(N/2)-1}x[2m]W_{N/2}^{mk},0\le k\le N-1 \tag{5}$$

$$B[k]=\sum_{m=0}^{(N/2)-1}x[2m+1]W_{N/2}^{mk},0\le k\le N-1 \tag{6}$$

로 두면, $X[k]$는

$$X[k]=A[k]+W_N^kB[k] \tag{7}$$

으로 쓸 수 있다. 여기에서 W_N^k는 주기성을 가지는데, 특히 k가 $N/2$ 이상일 때,

$$\begin{aligned}W_N^{k+\frac{N}{2}}&=e^{-j\frac{2\pi}{N}(k+\frac{N}{2})}=e^{-j\frac{2\pi}{N}k}\cdot e^{-j\frac{2\pi}{N}\frac{N}{2}}\\&=W_N^k\cdot e^{-j\pi}=W_N^k\cdot(\cos\pi-j\sin\pi)\\&=-W_N^k\end{aligned} \tag{8}$$

임을 알 수 있다. 이는 복소평면에서 W_N^k이 X 축 대칭이라는 것과 같은 의미이다. 따라서 $k \geq N/2$인 구간에서 $X[k]$는,

$$X[k] = A[k] - W_N^k B[k] \quad (9)$$

이 된다. 즉, $X[k]$를 계산할 때 $k \geq N/2$인 구간은 $k < N/2$ 구간의 연산결과를 이용하여 간단하게 구할 수 있음을 알 수 있다.

이산 푸리에 변환의 역변환은

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn} = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, W_N^{-kn} \quad (10)$$

이다. 두 복소수의 곱의 켤레복소수는 각 복소수의 켤레복소수의 곱과 같으므로,

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, W_N^{-kn} = \left(\frac{1}{N}\sum_{k=0}^{N-1} X^*[k]\, W_N^{kn}\right)^* \quad (11)$$

이 된다. $x[n]$은 실수이므로,

$$x[n] = \left(\frac{1}{N}\sum_{k=0}^{N-1} X^*[k]\, W_N^{kn}\right)^* = \frac{1}{N}\sum_{k=0}^{N-1} X^*[k]\, W_N^{kn} \quad (12)$$

이다. 즉, 역 이산 푸리에 변환은 $X[k]$의 켤레복소수를 이산 푸리에 변환한 것과 같은 결과가 나타남을 알 수 있다.

2.2 신호점 재배열을 이용한 Radix-2 FFT

앞 절에서 설명한 재귀호출을 이용한 Radix-2 FFT는 재귀호출 시 발생하는 스택복사과정의 시간소모에 의해 고속동작이 어렵게 된다. 이를 극복하기 위해 신호점을 미리 연산순서에 맞게 재배열하고, 재배열된 신호점을 나비연산하여 고속동작을 구현할 수 있다.

신호점의 수가 8인 신호를 Radix-2 FFT 함수의 재귀호출에 의해 신호점의 수가 2일 때까지 분해한 다음 2점 DFT가 수행되는 신호점의 수열을 나열하면 아래와 같다.

표 1. N=8인 신호의 Radix-2 FFT 연산 순서

입력 신호	2점 DFT시 연산 순서
x[0]	x[0]
x[1]	x[4]
x[2]	x[2]
x[3]	x[6]
x[4]	x[1]
x[5]	x[5]
x[6]	x[3]
x[7]	x[7]

함수의 재귀호출을 사용하지 않고 FFT를 수행하기 위해서는 먼저 입력 신호의 순서를 위 표와 같이 재배치하여야 함을 알 수 있다. 위 표의 입력 신호 순서를 2진 데이터로 표시하면 아래 표처럼 입력신호 주소의 각 비트를 역전한 것과 같음을 알 수 있다.

표 2. 비트 역전에 의해 재배열된 신호 x[n]

입력 신호	이진 주소	비트 역전된 주소	2점 DFT시 연산 순서
x[0]	000	000	x[0]
x[1]	001	100	x[4]
x[2]	010	010	x[2]
x[3]	011	110	x[6]
x[4]	100	001	x[1]
x[5]	101	101	x[5]
x[6]	110	011	x[3]
x[7]	111	111	x[7]

이러한 경향은 2의 거듭제곱인 모든 정수 N에 대해 적용된다. 구현된 Radix-2 비트역전 코드는 아래 그림과 같다. 이렇게 입력된 시간 영역의 신호를 Radix에 따라 재배열하는 DIT (Decimation-In-Time) 방법 외에 나비연산을 먼저 수행한 후 주파수 영역의 신호를 재배열하는 DIF (Decimation-In-Frequency)도 같은 결과를 출력하는데, 연산방식과 연산속도는 같으므로 여기서는 DIT로 연산을 구현하였다.

```
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < exp; j++)
    {
        int mask = 1 << j;
        int shift_count = exp - ((j << 1) + 1);

        if (shift_count >= 0)
        {
            addr[i] += (i & mask) << shift_count;
        }
        else
        {
            addr[i] += (i & mask) >> -shift_count;
        }
    }
}

for (int i = 0; i < N; i++)
{
    result.Points.Add(input.Points[addr[i]]);
}
```

그림 2. Radix-2 비트역전 코드

2.2 Radix-2 FFT 나비연산

비트역전에 의해 입력 신호를 재배열한 후 나비연산이 수행된다. 나비연산이라 함은 아래 그림과 같이 그 연산의 흐름이 나비모양과 닮았다 해서 붙여진 것이다. Radix-2 FFT는

$$X[k] = A[k] + W_N^k B[k], \qquad 0 \le k < \frac{N}{2} \qquad (13)$$
$$X[k+\frac{N}{2}] = A[k] - W_N^k B[k], \quad \frac{N}{2} \le k < N$$

이므로, N=8일 때 X[k]는

$$\begin{aligned} X[0] &= A[0] + W_8^0 B[0] \quad X[4] = A[0] - W_8^0 B[0] \\ X[1] &= A[1] + W_8^1 B[1] \quad X[5] = A[1] - W_8^1 B[1] \\ X[2] &= A[2] + W_8^2 B[2] \quad X[6] = A[2] - W_8^2 B[2] \\ X[3] &= A[3] + W_8^3 B[3] \quad X[7] = A[3] - W_8^3 B[3] \end{aligned} \qquad (14)$$

이 된다. 이 연산을 흐름도로 표현하면,

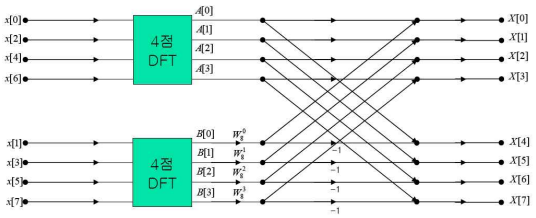

그림 3. N=8인 FFT 연산 흐름도

그림 3과 같다. 4점 DFT는 다시 그림 4와 같이 표현되고,

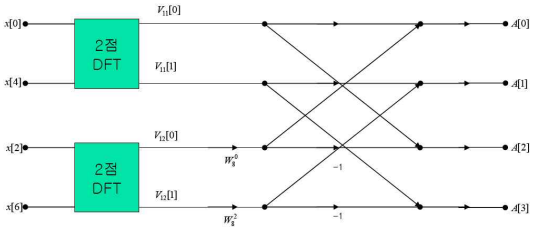

그림 4. 4점 DFT의 연산 흐름도

2점 DFT도 다시 그림 5와 같이 나타낼 수 있으므로,

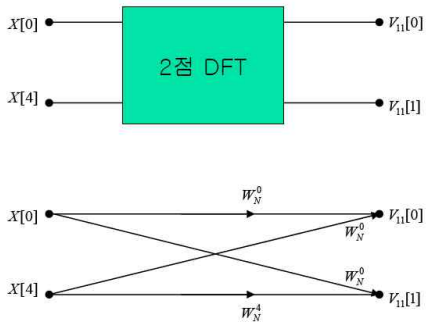

그림 5. 2점 DFT의 연산 흐름도

결국 전체 흐름도는 그림 6과 같다.

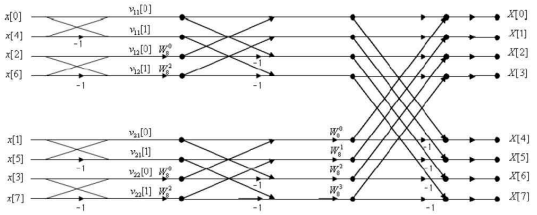

그림 6. 8점 FFT의 연산 흐름도

8점 FFT 의 경우 위 그림과 같이 $\log_2 8 = 3$ 번의 나비연산 과정을 거치게 된다. 구현을 위해 주목할 사항은 회전자(twiddle factor) 연산에 걸리는 시간 단축을 위해 반복되는 삼각함수 연산 결과는 메모리에 배열로 저장해 두는 것이다. 위 흐름도를 이용하여 구현된 나비연산 코드는 그림 7과 같다.

2.3 Radix-2 연산속도 비교

신호점의 수에 따라 DFT와 재귀호출을 사용한 Radix-2 FFT, 재귀호출을 사용하지 않는 Radix-2 FFT의 연산속도를 비교하여 표 3에 나타내었다. (Intel Core2 Duo CPU E6750 @ 2.26GHz PC 사용)

```
double[] temp = new double[2];

for (int i = 0; i < exp; i++)
{
    for (int k = 0; k < N; k += (2 << i))
    {
        for (int j = 0; j < (1 << i); j++)
        {
            double[] a = new double[2];
            double[] b = new double[2];
            a[0] = result.Points[k + j].YValues[0];
            a[1] = result.Points[k + j].YValues[1];
            b[0] = result.Points[k + j + (1 << i)].YValues[0];
            b[1] = result.Points[k + j + (1 << i)].YValues[1];

            // X[k]=a[k]+Wb[k]
            // W = cos(2*pi*k/N)-jsin(2*pi*k/N)
            // Wb[k] = (b[0]*cos + b[1]*sin) + j(b[1]*cos - b[0]*sin)
            double[] Wb = new double[2];
            Wb[0] = b[0] * COS[j << (exp - 1 - i)] + b[1] * SIN[j << (exp - 1 - i)];
            Wb[1] = b[1] * COS[j << (exp - 1 - i)] - b[0] * SIN[j << (exp - 1 - i)];

            temp[0] = a[0] + Wb[0];
            temp[1] = a[1] + Wb[1];
            result.Points[k + j].YValues[0] = temp[0];
            result.Points[k + j].YValues[1] = temp[1];

            // X[k+N/2]=a[k]-Wb[k]
            temp[0] = a[0] - Wb[0];
            temp[1] = a[1] - Wb[1];
            result.Points[k + j + (1 << i)].YValues[0] = temp[0];
            result.Points[k + j + (1 << i)].YValues[1] = temp[1];
        }
    }
}

return result;
```

그림 7. 구현된 Radix-2 나비연산 코드

표 3. 신호점의 수 N에 따른 각 알고리즘별 연산 시간

N	연산 시간 (ms)		
	DFT	Recursive FFT	Non-Recursive FFT
2	0.00031	0.02750	0.02578
4	0.00187	0.18156	0.02953
8	0.00718	0.5347	0.04000
16	0.031	1.218	0.047
32	**0.110**	2.657	**0.093**
64	0.46	5.62	0.203
128	1.89	11.625	0.468
256	7.657	25.79	1.063
512	30.78	53.12	2.406
1024	**122.97**	**111.40**	5.469
2048	493.7	254.7	12.172
4096	1984.4	517.2	22.812
8192	7844	1110	40.417

위 결과를 살펴보면 함수 재귀호출을 사용하는 Recursive FFT의 경우 재귀호출이 발생하는 N=4 인 신호의 연산시간이 재귀호출이 발생하지 않는 N=2 인 신호의 연산시간보다 무려 6.6 배 이상 소요되는 것으로 나타났다. 이는 재귀호출이 상당히 큰 연산시간을 소모한다는 사실을 의미하는 것이다. 한편, 재귀호출을 사용하지 않는 Non-Recursive FFT의 경우 N=32 미만에서 DFT보다 느린 속도를 보이는데, 이는 비트 역전 방법으로 신호점을 재배열하는데 소요되는 연산시간 때문이다. Recursive FFT의 경우 N=1024이상이 되어서야 DFT 보다 빠른 연산속도를 보임을 알 수 있다. 이것은 DFT는 N의 수가 증가함에 따라 곱셈횟수가 N의 제곱번 증가하는 반면, FFT의 경우 2N번 증가하기 때문이다. N=8192일 때 DFT와 Non-Recursive FFT의 연산속도는 무려 194배 임을 실험 결과로 알 수 있다.

3. Radix-3 FFT

3.1 Radix-3 FFT의 정의

신호점의 수를 $N=2^m$ (m은 정수)으로 두면, N은 아래와 같이 나누어 질 수 있다.

$$\begin{aligned} N=2^m &= 2 \cdot 2^{m-1} = 2^{m-1}+2^{m-1} \\ &= 2 \cdot 2^{m-2}+2 \cdot 2^{m-2} \\ &= (2^{m-2}+2^{m-2})+(2^{m-2}+2^{m-2}) \\ &= \underbrace{(2^0+2^0)+\cdots+(2^0+2^0)}_{2^{m-1}\text{개 항}} \end{aligned} \tag{15}$$

위 수열처럼 Radix-2 FFT는 입력 신호를 2의 거듭제곱수로 두고, 이를 짝수 수열($n=2m$)과 홀수 수열($n=2m+1$)로 나누고, 나뉜 각 수열을 다시 Radix-2 FFT하여 전체 결과 값을 구하는 방법이다. 마찬가지로 Radix-3 FFT는 신호점의 수를 3의 거듭제곱수로 두고, 이를 첫 번째 수열($n=3m$), 두 번째 수열($n=3m+1$)과 세 번째 수열($n=3m+2$)로 나누어 이산 푸리에 변환식을 3개의 이산 푸리에 변환식으로 나누어 계산하는 방법이다. 이처럼 3의 거듭제곱수의 신

호점을 3분하여 연산하므로 이를 Radix-3 FFT라 고 부른다.

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n]\, W_N^{kn} \qquad (16)\\
&= \sum_{m=0}^{\frac{N}{3}-1} x[3m]\, W_N^{3mk} + \sum_{m=0}^{\frac{N}{3}-1} x[3m+1]\, W_N^{(3m+1)k}\\
&\quad + \sum_{m=0}^{\frac{N}{3}-1} x[3m+2]\, W_N^{(3m+2)k}\\
&= \sum_{m=0}^{\frac{N}{3}-1} x[3m]\, W_N^{3mk} + W_N^{k} \sum_{m=0}^{\frac{N}{3}-1} x[3m+1]\, W_N^{3mk}\\
&\quad + W_N^{2k} \sum_{m=0}^{\frac{N}{3}-1} x[3m+2]\, W_N^{3mk}
\end{aligned}
$$

식 (16)에서

$$W_N^3 = e^{-j\frac{2\pi}{N}3} = e^{-j\frac{2\pi}{N/3}} = W_{N/3} \qquad (17)$$

이므로, $X[k]$는

$$
\begin{aligned}
X[k] &= \sum_{m=0}^{\frac{N}{3}-1} x[3m]\, W_{N/3}^{mk} + W_N^{k} \sum_{m=0}^{\frac{N}{3}-1} x[3m+1]\, W_{N/3}^{mk} \qquad (18)\\
&\quad + W_N^{2k} \sum_{m=0}^{\frac{N}{3}-1} x[3m+2]\, W_{N/3}^{mk}
\end{aligned}
$$

으로 쓸 수 있다. 위 식을 살펴보면 이산 푸리에 변환은 결국 세 개의 이산 푸리에 변환으로 각각 나누어짐을 알 수 있다. 위 식의 각 항을

$$A[k] = \sum_{m=0}^{\frac{N}{3}-1} x[3m]\, W_{N/3}^{mk}, 0 \le k < N \qquad (19)$$

$$B[k] = \sum_{m=0}^{\frac{N}{3}-1} x[3m+1]\, W_{N/3}^{mk}, 0 \le k < N \qquad (20)$$

$$C[k] = \sum_{m=0}^{\frac{N}{3}-1} x[3m+2]\, W_{N/3}^{mk}, 0 \le k < N \qquad (21)$$

로 두면, $X[k]$는

$$X[k] = A[k] + W_N^{k} B[k] + W_N^{2k} C[k] \qquad (22)$$

로 쓸 수 있다. W_N^k의 주기성을 이용하면,

$$
\begin{aligned}
W_N^{k+\frac{N}{3}} &= W_N^{k} \cdot e^{-j\frac{2\pi}{N}\frac{N}{3}} = W_N^{k} \cdot e^{-j\frac{2\pi}{3}} \qquad (23)\\
&= W_N^{k} \cdot (\cos\frac{2\pi}{3} - j\sin\frac{2\pi}{3})\\
&= (-\frac{1}{2} - j\frac{\sqrt{3}}{2})\, W_N^{k}
\end{aligned}
$$

$$
\begin{aligned}
W_N^{2(k+\frac{N}{3})} &= W_N^{2k} \cdot e^{-j\frac{2\pi}{N}\frac{2N}{3}} = W_N^{2k} \cdot e^{-j\frac{4\pi}{3}} \qquad (24)\\
&= W_N^{2k} \cdot (\cos\frac{4\pi}{3} - j\sin\frac{4\pi}{3})\\
&= (-\frac{1}{2} + j\frac{\sqrt{3}}{2})\, W_N^{2k}
\end{aligned}
$$

$$
\begin{aligned}
W_N^{k+\frac{2N}{3}} &= W_N^{k} \cdot e^{-j\frac{2\pi}{N}\frac{2N}{3}} = W_N^{k} \cdot e^{-j\frac{4\pi}{3}} \qquad (25)\\
&= W_N^{k} \cdot (\cos\frac{4\pi}{3} - j\sin\frac{4\pi}{3})\\
&= (-\frac{1}{2} + j\frac{\sqrt{3}}{2})\, W_N^{k}
\end{aligned}
$$

$$
\begin{aligned}
W_N^{2(k+\frac{2N}{3})} &= W_N^{2k} \cdot e^{-j\frac{2\pi}{N}\frac{4N}{3}} = W_N^{2k} \cdot e^{-j\frac{8\pi}{3}} \qquad (26)\\
&= W_N^{2k} \cdot (\cos\frac{8\pi}{3} - j\sin\frac{8\pi}{3})\\
&= (-\frac{1}{2} - j\frac{\sqrt{3}}{2})\, W_N^{2k}
\end{aligned}
$$

이므로, $X[k]$는

$$
\begin{aligned}
X[k] &= A[k] + W_N^{k} B[k] + W_N^{2k} C[k] \qquad (27)\\
&\quad 0 \le k < \frac{N}{3}
\end{aligned}
$$

$$
\begin{aligned}
X[k] &= A[k] + (-\frac{1}{2} - j\frac{\sqrt{3}}{2})\, W_N^{k} B[k] \qquad (28)\\
&\quad + (-\frac{1}{2} + j\frac{\sqrt{3}}{2})\, W_N^{2k} C[k]\\
&\quad \frac{N}{3} \le k < \frac{2N}{3}
\end{aligned}
$$

$$
\begin{aligned}
X[k] &= A[k] + (-\frac{1}{2} + j\frac{\sqrt{3}}{2})\, W_N^{k} B[k] \qquad (29)\\
&\quad + (-\frac{1}{2} - j\frac{\sqrt{3}}{2})\, W_N^{2k} C[k]\\
&\quad \frac{2N}{3} \le k < N
\end{aligned}
$$

로 나눌 수 있다. 즉, N점 DFT를 $0 \le k < N/3$의 연산만으로 완료할 수 있다는 의미이다.

3.3 신호점 재배열을 이용한 Radix-3 FFT

이 절은 Radix-2 FFT와 마찬가지로 나비연산을 수행하기 위해 Radix-3 FFT 신호점 재배열 방법 중 한가지를 제안하고자 한다. 그 구현과정에서 부가되는 진법변환 연산에 의해 성능은 만족스럽지 않지만 그 타당성은 확인할 수 있다.

Radix-3 FFT를 위해 입력 신호를 재배열하는 방법은 Radix-2처럼 간단한 2진수 비트 역전으로 구현되지 않는다. 그러나 신호점의 주소를 3진수로 표기하면 마치 2진수 비트역전과 같이 3진수 자리 역전에 의해 신호점 재배열이 가능함을 발견하였다. 아래 표는 신호점의 수가 9인 신호를 3진수 자리 역전에 의해 재배열하는 과정을 나타낸 것이다.

표 4. 3진수 자리 역전에 의한 신호 재배열

입력신호 주소		재배열 신호 주소	
10진수	3진수	3진수	10진수
0	00	00	0
1	01	10	3
2	02	20	6
3	10	01	1
4	11	11	4
5	12	21	7
6	20	02	2
7	21	12	5
8	22	22	8

3진수 자리 역전을 하기 위해서는 먼저 10진수 주소를 3진수로 바꾸고, 각 3진수 주소별로 자리를 역전하여 재배열 한 후 이를 다시 10진수 주소로 변환해야한다. 이 작업을 수행하는 코드는 그림 8과 같다.

```
for (int i = 1; i < N; i++)
{
    int j = 0;
    ad[j]++;
    while (ad[j++] >= 3)
    {
        ad[j]++;
        ad[j-1] = 0;
    }

    for (j = 0; j < m; j++)
    {
        addr[i] += ad[j] * (int)Math.Pow(3, m - 1 - j);
    }
```

그림 8. 3진수 자리 역전을 이용해 Radix-3 FFT용 입력 신호를 재배열하는 코드

시험 결과 이 방식은 10진수와 3진수간 진법변환에 나눗셈 연산이 여러 차례 수행되어 함수 재귀호출 방식보다 2배 이상 느린 속도를 나타내었다. 나비연산을 거친 결과는 DFT와 비교하여 동일한 값이 출력되어 그 타당성은 확인할 수 있었다.

4. 결 론

본 논문에서는 Radix-2 FFT에 대해서 살펴보고 DFT, 재귀호출을 사용한 Radix-2 FFT, 신호점 재배열과 나비연산을 사용한 Radix-2 FFT의 속도를 비교하여 최후자의 속도가 전자들보다 월등히 빠름을 알 수 있었다.

Radix-3 FFT의 경우 비트역전에 의한 신호점 재배열이 불가능하기 때문에 고속의 코드 구현에 어려움이 있다. 본 논문은 Radix-3 FFT를 위하여 입력 신호점의 주소를 3진수로 나타내고, 3진수 자리 역전에 의해 신호점 재배열이 가능함을 밝혔다. 진법 변환에 소요되는 나눗셈 연산시간에 의해 성능은 만족스럽지 않지만 그 타당성은 확인할 수 있었다. 향후 진법 변환에 관한 연구를 통해 성능을 향상시킬 예정이다.

참 고 문 헌

1. J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series“ Math. Comp., vol.

19, April 1965, pp, 297-301
2. Eric Dubois, Anastasios N. Venetsanopoulos, “A New Algorithm for the Radix-3 FFT“, IEEE Trans. acoustics, speech, and signal processing, VOL. ASSP-26, NO. 3, June 1978
3. R. C. Singleton, “An algorithm for computing the mixed radix fast Fourier transform,” IEEE Trans. Audio Electroacoust., vol. AU-17, pp. 93-103, June 1969.
4. Yoiti Suzuki, Toshio Sone, AND Ken’iti Kido, “A New FFT Algorithm of Radix 3, 6, and 12“, IEEE Trans. acoustics, speech, and signal processing, VOL. ASSP-34. NO. 2, APRIL 1986
5. 장영범, 허은성, 박진수, 홍대기, “OFDM용 고속 Radix-8 FFT 구조“, 전자공학회논문지-SP, 제44권 제5호, 2007.9, pp. 84~93
6. 서석호, 박세승, 이강현, “효율적인 FFT Radix-4의 구현“, 한국정보기술학회 2007년도 하계학술발표논문집 2007.6, pp. 249~252
7. 리우항, 이한호, “A High-Speed Low-Complexity 128/64-point Radix-2^4 FFT Processor for MIMO-OFDM Systems“, 전자공학회논문지-SD, 제46권 제2호 2009.2, pp. 15~23