Parallele FFT-Algorithmen

Jörg Haeger

11. Juli 1996

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Die diskrete Fourier-Transformation (DFT)

Definition 1 (DFT) Die *diskrete Fourier-Transformation (DFT) der Länge N* ist gegeben durch

$$\mathbf{T} : \mathbf{C}^N \to \mathbf{C}^N,$$

$$y_j := \mathbf{T}_j(x) := \sum_{k=0}^{N-1} x_k \omega_N^{jk} \tag{1}$$

mit

$$\omega_N := e^{\frac{-2\pi i}{N}} = \cos(2\pi/N) - i\sin(2\pi/N). \tag{2}$$

In der Definition läßt sich direkt ablesen, daß jeder Ergebniswert y_j von allen Eingangswerten x_k abhängt. Ein naiver iterativer Algorithmus zur Berechnung der DFT hat also eine Zeitkomplexität von $O(N^2)$.

Definition 2 (2D-DFT) Die *2D-DFT* ist gegeben durch

$$\mathbf{T} : \mathbf{C}^{N_1 \times N_2} \to \mathbf{C}^{N_1 \times N_2},$$

$$y_{j_1,j_2} := \mathbf{T}_{j_1,j_2}(x) := \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} x_{k_1,k_2} e^{-2\pi i(j_1 k_1/N_1 + j_2 k_2/N_2)}. \tag{3}$$

Die 2D-DFT läßt sich auf die 1D-DFT zurückführen

$$\mathbf{T}_{j_1,j_2}(x) = \sum_{k_1=0}^{N_1-1} \Big(\sum_{k_2=0}^{N_2-1} x_{k_1,k_2} e^{-2\pi i j_2 k_2/N_2} \Big) e^{-2\pi i j_1 k_1/N_1} \tag{4}$$

$$= \sum_{k_1=0}^{N_1-1} \Big(\sum_{k_2=0}^{N_2-1} x_{k_1,k_2} \omega_{N_2}^{j_2 k_2} \Big) \omega_{N_1}^{j_1 k_1}. \tag{5}$$

2 Permutationen und Graphen

2.1 Permutationen

Im Zusammenhang mit *Butterfly-* und *de Bruijn-Graphen* ist die folgende Permutation und ihre Umkehrung von Bedeutung

Definition 3 (Shuffle- und Unshuffle-Permutation) Die *Shuffle-Permutation* $\sigma_{l,p}$ ist eine Abbildung

$$\sigma_{l,p} : \mathbb{N} \to \mathbb{N},$$

$$\begin{aligned} &\sigma_{l,p}((a_{n-1}, \ldots, a_{p+l}, a_{p+l-1}, a_{p+l-2}, \ldots, a_p, a_{p-1}, \ldots, a_0)_2) \\ &\quad = (a_{n-1}, \ldots, a_{p+l}, a_{p+l-2}, \ldots, a_p, a_{p+l-1}, a_{p-1}, \ldots, a_0)_2 \end{aligned} \tag{6}$$

Es werden also in der Dualdarstellung einer Zahl l Stellen ab der Position p zyklisch nach links rotiert.
Die *Unshuffle-Permutation* $\sigma_{l,p}^{-1}$ ist die Umkehrabbildung zu $\sigma_{l,p}$.

Beispiele: $\sigma_{4,2}((10100110)_2) = (10001110)_2$, $\quad \sigma_{4,2}^{-1}((10001110)_2) = (10100110)_2$.

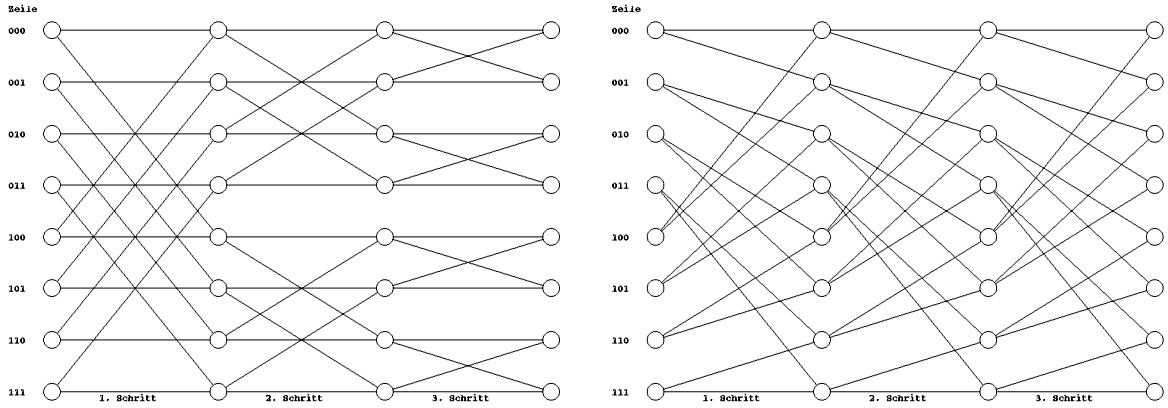

Abbildung 1: links: Butterfly-Graph, rechts: iterativ fortgesetzter de Bruijn-Graph (Basis 2, Dimension 3).

2.2 Graphen

2.2.1 Butterfly-Graphen

Ein n dimensionaler *Butterfly-Graph* zur Basis 2 besteht aus n Stufen und 2^n Knoten pro Stufe. Für die Stufen $t = 1, \ldots, n$ ist ein Knoten $a = (a_{n-1}, \ldots, a_0)_2$ mit den beiden Vorgängern $(a_{n-1}, \ldots, a_{n-t+1}, 0, a_{n-t-1}, \ldots, a_0)_2$ und $(a_{n-1}, \ldots, a_{n-t+1}, 1, a_{n-t-1}, \ldots, a_0)_2$ verbunden (Abbildung 1).

2.2.2 De Bruijn-Graphen

Ein n dimensionaler *de Bruijn-Graph* zur Basis 2 besteht aus 2^n Knoten und $2(n + 1)$ Kanten. Jeder Knoten hat 2 ein- und 2 ausgehende Kanten. Bei der Beschreibung von Verbindungsnetzwerken werden de Bruijn-Graph iterativ verwendet. Ein Knoten $a = (a_{n-1}, \ldots, a_0)_2$ ist mit seinen Vorgaengern $(0, a_{n-1}, \ldots, a_1)_2$ und $(1, a_{n-1}, \ldots, a_1)_2$ und seinen Nachfolgern $(a_{n-2}, \ldots, a_1, 0)_2$ und $(a_{n-2}, \ldots, a_1, 1)_2$ verbunden (Abbildung 1).

3 Die schnelle Fourier-Transformation (FFT)

Bei der *schnellen Fourier-Transformation (FFT)* handelt es sich um einen Algorithmus zur Berechnung der DFT, der durch Anwendung des *Divide and Conquer*-Paradigmas eine Zeitkomplexität von $O(N \log N)$ erreicht. Im folgenden soll der iterative FFT-Algorithmus und seine Datenflußstruktur hergeleitet werden, da diese auch die Grundlage paralleler Implementierungen bilden.

Sei $N := 2^p > 1, p \in \mathbb{N}$ und $n := N/2$.
Die Berechnung der Komponenten y_j wird getrennt für gerade und ungerade Indizes j betrachtet. Ziel ist jeweils, die Berechnung der Komponenten y_j auf die Berechnung einer DFT der halben Länge n zurückzuführen.

Für $j = 0, \ldots n-1$ gilt

$$y_{2j} = \sum_{k=0}^{N-1} x_k \omega_N^{2jk} = \sum_{k=0}^{n-1} (x_k \omega_N^{2jk} + x_{n+k} \omega_N^{2j(n+k)}) \tag{7}$$

Um die beiden Summanden zusammenzufassen betrachten wir

$$\omega_N^{2j(n+k)} = \omega_N^{jN+2jk} = \omega_N^{2jk} = \omega_n^{jk} \tag{8}$$

Wir erhalten also

$$y_{2j} = \sum_{k=0}^{n-1} (x_k + x_{n+k}) \omega_n^{jk} \tag{9}$$

Analog erhalten wir für ungerade Indizes von y

$$y_{2j+1} = \sum_{k=0}^{N-1} x_k \omega_N^{(2j+1)k} = \sum_{k=0}^{n-1} (x_k \omega_N^{(2j+1)k} + x_{n+k} \omega_N^{(2j+1)(n+k)}) \tag{10}$$

$$= \sum_{k=0}^{n-1} (x_k \omega_N^{2jk} \omega_N^k + x_{n+k} \omega_N^{2j(n+k)} \omega_N^{n+k}) \tag{11}$$

Zum Zusammenfassen der Summanden benötigen wir noch, daß gilt

$$\omega_N^{n+k} = \omega_N^{N/2+k} = \cos(\frac{2\pi}{N}(N/2+k)) - i \sin(\frac{2\pi}{N}(N/2+k)) \tag{12}$$

$$= \cos(\pi + \frac{2\pi}{N}k) - i \sin(\pi + \frac{2\pi}{N}k) = -\cos(\frac{2\pi}{N}k) + i \sin(\frac{2\pi}{N}k) = -\omega_N^k$$

Damit ergibt sich für diesen Fall

$$y_{2j+1} = \sum_{k=0}^{n-1} (x_k - x_{n+k}) \omega_N^k \quad \omega_n^{jk} \tag{13}$$

Die Gleichungen 9 und 13 zeigen, daß wir $(y_{2j})_{0 \le j < n-1}^T$ durch die DFT der Länge n von $(x_j + x_{n+j})_{0 \le j < n-1}^T$ und $(y_{2j+1})_{0 \le j < n-1}^T$ durch die DFT von $((x_j - x_{n+j})\omega_N^j)_{0 \le j < n-1}^T$ erhalten. Für $n > 1$ läßt sich dieses Verfahren auf die beiden DFTs der Länge n getrennt rekursiv anwenden.

Dieses Vorgehen führt zu einem *Butterfly-Graph* der Dimension $\mathrm{ld}\, N$ zur Basis 2 als Datenflußgraph der FFT (Abbildung 2).
Im ersten Schritt ergeben sich folgende Funktionen in den Knoten

$$f_j^{(1)}(a,b) = \begin{cases} a+b & \text{falls } 0 \le j < n \\ (a-b)\omega_N^{j-n} & \text{falls } n \le j < N \end{cases} \tag{14}$$

Auf die Datenflußstruktur haben die Gewichte ω_N^{j-n} keine Auswirkung, so daß es im Folgenden ausreicht nur die beiden Typen

$$\begin{array}{ll} f_1(a,b) = a+b & \text{für die erste Hälfte und} \\ f_2(a,b) = (a-b)W, W \in \mathbf{C} & \text{für die zweite Hälfte} \end{array} \tag{15}$$

zu unterscheiden.

Durch das in jedem Schritt vorgenommene Anordnen der Ergebnisse zu geraden Indizes in der oberen Hälfte des betrachteten Ausschnitts, befindet sich das Ergebnis y_j der DFT in der letzten Spalte in der durch die gespiegelte $\mathrm{ld}\, N$-stellige Dualdarstellung von j bestimmten Zeile *(Bitreversal)*.

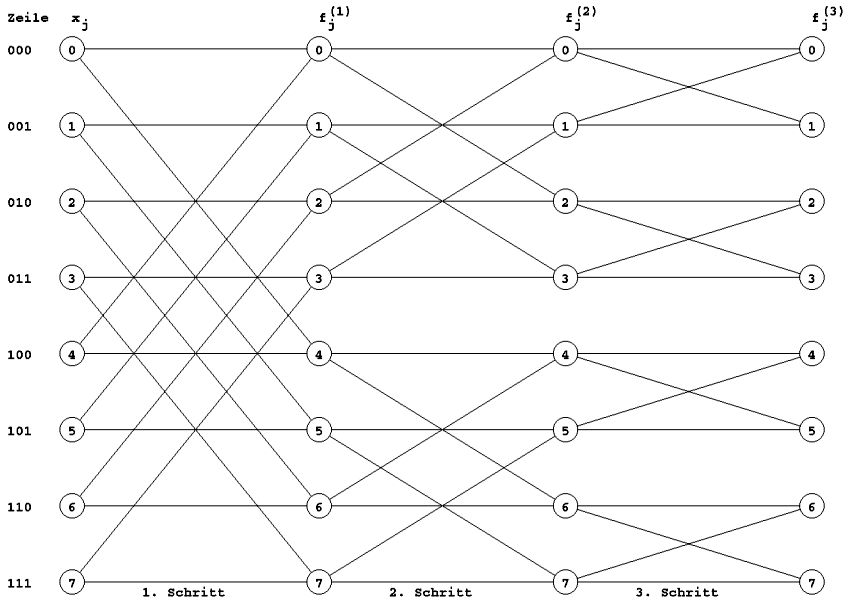

Abbildung 2: Datenflußgraph der FFT ($N = 8$).

4 Parallele 2D-FFT Algorithmen

Parallele 2D-FFT Algorithmen sind in der digitalen Bildverarbeitung von Bedeutung, wenn Bilder aus z.B. 512×512 Pixeln, also 2^{18} einzelnen Werten, möglichst in Echtzeit transformiert werden sollen. In diesem Abschnitt sollen zwei vom Ansatz und der Verteilung der Daten auf die Prozessoren her verschiedene parallele 2D-FFT Algorithmen für den Einsatz auf MIMD-Systemen (**M**ultiple **I**nstruction **M**ultiple **D**ata) vorgestellt werden. Als Kommunikationsnetzwerk des Zielsystems wird jeweils ein Basis 2 de Bruijn-Graph angenommen.

Es gibt zwei Möglichkeiten Algorithmen zur Berechnung der 2D-DFT zu realisieren

- durch geschachtelte Anwendung der 1D-FFT
- durch Übertragung des Vorgehens beim Entwickeln der 1D-FFT auf 2 Dimensionen

Für die folgenden Algorithmen habe die Eingangsmatrix die Größe $N \times N$ mit $N := 2^p > 1, p \in \mathbb{N}$ und es sei $n := N/2$.

4.1 Algorithmus 1: Realisierung durch 1D-FFTs

Ein 2D-FFT Algorithmus läßt sich gemäß der Gleichung

$$y_{j_1,j_2}(x) = \sum_{k_2=0}^{N-1} \Big(\sum_{k_1=0}^{N-1} x_{k_1,k_2} \omega_N^{j_1 k_1} \Big) \omega_N^{j_2 k_2} \tag{16}$$

unter Verwendung des 1D-FFT Algorithmus konstruieren. Zunächst werden die Spalten der Eingabematrix durch die 1D-FFT transformiert. Anschließend werden dann auf gleiche Weise die Zeilen transformiert. Die Datenflußstruktur läßt sich wie bei der 1D-FFT durch einen Butterfly-Graph beschreiben, in dem jedoch jetzt in der ersten Hälfte die Spalten- und in der zweiten die Zeilentransformationen stattfinden. Hieraus ergeben sich andere Gewichte der Knotenfunktionen. Aus der Reihenfolge

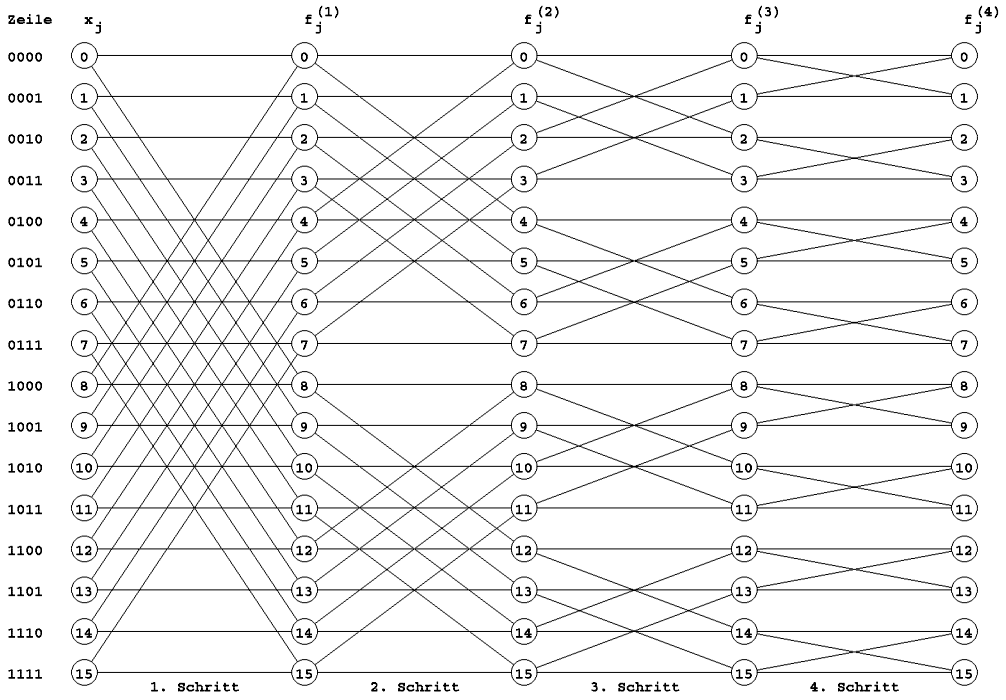

Abbildung 3: Algorithmus 1, Ansatz: 2-stufiger Butterfly-Graph zur Basis 2 ($N^2 = 16$).

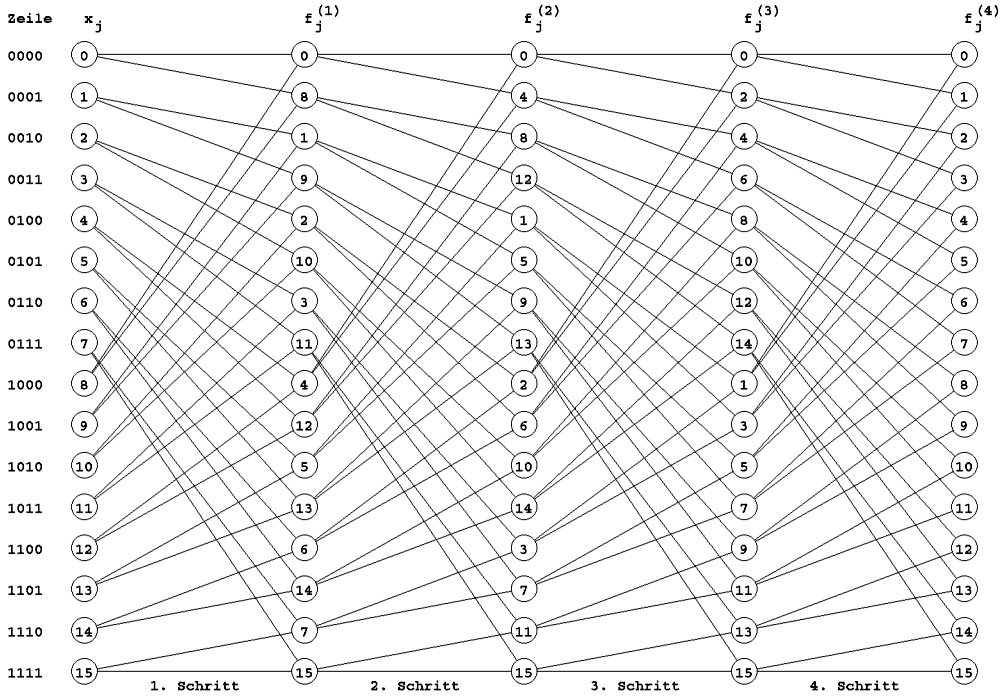

Abbildung 4: Algorithmus 1, 1. Schritt: Übergang zu einem homogenen Datenflußgraph ($N^2 = 16$).

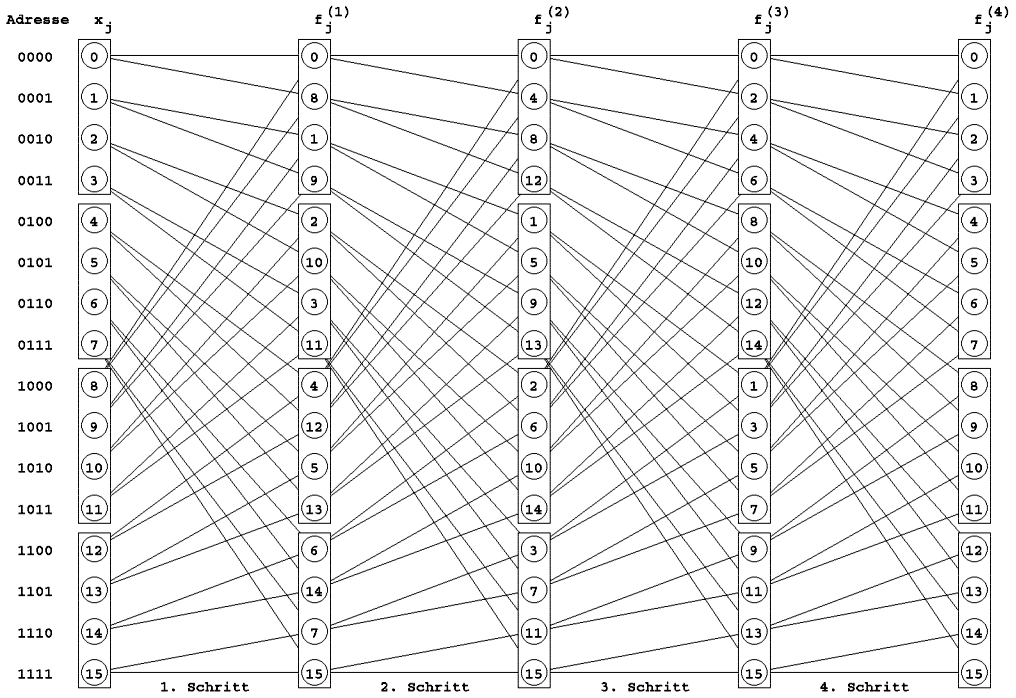

Abbildung 5: Algorithmus 1, 2. Schritt: Skalierung auf P Prozessoren ($N^2 = 16$, $P = 4$).

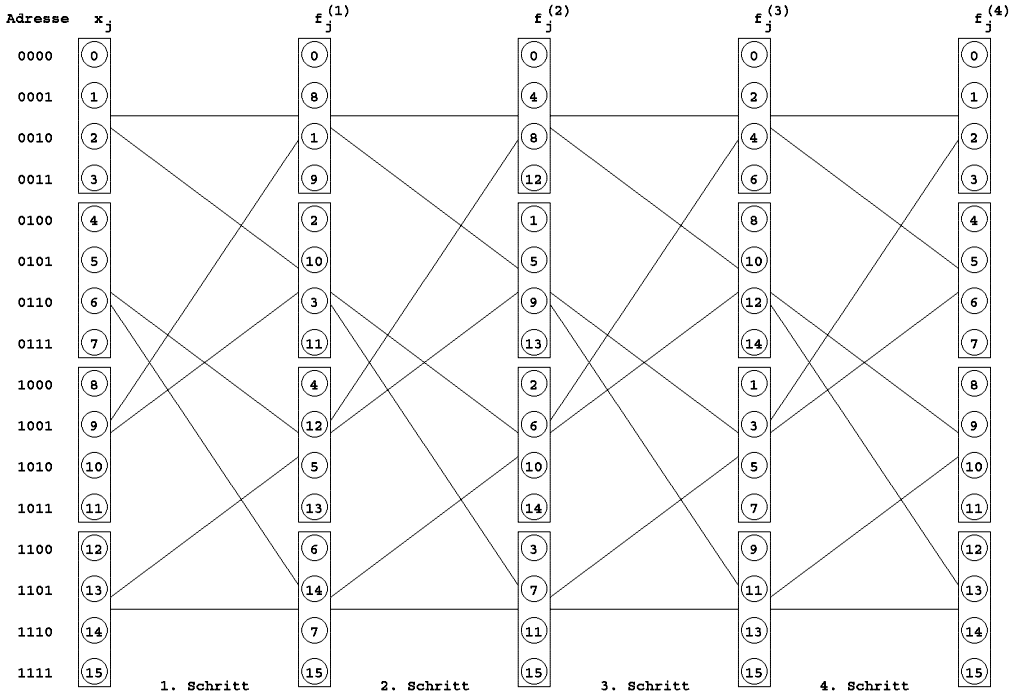

Abbildung 6: Algorithmus 1, Ergebnis: Einbettung in den Kommunikationsgraph des Zielsystems ($N^2 = 16$ und $P = 4$).

der Transformationen ergibt folgende Datenverteilung auf die Knoten des Butterfly-Graphen (Abbildung 3)

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

Ein paralleler 2D-FFT Algorithmus für P Prozessoren auf Basis dieses Butterfly-Graphen läßt sich wie folgt konstruieren

- die Knoten im Butterfly-Graph werden mit jedem Schritt Shuffle-permutiert ($\sigma_{\mathrm{ld}\, N^2,0}$), um einen bzgl. der Verbindungen homogenen de Bruijn-Graph zu erhalten (Abbildung 4)
- der de Bruijn-Graph der Dimension $\mathrm{ld}\, N^2$ wird in einen de Bruijn-Graph der Dimension $\mathrm{ld}\, P$ eingebettet, der dem Kommunikationsnetzwerk des Zielsystems entspricht (Abbildungen 5 und 6)

4.2 Algorithmus 2: Basis 4 2D-FFT

Ein anderer Ansatz überträgt das Vorgehen bei der Entwicklung des 1D-FFT Algorithmus. Für $j = 0, \ldots, n-1$ ergibt sich

$$y_{2j_1,2j_2} = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} x_{k_1,k_2}\omega_N^{2j_2k_2+2j_1k_1} \tag{17}$$

$$= \sum_{k_1=0}^{N-1}(\sum_{k_2=0}^{N-1} x_{k_1,k_2}\omega_N^{2j_2k_2})\omega_N^{2j_1k_1}$$

$$= \sum_{k_1=0}^{N-1}(\sum_{k_2=0}^{n-1}(x_{k_1,k_2} + x_{k_1,n+k_2})\omega_n^{j_2k_2})\omega_N^{2j_1k_1}$$

$$= \sum_{k_1=0}^{n-1}(\sum_{k_2=0}^{n-1}(x_{k_1,k_2} + x_{k_1,n+k_2})\omega_n^{j_2k_2} + \sum_{k_2=0}^{n-1}(x_{n+k_1,k_2} + x_{n+k_1,n+k_2})\omega_n^{j_2k_2})\omega_n^{j_1k_1}$$

$$= \sum_{k_1=0}^{n-1}(\sum_{k_2=0}^{n-1}(x_{k_1,k_2} + x_{k_1,n+k_2} + x_{n+k_1,k_2} + x_{n+k_1,n+k_2})\omega_n^{j_2k_2})\omega_n^{j_1k_1}$$

$$= \sum_{k_1=0}^{n-1}\sum_{k_2=0}^{n-1}(x_{k_1,k_2} + x_{k_1,n+k_2} + x_{n+k_1,k_2} + x_{n+k_1,n+k_2})\omega_n^{j_2k_2+j_1k_1}$$

Die Berechnung der Matrix $(y_{2j_1,2j_2})_{0\le j_1,j_2<n-1}$ wird also auf die Berechnung der DFT der $n \times n$-Matrix $(x_{j_1,j_2} + x_{n+j_1,n+j_2})_{0\le j_1,j_2<n-1}$ zurückgeführt. Die Werte $y_{2j_1,2j_2+1}$, $y_{2j_1+1,2j_2}$ und $y_{2j_1+1,2j_2+1}$ ergeben sich analog unter Berücksichtigung der auftretenden Gewichte.

Die Datenflußstruktur dieses 2D-FFT Algorithmus läßt sich durch einen $\log_4 N^2$-dimensionalen Butterfly-Graph zur Basis 4 (Abbildung 7) beschreiben, wenn die Eingangsmatrix wie folgt auf die Knoten verteilt wird

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

und in den Knoten folgende Funktionen

$$\begin{aligned} g_1(a,b,c,d) &= f_1(f_1(a,c), f_1(b,d)) \\ g_2(a,b,c,d) &= f_2(f_1(a,c), f_1(b,d)) \\ g_3(a,b,c,d) &= f_1(f_2(a,c), f_2(b,d)) \\ g_4(a,b,c,d) &= f_2(f_2(a,c), f_2(b,d)) \end{aligned} \tag{18}$$

für folgende Bereiche

g_1	g_2
g_3	g_4

verwendet werden, wobei f_1 und f_2 durch Gleichung 15 gegeben sind.

Die Entwicklung eines parallelen Algorithmus für P Prozessoren, der der Struktur dieses Butterfly-Graphen entspricht, verläuft wie folgt

- jeder Schritt im Basis 4 Butterfly-Graph wird in 2 Schritte zerlegt, wobei im ersten ein Spalten- und im zweiten ein Zeilentransformationsschritt durchgeführt wird (Abbildung 8)
- der entstandene Basis 2 Butterfly-Graph mit $\operatorname{ld} N^2$ Ebenen wird durch Einbetten in einen Butterfly-Graph der Dimension $\operatorname{ld} 2P$ auf $2P$ Prozessoren skaliert (Abbildung 9)
- die letzten $\operatorname{ld} N^2 - \operatorname{ld} 2P$ Schritte lassen sich durch einen beliebigen iterativen 2D-FFT Algorithmus lokal auf den einzelnen Prozessoren berechnen (Abbildung 10)
- im Butterfly-Graph der Dimension $\operatorname{ld} 2P$ werden mit jedem der $\operatorname{ld} 2P$ Schritte die Knoten Shuffle-permutiert ($\sigma_{\operatorname{ld} 2P,0}$), so daß ein homogener iterierter de Bruijn-Graph entsteht (Abbildung 11)
- der de Bruijn-Graph wird in einen de Bruijn-Graph der Dimension $\operatorname{ld} P$ eingebettet, so daß Funktionen, die identische Daten benötigen, sich auf demselben Prozessor befinden (Abbildung 12)

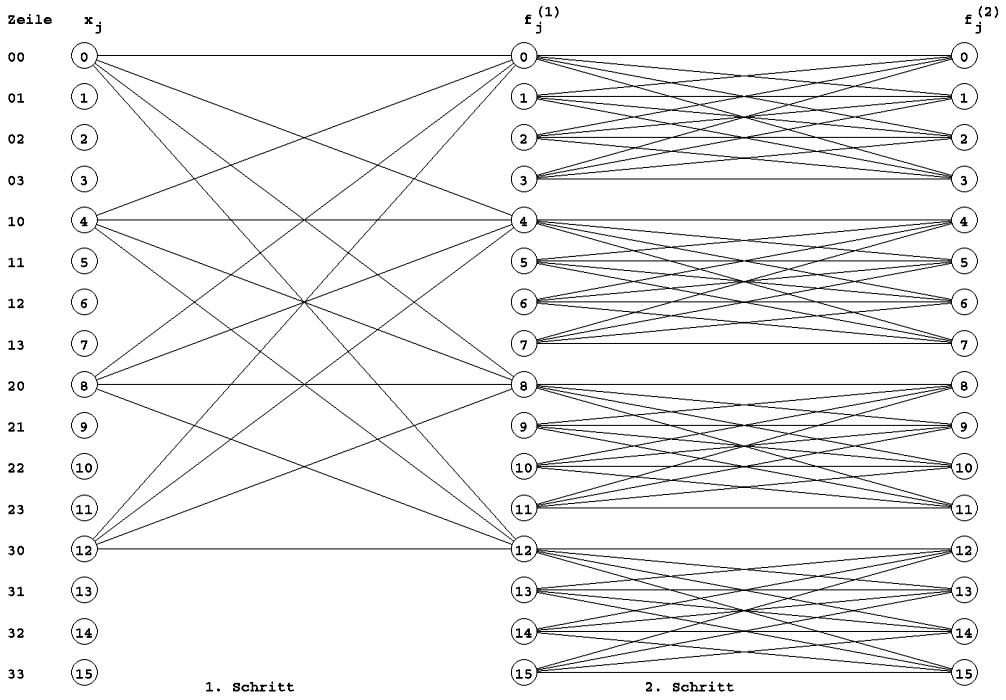

Abbildung 7: Algorithmus 2, Ansatz: $\log_4 N^2$-dimensionaler Butterfly-Graph zur Basis 4 (Kanten des 1. Schrittes nur exemplarisch, $N^2 = 16$).

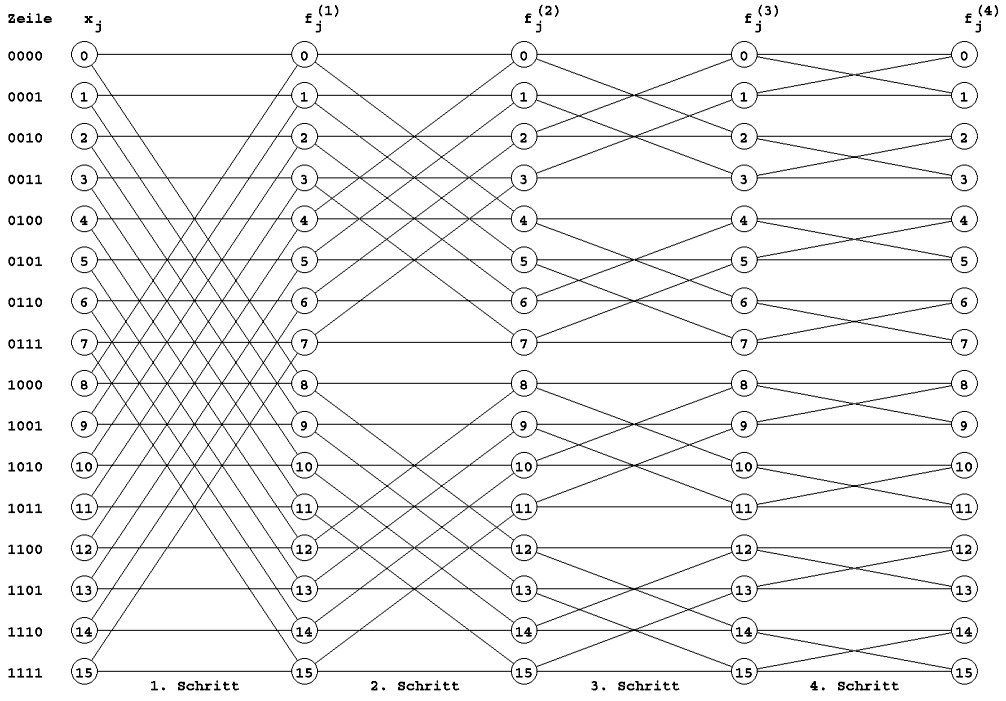

Abbildung 8: Algorithmus 2, Schritt 1: $\operatorname{ld} N^2$-dimensionaler Butterfly-Graph zur Basis 2 ($N^2 = 16$).

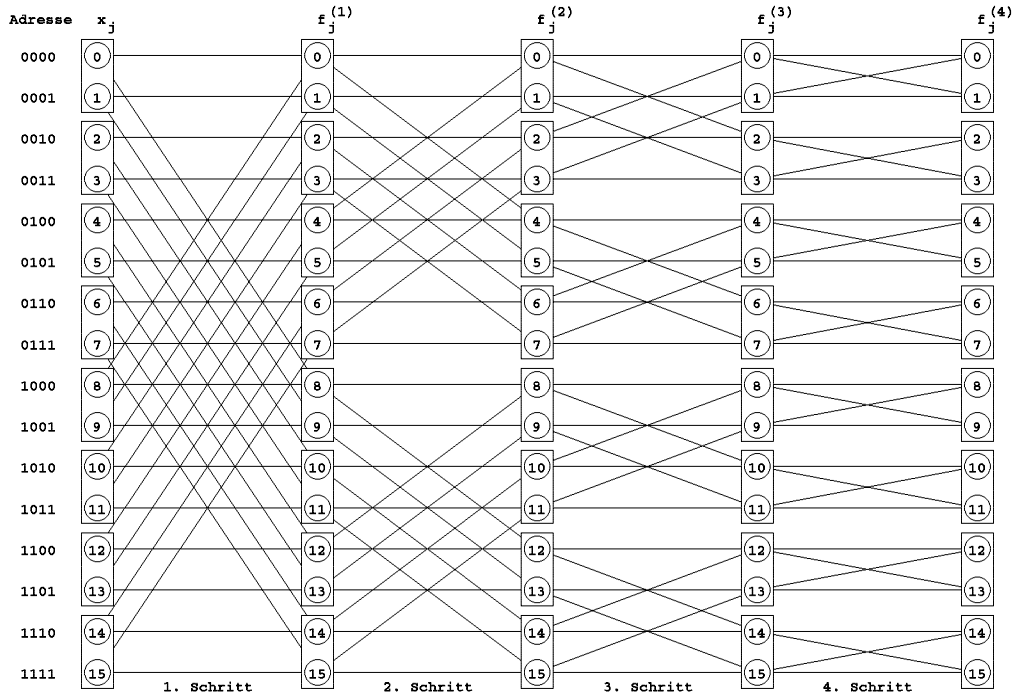

Abbildung 9: Algorithmus 2, Schritt 2: Skalierung auf $2P$ Prozessoren ($N^2 = 16$ und $P = 4$).

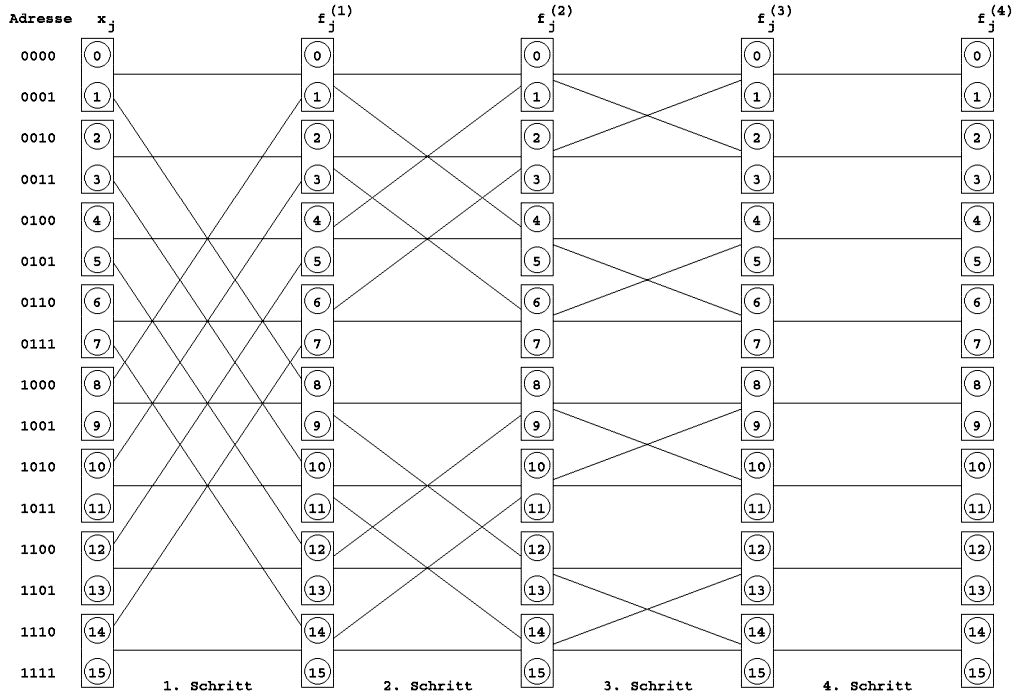

Abbildung 10: Algorithmus 2, Schritt 3: Butterfly-Graph mit $\mathrm{ld}\, 2P$ globalen Schritten ($P = 4$).

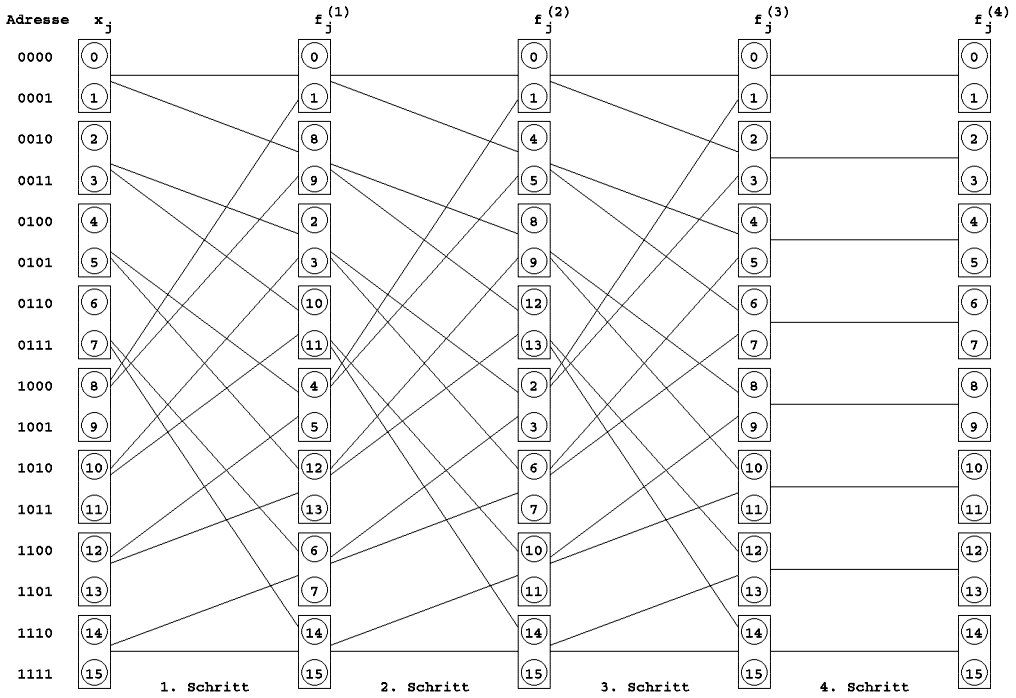

Abbildung 11: Algorithmus 2, Schritt 4: durch Permutation homogenisierter iterierter de Bruijn-Graph mit $\mathrm{ld}\, 2P$ Schritten ($P = 4$).

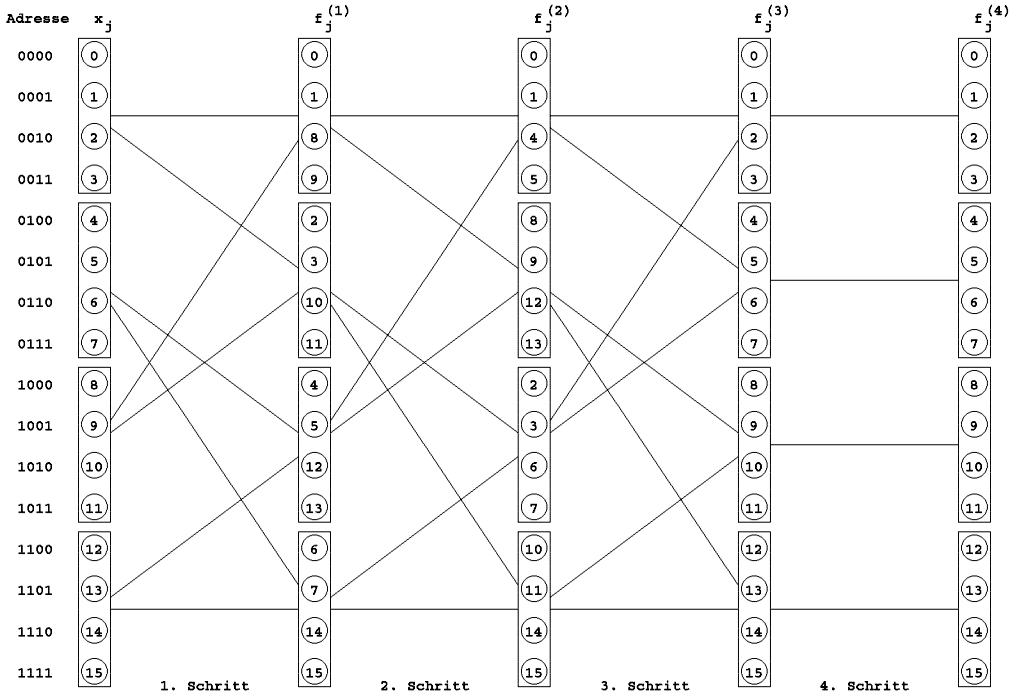

Abbildung 12: Algorithmus 2, Ergebnis: Einbettung in den Kommunikationsgraph des Zielsystems ($N^2 = 16$ und $P = 4$).

5 Laufzeitergebnisse

Um das Laufzeitverhalten der Algorithmen vergleichen zu können, folgen Meßergebnisse, die auf einem als de Bruijn-Netzwerk konfigurierten Transputer-basierten System aus bis zu 128 Prozessoren ermittelt wurden. Die in Abbildung 13 rechts dargestellte Effizienz ist wie folgt definiert

$$\text{Speedup} = \frac{\text{sequentielle Laufzeit}}{\text{parallele Laufzeit}} (\geq 1) \tag{19}$$

$$\text{Effizienz} = \frac{\text{Speedup}}{\text{Anzahl an Prozessoren}} (\leq 1) \tag{20}$$

N = 256		runtime	Mflops
pe's	system	[sec]	
	IBM RS-6000 220	3,01	1,74
	IBM RS-6000 350	1,13	4,64
	IBM RS-6000 580	0,80	6,55
1	T800 25 MHz	7,87	0,67
1	T800 25 MHz (par)	8,95	0,59
1	T805 30 MHz	7,68	0,68
1	T805 30 MHz (par)	10,96	0,48
1	T800 20 MHz	9,77	0,54
1	T800 20 MHz (par)	10,69	0,49
2	T800 20 MHz (par)	6,36	0,82
4	T800 20 MHz (par)	3,41	1,54
8	T800 20 MHz (par)	1,76	2,98
16	T800 20 MHz (par)	0,93	5,66
32	T800 20 MHz (par)	0,54	9,77
64	T800 20 MHz (par)	0,31	16,83
128	T800 20 MHz (par)	0,18	28,56

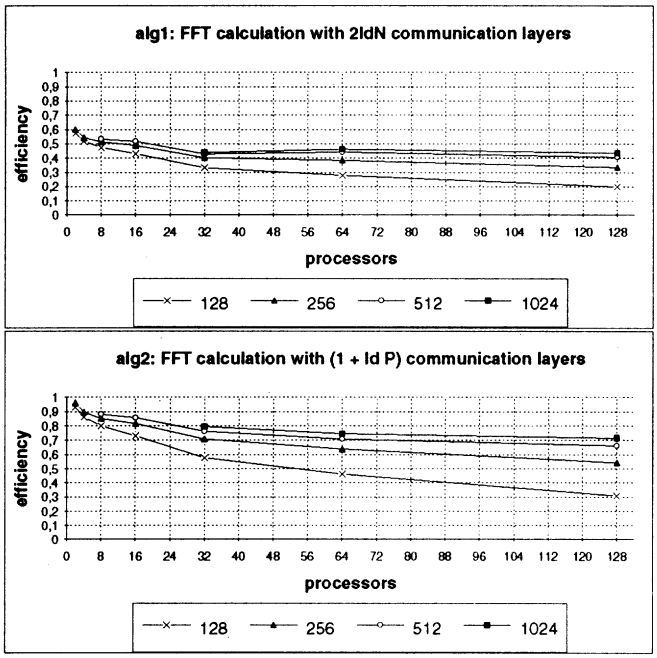

Abbildung 13: links: Vergleich der Laufzeit auf unterschiedlichen Standardprozessoren und Transputern, rechts: Effizienz der Algorithmen 1 und 2 für $N \times N$- Bilder (Ergebnisse übernommen aus [1]).

Literatur

[1] Michael Nölle und Nils Jungclaus (199x): Efficient Implementation of FFT-Like Algorithms on MIMD Systems,

[2] Michael Nölle (1995): Parallele Algorithmen für die digitale Bildverarbeitung,