die FFT

Rüdiger Knörig

20. November 2000

Inhaltsverzeichnis

1 Herleitung aus der DFT **3**
1.1 Notwendigkeit der FFT 3
1.2 der Grundgedanke der FFT 3
1.3 der Schluß zur FFT 4
1.4 Berechnung der inversen FFT 6
1.5 Gegenüberstellung DFT vs. FFT 7

2 Implementierung **8**
2.1 Berechnung der *bit reverse order* Sequenz 8
2.2 die eigentliche Transformation 9

1 Herleitung aus der DFT

1.1 Notwendigkeit der FFT

Signale können auf Digitalrechnern nur in diskreter Form verarbeitet werden; somit ist auch für die Transformation in den Frequenzbereich eine angepaßte Form der Fouriertransformation anzuwenden, die DFT. Nach der Transformationsgleichung der DFT (s. [5]),

$$X(jk\Delta\Omega) = \sum_{k=0}^{N-1} x(n) \cdot e^{-jkn\Delta\Omega} \text{ , mit } \Delta\Omega = \frac{2\pi}{N} \text{ und } k \in [0..N-1] \qquad (1)$$

sind für die Transformation eines Eingangsvektors der Länge N N^2 komplexe Multiplikationen und N^2 komplexe Additionen nötig. Bereits von Gauß wurde ein Rechenschema beschrieben, welches die Anzahl an zeitaufwendigen komplexen Rechenoperationen gerade bei den für die Praxis typischen Segmentlängen von $N > 64$ drastisch verringert.

1.2 der Grundgedanke der FFT

Um die Verständlichkeit der folgenden Ausführungen zu erhöhen, wird zunächst durch die Substitution

$$W_N = e^{-j\frac{2\pi}{N}} \Rightarrow X(k) = \sum_{k=0}^{N-1} x(n) \cdot W_N^{kn} \qquad (2)$$

Gleichung 1 in eine kompaktere Form überführt. Nimmt man nun an, daß die Segmentlänge eine 2er-Potenz[1] ist, so läßt sich die Folge der Eingangswerte in die Folge der geraden und ungeraden Indizies zerlegen. Da die Fouriertransformation ein linearer Operator ist, gilt

$$\begin{aligned} x(n) &= x(2l) + x(2l+1) \text{ , } l \in [0..\tfrac{N}{2}] && (3)\\ \Rightarrow X(k) &= F\{x(2l) + x(2l+1)\} && (4)\\ \Rightarrow X(k) &= F\{x(2l)\} + F\{x(2l+1)\} && (5) \end{aligned}$$

Setzt man für die Transformation Gleichung 2 ein, dann erhält man

$$\begin{aligned} X(k) &= \sum_{l=0}^{\frac{N}{2}-1} x(2l) \cdot W_N^{2lk} + \sum_{l=0}^{\frac{N}{2}-1} x(2l+1) \cdot W_N^{(2l+1)k} && (6)\\ X(k) &= \sum_{l=0}^{\frac{N}{2}-1} x(2l) \cdot W_N^{2lk} + \sum_{l=0}^{\frac{N}{2}-1} x(2l+1) \cdot W_N^{2lk} \cdot W_N^{k} && (7) \end{aligned}$$

[1] $N = 2^k, k = \{0, 1, 2, 3, ...\}$

Mit

$$W_N^{2lk} = e^{-j2lk\frac{2\pi}{N}} \quad (8)$$

$$= e^{-jlk\frac{2\pi}{\frac{N}{2}}} \quad (9)$$

$$= W_{\frac{N}{2}}^{lk} \quad (10)$$

gilt

$$X(k) = \underbrace{\sum_{l=0}^{\frac{N}{2}-1} x(2l) \cdot W_{\frac{N}{2}}^{lk}}_{\frac{N}{2}-DFT} + \underbrace{\left[\sum_{l=0}^{\frac{N}{2}-1} x(2l+1) \cdot W_{\frac{N}{2}}^{lk}\right]}_{\frac{N}{2}-DFT} \cdot W_N^k \quad (11)$$

Somit läßt sich die Transformation eines Eingangsvektors der Länge N auch als Summe zweier $\frac{N}{2}$ - DFT's schreiben, wobei die Transformationswerte der linken DFT noch mit einem Koeffizientenvektor W_N^k multipliziert werden müssen. Für die Berechnung der beiden DFT-Transformationen sind jeweils $2 \cdot \left(\frac{N}{2}\right)^2 = \frac{N^2}{2}$ komplexe Rechenoperationen nötig; bei der rechten Transformation in Gleichung 11 kommen noch N komplexe Multiplikation hinzu. Insgesamt werden jetzt nur noch $N^2 + N + 1$ komplexe Rechenoperationen statt der ursprünglichen $2N^2$ Rechenoperationen komplexer Natur benötigt.

1.3 der Schluß zur FFT

Auch die $\frac{N}{2}$-DFT's aus Gleichung 11 können nach dem gleichen Schema zerlegt werden, womit man letztendlich zu der eigentlichen FFT kommt. Als Beispiel soll hier eine FFT mit $N = 8$ gezeigt werden. Stellt man Gleichung 11 als Signalflußgraph dar (s. Darstellung 1), so erhält man einen oft als „Schmetterling“ oder „Butterfly“ bekannten Graphen. Für diese Graphen gilt die Konvention,

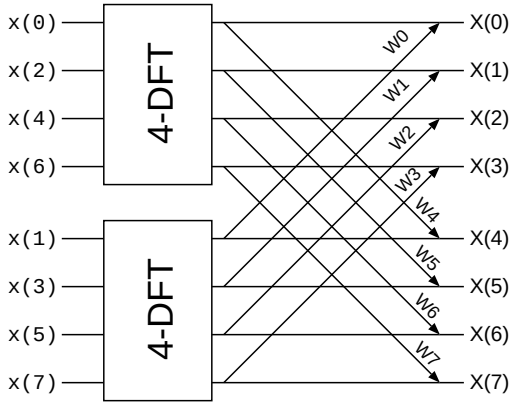

Abbildung 1: Butterfly-Darstellung von Gleichung 11, erste Form

daß zwei zusammengeführte Pfeile die Addition der repräsentierten Größen und

Zahlen über Pfeilen Multiplikatoren bedeuten. Die Berechnung kann noch weiter vereinfacht werden, wenn man die Beziehung

$$W_N^{k+\frac{N}{2}} = e^{-jk\frac{2\pi}{N}} \cdot \underbrace{e^{-j\frac{N}{2}\frac{2\pi}{N}}}_{=e^{-j\pi}=(-1)} \tag{12}$$

$$= -W_N^k \tag{13}$$

$$\Rightarrow W_8^4 = -W_8^0 \, , \, W_8^5 = -W_8^1 \tag{14}$$

berücksichtigt. Bei Verwendung eines Graphen nach Abbildung 2 braucht man nur $\frac{N}{2}$ Werte der DFT-Matrix zu kennen; zudem können $\frac{N}{2}$ komplexe Multiplikationen durch die einfache Multiplikation mit (-1) ersetzt werden. Nach diesem

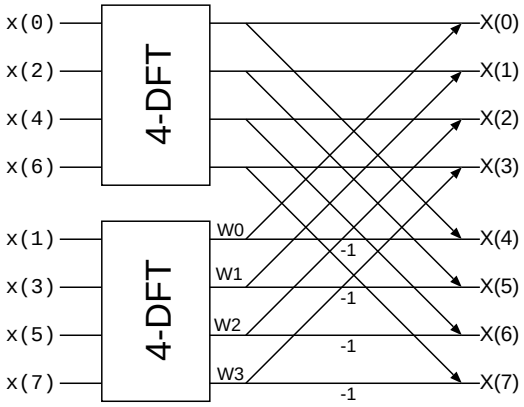

Abbildung 2: Butterfly-Darstellung von Gleichung 11, verbessert

Schema kann man nun jede der 4-DFT's durch zwei 2-DFT's ersetzen. Diese Unterteilung kann bis $N = 1$ fortgesetzt werden; für diesen Fall reduziert sich die DFT-Matrix auf den Skalar $e^{-j\frac{\pi}{1}\cdot 0} = 1$. Da die Multiplikation mit eins redundant ist, entsteht (s. Abbildung 3) für $N = 2$ der einfachste Schmetterling. Setzt

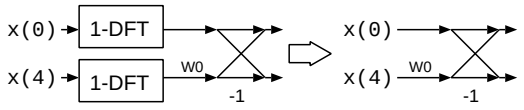

Abbildung 3: Schmetterlingsdiagramm für eine 2-Punkt-FFT

man aus dieser 2-FFT eine 4-FFT und aus der 4-DFT dann letztendlich eine 8-FFT zusammen, erhält man einen Schmetterlingsgraphen wie in Abbildung 4. Die Umsetzung dieses Signalflußplanes ist dank der wiederkehrenden und leicht zu parameterisierenden Struktur der Butterflys relativ unproblematisch. Zu beachten ist allerdings, daß die Eingangswerte vor dem „Anlegen" erst umsortiert werden müssen. Die hierfür notwendige Vorschrift ist relativ einfach; es handelt

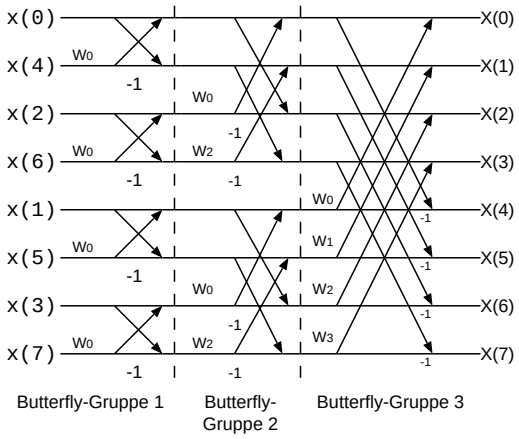

Abbildung 4: Schmetterlingsdiagramm für eine 8-Punkt-FFT

sich um die sog. *reversed bit order*. Dazu ist die Bitreihenfolge der Indizies in Binärcodierung einfach umzudrehen, was Tabelle 1 für $N = 8$ veranschaulicht: Die Umsortierung kann mittels einer in einem Array gespeicherten Zuordnungs-

Index	Binärcodierung	reversed bit order	resultierender Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Tabelle 1: Schema der *bit reverse order*

tabelle durchgeführt werden oder im Falle einer *in-place-computation*[2] mittels des durch Rabbiner und Gold in [2] beschriebenen und in Abbildung 5 dargestellten Algorithmuses geschehen.

1.4 Berechnung der inversen FFT

Gemäß [5] lautet die Vorschrift zur Berechnung der inversen Transformation

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{jkn\frac{2\pi}{N}} \tag{15}$$

[2] Eingangswerte werden durch die Transformationskoeffizienten überschrieben

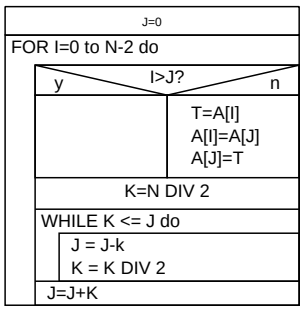

Abbildung 5: Umsortieralgorithmus nach Rabbiner und Gold

Mit der Substitution

$$W_N = e^{-j\frac{2\pi}{N}} \Rightarrow x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) \cdot W_N^{-nk} \tag{16}$$

erkennt man bei Gegenüberstellung von Gleichung 16 und Gleichung 2, daß sich, abgesehen vom Vorfaktor $\frac{1}{N}$, die iFFT bei Substitution von W_N^k durch $W_N^{-k} = (W_N)^*$ in das bereits bekannte Schema ergibt. Nutzt man die Relation

$$A \cdot B = (A^* \cdot B^*)^* \Rightarrow x(n) = (FFT(X^*(k)))^* \tag{17}$$

, dann läßt sich die Realisierung der FFT ohne Abänderungen nutzen, wenn zuvor der Eingangsvektor und anschließend der Ergebnisvektor konjugiert werden. Diese Variante ist schon deshalb sinnvoll, da bei der Verarbeitung reeler Eingangssignale der letzte Schritt redundant[3] ist. Bei der Umsetzung für Streaming-Anwendung ist es allerdings sinnvoller, statt der Konjugierung der Eingangswerte einen konjugiert-komplexen Koeffizientenvektor zu benutzen.

1.5 Gegenüberstellung DFT vs. FFT

Im Abschnitt 1.1 wurde die Anzahl an komplexen Multiplikationen[4] für die DFT mit N^2 hergeleitet. Für die FFT gilt, daß es genau $\log_2 N$ Butterfly-Gruppen gibt, welche jeweils $\frac{N}{2}$ komplexe Multiplikationen benötigen. Somit sind für eine N-Punkte-FFT

$$K = \frac{N}{2}\log_2 N \tag{18}$$

komplexe Multiplikationen notwendig. Tabelle 2 veranschaulicht deutlich den Vorteil der FFT gerade bei größeren Segmentlängen. Allerdings ist zu berücksichtigen, daß die FFT komplexere Adressierungen nötig macht, womit auch die Struktur des Digitalrechners ein wesentlicher Faktor für die Geschwindigkeit der Transformation ist. Signalprozessoren können die Berechnung der FFT durch folgende Strukturmerkmale beschleunigen:

[3] die zurücktransformierten Werte sind wieder reell

[4] zum Vergleich des Rechenaufwandes wurde die zeitaufwendigste Operation herangezogen

N	DFT	FFT	prozentualer Anteil
8	64	11	17.19
16	256	32	12.50
32	1024	80	7.81
64	4096	191	4.66
128	16384	447	2.73
256	65536	1024	1.56
512	262144	2304	0.88
1024	1048576	5120	0.49
2048	4194304	11264	0.27
4096	16777216	24575	0.15

Tabelle 2: Gegenüberstellung FFT vs. DFT

- Bereitstellung der Transformationskoeffizienten im Konstanten-ROM
- Bereitstellung von Maschinenbefehlen zur Berechnung der *reverse bit order* oder
- Bereitstellung einer Umcodierungstabelle zur *reverse bit order* im Konstanten-ROM
- Möglichkeit zur registerindirekten Adressierung mit mehreren Offset-Registern
- Unterstützung komplexer Rechenoperationen, z.B. durch entsprechende SIMD[5]-Befehle

2 Implementierung

Die folgenden Programmausschnitte sind eher als Anschauungsmaterial zu betrachten; wer eine wirklich gute allgemeine[6] Umsetzung sucht, wird unter [4] fündig.

2.1 Berechnung der *bit reverse order* Sequenz

```
uint *FFT::bitreverse(uint N,uint bits)
{
        uint *x,temp,i,j;

        if((x = new uint[N]) != 0)
        {
```

[5]single instruction multiple data
[6]d.h. nicht für eine bestimmte Struktur optimierte

```
                for(i=0;i<N;i++)
                {
                        temp = i;
                        for(j=0;j<bits;j++)
                        {
                                x[i] = x[i] << 1; // Platz schaffen
                                x[i] |= temp & 0x1; // LSB herausgreifen
                                temp = temp >> 1; // weiterrücken
                        }
                }
        }
        return(x);
}
```

2.2 die eigentliche Transformation

Zuerst empfiehlt es sich, die benötigten komplexen Operationen durch Makrofunktionen im Syntax einer 3-Adreß-Maschine umzusetzen:

```
#define CADD(a,b,c) c.re = (a.re + b.re); c.im = (a.im + b.im)
#define CMUL(a,b,c) c.re = (a.re*b.re-a.im*b.im);\
                    c.im = (a.re*b.im + a.im*b.re)
```

Somit kann die Implementation übersichtlich, fehlertolerant und doch schnell erfolgen

```
/** transformiert den Eingangsvektor x per
                FFT in den Ausgangsvektor X.
                direction = 1 : Hintransformation
                direction = -1 : Rücktransformation */

void FFT::transform(complex *x, complex *X, int direction)
{
        uint step;
        // Butterfly-Gruppe bzw. vertikaler Versatz der diagonalen Äste
        uint stepw;
        // Inkrement des W-Faktors
        uint jump;
        // Abstand der Butterfly-Anfänge (vertikal) in den Gruppen
        uint i;
        // Adressierung im Butterfly (jeweils die Elemente bis zur Hälfte);
        uint j; // Anfang des aktuellen Butterflies
        complex temp,temp2;
```

```
        // zuerst wird der Eingangsvektor umsortiert
        for(i=0;i<N;i++)
        {
                X[i] = x[reverse[i]];
                X[i].im *= direction;
                // für die iFFT muß die konjugiert-komplexe
                // Eingangsfolge verwendet werden
        }

        stepw = N;
        for(step=1;step < N; step = step << 1) // Butterfly-Gruppe
        {
                stepw = stepw >> 1;
                jump = step << 1;

                for(j=0;j<N;j+=jump) // Butterfly
                {
                        for(i=0;i<step;i++)
                        {
                                // temp = x[i+j+step] * W[i*stepw]
                                CMUL(X[i+j+step],W[i*stepw],temp);
                                //  x[i+j+step] = x[i+j]*temp*-1
                                // temp2 = temp * -1
                                temp2.re = temp.re * -1;
                                temp2.im = temp.im * -1;
                                CADD(X[i+j],temp2,X[i+j+step]);
                                // x[i+j] = x[i+j] + temp;
                                CADD(X[i+j],temp,X[i+j]);
                        }
                }
        }
        if(direction == -1)
        {
                // für die iFFT muß jetzt noch der
                // Ergebnisvektor konjugiert werden
                for(i=0;i<N;i++)
                {
                        X[i].re /= N;
                        X[i].im /= N;
                }
        }

}
```

Literatur

[1] Roland Best, *Digitale Signalverarbeitung und -simulation*, AT Verlag Aarau, Stuttgart 1989

[2] Laurence R. Rabbiner und Bernhard Gold, *Theory and Application of Digital Signal Processing*, Prentice Hall 1975

[3] Alan V. Oppenheim und Ronald W. Schafer, *Digital Signal Processing*, Prentice Hall 1975

[4] *www.fftw.org* freie und schnelle Implementierung der FFT incl. Dokumentation

[5] Peter Noll, *Signale und Systeme*, Institut für Fernmeldetechnik der TU Berlin, 1998