Aus der Fakultät Mathematik

Parallele nichtäquidistante schnelle Fourier-Transformation und schnelle Summation

Diplomarbeit

im Rahmen des Diplomstudiengangs
Mathematik

Vorgelegt von
Michael Pippig
aus Chemnitz

Chemnitz, 11. März 2009

Prüfer/Betreuer	Prof. Dr. Daniel Potts, Dr. Stefan Kunis

Inhaltsverzeichnis

1

Einleitung

Die Entwicklung effizienter Algorithmen ist eine wesentliche Aufgabe der angewandten Mathematik. Ein bekanntes Beispiel ist die schnelle Fourier-Transformation (FFT) [4, 14], welche die Komplexität einer diskreten Fourier-Transformation der Größe n von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n \log n)$ reduziert. Sie nimmt auf Grund ihrer breiten Anwendungsmöglichkeiten in nahezu allen Bereichen aus Naturwissenschaft und Technik eine besondere Rolle im wissenschaftlichen Rechnen ein. Die nichtäquidistante schnelle Fourier-Transformation (NFFT) [6, 3, 31, 33, 29, 16, 20] ist eine Verallgemeinerung der FFT auf nichtäquidistante Daten. Mit ihrer Hilfe ist es gelungen ein großes Feld weiterer Problemklassen für die Lösung mit schnellen Fourier-Algorithmen zu erschließen. Als Beispiele seien die Magnetresonanztomographie [11] und die schnelle Summation radialer Kernfunktionen [28] aufgeführt. Letztere findet Anwendung in der Simulation von Partikelsystemen [26], bei der Lösung von Integralgleichungen oder bei der effizienten Realisierung der diskreten Gauß-Transformation. Während bisher die Komplexität und numerische Genauigkeit die wichtigsten Kriterien für den Erfolg eines Algorithmus waren, gewann in den letzten Jahren die Parallelisierbarkeit zunehmend an Bedeutung. Besonders im Bereich der Computersimulationen haben wegen der großen Datenmengen und der enormen Rechenzeiten massiv parallele Rechner Einzug gehalten. Mit immer leistungsfähigeren Rechnern entsteht somit aber auch die Notwendigkeit schneller paralleler Algorithmen, welche in der Lage sind die zur Verfügung stehenden Ressourcen optimal zu nutzen. In dieser Arbeit werden effiziente parallele Algorithmen für die dreidimensionale nichtäquidistante schnelle Fourier-Transformation sowie deren Anwendung in der schnellen Summation entwickelt und auf ihre Skalierbarkeit untersucht. Ausgehend von der seriellen dreidimensionalen Implementation und meinen Ansätzen zur Parallelisierung der NFFT [25], entwickeln wir erstmalig eine hochskalierende parallele NFFT und deren Adjungierte. Diese Algorithmen implementieren wir in C und MPI mit Hilfe des parallelen Programmpakets der FFTW [14, 12] und stellen sie in einer Programmbibliothek zur Verfügung. Ihre gute Skalierung weisen wir durch Laufzeittests auf drei Höchstleistungsrechnern nach. Mit Hilfe dieser Algorithmen parallelisieren wir die schnelle Summation auf eine einfache Weise. Der Inhalt der vorliegenden Arbeit gliedert sich wie folgt.

Kapitel 2 dient als Überblick zur seriellen nichtäquidistanten Fourier-Transformation. Wir führen die benötigten mathematischen Grundlagen und Bezeichnungen ein und erläutern daraufhin die Problemstellung der nichtäquidistanten diskreten Fourier-Transformation. Die Approximationsidee der nichtäquidistanten schnellen Fourier-Transformation aus [29] wird erläutert und der resultierende serielle Algorithmus mit der zugehörigen Fehlerabschätzung vorgestellt.

Zu Beginn von Kapitel 3 erläutern wir die Problemstellung der parallelen FFT. Die parallele Verteilung der Daten für die FFT erfordert eine Erweiterung der Notation und führt uns zu einem Ansatz für die parallele Datenverteilung der NFFT. Diese bildet das Fundament der im Folgenden entwickelten parallelen Algorithmen für die NFFT und die adjungierte NFFT. Die Details der Implementation dieser Algorithmen mit Hilfe der FFTW werden ausführlich dargestellt. Bei der Analyse des Speicherbedarfs kommt die parallele Datenaufteilung der FFTW maßgeblich zum Tragen. Die Verwendung einer alternativen FFT-Bibliothek mit einer anderen Aufteilung der Daten wird ebenfalls diskutiert, deren Speicherbedarf analysiert und mit der aktuellen Implementation verglichen. Zum Abschluss des Kapitels wird die Benutzerschnittstelle der entwickelten Programmbibliothek erläutert.

Im Kapitel 4 befassen wir uns mit den numerischen Tests der parallelen Algorithmen mit bis zu 512 Prozessoren auf den drei Höchstleistungsrechnern CHiC, JUMP und JUGENE. Dabei wird besonderes Augenmerk auf die Skalierbarkeit gelegt. Neben Vergleichen der Laufzeiten, Beschleunigungen und Effizienzen der parallelen NFFT mit der renommierten Programmbibliothek der FFTW, wird auch der verbleibende serielle Anteil der Algorithmen experimentell bestimmt und analysiert.

Abschließend widmen wir uns im Kapitel 5 einer Anwendung der NFFT in der schnellen Summation. Bekannte Algorithmen hierfür beruhen auf der schnellen Multipol-Methode (FMM) [17, 15, 1] oder hierarchischen Matrizen [18]. Der Vorteil des in [27, 28] vorgestellten Algorithmus liegt in einem einfachen modularen Aufbau, dessen Fundament die Anwendung einer NFFT und einer adjungierten NFFT bilden. Wir verwenden daher unsere parallelen Algorithmen aus Kapitel 3 für die Parallelisierung der dreidimensionalen schnellen Summation. Zusätzlich schlagen wir eine alternative Approximation der Kernfunktion vor, um ihre Auswertung im sogenannten Nahfeld zu beschleunigen.

An dieser Stelle möchte ich mich bei Allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben. Besonderer Dank gebührt dabei meinen Betreuern, Herrn Prof. Dr. Daniel Potts und Herrn Dr. Stefan Kunis, für die vielen Ratschläge, ihr großes Engangement und das entgegengebrachte Vertrauen. Weiterer Dank gilt Herrn Prof. Dr. Matthias Bolten und Herrn Toni Volkmer für die zahlreichen Hinweise bei der Programmentwicklung unter Linux. Schließlich möchte ich mich bei meiner Familie und meiner Verlobten Jule für die Unterstützung in allen Belangen bedanken.

2

Die serielle NDFT und NFFT

2.1 Notation

Wir bezeichnen in gewohnter Weise die natürlichen Zahlen $\mathbb{N} := \{1, 2, \ldots\}$, die ganzen Zahlen $\mathbb{Z}$, die reellen Zahlen $\mathbb{R}$ und die komplexen Zahlen $\mathbb{C}$ und vereinbaren die Abkürzungen $\mathbb{R}_+ := \{x \in \mathbb{R} : x \geq 0\}$ und $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$.

Sei der Torus $\mathbb{T}^d := \mathbb{R}^d/\mathbb{Z}^d$ der Dimension $d \in \mathbb{N}$ als Faktorgruppe mit der üblichen periodischen Addition gegeben, wobei wir seine Elemente mit ihren eindeutig bestimmten Repräsentanten in $[-\frac{1}{2}, \frac{1}{2})^d$ identifizieren. Schreiben wir $\boldsymbol{r} \in \mathbb{T}^d$ für einen beliebigen Vektor $\boldsymbol{r} \in \mathbb{R}^d$, so identifizieren wir diesen Vektor mit der zugehörigen Äquivalenzklasse und deren Repräsentant in $[-\frac{1}{2}, \frac{1}{2})^d$. Wir wollen im Folgenden nichtäquidistante Stützstellen $\boldsymbol{x}$ aus dem Torus $\mathbb{T}^d$ wählen. Schreiben wir $\boldsymbol{x} + \boldsymbol{r}$ für $\boldsymbol{x} \in \mathbb{T}^d$, $\boldsymbol{r} \in \mathbb{R}^d$, so interpretieren wir $\boldsymbol{r} \in \mathbb{T}^d$ und wenden die periodische Addition auf dem Torus an.

Den Hilbert-Raum $L^2(\mathbb{T}^d)$ der quadratisch integrierbaren Funktionen auf dem Torus $\mathbb{T}^d$

$$L^2(\mathbb{T}^d) := \Big\{ f : \mathbb{T}^d \to \mathbb{C} : \|f\|_{L^2(\mathbb{T}^d)} := \left(\int_{\mathbb{T}^d} |f(\boldsymbol{x})|^2 \mathrm{d}\boldsymbol{x} \right)^{1/2} < \infty \Big\},$$

mit dem zugehörigen Skalarprodukt

$$\langle f, g \rangle_{L^2(\mathbb{T}^d)} := \int_{\mathbb{T}^d} f(\boldsymbol{x}) \overline{g(\boldsymbol{x})} \mathrm{d}\boldsymbol{x},$$

sowie den Banach-Raum $C[A]$ der stetigen Funktionen auf einer kompakten Menge $A \subseteq \mathbb{T}^d$ mit der zugehörigen Norm

$$C[A] := \big\{ f : A \to \mathbb{C} : f \text{ stetig auf } A \big\}, \quad \|f\|_{C[A]} := \max_{\boldsymbol{x} \in A} |f(\boldsymbol{x})|,$$

führen wir auf bekannte Weise ein. Den linearen Raum der r-mal stetig differenzierbaren Funktionen auf dem Torus $\mathbb{T}^d$ bezeichnen wir mit $C^r[\mathbb{T}^d]$, $r \in \mathbb{N}_0$.

Seien $\boldsymbol{a} = (a_t)_{t=0}^{d-1} \in \mathbb{R}^d$ und $\boldsymbol{b} = (b_t)_{t=0}^{d-1} \in \mathbb{R}^d$ gegeben. Wir definieren die komponentenweise Auf- bzw. Abrundung $\lceil \boldsymbol{a} \rceil := (\lceil a_t \rceil)_{t=0}^{d-1} \in \mathbb{Z}^d$, $\lfloor \boldsymbol{a} \rfloor := (\lfloor a_t \rfloor)_{t=0}^{d-1} \in \mathbb{Z}^d$ und das komponentenweise Produkt $\boldsymbol{a} \odot \boldsymbol{b} := (a_t b_t)_{t=0}^{d-1} \in \mathbb{R}^d$ und schreiben $\boldsymbol{a} \leq \boldsymbol{b}$, wenn $a_t \leq b_t$ für $t = 0, \ldots, d-1$ gilt. Analog sind die Relationen $\boldsymbol{a} < \boldsymbol{b}$ und $\boldsymbol{a} \ll \boldsymbol{b}$ definiert. Im Fall $\boldsymbol{a} \leq \boldsymbol{b}$ können wir den d-dimensionalen halboffenen Quader $[\boldsymbol{a}, \boldsymbol{b}) \subseteq \mathbb{R}^d$ durch ein Kreuzprodukt in der folgenden Weise definieren

$$[\boldsymbol{a}, \boldsymbol{b}) := \bigtimes_{t=0}^{d-1} [a_t, b_t)\,.$$

Den abgeschlossenen Quader $[\boldsymbol{a}, \boldsymbol{b}] \subseteq \mathbb{R}^d$ führen wir analog ein. Einen d-dimensionalen Quader $[\boldsymbol{B}; \boldsymbol{L})_{\mathbb{T}^d} \subseteq \mathbb{T}^d$ auf dem Torus $\mathbb{T}^d$ mit dem Multi-Offset $\boldsymbol{B} = (B_t)_{t=0}^{d-1} \in \mathbb{T}^d$ und der Multi-Länge $\boldsymbol{L} = (L_t)_{t=0}^{d-1} \in [0,1]^d$ definieren wir durch

$$[\boldsymbol{B}; \boldsymbol{L})_{\mathbb{T}^d} := \left\{ \boldsymbol{x} \in \mathbb{T}^d : \boldsymbol{x} = \boldsymbol{B} + \boldsymbol{r},\, \boldsymbol{r} \in [\boldsymbol{0}, \boldsymbol{L}) \right\}.$$

Sei $\boldsymbol{N} = (N_t)_{t=0}^{d-1} \in 2\mathbb{N}^d$. Die Inverse $\boldsymbol{N}^{-1} := \left(\frac{1}{N_t}\right)_{t=0}^{d-1} \in (0, \frac{1}{2}]^d$ führen wir ebenfalls komponentenweise ein. Als Multi-Frequenzbereich bezeichnen wir die Multi-Indexmenge $I_{\boldsymbol{N}} := \mathbb{Z}^d / \boldsymbol{N}\mathbb{Z}$ mit der üblichen $\boldsymbol{N}$-periodischen Addition, wobei wir die Elemente dieser Faktorgruppe mit ihren eindeutig bestimmten Repräsentanten in $[-\frac{1}{2}\boldsymbol{N}, \frac{1}{2}\boldsymbol{N}) \cap \mathbb{Z}^d$ identifizieren. Die Schreibweise $\boldsymbol{o} + \boldsymbol{r}$ mit $\boldsymbol{o} \in I_{\boldsymbol{N}}$ und $\boldsymbol{r} \in \mathbb{Z}^d$ erklären wir durch die Addition der zugehörigen Äquivalenzklassen in $I_{\boldsymbol{N}}$. Zur Vereinfachung der Notation wollen wir mit einem Multi-Index $\boldsymbol{k} \in I_{\boldsymbol{N}}$ sowohl Elemente einer Matrix als auch eines Vektors adressieren. Ein Element $f \in \operatorname{span}\left\{ \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k} \cdot} : \boldsymbol{k} \in I_{\boldsymbol{N}} \right\}$ bezeichnen wir als trigonometrisches Polynom vom Grad $\boldsymbol{N} \in 2\mathbb{N}^d$.

Den d-dimensionalen Block $[\boldsymbol{o}; \boldsymbol{l})_{\boldsymbol{N}} \subseteq I_{\boldsymbol{N}}$ mit dem Multi-Offset $\boldsymbol{o} \in I_{\boldsymbol{N}}$ und der Multi-Länge $\boldsymbol{l} \in [\boldsymbol{0}, \boldsymbol{N}] \cap \mathbb{Z}^d$ definieren wir als

$$[\boldsymbol{o}; \boldsymbol{l})_{\boldsymbol{N}} := \left\{ \boldsymbol{k} \in I_{\boldsymbol{N}} : \boldsymbol{k} = \boldsymbol{o} + \boldsymbol{r},\, \boldsymbol{r} \in [\boldsymbol{0}, \boldsymbol{l}) \cap \mathbb{Z}^d \right\},$$

und die zur Stützstelle $\boldsymbol{x} \in \mathbb{T}^d$ zugeordnete Multi-Indexmenge $I_{\boldsymbol{N},m}(\boldsymbol{x})$, $m \in \mathbb{N}$, durch

$$I_{\boldsymbol{N},m}(\boldsymbol{x}) := [\, \lfloor \boldsymbol{N} \odot \boldsymbol{x} \rfloor - m\boldsymbol{1}\,;\, (2m+2)\boldsymbol{1}\,)_{\boldsymbol{N}}\,. \tag{2.1}$$

$B_\varepsilon(\boldsymbol{y}) := \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \varepsilon\}$ bezeichne eine abgeschlossene Kugel mit Mittelpunkt $\boldsymbol{y} \in \mathbb{R}^d$ und Radius $\varepsilon > 0$. Wir lassen die Angabe des Mittelpunktes für $\boldsymbol{y} = \boldsymbol{0}$ weg, d.h. $B_\varepsilon := B_\varepsilon(\boldsymbol{0})$.

Das innere Produkt zwischen einem Multi-Index $\boldsymbol{k} \in I_{\boldsymbol{N}}$ und einer Stützstelle $\boldsymbol{x} \in \mathbb{T}^d$ definieren wir in der gewohnten Weise als $\boldsymbol{k}\boldsymbol{x} := k_0 x_0 + k_1 x_1 + \ldots + k_{d-1} x_{d-1} \in \mathbb{R}$. Außerdem führen wir die Abkürzung $\boldsymbol{1} := (1)_{t=0}^{d-1} \in \mathbb{N}^d$ ein. Zu $M \in \mathbb{N}$ definieren wir eine Indexmenge $\mathcal{M} := \{1, \ldots, M\}$. M soll im Folgenden die Anzahl der Stützstellen $\boldsymbol{x}_j \in \mathbb{T}^d$, $j \in \mathcal{M}$, bezeichnen.

Die charakteristische Funktion $\chi_{\mathcal{A}} : \mathbb{R}^d \to \{0, 1\}$ einer Menge $\mathcal{A} \subseteq \mathbb{R}^d$ führen wir in gewohnter Weise ein, d.h.

$$\chi_{\mathcal{A}}(\boldsymbol{x}) := \begin{cases} 1 & : \boldsymbol{x} \in \mathcal{A}, \\ 0 & : \boldsymbol{x} \notin \mathcal{A}. \end{cases}$$

2.2 Die Nichtäquidistante diskrete Fourier-Transformation (NDFT)

Für gegebene Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, sind wir an der Berechnung des trigonometrischen Polynoms

$$f(\boldsymbol{x}) := \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}} \tag{2.2}$$

an den ebenfalls gegebenen Stützstellen $\boldsymbol{x}_j \in \mathbb{T}^d$, $j \in \mathcal{M}$, interessiert. Somit besteht das Problem in der Berechnung von $f_j := f(\boldsymbol{x}_j)$, $j \in \mathcal{M}$. In Matrix-Vektor-Schreibweise stellen wir dies dar als

$$\boldsymbol{f} = \boldsymbol{A} \hat{\boldsymbol{f}}$$

mit

$$\boldsymbol{f} := (f_j)_{j \in \mathcal{M}} \in \mathbb{C}^M, \quad \boldsymbol{A} := \left(\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j} \right)_{j \in \mathcal{M};\, \boldsymbol{k} \in I_{\boldsymbol{N}}} \in \mathbb{C}^{M \times |I_{\boldsymbol{N}}|}, \quad \hat{\boldsymbol{f}} := \left(\hat{f}_{\boldsymbol{k}} \right)_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \in \mathbb{C}^{|I_{\boldsymbol{N}}|}.$$

Der direkte Algorithmus für die Berechnung dieses Matrix-Vektor-Produkts heißt nichtäquidistante diskrete Fourier-Transformation (NDFT) und benötigt $\mathcal{O}(M|I_{\boldsymbol{N}}|)$ arithmetische Operationen. Eine eng verwandte Aufgabenstellung ist die adjungierte NDFT

$$\hat{\boldsymbol{h}} = \boldsymbol{A}^{\mathsf{H}} \boldsymbol{f}, \qquad \hat{h}_{\boldsymbol{k}} = \sum_{j \in \mathcal{M}} f_j \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}, \quad \boldsymbol{k} \in I_{\boldsymbol{N}},$$

wobei $\hat{\boldsymbol{h}} = (\hat{h}_{\boldsymbol{k}})_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \in \mathbb{C}^{|I_{\boldsymbol{N}}|}$ und $\boldsymbol{A}^{\mathsf{H}} = \overline{\boldsymbol{A}}^{\top}$ die konjugiert Transponierte der nichtäquidistanten Fourier-Matrix $\boldsymbol{A}$ ist.

2.3 Die Nichtäquidistante schnelle Fourier-Transformation (NFFT)

Die NFFT [6, 3, 31, 33, 29, 16, 20] ist ein schneller approximativer Algorithmus für die nichtäquidistante diskrete Fourier-Transformation. Sie reduziert die Komplexität von $\mathcal{O}(M|I_{\boldsymbol{N}}|)$ auf $\mathcal{O}\left(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}| + (\log(1/\varepsilon))^d M\right)$, wobei ε die gewünschte Genauigkeit darstellt. Wir folgen der Darstellung der Algorithmen für die NFFT und deren Adjungierte aus [20]. Die Grundidee ist die Verwendung einer herkömmlichen FFT und einer **1**-periodischen Fensterfunktion $\tilde{\varphi}$, welche im Ortsbereich $\mathbb{T}^d$ und im Frequenzbereich $\mathbb{Z}^d$ gut lokalisiert ist.

2.3.1 Die Fensterfunktion

Gegeben seien univariate Fensterfunktionen $\varphi_t : \mathbb{R} \to \mathbb{R}$, $t = 0, \ldots, d-1$, deren 1-periodisierte Funktionen

$$\tilde{\varphi}_t(x_t) := \sum_{r \in \mathbb{Z}} \varphi_t(x_t + r)$$

uniform konvergente Fourier-Reihen besitzen und gut lokalisiert im Ortsbereich $\mathbb{T}$ und im Frequenzbereich $\mathbb{Z}$ sind. Dann können wir für $t = 0, \ldots, d-1$ die periodisierten Funktionen $\tilde{\varphi}_t$ durch ihre Fourier-Reihen

$$\tilde{\varphi}_t(x_t) = \sum_{k_t \in \mathbb{Z}} c_{k_t}(\tilde{\varphi}_t) \, \mathrm{e}^{-2\pi \mathrm{i} k_t x_t}$$

mit den Fourier-Koeffizienten

$$c_{k_t}\left(\tilde{\varphi}_t\right) := \int_{\mathbb{T}} \tilde{\varphi}_t\left(x_t\right) \mathrm{e}^{+2\pi \mathrm{i} k_t x_t}\,\mathrm{d}x_t = \int_{\mathbb{R}} \varphi_t\left(x_t\right) \mathrm{e}^{+2\pi \mathrm{i} k_t x_t}\,\mathrm{d}x_t, \quad k_t \in \mathbb{Z},$$

darstellen. Die multivariate Fensterfunktion $\varphi : \mathbb{R}^d \to \mathbb{R}$ definieren wir als Tensorprodukt

$$\varphi\left(\boldsymbol{x}\right) := \varphi_0\left(x_0\right) \varphi_1\left(x_1\right) \ldots \varphi_{d-1}\left(x_{d-1}\right).$$

Somit folgt für die Fourier-Koeffizienten der $\mathbf{1}$-periodisierten Funktion $\tilde{\varphi} = \sum_{\boldsymbol{r} \in \mathbb{Z}^d} \varphi(\cdot + \boldsymbol{r})$

$$c_{\boldsymbol{k}}\left(\tilde{\varphi}\right) = c_{k_0}\left(\tilde{\varphi}_0\right) c_{k_1}\left(\tilde{\varphi}_1\right) \ldots c_{k_{d-1}}\left(\tilde{\varphi}_{d-1}\right).$$

Im Weiteren schränken wir uns auf die Gauß-Funktion [6, 31, 5] als eine mögliche Fensterfunktion ein, d.h. wir verwenden für $t = 0, \ldots, d-1$

$$\varphi_t\left(x_t\right) = \left(\pi b_t\right)^{-1/2} \mathrm{e}^{-\frac{\left(n_t x_t\right)^2}{b_t}}, \quad c_{k_t}\left(\tilde{\varphi}_t\right) = \frac{1}{n_t} \mathrm{e}^{-b_t\left(\frac{\pi k_t}{n_t}\right)^2},$$

mit dem sogenannten Abschneideparameter $m \in \mathbb{N}$, der Überabtastrate $\sigma_t \in \mathbb{R}$ und den Parametern $n_t \in 2\mathbb{N}$ und $b_t := \frac{2\sigma_t}{2\sigma_t - 1} \frac{m}{\pi}$, welche sich bei der Erläuterung des Algorithmus noch ergeben werden.

Schnelle Berechnung der Gauß-Funktion

Im Folgenden wird such herausstellen, dass wir die Funktionswerte $\varphi_t(x_t - \frac{l'_t}{n_t})$, $l'_t \in I_{n_t,m}(x_t)$, für viele Stützstellen $x_t \in \mathbb{T}$ benötigen. Für ihre schnelle Berechnung nutzen wir eine Eigenschaft der Gauß-Funktion, welche in [16] beschrieben wurde. Offenbar besitzt $l'_t \in I_{n_t,m}(x_t)$ die Gestalt $l'_t = u_t + l$ mit $u_t := \lfloor n_t x_t \rfloor - m \in I_{n_t}$ und einem $l \in \{0, \ldots, 2m+1\}$. Wir reduzieren die Anzahl der benötigten Aufrufe der Funktion `exp()` durch eine geschickte Faktorisierung der Gauß-Funktion

$$\sqrt{\pi b}\varphi_t\left(x_t - \frac{l'_t}{n_t}\right) = \mathrm{e}^{-\frac{\left(n_t x_t - l'_t\right)^2}{b_t}} = \mathrm{e}^{-\frac{\left(n_t x_t - u_t\right)^2}{b_t}} \left(\mathrm{e}^{\frac{2\left(n_t x_t - u_t\right)}{b_t}}\right)^l \mathrm{e}^{-\frac{l^2}{b_t}}.$$

Für festes $x_t \in \mathbb{T}$ ist der erste Faktor eine Konstante. Der zweite Faktor lässt sich durch schrittweise Multiplikation mit der Konstanten $\mathrm{e}^{2(n_t x_t - u_t)/b_t}$ für $l = 0, \ldots, 2m+1$ berechnen. Die Funktionswerte e^{-l^2/b_t}, $l = 0, \ldots, 2m+1$, sind unabhängig von der Stützstelle $x_t \in \mathbb{T}$. Sie müssen also nur einmal berechnet werden. Für M Stützstellen wird somit die Anzahl der Aufrufe der Funktion `exp()` von $(2m+2)M$ auf $2M + (2m+2)$ und im dreidimensionalen Fall auf Grund der Definition der Fensterfunktion φ als Tensorprodukt von $3(2m+2)M$ auf $6M + 3(2m+2)$ reduziert.

2.3.2 Der Ansatz

Sei die Größe der FFT durch $\boldsymbol{n} \in 2\mathbb{N}^d$, $\boldsymbol{n} \geq \boldsymbol{N}$, gegeben. Wir wollen das trigonometrische Polynom f, gegeben in der definierenden Gleichung (2.2), durch eine Linearkombination $\check{s}$ von Translaten der Fensterfunktion $\tilde{\varphi}$ approximieren, d.h.

$$f\left(\boldsymbol{x}\right) \approx \check{s}\left(\boldsymbol{x}\right) := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n}}} g_{\boldsymbol{l}}\, \tilde{\varphi}\left(\boldsymbol{x} - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right), \tag{2.3}$$

mit Koeffizienten $g_{\boldsymbol{l}} \in \mathbb{C}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}$.

2.3.3 Approximation im Frequenzbereich

Durch einen Wechsel in den Frequenzbereich erhalten wir

$$\breve{s}\left(\boldsymbol{x}\right) = \sum_{\boldsymbol{r}\in\mathbb{Z}^d} \sum_{\boldsymbol{k}\in I_{\boldsymbol{n}}} \hat{g}_{\boldsymbol{k}}\, c_{\boldsymbol{k}+\boldsymbol{n}\odot\boldsymbol{r}}\left(\tilde{\varphi}\right)\, \mathrm{e}^{-2\pi\mathrm{i}(\boldsymbol{k}+\boldsymbol{n}\odot\boldsymbol{r})\boldsymbol{x}} \tag{2.4}$$

mit den diskreten Fourier-Koeffizienten

$$\hat{g}_{\boldsymbol{k}} := \sum_{\boldsymbol{l}\in I_{\boldsymbol{n}}} g_{\boldsymbol{l}}\, \mathrm{e}^{+2\pi\mathrm{i}\boldsymbol{k}\left(\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)}, \quad \boldsymbol{k}\in I_{\boldsymbol{n}}.$$

Ein Vergleich der Summanden in den Gleichungen (2.2) und (2.4) sowie die Annahme, dass $c_{k_t}\left(\tilde{\varphi}_t\right)$ hinreichend klein wird für $|k_t| \geq \frac{1}{2}(n_t - N_t)$, $t = 0, \ldots, d-1$, suggeriert

$$\hat{g}_{\boldsymbol{k}} := \begin{cases} \frac{\hat{f}_{\boldsymbol{k}}}{c_{\boldsymbol{k}}(\tilde{\varphi})} & : \boldsymbol{k}\in I_{\boldsymbol{N}}, \\ 0 & : \boldsymbol{k}\in I_{\boldsymbol{n}}\backslash I_{\boldsymbol{N}}. \end{cases} \tag{2.5}$$

Die Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l}\in I_{\boldsymbol{n}}$, können wir mit einer d-variaten FFT der Größe $n_0\times n_1\times\ldots\times n_{d-1}$ berechnen, d.h.

$$g_{\boldsymbol{l}} = \frac{1}{|I_{\boldsymbol{n}}|} \sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}}\, \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\left(\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)}, \quad \boldsymbol{l}\in I_{\boldsymbol{n}}.$$

Die Approximation (2.3) führt zu einem Alias-Fehler.

2.3.4 Approximation im Ortsbereich

Um die Auswertung von (2.3) zu vereinfachen, ersetzen wir zuerst die periodisierte Funktion $\tilde{\varphi}$ durch φ. Für die Abschätzung des dabei entstehenden Fehlers verweisen wir auf [21]. Wenn die Funktion φ im Ortsbereich $\mathbb{R}^d$ gut lokalisiert ist, kann sie durch eine Funktion

$$\psi\left(\boldsymbol{x}\right) := \varphi\left(\boldsymbol{x}\right)\chi_{[-m\boldsymbol{n}^{-1},m\boldsymbol{n}^{-1}]}\left(\boldsymbol{x}\right)$$

mit kompaktem Träger $\operatorname{supp}\psi \subseteq \left[-m\boldsymbol{n}^{-1}, m\boldsymbol{n}^{-1}\right]$, $m\in\mathbb{N}$, $m\mathbf{1} \ll \boldsymbol{n}$, approximiert werden. Wir definieren die $\mathbf{1}$-periodisierte Funktion $\tilde{\psi} := \sum_{\boldsymbol{r}\in\mathbb{Z}^d}\psi\left(\cdot+\boldsymbol{r}\right)$ und mit ihrer Hilfe die Approximation

$$\breve{s}(\boldsymbol{x}) \approx s\left(\boldsymbol{x}\right) := \sum_{\boldsymbol{l}\in I_{\boldsymbol{n},m}(\boldsymbol{x})} g_{\boldsymbol{l}}\,\tilde{\psi}\left(\boldsymbol{x}-\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right),$$

welche zu einem Abschneidefehler führt.

2.3.5 Der serielle Algorithmus

Zusammenfassend erhalten wir die Approximation

$$f_j := f(\boldsymbol{x}_j) \approx \breve{s}(\boldsymbol{x}_j) \approx s(\boldsymbol{x}_j) =: s_j, \quad j\in\mathcal{M},$$

für die gesuchten Funktionswerte f_j, $j\in\mathcal{M}$, des Ausgangsproblems (2.2) und Algorithmus 2.1 für deren schnelle Berechnung in $\mathcal{O}\left(|I_{\boldsymbol{n}}|\log|I_{\boldsymbol{n}}| + mM\right)$ arithmetischen Operationen.

Algorithmus 2.1 NFFT

Eingabe:

$d \in \mathbb{N}$,	{Dimension}
$M \in \mathbb{N}$,	{Anzahl der Stützstellen}
$\boldsymbol{N} \in 2\mathbb{N}^d$,	{Größe der NFFT}
$\boldsymbol{n} \in 2\mathbb{N}^d$,	{Größe der FFT}
$\boldsymbol{x}_j \in \mathbb{T}^d$, $j \in \mathcal{M}$,	{Stützstellen}
$\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$,	{Fourier-Koeffizienten}

1: Für $\boldsymbol{k} \in I_{\boldsymbol{N}}$ berechne

$$\hat{g}_{\boldsymbol{k}} := \frac{\hat{f}_{\boldsymbol{k}}}{|I_{\boldsymbol{n}}|\, c_{\boldsymbol{k}}\left(\tilde{\varphi}\right)}.$$

2: Für $\boldsymbol{l} \in I_{\boldsymbol{n}}$ berechne mit d-variater FFT

$$g_{\boldsymbol{l}} := \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \hat{g}_{\boldsymbol{k}}\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}.$$

3: Für $j \in \mathcal{M}$ berechne

$$s_j := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}}\, \tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right).$$

Ausgabe:

$s_j \in \mathbb{C}$, $j \in \mathcal{M}$	{Approximation für f_j}

Wir können Algorithmus 2.1 mit $\boldsymbol{s} := (s_j)_{j \in \mathcal{M}} \in \mathbb{C}^M$ in Matrix-Vektor-Notation als

$$\boldsymbol{f} = \boldsymbol{A}\hat{\boldsymbol{f}} \approx \boldsymbol{B}\boldsymbol{F}\boldsymbol{D}\hat{\boldsymbol{f}} = \boldsymbol{s}$$

schreiben, wobei $\boldsymbol{B}$ die reelle und schwach besetzte $M \times |I_{\boldsymbol{n}}|$ Matrix

$$\boldsymbol{B} := \left(\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)\right)_{j \in \mathcal{M};\, \boldsymbol{l} \in I_{\boldsymbol{n}}},$$

$\boldsymbol{F}$ eine Fourier-Matrix der Größe $|I_{\boldsymbol{n}}| \times |I_{\boldsymbol{n}}|$,

$$\boldsymbol{F} := \left(\mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}\right)_{\boldsymbol{k}, \boldsymbol{l} \in I_{\boldsymbol{n}}},$$

und $\boldsymbol{D}$ die reelle und 'diagonale' $|I_{\boldsymbol{n}}| \times |I_{\boldsymbol{N}}|$ Matrix

$$\boldsymbol{D} := \left(\boldsymbol{O}_t \,\middle|\, \operatorname{diag}\left(\frac{1}{|I_{\boldsymbol{n}}| c_{\boldsymbol{k}}\left(\tilde{\varphi}\right)}\right)_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \,\middle|\, \boldsymbol{O}_t\right)^{\mathrm{T}}$$

mit Nullmatrizen $\boldsymbol{O}_t$ der Größe $|I_{\boldsymbol{N}}| \times \frac{|I_{\boldsymbol{n}}| - |I_{\boldsymbol{N}}|}{2}$ ist.

Die Berechnung des adjungierten Matrix-Vektor-Produkts können wir durch

$$\hat{\boldsymbol{h}} = \boldsymbol{A}^{\mathrm{H}}\boldsymbol{f} \approx \boldsymbol{D}^{\top}\boldsymbol{F}^{\mathrm{H}}\boldsymbol{B}^{\top}\boldsymbol{f} =: \hat{\boldsymbol{t}}$$

approximieren, wobei $\hat{\boldsymbol{t}} = (\hat{t}_{\boldsymbol{k}})_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \in \mathbb{C}^{|I_{\boldsymbol{N}}|}$, und wir erhalten somit Algorithmus 2.2 für die adjungierte NFFT.

Algorithmus 2.2 `adjungierte NFFT`

Eingabe:

$d \in \mathbb{N}$,	{Dimension}
$M \in \mathbb{N}$,	{Anzahl der Stützstellen}
$\boldsymbol{N} \in 2\mathbb{N}^d$,	{Größe der NFFT}
$\boldsymbol{n} \in 2\mathbb{N}^d$,	{Größe der FFT}
$\boldsymbol{x}_j \in \mathbb{T}^d$, $j \in \mathcal{M}$,	{Stützstellen}
$f_j \in \mathbb{C}$, $j \in \mathcal{M}$,	{Funktionswerte}

1: Für $\boldsymbol{l} \in I_{\boldsymbol{n}}$ setze

$$g_{\boldsymbol{l}} := 0\,.$$

2: Für $j \in \mathcal{M}$ und $\boldsymbol{l} \in I_{\boldsymbol{n},m}\left(\boldsymbol{x}_j\right)$ berechne

$$g_{\boldsymbol{l}} := g_{\boldsymbol{l}} + f_j\,\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)\,.$$

3: Für $\boldsymbol{k} \in I_{\boldsymbol{N}}$ berechne mit d-variater inverser FFT

$$\hat{g}_{\boldsymbol{k}} := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n}}} g_{\boldsymbol{l}}\,\mathrm{e}^{+2\pi\mathrm{i}\boldsymbol{k}\left(\boldsymbol{n}^{-1}\odot\boldsymbol{l}\right)}.$$

4: Für $\boldsymbol{k} \in I_{\boldsymbol{N}}$ berechne

$$\hat{t}_{\boldsymbol{k}} := \frac{\hat{g}_{\boldsymbol{k}}}{|I_{\boldsymbol{n}}|\,c_{\boldsymbol{k}}(\tilde{\varphi})}.$$

Ausgabe:

$\hat{t}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$	{Approximation für $\hat{h}_{\boldsymbol{k}}$}

2.3.6 Fehlerabschätzungen

Für die Algorithmen 2.1 und 2.2 wurden Fehlerabschätzungen mit verschiedenen Fensterfunktionen in [6, 31, 5, 7] bewiesen. Nach [7] kann der relative Fehler der d-dimensionalen NFFT mit $N_0 = \cdots = N_{d-1}$, $n_0 = \cdots = n_{d-1}$, $\sigma = \frac{n_0}{N_0} > \frac{3}{2}$, und der Gauß-Fensterfunktion abgeschätzt werden durch

$$\frac{|f_j - s_j|}{\sum_{\boldsymbol{k}\in I_{\boldsymbol{N}}} |\hat{f}_{\boldsymbol{k}}|} \leq d2^{d+1}\mathrm{e}^{-m\pi(1-1/(2\sigma-1))}, \quad j \in \mathcal{M}. \tag{2.6}$$

Numerische Tests verschiedener Fensterfunktionen und Parameter σ, m sind in [22] zu finden.

3

Die parallele NFFT

Bei der Parallelisierung eines Algorithmus werden wir mit zwei Problemen konfrontiert. Zum einen müssen die Daten möglichst gleichmäßig auf alle Prozessoren verteilt werden, um dem Speicherbedarf großer Transformationen gerecht zu werden. Zum anderen muss die Abfolge der Berechnungen effizient organisiert werden. Unser Ziel ist die Parallelisierung der NFFT und ihrer Adjungierten, die in den Algorithmen 2.1 und 2.2 dargestellt wurden. Ein zentraler Bestandteil dieser beiden Algorithmen ist der Aufruf einer FFT, deren effiziente Parallelisierung von der verwendeten Rechnerarchitektur abhängt. Es existieren bereits mehrere Programmpakete für die parallele FFT [12, 8, 24], sodass die Idee naheliegt die serielle FFT in den Algorithmen 2.1 und 2.2 durch eine dieser parallelen FFTs zu ersetzen.

Wir werden in diesem Kapitel aufbauend auf einer gegebenen parallelen FFT die parallelen Algorithmen für die NFFT und die adjungierte NFFT mit der Gauß-Fensterfunktion herleiten. Dabei beschränken wir uns auf den dreidimensionalen Fall, d.h. es gilt von nun an $d = 3$. Die Abhängigkeit der Algorithmen von der verwendeten Programmbibliothek für die parallele FFT wird untersucht und es werden Grenzen der effizienten Parallelisierung aufgezeigt. Außerdem geben wir einen Ausblick für zukünftige Verbesserungen der aktuellen Implementation. Das Kapitel schließt mit der Erläuterung der Benutzerschnittstelle unserer parallelen Programmbibliothek für die NFFT.

Mit $\boldsymbol{P} = (P_0, P_1, P_2) \in \mathbb{N}^3$ bezeichnen wir die Anzahl der an der NFFT bezüglich jeder Dimension beteiligten Prozessoren, d.h. es sind $P := P_0 \times P_1 \times P_2$ Prozessoren gegeben, welche bereits auf ein dreidimensionales Gitter aufgeteilt wurden. Jeden Prozessor identifizieren wir im Folgenden mit einem Index $\boldsymbol{p} = (p_0, p_1, p_2) \in \mathcal{P} := \mathbb{Z}^3/\boldsymbol{P}\mathbb{Z}$ und bezeichnen die Faktorgruppe $\mathcal{P}$ als Prozessorennetz. Dabei wählen wir die Repräsentanten $\boldsymbol{p}$ immer im dreidimensionalen Quader $[\boldsymbol{0}, \boldsymbol{P})$.

3.1 Entwurf

3.1.1 Datenparallelisierung der FFT

Sei $\boldsymbol{n} = (n_0, n_1, n_2) \in 2\mathbb{N}^3$ die Größe einer dreidimensionalen FFT. Für gegebene Fourier-Koeffizienten $\hat{g}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}$, können wir mit der FFT die Koeffizienten

$$g_{\boldsymbol{l}} = \sum_{\boldsymbol{k} \in I_{\boldsymbol{n}}} \hat{g}_{\boldsymbol{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \left(\boldsymbol{n}^{-1} \odot \boldsymbol{l} \right)}, \quad \boldsymbol{l} \in I_{\boldsymbol{n}}, \tag{3.1}$$

effizient berechnen.

Des weiteren sei ein Prozessorennetz $\mathcal{P}$ und eine Zerlegung $\{I_{\boldsymbol{n}}^{\boldsymbol{p}} \subseteq I_{\boldsymbol{n}} : \boldsymbol{p} \in \mathcal{P}\}$ der Multi-Indexmenge $I_{\boldsymbol{n}}$ in die lokalen Multi-Indexmengen $I_{\boldsymbol{n}}^{\boldsymbol{p}}$, $\boldsymbol{p} \in \mathcal{P}$, gegeben. D.h. $I_{\boldsymbol{n}}$ ergibt sich als disjunkte Vereinigung

$$I_{\boldsymbol{n}} = \dot{\bigcup_{\boldsymbol{p} \in \mathcal{P}}} I_{\boldsymbol{n}}^{\boldsymbol{p}}.$$

Für einen beliebigen parallelen Algorithmus zur FFT schreiben wir

$$g_{\boldsymbol{l}} = \sum_{\boldsymbol{r} \in \mathcal{P}} \sum_{\boldsymbol{k} \in I_{\boldsymbol{n}}^{\boldsymbol{r}}} \hat{g}_{\boldsymbol{k}} \, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \left(\boldsymbol{n}^{-1} \odot \boldsymbol{l} \right)}, \quad \boldsymbol{l} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}, \tag{3.2}$$

wenn jeder Prozessor $\boldsymbol{p} \in \mathcal{P}$ vor der Transformation die Fourier-Koeffizienten $\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$, besitzen muss und nach der FFT die Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$, erhält. Die Zerlegung der Summe ist nur symbolisch zu verstehen und gibt weder Auskunft über die vom Algorithmus vorgegebene Summationsreihenfolge noch über die benötigte Kommunikation zwischen den Prozessoren.

Durch die Wahl der Programmbibliothek für die FFT wird auch die Art der Zerlegung der Multi-Indexmenge $I_{\boldsymbol{n}}$ festgelegt. Wir gehen im Folgenden davon aus, dass uns ein paralleler Algorithmus und eine Zerlegung $\{I_{\boldsymbol{n}}^{\boldsymbol{p}} \subseteq I_{\boldsymbol{n}} : \boldsymbol{p} \in \mathcal{P}\}$ gegeben sind, welche (3.2) erfüllen. Letztere werden wir verwenden um eine Zerlegung der Indexmengen für die NFFT zu konstruieren.

3.1.2 Datenparallelisierung der NFFT

Die NFFT besitzt im Gegensatz zur FFT zwei verschiedene Indexmengen für die Ein- und Ausgabedaten. Wir benötigen eine Zerlegung $\{\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}} \subseteq I_{\boldsymbol{N}} : \boldsymbol{p} \in \mathcal{P}\}$ der Multi-Indexmenge $I_{\boldsymbol{N}}$ für die Eingangsdaten $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, und eine Zerlegung $\{\mathcal{M}^{\boldsymbol{p}} \subseteq \mathcal{M} : \boldsymbol{p} \in \mathcal{P}\}$ der Indexmenge $\mathcal{M}$ der Ausgangsdaten $s_j \in \mathbb{C}$, $j \in \mathcal{M}$, und schreiben dann analog zu Abschnitt 3.1.1

$$s_j \approx f\left(\boldsymbol{x}_j\right) = \sum_{\boldsymbol{r} \in \mathcal{P}} \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{r}}} \hat{f}_{\boldsymbol{k}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} \boldsymbol{x}_j}, \quad j \in \mathcal{M}^{\boldsymbol{p}}, \tag{3.3}$$

wenn ein beliebiger Prozessor $\boldsymbol{p} \in \mathcal{P}$ die Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, und die Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$, vor einer parallelen NFFT zur Verfügung stellen muss um nach der Transformation die Funktionswerte s_j, $j \in \mathcal{M}^{\boldsymbol{p}}$, zu besitzen. Wir entwickeln nun auf natürliche Weise eine Zerlegung der Indexmengen für die NFFT und einen parallelen Algorithmus, welcher (3.3) erfüllt.

Nach Definition (2.5) sind nur die Fourier-Koeffizienten $\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, ungleich Null. Um die lokalen Eingangsdaten für die FFT bereitstellen zu können, benötigt Prozessor $\boldsymbol{p}$ also nur die Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}} \cap I_{\boldsymbol{n}}^{\boldsymbol{p}}$. Wir setzen

$$\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}} := I_{\boldsymbol{N}} \cap I_{\boldsymbol{n}}^{\boldsymbol{p}} \tag{3.4}$$

und erhalten eine Zerlegung $\{\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}} \subseteq I_{\boldsymbol{n}} : \boldsymbol{p} \in \mathcal{P}\}$ der Multi-Indexmenge $I_{\boldsymbol{N}}$, sodass ein beliebiger Prozessor $\boldsymbol{p} \in \mathcal{P}$ nur die Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, benötigt um die FFT zu initialisieren.

Wir zerlegen den Torus $\mathbb{T}^3$ so in paarweise disjunkte Teilmengen $Q^{\boldsymbol{p}} \subseteq \mathbb{T}^3$, dass für alle $\boldsymbol{x} \in Q^{\boldsymbol{p}}$ gilt $\lfloor \boldsymbol{n} \odot \boldsymbol{x} \rfloor - m\mathbf{1} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$. Die Vorteile dieser Zerlegung werden in Abschnitt 3.2.3 erläutert. Betrachtet man die äquivalenten Formulierungen

$$\lfloor \boldsymbol{n} \odot \boldsymbol{x} \rfloor - m\mathbf{1} \in I_{\boldsymbol{n}}^{\boldsymbol{p}} \quad \Leftrightarrow \quad \lfloor \boldsymbol{n} \odot \boldsymbol{x} \rfloor \in I_{\boldsymbol{n}}^{\boldsymbol{p}} + m\mathbf{1} \quad \Leftrightarrow \quad \lfloor \boldsymbol{x} \rfloor \in (I_{\boldsymbol{n}}^{\boldsymbol{p}} + m\mathbf{1}) \odot \boldsymbol{n}^{-1},$$

so liegt die Definition

$$Q^{\boldsymbol{p}} := (I_{\boldsymbol{n}}^{\boldsymbol{p}} + m\mathbf{1}) \odot \boldsymbol{n}^{-1} + [\mathbf{0}, \mathbf{1})$$

nahe. Wir definieren die zugehörige Indexmenge der lokalen Stützstellen $\mathcal{M}^{\boldsymbol{p}}$ durch

$$\mathcal{M}^{\boldsymbol{p}} := \{j \in \mathcal{M} : \boldsymbol{x}_j \in Q^{\boldsymbol{p}}\} = \{j \in \mathcal{M} : \lfloor \boldsymbol{n} \odot \boldsymbol{x}_j \rfloor - m\mathbf{1} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}\}.$$

Die Anzahl der lokalen Stützstellen ist also durch $|\mathcal{M}^{\boldsymbol{p}}|$ gegeben.

Jedem Prozessor $\boldsymbol{p} \in \mathcal{P}$ ordnen wir die lokalen Eingangsdaten $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, und $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$, zu. Der erste Schritt in Algorithmus 2.1 lässt sich nun leicht parallelisieren, indem wir die Berechnungen für jeden Prozessor auf die lokalen Daten einschränken. Auch die FFT in Schritt 2 wird einfach durch eine parallele FFT ersetzt. Dies ist ohne Weiteres möglich, da die geforderte Datenverteilung der parallelen FFT nach Konstruktion der lokalen Multi-Indexmengen $\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, $\boldsymbol{p} \in \mathcal{P}$, bereits nach dem ersten Schritt vorliegt. Zur Berechnung von Schritt 3 müssen sich jeweils für ein $j \in \mathcal{M}^{\boldsymbol{p}}$ alle Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$, auf einem Prozessor $\boldsymbol{p}$ befinden. Dies ist aber nach der parallelen FFT nicht zwingend der Fall. Es müssen also Daten zwischen den Prozessoren ausgetauscht werden um den dritten Schritt zu ermöglichen.

In $I_{\boldsymbol{n},m}^{\boldsymbol{p}} \subseteq I_{\boldsymbol{n}}$ fassen wir die Multi-Indizes $\boldsymbol{l} \in I_{\boldsymbol{n}}$ aller Koeffizienten $g_{\boldsymbol{l}}$ zusammen, welche von Prozessor $\boldsymbol{p}$ für Schritt 3 in Algorithmus 2.1 benötigt werden, d.h.

$$I_{\boldsymbol{n},m}^{\boldsymbol{p}} := \bigcup_{j \in \mathcal{M}^{\boldsymbol{p}}} I_{\boldsymbol{n},m}(\boldsymbol{x}_j). \tag{3.5}$$

Es müssen also für jeden Prozessor $\boldsymbol{p} \in \mathcal{P}$ die Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n},m}^{\boldsymbol{p}} \setminus I_{\boldsymbol{n}}^{\boldsymbol{p}}$, durch Kommunikation mit den restlichen Prozessoren zur Verfügung gestellt werden. Zu einem gegebenen Multi-Index $\boldsymbol{l} \in I_{\boldsymbol{n}}$ fassen wir alle Prozessoren, welche den Koeffizient $g_{\boldsymbol{l}}$ für Schritt 3 in Algorithmus 2.1 benötigen, in der Menge

$$\mathcal{P}_{\boldsymbol{l}} := \left\{\boldsymbol{r} \in \mathcal{P} : \boldsymbol{l} \in I_{\boldsymbol{n},m}^{\boldsymbol{r}}\right\},$$

zusammen und bezeichnen mit $\boldsymbol{p}_{\boldsymbol{l}} \in \mathcal{P}$ den eindeutig bestimmten Prozessor mit $\boldsymbol{l} \in I_{\boldsymbol{n}}^{\boldsymbol{p}_{\boldsymbol{l}}}$. Der Prozessor $\boldsymbol{p}_{\boldsymbol{l}}$ besitzt also bereits nach der parallelen FFT den Koeffizienten $g_{\boldsymbol{l}}$ und wird diesen an alle Prozessoren $\boldsymbol{p} \in \mathcal{P}_{\boldsymbol{l}}$ verteilen.

3.1.3 Der parallele Algorithmus

Wir sind nun in der Lage Algorithmus 3.1 für die parallele NFFT zu formulieren, welcher auf jedem Prozessor $\boldsymbol{p} \in \mathcal{P}$ ausgeführt wird.

Algorithmus 3.1 `parallele 3D-NFFT`

Eingabe:

$\lvert\mathcal{M}^{\boldsymbol{p}}\rvert \in \mathbb{N}_0$,	{Anzahl der lokalen Stützstellen}
$\boldsymbol{N} \in 2\mathbb{N}^3$,	{Größe der NFFT}
$\boldsymbol{n} \in 2\mathbb{N}^3$,	{Größe der FFT}
$\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$,	{Lokale Stützstellen}
$\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$,	{Lokale Fourier-Koeffizienten}
$\mathcal{P}_{\boldsymbol{l}}, \boldsymbol{p}_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}$,	{Daten zur Kommunikation}

1: Für $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$ berechne

$$\hat{g}_{\boldsymbol{k}} := \frac{\hat{f}_{\boldsymbol{k}}}{\lvert I_{\boldsymbol{n}}\rvert\, c_{\boldsymbol{k}}\left(\tilde{\varphi}\right)}.$$

2: Für $\boldsymbol{l} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$ berechne mit paralleler 3D-FFT

$$g_{\boldsymbol{l}} := \sum_{\boldsymbol{r} \in \mathcal{P}} \sum_{\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{r}}} \hat{g}_{\boldsymbol{k}}\, \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}.$$

3: Für $\boldsymbol{l} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$ sende $g_{\boldsymbol{l}}$ an alle Prozessoren $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{l}}$.

4: Für $\boldsymbol{l} \in I_{\boldsymbol{n},m}^{\boldsymbol{p}}$ empfange $g_{\boldsymbol{l}}$ von Prozessor $\boldsymbol{p}_{\boldsymbol{l}}$.

5: Für $j \in \mathcal{M}^{\boldsymbol{p}}$ berechne

$$s_j := \sum_{\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}}\, \tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right).$$

Ausgabe:

$s_j \in \mathbb{C}$, $j \in \mathcal{M}^{\boldsymbol{p}}$	{Approximation für f_j}

Die Parallelisierung der adjungierten NFFT nach Algorithmus 2.2 beruht auf den gleichen Zerlegungen der Indexmengen $I_{\boldsymbol{N}}$ und $\mathcal{M}$. Für die parallele Formulierung der Schritte 1 und 2 müssen wir beachten, dass ein Prozessor $\boldsymbol{r} \in \mathcal{P}$ nur die Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{r}}$, besitzt. Wir definieren daher für $\boldsymbol{l} \in I_{\boldsymbol{n}}$ die Summen

$$g_{\boldsymbol{l}}^{\boldsymbol{r}} := \sum_{j \in \mathcal{M}^{\boldsymbol{r}}} f_j \tilde{\psi}(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}), \quad \boldsymbol{r} \in \mathcal{P},$$

welche auf Prozessor $\boldsymbol{r}$ zu berechnen sind. Auf Grund des kompakten Trägers von $\tilde{\psi}$ gilt $g_{\boldsymbol{l}}^{\boldsymbol{r}} = 0$, falls $\boldsymbol{r} \notin \mathcal{P}_{\boldsymbol{l}}$. Also können wir die Summe zur Berechnung der Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}$, zerlegen in

$$g_{\boldsymbol{l}} = \sum_{j \in \mathcal{M}} f_j \tilde{\psi}(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}) = \sum_{\boldsymbol{r} \in \mathcal{P}} \sum_{j \in \mathcal{M}^{\boldsymbol{r}}} f_j \tilde{\psi}(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}) = \sum_{\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{l}}} g_{\boldsymbol{l}}^{\boldsymbol{r}}.$$

Dies spiegelt sich in den Schritten 1 bis 5 von Algorithmus 3.2 für die parallele adjungierte NFFT wider. Die Schritte 6 und 7 ergeben sich direkt aus den Schritten 3 und 4 von Algorithmus 2.2 unter Beachtung der parallelen Datenverteilung.

Algorithmus 3.2 `parallele adjungierte 3D-NFFT`

Eingabe:

$|\mathcal{M}^{\boldsymbol{p}}| \in \mathbb{N}_0$, {Anzahl der lokalen Stützstellen}
$\boldsymbol{N} \in 2\mathbb{N}^3$, {Größe der NFFT}
$\boldsymbol{n} \in 2\mathbb{N}^3$, {Größe der FFT}
$\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, {Lokale Stützstellen}
$f_j \in \mathbb{C}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, {Lokale Funktionswerte}
$\mathcal{P}_{\boldsymbol{l}}, \boldsymbol{p}_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}$, {Daten zur Kommunikation}

1: Für $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m}$ setze

$$g^{\boldsymbol{p}}_{\boldsymbol{l}} := 0\,.$$

2: Für $j \in \mathcal{M}^{\boldsymbol{p}}$ und $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$ berechne

$$g^{\boldsymbol{p}}_{\boldsymbol{l}} := g^{\boldsymbol{p}}_{\boldsymbol{l}} + f_j\,\tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)\,.$$

3: Für $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m}$ sende $g^{\boldsymbol{p}}_{\boldsymbol{l}}$ an Prozessor $\boldsymbol{p}_{\boldsymbol{l}}$.

4: Für $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n}}$ empfange $g^{\boldsymbol{r}}_{\boldsymbol{l}}$ von allen Prozessoren $\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{l}}$.

5: Für $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n}}$ berechne

$$g_{\boldsymbol{l}} := \sum_{\boldsymbol{r} \in \mathcal{P}_{\boldsymbol{l}}} g^{\boldsymbol{r}}_{\boldsymbol{l}}$$

6: Für $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$ berechne mit paralleler inverser 3D-FFT

$$\hat{g}_{\boldsymbol{k}} := \sum_{r \in \mathcal{P}} \sum_{\boldsymbol{l} \in I^{\boldsymbol{r}}_{\boldsymbol{n}}} g_{\boldsymbol{l}}\, \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{k}\left(\boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)}.$$

7: Für $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$ berechne

$$\hat{t}_{\boldsymbol{k}} := \frac{\hat{g}_{\boldsymbol{k}}}{|I_{\boldsymbol{n}}|\, c_{\boldsymbol{k}}(\tilde{\varphi})}.$$

Ausgabe:

$\hat{t}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$ {Approximation für $\hat{h}_{\boldsymbol{k}}$}

3.2 Details zur Implementation

3.2.1 Initialisierung der Daten

Die Zuordnung der lokalen Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}} \in \mathbb{C}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$, und der lokalen Stützstellen $\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, zu den Prozessoren $\boldsymbol{p} \in \mathcal{P}$ bleibt dem Nutzer überlassen. Er muss daher bereits vor der Ausführung von Algorithmus 3.1 die Multi-Indexmengen $\mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$, $\boldsymbol{p} \in \mathcal{P}$, und die Indexmengen $\mathcal{M}^{\boldsymbol{p}}$, $\boldsymbol{p} \in \mathcal{P}$, kennen. Wir führen zuerst die häufig in parallelen Programmpaketen für die FFT verwendete Zerlegung der Multi-Indexmenge $I_{\boldsymbol{n}}$ in Blöcke ein und leiten aus ihr eine kompakte Beschreibung der Indexmengen $\mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$ und $\mathcal{M}^{\boldsymbol{p}}$, $\boldsymbol{p} \in \mathcal{P}$, durch wenige an den Nutzer zu übergebende Parameter ab.

Datenverteilung mit Blockstruktur

Für $\boldsymbol{n} \in 2\mathbb{N}^3$ seien für alle Prozessoren $\boldsymbol{p} \in \mathcal{P}$ durch die FFT ein Multi-Offset $\boldsymbol{o^p} \in I_{\boldsymbol{n}}$ und eine Multi-Länge $\boldsymbol{n^p} \in [\boldsymbol{0}, \boldsymbol{n}] \cap \mathbb{Z}^3$ derart gegeben, dass wir eine Zerlegung der Multi-Indexmenge $I_{\boldsymbol{n}}$ in die Blöcke $I_{\boldsymbol{n}}^{\boldsymbol{p}} := [\boldsymbol{o^p}; \boldsymbol{n^p})_{\boldsymbol{n}}$ erhalten. Dann bezeichnen wir die Menge $\{I_{\boldsymbol{n}}^{\boldsymbol{p}} : \boldsymbol{p} \in \mathcal{P}\}$ als eine Zerlegung der Multi-Indexmenge $I_{\boldsymbol{n}}$ mit Blockstruktur.

Beispiel 3.1. *Abbildung 3.1 zeigt eine solche Blockverteilung der Multi-Indexmenge $I_{\boldsymbol{n}}$ mit $\boldsymbol{n} = (8, 4, 4)$ auf 8 Prozessoren, welche durch ihren Index im Prozessorennetz gekennzeichnet sind. Das Prozessorennetz wurde als $\mathcal{P} = \{\boldsymbol{p} = (p_0, p_1, p_2) : p_0, p_1, p_2 \in \{0, 1\}\}$, die Multi-Offsets als $\boldsymbol{o^p} = (4(p_0-1), 2(p_1-1), 2(p_2-1))$, $\boldsymbol{p} \in \mathcal{P}$, und die Multi-Längen als $\boldsymbol{n^p} = (4, 2, 2)$, $\boldsymbol{p} \in \mathcal{P}$, gewählt.*

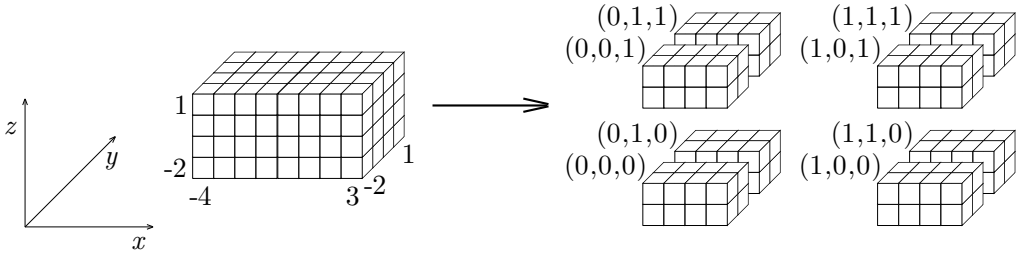


Abbildung 3.1: Beispiel einer Zerlegung mit Blockstruktur.

Sei $\boldsymbol{p} = (p_0, p_1, p_2) \in \mathcal{P}$. Setzen wir für $t = 0, 1, 2$

$$O_t^{p_t} = \begin{cases} o_t^{p_t} & : o_t^{p_t} \in I_{N_t}, \\ -\frac{N_t}{2} & : o_t^{p_t} \notin I_{N_t}, \end{cases}$$

und

$$N_t^{p_t} = \left| \, [o_t^{p_t}; n_t^{p_t})_{n_t} \cap I_{N_t} \, \right|,$$

so ermöglichen der Multi-Offset $\boldsymbol{O^p} = (O_0^{p_0}, O_1^{p_1}, O_2^{p_2}) \in I_{\boldsymbol{N}}$ und die Multi-Länge $\boldsymbol{N^p} = (N_0^{p_0}, N_1^{p_1}, N_2^{p_2}) \in [\boldsymbol{0}, \boldsymbol{N}] \cap \mathbb{Z}^3$ die Darstellung

$$[\boldsymbol{O^p}; \boldsymbol{N^p})_{\boldsymbol{N}} = I_{\boldsymbol{N}} \cap I_{\boldsymbol{n}}^{\boldsymbol{p}} =: \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}.$$

Wir erhalten also eine Zerlegung $\{\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}} \subseteq I_{\boldsymbol{N}} : \boldsymbol{p} \in \mathcal{P}\}$ der Multi-Indexmenge $I_{\boldsymbol{N}}$ mit Blockstruktur.

Setzen wir für alle $\boldsymbol{p} \in \mathcal{P}$ den Vektor $\boldsymbol{B^p} \in \mathbb{T}^3$ als

$$\boldsymbol{B^p} = (\boldsymbol{o^p} + m\boldsymbol{1}) \odot \boldsymbol{n}^{-1},$$

und $\boldsymbol{L^p} \in [0, 1]^3$ als

$$\boldsymbol{L^p} = \boldsymbol{n^p} \odot \boldsymbol{n}^{-1},$$

so ergibt sich

$$[\boldsymbol{B^p}; \boldsymbol{L^p})_{\mathbb{T}^d} = (I_{\boldsymbol{n}}^{\boldsymbol{p}} + m\boldsymbol{1}) \odot \boldsymbol{n}^{-1} + [\boldsymbol{0}, \boldsymbol{1}) =: Q^{\boldsymbol{p}}.$$

Wir erhalten also eine Zerlegung $\{Q^{\boldsymbol{p}} \subseteq \mathbb{T}^3 : \boldsymbol{p} \in \mathcal{P}\}$ des Torus $\mathbb{T}^3$ mit Blockstruktur.

Initialisierungsalgorithmus

Die Multi-Offsets $\boldsymbol{O^p}$, $\boldsymbol{B^p}$, $\boldsymbol{p} \in \mathcal{P}$, und die Multi-Längen $\boldsymbol{N^p}$, $\boldsymbol{L^p}$, $\boldsymbol{p} \in \mathcal{P}$, erhält der Nutzer des Programms durch den Initialisierungsalgorithmus 3.3 übergeben, welcher auf jedem Prozessor $\boldsymbol{p} \in \mathcal{P}$ aufgerufen wird.

Algorithmus 3.3 `Initialisierung`

Eingabe:
$\boldsymbol{N} \in 2\mathbb{N}^3$, {Größe der NFFT}
$\boldsymbol{n} \in 2\mathbb{N}^3$ mit $\boldsymbol{N} \leq \boldsymbol{n}$, {Größe der FFT}

1: Erhalte $\boldsymbol{o^p} \in I_{\boldsymbol{n}},\ \boldsymbol{n^p} \in [\boldsymbol{0}, \boldsymbol{n}] \cap \mathbb{Z}^3$ von der FFT Initialisierung.

2: Berechne
$$\boldsymbol{B^p} := (\boldsymbol{o^p} + m\boldsymbol{1}) \odot \boldsymbol{n}^{-1}\,.$$

3: Berechne
$$\boldsymbol{L^p} := \boldsymbol{n^p} \odot \boldsymbol{n}^{-1}\,.$$

4: Für $t = 0, 1, 2$ setze
$$O_t^{p_t} = \begin{cases} o_t^{p_t} & : o_t^{p_t} \in I_{N_t}, \\ -\frac{N_t}{2} & : o_t^{p_t} \notin I_{N_t}. \end{cases}$$

5: Für $t = 0, 1, 2$ berechne
$$N_t^{p_t} = \big|\, [o_t^{p_t}; n_t^{p_t})_{n_t} \cap I_{N_t} \,\big|.$$

Ausgabe:
$\boldsymbol{B^p} \in \mathbb{T}^3$, {Offset des Quaders $Q^{\boldsymbol{p}}$}
$\boldsymbol{L^p} \in [0, 1]^3$, {Länge des Quaders $Q^{\boldsymbol{p}}$}
$\boldsymbol{O^p} \in I_{\boldsymbol{N}}$, {Offset des Blocks $\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$}
$\boldsymbol{N^p} \in [\boldsymbol{0}, \boldsymbol{N}] \cap \mathbb{Z}^3$, {Länges des Blocks $\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$}

3.2.2 Eingabeformate der NFFT und FFT

In den folgenden Erläuterungen zu den Eingabeformaten der NFFT und der FFT gehen wir davon aus, dass ein dreidimensionales Feld $(a_{\boldsymbol{k}})_{\boldsymbol{k} \in [\boldsymbol{0}, \boldsymbol{N}) \cap \mathbb{Z}^3}$ als $(a_{\tilde{k}})_{\tilde{k} \in [0, N_0 N_1 N_2) \cap \mathbb{Z}}$ eindimensional mit Hilfe des linearisierten Index $\tilde{k} := k_2 + N_2 k_1 + N_2 N_1 k_0$ im Speicher hinterlegt wird. Der linearisierte Index muss immer erst im Moment des Speicherzugriffes berechnet werden, während sonst mit den drei Komponenten k_0, k_1, k_2 des Multi-Index $\boldsymbol{k}$ gearbeitet wird. Zur Vereinfachung werden wir deshalb einen Multi-Index $[\boldsymbol{k}]_{\mathsf{NFFT}} = ([k_0]_{\mathsf{NFFT}}, [k_1]_{\mathsf{NFFT}}, [k_2]_{\mathsf{NFFT}}) \in [\boldsymbol{0}, \boldsymbol{N}) \cap \mathbb{Z}^3$ mit dem linearisierten Index $[k_2]_{\mathsf{NFFT}} + N_2[k_1]_{\mathsf{NFFT}} + N_2 N_1 [k_0]_{\mathsf{NFFT}}$ und analog einen Multi-Index $[\boldsymbol{k}]_{\mathsf{FFT}} = ([k_0]_{\mathsf{FFT}}, [k_1]_{\mathsf{FFT}}, [k_2]_{\mathsf{FFT}}) \in [\boldsymbol{0}, \boldsymbol{n}) \cap \mathbb{Z}^3$ mit dem linearisierten Index $[k_2]_{\mathsf{FFT}} + n_2[k_1]_{\mathsf{FFT}} + n_2 n_1 [k_0]_{\mathsf{FFT}}$ identifizieren.

Sei ein Multi-Index $\boldsymbol{k} = (k_0, k_1, k_2) \in I_{\boldsymbol{N}}$ gegeben. Wir ordnen diesem den eindeutig bestimmten Multi-Index $[\boldsymbol{k}]_{\mathsf{NFFT}} = ([k_0]_{\mathsf{NFFT}}, [k_1]_{\mathsf{NFFT}}, [k_2]_{\mathsf{NFFT}}) \in [\boldsymbol{0}, \boldsymbol{N}) \cap \mathbb{Z}^3$ zu, welcher den

Speicherort von $\hat{f}_{\boldsymbol{k}}$ im Eingabeformat der NFFT angibt. Dann gilt für $t = 0, 1, 2$

$$[k_t]_{\mathsf{NFFT}} = k_t + \frac{N_t}{2}, \tag{3.6a}$$

$$k_t = [k_t]_{\mathsf{NFFT}} - \frac{N_t}{2}. \tag{3.6b}$$

Die meisten FFT Programmbibliotheken verwenden ein anderes Eingabeformat als die NFFT. Ordnen wir einem Multi-Index $\boldsymbol{l} = (l_0, l_1, l_2) \in I_{\boldsymbol{n}}$ den eindeutig bestimmten Multi-Index $[\boldsymbol{l}]_{\mathsf{FFT}} = ([l_0]_{\mathsf{FFT}}, [l_1]_{\mathsf{FFT}}, [l_2]_{\mathsf{FFT}}) \in [\boldsymbol{0}, \boldsymbol{n}) \cap \mathbb{Z}^3$ zu, welcher den Speicherort von $\hat{g}_{\boldsymbol{l}}$ im Eingabeformat der FFT angibt, so gilt für $t = 0, 1, 2$

$$[l_t]_{\mathsf{FFT}} = \begin{cases} l_t + n_t & : -\frac{n_t}{2} \leq l_t < 0, \\ l_t & : 0 \leq l_t < \frac{n_t}{2}, \end{cases}$$

$$l_t = \begin{cases} [l_t]_{\mathsf{FFT}} - n_t & : \frac{n_t}{2} \leq [l_t]_{\mathsf{FFT}} < n_t, \\ [l_t]_{\mathsf{FFT}} & : 0 \leq [l_t]_{\mathsf{FFT}} < \frac{n_t}{2}, \end{cases}$$

und somit ebenfalls für $t = 0, 1, 2$

$$[l_t]_{\mathsf{FFT}} = (l_t + n_t) \bmod n_t, \tag{3.7a}$$

$$l_t = \left([l_t]_{\mathsf{FFT}} + \frac{n_t}{2}\right) \bmod n_t - \frac{n_t}{2}. \tag{3.7b}$$

Um die Zuweisung in Schritt 1 von Algorithmus 2.1 ausführen zu können, benötigen wir für einen vorgegebenen Speicherort $[\boldsymbol{k}]_{\mathsf{NFFT}}$ von $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, den Index $[\boldsymbol{k}]_{\mathsf{FFT}}$ des Speicherortes von $\hat{g}_{\boldsymbol{k}}$. Es gilt für $t = 0, 1, 2$

$$[k_t]_{\mathsf{FFT}} \overset{(3.7a)}{=} (k_t + n_t) \bmod n_t \overset{(3.6b)}{=} \left([k_t]_{\mathsf{NFFT}} - \frac{N_t}{2} + n_t\right) \bmod n_t, \tag{3.8}$$

und somit können wir die Indizes für die Speicherung der Fourier-Koeffizienten aus dem Eingabeformat der NFFT in das Eingabeformat der FFT gemäß (3.8) transformieren. Die Rücktransformation ist für $\boldsymbol{k} \in I_{\boldsymbol{N}}$ durch folgende Gleichung möglich. Es gilt für $t = 0, 1, 2$

$$[k_t]_{\mathsf{NFFT}} \overset{(3.6a)}{=} k_t + \frac{N_t}{2} \overset{(3.7b)}{=} \left([k_t]_{\mathsf{FFT}} + \frac{n_t}{2}\right) \bmod n_t - \frac{n_t}{2} + \frac{N_t}{2}. \tag{3.9}$$

3.2.3 Kommunikation

Für die Berechnung von Schritt 3 in Algorithmus 2.1 benötigt Prozessor $\boldsymbol{p}$ gemäß (3.5) die Koeffizienten

$$g_{\boldsymbol{l}}, \quad \boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m} = \bigcup_{j \in \mathcal{M}^{\boldsymbol{p}}} I_{\boldsymbol{n},m}(\boldsymbol{x}_j).$$

Wir müssten also die Vereinigung der Mengen $I_{\boldsymbol{n},m}(\boldsymbol{x}_j)$, $j \in \mathcal{M}^{\boldsymbol{p}}$, für jede Wahl von Stützstellen $\boldsymbol{x}_j \in \mathbb{T}^3$, $j \in \mathcal{M}$, neu bestimmen. Um diese Abhängigkeit der Menge $I^{\boldsymbol{p}}_{\boldsymbol{n},m}$, und somit auch des parallelen Algorithmus, von der konkreten Wahl der Stützstellen zu vermeiden stellen wir stattdessen die Koeffizienten

$$g_{\boldsymbol{l}}, \quad \boldsymbol{l} \in \bigcup_{\boldsymbol{x} \in Q^{\boldsymbol{p}}} I_{\boldsymbol{n},m}(\boldsymbol{x}) = \bigcup_{\boldsymbol{x} \in (I^{\boldsymbol{p}}_{\boldsymbol{n}} + m\boldsymbol{1}) \odot \boldsymbol{n}^{-1}} I_{\boldsymbol{n},m}(\boldsymbol{x})$$

durch Kommunikation der Prozessoren zur Verfügung. Nach der Konstruktion der Indexmenge $\mathcal{M}^{\boldsymbol{p}}$ gilt $\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$ für alle $j \in \mathcal{M}^{\boldsymbol{p}}$. Deshalb folgern wir

$$\bigcup_{j \in \mathcal{M}^{\boldsymbol{p}}} I_{\boldsymbol{n},m}\left(\boldsymbol{x}_j\right) \subseteq \bigcup_{\boldsymbol{x} \in Q^{\boldsymbol{p}}} I_{\boldsymbol{n},m}\left(\boldsymbol{x}\right),$$

d.h. ein beliebiger Prozessor $\boldsymbol{p}$ erhält somit wirklich alle Koeffizienten, die er für Schritt 3 in Algorithmus 2.1 benötigt.

Wir wollen nun die Wahl der Zerlegung des Torus $\mathbb{T}^3$ in die Blöcke $Q^{\boldsymbol{p}} \subseteq \mathbb{T}^3$ begründen. Es stellt sich die Frage, welcher Prozessor eine Stützstelle $\boldsymbol{x} \in \mathbb{T}^3$ erhalten soll um die Kommunikationsschritte 3 und 4 aus Algorithmus 3.1 möglichst effizient ausführen zu können.

Die Zerlegung des Torus $\mathbb{T}^3$ in die Blöcke $Q^{\boldsymbol{p}} \subseteq \mathbb{T}^3$ ist derart gewählt, dass der erste Koeffizient $g_{\boldsymbol{l}}$, $\boldsymbol{l} = \lfloor \boldsymbol{n} \odot \boldsymbol{x} \rfloor - m\mathbf{1}$, der Summe

$$s_j = \sum_{\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_j)} g_{\boldsymbol{l}}\, \tilde{\psi}\left(\boldsymbol{x}_j - \boldsymbol{n}^{-1} \odot \boldsymbol{l}\right)$$

aus Schritt 5 in Algorithmus 3.1 bereits für alle $\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, auf Prozessor $\boldsymbol{p}$ zur Verfügung steht. Daher muss $\boldsymbol{p}$ bezüglich jeder Dimension nur mit seinem direkten Nachfolger kommunizieren um alle Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m}$, zu erhalten, nicht aber mit seinen Vorgängern. Die versendete Datenmenge kann dabei zwar nicht reduziert werden, aber die Wartezeit zum Aufbau der Kommunikationen wird geringer. Den Austausch der benötigten Daten realisieren wir durch einen Sendevorgang im Ring, der bezüglich jeder Dimension durchgeführt wird.

Beispiel 3.2. *Abbildung 3.2 illustriert das dreimalige Senden im Ring mit einem Abschneideparameter $m = 1$. Es müssen also jeweils $2m + 1 = 3$ Scheiben bezüglich jeder Dimension versendet werden. Die grauen Zellen stehen für die nach der FFT bereits vorhandenen Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n}}$, eines Prozessors $\boldsymbol{p} \in \mathcal{P}$, während die weißen Zellen Schrittweise durch das Senden in x-, y- und z-Richtung hinzukommen bis alle Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m}$, für Prozessor $\boldsymbol{p}$ zur Verfügung stehen.*

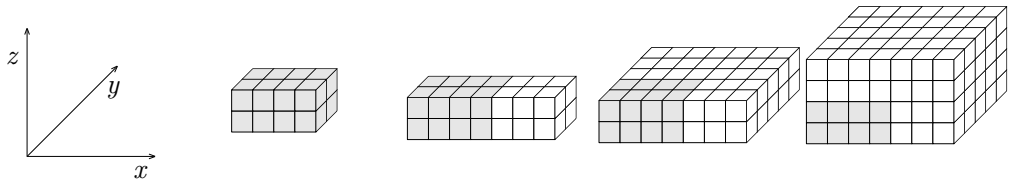


Abbildung 3.2: Dreidimensionales Senden der Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m}$.

3.2.4 Sortierung der Stützstellen

Die Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}$, sind im Allgemeinen nicht sortiert. Es kann also vorkommen, dass zwei Stützstellen $\boldsymbol{x}_{j_1}$ und $\boldsymbol{x}_{j_2}$ existieren mit $I_{\boldsymbol{n},m}(\boldsymbol{x}_{j_1}) = I_{\boldsymbol{n},m}(\boldsymbol{x}_{j_2})$, die zugehörigen Werte s_{j_1} und s_{j_2} aber nicht aufeinanderfolgend berechnet werden. Dies führt insbesondere zu Zeiteinbußen, wenn alle Koeffizienten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n},m}(\boldsymbol{x}_{j_1})$, welche zur Berechnung von s_{j_1} benötigt wurden, bereits aus dem Cache gelöscht wurden, bevor sie wiederum zur Berechnung

von s_{j_2} benötigt werden. Wir wirken diesem mehrfachen Laden von vielfach genutzten Koeffizienten entgegen, indem die Stützstellen sortiert werden, bevor Schritt 5 in Algorithmus 3.1 ausgeführt wird. Um dabei die vom Nutzer übergebene Anordnung der Stützstellen nicht zu verlieren, legen wir ein Feld der Länge $2|\mathcal{M}^{\boldsymbol{p}}|$ an. In diesem Feld werden für $j \in \mathcal{M}^{\boldsymbol{p}}$ die Paare $(\lfloor \boldsymbol{n} \odot \boldsymbol{x}_j \rfloor - m\mathbf{1}, j) \in I_{\boldsymbol{n}} \times \mathcal{M}^{\boldsymbol{p}}$ gespeichert und anschließend nach ersterem Index sortiert. Die Sortierung der Multi-Indizes führen wir zuerst bezüglich der ersten, dann bezüglich der zweiten und schließlich bezüglich der dritten Komponente durch. Wollen wir nun Schritt 5 in Algorithmus 3.1 ausführen, so berechnen wir die Werte s_j, $j \in \mathcal{M}^{\boldsymbol{p}}$, in der Reihenfolge der zweiten Indizes. Somit werden Stützstellen, welche in Algorithmus 3.1 viele Koeffizienten gemeinsam benötigen auch direkt hintereinander bearbeitet. Die Reihenfolge der übergebenen Stützstellen bleibt dabei erhalten, da nur die Abfolge der Bearbeitung der Stützstellen geändert wird. Es zeigt sich, dass durch die Verbesserung der Cache-Zugriffe mehr Zeit gespart wird, als wir für die Sortierung der Stützstellen benötigen.

3.2.5 Die FFT Programmbibliothek

Wir verwenden für die parallele FFT die renommierte FFTW [12]. In der Version 3.3alpha1 dieses Programmpakets wird eine parallele FFT zur Verfügung gestellt, welche hohe Leistung und Kompatibilität mit vielen Rechnerarchitekturen in sich vereinigt.

Die parallele FFTW basiert auf einer eindimensionalen Datenaufteilung mit Blockstruktur. Durch den Aufruf einer Initialisierungsfunktion erhält er auf jedem Prozessor $\boldsymbol{p} \in \mathcal{P}$ den Multi-Offset $\boldsymbol{o}^{\boldsymbol{p}} \in I_{\boldsymbol{n}}$ im Eingabeformat der FFT, welches in Abschnitt 3.2.2 beschrieben wurde, und die Multi-Länge $\boldsymbol{n}^{\boldsymbol{p}} \in [\mathbf{0}, \boldsymbol{n}] \cap \mathbb{Z}^3$. Das Verteilen der Daten auf die Prozessoren muss in den Blöcken $[\boldsymbol{o}; \boldsymbol{n}^{\boldsymbol{p}})_{\boldsymbol{n}}$ erfolgen und ist dem Nutzer überlassen. Die Multi-Offsets $\boldsymbol{o}^{\boldsymbol{p}} = (o_0^{p_0}, o_1^{p_1}, o_2^{p_2})$, $\boldsymbol{p} \in \mathcal{P}$, und die Multi-Längen $\boldsymbol{n}^{\boldsymbol{p}} = (n_0^{p_0}, n_1^{p_1}, n_2^{p_2})$, $\boldsymbol{p} \in \mathcal{P}$, sind nur bezüglich der ersten Dimension variabel. Für die anderen Dimensionen gilt $o_1^{p_1} = -\frac{n_1}{2}$, $o_2^{p_2} = -\frac{n_2}{2}$, $n_1^{p_1} = n_1$ und $n_2^{p_2} = n_2$. Es werden also immer ganze Scheiben der Größe $n_1 \times n_2$ an die Prozessoren verteilt. Die Anzahl der bezüglich jeder Dimension beteiligten Prozessoren ist demnach durch $\boldsymbol{P} = (P_0, 1, 1)$ gegeben und das Prozessorennetz durch

$$\mathcal{P} = \{(p_0, 0, 0) : p_0 = 0, \ldots, P_0 - 1\}.$$

Beispiel 3.3. *Abbildung 3.3 zeigt eine Zerlegung der Multi-Indexmenge $I_{\boldsymbol{n}}$ mit Blockstruktur für $\boldsymbol{n} = (8, 4, 4)$ und 8 Prozessoren. Die Prozessoren werden durch die erste Komponente p_0 ihres Index $(p_0, 1, 1) \in \mathcal{P}$ eindeutig gekennzeichnet.*

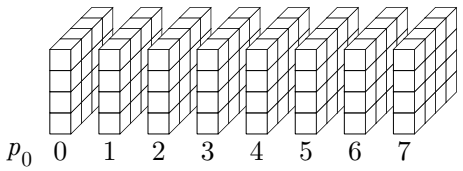


Abbildung 3.3: Zerlegung von $I_{\boldsymbol{n}}$ mit Blockstruktur für $\boldsymbol{n} = (8, 4, 4)$ und $\boldsymbol{P} = (8, 1, 1)$ bezüglich der ersten Dimension.

Nach der Initialisierung der Eingangsdaten berechnet die parallele FFTW zuerst die zweidimensionalen FFTs bezüglich der letzten beiden Dimensionen. Die benötigten Daten stehen

nach der Initialisierung bereits lokal jedem Prozessor zur Verfügung. Um die eindimensionalen FFTs bezüglich der ersten Dimension berechnen zu können ist eine Umverteilung der Daten notwendig. Es erfolgt wiederum eine eindimensionale Datenaufteilung mit Blockstruktur, wobei diesmal die zweite Dimension der Mulit-Offsets und der Multi-Längen variabel ist. Der Nutzer kann entscheiden, ob diese Umordnung der Daten nach der Transformation wieder rückgängig gemacht werden soll.

Offenbar muss die Größe der FFT bezüglich der ersten beiden Dimensionen mindestens der Anzahl der verwendeten Prozessoren entsprechen, damit eine gute Lastverteilung möglich ist. Natürlich kann das Feld der Eingangsdaten auch vor der FFT transponiert werden um die Größe der FFT bezüglich der ersten beiden Dimensionen zu maximieren. Aber auch dann ist die Anzahl der Prozessoren beschränkt durch

$$P_{\max} := \max\left\{ \min\{n_0, n_1\}, \min\{n_1, n_2\}, \min\{n_2, n_0\}\right\}. \tag{3.10}$$

Beispiel 3.4. *Abbildung 3.4 zeigt eine Zerlegung der Multi-Indexmenge $I_{\boldsymbol{n}}$ mit Blockstruktur für $\boldsymbol{n} = (8, 4, 4)$ und 8 Prozessoren bezüglich der zweiten Dimension. Die Prozessoren werden durch die erste Komponente p_0 ihres Index $(p_0, 1, 1) \in \mathcal{P}$ eindeutig gekennzeichnet. Da die Größe der FFT bezüglich der zweiten Dimension nur 4 beträgt, können auch nur 4 Prozessoren genutzt werden um die verbleibenden eindimensionalen FFTs bezüglich der ersten Dimension zu berechnen.*

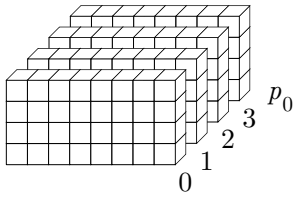


Abbildung 3.4: Zerlegung von $I_{\boldsymbol{n}}$ mit Blockstruktur für $\boldsymbol{n} = (8, 4, 4)$ und $\boldsymbol{P} = (8, 1, 1)$ bezüglich der zweiten Dimension.

3.2.6 Speicherbedarf

Wir schätzen nun den benötigten Speicherbedarf der seriellen NFFT und der parallelen NFFT ab. Zur Vereinfachung der Notation wollen wir dabei die Größen der NFFT und der FFT bezüglich jeder Dimension mit $N := N_0 = N_1 = N_2 \in 2\mathbb{N}$ und $n := n_0 = n_1 = n_2 \in 2\mathbb{N}$ gleichsetzen. Außerdem sei ein eindimensionales Prozessorennetz $\mathcal{P} = \mathbb{Z}^3/\boldsymbol{P}\mathbb{Z}$ mit $\boldsymbol{P} = (P_0, 1, 1)$ gegeben. Die Anzahl der an der parallelen NFFT beteiligten Prozessoren ergibt sich also durch $P = P_0 \times 1 \times 1$. Für große Ganzzahlen verwenden wir analog zur FFTW den C-Standarddatentyp `ptrdiff_t` um große Transformationen auf 64-Bit-Systemen einfach zu ermöglichen. Er entspricht dem Datentyp `int` auf 32-Bit-Systemen und dem Datentyp `long` auf 64-Bit-Systemen.

Die Tabelle 3.1 listet die größten Felder auf, welche zum Ausführen der NFFT benötigt werden, und den für sie erforderlichen Speicherplatz. Es wird außerdem der Datentyp eines jeden Eintrags dieses Feldes angegeben. Neben den aus Algorithmus 2.1 bekannten Eingangs- und Ausgangsdaten wurde auch der Speicherbedarf für die Sortierung der Stützstellen gemäß Abschnitt 3.2.4 aufgenommen. Für die Ausgangsdaten der FFT $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}$, wird kein

Name	Datentyp	Größe	Speicherbedarf in Byte
$\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$	`double complex`	N^3	$16N^3$
$\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}$	`double complex`	n^3	$16n^3$
$g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}$	`double complex`	n^3	0
f_j, $j \in \mathcal{M}$	`double complex`	M	$16M$
$\boldsymbol{x}_j$, $j \in \mathcal{M}$	`double`	$3M$	$24M$
Sortierung	`ptrdiff_t`	$2M$	$16M$

Tabelle 3.1: Speicherbedarf der größten Felder in der seriellen NFFT.

Name	Datentyp	Größe	Speicherbedarf in Byte
$\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$	`double complex`	$(n/P)N^2$	$16nN^2/P$
$\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$	`double complex`	n^3/P	$16n^3/P$
$g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n},m}^{\boldsymbol{p}}$	`double complex`	$(n/P + 2m + 1)n^2$	$16(2m+1)n^2$
f_j, $j \in \mathcal{M}$	`double complex`	M/P	$16M/P$
$\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$	`double`	$3M/P$	$24M/P$
Sortierung	`ptrdiff_t`	$2M/P$	$16M/P$

Tabelle 3.2: Speicherbedarf der größten Felder in der parallelen NFFT.

zusätzlicher Speicher benötigt, da die Eingangsdaten $\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}$, überschrieben werden. Jede Stützstelle $\boldsymbol{x}_j$ besitzt 3 Koordinaten vom Datentyp `double`, weshalb wir ein Feld der Größe $3|\mathcal{M}| = 3M$ anlegen müssen. Eine Variable vom Datentyp `double complex` benötigt 16 Byte, eine Variable vom Datentyp `double` 8 Byte und eine Variable vom Datentyp `ptrdiff_t` in 64 Bit benötigt 16 Byte Speicherplatz.

Tabelle 3.2 listet die größten Felder und deren Speicherbedarf für die parallele NFFT auf. Wir gehen von einer gleichmäßigen Verteilung sowohl der Fourier-Koeffizienten $\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}$, als auch der Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}$, auf die Prozessoren $\boldsymbol{p} \in \mathcal{P}$ aus. Dann werden einem Prozessor maximal n/P Scheiben der Größe N^2 bei der Aufteilung der Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{N}}$, und n/P Scheiben der Größe n^2 bei der Aufteilung der Fourier-Koeffizienten $\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I_{\boldsymbol{n}}$, zugewiesen. Bei der parallelen FFT werden ebenfalls die Eingangsdaten überschrieben, allerdings muss zusätzlicher Speicher für die in Schritt 4 aus Algorithmus 3.1 empfangenen Daten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n},m} \setminus I_{\boldsymbol{n}}^{\boldsymbol{p}}$, bereit gestellt werden. Wir müssen also zusätzlich $2m+1$ Scheiben der Größe n^2 vom Datentyp `double complex` speichern. Für eine gleichmäßige Verteilung der Stützstellen auf die Prozessoren können wir $|\mathcal{M}^{\boldsymbol{p}}| = M/P$ für alle Prozessoren $\boldsymbol{p} \in \mathcal{P}$ annehmen und erhalten die restlichen drei Feldgrößen aus Tabelle 3.2 analog wie bei der seriellen NFFT indem wir die Anzahl der Stützstellen M durch die Anzahl der lokalen Stützstellen M/P ersetzen.

Es ergibt sich für die serielle NFFT also insgesamt ein Speicherbedarf von

$$8(2N^3 + 2n^3 + 7M) \text{ Byte}, \tag{3.11}$$

wohingegen die parallele NFFT

$$8(2nN^2 + 2n^3 + 7M)/P + 2(2m+1)n^2 \text{ Byte} \tag{3.12}$$

an Speicher auf jedem Prozessor benötigt.

Für $n = 2N$ und $M = N^3$ ergibt sich also mit Hilfe von Formel (3.11) ein Speicherbedarf von

$$200N^3 \text{ Byte} \tag{3.13}$$

für die serielle NFFT. Formel (3.12) liefert mit $n = 2N$ und $M = N^3$ und dem Abschneideparameter $m = 4$ einen Speicherbedarf von

$$8(27N^3/P + 72N^2) \text{ Byte} \tag{3.14}$$

für die parallele NFFT. Dabei ist zu beachten, dass die Anzahl der Prozessoren P auf Grund der FFTW gemäß Formel (3.10) durch $P \leq P_{\max} = n = 2N$ beschränkt ist.

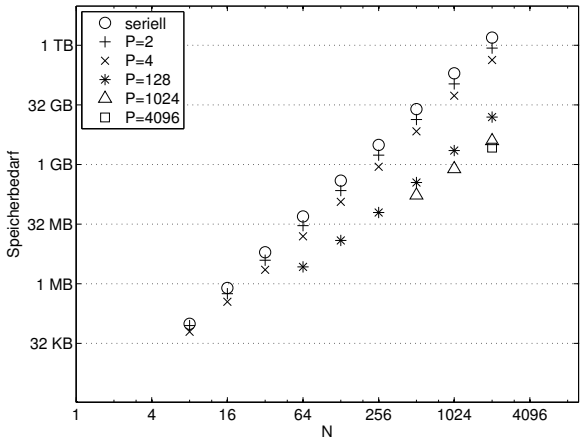


Abbildung 3.5: Benötigter Speicherplatz in Abhängigkeit der Größe der NFFT.

In Abbildung 3.5 wird der Speicherbedarf der NFFT für die Größen $N = 2^j$, $j = 3, \ldots, 11$, zwischen dem seriellen Algorithmus und dem parallelen Algorithmus für ausgewählte Prozessorenanzahlen P verglichen. Es ist zu sehen, dass wir z.B. für eine NFFT der Größe $2048 \times 2048 \times 2048$ den Speicherbedarf auf minimal $2,67$ GByte für jeden Prozessor senken können, indem wir 4096 Prozessoren verwenden. Eine weitere Erhöhung der Prozessoren kann nicht zur Reduzierung des Speicherbedarfs dienen. Mit den Formeln (3.13) und (3.14) können wir also für eine fixierte Problemgröße N und einen durch die Rechnerarchitektur vorgegebenen nutzbaren Arbeitsspeicher pro Prozessor abschätzen, welche Anzahl an Prozessoren mindestens nötig ist, um die NFFT ausführen zu können. Sei also der verfügbare Arbeitsspeicher pro Prozessor Z Byte groß, $Z \in \mathbb{R}_+$. Wir erhalten mit Formel (3.14) für den benötigten Speicherplatz

$$8(27N^3/P + 72N^2) \leq Z, \tag{3.15}$$

und durch Umstellen nach P

$$P \geq \frac{216N^3}{Z - 576N^2}. \tag{3.16}$$

Indem wir die Anzahl der Prozessoren mit $P_{\max} = 2N$ maximal wählen, minimieren wir nach Formel (3.14) den benötigten Speicher pro Prozessor. Einsetzen von $P_{\max} = 2N$ in Formel (3.15) und Umstellen nach N ergibt dann

$$N \leq \sqrt{\frac{Z}{684}}, \tag{3.17}$$

also die maximal mögliche Transformationsgröße $N_{\max}$ einer parallelen NFFT unter Verwendung der FFTW auf einem Parallelrechner mit Z Byte Arbeitsspeicher pro Prozessor. Dabei

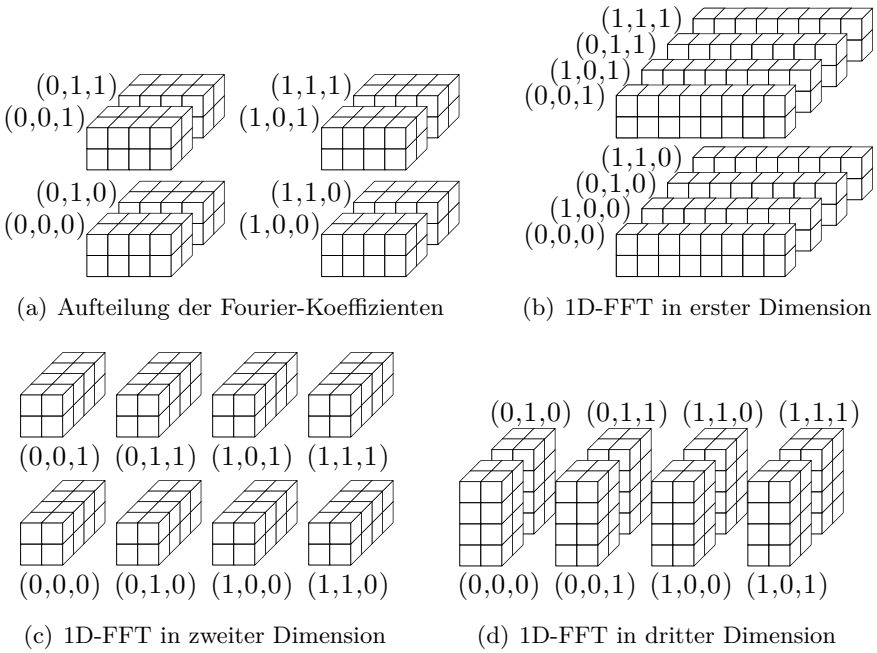
nehmen wir an, die dazu benötigte Anzahl von Prozessoren stehe zur Verfügung. Mit Tabelle 3.3 können wir erkennen, dass eine Verwendung von mehr als 5008 Prozessoren auf vielen aktuellen Parallelrechnern durch den mangelnden Speicherplatz verhindert wird.

Arbeitsspeicher in GByte	0,5	1	2	4
$N_{\max}$	884	1.252	1.770	2.504
$P_{\max}$	1.768	2.504	3.540	5.008

Tabelle 3.3: Maximale Transformationsgröße $N_{\max}$ und Prozessorenanzahl $P_{\max}$ der parallelen NFFT mit FFTW in Abhängigkeit vom verfügbaren Arbeitsspeicher.

3.2.7 Alternative FFT Programmbibliothek

Eine Alternative zur von uns verwendeten FFTW bietet die von IBM speziell für die Parallelrechnerarchitektur BlueGene L entwickelte schnelle Fourier-Transformation (BGFFT) [9]. In diesem Algorithmus werden die Daten bezüglich aller drei Dimensionen aufgeteilt und dann dreimal umverteilt um jeweils eindimensionale FFTs entlang der x-, der y- und schließlich der z-Richtung zu berechnen, vgl. Abbildung 3.6.



(a) Aufteilung der Fourier-Koeffizienten

(b) 1D-FFT in erster Dimension

(c) 1D-FFT in zweiter Dimension

(d) 1D-FFT in dritter Dimension

Abbildung 3.6: Datenaufteilung der BGFFT für $\boldsymbol{n} = (8, 4, 4)$ und $\boldsymbol{P} = (2, 2, 2)$.

Die Anzahl der Prozessoren wird daher durch $\tilde{P}_{\max} := \min\{n_0 n_1, n_1 n_2, n_2 n_0\}$ von oben beschränkt, d.h. es können oftmals mehr Prozessoren zum Einsatz kommen als bei Benutzung der FFTW und somit der Speicherbedarf pro Prozessor und die Rechenzeit verkleinert werden. Beispielsweise kann bereits ab einer Transformationsgröße der NFFT von $128 \times 128 \times 128$ eine Prozessorenanzahl von 65.536 genutzt werden. Dies entspricht der Gesamtprozessorenanzahl

des Jülicher Höchstleistungsrechners JUGENE. Die folgenden Vergleiche des Speicherplatzbedarfs der NFFT unter Verwendung der BGFFT mit dem der NFFT unter Verwendung der FFTW sollen eine weitere Motivation für die Bevorzugung der BGFFT geben.

Zunächst müssen wir erneut den Speicherplatzbedarf der größten Felder der NFFT abschätzen. Wir wollen zur Vereinfachung der Notation die Größen der NFFT und der FFT bezüglich jeder Dimension mit $N := N_0 = N_1 = N_2 \in 2\mathbb{N}$ und $n := n_0 = n_1 = n_2 \in 2\mathbb{N}$ gleichsetzen. Außerdem sei ein dreidimensionales Prozessorennetz $\mathcal{P} = \mathbb{Z}^3/\boldsymbol{P}\mathbb{Z}$ mit $\boldsymbol{P} = (p, p, p)$, $p \in \mathbb{N}$, gegeben. Daher ergibt sich die Gesamtanzahl der an der NFFT beteiligten Prozessoren durch $P = p^3$. Die Tabelle 3.4 listet die größten Felder auf, welche zum Ausführen der NFFT gebraucht werden, und den benötigten Speicherplatz um sie anlegen zu können. Bei der BGFFT werden die Eingangsdaten nicht überschrieben und es muss zusätzlicher Speicher für die in Schritt 4 aus Algorithmus 3.1 empfangenen Daten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n},m} \setminus I^{\boldsymbol{p}}_{\boldsymbol{n}}$, verwendet werden. Die Ein- und Ausgabedaten der FFT werden auf die Prozessoren möglichst gleichmäßig verteilt. Die restlichen Felder ergeben sich analog zu denen in Tabelle 3.2 unter Beachtung der dreidimensionalen Datenaufteilung.

Name	Datentyp	Größe	Speicherbedarf in Byte
$\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}^{\boldsymbol{p}}_{\boldsymbol{N}}$	`double complex`	$(n/p)^3$	$16n^3/p^3$
$\hat{g}_{\boldsymbol{k}}$, $\boldsymbol{k} \in I^{\boldsymbol{p}}_{\boldsymbol{n}}$	`double complex`	$(n/p)^3$	$16n^3/p^3$
$g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I^{\boldsymbol{p}}_{\boldsymbol{n},m}$	`double complex`	$(n/p + 2m + 1)^3$	$16(n/p + 2m + 1)^3$
f_j, $j \in \mathcal{M}$	`double complex`	M/p^3	$16M/p^3$
$\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$	`double`	$3M/p^3$	$24M/p^3$
Sortierung	`ptrdiff_t`	$2M/p^3$	$16M/p^3$

Tabelle 3.4: Speicherbedarf der größten Felder in der parallelen NFFT mit BGFFT.

Wir benötigen also für die parallele NFFT unter Verwendung der BGFFT

$$8\left(4n^3 + 7M\right)/p^3 + 16(n/p + 2m + 1)^3 \text{ Byte}$$

an Arbeitsspeicher. Setzen wir wieder die Größe der FFT $n = 2N$, die Anzahl der Stützstellen $M = N^3$ und den Abschneideparameter $m = 4$, so erhalten wir einen Speicherbedarf von

$$312N^3/p^3 + 16\left(2N/p + 9\right)^3 \text{ Byte.} \tag{3.18}$$

Dieser wird wiederum minimal, wenn die Anzahl der Prozessoren p maximal gewählt wird. Der Vorteil der BGFFT gegenüber der FFTW ist, dass die Anzahl der verwendbaren Prozessoren P nicht durch die Größe der FFT n bezüglich der ersten Dimension, sondern durch $P = p^3 \leq n^2$ nach oben beschränkt wird. Da $p \in \mathbb{N}$ und $n = 2N$ gilt, können wir maximal $p_{\max} = \lfloor \sqrt[3]{4N^2} \rfloor$ Prozessoren bezüglich jeder Dimension für die NFFT verwenden. Wegen der unstetigen Abhängigkeit der Prozessorenanzahl $p_{\max}$ von der Transformationsgröße N, schätzen wir $p_{\max} = \lfloor \sqrt[3]{4N^2} \rfloor > \sqrt[3]{4N^2} - 1$ nach unten ab und erhalten aus Formel (3.18) eine obere Schranke für den Speicherbedarf der mit der BGFFT implementierten parallelen NFFT bei maximaler Prozessorenanzahl von

$$\frac{312N^3}{(\sqrt[3]{4N^2} - 1)^3} + 16\left(\frac{2N}{\sqrt[3]{4N^2} - 1} + 9\right)^3 \text{ Byte.}$$

Anhand dieser Formel können wir eine Abschätzung für die maximale Transformationsgröße $N_{\max} \in 2\mathbb{N}$ der NFFT bezüglich jeder Dimension für eine vorgegebene Größe des verfügbaren Arbeitsspeichers pro Prozessor bestimmen.

Arbeitsspeicher in GByte	0,5	1	2	4
$N_{\max}$	4.697.886	9.472.260	19.065.154	38.320.792
P	$8,83 \cdot 10^{13}$	$3,59 \cdot 10^{14}$	$1,45 \cdot 10^{15}$	$5,87 \cdot 10^{15}$

Tabelle 3.5: Abschätzung der maximalen Transformationsgröße $N_{\max}$ und die benötigte Prozessorenanzahl P für die parallele NFFT mit BGFFT in Abhängigkeit vom verfügbaren Arbeitsspeicher.

Tabelle 3.5 zeigt, dass sich die Größe des Arbeitsspeichers eines jeden Prozessors nun bei weitem nicht mehr so einschränkend auf die maximale Transformationsgröße $N_{\max}$ auswirkt. Wir weisen aber darauf hin, dass sie nur erreicht werden kann, wenn auch die maximale Anzahl an Prozessoren $P = p_{\max}^3$ verwendet wird. Dies ist bei den Größenordnungen der Prozessorenanzahlen in Tabelle 3.5 mit dem heutigen Stand der Technik noch nicht zu erreichen. Wir geben zum Abschluss noch ein realistisches Beispiel.

Beispiel 3.5. *Der Höchstleistungsrechner JUGENE aus dem Forschungszentrum Jülich besitzt 65.536 Prozessoren mit jeweils 0,5 GByte Arbeitsspeicher. Wählen wir die Anzahl der Prozessoren bezüglich aller drei Dimensionen gleich, können wir maximal $40^3 = 64.000$ Prozessoren nutzen. Die maximale Transformationsgröße einer NFFT erhalten wir durch Gleichsetzen von Formel* (3.18) *mit 2^{29} Byte. Durch Lösung der resultierenden kubischen Gleichung mit $P = 64.000$ nach N erhalten wir als maximale Transformationsgröße bezüglich jeder Dimension $N = 4.230$.*

3.3 Benutzerschnittstelle

Die Benutzerschnittstelle der parallelen NFFT orientiert sich stark an denen der FFTW [12] und der NFFT [19]. Am folgenden Quelltext 3.1 erläutern wir die Nutzung der parallelen NFFT.

Die Konstante `d` vom Datentyp `int` gibt die Dimension der NFFT an. Sie ist nur in Hinblick auf eine einfache Erweiterung der Bibliothek auf parallele NFFTs beliebiger Dimension und zur besseren Lesbarkeit bereits in den Quelltext aufgenommen worden. Wir setzen immer `d=3`, da unsere Algorithmen nur für den dreidimensinalen Fall zutreffen.

Zuerst rufen wir die Initialisierungsfunktionen `MPI_Init(int *argc, char ***argv)` und `fftw_mpi_init()` auf, um die Arbeit mit den parallelen Bibliotheken zu ermöglichen. Die Variable `np_fftw` vom Typ `int` soll die Anzahl der Prozessoren enthalten, welche für die FFTW verwendet werden. Wir weisen ihr in Zeile 12 die Anzahl aller verfügbaren Prozessoren zu, um möglichst alle Prozessoren für die FFTW zu verwenden. Die Felder `N` und `n` vom Datentyp `ptrdiff_t` bekommen die Größen der NFFT und der FFT bezüglich aller Dimensionen zugewiesen. Die Anzahl der Stützstellen wird in der Variablen `M` vom Datentyp `ptrdiff_t` gespeichert. Wir setzen sie in Zeile 16 gleich der Anzahl der Fourier-Koeffizienten für die NFFT. Mit Hilfe der Funktion

```
void nfft_mpi_init_guru(nfft_mpi_plan *ths, int d, ptrdiff_t *N,
     ptrdiff_t *n, int m, int *np_fftw, unsigned nfft_flags,
     unsigned fftw_flags);
```

Quelltext 3.1: Anwendung der parallelen NFFT.

```
1  #include <nfft-mpi.h>
2
3  int main(int argc, char **argv) {
4    const int d = 3;
5    ptrdiff_t M, N[d], n[d], local_x_num, local_x_start, i, j, k;
6    int np_fftw, m = 4;
7    nfft_mpi_plan myplan;
8
9    MPI_Init(&argc, &argv);
10   fftw_mpi_init();
11
12   MPI_Comm_size(MPI_COMM_WORLD, &np_fftw);
13
14   N[0] = N[1] = N[2] = 8;
15   n[0] = 2*N[0]; n[1] = 2*N[1]; n[2] = 2*N[2];
16   M = N[0]*N[1]*N[2];
17
18   /** init nfft_par */
19   nfft_mpi_init_guru(&myplan, d, N, n, m, &np_fftw, MALLOC_F_HAT|
        FFTW_INIT, FFTW_MEASURE);
20
21   /** count local nodes with some function
        user_counts_local_nodes(lower_border, upper_border) */
22   local_x_num = user_counts_local_nodes(myplan.lower_border,
        myplan.upper_border);
23
24   /** init local nodes */
25   nfft_mpi_init_nodes(&myplan, &local_x_num, &local_x_start, MALLOC_F|
        MALLOC_X, 0);
26
27   /** init local Fourier-coeff. to some function users_f_hat(i, j, k) */
28   for(i = myplan.local_N0_start; i < myplan.local_N0_start +
        myplan.local_N0; i++)
29     for(j = 0; j< N[1]; j++)
30       for(k = 0; k < N[2]; k++)
31         myplan.f_hat[k*(N[2]*(j+N[1]*i))] = users_f_hat( i%N[0], j, k);
32
33   /** init local nodes to some function users_x(i) */
34   for(i = 0; i < local_x_num; i++)
35     for(j = 0; j < d; j++)
36       myplan.x[d*i+j] = users_x(d*i+j);
37
38   /** execute par NFFT */
39   nfft_trafo_par3d(&myplan);
40
41   /** destroy plan and free allocated memory */
42   nfft_mpi_finalize(&myplan, FREE_F_HAT| FREE_F| FREE_X);
43
44   fftw_mpi_cleanup();
45   MPI_Finalize();
46 }
```

initialisieren wir eine Variable `myplan` des Typs `nfft_mpi_plan`. In dieser werden alle notwendigen Informationen für die parallele NFFT gespeichert. Für den Nutzer relevante Elemente eines solchen Plans werden in Tabelle 3.6 zusammengefasst.

Bezeichnung	Element des Plans	Datentyp
$(\hat{f}_{\boldsymbol{k}})_{\boldsymbol{k}\in\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}}$	`f_hat`	`double complex*`
$(f_j)_{j\in\mathcal{M}^{\boldsymbol{p}}}$	`f`	`double complex*`
$(\boldsymbol{x}_j)_{j\in\mathcal{M}^{\boldsymbol{p}}}$	`local_x`	`double*`
$M^{\boldsymbol{p}}$	`local_x_num`	`ptrdiff_t`
	`local_x_num_start`	`ptrdiff_t`
$\boldsymbol{N}$	`N`	`ptrdiff_t*`
$\boldsymbol{n}$	`n`	`ptrdiff_t*`
$O_0^{p_0}$	`local_N0_start`	`ptrdiff_t`
$N_0^{p_0}$	`local_N0`	`ptrdiff_t`
	`lower_border`	`double`
	`upper_border`	`double`

Tabelle 3.6: Ausgewählte Elemente eines Plans vom Typ `nfft_mpi_plan`.

Analog zur NFFT [19] werden durch `nfft_flags` Optionen zur Initialisierung des Plans übergeben. Die Option `FFTW_INIT` bewirkt, dass die FFTW initialisiert und aller benötigter Speicher für die FFT allokiert wird. Durch `MALLOC_F_HAT` wird der Speicher für die lokalen Fourier-Koeffizienten $f_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, reserviert. Im Gegensatz zur seriellen NFFT [19] können die Optionen `MALLOC_F` und `MALLOC_X` zur Reservierung des Speichers für die lokalen Ausgabedaten s_j, $j \in \mathcal{M}^{\boldsymbol{p}}$, und die lokalen Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$, an dieser Stelle noch nicht gewählt werden, da die Aufteilung der lokalen Stützstellen und damit auch die Größe des benötigten Speicherplatzes noch nicht bekannt sind. Die aus der seriellen NFFT [19] bekannten Optionen zur Vorberechnung der Fensterfunktion und auch die Option `FFT_OUT_OF_PLACE` sind derzeit nicht implementiert. Die Optionen zum Planen der FFTW werden mit der Variable `fftw_flags` übergeben und können aus der Dokumentation der FFTW [13] entnommen werden.

Innerhalb der Initialisierung wird überprüft, ob die FFTW mit der von uns gewählten Anzahl `np_fftw` von Prozessoren ausführbar ist. Anderenfalls wird `np_fftw` verkleinert um den Beschränkungen der FFTW gerecht zu werden. Die Elemente `local_N0_start` und `local_N0` des Plans `myplan` bekommen durch die Initialisierungsfunktion den Offset $O_0^{p_0}$ und die Länge $N_0^{p_0}$ zugewiesen. Außerdem werden die beiden Elemente `lower_border` und `upper_border` vom Typ `double` berechnet. Sie sind die Grenzen eines 1-periodischen Intervalls auf dem Torus $\mathbb{T}$, welches bei `lower_border` beginnt, bis `upper_border` reicht und die erste Komponente aller lokalen Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$, enthält. Genauer gilt $\boldsymbol{x}_j = (x_{j0}, x_{j1}, x_{j2}) \in Q^{\boldsymbol{p}}$ für $\texttt{lower\_border} \leq \texttt{upper\_border}$ genau dann, wenn

$$x_{j0} \in [\texttt{lower\_border}, \texttt{upper\_border})$$

und für $\texttt{upper\_border} < \texttt{lower\_border}$ genau dann, wenn

$$x_{j0} \in \left[\texttt{lower\_border}, \tfrac{1}{2}\right) \cup \left[-\tfrac{1}{2}, \texttt{upper\_border}\right).$$

Der Nutzer hat nun die Aufgabe die Anzahl der lokalen Stützstellen zu bestimmen, d.h. die Anzahl der Stützstellen deren erste Komponenten in besagtem 1-periodischen Intervall zwi-

schen `lower_border` und `upper_border` liegen, und sie mit der Variablen `local_x_num` vom Typ `ptrdiff_t` an die Initialisierungsfunktion

```
void nfft_mpi_init_nodes(nfft_mpi_plan *ths, ptrdiff_t *local_x_num,
     ptrdiff_t *local_x_start, unsigned nfft_flags,
     unsigned nfft_finalize_flags);
```

zu übergeben. Diese Funktion gibt, sofern wir die Optionen `FREE_F` und `FREE_X` über `nfft_finalize_flags` gesetzt haben, den Speicher der Felder `f` und `local_x` frei und allokiert anschließend, sofern die Optionen `MALLOC_X` und `MALLOC_F` über `nfft_flags` gesetzt sind, ein Feld der Länge `local_x_num` vom Datentyp `double complex` für `f` und ein Feld der Länge `3*local_x_num` vom Datentyp `double` für `local_x`. Bei erstmaliger Benutzung setzen wir `nfft_finalize_flags=0`, da noch kein Speicher für `f` oder `local_x` reserviert wurde. Wollen wir hingegen zu einem späteren Zeitpunkt die Menge der Stützstellen austauschen und eine erneute parallele NFFT der gleichen Größe durchführen, so müssen wir lediglich die neue Anzahl der lokalen Stützstellen `local_x_num` bestimmen und abermals den Funktionsaufruf

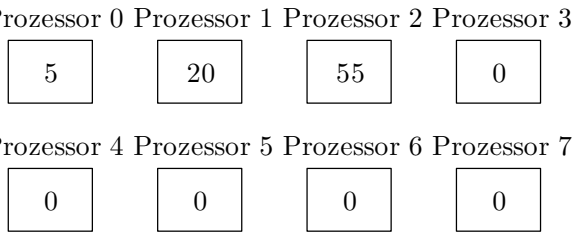
```
nfft_mpi_init_nodes(&myplan, &local_x_num, &local_x_start, MALLOC_F|
    MALLOC_X, FREE_F| FREE_X);
```

starten. Es wird dann automatisch der alte Speicher freigegeben und der neue allokiert. Es ist zu beachten, dass wir nicht nochmals die Initialisierungsfunktion `nfft_mpi_init_guru` aufrufen müssen und damit die zeitaufwendige Planung der FFTW nur vor der ersten Transformation anfällt.

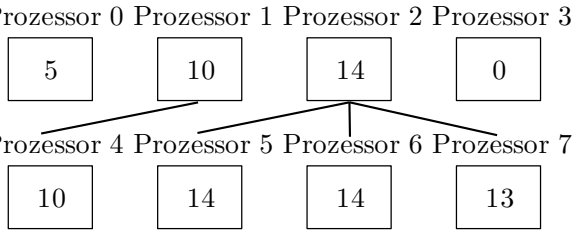
Schränken wir uns bei der Berechnung der parallelen FFT auf einen Teil $\mathcal{P}_{\mathrm{FFT}} \subseteq \mathcal{P}$ der verfügbaren Prozessoren $\mathcal{P}$ ein, kann auch Algorithmus 3.1 für die parallele NFFT nur diesen Teil der Prozessoren nutzen. Um dennoch zumindest für einen Teil der parallelen NFFT alle Prozessoren zu verwenden, fassen wir die Prozessoren in $|\mathcal{P}_{\mathrm{FFT}}|$ Gruppen zusammen, von denen jede die lokalen Stützstellen $\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, eines Prozessors $\boldsymbol{p} \in \mathcal{P}_{\mathrm{FFT}}$ unter sich aufteilt. Nach Schritt 2 in Algorithmus 3.1 werden die lokalen Ausgabedaten $g_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{n}}^{\boldsymbol{p}}$, der FFT an alle Prozessoren einer Gruppe versandt. Schritt 3 wird dann von allen Prozessoren gemeinsam berechnet.

Da sich bei der Verteilung die Anzahl der lokalen Stützstellen für einige Prozessoren ändert, wird `local_x_num` als Zeiger an die Initialisierungsfunktion `nfft_mpi_init_nodes` übergeben. Ordnet man alle lokalen Stützstellen einer Gruppe in einem eindimensionalen Feld an, so gibt für jeden Prozessor $\boldsymbol{p} \in \mathcal{P}$ die Variable `local_x_start` die Nummer der ersten Stützstelle an, welche $\boldsymbol{p}$ zugeordnet wurde. Sie dient zur einfachen Aufteilung eines Feldes auf die Prozessoren einer Gruppe. Werden alle Prozessoren für die FFTW verwendet, so gilt für jeden Prozessor `local_x_start=0` und wir können sie ignorieren.

Beispiel 3.6. *Abbildung 3.7(a) zeigt ein System von 8 Prozessoren und die Anzahl der lokalen Stützstellen. Es wurden nur 4 Prozessoren für die FFT verwendet, deshalb haben auch nur diese durch Algorithmus 3.1 Stützstellen zugewiesen bekommen. In Abbildung 3.7(b) ist die Verteilung der Stützstellen auf die übrigen Prozessoren illustriert, wobei Prozessoren einer gemeinsamen Gruppe mit Strichen verbunden sind. Die zugehörigen Offsets* `local_x_start` *und die resultierenden Anzahlen der lokalen Stützstellen werden in Tabelle 3.7 zusammengefasst.*



(a) Ungleichmäßige Stützstellenverteilung



(b) Verteilung der Stützstellen auf alle Prozessoren.

Abbildung 3.7: Beispiel zur Stützstellenzuweisung.

Prozessor	0	1	2	3	4	5	6	7
`local_x_start`	0	0	0	0	10	14	28	42
`local_x_num`	5	10	14	0	10	14	14	13

Tabelle 3.7: Offsets der Stützstellenmengen für Abb. 3.7(b).

Nun füllen wir die Felder für die lokalen Fourier-Koeffizienten und die lokalen Stützstellen mit den entsprechenden Daten. Das Feld der lokalen Fourier-Koeffizienten $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, wird analog zur Datenaufteilung der parallelen FFTW [12] durch den Offset `local_N0_start` und die Länge `local_N0` beschrieben. Für ein gegebenes globales Feld `f_hat` von Fourier-Koeffizienten der Länge $N_0 \times N_1 \times N_2$ muss also jeder Prozessor $\boldsymbol{p} \in \mathcal{P}$ die Fourier-Koeffizienten `f_hat[k+N[2]*(j+N[1]*(i%N[0]))]` mit

```
local_N0_start  <=  i  <  local_N0_start + local_N0
      0         <=  j  <             n[1]
      0         <=  k  <             n[2]
```

erhalten. Da der Fall `local_N0_start + local_N0` $\geq$ `N[0]` eintreten kann, ist die Anwendung der Modulo-Funktion auf den Index `i` notwendig. Dadurch wird das Feld der Fourier-Koeffizienten bei einer Feldüberschreitung periodisch fortgesetzt.

Die Stützstellen werden wie bereits bei der Bestimmung von `local_x_num` beschrieben so verteilt, dass die erste Komponente einer jeden lokalen Stützstelle $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$, im 1-periodischen Intervall auf $\mathbb{T}$ zwischen `lower_border` und `upper_border` liegt. Die Verteilung könnte durch einen parallelen Sortieralgorithmus vorgenommen werden.

Anschließend führen wir die parallele NFFT mit der Anweisung

```
void nfft_trafo_par3d(nfft_mpi_plan *ths);
```

aus.

Den reservierten Speicher der parallelen NFFT geben wir mit der Funktion

```
void nfft_mpi_finalize(nfft_mpi_plan *ths, unsigned nfft_finalize_flags);
```

frei. Dabei werden die Felder `f_hat`, `f` und `local_x` nur gelöscht, wenn die entsprechenden Optionen `FREE_F_HAT`, `FREE_F` oder `FREE_X` mit `nfft_finalize_flags` übergeben werden. Anschließend wird aller verwendeter Speicher für die FFTW freigegeben und MPI beendet.

Die adjungierte parallele NFFT wird völlig analog aufgerufen. Wir müssen nur beachten, dass die Werte f_j, $j \in \mathcal{M}$, zusammen mit den Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}$, den Prozessoren zugeordnet werden. Jeder Prozessor erhält also wie in Algorithmus 3.2 beschrieben alle Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}^{\boldsymbol{p}}$, und die zugehörigen Werte f_j, $j \in \mathcal{M}^{\boldsymbol{p}}$. Auch die Variablen `local_x_start` und `local_x_num` sind für die Verteilung dieser Funktionswerte analog zu gebrauchen. Wir rufen die parallele adjungierte NFFT mit der Anweisung

```
void nfft_adjoint_par3d(nfft_mpi_plan *ths);
```

auf. Quelltext 3.2 zeigt ein Beispielprogramm zur Ausführung der parallelen adjungierten NFFT.

Quelltext 3.2: Anwendung der parallelen adjungierten NFFT.

```
1  #include <nfft-mpi.h>
2
3  int main(int argc, char **argv) {
4    const int d = 3;
5    ptrdiff_t M, N[d], n[d], local_x_num, local_x_start;
6    int np_fftw, m = 4;
7    nfft_mpi_plan myplan;
8
9    MPI_Init(&argc, &argv);
10   fftw_mpi_init();
11
12   MPI_Comm_size(MPI_COMM_WORLD, &np_fftw);
13
14   N[0] = N[1] = N[2] = 8;
15   n[0] = 2*N[0]; n[1] = 2*N[1]; n[2] = 2*N[2];
16   M = N[0]*N[1]*N[2];
17
18   /** init nfft_par */
19   nfft_mpi_init_guru(&myplan, d, N, n, m, &np_fftw, MALLOC_F_HAT|
        FFTW_INIT, FFTW_MEASURE);
20
21   /** count local nodes with some function
        user_counts_local_nodes(lower_border, upper_border) */
22   local_x_num = user_counts_local_nodes(myplan.lower_border,
        myplan.upper_border);
23
24   /** init local nodes */
25   nfft_mpi_init_nodes(&myplan, &local_x_num, &local_x_start, MALLOC_F|
        MALLOC_X, 0);
26
27   /** init local nodes to some function users_x(i) */
28   for(i = 0; i < local_x_num; i++)
29     for(j = 0; j < d; j++)
30       myplan.x[d*i+j] = users_x(d*i+j);
31
32   /** init local coefficients to some function users_f(i) */
33   for(i = 0; i < local_x_num; i++)
34     myplan.f[d*i+j] = users_f(i);
35
36   /** execute par NFFT */
37   nfft_adjoint_par3d(&myplan);
38
39   /** destroy plan and free allocated memory */
40   nfft_mpi_finalize(&myplan, FREE_F_HAT| FREE_F| FREE_X);
41
42   fftw_mpi_cleanup();
43   MPI_Finalize();
44 }
```

4

Numerische Ergebnisse

Die Implementierungen der Algorithmen 3.1 und 3.2 für die dreidimensionale parallele NFFT und ihre Adjungierte wurden auf drei Höchstleistungsrechnern getestet. Dabei ist zu beachten, dass eine gleichmäßige Verteilung der Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}$, auf dem Torus $\mathbb{T}^d$ vorausgesetzt wird. Bei einer stark ungleichmäßigen Verteilung könnte es sogar passieren, dass ein Prozessor $\boldsymbol{p} \in \mathcal{P}$ allein alle Stützstellen zugewiesen bekommt, d.h. $\mathcal{M}^{\boldsymbol{p}} = \mathcal{M}$. Er müsste dann in Schritt 5 von Algorithmus 3.1 allein alle Werte s_j, $j \in \mathcal{M}$, berechnen. In diesem Fall ist keine gute Skalierung zu erwarten. Analog verhält es sich mit Algorithmus 3.2 in Schritt 2.

Die Größe der NFFT wurde bezüglich jeder Dimension gleichgesetzt, d.h. es gilt $N := N_0 = N_1 = N_2 \in 2\mathbb{N}$. Für alle Tests wurde der Abschneideparameter $m = 4$ sowie die Größe der FFT mit $n_0 = n_1 = n_2 = 2N$ gewählt. Nach Formel (2.6) erhalten wir daher eine obere Abschätzung des maximalen relativen Fehlers durch $0,011$. Das Prozessorennetz wurde wegen der eindimensionalen Datenaufteilung der FFTW als $\mathcal{P} = \mathbb{Z}^3/\boldsymbol{P}\mathbb{Z}$ mit $\boldsymbol{P} = (P, 1, 1)$ gewählt. Wir haben in Tabelle 4.1 gemäß Formel (3.14) den nötigen Speicherplatz für $P = 1, 2, 4$ Prozessoren und die ausgewählten Transformationsgrößen $N = 64, 128, 196$ und 256 eingetragen. Diese Daten werden wir im Folgenden nutzen um die Transformationsgröße N der NFFT so zu wählen, dass die NFFT noch mit einem Prozessor berechenbar ist.

Wir vergleichen die Laufzeiten der parallelen NFFT mit den Laufzeiten der FFTW, welche innerhalb der Algorithmen 3.1 und 3.2 für die NFFT aufgerufen wird. Es stellt sich die Frage,

P \ N	64	128	196	256
1	0,05	0,43	1,54	3,41
2	0,03	0,22	0,78	1,72
4	0,02	0,11	0,4	0,88

Tabelle 4.1: Benötigter Hauptspeicher in GByte.

welchen Einfluss die Skalierung der parallelen FFTW auf die Skalierung der parallelen NFFT hat. Dazu messen wir nur die Laufzeiten der Transformationen, d.h. wir berücksichtigen nicht die Initialisierungen. Besonders das Anlegen der Pläne vom Typ `nfft_mpi_plan` und `fftw_mpi_plan` kann für große Transformationen viel Zeit in Anspruch nehmen. Da ein solcher Plan aber meist mehrmals genutzt wird oder auch gespeichert werden kann, soll diese Zeit hier nicht berücksichtigt werden. Die Planung der FFTW wurde mit dem Flag `FFTW_MEASURE` durchgeführt, sodass mehrere Testtransformationen durchgeführt werden um den FFT-Algorithmus mit der optimalen Laufzeit zu bestimmen.

Des Weiteren weisen wir darauf hin, dass während der Initialisierung der NFFT eigentlich eine Sortierung der Stützstellen $\boldsymbol{x}_j$, $j \in \mathcal{M}$, erfolgen muss, sodass jeder Prozessor $\boldsymbol{p} \in \mathcal{P}$ aus unserem Prozessorennetz nur Stützstellen $\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, erhält. Dies könnte mit einem parallelen Sortieralgorithmus implementiert werden, welcher aber momentan noch nicht zur Verfügung steht. Für die Testläufe wurden für jeden Prozessor $\boldsymbol{p} \in \mathcal{P}$ zufällig verteilte Stützstellen im Quader $Q^{\boldsymbol{p}}$ erzeugt um die Sortierung zu umgehen.

Für den Vergleich der Laufzeiten führen wir einige Definitionen ein. Sei $T(P) \in \mathbb{R}_+$ die Laufzeit eines parallelen Algorithmus mit P Prozessoren, so definieren wir die relative Beschleunigung $\psi(P) \in \mathbb{R}_+$ durch $\psi(P) = \frac{T(1)}{T(P)}$. Wir verwenden also nicht die absolute Beschleunigung, bei deren Definition die Laufzeit des seriellen Algorithmus im Zähler des Bruches verwendet wird. Mit der relativen Beschleunigung können wir den Geschwindigkeitszuwachs des Algorithmus in Abhängigkeit der Prozessorenanzahl einschätzen. Im Idealfall sollte $\psi(P) = P$ gelten. Das wohlbekannte Gesetz von Amdahl [30, S. 162] zeigt uns jedoch bereits die Grenzen dieser idealen Skalierung auf. Es besagt, dass die Beschleunigung eines Algorithmus mit der Laufzeit $T(1) = \sigma + \varphi$, wobei σ der nicht parallelisierbare und φ der zu parallelisierende Anteil ist, abgeschätzt werden kann durch

$$\psi(P) = \frac{T(1)}{T(P)} \leq \frac{\sigma + \varphi}{\sigma + \frac{\varphi}{P}}. \tag{4.1}$$

Lassen wir die Anzahl der Prozessoren P in Amdahls Gesetz (4.1) gegen unendlich gehen, so erkennen wir, dass selbst der kleinste nicht parallelisierbare Programmanteil, unabhängig von der verwendeten Prozessorenanzahl P, eine maximale Beschleunigung von $\frac{T(1)}{\sigma}$ erzwingt. Dabei wird in dieser Abschätzung noch nicht einmal der Mehraufwand des parallelen Algorithmus berücksichtigt, welcher die Beschleunigung weiter verringert. Wir wollen dieses Gesetz anpassen indem wir die Gesamtlaufzeit der NFFT $T(P) = T_{\mathsf{FFTW}}(P) + (T(P) - T_{\mathsf{FFTW}}(P))$ zerlegen in die Laufzeit $T_{\mathsf{FFTW}}(P)$, welche durch die FFTW verbraucht wird, und jene Laufzeit $T(P) - T_{\mathsf{FFTW}}(P)$, welche durch den Rest des Programms verbraucht wird und somit von unserer Parallelisierung abhängt. Im besten Fall können wir den Anteil $T(1) - T_{\mathsf{FFTW}}(1)$ mit P Prozessoren um den Faktor P beschleunigen, d.h. $T(P) - T_{\mathsf{FFTW}}(P) \leq \frac{T(1) - T_{\mathsf{FFTW}}(1)}{P}$. Dann ergibt sich als Abschätzung der Beschleunigung

$$\psi(P) = \frac{T(1)}{T(P)} = \frac{T(1)}{T_{\mathsf{FFTW}}(P) + T(P) - T_{\mathsf{FFTW}}(P)} \leq \frac{T(1)}{T_{\mathsf{FFTW}}(P) + \frac{T(1) - T_{\mathsf{FFTW}}(1)}{P}}. \tag{4.2}$$

Die Gesamtlaufzeit $T(P)$ der parallelen NFFT und die Laufzeit $T_{\mathsf{FFTW}}(P)$ der FFTW messen wir in den Laufzeittests für verschiedene Prozessorenanzahlen P und insbesondere auch für einen Prozessor. Somit können wir (4.2) tatsächlich zur Abschätzung der maximal erreichbaren Beschleunigung $\psi(P) = \frac{T(1)}{T(P)}$ nutzen.

Weiterhin definieren wir die Effizienz $\varepsilon(P) \in \mathbb{R}_+$ durch $\varepsilon(P) = \frac{\psi(P)}{P}$. Sie gibt die Auslastung der Prozessoren an und sollte zwischen 0 und 1 liegen. Wir können die Effizienz ebenfalls mit

Hilfe der Abschätzung für die Beschleunigung durch

$$\varepsilon(P) = \frac{\psi(P)}{P} \leq \frac{T(1)}{PT_{\mathsf{FFTW}}(P) + T(1) - T_{\mathsf{FFTW}}(1)} \tag{4.3}$$

nach oben beschränken.

Zur experimentellen Messung des seriellen Anteils $S := \frac{\sigma}{T(1)}$ eines parallelen Algorithmus wurde in [30, S. 167 ff] der sogenannte Karp-Flatt-Abstand

$$S = \frac{\sigma}{T(1)} = \frac{\frac{1}{\psi} - \frac{1}{P}}{1 - \frac{1}{P}} \tag{4.4}$$

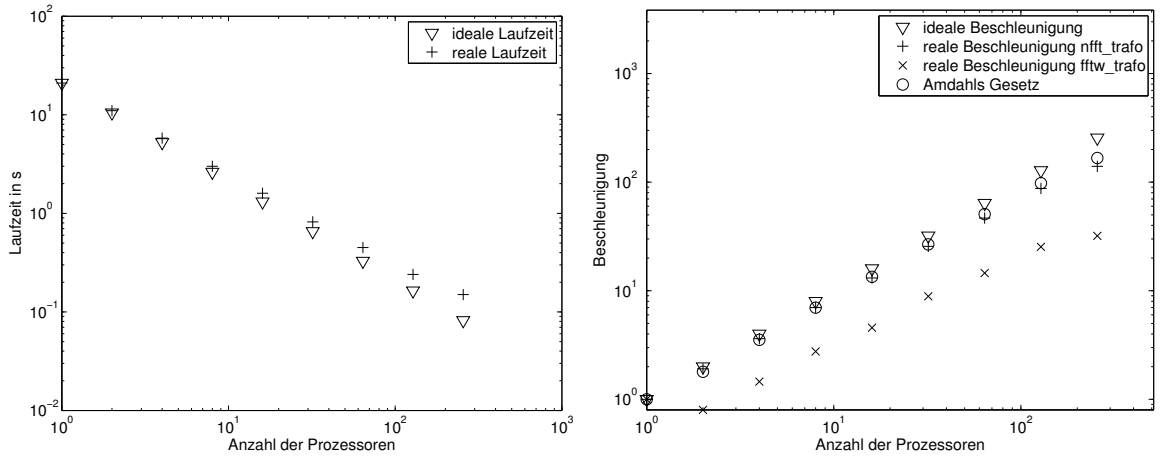
vorgeschlagen. Wir werden mit Hilfe des seriellen Anteils S Rückschlüsse auf die Güte unseres Algorithmus ziehen.

4.1 Der Chemnitzer Hochleistungs-Linux-Cluster (CHiC)

Der Chemnitzer Hochleistungs-Linux-Cluster (CHiC) stellt 530 Knoten mit jeweils 4 GByte Arbeitsspeicher zur Verfügung. Da jeder Knoten über zwei Dual-Core AMD Opteron$^{\text{TM}}$ Prozessoren 2218 Stepping 2 mit 2,6 GHz verfügt, können wir im Allgemeinen für jeden Prozess 1 GByte Arbeitsspeicher verwenden.

Für die Tests wurde der Compiler `MVAPICH2-1.0.1` mit der Optimierungsoption `-O3` benutzt.

Die Größe der NFFT wurde mit $N = 128$ so gewählt, dass die parallele NFFT auch auf einem Prozessor berechnet werden kann. Nach Tabelle 4.1 wäre dies mit einer NFFT der Größe $N = 196$ wegen des Speicherbedarfs von 1,54 GByte Arbeitsspeicher nicht mehr möglich.



(a) Laufzeit $T(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren. (b) Beschleunigung $\psi(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren.

Abbildung 4.1: Laufzeitmessungen der parallelen NFFT für $N = 128$ auf CHiC.

In Abbildung 4.1(a) sind die Laufzeiten der parallelen NFFT gemäß Algorithmus 3.1 mit $N = 128$ für die Prozessorenanzahlen $P = 2^j$, $j = 0, 1, \ldots, 8$, abgetragen. Des Weiteren wurde zum Vergleich die ideale Laufzeit ergänzt, welche sich als Quotient $\frac{T(1)}{P}$ der Laufzeit mit einem Prozessor $T(1)$ und der verwendeten Prozessorenanzahl P ergibt.

Die Beschleunigung können wir in Abbildung 4.1(b) sehen. Dabei entspricht die ideale Beschleunigung der Anzahl der verwendeten Prozessoren und gibt an, um welchen Faktor die Berechnung bestenfalls schneller werden kann. Da die FFTW bereits parallelisiert verwendet wird, haben wir keinen Einfluss auf ihre Laufzeit. Wir können anhand von Amdahls Gesetz (4.2) sehen, welche Beschleunigung maximal durch die Parallelisierung des restlichen Algorithmus möglich ist. Auch diese Werte sind als Amdahls Gesetz in Abbildung 4.1(b) zu sehen. An ihnen können wir sehr gut die Abhängigkeit der Laufzeit unseres Algorithmus von der Laufzeit der FFTW, deren Größe mit $n := n_0 = n_1 = n_2 = 256$ gewählt ist, sehen.

Die Laufzeit der FFTW nimmt beim Übergang von einem auf zwei Prozessoren leicht zu. Danach kann sie bei weiterer Verdopplung der Prozessorenanzahl die Laufzeiten beinahe halbieren. Daraus resultiert auch die Lücke zur idealen Beschleunigung ab $P = 2$, welche bis zur maximalen Prozessorenanzahl kaum zunimmt. Es ist auch zu beachten, dass die Laufzeit der parallelen FFTW nur etwa ein Dreizehntel der Laufzeit der parallelen NFFT auf CHiC beträgt.

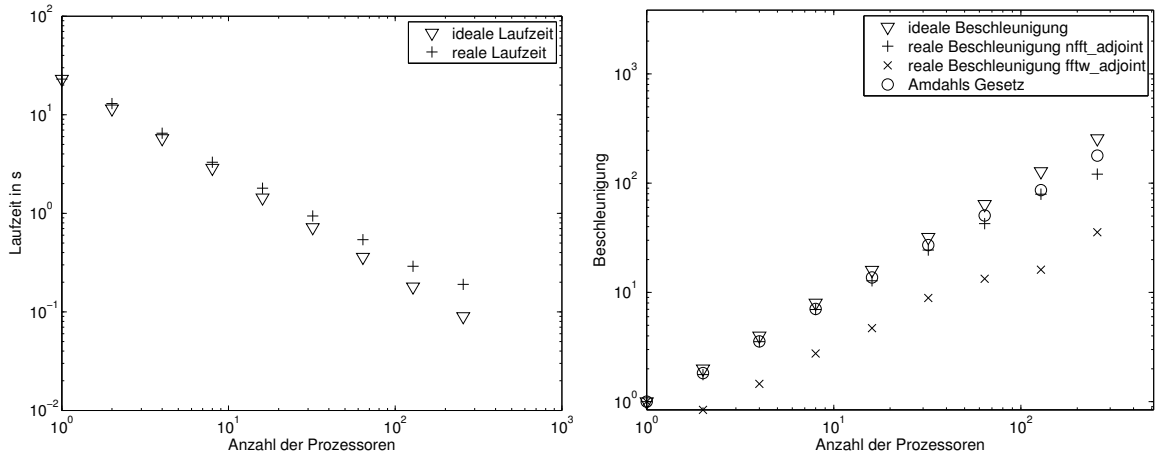
Wir können feststellen, dass die Beschleunigung der NFFT nur wenig kleiner ist als die mögliche Beschleunigung nach Amdahls Gesetz. Dies überrascht uns, wenn wir bedenken, dass wir in der Abschätzung nach Amdahl noch nicht den Mehraufwand des parallelen Algorithmus und keine Arbeitsverteilung unter den Prozessoren berücksichtigt haben. Wir bestimmen deshalb über den Karp-Flatt-Abstand den seriellen Anteil S unseres Algorithmus in Tabelle 4.2.

P	2	4	8	16	32	64	128	256
S_{NFFT}	0.0476	0.0349	0.0204	0.0146	0.0080	0.0059	0.0036	0.0032

Tabelle 4.2: Karp-Flatt-Abstand der parallelen NFFT auf CHiC.

Der serielle Anteil liegt durchgängig unter 5%, was auf eine gute Parallelisierung hinweist. Es fällt aber weiterhin auf, dass der serielle Anteil mit wachsender Prozessorenanzahl bis unter 0,4% sinkt. Ein Grund dafür ist in der Sortierung der Stützstellen zu suchen. Wie in Abschnitt 3.2.4 beschrieben, findet während der Transformation eine Sortierung der lokalen Stützstellen $\boldsymbol{x}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}^{\boldsymbol{p}}$, statt um die Cache-Zugriffe zu verbessern. Die dafür benötigte Zeit geht in die Laufzeit der parallelen NFFT ein. Durch die Erhöhung der Prozessorenanzahl, verringert sich die Anzahl $|\mathcal{M}^{\boldsymbol{p}}|$ der lokalen Stützstellen. Dadurch reduziert sich die benötigte Zeit zum Sortieren und der serielle Anteil der NFFT nimmt ab. Der serielle Anteil sinkt also, da die Zuordnung der lokalen Stützstellen während der Initialisierung einer Vorsortierung in die Quader $Q^{\boldsymbol{p}}$ entspricht und bei der Laufzeitmessung der parallelen NFFT nicht berücksichtigt wird.

Die Laufzeiten der parallelen adjungierten NFFT gemäß Algorithmus 3.2 und der inversen FFTW sehen wir in Abbildung 4.2. Ihr Verhalten ist völlig analog zum Verhalten der Laufzeiten der NFFT und der FFTW.



(a) Laufzeit $T(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren. (b) Beschleunigung $\psi(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren.

Abbildung 4.2: Laufzeitmessungen der parallelen adjungierten NFFT für $N = 128$ auf CHiC.

P	2	4	8	16	32	64	128	256
$S_{\mathsf{NFFT}^{\mathsf{H}}}$	0.1304	0.0435	0.0211	0.0168	0.0099	0.0080	0.0048	0.0044

Tabelle 4.3: Karp-Flatt-Abstand der parallelen adjungierten NFFT auf CHiC.

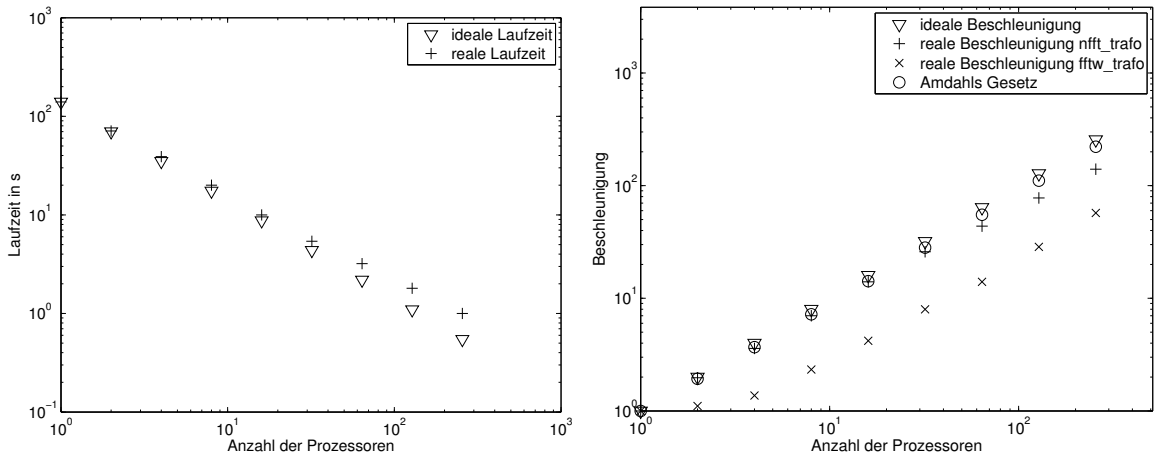
Auch der Karp-Flatt-Abstand der adjungierten NFFT in Tabelle 4.3 verhält sich analog, bis auf einen besonders hohen Wert für den seriellen Anteil $S_{\mathsf{NFFT}^{\mathsf{H}}} = 13\%$ bei 2 Prozessoren.

4.2 Der Jülicher Multiprozessor (JUMP)

Der Jülicher Multiprozessor (JUMP) besitzt 14 Knoten. Jeweils 32 Prozessoren des Typs 64-Bit Power6 4.7 GHz teilen sich auf einem Knoten 112 GByte Arbeitsspeicher. Es ergibt sich also eine Gesamtzahl von 448 Prozessoren von denen jeder 3,5 GByte Arbeitsspeicher bei voller Auslastung der Knoten verwenden kann. Maximal 8 Knoten werden für einen einzelnen Programmlauf zur Verfügung gestellt, d.h. wir können die parallele NFFT mit höchstens 256 Prozessoren testen.

Für die Tests wurde auf den Compiler `IBM XL C/C++ Enterprise Edition for AIX, V9.0`, zurückgegriffen. Neben der Optimierungsoption `-O3` und `-q64` für Kompilation in 64 Bit wurden die architekturspezifischen Optimierungsoptionen `-qhot` und `-qstrict` gesetzt.

Mit Tabelle 4.1 erkennen wir, dass eine NFFT der Größe $N = 256$ noch mit einem Prozessor zu berechnen ist. Die entsprechenden Laufzeiten und die Beschleunigung für die parallele NFFT und FFTW sowie für die parallele adjungierte NFFT und inverse FFTW können wir in den Abbildungen 4.3 und 4.4 sehen. Die Größe der FFTW ist mit $n = n_0 = n_1 = n_2 = 512$ wieder wie in der NFFT gewählt.



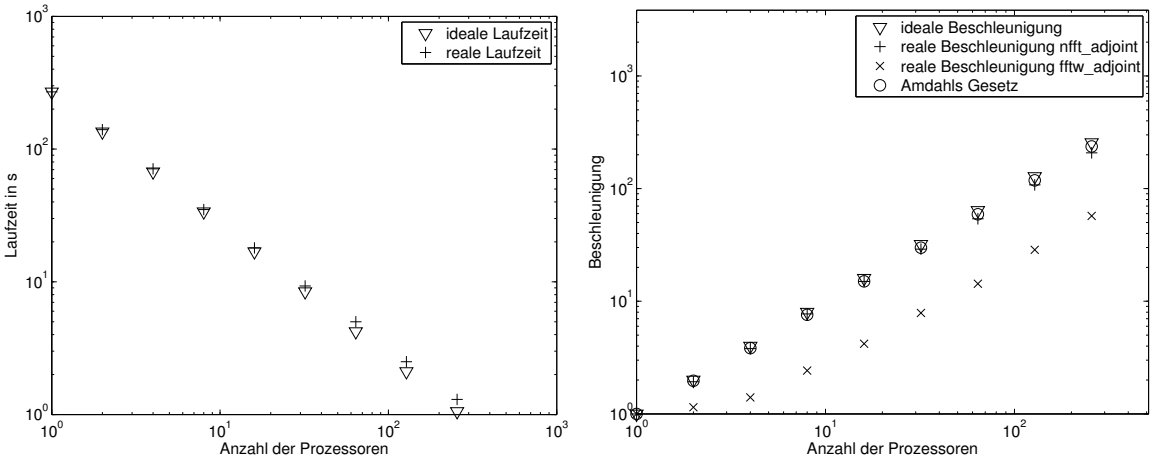
(a) Laufzeit $T(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren. (b) Beschleunigung $\psi(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren.

Abbildung 4.3: Laufzeitmessungen der parallelen NFFT für $N = 256$ auf JUMP.

Es stellte sich heraus, dass die Laufzeit der FFTW ab 128 Prozessoren im ersten Aufruf der FFTW sehr stark zunimmt. Ab der zweiten Transformation hingegen liegen die Laufzeiten wieder bei den erwarteten Werten. So benötigte eine FFT der Größe $n = 512$ bezüglich jeder Dimension für den ersten Lauf etwa 4 Sekunden, während jede weitere Transformation auch mit anderen Daten und sogar mit einem neu erstellten FFTW-Plan nur 0,11 Sekunden benötigte. Da dieses Verhalten nur auf dem Parallelrechner JUMP und auch nicht für die inverse Fourier-Transformation festzustellen war, liegt die Vermutung nah, dass der Mangel im Zusammenspiel der FFTW mit der Rechnerarchitektur zu suchen ist. Es wurde für die im Folgenden beschriebenen Testläufe auf JUMP eine FFT vor den eigentlichen Laufzeitmessungen ausgeführt. Die somit gemessenen Laufzeiten entsprechen den zu erwartenden Werten. Sie liegen sowohl für die NFFT gemäß Algorithmus 3.1 als auch für die adjungierte NFFT gemäß Algorithmus 3.2 nur knapp oberhalb der idealen Laufzeit bis zu 256 Prozessoren, was der maximal verfügbaren Anzahl an Prozessoren für einen Programmlauf auf JUMP entspricht. Damit ist auch die Beschleunigung nur knapp unter den nach Abschätzung (4.2) bestenfalls erreichbaren Werten. Die FFTW skaliert wiederum besonders durch den Übergang von einem auf zwei Prozessoren nicht ideal. Das Verhältnis der Laufzeiten der parallelen FFTW der parallelen NFFT beträg auf JUMP allerdings 1 zu 22.

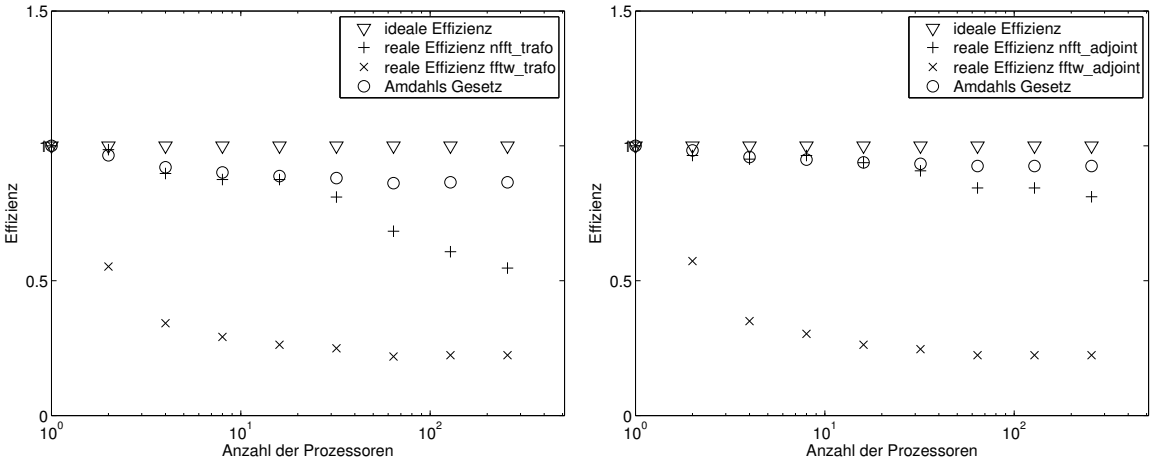
Ein Vergleich der Effizienzen in Abbildung 4.5 zeigt, dass die adjungierte NFFT eine bessere Prozessorennutzung für große Prozessorenzahlen ermöglicht. Bei der Bewertung dieses Verhaltens müssen wir beachten, dass der adjungierte Algorithmus auf JUMP etwa die doppelte Laufzeit im Vergleich zur parallelen NFFT benötigt. Somit ergibt sich für die parallele adjungierte NFFT ein größeres Parallelisierungspotential. Laufzeitmessungen der einzelnen Schritte der Algorithmen 3.1 und 3.2 haben belegt, dass die Unterschiede bei der Berechnung von Schritt 5 in Algorithmus 3.1 und Schritt 2 in Algorithmus 3.2 entstehen.

Der Karp-Flatt-Abstand in Tabelle 4.4 verhält sich ebenfalls ähnlich wie bereits auf dem Parallelrechner CHiC. Er liegt unter 5% und wir können einen Abfall des seriellen Anteils beider Algorithmen für wachsende Prozessorenanzahl P beobachten, was wiederum auf den Laufzeitgewinn durch die Vorsortierung der Stützstellen innerhalb der Initialisierung zurück zu führen ist.



(a) Laufzeit $T(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren. (b) Beschleunigung $\psi(P)$ für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren.

Abbildung 4.4: Laufzeitmessungen der parallelen adjungierten NFFT für $N = 256$ auf JUMP.



(a) Effizienz $\varepsilon(P)$ der parallelen NFFT für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren. (b) Effizienz $\varepsilon(P)$ der parallelen adjungierten NFFT für $P = 2^0, 2^1, \ldots, 2^8$ Prozessoren.

Abbildung 4.5: Effizienz der parallelen NFFT und der parallelen adjungierten NFFT für $N = 256$ auf JUMP.

P	2	4	8	16	32	64	128	256
S_{NFFT}	0.0143	0.0381	0.0204	0.0095	0.0076	0.0073	0.0051	0.0032
$S_{\mathsf{NFFT}^{\mathsf{H}}}$	0.0370	0.0173	0.0053	0.0044	0.0033	0.0029	0.0015	0.0009

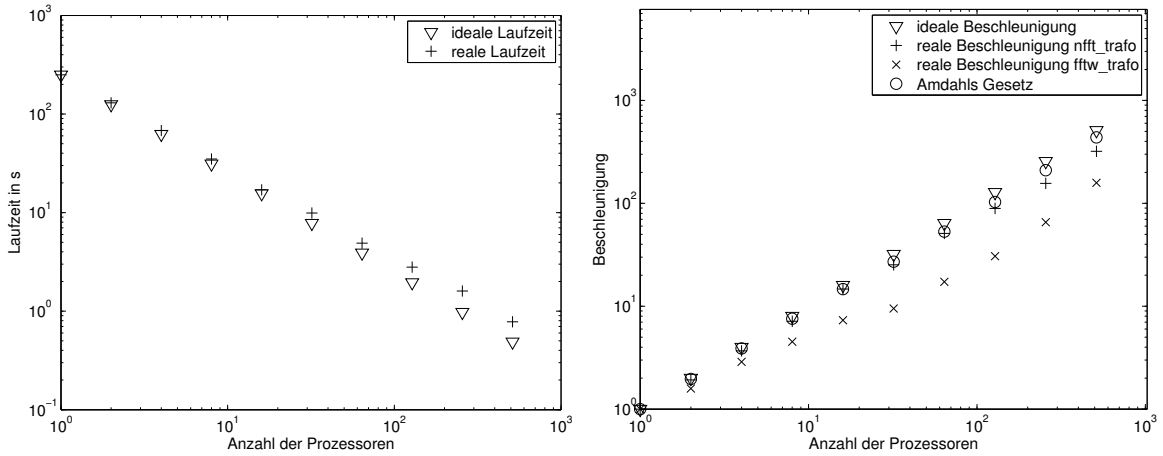
Tabelle 4.4: Karp-Flatt-Abstand der parallelen NFFT und ihrer Adjungierten auf JUMP.

4.3 Der Jülicher BlueGene P (JUGENE)

Der Parallelrechner JUGENE besitzt 16384 Knoten deren Hauptspeicher jeweils auf 2 GByte beschränkt ist. Jeder Knoten besitzt wiederum vier 32-Bit PowerPC 450 Prozessoren mit 850 MHz. Es stehen also maximal 65536 Prozessoren mit jeweils 512 MByte Arbeitsspeicher auf JUGENE zur Verfügung.

Für die Tests wurde der Compiler `IBM XL C Advanced Edition for Blue Gene/P, V9.0`, verwendet. Neben der Optimierungsoption `-O3` wurden noch `-qtune=450` für architekturspezifische Optimierungen und `-qarch=450` gesetzt, d.h. die für jeden Prozessor vorhandene zweite Gleitkommaberechnungseinheit wurde nicht in die Tests einbezogen.

Nach Tabelle 4.1 können wir $N = 196$ als Transformationsgröße wählen, da wir bei der Berechnung mit einem einzelnen Prozessor auf die gesamten 2 GByte Arbeitsspeicher aller 4 Prozessoren eines Knotens zugreifen können. Auch mit 2 Prozessoren reicht jeweils 1 GByte Arbeitsspeicher um die parallele NFFT auszuführen. Sobald wir 4 Prozessoren oder mehr verwenden, benötigen wir für die NFFT weniger als die verfügbaren 512 MByte Arbeitsspeicher. Obwohl $N = 196$ ab $P = 8$ nicht mehr durch die Anzahl der Prozessoren teilbar ist und somit die Rechenarbeit nicht mehr gleichmäßig verteilt wird, sind sowohl die Laufzeiten der parallelen NFFT als auch ihrer Adjungierten nahezu ideal. Dies ist in den Abbildungen 4.6 und 4.7 zu sehen.

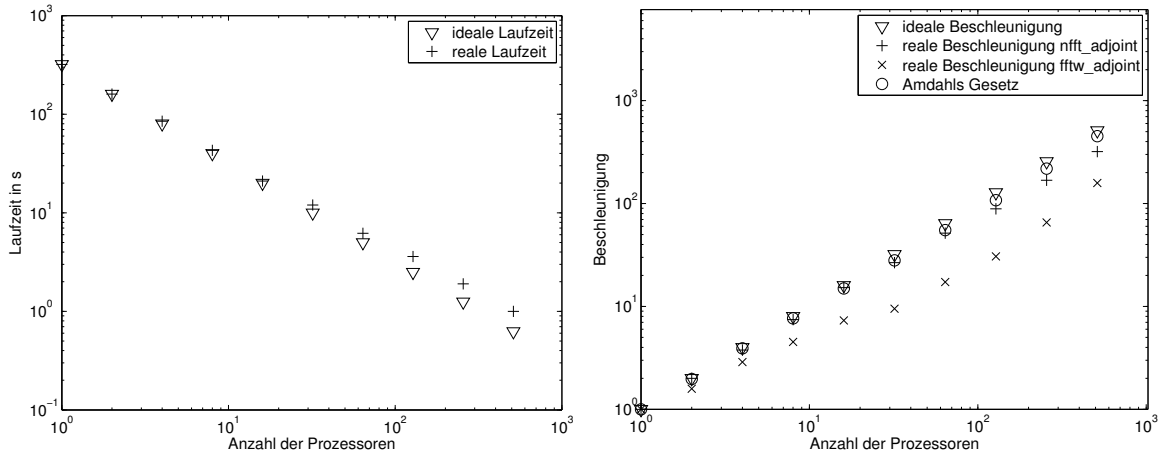


(a) Laufzeit $T(P)$ für$P = 2^0, 2^1, \ldots, 2^9$ Prozessoren. (b) Beschleunigung $\psi(P)$ für $P = 2^0, 2^1, \ldots, 2^9$ Prozessoren.

Abbildung 4.6: Laufzeitmessungen der parallelen NFFT für $N = 196$ auf JUGENE.

Da die Größe der FFTW bezüglich der ersten Dimension $n_0 = 388$ auch die maximale Anzahl der Prozessoren für die FFTW beschränkt, ist eine gute Skalierung nur bis $P = 256$ zu erwarten. Dennoch kann sowohl die NFFT als auch die FFTW mit $P = 512$ Prozessoren noch einen deutlichen Geschwindigkeitsgewinn erzielen. Eine weitere Erhöhung der Prozessorenanzahl bringt aber keine Laufzeitverringerung mehr mit sich, was wiederum die Verwendung einer anderen FFT nahelegt, welche die Nutzung weiterer Prozessoren ermöglicht.

Der Karp-Flatt-Abstand in Tabelle 4.5 liegt durchgängig unter 4% und fällt wie bereits auf den anderen Parallelrechnern bei steigender Prozessorenanzahl.

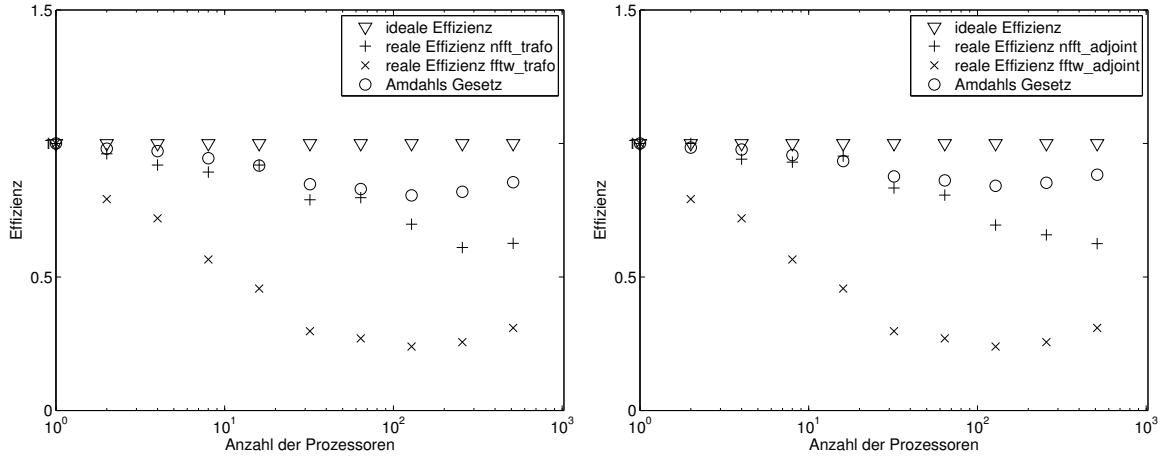


(a) Laufzeit $T(P)$ für $P = 2^0, 2^1, \ldots, 2^9$ Prozessoren. (b) Beschleunigung $\psi(P)$ für $P = 2^0, 2^1, \ldots, 2^9$ Prozessoren.

Abbildung 4.7: Laufzeitmessungen der parallelen adjungierten NFFT für $N = 196$ auf JUGENE.

P	2	4	8	16	32	64	128	256	512
S_{NFFT}	0.0400	0.0293	0.0171	0.0059	0.0086	0.0040	0.0034	0.0025	0.0012
$S_{\mathsf{NFFT}^{\mathsf{H}}}$	0	0.0208	0.0107	0.0033	0.0065	0.0038	0.0035	0.0020	0.0012

Tabelle 4.5: Karp-Flatt-Abstand der parallelen NFFT und ihrer Adjungierten auf JUGENE.



(a) Effizienz $\varepsilon(P)$ der parallelen NFFT für $P = 2^0, 2^1, \ldots, 2^9$ Prozessoren. (b) Effizienz $\varepsilon(P)$ der parallelen adjungierten NFFT für $P = 2^0, 2^1, \ldots, 2^9$ Prozessoren.

Abbildung 4.8: Effizienz der parallelen NFFT und der parallelen adjungierten NFFT für $N = 196$ auf JUGENE.

Abbildung 4.8 zeigt noch die Effizienz der parallelen NFFT und ihrer Adjungierten im Vergleich zur parallelen FFTW. Es lässt sich gut erkennen, dass die Effizienz der NFFT deutlich höher als die der FFTW ist. Bis zu 64 Prozessoren liegt die erreichte Effizienz der NFFT und der adjungierten NFFT sehr nah an der maximal erreichbaren Effizienz welche mit Formel (4.3) nach Amdahl abgeschätzt wurde. Bei der Beurteilung der Laufzeiten der FFTW sollten wir wiederum bedenken, dass das Verhältnis der Laufzeiten der FFTW und der NFFT etwa 1 zu 13 beträgt. Dennoch ist das Laufzeitverhalten der FFTW ohne weitere Analysen nicht nachzuvollziehen. Uns fällt besonders die späte Steigerung der Effizienz auf.

5

Die schnelle Summation

Für gegebene Koeffizienten $\alpha_k \in \mathbb{C}$, $k = 1, \ldots, M_1$, Quellknoten $\boldsymbol{x}_k \in \mathbb{R}^d$, $k = 1, \ldots, M_1$, und Zielknoten $\boldsymbol{y}_j \in \mathbb{R}^d$, $j = 1, \ldots, M_2$, sind wir an der schnellen Berechnung der Summen

$$f(\boldsymbol{x}) := \sum_{k=1}^{M_1} \alpha_k \mathcal{K}(\boldsymbol{x}_k - \boldsymbol{x}), \tag{5.1}$$

an den Stellen $\boldsymbol{y}_j \in \mathbb{R}^d$, $j = 1, \ldots, M_2$, mit einer radialen Kernfunktion $\mathcal{K}(\boldsymbol{x}) = K(\|\boldsymbol{x}\|_2)$ interessiert. Dabei soll $\|\boldsymbol{x}\|_2 := (x_0^2 + \cdots + x_{d-1}^2)^{1/2}$ für $\boldsymbol{x} = (x_t)_{t=0}^{d-1}$ den Euklidischen Abstand bezeichnen und die Funktion $K : \mathbb{R}_+ \to \mathbb{R}$ bis auf eine Singularität im Ursprung hinreichend glatt sein.

Beispiele für radiale Kernfunktionen werden z.B. in [10] gegeben. Zu ihnen gehören die Funktionen

$$\mathcal{K}_0(\boldsymbol{x}) = \log \|\boldsymbol{x}\|_2, \qquad \mathcal{K}_\beta(\boldsymbol{x}) = \frac{1}{\|\boldsymbol{x}\|_2^\beta}, \quad \beta \in \mathbb{N},$$

und die verallgemeinerten Multi-Quadriken

$$\mathcal{K}_{-1}(\boldsymbol{x}, c) = (\|\boldsymbol{x}\|_2^2 + c^2)^{\frac{1}{2}}, \qquad \mathcal{K}_\beta(\|\boldsymbol{x}\|_2, c) = (\|\boldsymbol{x}\|_2^2 + c^2)^{-\frac{\beta}{2}}, \quad \beta \in \mathbb{N} \text{ ungerade}.$$

Die naive Berechnung von (5.1) benötigt $\mathcal{O}(M_1 M_2)$ arithmetische Operationen. Ein bekannter serieller Algorithmus zur schnellen Auswertung der Summen ist die schnelle Multipol-Methode (FMM) [17, 15, 1]. Er benötigt nur $\mathcal{O}\left((M_1 + M_2)(\log \frac{1}{\varepsilon})^d\right)$ arithmetische Operationen, wobei ε die gewünschte Genauigkeit ist. In [28] wurde ein weiterer schneller serieller Algorithmus vorgestellt, welcher auf der Verwendung der NFFT basiert und $\mathcal{O}(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}| + (M_1 + M_2)(\log \frac{1}{\varepsilon})^d)$ arithmetische Operationen benötigt.

5.1 Realisierung mittels der NFFT

Die in [28] vorgeschlagene Idee zur schnellen Berechnung der Summen (5.1) beruht ähnlich wie die FMM auf einer Aufteilung des Definitionsbereiches der Kernfunktion $\mathcal{K}$ in eine kleine Umgebung B_{ε_I}, $\varepsilon_I > 0$, des Ursprungs, dem sogenannten Nahfeld, und dem restlichen Definitionsbereich, welcher als Fernfeld bezeichnet wird. Zuerst wird der Kernfunktion $\mathcal{K}$ eine 1-periodische Approximante $\tilde{\mathcal{K}} \in C^r[\mathbb{T}^d]$ zugeordnet, welche wiederum mit einem trigonometrischen Polynom $T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ vom Grad $\boldsymbol{N}$ angenähert werden kann. Eine gute Approximation der Kernfunktion $\mathcal{K}$ im Nahfeld ist auf Grund der Singularität nicht mit Hilfe der stetigen Funktion $T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ zu erwarten. Daher erfolgt die Auswertung der Kernfunktion in der Nähe des Ursprungs exakt und im Fernfeld über das trigonometrische Polynom $T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ mit Hilfe der NFFT.

Zur Anzahl der Quell- und Zielknoten $M_1, M_2 \in \mathbb{N}$ definieren wir die Indexmengen $\mathcal{M}_1 := \{1, \ldots, M_1\}$, $\mathcal{M}_2 := \{1, \ldots, M_2\}$. Wir nehmen zur Vereinfachung an, dass die Quellknoten $\boldsymbol{x}_k \in \mathbb{R}^d$, $k \in \mathcal{M}_1$, und die Zielknoten $\boldsymbol{y}_j \in \mathbb{R}^d$, $j \in \mathcal{M}_2$, die Bedingungen $\|\boldsymbol{x}_k\|_2$, $\|\boldsymbol{y}_j\|_2 < \frac{1}{4} - \frac{\varepsilon_B}{2}$ und somit $\|\boldsymbol{x}_k - \boldsymbol{y}_j\|_2 < \frac{1}{2} - \varepsilon_B$ für $0 < \varepsilon_B < \frac{1}{2}$ und alle $k \in \mathcal{M}_1$, $j \in \mathcal{M}_2$, erfüllen. Der Parameter ε_B sichert, dass $\mathcal{K}$ später nur in der Kugel $B_{\frac{1}{2}-\varepsilon_B}$ berechnet werden muss, was für die Konstruktion der $\mathbf{1}$-periodischen Approximante $\tilde{\mathcal{K}}$ wichtig ist. Bei der Definition von $\tilde{\mathcal{K}} : \mathbb{T}^d \to \mathbb{R}$ durch

$$\tilde{\mathcal{K}}(\boldsymbol{x}) := \begin{cases} K_I(\|\boldsymbol{x}\|_2) & : \|\boldsymbol{x}\|_2 < \varepsilon_I, \\ K_B(\|\boldsymbol{x}\|_2) & : \frac{1}{2} - \varepsilon_B < \|\boldsymbol{x}\|_2 < \frac{1}{2}, \\ K_B(\frac{1}{2}) & : \|\boldsymbol{x}\|_2 \geq \frac{1}{2}, \\ K(\|\boldsymbol{x}\|_2) & : \text{sonst}, \end{cases}$$

mit $0 < \varepsilon_I < \frac{1}{2} - \varepsilon_B < \frac{1}{2}$, werden die Funktionen K_I und K_B so gewählt, dass $\tilde{\mathcal{K}} \in C^r[\mathbb{T}^d]$, $r \in \mathbb{N}_0$, also r-mal stetig differenzierbar ist. Diese Regularisierung wird in [10] näher beschrieben. Nun wird $\tilde{\mathcal{K}}$ approximiert durch das trigonometrische Polynom

$$T_{\boldsymbol{N}}(\tilde{\mathcal{K}})(\boldsymbol{x}) := \sum_{\boldsymbol{l} \in I_{\boldsymbol{N}}} b_{\boldsymbol{l}} \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}},$$

mit Koeffizienten $b_{\boldsymbol{l}} \in \mathbb{C}$, $\boldsymbol{l} \in I_{\boldsymbol{N}}$, die wir als Ergebnis der diskreten Fourier-Transformation

$$b_{\boldsymbol{l}} := \frac{1}{|I_{\boldsymbol{N}}|} \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \tilde{\mathcal{K}}(\boldsymbol{N}^{-1} \odot \boldsymbol{k}) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} (\boldsymbol{N}^{-1} \odot \boldsymbol{l})}, \quad \boldsymbol{l} \in I_{\boldsymbol{N}},$$

wählen. Setzen wir in unser Ausgangsproblem (5.1) für die Kernfunktion $\mathcal{K} = (\mathcal{K} - \tilde{\mathcal{K}}) + (\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})) + T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ ein und nehmen an, dass der Term $\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ auf Grund der guten Eigenschaften von $\tilde{\mathcal{K}}$ klein wird, so erhalten wir die Approximation

$$f(\boldsymbol{x}) \approx \check{f}(\boldsymbol{x}) := \sum_{k \in \mathcal{M}_1} \alpha_k (\mathcal{K} - \tilde{\mathcal{K}})(\boldsymbol{x}_k - \boldsymbol{x}) + \sum_{k \in \mathcal{M}_1} \alpha_k T_{\boldsymbol{N}}(\tilde{\mathcal{K}})(\boldsymbol{x}_k - \boldsymbol{x}). \tag{5.2}$$

Die Werte $\check{f}(\boldsymbol{y}_j)$, $j \in \mathcal{M}_2$, berechnen wir in zwei Schritten durch die einzelne Auswertung der beiden Summen in (5.2). Das trigonometrische Polynom $T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ lässt sich wegen

$$\sum_{k \in \mathcal{M}_1} \alpha_k T_{\boldsymbol{N}}(\tilde{\mathcal{K}})(\boldsymbol{x}_k - \boldsymbol{y}_j) = \sum_{k \in \mathcal{M}_1} \alpha_k \sum_{\boldsymbol{l} \in I_{\boldsymbol{N}}} b_{\boldsymbol{l}} \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l} (\boldsymbol{x}_k - \boldsymbol{y}_j)} = \sum_{\boldsymbol{l} \in I_{\boldsymbol{N}}} \left(b_{\boldsymbol{l}} \sum_{k \in \mathcal{M}_1} \alpha_k \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}_k} \right) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{y}_j},$$

an allen Zielknoten $\boldsymbol{y}_j$, $j \in \mathcal{M}_2$, mit Hilfe einer adjungierten NFFT, einer Multiplikation mit den Koeffizienten $b_{\boldsymbol{l}}$, $\boldsymbol{l} \in I_{\boldsymbol{N}}$, und einer NFFT in $\mathcal{O}(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}| + (M_1 + M_2)(\log \frac{1}{\varepsilon})^d)$ arithmetischen Operationen auswerten. Um bei der Berechnung der Werte $(\mathcal{K} - \tilde{\mathcal{K}})(\boldsymbol{x}_k - \boldsymbol{y}_j)$ die gewünschte Komplexität des Algorithmus zu erhalten, müssen wir eine genügende Gleichverteilung der Quellknoten fordern, d.h. es soll eine Konstante $\nu \in \mathbb{N}$ existieren, sodass jede Kugel $B_{\varepsilon_I}(\boldsymbol{y})$, $\boldsymbol{y} \in \mathbb{T}^d$, höchstens ν Quellknoten $\boldsymbol{x}_k$ enthält. Wegen $\|\boldsymbol{y}_j - \boldsymbol{x}_k\|_2 < \frac{1}{2} - \varepsilon_B$ und $\operatorname{supp}(\mathcal{K} - \tilde{\mathcal{K}}) \cap B_{\frac{1}{2}-\varepsilon_B} = B_{\varepsilon_I}$ benötigen wir für die Berechnung der Summen

$$\sum_{k \in \mathcal{M}_1} \alpha_k (\mathcal{K} - \tilde{\mathcal{K}})(\boldsymbol{y}_j - \boldsymbol{x}_k), \quad j \in \mathcal{M}_2,$$

höchstens νM_2 arithmetische Operationen, was einer Komplexität von $\mathcal{O}(M_2)$ entspricht. Dem Nahfeld $B_{\varepsilon_I}(\boldsymbol{y})$ eines Zielknotens $\boldsymbol{y} \in \mathbb{T}$ ordnen wir die Nahfeldindizes

$$\mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}) := \{k \in \mathcal{M}_1 : \boldsymbol{x}_k \in B_{\varepsilon_I}(\boldsymbol{y})\}$$

aller im Nahfeld von $\boldsymbol{y}$ enthaltenen Quellknoten zu.

5.1.1 Der serielle Algorithmus

Folgender Algorithmus 5.1 fasst die auf der NFFT basierende schnelle Summation für die Berechnung von (5.1) zusammen.

5.1.2 Der parallele Algorithmus

Im Folgenden entwickeln wir einen parallelen Algorithmus für die schnelle Summation basierend auf den parallelen Algorithmen zur NFFT aus Kapitel 3. Es seien ein Prozessorennetz $\mathcal{P}$, sowie die Mengen $\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$, $Q^{\boldsymbol{p}}$, $\boldsymbol{p} \in \mathcal{P}$, wie in Abschnitt 3.1.1 gegeben. Wir zerlegen die Indexmengen $\mathcal{M}_1$, $\mathcal{M}_2$ derart in $\mathcal{M}_1^{\boldsymbol{p}} := \{k \in \mathcal{M}_1 : \boldsymbol{x}_k \in Q^{\boldsymbol{p}}\}$, $\mathcal{M}_2^{\boldsymbol{p}} := \{j \in \mathcal{M}_2 : \boldsymbol{y}_j \in Q^{\boldsymbol{p}}\}$, $\boldsymbol{p} \in \mathcal{P}$, dass die Quell- und Zielknoten $\boldsymbol{x}_k$, $k \in \mathcal{M}_1^{\boldsymbol{p}}$, $\boldsymbol{y}_j$, $j \in \mathcal{M}_2^{\boldsymbol{p}}$, innerhalb des Quaders $Q^{\boldsymbol{p}}$ auf Prozessor $\boldsymbol{p} \in \mathcal{P}$ liegen. Nach der Definition der Indexmenge $\mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)$ liegen genau die Quellknoten $\boldsymbol{x}_k$, $k \in \mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)$, im Nahfeld $B_{\varepsilon_I}(\boldsymbol{y}_j)$ des Zielknotens $\boldsymbol{y}_j$. Besitzt Prozessor $\boldsymbol{p} \in \mathcal{P}$ die Zielknoten $\boldsymbol{y}_j \in Q^{\boldsymbol{p}}$, $j \in \mathcal{M}_2^{\boldsymbol{p}}$, so muss er auch alle Quellknoten $\boldsymbol{x}_k$, $k \in \mathcal{M}_{1,\varepsilon_I}^{\boldsymbol{p}}$ mit

$$\mathcal{M}_{1,\varepsilon_I}^{\boldsymbol{p}} := \bigcup_{j \in \mathcal{M}_2^{\boldsymbol{p}}} \mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)$$

besitzen, um die Nahfeldanteile $\sum_{k \in \mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)} (\mathcal{K} - \tilde{\mathcal{K}})(\boldsymbol{x}_k - \boldsymbol{y}_j)$ auswerten zu können. Die Quellknoten $\boldsymbol{x}_k$, $k \in \mathcal{M}_{1,\varepsilon_I}^{\boldsymbol{p}}$, liegen im Allgemeinen nicht auf einem Prozessor, weshalb ein Senden der nicht vorhandenen Knoten erforderlich ist. Zu einem gegebenen Index $k \in \mathcal{M}_1$ fassen wir daher alle Prozessoren, welche den Quellknoten $\boldsymbol{x}_k$ mindestens einmal zur Berechnung der Funktion $\mathcal{K} - \tilde{\mathcal{K}}$ in einem der Nahfelder $B_{\varepsilon_I}(\boldsymbol{y}_j)$, $j \in \mathcal{M}_2^{\boldsymbol{p}}$, benötigen, in der Menge

$$\mathcal{P}_k := \left\{\boldsymbol{r} \in \mathcal{P} : \exists\, j \in \mathcal{M}_2^{\boldsymbol{p}} \text{ mit } k \in \mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)\right\},$$

zusammen und bezeichnen mit $\boldsymbol{p}_k \in \mathcal{P}$ den eindeutig bestimmten Prozessor mit $k \in \mathcal{M}_1^{\boldsymbol{p}_k}$. Der Prozessor $\boldsymbol{p}_k$ besitzt also bereits nach der parallelen NFFT den Quellknoten $\boldsymbol{x}_k$ und wird diesen an alle Prozessoren $\boldsymbol{p} \in \mathcal{P}_k$ senden. Wir können nun den parallelen Algorithmus 5.2 für die auf der NFFT basierende schnelle Summation formulieren.

Algorithmus 5.1 `schnelle Summation`

Eingabe:

$d \in \mathbb{N}$,	{Dimension}
$M_1 \in \mathbb{N}$,	{Anzahl der Quellknoten}
$M_2 \in \mathbb{N}$,	{Anzahl der Zielknoten}
$\boldsymbol{N} \in 2\mathbb{N}^d$,	{Größe der NFFT}
$\boldsymbol{x}_k \in \mathbb{T}^d$, $\|\boldsymbol{x}_k\|_2 \leq \frac{1}{4} - \frac{\varepsilon_B}{2}$, $k \in \mathcal{M}_1$,	{Quellknoten}
$\boldsymbol{y}_j \in \mathbb{T}^d$, $\|\boldsymbol{y}_j\|_2 \leq \frac{1}{4} - \frac{\varepsilon_B}{2}$, $j \in \mathcal{M}_2$,	{Zielknoten}
$\alpha_k \in \mathbb{C}$, $k \in \mathcal{M}_1$,	{Koeffizienten von f}
$b_{\boldsymbol{l}} \in \mathbb{C}$, $\boldsymbol{l} \in I_{\boldsymbol{N}}$,	{Koeffizienten von $T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$}
$\varepsilon_I \in \mathbb{R}_+$,	{Radius des Nahfeldes}
K,	{Kernfunktion}
K_I,	{Nahfeldregularisierung von $\mathcal{K}$}

1: Für $\boldsymbol{l} \in I_{\boldsymbol{N}}$ berechne mit d-variater adjungierter NFFT

$$a_{\boldsymbol{l}} := \sum_{k \in \mathcal{M}_1} \alpha_k \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}_k}.$$

2: Für $\boldsymbol{l} \in I_{\boldsymbol{N}}$ berechne

$$c_{\boldsymbol{l}} := b_{\boldsymbol{l}} a_{\boldsymbol{l}}.$$

3: Für $j \in \mathcal{M}_2$ berechne mit d-variater NFFT

$$d_j := \sum_{\boldsymbol{l} \in I_{\boldsymbol{N}}} c_{\boldsymbol{l}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{y}_j}.$$

4: Für $j \in \mathcal{M}_2$ berechne

$$\check{f}(\boldsymbol{y}_j) := d_j + \sum_{k \in \mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)} (K - K_I)(\|\boldsymbol{x}_k - \boldsymbol{y}_j\|_2).$$

Ausgabe:

$\check{f}(\boldsymbol{y}_j) \in \mathbb{R}$, $j \in \mathcal{M}_2$	{Approximation für $f(\boldsymbol{y}_j)$}

Algorithmus 5.2 `parallele schnelle Summation in 3D`

Eingabe:

$\lvert\mathcal{M}_1^{\boldsymbol{p}}\rvert \in \mathbb{N}_0$,	{Anzahl der lokalen Quellknoten}
$\lvert\mathcal{M}_2^{\boldsymbol{p}}\rvert \in \mathbb{N}_0$,	{Anzahl der lokalen Zielknoten}
$\boldsymbol{N} \in 2\mathbb{N}^3$,	{Größe der NFFT}
$\boldsymbol{x}_k \in Q^{\boldsymbol{p}}$, $\lVert\boldsymbol{x}_k\rVert_2 \leq \frac{1}{4} - \frac{\varepsilon_B}{2}$, $k \in \mathcal{M}_1^{\boldsymbol{p}}$,	{Lokale Quellknoten}
$\boldsymbol{y}_j \in Q^{\boldsymbol{p}}$, $\lVert\boldsymbol{y}_j\rVert_2 \leq \frac{1}{4} - \frac{\varepsilon_B}{2}$, $j \in \mathcal{M}_2^{\boldsymbol{p}}$,	{Lokale Zielknoten}
$\alpha_k \in \mathbb{C}$, $k \in \mathcal{M}_1^{\boldsymbol{p}}$,	{Lokale Koeffizienten von f}
$b_{\boldsymbol{l}} \in \mathbb{C}$, $\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$,	{Lokale Koeffizienten von $T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$}
$\varepsilon_I \in \mathbb{R}_+$,	{Radius des Nahfeldes}
K,	{Kernfunktion}
K_I,	{Nahfeldregularisierung von $\mathcal{K}$}
$\mathcal{P}_k, \boldsymbol{p}_k,\ k \in \mathcal{M}_1$,	{Daten zur Kommunikation}

1: Für $\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$ berechne mit paralleler adjungierter 3D-NFFT

$$a_{\boldsymbol{l}} := \sum_{\boldsymbol{r}\in\mathcal{P}} \sum_{k\in\mathcal{M}_1^{\boldsymbol{r}}} \alpha_k \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l}\boldsymbol{x}_k}.$$

2: Für $\boldsymbol{l} \in \mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{p}}$ berechne

$$c_{\boldsymbol{l}} := b_{\boldsymbol{l}} a_{\boldsymbol{l}}.$$

3: Für $j \in \mathcal{M}_2^{\boldsymbol{p}}$ berechne mit paralleler 3D-NFFT

$$d_j := \sum_{\boldsymbol{r}\in\mathcal{P}} \sum_{\boldsymbol{l}\in\mathcal{I}_{\boldsymbol{N}}^{\boldsymbol{r}}} c_{\boldsymbol{l}} \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{l}\boldsymbol{y}_j}.$$

4: Für $k \in \mathcal{M}_{1,\varepsilon_I}^{\boldsymbol{p}}$ sende $\boldsymbol{x}_k$ an alle Prozessoren $\boldsymbol{r} \in \mathcal{P}_k$.

5: Für $k \in \mathcal{M}_{1,\varepsilon_I}^{\boldsymbol{p}}$ empfange $\boldsymbol{x}_k$ von Prozessor $\boldsymbol{p}_k$.

6: Für $j \in \mathcal{M}_2^{\boldsymbol{p}}$ berechne

$$\check{f}(\boldsymbol{y}_j) := d_j + \sum_{k\in\mathcal{M}_{1,\varepsilon_I}(\boldsymbol{y}_j)} (K - K_I)(\lVert\boldsymbol{x}_k - \boldsymbol{y}_j\rVert_2).$$

Ausgabe:

$\check{f}(\boldsymbol{y}_j) \in \mathbb{R},\ j \in \mathcal{M}_2^{\boldsymbol{p}}$	{Approximation für $f(\boldsymbol{y}_j)$}

5.1.3 Grenzen der Parallelisierung

Eine wesentliche Voraussetzung für die gute Skalierung der parallelen adjungierten NFFT nach Algorithmus 3.2 ist die gleichmäßige Verteilung der Stützstellen $\boldsymbol{x}_k$, $k \in \mathcal{M}_1$, auf dem Torus $\mathbb{T}$. Die Forderung $\|\boldsymbol{x}_k\| \leq \frac{1}{4} - \varepsilon_B$, $k \in \mathcal{M}_1$, von Algorithmus 5.2 zur parallelen schnellen Summation, führt aber zu einer Ballung der Stützstellen in der Kugel $B_{\frac{1}{4}-\varepsilon_B}$. Nehmen wir an, es sei eine gerade Anzahl $P \in 2\mathbb{N}$ von Prozessoren gegeben und der Torus $\mathbb{T}^d$ wird gemäß der Datenaufteilung der FFTW in P gleichgroße Quader $Q^{\boldsymbol{p}}$ entlang der ersten Dimension zerlegt, so enthält die Hälfte der Quader $Q^{\boldsymbol{p}}$ keine Knoten $\boldsymbol{x}_k$. Somit nimmt die Hälfte der Prozessoren auch nicht an der Berechnung von Schritt 2 in Algorithmus 3.2 teil. Selbiges folgt aus der Forderung $\|\boldsymbol{y}_j\| \leq \frac{1}{4} - \varepsilon_B$, $j \in \mathcal{M}_2$, für die parallele NFFT gemäß Algorithmus 3.1.

5.2 Optimierte trigonometrische Approximation der Kernfunktion

In [32] wurden Laufzeitvergleiche der seriellen FMM und der seriellen NFFT-basierten schnellen Summation in $d = 3$ Dimensionen für verschiedene Verteilungen der Knoten vorgestellt. Es stellt sich heraus, dass die auf der NFFT basierende schnelle Summation um einen Faktor 2 bis 4 langsamer ist als die FMM. Der entscheidende Unterschied ist auf die zeitaufwendige Berechnung im Nahfeld zurückzuführen. Wir wollen daher einen alternativen Ansatz zur Approximation der Kernfunktion $\mathcal{K}$ vorschlagen, welche es bei gleicher Approximationsgüte des Kernes im Fernfeld ermöglichen soll das Nahfeld kleiner zu wählen. Für den genauen Zusammenhang zwischen der Approximationsgüte des trigonometrischen Polynoms $T_{\boldsymbol{N}}$ zur periodisierten Kernfunktion $\tilde{\mathcal{K}}$ und dem Radius des Nahfeldes verweisen wir auf [28].

Um eine neue Approximationsidee zu motivieren, stellen wir die Grundgedanken der Fehlerabschätzung zur mehrdimensionalen schnellen Summation aus [28] vor. Der Fehler der in Abschnitt 5.1 konstruierten Approximation $\check{f}$ an f lässt sich abschätzen durch

$$\left|f(\boldsymbol{x}) - \check{f}(\boldsymbol{x})\right| = \Big| \sum_{k \in \mathcal{M}_1} \alpha_k (\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}}))(\boldsymbol{x}_k - \boldsymbol{x}) \Big| \leq \|\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[\mathbb{T}^d]} \sum_{k \in \mathcal{M}_1} |\alpha_k|. \tag{5.3}$$

In den Arbeiten [28, 10] wird nun die Norm $\|\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[\mathbb{T}^d]}$ für verschiedene Regularisierungen $\tilde{\mathcal{K}}$ abgeschätzt. Da $\tilde{\mathcal{K}} \in C^r[\mathbb{T}^d]$ ist kann der schnelle Abfall der Fourier-Koeffizienten $c_k(\tilde{\mathcal{K}})$ ausgenutzt werden.

Das trigonometrische Polynom

$$T_{\boldsymbol{N}}(\tilde{\mathcal{K}})(\boldsymbol{x}) := \sum_{\boldsymbol{l} \in I_{\boldsymbol{N}}} b_{\boldsymbol{l}} \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}},$$

mit den Koeffizienten $b_{\boldsymbol{l}} \in \mathbb{C}$, $\boldsymbol{l} \in I_{\boldsymbol{N}}$, aus der diskreten Fourier-Transformation

$$b_{\boldsymbol{l}} := \frac{1}{|I_{\boldsymbol{N}}|} \sum_{\boldsymbol{k} \in I_{\boldsymbol{N}}} \tilde{\mathcal{K}}(\boldsymbol{N}^{-1} \odot \boldsymbol{k}) \mathrm{e}^{-2\pi \mathrm{i} \boldsymbol{k} (\boldsymbol{N}^{-1} \odot \boldsymbol{l})}, \quad \boldsymbol{l} \in I_{\boldsymbol{N}},$$

ist eine Approximation an das trigonometrische Polynom $\sum_{\boldsymbol{l} \in I_{\boldsymbol{N}}} c_{\boldsymbol{l}}(\tilde{\mathcal{K}}) \mathrm{e}^{+2\pi \mathrm{i} \boldsymbol{l} \boldsymbol{x}}$, welches bekanntlich die Abweichung zu $\tilde{\mathcal{K}}$ gemessen in der $L^2(\mathbb{T}^d)$-Norm unter allen trigonometrischen Polynomen vom Grad $\boldsymbol{N}$ minimiert. Jedoch interessiert uns für die Fehlerabschätzung (5.3)

nicht der Fehler im quadratischen Mittel, sondern vielmehr die maximale absolute Abweichung $\|\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[\mathbb{T}^d]}$. Da die Funktion $\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})$ nie außerhalb der Kugel $B_{\frac{1}{2}-\varepsilon_B}$ ausgewertet wird, können wir uns auf $\|\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[B_{\frac{1}{2}-\varepsilon_B}]}$ einschränken.

Wir definieren zur Abkürzung den d-dimensionalen abgeschlossenen Ring $R_{\varepsilon_I,\varepsilon_B} := \{\boldsymbol{x} \in \mathbb{T}^d : \varepsilon_I \leq \|\boldsymbol{x}\|_2 \leq \frac{1}{2} - \varepsilon_B\}$ um den Ursprung und betrachten die Optimierungsaufgabe

$$\Big\|\mathcal{K} - \sum_{\boldsymbol{l}\in I_{\boldsymbol{N}}} b_{\boldsymbol{l}}^* \mathrm{e}^{+2\pi \mathrm{i}\boldsymbol{l}\boldsymbol{x}}\Big\|_{C[R_{\varepsilon_I,\varepsilon_B}]} = \inf_{\boldsymbol{b}\in\mathbb{C}^{|I_{\boldsymbol{N}}|}} \Big\|\mathcal{K} - \sum_{\boldsymbol{l}\in I_{\boldsymbol{N}}} b_{\boldsymbol{l}} \mathrm{e}^{+2\pi \mathrm{i}\boldsymbol{l}\boldsymbol{x}}\Big\|_{C[R_{\varepsilon_I,\varepsilon_B}]}.$$

Die Existenz eines Vektors $\boldsymbol{b}^* = (b_{\boldsymbol{l}}^*)_{\boldsymbol{l}\in I_{\boldsymbol{N}}} \in \mathbb{C}^{|I_{\boldsymbol{N}}|}$ und damit eines trigonometrischen Polynoms $T_{\boldsymbol{N}}^*(\mathcal{K}) := \sum_{\boldsymbol{l}\in I_{\boldsymbol{N}}} b_{\boldsymbol{l}}^* \mathrm{e}^{+2\pi \mathrm{i}\boldsymbol{l}\cdot}$ liefert folgender Satz aus der Approximationstheorie, siehe [2, S. 277 ff].

Satz 5.1 (Existenzsatz)**.** *Sei X ein linearer normierter Raum mit Norm $\|\cdot\|_X$ und $U := \{\phi : \phi = \sum_{k=0}^n a_k\varphi_k, \varphi_k \in X$ linear unabhängig$\}$ ein endlich dimensionaler Unterraum von X, so existiert mindestens eine beste Approximation $\phi^* := \sum_{k=0}^n a_k^*\varphi_k \in U$, d.h. es gilt*

$$\|f - \phi^*\|_X = \inf_{\phi\in U} \|f - \phi\|_X$$

Zur Anwendung von Satz 5.1 setzen wir $X = C[R_{\varepsilon_I,\varepsilon_B}]$ und $U = \operatorname{span}\{\mathrm{e}^{+2\pi \mathrm{i}\boldsymbol{l}\cdot} : \boldsymbol{l} \in I_{\boldsymbol{N}}\}$. Nun können wir die Funktion $\breve{\mathcal{K}}$ durch

$$\breve{\mathcal{K}}(\boldsymbol{x}) := \begin{cases} T_{\boldsymbol{N}}^*(\mathcal{K})(\boldsymbol{x}) & : \|\boldsymbol{x}\|_2 < \varepsilon_I, \\ \mathcal{K}(\boldsymbol{x}) & : \text{sonst}, \end{cases}$$

definieren, wobei wiederum $0 < \varepsilon_I < \frac{1}{2} - \varepsilon_B < \frac{1}{2}$ gilt. Setzen wir $\mathcal{K} = (\mathcal{K} - \breve{\mathcal{K}}) + (\breve{\mathcal{K}} - T_{\boldsymbol{N}}^*(\mathcal{K})) + T_{\boldsymbol{N}}^*(\mathcal{K})$ in unser Ausgangsproblem (5.1) für die schnelle Summation ein, und fordern, dass der Term $(\breve{\mathcal{K}} - T_{\boldsymbol{N}}^*(\mathcal{K}))(\boldsymbol{x})$ klein wird, so ergibt sich die Approximation

$$f(\boldsymbol{x}) \approx \breve{f}(\boldsymbol{x}) := \sum_{k\in\mathcal{M}_1} \alpha_k(\mathcal{K} - \breve{\mathcal{K}})(\boldsymbol{x}_k - \boldsymbol{x}) + \sum_{k\in\mathcal{M}_1} \alpha_k T_{\boldsymbol{N}}^*(\mathcal{K})(\boldsymbol{x}_k - \boldsymbol{x}). \tag{5.4}$$

Die Werte $\breve{f}(\boldsymbol{y}_j)$, $j \in \mathcal{M}_2$, können wir völlig analog zur Berechnung der Werte $\tilde{f}(\boldsymbol{y}_j)$, $j \in \mathcal{M}_2$, in (5.2) auswerten, da $T_{\boldsymbol{N}}^*$ ein trigonometrisches Polynom vom Grad $\boldsymbol{N}$ ist und die Funktion $\mathcal{K} - \breve{\mathcal{K}}$ auf Grund der Definition von $\breve{\mathcal{K}}$ den kompakten Träger B_{ε_I} besitzt. Wir erhalten also die Komplexität von $\mathcal{O}(|I_{\boldsymbol{N}}| \log |I_{\boldsymbol{N}}| + (M_1 + M_2)(\log \frac{1}{\varepsilon})^d)$ arithmetischen Operationen.

Der Fehler der Approximation $\breve{f}$ an f lässt sich für alle $\boldsymbol{x} \in B_{\frac{1}{2}-\varepsilon_B}$ analog zu (5.3) abschätzen durch

$$\Big|f(\boldsymbol{x}) - \breve{f}(\boldsymbol{x})\Big| = \Big|\sum_{k\in\mathcal{M}_1} \alpha_k(\breve{\mathcal{K}} - T_{\boldsymbol{N}}^*(\mathcal{K}))(\boldsymbol{x}_k - \boldsymbol{x})\Big| \leq \|\mathcal{K} - T_{\boldsymbol{N}}^*(\mathcal{K})\|_{C[R_{\varepsilon_I,\varepsilon_B}]} \sum_{k\in\mathcal{M}_1} |\alpha_k|. \tag{5.5}$$

Da $T_{\boldsymbol{N}}^*$ unter allen Polynomen vom Grad $\boldsymbol{N}$ die optimale Approximation an $\mathcal{K}$ in der Norm $\|\cdot\|_{C[R_{\varepsilon_I,\varepsilon_B}]}$ ist und $\mathcal{K}(\boldsymbol{x}) = \tilde{\mathcal{K}}(\boldsymbol{x})$ für $\boldsymbol{x} \in R_{\varepsilon_I,\varepsilon_B}$ gilt, erhalten wir

$$\|\mathcal{K} - T_{\boldsymbol{N}}^*(\mathcal{K})\|_{C[R_{\varepsilon_I,\varepsilon_B}]} \leq \|\mathcal{K} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[R_{\varepsilon_I,\varepsilon_B}]} = \|\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[R_{\varepsilon_I,\varepsilon_B}]} \leq \|\tilde{\mathcal{K}} - T_{\boldsymbol{N}}(\tilde{\mathcal{K}})\|_{C[\mathbb{T}^d]}.$$

Die Fehlerabschätzung (5.5) unserer neuen Approximation $\breve{f}$ ist also mindestens so gut wie die Fehlerabschätzung (5.3) der ursprünglichen Approximation $\check{f}$, unabhängig von der verwendeten Periodisierung $\tilde{\mathcal{K}}$. Die Bestimmung der optimalen Koeffizienten $b_{\boldsymbol{l}}^*$, $\boldsymbol{l} \in I_{\boldsymbol{N}}$, des trigonometrischen Polynoms $T_{\boldsymbol{N}}^*$ kann zumindest im eindimensionalen Fall mit Hilfe des Remez-Algorithmus [23, S. 168 ff] erfolgen. Sie soll in einer späteren Arbeit untersucht werden.

6

Zusammenfassung

6.1 Auswertung

In der vorliegenden Arbeit haben wir die parallelen Formulierungen der Algorithmen zur schnellen Fourier-Transformation nichtäquidistanter Daten und deren Adjungierten aus [29] für den dreidimensionalen Fall entwickelt. Beide Algorithmen wurden unter Verwendung der parallelen FFTW als Programmbibliothek in C mit MPI implementiert. Die Benutzerschnittstelle orientiert sich stark an den bekannten Programmbibliotheken der FFTW [12] und der NFFT [19]. Sie ermöglicht dem Nutzer eine einfache und komfortable Handhabung. Durch Laufzeittests mit bis zu 512 Prozessoren auf den drei Höchstleistungsrechnern CHiC, JUMP und JUGENE haben wir eine gute Skalierung nachgewiesen. So erreichten wir beispielsweise auf dem Parallelrechner JUGENE des Forschungszentrums Jülich für eine NFFT der Größe $196 \times 196 \times 196$ mit 512 Prozessoren eine Beschleunigung von 320, was einer Effizienz von $62,6\%$ entspricht. Die Grenzen der Parallelisierung wurden durch die Analyse des Speicherbedarfs aufgezeigt. Es stellte sich heraus, dass die Datenverteilung der verwendeten FFT-Programmbibliothek einen großen Einfluss auf die maximale Transformationsgröße der NFFT und auf die Anzahl der maximal verwendbaren Prozessoren hat. Deshalb haben wir neben der in der aktuellen Implementierung verwendeten FFTW eine alternative, parallele FFT vorgestellt und deren Speicherbedarf analysiert. Sie verspricht die Leistungsfähigkeit unserer Algorithmen weiter zu erhöhen. Mit der parallelen dreidimensionalen NFFT stellen wir ein leistungsfähiges Programmpaket zur Verfügung, welches bereits bis zu 512 Prozessoren gut skaliert und das Potential besitzt, zu einer hochskalierenden Programmbibliothek weiterentwickelt zu werden. Im zweiten Teil der Arbeit haben wir einen parallelen Algorithmus für die schnelle Summation [27, 28] entwickelt. Er beruht wesentlich auf den zuvor entworfenen parallelen Algorithmen zur NFFT und einer exakten Auswertung der Kernfunktion im Nahfeld. Um die Berechnungen im Nahfeld zu beschleunigen, wurde außerdem eine alternative Approximation für die Kernfunktion vorgeschlagen. Sie verringert die absolute Abweichung der Approximante von der Kernfunktion unter Erhaltung der Komplexität des Algorithmus.

6.2 Ausblick

Im Rahmen des vom Bundesministerium für Bildung und Forschung geförderten Projekts „ScaFaCoS - Skalierbare schnelle Löser für langreichweitige Wechselwirkungen“ ist die Weiterentwicklung der in dieser Arbeit vorgestellten Algorithmen geplant. Ziel des Projektes ist die Entwicklung und Implementierung hochskalierender Algorithmen zur Berechnung langreichweitiger Wechselwirkungen innerhalb einer frei verfügbaren Programmbibliothek. Sowohl die parallelen Algorithmen zur NFFT als auch die parallele schnelle Summation werden Bestandteile dieser Bibliothek sein.

Um den Anforderungen eines hochskalierenden Programmpakets gerecht zu werden, ist eine Implementation der NFFT mit einer alternativen FFT nötig. Eine weitere wesentliche Voraussetzung für die gute Skalierung der parallelen NFFT und ihrer Adjungierten ist die gleichmäßige Verteilung der Stützstellen. Es bleibt zu untersuchen, ob wir auch ohne diese Forderung eine gute Lastverteilung erreichen können.

Die in dieser Arbeit entworfene parallele schnelle Summation soll implementiert und getestet werden. Des Weiteren ist die Entwicklung und Implementation eines effizienten Algorithmus für die Berechnung der hier vorgeschlagenen optimierten Approximante der Kernfunktion notwendig.

Literaturverzeichnis

[1] R. K. Beatson und L. Greengard. A short course on fast multipole methods. In M. Ainsworth, J. Levesley, W. A. Light, und M. Marletta, Editoren, *Wavelets, Multilevel Methods and Elliptic PDEs*, Seiten 1 – 37, Oxford, 1997. Clarendon Press.

[2] I. S. Beresin und N. P. Shidkow. *Numerische Methoden I.* VEB Deutscher Verlag der Wissenschaften, 1970.

[3] G. Beylkin. On the fast Fourier transform of functions with singularities. *Appl. Comput. Harmon. Anal.*, 2:363 – 381, 1995.

[4] J. W. Cooley und J. W. Tukey. An algorithm for machine calculation of complex Fourier series. *Math. Comput.*, 19:297 – 301, 1965.

[5] A. J. W. Duijndam und M. A. Schonewille. Nonuniform fast Fourier transform. *Geophysics*, 64:539 – 551, 1999.

[6] A. Dutt und V. Rokhlin. Fast Fourier transforms for nonequispaced data. SIAM *J. Sci. Stat. Comput.*, 14:1368 – 1393, 1993.

[7] B. Elbel und G. Steidl. Fast Fourier transform for nonequispaced data. In C. K. Chui und L. L. Schumaker, Editoren, *Approximation Theory IX*, Seiten 39 – 46, Nashville, 1998. Vanderbilt University Press.

[8] M. Eleftheriou, J. E. Moreira, B. G. Fitch, und R. S. Germain. Parallel FFT subroutine library. http://www.alphaworks.ibm.com/tech/bgl3dfft.

[9] M. Eleftheriou, J. E. Moreira, B. G. Fitch, und R. S. Germain. A volumetric FFT for BlueGene/L. *Lecture Notes in Computer Science*, 2913:194 – 203, 2003.

[10] M. Fenn. Fast Fourier transform at nonequispaced nodes and applications. Dissertation, Universität Mannheim, 2006.

[11] J. A. Fessler und B. P. Sutton. Nonuniform fast Fourier transforms using min-max interpolation. IEEE *Trans. Signal Process.*, 51:560 – 574, 2003.

[12] M. Frigo und S. G. Johnson. FFTW, C subroutine library. http://www.fftw.org.

[13] M. Frigo und S. G. Johnson. FFTW manual. http://www.fftw.org/fftw3.3alpha_doc/.

[14] M. Frigo und S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the* IEEE, 93:216 – 231, 2005.

[15] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. MIT Press, Cambridge, 1988.

[16] L. Greengard und J.-Y. Lee. Accelerating the nonuniform fast Fourier transform. SIAM *Rev.*, 46:443 – 454, 2004.

[17] L. Greengard und V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325 – 348, 1987.

[18] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$–matrices, Part I: Introduction to $\mathcal{H}$–matrices. *Computing*, 62:89 – 108, 1999.

[19] J. Keiner, S. Kunis, und D. Potts. NFFT 3.0, C subroutine library. http://www.tu-chemnitz.de/~potts/nfft, 2006.

[20] J. Keiner, S. Kunis, und D. Potts. Using NFFT 3 - A software library for various nonequispaced fast Fourier transforms. *ACM Trans. Math. Software*, angenommen.

[21] S. Kunis. Nonequispaced FFT - generalisation and inversion. Dissertation, Institut für Mathematik, Universität zu Lübeck, http://www.tu-chemnitz.de/~skunis/, 2006.

[22] S. Kunis und D. Potts. Time and memory requirements of the nonequispaced FFT. *Sampl. Theory Signal Image Process.*, 7:77 – 100, 2008.

[23] G. Maeß. *Vorlesungen über numerische Mathematik II*. Akadamie-Verlag Berlin, 1988.

[24] D. Pekurovsky. P3DFFT, C subroutine library. http://www.sdsc.edu/us/resources/p3dfft.

[25] M. Pippig. Parallele schnelle Fourier-Transformation nichtäquidistanter Daten. In M. Bolten, editor, *Beiträge zum Wissenschaftlichen Rechnen, Ergebnisse des Gaststudentenprogramms 2007 des John von Neumann-Institus für Computing*, Seiten 91 – 105. 2007.

[26] G. Pöplau, D. Potts, und U. van Rienen. Calculation of 3D space-charge fields of bunches of charged particles by fast summation. In A. Anile, G. Alì, und G. Mascaly, Editoren, *Scientific Computing in Electrical Engineering*, Seiten 241 – 246. Springer, 2006.

[27] D. Potts und G. Steidl. Fast summation at nonequispaced knots by NFFTs. SIAM *J. Sci. Comput.*, 24:2013 – 2037, 2003.

[28] D. Potts, G. Steidl, und A. Nieslony. Fast convolution with radial kernels at nonequispaced knots. *Numer. Math.*, 98:329 – 351, 2004.

[29] D. Potts, G. Steidl, und M. Tasche. Fast Fourier transforms for nonequispaced data: A

tutorial. In J. J. Benedetto und P. J. S. G. Ferreira, Editoren, *Modern Sampling Theory: Mathematics and Applications*, Seiten 247 – 270. Birkhäuser, Boston, 2001.

[30] M. J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, first edition, 2004.

[31] G. Steidl. A note on fast Fourier transforms for nonequispaced grids. *Adv. Comput. Math.*, 9:337 – 353, 1998.

[32] T. Volkmer. Betriebspraktikum Jülich. Praktikumsbericht, Technische Universität Chemnitz, Fakultät für Mathematik, 2009.

[33] A. F. Ware. Fast approximate Fourier transforms for irregularly spaced data. SIAM *Rev.*, 40:838 – 856, 1998.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Chemnitz, den 11. März 2009

Michael Pippig