FFT

Die schnelle Fourier Transformation

Ein Vortrag von Holger Szillat

Betreut von Jürgen Schweizer

Proseminar “Rund um JPEG”
Sommersemester 2003
Universität Tübingen

szillat@informatik.uni-tuebingen.de

12.Mai 2003

Version 1.15

Übersicht

- *Motivation und Geschichte.*
- *Ein bißchen Mathematik.*
- *Die Diskrete Fourier Transformation.*
- *Die **Schnelle Fourier Transformation**.*
- *Kleines Fazit*

Etwas Geschichte

- Fourier Analyse von *Joseph Fourier* (1768-1830):
  Arbeit über math.Behandlung der Wärmeleitung (1807).
- Auch andere Mathematiker: Euler, Lagrange, Gauß:
  U.a. Umlaufbahnen von Himmelskörpern.
- Gauß hatte schon Idee für die *FFT*:
  Zerlege großes Problem in kleine Problemchen.

Etwas Geschichte

- 1965: *FFT* “erfunden” von **James W. Cooley** und **John W. Tukey** bei IBM.
  Spezialfall: $N = 2^p, p \in \mathbb{N}$
- Andere Wissenschaftler hatten ähnliche Ansätze.
  Beispiel: Algorithmus von **Runge**:
  Spezialfall: $N = 4m, m \in \mathbb{N}$
- Eigentlich nicht **ein** Verfahren, sondern viele!
- Für jeden Spezialfall kann man einen schnellen Algorithmus finden.

Etwas Motivation

Fourier Transformation wird eingesetzt für:

- (häufig) Signalverarbeitung
- Bildverarbeitung
- Lösung von Differentialgleichungen
- Spracherkennung und Akustik
- …

Etwas Mathematik

Zurückerinnern: $\rightarrow$ “Fourier Analyse”

- Funktionen mit $\cos$ und $\sin$ darstellen.
- **Orthonormalsystem** auf Vektorraum der Funktionen.
  $1, \cos(x), \sin(x), \cos(2x), \sin(2x), \ldots$
- Betrachten nur periodische Funktionen auf $[0; 2\pi]$.

Etwas Mathematik

Fourier Transformation in $\mathbb{R}$:

- Approximiere Funktion f durch:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos kx + b_k \sin kx$$

- Ziel: Stelle f als Linearkombination von Basisvektoren dar!
- Problem: “Vektorschreibweise”?

Etwas Mathematik

Fourier Transformation in $\mathbb{C}$:

- Benutze $e^{-2\pi ikt}$-Funktion als Basis.
- Dann:

$$f(t) := \int_{-\infty}^{+\infty} F(\omega) e^{-2\pi i\omega t} d\omega$$

- Ziel: Stelle f als Linearkombination von Basisvektoren dar!
- Problem: “Vektorschreibweise”?

Etwas Mathematik

Wollen Computer benutzen…

- Probleme:
  - “Unendliche” Basis als endlicher Vektor?!
  - Wie ist f gegeben?
  - Genaue Approximation nur bei unendlicher Rechenzeit!
- Also: “Handling” im Computer so nicht möglich!
- Kontinuierlich $\rightsquigarrow$ Diskret.

Lösung: Diskrete Fourier Transformation

Diskrete Fourier Transformation

- Nun also nur endliche Anzahl N an Stützstellen f_k von f bekannt.
- Approximiere nun **Fourier Koeffizienten**:

$$\begin{aligned} F(k) &= \int_{-\infty}^{\infty} f(t)\mathrm{e}^{-2\pi ikt}dt \\ &\approx \sum_{n=0}^{N-1} f_n \mathrm{e}^{-2\pi ikn/N} \end{aligned}$$

Diskrete Fourier Transformation

- Nun Vektorschreibweise möglich:

$$F(k) = < \begin{pmatrix} f(0) \\ f(1) \\ \dots \\ f(N-1) \end{pmatrix} | \frac{1}{\sqrt{N}} \begin{pmatrix} e^0 \\ e^{-2\pi i k 1/N} \\ \dots \\ e^{-2\pi i k (N-1)/N} \end{pmatrix} >$$

- Aber: Teure Multiplikationen!

Diskrete Fourier Transformation

Wie teuer ist die *DFT*?

- Berechnung der Koeffizienten:

$$F(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_n \mathbf{e}^{-2\pi i k n/N}$$

- **Ein** Koeffizient benötigt N Multiplikationen.
- Für N Koeffizienten also N^2 Multiplikationen notwendig!
  Komplexität: $O(N^2) \Rightarrow$ **Teuer!**
  $\rightarrow$ Bubblesort

Diskrete Fourier Transformation

- **Ein** Koeffizient als Skalarprodukt aufgefaßt.
- Betrachte Berechnung **aller** Koeffizienten als Matrixmultiplikation:

$$F = \mathit{DFT}_N \otimes f^t$$

mit:

$$\mathit{DFT}_N := \frac{1}{\sqrt{N}}(e^{-2\pi ikl/N})_{0\leq k,l\leq N-1}$$

- Dann bilden die **Fourier Koeffizienten** einen Vektor.

Diskrete Fourier Transformation

Vereinfachungen und Überlegungen:

- Setze: $\omega_n := e^{-2\pi i/n}$.
  Beachte: ω_n ist n-te Einheitswurzel.
  D.h.:
  $$\omega^0 = 1 \text{ und } \omega^n = \omega^0 = 1$$
- Die Potenzen $\omega_n^k, k = 0, \ldots, n$
  - liegen auf dem Einheitskreis.
  - bilden die Eckpunkte eines regelmäßigen n-Ecks.
- Die Exponenten lassen sich deßhalb mod n reduzieren!

Diskrete Fourier Transformation

Diskrete Fourier Transformation als Matrix:

- Definiere:

$$\begin{aligned} DFT_N &:= \frac{1}{\sqrt{N}}\left(e^{-2\pi ikl/N}\right)_{0\le k,l\le N-1} \\ &= \frac{1}{\sqrt{N}}\left(\omega_N^{kl}\right)_{0\le k,l\le N-1} \end{aligned}$$

- Offensichtlich: $DFT_N \in \mathbb{C}^{N\times N}$

Diskrete Fourier Transformation

Diskrete Fourier Transformation als Matrix: Beispiel für $N = 4$:

$$\begin{aligned} \mathit{DFT}_4 &= \frac{1}{\sqrt{4}} \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^8 \end{pmatrix} \\ &= \frac{1}{\sqrt{4}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} \end{aligned}$$

Denn: $\omega^1 = -i$, $\omega^3 = -\omega^1 = i$, sowie: $\omega^2 = -1 = -\omega^0$.

Inverse Diskrete Fourier Transformation

- DFT_N ist unitär, d.h. $DFT_N^{-1} = DFT_N^* = iDFT_N$.
- Inverse Diskrete Fourier Transformation als Matrix:

$$\begin{aligned} iDFT_4 &= (DFT_4)^* \\ &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \end{aligned}$$

- Einfach zu berechnen!

Auf dem Weg zur *FFT*:

Wie kann man die *DFT* einfacher berechnen?

- Idee: Wenn möglichst viele Einträge $= 0$, dann könnte man Multiplikationen sparen.
- Idee: Stelle DFT_N als Produkt einfacherer Matrizen dar.

Auf dem Weg zur *FFT*

Betrachte nochmal $N = 4$:

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ \hline 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{array}\right) \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix}, F = \Omega f^T$$

Sortiere F-Vektor nach geraden und ungeraden Indizes:

$$\begin{pmatrix} F_0 \\ F_2 \\ F_1 \\ F_3 \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ \hline 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{array}\right) \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix}, \hat{F} = \hat{\Omega} f^T$$

Behalte dabei aber den f-Vektor bei.

Auf dem Weg zur *FFT*

Betrachte nochmal:

$$\begin{pmatrix} F_0 \\ F_2 \\ F_1 \\ F_3 \end{pmatrix} = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ \hline 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{array}\right) \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix}, \hat{F} = \hat{\Omega} f^T$$

Dann gilt doch für die Untermatrizen $\hat{\Omega}_{ik}$:

$$\hat{\Omega}_{11} = \hat{\Omega}_{12}, \hat{\Omega}_{22} = \omega^2 \hat{\Omega}_{21}$$

Es gibt also ein “Muster”!

Auf dem Weg zur *FFT*

- Jetzt "Faktorisierung" von DFT_4 möglich:

$$
\begin{aligned}
DFT_4 &= \left(\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & 1 & \omega^2 \\ \hline 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{array}\right) \\
&= \left(\begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & \omega^2 \end{array}\right) \left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 0 & \omega^2 & 0 \\ 0 & \omega^1 & 0 & \omega^3 \end{array}\right) \\
&= \begin{pmatrix} \Omega_2 & 0_2 \\ 0_2 & \Omega_2 \end{pmatrix} \begin{pmatrix} I_2 & I_2 \\ D_2 & -D_2 \end{pmatrix}
\end{aligned}
$$

- Wobei I_2 die 2×2-Einheitsmatrix und D_2 die 2×2-Diagonalmatrix mit $D_2 = \text{diag}(\omega_4^0, \omega_4^1)$ ist.

Auf dem Weg zur *FFT*

Was hat das alles gebracht?

- Problem der Ordnung N zurückgeführt auf ein Problem der Ordnung $m = \frac{1}{2}N$.
- Faktorisierung vereinfacht Berechnung.
- Weniger Multiplikationen, mehr Additionen.

Auf dem Weg zur *FFT*

Zur Berechnung des Vektor F müssen nur folgende Rechenschritte ausgeführt werden:

Input		1.Schritt		2.Schritt	Output
f_0	0	$f_0^1 := f_0 + f_2$	0	$f_0^2 := f_0^1 + f_1^1$	F_0
f_1	1	$f_1^1 := f_1 + f_3$	2	$f_1^2 := (f_0^1 - f_1^1)\omega^0$	F_2
f_2	2	$f_2^1 := (f_0 - f_2)\omega^0$	1	$f_2^2 := f_2^1 + f_3^1$	F_1
f_3	3	$f_3^1 := (f_1 - f_3)\omega^1$	3	$f_3^2 := (f_2^1 - f_3^1)\omega^0$	F_3

Zu sehen: Anzahl der Multiplikationen: $4 \leq 16$.

Die *FFT* ganz allgemein

- Sei $N = 2^p$ mit $p \in \mathbb{N}$, $m = \frac{N}{2}$.
- Es ergibt sich:

$$\begin{aligned} \mathit{FFT}_N &= P_N \Omega_N \\ &= P_N \begin{pmatrix} \Omega_m & 0_m \\ 0_m & \Omega_m \end{pmatrix} \begin{pmatrix} I_m & I_m \\ D_m & -D_m \end{pmatrix} \end{aligned}$$

- Ein großes Problem in zwei kleine Problemchen zerlegt
  $\Rightarrow$ “Divide And Conquer”-Algorithmus.
- Ist nach p Schritten fertig. ($\rightarrow$ “Tiefe des Baumes”)
- Ist tatsächlich die *FFT*!
- Bleibt nur noch: P_N?

Die *FFT* – Letzter Schritt

- Permutationen wieder rückgängig machen.
- Betrachte Indizes der Eingabe und der Ausgabe:

Input		...		Output
f_0	0	...	0	F_0
f_1	1	...	2	F_2
f_2	2	...	1	F_1
f_3	3	...	3	F_3

- Was fällt auf?

$$\begin{aligned} 0_{10} = 00_2 &\rightarrow 00_2 = 0_{10} \\ 1_{10} = 01_2 &\rightarrow 10_2 = 2_{10} \\ 2_{10} = 10_2 &\rightarrow 01_2 = 1_{10} \\ 3_{10} = 11_2 &\rightarrow 11_2 = 3_{10} \end{aligned}$$

- Geht für beliebiges $N = 2^p, p \in \mathbb{N}$.

Problemlösung – Kleine Analyse

+ “Divide-and-Conquer” Schema.
+ Viel weniger Multiplikationen.
+ Permutationen nicht schlimm; nur anderer Zugriff.
+ Permutationen wieder rückgängig machen: Einfach durch Bitumkehr!
+ Komplexität: $O(N \log_2(N))$
  ($\rightarrow$ Quicksort)

Also: *FFT* ist richtig gut!

Kleines Fazit

- *DFT* ist teuer.
- *FFT* ist billiger: Für ein festes $N \in \mathbb{N}$ läßt sich der Algorithmus sehr einfach implementieren.
- *FFT* Algorithmen gibt es viele im Internet:
  - FXT
  - FFTW (“Fastest Fourier Transform in the West”)
  - ...