INAUGURAL – DISSERTATION
zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich–Mathematischen Gesamtfakultät
der
Ruprecht–Karls–Universität
Heidelberg

vorgelegt von
Diplom–Mathematiker Christian Wrobel
aus Augsburg
Tag der mündlichen Prüfung: 25.11.2003

Thema

Die Parallelisierung des filternden algebraischen Mehrgitterverfahrens zum Lösen partieller Differentialgleichungen

Gutachter: Prof. Dr. Gabriel Wittum
Prof. Dr. Peter Bastian

Science sans conscience n'est que ruine de l'âme.

(Wissenschaft ohne Gewissen ist nichts weiter
als der Ruin der Seele.)

François Rabelais, Schriftsteller (um 1494 – 1553)

Für Christine

Inhaltsverzeichnis

1 Einleitung

Viele Vorgänge in Naturwissenschaft und Technik werden durch partielle Differentialgleichungen modelliert. Zu ihrer numerischen Simulation werden sie in Raum und Zeit diskretisiert; es entstehen schließlich lineare Gleichungssysteme, die zu lösen sind. Feine Gebietsdetails, große Systeme von Differentialgleichungen und hohe Genauigkeitsanforderungen an die numerisch berechnete Lösung treiben die Anzahl der Unbekannten im linearen Gleichungssystem immer höher. Heute können daher vielfach nur stark vereinfachte Modelle zur Simulation eingesetzt werden, aber es besteht großes Interesse, diesem Mangel abzuhelfen. Sind heute lineare Gleichungssysteme mit Millionen von Unbekannten zu lösen, so werden es in Zukunft Milliarden sein.

Die dafür benötigte, immense Rechenleistung kann nur von Parallelrechnern bereitgestellt werden. Nur sie bieten sowohl den nötigen Hauptspeicher als auch die Prozessorleistung, um die Datenmengen zu verarbeiten. Wenn sehr große Prozessorzahlen benötigt werden, sind Parallelrechner vom MIMD–Typ (multiple instruction multiple data) am besten geeignet. Sie werden im SPMD (same program multiple data) Modell betrieben. Daher werden wir uns ausschließlich mit diesem Typ befassen.

Das bisher effizienteste Lösungsverfahren für lineare Gleichungssysteme ist das geometrische Mehrgitterverfahren [Hac85]. Die erste parallele Implementierung für ein lokales Mehrgitterverfahren für unstrukturierte Gitter stellte P. Bastian vor [Bas96b]. Die seitdem abgelaufene Weiterentwicklung zu einem auch in drei Raumdimensionen einsetzbaren, lokalen parallelen Mehrgitterverfahren für zeitabhängige, nichtlineare partielle Differentialgleichungen ist in [BBJ+97] [Lan99] [BBJ+99] [Lan00] [BBJ+00] [BJL+01] nachzulesen und in Form des Programmpakets *UG* [1] (unstructured grids) realisiert.

Trotzdem gibt es Klassen von partiellen Differentialgleichungen, für die bisher kein paralleles, effizientes geometrisches Mehrgitterverfahren erstellt werden konnte. Dazu zählt z. B. die Konvektions–Diffusions–Gleichung mit wirbelbehafteter Strömung. Darüber hinaus besteht auch bei geometrischen Mehrgitterverfahren eine Schwierigkeit, auf dem gröbsten Gitter effizient zu lösen, wenn es sehr viele Unbekannte umfaßt. Das kann z. B. auftreten, wenn die Geometrie feine Details aufweist und der Gittergenerator ein entsprechend feines Ausgangsgitter erzeugt. Für solche Anwendungen sind die algebraischen Mehrgitterverfahren (kurz AMG) prädestiniert. Sie vergröbern eine gegebene Matrix schrittweise und verwenden die so entstehende Hierarchie von Matrizen, um einen Mehrgitterlöser anzuwenden [RS87] [VMB96]

[1] http://cox.iwr.uni-heidelberg.de/~ug

[Stü99a]. Seriell wurden damit große Erfolge erzielt. Allerdings steckt ihre Parallelisierung noch in den Kinderschuhen, betrachtet man z. B. die in diesem Jahr auf zwei großen Konferenzen zu diesem Themengebiet vorgestellten Arbeiten [Cop00] [Str00].

Die bisher existierenden parallelen AMG setzen vorzugsweise auf sehr einfachen seriellen Verfahren auf und erzielen daher nicht immer optimale Effizienz; sie sind oft nicht für große Prozessorzahlen und/oder große Anzahlen von Unbekannten effizient einsetzbar. Das entspricht auch der allgemeinen Erfahrung, daß man für eine Parallelisierung ein möglichst gutes serielles Verfahren wählen sollte und nicht ein möglichst einfach zu parallelisierendes. Diesen Weg wollen wir konsequent verfolgen: Ausgangspunkt ist das filternde algebraische Mehrgitterverfahren (kurz FAMG) von C. Wagner [Wag00a]. Es ist sehr robust, allgemein einsetzbar und zeigt gute Effizienz. Seine Parallelisierung ist allerdings nicht trivial; wir werden verschiedene Möglichkeiten entwickeln und diskutieren. Ziel wird es sein, so wenig Abstriche wie möglich gegenüber dem originalen Verfahren zu machen, um so die guten numerischen Eigenschaften zu erhalten. Inwiefern dabei auch eine gute parallele Skalierbarkeit, sowohl bezüglich der Anzahl der Unbekannten als auch der Prozessoren, zu erzielen ist, müssen umfangreiche Experimente zeigen. Der hier entwickelte Rahmenalgorithmus läßt sich auf alle AMG Verfahren anwenden, die auf dem Selektionsprinzip basieren.

Das hier entwickelte Verfahren wurde in das oben bereits angesprochene Programmpaket *UG* integriert, weil es die zur Zeit am weitesten entwickelte Software zum parallelen Lösen von partiellen Differentialgleichungen ist und eine hervorragende Basis für eigene Entwicklungen bietet. Darin werden unter anderem die Module PPIF (parallel programming interface [Bas93]) als message passing Schnittstelle und DDD (dynamic distributed data [Bir98]) zum Handhaben eines verteilten Graphen benutzt.

Die Arbeit ist interdisziplinär ausgerichtet. Sie vereinigt mathematische Modelle und Methoden mit Konzepten der Informatik. Insbesondere wird die Korrektheit der entwickelten Algorithmen bewiesen. Dementsprechend werden beim Leser Grundkenntnisse im numerischen Lösen von partiellen Differentialgleichungen und deren Parallelisierung vorausgesetzt, wie sie etwa in [KGGK94] [dF94] [Haa99c] bereitgestellt werden.

Die Arbeit gliedert sich wie folgt.

In Kapitel 2 wird die Problemstellung eingeführt und der bisherige Stand der Lösungsverfahren grob zusammengefaßt. Im Fokus stehen dabei Mehrgitterverfahren, parallele Verfahren und Robustheit.

Das hier verwendete serielle Ausgangsverfahren FAMG wird in Kapitel 3 vorgestellt und seine Auswahl begründet.

Die Grundlagen für die Parallelisierung von Lösungsverfahren aus Sicht der linearen Algebra werden in Kapitel 4 gelegt. Erstmals werden auch durchgehend Beweise für die Korrektheit der parallelen linearen Algebra angegeben. Dafür sind neue Techniken erforderlich. Auf dieser Grundlage wird die Parallelisierung eines einfachen geometrischen Mehrgitterverfahrens dargestellt.

Hauptteil der Arbeit bildet Kapitel 5, in dem die Parallelisierung des FAMG entwickelt und bewiesen wird. Es werden an mehreren Stellen Varianten diskutiert, weil manchmal der natürliche und intuitive Weg ungünstig ist und ein trickreiches Vorgehen nötig ist. Die wichtigsten Bausteine sind: ein ausreichender Überlapp, eine gute Steuerung des Markierungsablaufs, für den auch eine parallele Graphfärbung erstellt wird, sowie das Sicherstellen der korrekten parallelen Darstellungsform der beteiligten algebraischen Objekte (Fein– und Grobgittermatrix ebenso wie Prolongations– und Restriktionsmatrix), das nur durch Beweise erfolgen kann.

In Kapitel 6 werden ausführliche Experimente durchgeführt, um die Anwendbarkeit des neuen Verfahrens zu demonstrieren und Leistungsmerkmale wie numerische Güte, Robustheit und parallele Skalierbarkeit darzustellen. Es werden anisotrope Poissongleichungen und Konvektions–Diffusions–Gleichungen mit bis zu 16,7 Millionen Unbekannten auf 128 Prozessoren gerechnet.

Abschließend werden die Ergebnisse in Kapitel 7 zusammengefaßt und Anregungen zu weiteren Arbeiten gegeben.

Dank

Mein Dank gilt Herrn G. Wittum für die Überlassung des Themas und sein Interesse an der Arbeit. Erst die großzügige Ausstattung mit allen nötigen Ressourcen — allen voran Rechenzeit auf großen Parallelrechnern — machte diese Arbeit möglich.

Allen Kollegen am ICA 3 der Universität Stuttgart und IWR der Universität Heidelberg danke ich für die sehr angenehme Arbeitsatmosphäre und ihre vielfältige Unterstützung, insbesondere dem *UG*–core Team für die Bereitstellung des Softwarepakets *UG*, das eine hervorragende Basis für die Realisierung des hier entwickelten Verfahrens darstellte: C. Wieners für seine Version der parallelen linearen Algebra, die parallelen Löser und seine Finite Elemente Anwendungsklasse, S. Lang und K. Birken für jede erdenkliche Hilfe bei der Überwindung so mancher Hürde bei der Parallelisierung und K. Johannsen für seine parallelisierte Anwendungsklasse für die Konvektions–Diffusions–Gleichung. Außerdem C. Wagner für die Überlassung des FAMG, die Unterstützung bei allen notwendigen Umbauten und seine geduldigen Erklärungen.

Bei der Durchführung der umfangreichen Rechnungen war mir die Betreibermannschaft der IBM RS6000 SP im Rechenzentrum der Universität Karlsruhe stets eine große Hilfe bei der Umschiffung vieler Klippen, die die Verwendung eines solchen Supercomputers mit sich bringt.

Über all die Durststrecken hinweg leistete mir meine Frau Christine Wrobel Beistand und mußte so manche Entbehrung hinnehmen. Auch für ihre tatkräftige Hilfe bei der Erstellung des druckfertigen Textes danke ich ihr sehr.

2 Iterative Löser

Zunächst werden die zu lösenden Gleichungen vorgestellt. Anschließend werden die wichtigsten derzeit bekannten Lösungsverfahren zusammengestellt: direkte Löser, iterative Löser und speziell geometrische und algebraische Mehrgitterverfahren.

2.1 Problemstellung

Inhalt dieser Arbeit ist eine neue, parallele Lösungsmethode für lineare Gleichungssysteme. Solche Gleichungssysteme entstehen typischerweise aus der Diskretisierung partieller Differentialgleichungen zweiter Ordnung

$$\nabla \cdot [D(x)\nabla u(x) + v(x)u(x)] - \sigma(x)u(x) = f(x), \text{ für } x \in \Omega$$

und geeigneten Randbedingungen.

Ist die Differentialgleichung linear, kann sie für das Lösen sofort diskretisiert werden. Übliche Methoden dazu sind Finite Differenzen (siehe z. B. [Hac86]), Finite Elemente (siehe z. B. [Bra97]) und Finite Volumen (siehe z. B. [Hac89b]). Eine verbreitete Methode für nichtlineare Differentialgleichungen besteht darin, sie nach der Diskretisierung im Rahmen eines Newtonverfahrens (siehe z. B. [Sto89]) zu linearisieren. In beiden Fällen ist das Ergebnis des Diskretisierungsprozesses ein lineares Gleichungssystem, das zu lösen ist

$$Au = f. \tag{2.1}$$

Eine wesentliche Eigenschaft dieser linearen Gleichungssysteme ist, daß sie *dünn besetzt* (englisch *sparse*) sind: Jede Zeile der Matrix enthält eine im Verhältnis zur Gesamtzahl der Unbekannten verschwindend geringe Anzahl von Einträgen ungleich Null. Das spiegelt die Lokalität der Differentialgleichung wider: Jede Unbekannte wird unmittelbar nur von ihren (geometrisch) direkten Nachbarn beeinflußt, die alleine nichttriviale Matrixeinträge verursachen. Für strukturierte Gitter kann man die Matrix (abgesehen von den Randbedingungen) durch Angabe ihres Sterns beschreiben (siehe dazu [Hac91]). Auf diese Dünnbesetztheit sollte auch der Löser Rücksicht nehmen, zumal das lineare Gleichungssystem für technisch interessante Anwendungen

sehr viele Unbekannte enthalten kann. Derzeit können es einige Millionen Unbekannte sein und für die Zukunft ist ein weiteres starkes Anwachsen unausweichlich, um immer komplexere Problemstellungen genügend genau berechnen zu können.

Daher sollten Lösungsverfahren einen Aufwand in Zeit und Speicher direkt proportional zur Anzahl der Unbekannten haben, eine Forderung, die von vielen gebräuchlichen Verfahren *nicht* erfüllt wird!

Dünn besetzte lineare Gleichungssysteme können aber auch durch Diskretisierung anderer partieller Differentialgleichungstypen oder aus gänzlich anderer Quelle stammen. Die Herkunft spielt für die betrachteten Lösungsverfahren aber zunächst keine Rolle.

2.2 Direkte Löser

Das klassische Verfahren zum Lösen linearer Gleichungen ist die *Gauß–Elimination*. Sie ist ein *direkter Löser* und in jedem Lehrbuch zur Numerik zu finden (z. B. [Sto89]). Die gegebene Matrix A wird in eine untere und obere Dreiecksmatrix L und U zerlegt, die als Produkt

$$A = L \cdot U$$

ergeben. Damit kann die lineare Gleichung (2.1) durch Vorwärts– und Rückwärtseinsetzen gelöst werden. Der dazu nötige Aufwand beträgt mit n als Zahl der Unbekannten für allgemeine, vollbesetzte Matrizen $O(n^2)$ im Speicher und $O(n^3)$ in der Zeit.

Für die hier betrachteten dünn besetzten Matrizen kann bei Verwendung strukturierter Gitter die entstehende **Bandstruktur** ausgenutzt werden und spezielle Varianten von direkten Lösern entwickelt werden. Sie haben dann einen Aufwand von $O(n^{3/2})$ im Speicher und $O(n^2)$ in der Zeit. Die dünn besetzte Struktur wird kaum ausgenutzt und erheblich aufgefüllt (englisch *fill in*), was zunächst zu einem überproportionalen Speicherbedarf führt, dessen Bearbeitung dann auch einen überlinearen Zeitaufwand verursacht. Auch spezielle Techniken wie Bandbreitenoptimierung, Skylineverfahren usw. können das Problem nur lindern, aber nicht grundsätzlich lösen [Meu99].

Diese Verfahrensklasse ist also nicht für das Lösen sehr großer Gleichungssysteme geeignet. Ein gänzlich anderes und besser geeignetes Vorgehen soll nun beschrieben werden.

2.3 Iterative Löser

Im Gegensatz zu direkten Lösern verfolgen iterative Löser einen vollkommen anderen Ansatz. Sie berechnen nicht mehr in einem Verarbeitungsschritt die (bis auf Maschinengenauigkeit) exakte Lösung des Gleichungssystems, sondern zu einer vorgegebenen Näherungslösung $u^{(n)}$ eine Korrektur $\Delta^{(n)}$, die eine (hoffentlich) bessere, neue

Näherungslösung $u^{(n+1)} := u^{(n)} + \Delta^{(n)}$ liefert. Man führt so lange weitere Iterationsschritte durch, bis die erreichte Genauigkeit ausreichend gut ist. Was „ausreichend gut“ im Einzelfall ist, muß jeder Anwender selbst spezifizieren. In der konkreten Berechnung der Korrektur $\Delta^{(n)}$ unterscheiden sich die einzelnen iterativen Verfahren.

Üblicherweise benötigen sie nur einen Speicheraufwand $O(n)$ und die Ausführungszeit für einen Iterationsschritt beträgt ebenfalls nur $O(n)$. Beide Maße sind also optimal. Allerdings ist damit noch keine (im allgemeinen auch nur annähernd) exakte Lösung berechnet. Wie viele solcher einzelnen Iterationsschritte benötigt werden, hängt von der Kondition der Matrix ab, die wiederum von der zugrunde liegenden Differentialgleichung und dem Diskretisierungsverfahren abhängt, und im allgemeinen leider auch von der Anzahl der Unbekannten. Trotzdem ist für sehr große Gleichungssysteme der Gesamtaufwand zur Berechnung einer ausreichend guten Lösung bei iterativen Verfahren meist geringer als bei direkten Lösern.

2.3.1 Definition; Grundalgorithmus

Eine übliche Formulierung iterativer Lösungsverfahren verwendet den Begriff der *angenäherten Inversen* W^{-1}, wobei die reguläre Matrix W eine derartige Approximation an A ist, daß die Gleichung

$$W\Delta^{(n)} = Au^{(n)} - f$$

einfach zu lösen ist (*dritte Normalform* nach [Hac91]). Der iterative Charakter wird in der *zweiten Normalform* nach [Hac91] noch deutlicher

$$u^{(n+1)} = u^{(n)} - W^{-1}(Au^{(n)} - f).$$

Formal kann man das Iterationsverfahren mit Hilfe der Matrix

$$M = \mathbb{1} - W^{-1}A \tag{2.2}$$

auch als

$$u^{(n+1)} = Mu^{(n)} + W^{-1}f$$

schreiben. Die Matrix M heißt **Iterationsmatrix**. Um die Konvergenz des Iterationsprozesses sicherzustellen, muß die Korrektur oft noch um einen Faktor ω *gedämpft* werden

$$u^{(n+1)} = u^{(n)} - \omega W^{-1}(Au^{(n)} - f).$$

Ein für die Praxis wichtiges Maß für die Konvergenzgeschwindigkeit eines Iterationsverfahrens ist die durchschnittliche Konvergenzrate k.

Definition 2.3.1 (Durchschnittliche Konvergenzrate)
Es seien n Schritte mit einem Iterationsverfahren durchgeführt worden. Dann heißt

$$k := \left(\frac{\|Au^{(n)} - f\|}{\|Au^{(0)} - f\|}\right)^{1/n}$$

die **durchschnittliche Konvergenzrate**.

Oft wird man n so wählen, daß der Anfangsdefekt $\|Au^{(0)} - f\|$ um einen gewählten Faktor (z. B. 10^{-8}) reduziert wird.

Das Vorgehen ist in Algorithmus 2.1 LinearSolver zusammengefaßt. Aufgerufen wird er mit einer konsistenten Kombination $u^{(0)}$ und $d^{(0)}$, am einfachsten mit $u^{(0)} = 0$ und $d^{(0)} = f$, wobei f die gegebene rechte Seite der linearen Gleichung ist.

Algorithmus 2.1 (LinearSolver(u, d))

(1) $\delta := ||d||$
(2) $\epsilon := \max\{ \delta\epsilon_{rel}, \epsilon_{abs} \}$
(3) while $\delta > \epsilon$
(4) $\quad c := 0$
(5) $\quad$ Iter(c, d) (1 Iterationsschritt ausführen)
(6) $\quad u := u + c$ (Lösungsupdate)
(7) $\quad \delta := ||d||$

Algorithmus 2.2 (Iter(c, d))

(1) Löse näherungsweise $At = d$: $\quad t := W^{-1}d$
(3) $d := d - At$
(4) $c := c + t$

Ein Glätter ist eine spezielle Funktion vom Typ Iter. Üblicherweise wird darin die Gleichung $At = d$ nur sehr grob gelöst. Beispiele sind das Jacobi-, Gauß–Seidel-, ILU-, SOR-Verfahren usw. [Hac91] [Meu99].

2.3.2 Einfache, klassische Iterationsverfahren

Die ersten derartigen Iterationsverfahren wurden von Carl Friedrich Gauß um 1820 und seinem Schüler Carl Gustav Jacobi (1845) und dessen Schüler Phillip Ludwig Seidel (1874) eingeführt [Hac91]

$W := D := \text{diag}(A)$ (Jacobi oder Gesamtschrittverfahren)
$W := D + L$ (Gauß–Seidel oder Einzelschrittverfahren)

wobei D nur die Diagonaleinträge von A enthält und L die echte untere Dreiecksmatrix von A ist.

Weitere wichtige Iterationsverfahren sind ILU–artige Verfahren (incomplete LU decomposition) und SOR-Verfahren (successive overrelaxation). Diese und eine Vielzahl von Modifikationen und weitere Verfahren finden sich in [Hac91] und [Meu99].

Analysen von Modellproblemen (z. B. [Hac91] und [Wag00a]) und praktische Erfahrungen zeigen unter anderem folgende zwei Eigenschaften von einfachen Iterationsverfahren.

Die Konvergenzrate strebt für eine steigende Anzahl von Unbekannten gegen 1, d. h. die Verfahren benötigen immer mehr Schritte, um eine geforderte Genauigkeit der Lösung erzielen zu können. Das macht einen Teil ihrer optimalen Speicher– und Zeitkomplexität für einen einzelnen Schritt zunichte.

Die Ursache dafür liefert eine Eigenwertanalyse der Verfahren. Es werden fast nur die hochfrequenten Komponenten des Fehlers reduziert. Nach einigen Iterationsschritten sind diese Komponenten größtenteils entfernt und der Fehler um wenige Größenordnungen reduziert. Die noch verbleibenden mittel– und langwelligen Komponenten können aber kaum noch reduziert werden und ab jetzt werden selbst für geringe Fehlerreduktionen viele Iterationsschritte benötigt. Die Effizienz bricht drastisch ein. Wichtig ist festzuhalten, daß „lang–“ bzw. „kurzwellig“ keine absolute Eigenschaft ist, sondern immer nur bezüglich eines Diskretisierungsgitters — genauer dessen Gitterweite — angegeben werden kann. Genau diese Eigenschaft wird von den nun vorzustellenden Mehrgitterverfahren ausgenutzt.

2.3.3 Mehrgitterverfahren

R. Fedorenko stellte 1961 und 1964 als erster ein Verfahren vor, das heute als *Zwei–* bzw. *Mehrgitterverfahren* bezeichnet wird [Fed61] [Fed64]. Erst A. Brandt und W. Hackbusch verhalfen dem Mehrgitterverfahren zum Durchbruch [Bra73] [Hac76]. Seitdem wurde es immer populärer. Viele weiterführenden Quellen sind in [Hac85] und [Wes92] zu finden. Eine einfache Einführung in dieses Thema stellt [BHM00] bereit. Der bahnbrechende Vorteil dieser Verfahren ist, daß sie oft mit Speicher– und Zeitaufwand $O(n)$ eine konstante Güte der Lösung erzielen und daher auch zur Lösung sehr großer Gleichungssysteme ideal geeignet sind.

2.3.3.1 Idee

Im vorigen Abschnitt wurde herausgestellt, daß einfache Iterationsverfahren nur bzgl. des Gitters hochfrequente Fehlerkomponenten effektiv reduzieren können. Man bezeichnet sie im Zusammenhang mit Mehrgitterverfahren daher auch als **Glätter**. Die Grundidee der Mehrgitterverfahren ist es nun, nach wenigen Iterationsschritten eines Glätters den verbleibenden relativ glatten Defekt auf ein gröberes Gitter zu restringieren. Dort kann der Defekt wegen seiner Glattheit gut approximiert werden, ist aufgrund der größeren Gitterweite wieder höherfrequenter und kann daher durch einige Glättungsschritte wieder effizient reduziert werden. Dieses Vergröbern kann solange durchgeführt werden, bis das entstehende Gleichungssystem so klein ist, daß es effizient durch ein direktes Verfahren gelöst werden kann. Die so auf gröberen Gittern erhaltene Näherungslösung ist nach Prolongation auf feinere Gitter zurück eine gute Korrektur der dortigen Näherungslösung. Bei Bedarf kann noch einmal nachgeglättet werden. Algorithmus 2.3 faßt das Vorgehen zusammen; er ist als ein iteratives Verfahren Iter im Rahmenalgorithmus 2.1 LinearSolver zu verwenden.

Algorithmus 2.3 (MG(c, d, ℓ))

(1)	if($\ell = \ell_{min}$)	
(2)	BaseSolver($^{\ell}c, {}^{\ell}d$)	(Baselevellöser)
	else	
(3)	for $i := 1$ to ν_1: Smooth($^{\ell}c, {}^{\ell}d$)	(Vorglättung)
(4)	${}^{\ell-1}d := {}^{\ell}R\,{}^{\ell}d$	(Restriktion)
(5)	${}^{\ell-1}c := 0$	(Startlösung auf Grobgitter)
(6)	for $i := 1$ to γ: MG$(c, d, \ell - 1)$	(Grobgitterkorrektur)
(7)	${}^{\ell}t := {}^{\ell}P\,{}^{\ell-1}c$	(Prolongation)
(8)	${}^{\ell}d := {}^{\ell}d - {}^{\ell}A\,{}^{\ell}t$	(Defektupdate)
(9)	${}^{\ell}c := {}^{\ell}c + {}^{\ell}t$	(Lösungsupdate)
(10)	for $i := 1$ to ν_2: Smooth($^{\ell}c, {}^{\ell}d$)	(Nachglättung)

Anschaulich gesprochen werden Fehleranteile verschiedener Frequenz auf jeweils ausreichend grobe Gitter verteilt und dort mit einfachen Iterationsverfahren effizient reduziert.

2.3.3.2 Die einzelnen Komponenten

Grundsätzlich muß man bei einem Mehrgitterlöser zwei Phasen unterscheiden: den Aufbau der Gitterhierarchie mit entsprechender Steifigkeitsmatrix und das eigentliche Lösen darauf. Auf den Gitteraufbau werden wir in den nächsten beiden Unterabschnitten eingehen. Hier wollen wir eine geeignete Gitterhierarchie als gegeben annehmen.

Von den vorherigen Abschnitten können alle Löser verwendet werden. Die iterativen werden mit nur wenigen Schritten als Vor– und Nachglätter eingesetzt (Zeilen (3) und (10)). Typisch sind 1 bis 3 Schritte. Auf dem gröbsten Gitter müssen alle noch vorhandenen Fehlerkomponenten entfernt werden. Daher benötigt man dafür einen echten Löser (Zeile (2)): entweder viele Iterationsschritte eines iterativen, um eine Defektreduktion um viele Größenordnungen zu erreichen, oder einen direkten Löser, wenn die Anzahl der Unbekannten klein genug ist. Neu eingeführt werden müssen die Gittertransferoperationen **Restriktion** und **Prolongation** (Zeilen (4) und (7)). Am einfachsten kann man sie auf einem regelmäßig verfeinerten Dreiecksgitter unter Verwendung von linearen Ansatzfunktionen konstruieren. Da im Fall der regelmäßigen Verfeinerung die Ansatzräume auf den 2 Gitterebenen ineinander geschachtelt sind, können die Grobgitter–Basisfunktionen ${}^{\ell-1}\Phi_{i'}$ als Linearkombination der Feingitter–Basisfunktionen ${}^{\ell}\Phi_j$ dargestellt werden

$$ {}^{\ell-1}\Phi_{i'} = \sum_{j \in J} \omega_{i'j}\, {}^{\ell}\Phi_j. $$

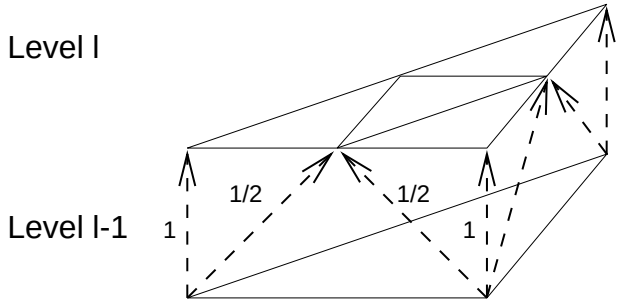


Abbildung 2.1: Prolongation an einer Dreiecksseite

Die Koeffizienten $\omega_{i'j}$ bilden zusammen die Restriktionsmatrix ${}^{\ell}R$. Die Lokalität der Basisfunktionen führt zu einer dünn besetzten Matrix ${}^{\ell}R$. Die adjungierte Matrix

$$ {}^{\ell}P = {}^{\ell}R^{\top} $$

definiert die Prolongationsmatrix ${}^{\ell}P$. In diesem Fall können die Koeffizienten $\omega_{ij'}$ als Interpolationsgewichte verstanden werden. Ein einfacher Fall ist in Abbildung 2.1 dargestellt und führt zur Formulierung

$$ p_{ij'} = \begin{cases} 1 & \text{falls } j' \text{ die Groblevelentsprechung von } i \text{ ist,} \\ \omega_{ij'} & \text{falls Knoten } i \text{ Interpolationsgewicht } \omega_{ij'} \text{ zu } j' \text{ hat,} \\ 0 & \text{sonst.} \end{cases} $$

Es gibt eine Vielzahl weiterer Möglichkeiten. Eine Vereinfachung mit entsprechend schlechteren Eigenschaften ist die Verwendung einer stückweise konstanten statt linearen Interpolation. Für komplizierte Gleichungen haben sich Interpolationsgewichte bewährt, die individuell für jeden Knoten berechnet werden und dabei z. B. die Einträge der Steifigkeitsmatrix berücksichtigen [dZ90] [Reu94] [WKW97]. Auch das hier verwendete Ausgangsverfahren FAMG bedient sich dieser Variante.

Nachdem nun auf dem gröberen Gitter das Hilfsproblem formuliert wurde: Löse

$$ {}^{\ell-1}A \; {}^{\ell-1}\Delta = {}^{\ell-1}d $$

mit ${}^{\ell-1}\Delta^{(0)} = 0$ als Startlösung (Zeile (5)), wird in Zeile (6) die Lösung näherungsweise durchgeführt. Es reicht hier ein iteratives Vorgehen. Üblicherweise kontrolliert man *nicht* die Genauigkeit der Lösung ${}^{\ell-1}\Delta$, sondern gibt die Anzahl der Iterationsschritte als Parameter γ vor. Da rekursiv weitere Hilfsprobleme gelöst werden, ergibt sich in Abhängigkeit von γ ein charakteristischer Zyklus der Abfolge der Komponenten, wie in Abbildung 2.2 dargestellt; die Namensgebung **V–** bzw. **W–Zyklus** ist offensichtlich.

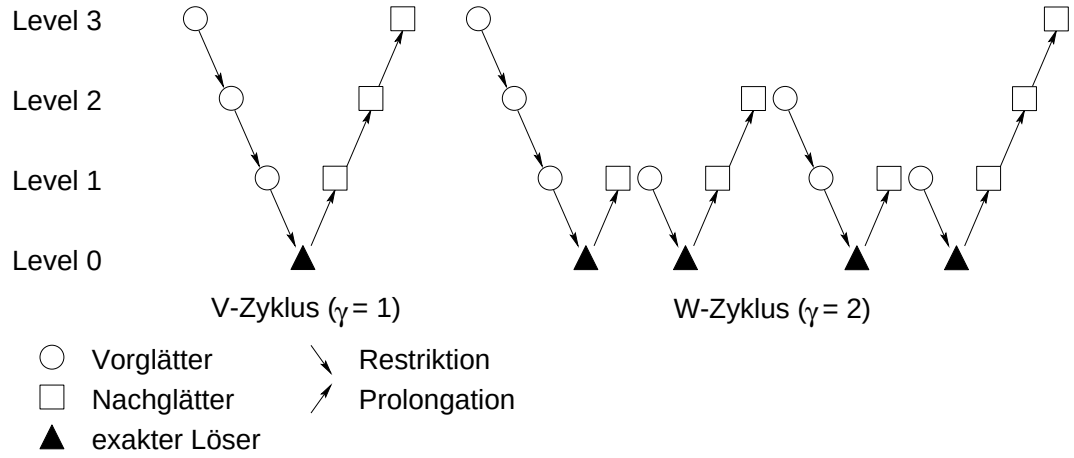


Abbildung 2.2: Übliche Mehrgitterzyklen

Nachdem die Korrektur ${}^{\ell-1}\Delta$ näherungsweise berechnet ist, kann sie auf das feinere Gitter prolongiert werden (Zeile (7)) und als Korrektur der bereits auf diesem Level bereitstehenden Korrektur aufaddiert werden (Zeile(9)) der Defekt muß daran angepaßt werden (Zeile(8)).

2.3.3.3 Geometrische Mehrgitterverfahren

Die am häufigsten eingesetzte Mehrgittermethode ist das **geometrische Mehrgitterverfahren**. Gegeben ist ein Gitter, das bereits alle relevanten Details der zu berechnenden Geometrie und der sie bestimmenden Koeffizienten auflöst. Darauf aufbauend wird das Gitter verfeinert, um die Lösung besser approximieren zu können. Diese Verfeinerung kann regelmäßig erfolgen oder nur lokal, um den Aufwand nur dort zu investieren, wo er auch tatsächlich benötigt wird. Bei letzterem Vorgehen bestimmt man die zu verfeinernden Bereiche mittels eines a priori Wissens über die Lösung oder man verwendet einen Verfeinerungsindikator (z. B. einen Fehlerschätzer); genauer ausgeführt und an Beispielen illustriert wird dieses Vorgehen z. B. in [BW93] [Bas96b] [BLE96].

In allen Fällen bildet die geometrische Hierarchie von Elementen die Grundlage dafür, eine Hierarchie von Gleichungssystemen (speziell zunächst von Matrizen) zu definieren. Entweder wird auf jeder Gitterebene erneut assembliert oder es wird mittels Galerkin–Assemblierung die auf dem feinsten Gitter assemblierte Steifigkeitsmatrix rein algebraisch auf die gröberen Level übertragen

$$ {}^{\ell-1}A := {}^{\ell}R\ {}^{\ell}A\ {}^{\ell}P. $$

Für technisch interessante Rechnungen kann die Größe des vorgegebenen Grobgitters zum Problem werden, weil auf ihm exakt gelöst werden muß. Sowohl direkte

als auch einfache iterative Löser besitzen schlechte Effizienz für große Gleichungssysteme (vergleiche Abschnitt 2.2 und 2.3.2). Abhilfe versprechen die algebraischen Mehrgitterverfahren.

2.3.3.4 Algebraische Mehrgitterverfahren

Gegeben sei ein Gitter, das bereits fein genug ist, um sowohl die Geometrie und die Koeffizienten der partiellen Differentialgleichung aufzulösen als auch die Lösung genau genug approximieren zu können. Dazu werden nun *gröbere* Level erzeugt, um die Mehrgitteridee einsetzen zu können. Streng genommen dürften solche Verfahren nur **Mehrlevelverfahren** (englisch **multilevel methods**) und nicht Mehrgitterverfahren genannt werden, da keine geometrische Struktur besteht. Wir wollen aber trotzdem beim üblichen Sprachgebrauch bleiben. Gute Übersichten sind in [Wag98] und [Stü99a] zu finden.

Das erste derartige Verfahren [BMR82] [Stü83] [RS87] und viele Weiterentwicklungen benutzen keinerlei Geometrieinformation, sondern nur die gegebene Feingittermatrix; daraus hat sich der Name **algebraisches Mehrgitterverfahren**, kurz AMG abgeleitet. Die Idee dabei ist immer, nach geeigneten Kriterien die Knotenmenge in Grobe und Feine zu unterteilen. Die Groben bilden das Gitter auf dem nächstgröberen Level und die Feinen werden daraus (über geeignet berechnete Gewichte) interpoliert. Zu dieser Klasse gehört auch das hier verwendete FAMG.

Inzwischen wurden auch Verfahren entwickelt, die in beschränktem Maße geometrische Information zur Konstruktion der gröberen Matrizen verwenden, z. B. [Sch96], [Wie00b] oder Aggregationsverfahren [Bra95] [VMB96].

Ein aktueller Überblick über den Entwicklungsstand algebraischer Mehrgitterverfahren ist in [Stü99a] und [Wag00a] enthalten.

2.3.3.5 Kombination beider Mehrgitterverfahren

Interessant ist die Verschmelzung beider Varianten, um sehr große, technisch relevante Anwendungen rechnen zu können. Ein Gittergenerator erzeugt ein Startgitter, das alle geometrischen Details und die Koeffizienten der partiellen Differentialgleichung genügend gut auflöst. Mittels AMG werden gröbere Level erzeugt, um das gegebene Gleichungssystem effizient lösen zu können. Sollte danach ein Verfeinerungsindikator anzeigen, daß die Lösung noch nicht ausreichend gut approximiert ist, können mittels geometrischem Mehrgitterverfahren (lokal) verfeinerte Gitterebenen erzeugt werden. Der Mehrgitterlöser führt dann jeweils die Iteration über *alle* Gitterebenen durch. Abbildung 2.3 veranschaulicht den Ablauf.

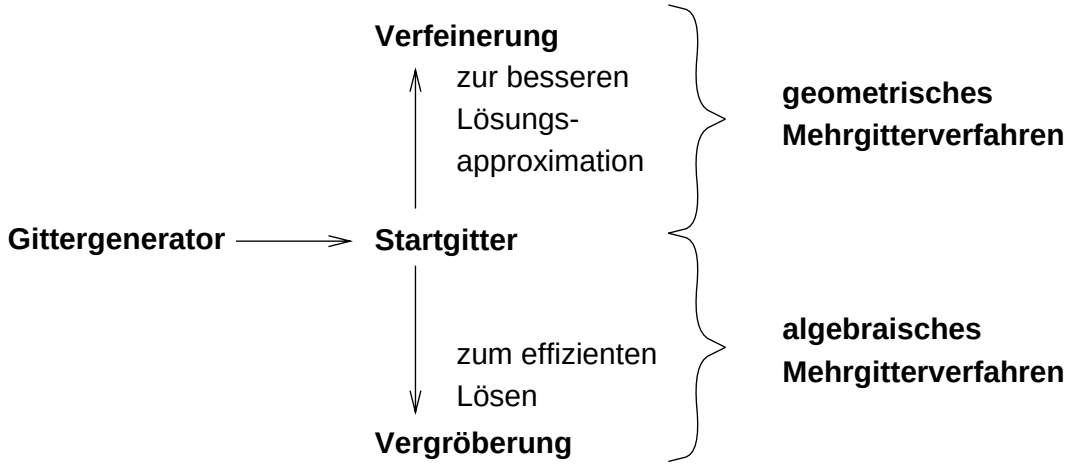


Abbildung 2.3: Zusammenspiel von geometrischem und algebraischem Mehrgitterverfahren

2.3.4 Krylovraummethoden

Das erste Verfahren dieser Klasse stellte 1952 E. Stiefel mit dem Verfahren der *konjugierten Gradienten* (kurz **cg**–Verfahren) vor [Sti52]. Dazu wird das Gleichungslösen

$$Au = f$$

zu einem Minimierungsproblem

$$\tfrac{1}{2}u^\top Au - u^\top f \to \min$$

umformuliert und mittels eines Gradientenverfahrens gelöst, das das Residuum auf dem Krylovraum

$$K(A, d, n) := \{d, Ad, A^2d, \ldots, A^{n-1}d\}$$

minimiert. Dabei ist der Defekt $d := f - Au$. Seitdem wurden eine Vielzahl von Modifikationen gefunden, die immer mehr Einschränkungen der Anwendbarkeit überwanden. Einen Überblick gibt [Joh00b]. Den vorläufigen Endpunkt stellt das Bi–CGSTAB Verfahren dar [vdV92], das auf unsymmetrische Matrizen angewendet werden kann, ausreichend stabil ist und keine Anwendung der transponierten Matrix benötigt.

Die Verfahren können selbständig oder mit einem anderen iterativen Verfahren als *Vorkonditionierer* eingesetzt werden. Im Zusammenhang mit Mehrgitterverfahren bevorzugen wir die umgekehrte Sichtweise: Das Krylovverfahren dient als *Beschleuniger* für das Mehrgitterverfahren. Nach [Hac91] lohnt sich dieses Vorgehen für das cg-Verfahren erst dann, wenn das pure Mehrgitterverfahren so schlecht wird, daß seine Konvergenzrate mindestens 0,4 beträgt.

2.3.5 Parallele Löser

Um die Ressourcen, die Parallelrechner bereitstellen, für das Lösen von linearen Gleichungssystemen effizient nutzen zu können, müssen die bisher vorgestellten, seriellen Algorithmen parallelisiert werden. Dieses Thema gewinnt zunehmend an Bedeutung und kann in diesem Abschnitt nur grob angerissen werden. Für ausführlichere Darstellungen sei auf [KST95] [SBG96] [Haa99c] verwiesen. Der Hauptteil dieser Arbeit beschäftigt sich in Kapitel 5 eben mit diesem Thema für das FAMG Verfahren.

Der für die Parallelisierung nötige Aufwand hängt sehr stark vom Ausgangsverfahren und den weiteren Rahmenbedingungen ab: Typ des Parallelrechners, Anzahl der einzusetzenden Prozessoren, verwendete Programmiersprache und Bibliotheken und gewünschte Effizienz. Da insbesondere elliptische Differentialgleichungen eine globale Abhängigkeit der Lösung von den Randdaten besitzen, sollte es nicht verwundern, daß im Prinzip jede Parallelisierung einen Kompromiß zwischen Laufzeiteffizienz, d.h. möglichst geringer und damit grobgranularer Kommunikation, und numerischer Güte finden muß.

Das größte Problem haben direkte Löser, weil die Gauß–Elimination ein strikt sequentieller Vorgang ist. Ideen, diese sequentielle Abhängigkeit aufzubrechen, sind in [Fro90] und [Meu99] zu finden. Trotz allen Bemühungen ist ein effizienter Einsatz nur für wenige Prozessoren möglich.

Iterative Verfahren haben den grundsätzlichen Vorteil, daß sie nicht exakt lösen, sondern nur eine Näherungslösung verbessern. Die Parallelisierung stellt zunächst eine Abwandlung der Näherung dar. Gewisse Güteeigenschaften sollten dabei allerdings möglichst erhalten bleiben. Am häufigsten wird der Datenpartitionierungsansatz verwendet, bei dem jeder Prozessor einen Teil des Rechengebiets samt den dazugehörigen Teilen des linearen Gleichungssystems erhält. Es bietet sich nun an, daß für einen Iterationsschritt jeder Prozessor zunächst auf seinen lokalen Daten das serielle Iterationsverfahren durchführt und anschließend in einem Datenaustauschschritt die neuen Ergebnisse für die auf anderen Prozessoren befindlichen Nachbarknoten erhält und in seine Ergebnisse integriert. Dieses Vorgehen wird als globales Block–Jacobi–Verfahren mit inexaktem Blocklöser bezeichnet.

Die Krylovraumverfahren lassen sich gut parallelisieren, wie in Abschnitt 4.2.12 vorgeführt wird. Sie sind allerdings auf einen guten Vorkonditionierer angewiesen, so daß die Hauptarbeit der Parallelisierung in der Parallelisierung dieses Vorkonditionierers steckt und das Problem rekursiv wieder auf diesen Abschnitt abgewälzt ist.

Eine bedeutende Klasse von geeigneten, parallelen Vorkonditionierern sind die **Gebietszerlegungsmethoden**. Sie werden als überlappende (**Schwarz–Methode**) oder nichtüberlappende Verfahren (**Schurkomplement** oder **Substrukturierungsmethode**) eingesetzt. Die Idee ist immer, auf den Teilgebieten unabhängig voneinander zu lösen und dann durch geeignete Verfahren im Bereich der Teilgebietsränder

eine globale Lösung herzustellen. Einen Überblick gibt [CM94] [GS95] [SBG96]; eine gute Informationsquelle ist auch die jährlich stattfindende Konferenz „International Symposium on Domain Decomposition Methods for Partial Differential Equations“, zu der bei SIAM jeweils ein Konferenzband erschienen ist.

Hauptproblem für ein gutes, d. h. von Gitterweite und Prozessorzahl unabhängiges und robustes Verfahren ist der globale Informationsaustausch. Mit dem BPS Vorkonditionierer [BPS86] und dem BPX Vorkonditionierer [BPX90] war der Anfang einer großen Vielfalt solcher Verbesserungen gelegt, die letztendlich zu mehrgitterähnlichen Algorithmen geführt haben.

Allen Gebietszerlegungsverfahren gemeinsam ist jedoch, daß sie nicht gut mit der Prozessoranzahl skalieren, und zwar sowohl wegen schlechter paralleler Effizienz als auch wegen nachlassender Konvergenzgüte.

2.3.6 Robustheit und Parallelisierung von Mehrgitterverfahren

Für praktische Anwendungen sollte das Lösungsverfahren gewisse Robustheitseigenschaften besitzen, weil man oft keine explizite Kontrolle über kritische Parameter hat und man damit nicht mehr maßgeschneiderte Speziallöser einsetzen kann.

In diesem Sinne sind alle Parameter, die die Konvergenzgüte des Lösers beeinflussen, als kritische Parameter anzusehen, insbesondere auch die Größe des Gleichungssystems und die Anzahl der verwendeten Prozessoren. Hier interessieren uns aber Parameter, die in der gegebenen Differentialgleichung bereits enthalten sind und deren Eigenschaften entscheidend beeinflussen: Für bestimmte (asymptotische) Parameterwerte ändert sich die Ordnung oder der Typ der Differentialgleichung. Eine solche Differentialgleichung heißt **singulär gestört**.

Hier werden zwei typische Beispiele diskutiert. In beiden Fällen geschieht der Typwechsel für den Parameterwert $\varepsilon = 0$. Die **anisotrope Poissongleichung**

$$\nabla \cdot \left(\begin{pmatrix} \varepsilon & 0 \\ 0 & 1 \end{pmatrix} \nabla u \right) = f \tag{2.3}$$

ist für $\varepsilon > 0$ eine elliptische Differentialgleichung zweiter Ordnung, für $\varepsilon = 0$ dagegen eine parabolische Differentialgleichung zweiter Ordnung. Die **Konvektions–Diffusions–Gleichung**

$$-\varepsilon \Delta u + v \nabla u = f \tag{2.4}$$

mit einem gegebenen Geschwindigkeitsfeld $v \in [\mathcal{C}^1(\mathbb{R}^2)]^2$ ist für $\varepsilon > 0$ eine elliptische Differentialgleichung zweiter Ordnung, für $\varepsilon = 0$ aber hyperbolisch von erster Ordnung.

Für singulär gestörte Differentialgleichungen geht der Parameter ε natürlich auch in das diskretisierte lineare Gleichungssystem ein

$$A(\varepsilon)u = f \tag{2.5}$$

und beeinflußt so auch die Lösbarkeitseigenschaften. G. Wittum führte den Begriff der **K(ε)–Robustheit** ein [Wit89]. Darauf aufbauend fordern wir für ein paralleles Verfahren zusätzlich noch eine Unabhängigkeit von der Prozessoranzahl p.

Definition 2.3.2 (Robustheit)
Ein paralleles Lösungsverfahren $M(\varepsilon)$ für (2.5) heißt **robust**, wenn für die asymptotische Konvergenzrate (oder Spektralradius) ϱ des Iterationsverfahrens $M(\varepsilon)$ gilt:

$$\varrho(M(\varepsilon)) \leq c < 1 \quad \forall \varepsilon > 0 \ \forall p.$$

Klassische Mehrgitterverfahren sind für singulär gestörte Differentialgleichungen üblicherweise nicht robust, so auch für die beiden angeführten Modellprobleme. Die Ursache dafür ist, daß die Grobgitterkorrektur für $\varepsilon \to 0$ zunehmend versagt. Um Mehrgitterverfahren robust zu machen, wurden mehrere Strategien entwickelt.

2.3.6.1 Halbvergröberung

Eine Methode ist die **Halbvergröberung** (englisch **semi–coarsening**). Sie soll zunächst aus der Sicht eines algebraischen Mehrgitterverfahrens motiviert werden [Stü99a]. Der Glätter kann den Fehler nicht mehr in allen Raumrichtungen gleich gut glätten, sondern nur noch in Richtung der starken Matrixeinträge. Das entspricht bei der anisotropen Poissongleichung der Richtung des starken Diffusionskoeffizienten und bei der Konvektions–Diffusions–Gleichung der Richtung der starken Strömung bei kleinem ε. Quer zu dieser Richtung bleibt der Fehler auch nach einigen Glättungsschritten noch ziemlich rauh und kann daher auf einem gröberen Gitter *nicht* gut approximiert werden. Es darf folglich nur in Richtung der starken Matrixeinträge vergröbert werden. Ein algebraisches Mehrgitterverfahren wird diese Vergröberungsrichtung jeweils lokal herausfinden.

Eine andere Motivation ergibt sich aus physikalischen Eigenschaften der Lösung. Gitter, die in der beschriebenen Richtung vergröbert werden, können innere Grenzschichten und Randschichten, sogenannte *Layer*, am besten auflösen.

Schwieriger ist die Situation bei geometrischen Mehrgitterverfahren, bei denen bereits eine Gitterhierarchie vorliegt und verwendet werden sollte. Selbst wenn man beim Gitteraufbau bereits ein a priori Wissen um die richtige Halbvergröberungsrichtung einfließen läßt, lassen sich solche Verfahren kaum von einfachen Modellproblemen auf technisch interessante Probleme übertragen, in denen die Effekte nicht mehr mit dem Gitter ausgerichtet sind und lokal unterschiedlich ausgeprägt sind. Ein Versuch besteht darin, jeweils in alle Richtungen die Halbvergröberung durchzuführen und so auf jedem Level viele Gitter zu erhalten, von denen hoffentlich immer eines lokal ausreichend gut ist [Hac89a] [Mul89] [Hac92] [NvR93] [OW95] [WO98].

2.3.6.2 Robuste Glätter

Bei diesem Ansatz wird ein einziges, regelmäßig verfeinertes Gitter verwendet, aber der Glätter wird speziell konstruiert. Er sollte nach [Hac85], [Wes81] und [Ket81] für den entarteten Fall, also z. B. $\varepsilon = 0$, ein *exakter Löser* sein. G. Wittum wies nach, daß mit fallendem Parameterwert die Güte des Glätters in einem ausreichenden Maße besser werden muß [Wit89]. An den beiden angeführten Modellproblemen läßt sich die Idee sehr gut nachvollziehen.

Für die anisotrope Poissongleichung (2.3) führt der Übergang $\varepsilon \to 0$ zu einem System von eindimensionalen Differentialgleichungen, die nur noch äußerst schwach gekoppelt sind. Auf strukturierten Dreiecksgittern mit linearen Finiten Elementen diskretisiert ergibt sich so ein System von tridiagonalen linearen Gleichungssystemen, die untereinander nur noch äußerst schwach gekoppelt sind. Für ein tridiagonales Gleichungssystem ist ein einziger ILU Schritt äquivalent zu einer exakten Gauß–Elimination, dann auch **Thomasalgorithmus** genannt. G. Wittum entwickelte eine Modifikation ILU_β, bewies für einige Modellprobleme eine ausreichende Güte und erzielte damit ein robustes Mehrgitterverfahren [Wit89]. Voraussetzung ist allerdings, daß die Unbekannten im Gleichungssystem so angeordnet sind, daß blockweise alle stark gekoppelten Unbekannten unmittelbar hintereinander stehen. Die Frequenzfilter Verfahren sind aus dem gleichen Grunde ebenfalls robuste Löser, wenn auch keine klassischen Mehrgitterverfahren [Wit92][Wag95][Hel96] [Hal98] [Buz98].

Für unstrukturierte Gitter ist diese Anordnung für den allgemeinen Fall nicht mehr möglich. Für 3 Raumdimensionen funktioniert obiges ILU_β Verfahren ebenso gut, wenn es 1 stark gekoppelte und 2 schwach gekoppelte Richtungen gibt. Für den umgekehrten Fall benötigt man einen Glätter, der gesamte, stark gekoppelte Ebenen exakt lösen kann; dafür kommt nur ein Mehrgitterverfahren in Frage. Die Unbekannten müssen dafür ebenenweise angeordnet werden. Der allgemeine, unstrukturierte, 3–dimensionale Fall ist somit noch weniger lösbar.

Für die Parallelisierung ist entscheidend, daß die einzelnen, stark gekoppelten Substrukturen (Linien bzw. Ebenen) jeweils *vollständig* auf einem Prozessor liegen, da nur so der Glätter darauf wie ein exakter Löser arbeiten kann. Das erfordert ein großes a priori Wissen schon in der Phase der Lastverteilung und das Herstellen eines Lastgleichgewichts wird zusätzlich erschwert. Die Anwendung auf allgemeine, unstrukturierte Probleme erscheint fast aussichtslos.

Bei der Konvektions–Diffusions–Gleichung mündet der Grenzfall $\varepsilon \to 0$ in einzelne, entkoppelte Stromlinien. Theoretische Untersuchungen existieren für den eindimensionalen Fall [Hac84]. J. Bey und G. Wittum geben ein Verfahren an, das für Modellprobleme Robustheit zeigt [BW97]. Die Idee ist, daß für die pure Strömung entlang einer Stromlinie das Gauß–Seidelverfahren bereits in einem Schritt ein exakter Löser ist, wenn die Unbekannten entlang der Strömung angeordnet sind (**down wind numbering**). Die Auswirkungen sind die gleichen, wie sie für die anisotrope

Poissongleichung diskutiert wurden, da die Robustheit ebenfalls auf einer problemspezifischen Anordnung der Unbekannten, die a priori durchgeführt werden muß, und einem darauf exakt arbeitenden, einfachen iterativen Verfahren beruht. Erschwerend kommt hinzu, daß bei wirbelbehafteten Strömungen zyklische Abhängigkeiten entlang geschlossener Charakteristiken entstehen. Dafür ist obiges Verfahren nicht geeignet. Ein Verfahren dieser Mächtigkeit wurde von K. Johannsen vorgestellt [Joh00a] [Joh00b]. Es vereinigt die stromabwärts gerichtete Numerierung der Unbekannten mit einem Verfahren zum Aufschneiden der geschlossenen Charakteristiken und einem frequenzfilternden Löser entlang dieser Schnittlinien.

Die Schwierigkeiten für die Parallelisierung sind noch größer: Es müssen nicht nur jeweils komplette Stromlinien auf einem Prozessor liegen, sondern auch alle Stromlinien, die gemeinsam einen Wirbel bilden. Das beschränkt die Zahl der einsetzbaren Prozessoren erheblich, erfordert großes a priori Wissen über die Strömung schon in der Lastverteilungsphase und läßt dem Lastbalancierer kaum Handlungsspielraum.

2.3.6.3 Matrixabhängiger Gittertransfer

Als dritte Klasse von robusten Mehrgitterverfahren haben sich diejenigen herausgebildet, die mit einer einzigen regelmäßig verfeinerten Mehrgitterhierarchie und nicht zu speziellen Glättern auskommen wollen. Dafür versuchen sie, die Gittertransferoperatoren für jeden Punkt individuell zu bestimmen [ABDP81] [dZ90] [Reu94] [WKW97]. Soweit das gelingt, lassen sich solche Verfahren gut parallelisieren, sofern sie nicht einen zu großen Bereich um einen Punkt in die Berechnung einbeziehen.

Zusammenfassend ist der erste Ansatz über eine lokale, problemangepaßte Halbvergröberung verbunden mit matrixabhängig berechneten Transferoperatoren für einen parallelen robusten Löser wesentlich vielversprechender als der Ansatz über robuste Glätter, weil er nicht nur für spezielle Modellprobleme, sondern allgemein anwendbar ist. Diesen Weg beschreitet auch das hier entwickelte parallele FAMG Verfahren.

3 Das FAMG Verfahren

Zunächst wird die Auswahl des seriellen Ausgangsverfahrens begründet, das hier verwendet wird, um einen neuen, parallelen Löser zu erstellen. Anschließend werden die Grundideen des Verfahrens besprochen und ihre algorithmische Realisierung dargestellt.

3.1 Auswahl des seriellen Ausgangsverfahrens

Da das hier zu entwickelnde Verfahren sowohl als selbständiges Lösungsverfahren als auch als Grobgitterlöser für bestehende geometrische Mehrgitterverfahren, deren Ausgangsgitter für eine effiziente Lösung durch einfache iterative oder direkte Verfahren zu groß sind, einsetzbar sein soll, kommt nach Kapitel 2 nur ein **algebraisches Mehrgitterverfahren** in Frage. Zu Beginn der Arbeit gab es noch kein derartiges Verfahren. Die Parallelisierung von algebraischen Mehrgitterverfahren ist immer noch ein aktuelles Forschungsgebiet. Es beschäftigen sich mehrere Arbeitsgruppen mit diesem Thema. Ziel ist es, robuste und mit der Prozessoranzahl gut skalierbare Verfahren zu entwickeln, die auch für sehr viele Unbekannte nicht zu sehr an Effizienz verlieren.

AMGe artige Verfahren sind zunächst auf Finite Element Anwendungen beschränkt und sind für uns damit zu eingeschränkt. G. Haase/S. Reitzinger[1] und eine Gruppe um van Emden Henson[2] führen gerade Parallelisierungen durch. Aggregationsverfahren sind sehr einfach zu parallelisieren, scheiden aber aus, weil sie für zunehmende Problemgröße oft in ihrer Konvergenzgüte nachlassen [Rob93] [LMR+98] [Mav98] [VBM98], und wir aber auch sehr große Gleichungssysteme rechnen wollen. Dafür wurde schon sehr früh eine erste Parallelisierung durchgeführt [Rob91] [Rob93]. ILU–artige und Blockeliminationsverfahren [BS97] [BW99] stellen aufgrund ihres prinzipiell schrittweisen Vorgehens grundsätzlich jeder Parallelisierung die gleichen großen Schwierigkeiten entgegen wie direkte Löser. Bekannte Ansätze zur Parallelisierung direkter Löser sind nur für sehr wenige Prozessoren tragfähig [Fro90] und nicht allgemein einsetzbar. Daher verheißen auch diese beiden AMG Klassen keinen Erfolg für eine Parallelisierung.

[1] PEBBLES (Parallel and Element Based Grey Box Linear Equation Solver); http://www.sfb013.uni-linz.ac.at/~reitz/pebbles.html

[2] http://www.llnl.gov/casc

Folglich bleiben nur noch AMG Verfahren nach dem Selektionsprinzip übrig, die allgemein einsetzbar, robust und ziemlich h–unabhängig sind [Stü99a]. Diesen Ansatz wählen auch die meisten zwischenzeitlich publizierten parallelen AMG Verfahren [KS99] [WJ99]. Ein Hauptproblem besteht darin, daß sie üblicherweise — wie etwa Verfahren vom Ruge–Stüben Typ — einen genügend starken Glätter benötigen; meist ist es ein Gauß–Seidel Glätter. Für größere Prozessorzahlen läßt die Güte der Glätter aber mit Ausnahme des Jacobiverfahrens nach. V. Henson beobachtete dieses Verhalten sehr stark. Da das FAMG Verfahren von C. Wagner mit Jacobiglätter auch für schwierige Gleichungen auskommt [Wag00a] [Wag00b], erscheint es als der vielversprechendste Kandidat für eine Parallelisierung und wurde daher ausgewählt.

3.2 Motivation

Bisherige Erfahrungen zeigten schon, daß die Auswahl der Elternknoten für die Interpolation eines Knotens extrem wichtig ist [Stü99b]. Das große Problem dabei ist allerdings, daß aus der puren Betrachtung der Matrixeinträge nicht einmal zwischen 2 extremen Elternpaaren, wie sie in Abbildung 3.1 dargestellt sind, unterschieden werden kann: Für eine isotrope Gleichung haben alle 4 zu betrachtenden Matrixeinträge den gleichen Wert, obwohl das gestreckte Elternpaar (rechter Bildteil) die wesentlich besseren Interpolationseigenschaften besitzt und das Mehrgitterverfahren damit wesentlich besser konvergiert.

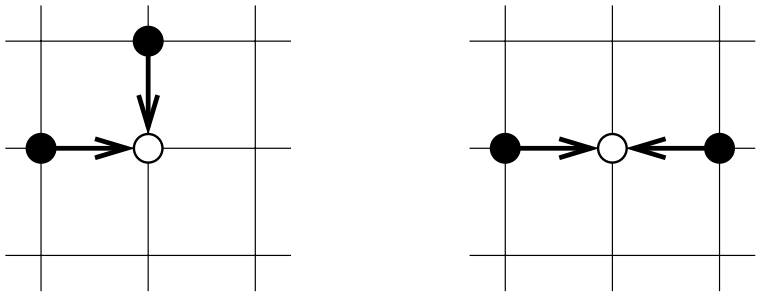


Abbildung 3.1: Schlechtes (links) und gutes Elternpaar (rechts)

In [Stü99a] werden eine Vielzahl von Heuristiken beschrieben, die alle versuchen, durch Vergleiche der Größe der Matrixeinträge *und* eine geschickte Steuerung der Markierungsfront, möglichst nur gute Elternpaare zu erzeugen. Der Mangel an einer expliziten Kontrolle wird in [Stü99a] ausdrücklich beklagt. C. Wagner gelang es nun in [Wag00b] schon bei der Elternpaarauswahl gute von schlechten Paaren zu unterscheiden. Die grundlegende Erkenntnis ist, daß die Norm eines geglätteten Vektors, der den Interpolationsfehler für die Prolongation mit verschiedenen Elternpaaren beschreibt, gute von schlechten Paaren unterscheiden läßt. Die Norm ist um so kleiner,

je besser das Elternpaar den zwischenliegenden Knoten interpolieren kann. Diese Bewertung erfordert keinerlei geometrische Information und ist daher für ein AMG bestens geeignet, gute Elternpaare aus der Menge der möglichen Paare herauszufinden.

Der zentrale Punkt ist folglich, eine gute Prolongations– und Restriktionsmatrix zu konstruieren. Das erfordert sowohl die Auswahl der Elternpaare als auch die Berechnung der Koeffizienten der beiden Gittertransfermatrizen.

3.3 Konstruktion der Prolongation und Restriktion

Dieser Abschnitt faßt die Ergebnisse in [Wag00a] zusammen. Für weitere Einzelheiten sei auf diese Arbeit verwiesen. Da sich alle Ausführungen auf ein festes Paar ℓ und $\ell - 1$ von Leveln bezieht, wird der Levelindex weggelassen. Für die Herleitung sei die Matrix A als symmetrisch, positiv definit angenommen. J bezeichne in Vorgriff auf Definition 4.1.3 die Indexmenge zur Matrix A. Wir werden hauptsächlich die Konstruktion der Prolongation beschreiben. Für die Restriktion geht man analog vor; deren Größen werden mit einem Querbalken gekennzeichnet.

Ziel ist es, eine gute Konvergenz des Mehrgitterverfahrens zu erzielen. Dazu sollte die Norm der Iterationsmatrix des Zweigitterverfahrens mit einem Vor– und Nachglättungsschritt des Glätters S_1 bzw. S_2

$$\begin{aligned} T_{TL} :=& S_2(I - PA_c^{-1}RA)S_1 \\ =& S_2(I - PA_c^{-1}RA)(I - PA_c^{-1}RA)S_1 \end{aligned} \tag{3.1}$$

mit der Grobgittermatrix $A_c := RAP$ möglichst klein sein. Das gelingt, wenn P und R derartig konstruiert werden, daß die beiden Faktoren

$$S_2(I - PA_c^{-1}RA) \tag{3.2}$$

und

$$(I - PA_c^{-1}RA)S_1 \tag{3.3}$$

in (3.1) klein sind.

Aufgabe ist es nun, die Menge der Knoten J in die zwei disjunkten Mengen der grobmarkierten Knoten $\mathcal{C}$ (auch C–Knoten genannt) und der feinmarkierten Knoten $\mathcal{F}$ (auch F–Knoten genannt) aufzuteilen. Dabei wird jedem F–Knoten i eine Menge $\mathbb{P}_i$ von C–Knoten als Elternknoten zugeordnet, aus denen er interpoliert wird. Die entsprechenden Prolongations– und Restriktionsgewichte müssen ebenfalls noch berechnet werden.

Die Energienorm von (3.3) läßt sich unter Verwendung der trivialen Restriktion

$$R_{i'j}^{inj} := \delta_{i'j}$$

abschätzen

$$\|(I - PA_c^{-1}RA)S_1\|_A = \|A^{1/2}(I - PA_c^{-1}RA)S_1A^{-1/2}\|_2 \tag{3.4}$$
$$\leq \|I - A^{1/2}PA_c^{-1}RA^{1/2}\|_2 \, \|A^{1/2}(I - PR^{inj})S_1A^{-1/2}\|_2. \tag{3.5}$$

Obwohl wir letztendlich R und P einzeln berechnen werden, nehmen wir hier $R = P^\top$ an. Damit ist der erste obige Faktor $\|I - A^{1/2}PA_c^{-1}RA^{1/2}\|_2 = 1$ und P und R sollten so bestimmt werden, daß die Konstante C in der Variationsungleichung für den zweiten Faktor möglichst klein wird

$$\|A^{1/2}(I - PR^{inj})S_1v\|_2 \leq C\|A^{1/2}v\|_2 \ \forall v. \tag{3.6}$$

$\|A^{1/2}v\|_2$ wird für glatte Vektoren sehr klein. Der Gittertransfer sollte nach den Überlegungen in Abschnitt 2.3.3 vor allem für glatte Vektoren gut funktionieren. Man wird P und R folglich so konstruieren, daß (3.6) besonders für glatte Vektoren mit einer kleinen Konstante erfüllbar ist. Diese Bedingung kann als **Filterbedingung** für den Testvektor t formuliert werden

$$(I - PR^{inj})S_1t = 0. \tag{3.7}$$

Den Testvektor wird man als glatt wählen, z. B. $t = (1, \ldots, 1)^\top$. Für weitergehende Überlegungen siehe [Wag00a, Abschnitt 3.5]. Von dieser Idee der Filterung rührt auch die Namensgebung **Filterndes AMG** (FAMG) her. Ähnliche Filtereigenschaften wurden auch in [Wit92] [Wag95] [WW97] verwendet.

Eine andere Möglichkeit, um in (3.6) eine kleine Konstante zu erreichen, besteht darin, die Einträge der Matrix $A^{1/2}(I - PR^{inj})S_1$ klein zu halten. Mit

$$D := \mathrm{diag}(A)$$

ist aufgrund der Filterbedingung (3.7) die Approximation $\sqrt{2}\, D^{1/2}$ für $A^{1/2}$ gerechtfertigt. Mittels der Frobeniusnorm

$$\|A\|_F := \sqrt{\sum_{i,\, j \in J} a_{ij}^2}$$

kann diese Forderung als

$$\min \|D^{1/2} \, (I - PR^{inj})S_1\|_F \tag{3.8}$$

formuliert werden.

Da die Anwendung des Glätters S_1 auf einen Vektor eine zeitkritische Operation für das FAMG darstellt, sollte er so einfach wie möglich gestaltet werden, ohne allerdings

zu sehr an Güte einzubüßen. In der Praxis hat sich folgender Kompromiß aus 3 speziellen Jacobischritten bewährt. Der Glätter an dieser Stelle sollte ähnlich dem Glätter im nachfolgenden Mehrgitterzyklus sein.

Zunächst genügt es, eine ausgedünnte Matrix $\acute{A}$ zu verwenden, da nach Praxiserfahrung die kleinen Elemente keinen wesentlichen Beitrag zur Güte des Ergebnisses leisten. Der Parameter σ gibt die relative Abschneideschwelle vor

$$\acute{a}_{ij} := \begin{cases} a_{ij} & \text{falls } i = j \text{ oder } |a_{ij}| \geq \sigma \max\limits_{j \neq i} |a_{ij}| \text{ oder } |a_{ji}| \geq \sigma \max\limits_{j \neq i} |a_{ji}| \\ 0 & \text{sonst.} \end{cases} \tag{3.9}$$

$\acute{N}(i)$ bezeichne die Menge der stark gekoppelten, direkten Nachbarknoten des Knotens i

$$\acute{\mathcal{N}}(i) := \{j \in J \setminus \{i\} \mid \acute{a}_{ij} \neq 0 \vee \acute{a}_{ji} \neq 0\}.$$

Im Gegensatz zur späteren Definition 4.1.19 ist hier der Knoten i nicht in $\acute{\mathcal{N}}(i)$ enthalten. Damit kann nun die Anwendung des speziellen Glättungsoperators $\acute{S}$ auf einen Vektor $q_i, i \in J$ für (3.8) definiert werden (ω ist ein Dämpfungsparameter)

$$(\acute{S}^\top q_i)_j := \begin{cases} y_j - \omega \sum\limits_{k \in \acute{\mathcal{N}}(j)} \frac{\acute{a}_{kj}}{\acute{a}_{kk}} y_k & \text{für } j \in \acute{N}(i) \\ y_j & \text{für } j \in \acute{\mathcal{N}}(\acute{\mathcal{N}}(i)) \setminus \acute{\mathcal{N}}(i) \\ 0 & \text{sonst} \end{cases} \tag{3.10}$$

mit

$$y_i := \begin{cases} z_j - \omega \sum\limits_{k \in \acute{\mathcal{N}}(j)} \frac{\acute{a}_{kj}}{\acute{a}_{kk}} z_k & \text{für } j \in \acute{\mathcal{N}}(\acute{\mathcal{N}}(i)) \cup \acute{\mathcal{N}}(i) \\ 0 & \text{sonst} \end{cases}$$

$$z_j := \begin{cases} (q_i)_j - \omega \frac{\acute{a}_{ij}(q_i)_i}{\acute{a}_{ii}} & \text{für } j \in \acute{\mathcal{N}}(i) \cup \{i\} \\ 0 & \text{sonst.} \end{cases}$$

Und analog gelte für die Restriktion

$$(\overline{\acute{S}}^\top \overline{q}_i)_j := \begin{cases} \overline{y}_j - \omega \sum\limits_{k \in \acute{\mathcal{N}}(j)} \frac{\acute{a}_{jk}}{\acute{a}_{kk}} \overline{y}_k & \text{für } j \in \acute{N}(i) \\ \overline{y}_j & \text{für } j \in \acute{\mathcal{N}}(\acute{\mathcal{N}}(i)) \setminus \acute{\mathcal{N}}(i) \\ 0 & \text{sonst} \end{cases} \tag{3.11}$$

mit

$$\overline{y}_i := \begin{cases} \overline{z}_j - \omega \sum\limits_{k \in \acute{\mathcal{N}}(j)} \frac{\acute{a}_{jk}}{\acute{a}_{kk}} \overline{z}_k & \text{für } j \in \acute{\mathcal{N}}(\acute{\mathcal{N}}(i)) \cup \acute{\mathcal{N}}(i) \\ 0 & \text{sonst} \end{cases}$$

$$\overline{z}_j := \begin{cases} (\overline{q}_i)_j - \omega \frac{\acute{a}_{ji}(\overline{q}_i)_i}{\acute{a}_{ii}} & \text{für } j \in \acute{\mathcal{N}}(i) \cup \{i\} \\ 0 & \text{sonst.} \end{cases}$$

Es gelte zur Vereinfachung $Q^\top := I - PR^{inj}$. q_i bezeichne die zum Knoten $i \in J$ gehörende Spalte von Q. Sie habe zu einem gegebenen Elternpaar $\mathbb{P}$ folgende Struktur

$$(q_i)_k = \begin{cases} 1 & \text{falls } k = i \\ -p_{ik} & \text{falls } k \in \mathbb{P} \\ 0 & \text{sonst} \end{cases} \quad \text{, für } i \in \mathcal{F}$$

$$q_i \equiv 0 \quad \text{, für } i \in \mathcal{C}.$$

Damit läßt sich das Minimierungsproblem (3.8) als

$$\begin{aligned} \min \|D^{1/2} Q^\top \acute{S}\|_F^2 &= \min \sum_{i,\, j \in J} |a_{ii}| \, (Q^\top \acute{S})_{ij}^2 = \min \sum_{i,\, j \in J} |a_{ii}| \, (\acute{S}^\top Q)_{ji}^2 \\ &= \min \sum_{i \in J} |a_{ii}| \, \|\acute{S}^\top q_i\|_2^2 = \sum_{i \in J} \min |a_{ii}| \, \|\acute{S}^\top q_i\|_2^2 \end{aligned} \tag{3.12}$$

schreiben und man erkennt, daß über die Vektoren q_i unabhängig voneinander minimiert werden kann. Diese Struktur ist aber viel zu feingranular, um sie für eine Parallelisierung zu nutzen, und entspricht nicht der vorhandenen Datenaufteilung.

Für eine gegebene Elternknotenmenge $\mathbb{P}_i$ bestimmen wir die optimalen Einträge p_{ik} in $q_i(\mathbb{P}_i)$ für die Prolongation nach (3.12) und der Filterbedingung (3.7) als

$$q_i(\mathbb{P}_i) := \arg\min_{q_i} \|\acute{S}^\top q_i\|_2 \text{ unter } q_i^\top \acute{S} t = 0. \tag{3.13}$$

Wir möchten nicht a priori eine beste Elternknotenmenge für einen Knoten $i \in J$ fixieren, sondern eine ganze Menge $\mathbb{P}_i^+$ sehr guter Elternknotenmengen $\mathbb{P}_i$ bestimmen (Toleranzparameter θ), um später beim Festlegen der C/F Markierungen flexibler zu sein. Als Einschränkung fordern wir

$$|\mathbb{P}_i| \leq 2$$

und

$$\mathbb{P}_i \subseteq \acute{\mathcal{N}}(i), \tag{3.14}$$

d. h. wir verlangen nach [Stü99a] eine *direkte Interpolation*. Obwohl auch 1–elementige Elternknotenmengen zugelassen sind, stellen sie doch die Ausnahme dar und wir sprechen allgemein von einem Elternpaar. Wir wählen also

Definition 3.3.1 (Gute Elternpaare $\mathbb{P}_i^+$)
Die guten Elternpaare bilden die Menge

$$\begin{aligned}\mathbb{P}_i^+ \quad := \quad & \big\{\mathbb{P}^* \subseteq \acute{\mathcal{N}}(i) \big| \; |\mathbb{P}^*| \leq 2 \text{ und} \\ & |a_{ii}| \, \|\acute{S}^\top q_i(\mathbb{P}^*)\|_2 \, \|\overline{\acute{S}}^\top \bar{q}_i(\mathbb{P}^*)\|_2 \leq \delta \text{ und} \qquad (3.15) \\ & \forall \mathbb{P} \subseteq \acute{\mathcal{N}}(i) \text{ mit } |\mathbb{P}| \leq 2 : \\ & \theta \, \|\acute{S}^\top q_i(\mathbb{P}^*)\|_2 \, \|\overline{\acute{S}}^\top \bar{q}_i(\mathbb{P}^*)\|_2 \leq \|\acute{S}^\top q_i(\mathbb{P})\|_2 \, \|\overline{\acute{S}}^\top \bar{q}_i(\mathbb{P})\|_2 \big\}. \qquad (3.16)\end{aligned}$$

(3.15) sichert eine *absolute* Güte der Elternpaare zu. $|a_{ii}| \, \|\acute{S}^\top q_i(\mathbb{P}^*)\|_2 \, \|\overline{\acute{S}}^\top \bar{q}_i(\mathbb{P}^*)\|_2$ ist eine Näherung für den Interpolationsfehler. Im Regelfall führt der Wert von δ zu keiner Einschränkung, aber ein Knoten, der überhaupt nicht gut interpoliert werden kann, soll kein (schlechtes) Elternpaar erhalten, sondern grob markiert werden.

Die von uns gewählten Parameterwerte sind in Abschnitt 6.3 zusammengefaßt.

3.4 Markierungsalgorithmus

Nun stehen für jeden Knoten i eine Menge guter Elternpaare $\mathbb{P}_i^+$ bereit. Die Aufgabe ist jetzt festzulegen, welche Knoten feinmarkiert werden und welches Elternpaar dazu verwendet werden soll. Dabei sollen zwei Ziele erreicht werden:

Z1 Jeder F–Knoten $i \in \mathcal{F}$ wird durch ein *gutes* Elternpaar $\mathbb{P} \in \mathbb{P}_i^+$ nach Definition 3.3.1 interpoliert.

Z2 Die Anzahl der Grobgitterknoten und die Anzahl der (von Null verschiedenen) Matrixeinträge auf dem groben Level sollen minimiert werden.

Während **Z1** strikt eingehalten wird, kann **Z2** als globale Eigenschaft nur durch lokale Heuristiken angenähert werden, um den Aufwand effizient zu halten. Um **Z2** quantitativ zu erfassen, sei $n^e(\mathbb{P})$ die Anzahl der neu entstehenden Matrixeinträge in der Grobgittermatrix, wenn das Elternpaar $\mathbb{P}$ verwendet wird, und $n^c(\mathbb{P})$ die Anzahl neu entstehender Grobgitterknoten. Mit Wichtungsparametern α^e und α^c definieren wir das Gewicht eines Elternpaars

$$w(\mathbb{P}) := \alpha^e n^e(\mathbb{P}) + \alpha^c n^c(\mathbb{P}). \qquad (3.17)$$

Die grundsätzliche Idee des Markierungsalgorithmus ist, das momentan beste Elternpaar $\mathbb{P}_i$, d. h. das mit kleinstem Gewicht $w(\mathbb{P}_i)$, zu verwenden, um den zugehörigen Knoten i als fein und die Knoten $j \in \mathbb{P}_i$ als grob zu markieren. Diese getroffene Auswahl kann nicht mehr revidiert werden. Die Datenstrukturen der Punkte i und j müssen an die neue Situation angepaßt werden: Für j können alle Elternlisten

entfernt werden und die Gewichte der Elternpaare in der Nachbarschaft um i und j müssen aktualisiert werden, weil sich (3.17) in der Umgebung von i und j ändert. Das Vorgehen wird in folgendem Algorithmus zusammengefaßt. Vor dem Aufruf müssen schon alle Gewichte $w(\mathbb{P}_J)$ berechnet sein. $\mathcal{L}$ enthält alle Knoten, die markiert werden sollen; hier gilt also $\mathcal{L} := J$.

Algorithmus 3.1 (Label($\mathcal{L}$))
(1) $\mathcal{F} := \mathcal{C} := \emptyset$
(2) $\mathcal{D} := \mathcal{L}$
(3) solange($\mathbb{P}_\mathcal{L} \neq \emptyset$)
(4) wähle ein $\mathbb{P}_i \in \mathbb{P}_\mathcal{L}$ mit minimalem $w(\mathbb{P}_i)$
(5) EliminateFNode($\mathbb{P}_i$)
(6) $\mathcal{C} := \mathcal{C} \cup \mathcal{D}$

Algorithmus 3.2 (EliminateFNode($\mathbb{P}_i$))
(1) $\mathcal{F} := \mathcal{F} \cup \{i\}$
(2) $\mathcal{D} := \mathcal{D} \setminus \{i\}$
(3) erzeuge $q_i(\mathbb{P}_i)$
(4) $\mathbb{P}_\mathcal{L} := \mathbb{P}_\mathcal{L} \setminus \mathbb{P}_i^+$
(5) $\mathbb{P}_J := \mathbb{P}_J \setminus \{\mathbb{P} \in \mathbb{P}_J \mid i \in \mathbb{P}\}$
(6) $\forall j \in \mathbb{P}_i$: EliminateCNode(j)
(7) aktualisiere alle betroffenen Gewichte $w(\mathbb{P}_J)$

Algorithmus 3.3 (EliminateCNode(i))
(1) $\mathcal{C} := \mathcal{C} \cup \{i\}$
(2) $\mathcal{D} := \mathcal{D} \setminus \{i\}$
(3) erzeuge $P_{ii'} := 1;\ \ R_{i'i} := 1$
(4) aktualisiere alle betroffenen Gewichte $w(\mathbb{P}_J)$

Sollten nach Label noch Knoten unbearbeitet in $\mathcal{D}$ stehen, so ist es nicht mehr möglich, sie gut zu interpolieren und sie müssen folglich zwangsweise grob markiert werden (Zeile (6)). Dadurch werden die restlichen Gewichte $w(\mathbb{P}_J)$ inkonsistent; da sie aber nicht mehr benötigt werden, stört das nicht.

Der Aufbau der Hierarchie von Matrizen wird in folgendem Algorithmus zusammengefaßt:

Algorithmus 3.4 (FAMGConstruct(^0A))
(1) for $\ell := 0$ step -1
(2) if(Abbruchkriterium) break
(3) $\forall i \in {}^\ell J$ Calculate($\mathbb{P}_i^+$)
(4) Label($^\ell J$)
(5) ${}^{\ell-1}A := {}^\ell R\, {}^\ell A\, {}^\ell P$

3.5 Aufwandsabschätzung

Wir wollen hier nur das Ergebnis aus [Wag00a] nennen. Die zwei wesentlichen Algorithmusteile sind das Bestimmen der guten Elternpaare und der Markierungsalgorithmus Label.

Es sei

$$n := |J|$$

und

$$m_s := \frac{1}{n} \sum_{i \in J} |\acute{\mathcal{N}}(i)|$$

die mittlere Anzahl der nach (3.9) stark gekoppelten Nachbarknoten. Dann erhält man für die Bestimmung der guten Elternpaare $\mathbb{P}_J^+$ einen Aufwand von $\mathcal{O}(n \cdot m_s^\beta)$ mit $2 \leq \beta \leq 3$. β hängt entscheidend vom Matrixgraphen ab und kann a priori, vor allem für die Grobgittermatrizen, nicht genauer bestimmt werden. Die Größe des tatsächlichen Rechenzeitaufwands relativiert sich aber, da bei Verwendung spezieller Datenstrukturen die meisten Operationen innerhalb des Caches ausführbar sind und somit eine um Größenordnungen niedrigere Konstante anzusetzen ist als bei den Operationen im Mehrgitterzyklus.

Sei m die durchschnittliche Zahl *aller* Nachbarknoten, ohne Berücksichtigung der Stärke der Kopplung. Dann beträgt der Zeitaufwand des Markierungsalgorithmus Label $\mathcal{O}(nm^2)$. Allerdings ist dieses Ergebnis nur unter Verwendung sehr spezieller Listenstrukturen möglich, die viele Suchvorgänge vermeiden helfen.

Die Laufzeit beider Algorithmusteile verhält sich also zunächst direkt proportional zur Anzahl der Knoten. Wenn die Größe der Matrixsterne m bzw. m_s nicht übermäßig wächst, liegt ein Verfahren optimaler Komplexität vor und ist daher auch für extrem große Knotenanzahlen gut geeignet.

3.6 FAMG Löser

Auf der erstellten Hierarchie von Matrizen setzt man für das eigentliche Lösen einen klassischen Mehrgitteralgorithmus ein, wie er in Algorithmus 2.3 eingeführt wurde. Der hier eingesetzte Glätter sollte mit dem Glätter in der Elternpaarauswahl (3.10) und (3.11) abgestimmt sein. In der Praxis hat sich die Kombination eines normalen Jacobischritts und eines nur auf den feinen Knoten arbeitenden Jacobischritts JacobiFine nach Algorithmus 3.5 am besten bewährt; D ist dabei wieder die Diagonalmatrix $D := \mathrm{diag}(A)$.

Algorithmus 3.5 (JacobiFine($^\ell d$))

(1) $t_i := \begin{cases} 0 & \text{wenn } i \in \mathcal{C}, \\ D_{ii}^{-1} d_i & \text{wenn } i \in \mathcal{F}. \end{cases}$

Für unsere Zwecke bietet es sich aber an, als Glätter im Mehrgitteralgorithmus nur den normalen Jacobischritt zu verwenden und den speziellen Feingitterjacobischritt konzeptionell als Teil der Restriktion bzw. Prolongation zu betrachten. Damit kann dann die Parallelisierung des Lösers allein durch Erweiterung der Restriktion und Prolongation erfolgen; alles weitere sind Standardkomponenten eines parallelen Mehrgitterlösers. Zu bemerken ist noch, daß in der Prolongation die letzten beiden Schritte schon in der Standardformulierung des Mehrgitterzyklus enthalten sind und hier nicht programmiert werden müssen.

Algorithmus 3.6 (FAMGRestriction(${}^{\ell}d$, ${}^{\ell-1}d$, ${}^{\ell}c$))

(1) ${}^{\ell}t :=$ JacobiFine(${}^{\ell}d$)
(2) ${}^{\ell}d := {}^{\ell}d - {}^{\ell}A\,{}^{\ell}t$
(3) ${}^{\ell}c := {}^{\ell}c + {}^{\ell}t$
(4) ${}^{\ell-1}d := {}^{\ell}R\,{}^{\ell}d$

Algorithmus 3.7 (FAMGProlongation(${}^{\ell-1}c$, ${}^{\ell}t$, ${}^{\ell}d$, ${}^{\ell}c$))

(1) ${}^{\ell}t := {}^{\ell}P\,{}^{\ell-1}c$
(2) ${}^{\ell}d := {}^{\ell}d - {}^{\ell}A\,{}^{\ell}t$
(3) ${}^{\ell}c := {}^{\ell}c + {}^{\ell}t$
(4) ${}^{\ell}t :=$ JacobiFine(${}^{\ell}d$)
MG(8) ${}^{\ell}d := {}^{\ell}d - {}^{\ell}A\,{}^{\ell}t$
MG(9) ${}^{\ell}c := {}^{\ell}c + {}^{\ell}t$

Algorithmus 3.8 (FAMG(c, d, ℓ))

(1) if($\ell = \ell_{min}$)
(2) BaseSolver(${}^{\ell}c, {}^{\ell}d$)
else
(3) for $i := 1$ to ν_1: Smooth(${}^{\ell}c, {}^{\ell}d$)
(4) FAMGRestriction(${}^{\ell}d$, ${}^{\ell-1}d$, ${}^{\ell}c$)
(5) ${}^{\ell-1}c := 0$
(6) for $i := 1$ to γ: FAMG($c, d, \ell - 1$)
(7) FAMGProlongation(${}^{\ell-1}c$, ${}^{\ell}t$, ${}^{\ell}d$, ${}^{\ell}c$)
(8) ${}^{\ell}d := {}^{\ell}d - {}^{\ell}A\,{}^{\ell}t$
(9) ${}^{\ell}c := {}^{\ell}c + {}^{\ell}t$
(10) for $i := 1$ to ν_2: Smooth(${}^{\ell}c, {}^{\ell}d$)

4 Parallele Mehrgitterverfahren

In den Kapiteln 2 und 3 wurden die grundlegenden Ideen und Algorithmen für das Lösen von linearen Gleichungssystemen vorgestellt; sie waren noch seriell. Hauptsächlich werden dabei lineare Algebra Operationen ausgeführt. Für die Darstellung der parallelen Algorithmen muß daher zunächst ein Modell einer parallelen linearen Algebra eingeführt werden, das erstmalig auch bewiesen wird. Dieses Modell wird dann dazu dienen, die Parallelisierung des in Kapitel 2 vorgestellten geometrischen Mehrgitterverfahrens zu beschreiben und in Kapitel 5 die Parallelisierung des FAMG zu entwickeln.

4.1 Parallele lineare Algebra

Das folgende Modell einer parallelen linearen Algebra lehnt sich sehr stark an das von Ch. Wieners entwickelte an [Wie00a] [Wie00b], das wiederum auf dem Modell von P. Bastian aufbaut und dieses stark erweitert [Bas96b]. G. Haase führt im Kontext von Gebietszerlegungsmethoden eine sehr ähnliche parallele lineare Algebra ein [Haa99a] [Haa99b]. Eine wesentliche Änderung zu Wieners Modell wird das Herauslassen des Begriffs „aktive Teilmenge“ einer Indexmenge sein, den man nur für die Formulierung *lokaler* Mehrgitterverfahren benötigt, die uns hier aber nicht beschäftigen. In Bastians Modell entspricht diesem der Begriff der *Vektorklasse*. Einige Verfeinerungen, Verbesserungen und zusätzliche Funktionalitäten werden für das parallele FAMG eingeführt. Insbesondere wird erstmals durch eine verallgemeinerte Darstellung der Restriktions– und Prolongationsoperation klar, was die entscheidenden Voraussetzungen sind.

Das Modell wird sich nur auf die Begriffe Index (oder Knoten) und den Matrixgraph stützen. Es ist also rein algebraisch ohne weitere geometrische Begriffe zu verwenden. Trotzdem wird so oft wie möglich die Anwendung dieses Modells auch auf geometrische Situationen besprochen.

Zunächst muß die Verteilung der Daten auf die Prozessoren und die dann möglichen parallelen Darstellungsformen definiert werden. Transformationen der Darstellungsform schaffen die Voraussetzung für die Anwendung der vorgestellten parallelen lineare Algebra Routinen.

4.1.1 Datenaufteilung

Definition 4.1.1 (Prozessor)
Die Menge der Prozessoren, auf die die parallele Anwendung verteilt wird, sei mit $\mathcal{P}$ bezeichnet. Ein Element $p \in \mathcal{P}$ heißt **Prozessor** oder englisch **processing element** (kurz **PE**).

Bemerkung 4.1.2
Einem realen Parallelrechner liegt immer eine Prozessortopologie zugrunde, die einer Struktur der Menge $\mathcal{P}$ entspricht. Da heute die Kommunikationsleistung zwischen 2 Prozessoren aufgrund leistungsfähiger Routingverfahren (*wormhole routing*) kaum noch von ihrer Position bzgl. dieser Topologie abhängt [Bas00], wollen wir auf $\mathcal{P}$ keine Struktur festlegen und ausnutzen. Daß wir trotzdem der Einfachheit halber die Prozessoren mit natürlichen Zahlen bezeichnen werden, soll dazu kein Widerspruch sein.

Definition 4.1.3 (Datenaufteilung)
Gegeben sei eine **Indexmenge** J. Ein Index $j \in J$ wird **Knoten** genannt und ihm wird eindeutig ein Prozessor zugeordnet:

$$\mu : \left\{ \begin{array}{lcl} J & \to & \mathcal{P} \\ j & \mapsto & p, \end{array} \right. \tag{4.1}$$

jeder solche Knoten heißt ein **Master** auf p, die Menge aller Masterknoten auf einem Prozessor heiße

$$\mathcal{M}^p := \{\, j \in J \mid \mu(j) = p \,\}. \tag{4.2}$$

Aus Konsistenz– und Effizienzgründen wird es nötig sein, neben den Masterknoten noch einige Knotenkopien auf jedem Prozessor zu halten, die sogenannten **Border**knoten. Die zugehörige Aufteilungsabbildung α ordnet jedem Knoten alle Prozessoren zu, auf denen der Knoten Kopien (auch **Instanzen** genannt) besitzt:

$$\alpha : \left\{ \begin{array}{lcl} J & \to & \mathfrak{P}(\mathcal{P}) \text{ (Potenzmenge von } \mathcal{P}) \\ j & \mapsto & \{\, p \in \mathcal{P} \mid j \text{ hat eine Instanz auf } p \,\}, \end{array} \right. \tag{4.3}$$

die Menge aller Borderknoten auf einem Prozessor heiße

$$\mathcal{B}^p := \{\, j \in J \mid p \in \alpha(j) \,\} \setminus \mathcal{M}^p. \tag{4.4}$$

Weiterhin induziert die Aufteilungsabbildung α eine nicht–disjunkte **Partitionierung** der Indexmenge J:

$$J^p := \{\, j \in J \mid p \in \alpha(j) \,\}. \tag{4.5}$$

Bemerkung 4.1.4
$J = \bigcup_{p \in \mathcal{P}} J^p$ ist eine globale Größe, die durch Einsammeln der verteilt vorliegenden Teilmengen J^p zu erhalten ist. Da dieser Vorgang eine Kommunikation erfordert, sollte J nie verwendet werden müssen, um gut skalierbare Algorithmen zu erhalten. Für alle unsere Algorithmen wird die Kenntnis der Abbildungen α und μ ausreichen, die rein lokal sind. Diese Abbildungen müssen aber sehr schnell ausführbar sein. Das hier verwendete Modul DDD stellt diese Funktionalität bereit [Bir97] [Bir98].

Bemerkung 4.1.5 (Ghostkopien)
Man kann die Menge der Borderknoten noch weiter unterteilen: je nachdem, ob sie nur zu Kommunikationszwecken dienen (sogenannte Ghostkopien [Bas96b]) oder auch für Berechnungen benötigt werden. Da diese Unterscheidung vor allem für lokale Mehrgitterverfahren und gute Lastverteilungen eine Rolle spielt, braucht sie hier nicht getroffen werden.

Bemerkung 4.1.6 (Aufteilungsabbildung fix)
Im folgenden sei immer eine Aufteilungsabbildung fest gegeben, d. h. alle Objekte in einem Kontext sind immer durch *dieselbe* Abbildung aufgeteilt.

Später wird die Eindeutigkeit der Masterkomponente explizit benötigt; sie soll daher in folgender Form festgehalten werden:

Lemma 4.1.7 (Eindeutigkeit der Masterabbildung)

$$\forall i \in J \;\; \exists_1 p \in \alpha(i) : \;\; i \in \mathcal{M}^p. \tag{4.6}$$

Beweis: Die Aussage folgt direkt aus Definition 4.1.3. □

4.1.2 Verteilte Vektoren und Matrizen

Definition 4.1.8 (Freiheitsgrade)
Jedem Knoten $j \in J$ werden n_j Werte zugeordnet, sogenannte **Freiheitsgrade**.

Definition 4.1.9 (Verteilter Vektor)
Sei $n^p := \sum_{j \in J^p} n_j$, $n := \sum_{j \in J} n_j$ und $x \in \mathbb{R}^n$. $x^p = (x_j^p)_{j \in J} \in \mathbb{R}^n$ ist der auf Prozessor p *lokal* vorhandene Teil des globalen Vektors x, d. h. x^p hat nur n^p Komponenten, die von 0 verschieden sein können ($x_j^p \equiv 0 \;\forall j \notin J^p$). Jedes $x_j^p \in \mathbb{R}^{n_j}$ heißt eine **Komponente** des Vektors. Die Begriffe Master und Border übertragen sich von den Knoten auf die an sie gebundenen Vektorkomponenten: x_j^p heißt **Masterkomponente**, wenn $p = \mu(j)$, und **Borderkomponente**, wenn $p \in \alpha(j) \setminus \{\mu(j)\}$.

Bemerkung 4.1.10 (Blockstruktur)
Definition 4.1.8 prägt jedem Vektor eine Blockstruktur auf. Z. B. würde eine Diskretisierung der Navier–Stokes Gleichung, die auf den Elementecken 2 Geschwindigkeits-

und 1 Druckfreiheitsgrad verwendet, zu einem konstanten $n_j \equiv 3$ führen; aber auch von Knoten zu Knoten wechselnde Anzahlen sind möglich, wie sie z. B. für obige Gleichung entsteht, wenn man an den Elementkanten 2 Geschwindigkeitsfreiheitsgrade und in der Elementmitte einen Druckfreiheitsgrad ansetzt [Ren96]. Eine solche Information muß natürlich zunächst aus geometrischen Informationen abgeleitet werden. Sobald aber alle Zahlen n_j bestimmt sind, braucht man im weiteren keinerlei Geometrieinformation mehr für die lineare Algebra. Für skalare Differentialgleichungen gilt $n_j \equiv 1$, d. h. ein Block besteht aus genau einer reellen Zahl.
In Zukunft betrachten wir diese Blöcke als elementare Einheiten, mit denen wir Berechnungen wie mit reellen Zahlen formulieren werden.

Bemerkung 4.1.11 (Geometrische Interpretation für Systeme)
$n_j \equiv 1$ für skalare Differentialgleichungen läßt einem Knoten in natürlicher Weise einen klassischen Freiheitsgrad entsprechen. Im bisher vorgestellten Modell gibt es jedoch 2 Arten, ein System von Differentialgleichungen zu behandeln: Man kann einem Knoten alle Information in einer geometrischen Position entsprechen lassen, wie das auch in der vorigen Bemerkung praktiziert wurde. Das führt dann auf echte Blöcke. Andererseits kann man auch jedem einzelnen Freiheitsgrad einen eigenen Knoten zuordnen, also auch für Systeme $n_j \equiv 1$ verwenden. Dabei hat man allerdings Struktur und damit Handlungsspielraum verloren aber gleichfalls Generalität gewonnen. Beide Anwendungen sind sinnvoll und für AMG Verfahren ist es bei weitem noch nicht abschließend geklärt, welcher Weg zu bevorzugen ist; wahrscheinlich wird es von der konkreten Anwendung abhängen [Stü99a] [Stü99b].

Auf den verteilten Vektoren aufbauend, konstruiert man eine verteilte Matrix.

Definition 4.1.12 (Verteilte Matrix)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n \times n'}$. Der auf einem Prozessor lokal darstellbare Anteil $A^p \in \mathbb{R}^{n \times n'}$ hat ein auf $J^p \times J'^p$ beschränktes Besetzungsmuster.

Bemerkung 4.1.13 (Blockstruktur einer Matrix)
Zu beachten ist, daß $A_{ij}, i \in J, j \in J'$ selbst eine Matrix der Dimension $n_i \times n'_j$ ist.

Beispiel 4.1.14
Für eine Steifigkeitsmatrix ist $J = J'$, für eine Interpolationsmatrix ist J die Indexmenge auf dem feineren und J' auf dem gröberen Level.

Beispiel 4.1.15
Gegeben sei ein System von 2 Differentialgleichungen auf einem 1–dimensionalen Gebiet, das mit 3 linearen Elementen diskretisiert sei. Das führt auf 4 Knoten $J = \{ i, j, k, l \}$, die je 2 Freiheitsgrade tragen. Die Diskretisierung der partiellen Differentialgleichung mit linearen Finiten Elementen führt auf eine 4×4 Blockmatrix, die ein tridiagonales Besetzungsmuster besitzt; jeder Block ist eine 2×2 Matrix. Die Datenaufteilung erfolge auf 2 Prozessoren $\mathcal{P} = \{p, q\}$ über $\mu(i) = \mu(j) = p$, $\mu(k) = \mu(l) = q$,

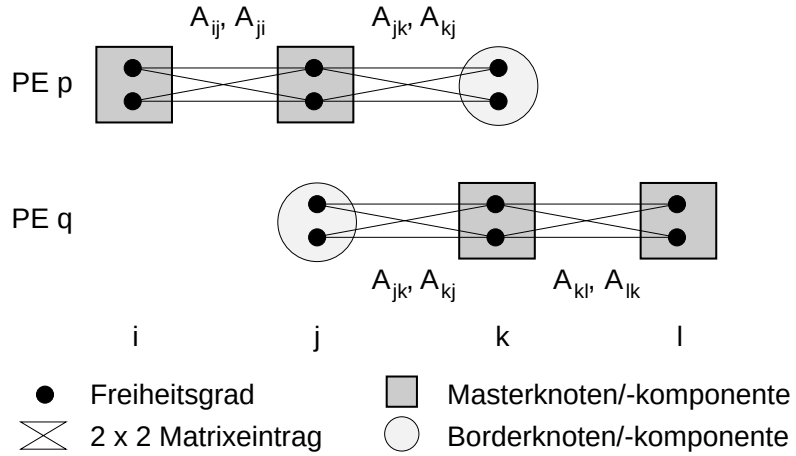


Abbildung 4.1: Beispiel einer Datenaufteilung für ein System von 2 partiellen Differentialgleichungen in 1 Raumdimension, diskretisiert mit linearen Finiten Elementen und verteilt auf 2 Prozessoren

$\alpha(i) = \{p\}$, $\alpha(j) = \alpha(k) = \{p, q\}$, $\alpha(l) = \{q\}$. Damit gilt $J^p = \{i, j, k\}$, $J^q = \{j, k, l\}$, $\mathcal{M}^p = \{i, j\}$, $\mathcal{M}^q = \{k, l\}$, $\mathcal{B}^p = \{k\}$, $\mathcal{B}^q = \{j\}$. Die gesamte Situation ist in Abbildung 4.1 dargestellt.

4.1.3 Parallele Darstellungsformen von Vektoren und Matrizen

Die Definition 4.1.9 des verteilten Vektors läßt vermuten, daß für jeden Knoten, der auf mehreren Prozessoren vorkommt, die jeweilige Vektorkomponente den gleichen Wert haben muß. Es ist jedoch auch eine andere Werteverteilung möglich und sogar nötig:

Definition 4.1.16 (Parallele Darstellungsformen für Vektoren)
Ein Vektor $x \in \mathbb{R}^n$ ist in **konsistenter** Form $\bar{x} = (\bar{x}^p)_{p \in \mathcal{P}}$ verteilt, wenn

$$\forall j \in J \ \forall p \in \alpha(j) : x_j = \bar{x}_j^p. \tag{4.7}$$

Ein Vektor $x \in \mathbb{R}^n$ ist in **additiver** (oder **inkonsistenter**) Form $\hat{x} = (\hat{x}^p)_{p \in \mathcal{P}}$ verteilt, wenn

$$\forall j \in J : x_j = \sum_{p \in \alpha(j)} \hat{x}_j^p. \tag{4.8}$$

Eine besondere Form der additiven Darstellung sei noch hervorgehoben: ein Vektor $x \in \mathbb{R}^n$ ist in **eindeutiger** Form $\check{x} = (\check{x}^p)_{p \in \mathcal{P}}$ verteilt, wenn die Masterkomponente den gesamten Wert der Komponente trägt:

$$\forall j \in J \ \forall p \in \alpha(j) : \check{x}_j^p = \begin{cases} x_j & \text{für } p = \mu(j) \\ 0 & \text{für } p \in \alpha(j) \setminus \{\mu(j)\}. \end{cases} \tag{4.9}$$

Bemerkung 4.1.17 (Eindeutigkeit der Verteilung eines Vektors)
Nur die konsistente und die eindeutige Verteilung sind bei gegebener Aufteilungsabbildung α eindeutig. Eine allgemeine additive Verteilung ist nicht eindeutig.

Bemerkung 4.1.18 (Eindeutigkeit der parallelen Darstellungsform)
Die Begriffe konsistent und inkonsistent (additiv) schließen sich *nicht* aus. So ist jeder Vektor, der auf allen Komponenten, die mehr als 1 Instanz haben, 0 ist, sowohl konsistent als auch inkonsistent (additiv) verteilt:

$$\forall x \in \mathbb{R}^n : \forall j \in \{\, j \in J \,\big|\, |\alpha(j)| > 1 \,\} : x_j = 0 \implies \bar{x} = \hat{x} = x.$$

Die Situation für Matrizen ist wesentlich schwieriger, da es entscheidend auch auf das Besetzungsmuster der lokalen Matrizen ankommen wird. Zunächst werden 3 Hilfsbegriffe eingeführt.

Definition 4.1.19 (Nachbarschaft eines Knotens)
Die Nachbarschaft eines Knotens $i \in J$ (auf einem Prozessor $p \in \mathcal{P}$) ist definiert als die Menge der Knoten $j \in J'$ (bzw. $j \in J'^p$), die durch die Matrix $A \in \mathbb{R}^{n \times n'}$ mit i direkt gekoppelt sind:

$$\mathcal{N}(i) := \{\, j \in J' \mid A_{ij} \neq 0 \;\vee\; A_{ji} \neq 0 \,\} \tag{4.10}$$

$$(\text{bzw. } \mathcal{N}^p(i) := \{\, j \in J'^p \mid A^p_{ij} \neq 0 \;\vee\; A^p_{ji} \neq 0 \,\}).$$

Wenn die Bezugsmatrix aus dem Kontext nicht klar ist, kann sie explizit genannt werden:

$$\mathcal{N}_A(i).$$

Lemma 4.1.20 (Relation $\mathcal{N}$ ist symmetrisch)

$$\forall i \in J \;\forall j \in J' : \quad i \in \mathcal{N}(j) \Leftrightarrow j \in \mathcal{N}(i). \tag{4.11}$$

Beweis: folgt unmittelbar aus Definition (4.1.19) □

Definition 4.1.21 (Matrixgraph)
Der zur Matrix $A \in \mathbb{R}^{n \times n'}$ gehörige **Matrixgraph** $\mathcal{G}(A) := (\mathcal{V}, \mathcal{E})$ ist definiert als:

$$\begin{aligned} \mathcal{V} &= J \cup J' \\ \mathcal{E} &= \{\, (i,j) \in J \times J' \mid A_{ij} \neq 0 \,\}. \end{aligned}$$

Zur Einordnung der Eigenschaften des eben definierten Graphen sollen die Bezeichnungen nach [Jun94] verwendet werden.

Bemerkung 4.1.22 (Einordnung in graphentheoretische Begriffe)
Wenn $J = J'$, ist der Matrixgraph als *gerichteter Graph* oder als *ungerichteter Multigraph* aufzufassen. Oft ist es vorteilhaft, den *dazugehörigen ungerichteten Graphen* zu betrachten, der durch Zusammenfassen zweier gerichteter Kanten (i, j) und (j, i) zu einer ungerichteten Kante (oder *Bogen*) $\{i, j\}$ entsteht.
Wenn $J \cap J' = \emptyset$, ist der Graph *bipartit* (oder *paar*).

Definition 4.1.23 (Aufteilungsabbildung α respektiert Graph $\mathcal{G}$)
Die Aufteilungsabbildungen α und α' nach Definition 4.1.3 **respektieren** einen Matrixgraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, wenn

$$\forall (i, j) \in \mathcal{E} : \alpha(i) \cap \alpha'(j) \neq \emptyset.$$

Bemerkung 4.1.24
Wenn Aufteilungsabbildungen α, α' einen Matrixgraphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ respektieren, kann jede Kante $e \in \mathcal{E}$ auf mindestens einem Prozessor dargestellt werden, weil *beide* Knoten zu dieser Kante auf diesem Prozessor vorhanden sind. Daß eine Aufteilungsabbildung diese Eigenschaft besitzt, ist nicht a priori gesichert und muß z. B. für den Graph der Steifigkeitsmatrix $\mathcal{G}(A)$ extra nachgewiesen werden; siehe Abschnitt 4.2.5.

Definition 4.1.25 (Parallele Darstellungsformen für Matrizen)
Eine Matrix $A \in \mathbb{R}^{n \times n'}$ ist in **konsistenter** Form $\bar{A} = \left(\bar{A}^p\right)_{p \in \mathcal{P}}$ verteilt, wenn

$$\text{die Aufteilungsabbildungen } \alpha, \alpha' \text{ den Matrixgraph } \mathcal{G}(\mathrm{A}) \text{ respektieren,} \tag{4.12a}$$
$$\text{für } \alpha, \alpha' \text{ gilt } \forall i \in J \ \forall j \in \mathcal{N}_A(i) : \quad \alpha(i) \subseteq \alpha'(j) \text{ und} \tag{4.12b}$$
$$\bar{A}^p_{ij} = A_{ij} \quad \forall i \in J \ \forall j \in \mathcal{N}_A(i) \ \forall p \in \alpha(i). \tag{4.12c}$$

Eine Matrix $A \in \mathbb{R}^{n \times n'}$ ist in **teilweise konsistenter** Form $\tilde{A} = (\tilde{A}^p)_{p \in \mathcal{P}}$ verteilt, wenn

$$\text{die Aufteilungsabbildungen } \alpha, \alpha' \text{ den Matrixgraph } \mathcal{G}(\mathrm{A}) \text{ respektieren,} \tag{4.13a}$$
$$\text{für } \alpha, \alpha' \text{ gilt } \forall i \in J \ \forall j \in \mathcal{N}_A(i) : \quad \mu'(j) \in \alpha(i) \text{ und} \tag{4.13b}$$
$$\tilde{A}^p_{ij} = A_{ij} \quad \forall i \in J \ \forall j \in \mathcal{N}_A(i) \ \forall p \in \alpha(i) \cap \alpha'(j). \tag{4.13c}$$

Eine Matrix $A \in \mathbb{R}^{n \times n'}$ ist in **additiver** (oder **inkonsistenter**) Form $\hat{A} = (\hat{A}^p)_{p \in \mathcal{P}}$ verteilt, wenn

$$\text{die Aufteilungsabbildung } \alpha, \alpha' \text{ den Matrixgraph } \mathcal{G}(\mathrm{A}) \text{ respektieren und} \tag{4.14a}$$
$$\sum_{p \in \alpha(i) \cap \alpha'(j)} \hat{A}^p_{ij} = A_{ij} \quad \forall i \in J \ \forall j \in \mathcal{N}_A(i). \tag{4.14b}$$

Bemerkung 4.1.26 (Aufteilungsabbildung respektiert Matrixgraph)
Die Forderung, daß die Aufteilungsabbildung den Matrixgraph respektieren muß, sichert zu, daß jeder Eintrag der globalen Matrix mindestens auf einem Prozessor auch tatsächlich vorhanden ist.

Lemma 4.1.27 (Keine konsistente Verteilung bei zushgd. Graph)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$ mit zusammenhängendem Matrixgraph $\mathcal{G}(A)$. Wenn die Matrix echt verteilt ist ($\exists p \in \mathcal{P} : J^p \neq J$), gibt es für jede lokale Matrix A^p einen konsistenten Vektor $\bar{x} \in \mathbb{R}^n$, sodaß $\exists j \in J^p : (A^p \bar{x}^p)_j \neq (Ax)_j$, d. h. eine Matrix mit zusammenhängendem Graph kann nicht konsistent verteilt werden.

Beweis: Zum Beweis nimmt man an, daß es eine konsistente Verteilung gäbe und führt dies zum Widerspruch. Für die Details siehe [Wie00b]. □

Bemerkung 4.1.28 (Konsistent verteilte Matrix)
Für jeden Index $j \in J^p$ muß also die gesamte Matrixzeile (aufgrund von Bedingung (4.12b)) mit den jeweiligen globalen Einträgen (Bedingung (4.12c)) auf dem Prozessor vorhanden sein. Da die Steifigkeitsmatrix für zusammenhängende Gebiete üblicherweise selbst zusammenhängend ist, kann man sie also nicht konsistent verteilen. Die teilweise konsistente Verteilung wird sich aber für unsere Zwecke als ausreichend erweisen. Die globale Eigenschaft, die sich im Zusammenhang des Matrixgraphen ausdrückt, kann also in der verteilten Datenstruktur nicht adäquat repräsentiert werden. Die vollständige globale Information muß — soweit erforderlich — nach der Matrix–Vektor–Multiplikation, die ein additives Ergebnis liefert (siehe Abschnitt 4.1.5.5), in einem gesonderten Kommunikationsschritt hergestellt werden, wie im nächsten Abschnitt beschrieben. Konsistent verteilt sein kann dagegen eine Diagonalmatrix (sie ist nicht zusammenhängend und damit gibt es keine Schwierigkeit mit Bedingung (4.12b)) oder eine Prolongationsmatrix, da sie eine gerichtete Verbindung von einem Level zu einem anderen darstellt und dadurch ebenfalls Bedingung (4.12b) eingehalten werden kann.

Bemerkung 4.1.29 (Teilweise konsistent verteilte Matrix)
Wenn in $\tilde{A}^p$ ein Matrixeintrag vorhanden sein kann (weil $i \in J^p$ und $j \in J'^p$), muß er den globalen Wert A_{ij} besitzen. Die im Vergleich zu [Wie00a] zusätzliche Bedingung (4.13b) an die Aufteilungsabbildung wird wichtig sein, um die Multiplikation $\tilde{A}\ \check{x}$ in Abschnitt 4.1.5.5 formulieren zu können. Diese Bedingung ist aber einfach einzuhalten, indem zusätzliche Knotenkopien angelegt werden. In der derzeitigen Implementierung von *UG* verzichtet man für geometrische Mehrgitterverfahren auf diese Bedingung und nimmt den dadurch in (4.24) entstehenden Fehler in Kauf.

Lemma 4.1.30 (diag($\tilde{A}$) ist konsistent)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$. Dann ist die Diagonalmatrix der teilweise konsistenten Matrix $\tilde{A}$ konsistent.

Beweis: (4.12) folgt aus (4.13) sofort, wenn man verwendet, daß eine Diagonalmatrix nur auf der Diagonalen Einträge hat, d. h. $\mathcal{N}_{\mathrm{diag}(A)}(i) = \{i\}\ \forall i \in J$. □

Bemerkung 4.1.31 (Additiv verteilte Matrix)
Aufgrund des Besetzungsmusters jeder verteilt gespeicherten Matrix nach Definition 4.1.12 kann man die Forderung an eine additiv verteilte Matrix auch in der

Kurzform angeben:

$$\sum_{p\in\mathcal{P}} \hat{A}^p = A.$$

Bemerkung 4.1.32 (Eindeutigkeit der Verteilung einer Matrix)
Auch bei Matrizen ist die additive Verteilung selbst bei fester Aufteilungsabbildung α nicht eindeutig. Die (teilweise) konsistente Form ist dagegen eindeutig.

4.1.4 Transformation der parallelen Darstellungsform

Die linearen Algebra Operationen im nächsten Abschnitt werden gewisse Bedingungen an die parallele Darstellungsform ihrer Operanden stellen. Erfüllt ein Operand nicht die Anforderung, so muß seine Verteilungsform angepaßt werden. Ein solcher Transformationsschritt wird in der Regel mit Kommunikation verbunden sein und das bedeutet sowohl zusätzliche Operationen, die Zeit kosten, als auch Wartezeiten bis zum Empfang der Nachrichten. Daher wird man bestrebt sein, die Darstellungsform solange wie möglich unverändert zu lassen und eine Transformation nur auf Anforderung hin (on–demand) auszuführen. Für die Defektberechnung z. B. werden wir sehen, daß sich die Anforderungen in idealer Weise ergänzen und keine Transformation nötig ist (siehe Abschnitt 4.2.6).

4.1.4.1 additiver $\mapsto$ konsistenter Vektor

Die Aufgabe ist,

$$\bar{x}_j^p := \sum_{q\in\alpha(j)} \hat{x}_j^q \quad \forall j \in J \ \forall p \in \alpha(j)$$

zu berechnen. Folgender Algorithmus führt das Einsammeln und Summieren der einzelnen Anteile aus (nach [Bas96b, Funktion VecCons]):

Algorithmus 4.1 (MakeAdditiveVectorConsistent($\hat{x}$))
$Q^p := \{\, q \in \mathcal{P} \setminus \{p\} \mid J^q \cap J^p \neq \emptyset \,\}$
(1) Gather: $\forall q \in Q^p \ \forall j \in J^p \cap J^q$: kopiere $\hat{x}_j^p$ in Nachrichtenpuffer $M_{p,q}$
(2) Datenaustausch: $\forall q \in Q^p$: sende $M_{p,q}$ an q und empfange $M_{q,p}$ von q
(3) Scatter: $\forall q \in Q^p \ \forall j \in J^p \cap J^q$: hole $\hat{x}_j^q$ aus $M_{q,p}$ und summiere

$$\hat{x}_j^p := \hat{x}_j^p + \hat{x}_j^q$$

Am Ende der Operation steht in $\hat{x}^p$ der konsistent verteilte Vektor und muß folglich als $\bar{x}^p$ bezeichnet werden.

Beweis: Nach Definition 4.1.16 ist $\hat{x}$ eine additive Verteilung des Vektors x mit den Komponenten $x_j = \sum_{p \in \mathcal{P}} \hat{x}_j^p$. In Schritt (1) bzw. (3) werden jeweils alle Komponenten $\hat{x}_j^q$ mit $(q,j) \in \{\, (q,j) \in \mathcal{P} \setminus \{p\} \times J^p \mid q \in \alpha(j) \,\}$ bearbeitet. Daher sind im Resultat

$$\hat{x}_j^p + \sum_{q \in \alpha(j) \setminus \{p\}} \hat{x}_j^q = \sum_{q \in \alpha(j)} \hat{x}_j^q$$

genau die Summanden enthalten, die die Definition 4.1.16 für die konsistente Darstellungsform verlangt. □

4.1.4.2 additiver $\mapsto$ eindeutiger Vektor

Die Aufgabe ist,

$$\check{x}_j^p := \begin{cases} \sum_{q \in \alpha(j)} \check{x}_j^q & \text{falls } p = \mu(j) \\ 0 & \text{sonst} \end{cases}, \quad \forall j \in J \ \forall p \in \alpha(j)$$

zu berechnen. Folgender Algorithmus führt das Einsammeln und Summieren der einzelnen Anteile aus:

Algorithmus 4.2 (MakeAdditiveVectorUnique($\hat{x}$))

$Q^p := \{\, q \in \mathcal{P} \setminus \{p\} \mid \mathcal{B}^p \cap \mathcal{M}^q \neq \emptyset \,\}$
$Q'^p := \{\, q \in \mathcal{P} \setminus \{p\} \mid \mathcal{M}^p \cap \mathcal{B}^q \neq \emptyset \,\}$

(1) Gather: $\forall q \in Q \ \forall j \in \mathcal{B}^p \cap \mathcal{M}^q$: kopiere $\hat{x}_j^p$ in Nachrichtenpuffer $M_{p,q}$ und lösche $\hat{x}_j^p$

(2) Datenaustausch: $\forall q \in Q^p$ sende $M_{p,q}$ an q und $\forall q \in Q'^p$ empfange $M_{q,p}$ von q

(3) Scatter: $\forall q \in Q'^p \ \forall j \in \mathcal{M}^p \cap \mathcal{B}^q$: hole $\hat{x}_j^q$ aus $M_{q,p}$ und summiere

$$\hat{x}_j^p := \hat{x}_j^p + \hat{x}_j^q$$

Am Ende der Operation steht in $\hat{x}^p$ der eindeutig verteilte Vektor und muß folglich als $\check{x}^p$ bezeichnet werden.

Beweis: Nach Definition 4.1.16 ist $\hat{x}$ eine additive Verteilung des Vektors x mit den Komponenten $x_j = \sum_{p \in \mathcal{P}} \hat{x}_j^p$. In Schritt (1) bzw. (3) werden auf Prozessor $p \in \mathcal{P}$ jeweils alle Komponenten $\hat{x}_j^q$ mit $(p,q,j) \in \{\, (p,q,j) \in \mathcal{P} \times \mathcal{P} \times J \mid q \neq p \ \wedge \ j \in \mathcal{B}^p \cap \mathcal{M}^q \,\}$ bloß mit vertauschter Bedeutung von p und q (was dem der Vertauschung „p schickt für q und q empfängt von p“ entspricht) bearbeitet. Daher sind für $j \in \mathcal{M}^p$ im Resultat

$$\hat{x}_j^p + \sum_{q \in \mathcal{P} \mid j \in \mathcal{B}^q} \hat{x}_j^q \overset{(4.4)}{=} \sum_{q \in \alpha(j)} \hat{x}_j^q = x_j$$

in den Masterkomponenten genau die Summanden enthalten, die die Definition 4.1.16 für die eindeutige Darstellungsform verlangt, und auch alle anderen Komponenten erfüllen $\hat{x}_j^p = 0 \ \forall j \in \mathcal{B}^p$, da sie in Schritt (1) auf 0 gesetzt wurden. □

4.1.4.3 additive $\mapsto$ teilweise konsistente Matrix

Die Aufgabe ist, die Aufteilungsabbildung α so zu erweitern, daß die Bedingung (4.13b) an das Besetzungsmuster sichergestellt ist, und

$$\tilde{A}^p_{ij} := \sum_{q \in \alpha(i) \cap \alpha'(j)} \hat{A}^q_{ij} \quad \forall i \in J \ \forall j \in J' \ \forall p \in \alpha(i) \cap \alpha'(j).$$

zu berechnen. Die Bedingung an α ist aufgrund der Vollständigkeit des Überlapps ${}_2\mathcal{U}$ nach Abschnitt 5.3.4 hier immer erfüllt. Folgender Algorithmus führt das Einsammeln und Summieren der einzelnen Anteile aus (nach [Bas96b, Funktion MatCons] mit kleinen Korrekturen):

Algorithmus 4.3 (MakeMatrixPartlyConsistent($\hat{A}$))

$Q^p := \{ q \in \mathcal{P} \setminus \{p\} \mid J^q \cap J^p \neq \emptyset \}$

(1) Gather: $\forall q \in Q^p \ \forall i \in J^p \cap J^q \ \forall j \in \mathcal{N}^p(i) \cap J'^q$: kopiere $\hat{A}^p_{ij}$ in Nachrichtenpuffer $M_{p,q}$ für $\hat{A}^q_{ij}$

(2) Datenaustausch: $\forall q \in Q^p$: sende $M_{p,q}$ an q und empfange $M_{q,p}$ von q

(3) Scatter: $\forall q \in Q^p \ \forall i \in J^p \cap J^q \ \forall j \in J'^p$: $\exists \hat{A}^q_{ij} \in M_{q,p} \Rightarrow$ hole $\hat{A}^q_{ij}$ aus $M_{q,p}$ und summiere

$$\hat{A}^p_{ij} := \hat{A}^p_{ij} + \hat{A}^q_{ij}$$

Am Ende der Operation steht in $\hat{A}^p$ die teilweise konsistent verteilte Matrix und muß folglich als $\tilde{A}^p$ bezeichnet werden.

Beweis: Nach Definition 4.1.25 ist $\hat{A}$ eine additive Verteilung der Matrix A mit den Komponenten $A_{ij} = \sum_{p \in \alpha(i) \cap \alpha'(j)} A^p_{ij}$. In Schritt (1) werden die Komponenten $\hat{A}^p_{ij}$ für alle $\hat{A}^q_{ij}$ mit $(q,i,j) \in \{ (q,i,j) \in \mathcal{P} \setminus \{p\} \times J^p \times J'^p \mid q \in \alpha(i) \ \wedge \ j \in \mathcal{N}^p(i) \cap J'^q \}$ in die Puffer $M_{p,q}$ gestellt. Daher gilt im Resultat

$$\forall i \in J^p \ \forall j \in J'^p : \quad \hat{A}^p_{ij} + \sum_{q \in \mathcal{P} \mid q \in \alpha(i) \setminus \{p\} \ \wedge \ j \in \mathcal{N}^q(i) \cap J'^p} \hat{A}^q_{ij}$$

Beachte: Die Information „$j \in \mathcal{N}^q(i) \cap J'^p$“ liegt nicht in der lokalen, algebraischen Datenstruktur ($\hat{A}^p$) auf dem Prozessor p vor, sondern nur im Nachrichtenpuffer $M_{q,p}$. Es gilt $j \in J'^p$ nach Voraussetzung. Man kann auch über $q = p$ summieren, da das genau den Eintrag $\hat{A}^p_{ij}$ ergibt. Daher folgt:

$$= \sum_{q \in \mathcal{P} \mid q \in \alpha(i) \ \wedge \ j \in \mathcal{N}^q(i)} \hat{A}^q_{ij}$$

da $\hat{A}_{ij}^q = 0 \ \forall j \notin \mathcal{N}^q(i)$ (nach (4.10)) und $\forall q \notin \alpha'(j)$ (nach Definition 4.1.12)

$$\begin{aligned} &= \sum_{q \in \mathcal{P} \mid q \in \alpha(i) \ \wedge \ q \in \alpha'(j)} \hat{A}_{ij}^q \\ &= \sum_{q \in \alpha(i) \cap \alpha'(j)} \hat{A}_{ij}^q \\ &= A_{ij}. \end{aligned}$$

D. h. alle darstellbaren Einträge $\hat{A}_{ij}^p$ tragen im Ergebnis den gleichen, globalen Wert A_{ij}, was nach Definition 4.1.25 die Bezeichnung $\tilde{A}^p$ für eine teilweise konsistent verteilte Matrix erfordert. □

Bemerkung 4.1.33
Es wird immer auf einem fixen Objekt gearbeitet; bis auf die Nachrichtenpuffereinträge werden keine Kopien angelegt. Während den Scatteroperationen befinden sich die Vektoren bzw. Matrizen folglich in einem intermediären Zustand: Sie sind weder in Ausgangsdarstellungsform noch in der Zielform; erst nach Abschluß aller Scatteraufrufe liegt wieder eine definierte Darstellungsform vor. In diesem Sinne muß die symbolisierte Darstellungsform verstanden werden: Es ist lediglich der Speicherplatz des Operanden gemeint.

Bemerkung 4.1.34
Da eine Transformation die Darstellungsform ändert, darf sie nicht mehrfach unmittelbar hintereinander auf das selbe Objekt angewendet werden, denn die Darstellungsform des Operanden stimmt dann nicht mehr. In der derzeitigen Implementierung ist die vorliegende Darstellungsform eines Objekt allerdings *nicht* einfach erkennbar, denn es trägt kein entsprechendes Merkmal, sodaß eine Operation *nicht* einfach prüfen kann, ob die Darstellungsformen der Operanden korrekt sind. Eine Konsequenz ist, daß das Programm die mehrmalige Anwendung einer Transformation ausführen würde, ohne einen Fehler erkennen zu können; jedoch werden unter Umständen manche Komponenten mit „falschen“ Ergebnissen belegt: So führt z. B. die irrtümliche Anwendung von MakeAdditiveVectorConsistent auf einen bereits konsistenten Vektor dazu, daß jede Komponente $\bar{x}_j^p$ das $|\alpha(j)|$–fache Vielfache des korrekten Wertes erhält. Man muß die Transformationen also an genau den richtigen Stellen im Algorithmus plazieren.

Bemerkung 4.1.35 (Organisation der Nachrichtenpuffer)
Wir wollen hier die Strukturierung der Nachrichtenpuffer nicht näher spezifizieren. Das soll der Implementierung überlassen bleiben.

Bemerkung 4.1.36 (Realisierung mit DDD)
Die hier verwendete Bibliothek DDD [Bir97] [Bir98] unterstützt auch bei der Realisierung der Transformationen den Anwender sehr stark. Das Wissen, welcher Knoten auf welchem anderen Prozessor Kopien besitzt (in der Algorithmusbeschreibung abstrakt

als Q^p und $j \in J^p \cap J^q$ formuliert), wird von DDD als sogenannte Interfaces verwaltet. DDD verwendet diese Information dazu, den gesamten Kommunikationsablauf durchzuführen. Der Anwender muß dafür nur noch 2 Callbackfunktionen anmelden: eine *Gatherfunktion*, die den bereitgestellten Nachrichtenpuffereintrag füllt, und eine *Scatterfunktion*, die auf Empfängerseite den gelieferten Nachrichtenpuffereintrag in das lokale Datenobjekt einbaut (indem z. B. der empfangene Wert auf den bereits vorliegenden aufaddiert wird). Da Matrizen in *UG* nicht bei DDD als Objekte angemeldet werden, um Speicherplatz zu sparen, muß sich der Benutzer bei der Programmierung der Transformationen für Matrizen selbst um die Strukturierung des Nachrichtenpuffers dafür kümmern. DDD unterstützt einen dabei noch durch global eindeutige Identifikationsnummern (GID) für die (Spalten-)Vektorkomponenten.

4.1.5 Parallele lineare Algebra Operationen

Nachdem nun verteilte Vektoren und Matrizen eingeführt worden sind, können auf ihnen die üblichen Operationen der linearen Algebra beschrieben werden. Die englische Abkürzung BLAS für basic linear algebra system wird häufig dafür verwendet. Grundsätzlich sollen die Operationen lokal auf dem *gesamten* Vektor bzw. Matrix ausgeführt werden, also auf Master– wie Borderkomponenten. Wichtig ist dabei, auf die parallele Darstellungsform zu achten, die sowohl Anforderung an die Operanden als auch Zusicherung an das Ergebnis ist. Erfüllt ein Operand die Anforderung nicht, muß seine Darstellungsform mit Transformationen aus dem vorhergehenden Abschnitt angepaßt werden.

Im folgenden sei immer $v, x, y, z \in \mathbb{R}^n, w \in \mathbb{R}^{n'}, A, D \in \mathbb{R}^{n \times n'}$.

4.1.5.1 Nullsetzen

$$\bar{x} := \hat{x} := \check{x} := 0 \tag{4.15}$$

$$\bar{A} := \tilde{A} := \hat{A} := 0, \tag{4.16}$$

d. h. das Ergebnis erfüllt gleichzeitig die Anforderungen aller paralleler Darstellungsformen.

Beweis: Ein Objekt, das vollständig auf 0 gesetzt ist, erfüllt nach Definition 4.1.16 bzw. 4.1.25 die Eigenschaften aller Darstellungsformen. □

4.1.5.2 Vektorraumoperationen für Vektoren

Für $\beta, \gamma \in \mathbb{R}$:

$$\bar{z} := \beta\bar{x} + \gamma\bar{y} \tag{4.17}$$

$$\hat{z} := \beta\hat{x} + \gamma\hat{y}. \tag{4.18}$$

Die Operation ist lokal ausführbar. Bei der Addition müssen beide Operanden die gleiche Verteilungsform haben, die sich auch auf das Ergebnis überträgt.

Beweis: a) $\bar{z} := \beta\bar{x} + \gamma\bar{y}$

Es wird lokal berechnet:

$$\begin{aligned}
&\beta\bar{x}^p + \gamma\bar{y}^p && \text{komponentenweise}\\
&= \left(\beta\bar{x}_j^p + \gamma\bar{y}_j^p\right)_{j\in J^p} && x, y \text{ konsistent, (4.7)}\\
&= \Big(\underbrace{\beta x_j + \gamma y_j}_{=:z_j}\Big)_{j\in J^p} && \text{komponentenweise}\\
&=: \bar{z}^p,
\end{aligned}$$

d. h. das Ergebnis liegt nach Definition 4.1.16 in konsistenter Form vor.

b) $\hat{z} := \beta\hat{x} + \gamma\hat{y}$

Es wird lokal berechnet:

$$\begin{aligned}
&\beta\hat{x}^p + \gamma\hat{y}^p && \text{komponentenweise}\\
&= \Big(\underbrace{\beta\hat{x}_j^p + \gamma\hat{y}_j^p}_{=:v_j^p}\Big)_{j\in J^p} && \text{komponentenweise} \qquad (4.19)\\
&=: v^p.
\end{aligned}$$

Für den Ergebnisvektor $(v^p)_{p\in\mathcal{P}}$ ist nun zu zeigen, daß er eine additive Verteilung des Ergebnisses $z := \beta x + \gamma y$ ist:

$$\begin{aligned}
\forall j \in J: \quad &\sum_{p\in\alpha(j)} v_j^p && \text{nach (4.19)}\\
&= \sum_{p\in\alpha(j)} \beta\hat{x}_j^p + \gamma\hat{y}_j^p && \text{Distributivgesetz}\\
&= \beta \sum_{p\in\alpha(j)} \hat{x}_j^p + \gamma \sum_{p\in\alpha(j)} \hat{y}_j^p && x, y \text{ additiv, (4.8)}\\
&= \beta x_j + \gamma y_j.
\end{aligned}$$

Damit ist nach Definition 4.1.16 die Bezeichnung $\hat{z}_j^p := v_j^p$ gerechtfertigt. □

4.1.5.3 Skalarprodukt

$$\beta := \hat{x}^\top\bar{y} = \bar{x}^\top\hat{y} = \sum_{p\in\mathcal{P}} \beta^p \quad \text{mit } \beta^p := \hat{x}^{p\top}\bar{y}^p \text{ bzw. } \beta^p := \bar{x}^{p\top}\hat{y}^p, \qquad (4.20)$$

d. h. genau ein Operand muß konsistent und der andere muß additiv sein. Nach Bildung der lokalen Skalarprodukte β^p muß noch in einer **globalen Kommunikation** die Summe aller Teil–Skalarprodukte gebildet werden. Ohne Kommunikation kann es nicht gehen, weil $x^\top y$ eine globale Information ist und daher lokale Anteile in irgendeiner Form eingesammelt werden müssen.

Beweis: Wir betrachten den Fall $\hat{x}^\top \bar{y}$; der Fall $\bar{x}^\top \hat{y}$ läßt sich aufgrund der Symmetrieeigenschaft des Skalarprodukts auf den Fall $\hat{x}^\top \bar{y}$ zurückführen.

Es wird lokal berechnet:

$$\hat{x}^{p\top} \bar{y}^p = \sum_{j \in J^p} \hat{x}_j^{p\top} \bar{y}_j^p =: \beta^p$$

Die globale Summierung ergibt:

$$\begin{aligned}
\beta := \sum_{p \in \mathcal{P}} \beta^p &= \sum_{p \in \mathcal{P}} \sum_{j \in J^p} \hat{x}_j^p \bar{y}_j && y \text{ konsistent, (4.7)} \\
&= \sum_{p \in \mathcal{P}} \sum_{j \in J^p} \hat{x}_j^p y_j && \text{Definition } J^p, \text{ (4.5)} \\
&= \sum_{p \in \mathcal{P}} \sum_{j \in J | p \in \alpha(j)} \hat{x}_j^p y_j
\end{aligned}$$

Diese Summation stellt eine Enumeration der Menge $\{ (p,j) \in \mathcal{P} \times J \mid p \in \alpha(j) \}$ dar. Eine andere Enumeration ergibt

$$\begin{aligned}
&= \sum_{j \in J} \sum_{p \in \alpha(j)} \hat{x}_j^p y_j && \text{Distributivgesetz} \\
&= \sum_{j \in J} \left(\sum_{p \in \alpha(j)} \hat{x}_j^p \right) y_j && x \text{ additiv, (4.8)} \\
&= \sum_{j \in J} x_j y_j = x^\top y.
\end{aligned}$$

□

4.1.5.4 Euklidische Vektornorm

$$\beta := ||\check{x}|| = \sqrt{\sum_{\boldsymbol{p \in \mathcal{P}}} \beta^p} \quad \text{mit } \beta^p := \sum_{j \in J^p} (\check{x}_j^p)^2, \tag{4.21}$$

d. h. die Norm eines Vektors kann dann berechnet werden, wenn er in eindeutiger Form verteilt ist. Nach Bildung der lokalen Summen (noch keine Normen!) muß noch in einer **globalen Kommunikation** die Gesamtsumme gebildet werden, bevor die Wurzel gezogen werden kann. Ohne Kommunikation kann es nicht gehen, weil $||x||$ eine globale Information ist und daher lokale Anteile in irgendeiner Form eingesammelt werden müssen.

Bemerkung 4.1.37 ($||\bar{x}||$)
Wenn man von dem Paradigma abweichen will, daß lokal der gesamte Vektor verarbeitet werden muß, kann man auch die Norm eines konsistent verteilten Vektors bilden, indem man lokal nur über die Masterkomponenten summiert:

$$\beta := ||\bar{x}|| = \sqrt{\sum_{\boldsymbol{p\in\mathcal{P}}} \beta^p} \text{ mit } \beta^p := \sum_{\boldsymbol{j\in\mathcal{M}^p}} (\bar{x}_j^p)^2. \tag{4.22}$$

Bemerkung 4.1.38 (Bezug zum Skalarprodukt)
In beiden Fällen wird lokal *nicht* die vollständige lineare Operation ausgeführt. Weiterhin muß man feststellen, daß sich die Norm *nicht* auf die Berechnung eines Skalarprodukts in der sonst üblichen Form $||x|| := \sqrt{x^\top x}$ zurückführen läßt, weil das parallele Skalarprodukt entgegengesetzte Darstellungsformen seiner Operanden erfordert (siehe Abschnitt 4.1.5.3); das kann von ein und demselben Vektor im allgemeinen *nicht* erfüllt werden (Ausnahme siehe Bemerkung 4.1.18).

Beweis: a) $||\check{x}||$
Die globale Summierung ergibt:

$$\begin{aligned}
\gamma &:= \sum_{p\in\mathcal{P}} \beta^p = \sum_{p\in\mathcal{P}} \sum_{j\in J^p} (\check{x}_j^p)^2 && \check{x} \text{ eindeutig, d.h. } \check{x}_j^p = 0 \ \forall j \notin \mathcal{M}^p, (4.9) && \text{(i)}\\
&= \sum_{p\in\mathcal{P}} \sum_{j\in\mathcal{M}^p} (\check{x}_j^p)^2 && \check{x} \text{ eindeutig, d.h. } \check{x}_j^p = x_j \ \forall j \in \mathcal{M}^p, (4.9) && \text{(ii)}\\
&= \sum_{p\in\mathcal{P}} \sum_{j\in\mathcal{M}^p} x_j^2 && \text{Eindeutigkeit der Abb. } \mu, \text{ Lemma 4.1.7}\\
&= \sum_{j\in J} x_j^2.
\end{aligned}$$

$$\Rightarrow \quad \beta := \sqrt{\gamma} = \sqrt{\textstyle\sum_{j\in J} x_j^2} = ||x||.$$

b) $||\bar{x}||$
$\bar{x}$ erfüllt nach Definition 4.1.16 ebenfalls für seine Masterkomponeten Bedingung (ii) in Teil a); Bedingung (i) wird dadurch eingehalten, daß in (4.22) die Borderkomponenten bei der Summierung ausgelassen werden. Damit kann der Beweis von Teil a) auch hier angewendet werden. □

4.1.5.5 Matrix–Vektor–Multiplikation

$$\bar{x} := \bar{A}\bar{w} \tag{4.23}$$

$$\hat{y} := \bar{D}\hat{x} \quad (D \text{ Diagonalmatrix}, J = J') \tag{4.24}$$

$$\hat{w} := \bar{A}^\top \hat{x} \tag{4.25}$$

$$\hat{x} := \tilde{A}\check{w} \tag{4.26}$$

$$\hat{x} := \hat{A}\bar{w} \tag{4.27}$$

Alle anderen Matrix–Vektor–Multiplikation sind nicht ohne weitere Einschränkungen z. B. an das Besetzungsmuster gültig [Haa99b].

Beweis: a) $\bar{x} := \bar{A}\bar{w}$

Es wird lokal berechnet (Matrixmultiplikation komponentenweise betrachtet):

$$\bar{A}^p \bar{w}^p = \Bigg(\underbrace{\sum_{j \in \mathcal{N}^p(i)} \bar{A}^p_{ij} \bar{w}^p_j}_{x^p_i} \Bigg)_{i \in J^p} =: x^p. \tag{4.28}$$

Für dieses lokale Ergebnis ist nun zu zeigen:

$$\forall i \in J \ \forall p \in \alpha(i): \quad x^p_i = x_i := (Aw)_i.$$

Es gilt:

$$\begin{aligned}
\forall i \in J \ \forall p \in \alpha(i): \quad x^p_i & && \text{Definition } x^p_i, \text{ (4.28)} \\
&= \sum_{j \in \mathcal{N}^p(i)} \bar{A}^p_{ij} \bar{w}^p_j && w \text{ konsistent, (4.7)} \\
&= \sum_{j \in \mathcal{N}^p(i)} \bar{A}^p_{ij} w_j && \bar{A} \text{ konsistent, (4.12b)} \\
&= \sum_{j \in \mathcal{N}(i)} \bar{A}^p_{ij} w_j && p \in \alpha(i) \text{ nach Vor., (4.12c)} \\
&= \sum_{j \in \mathcal{N}(i)} A_{ij} w_j = (Aw)_i.
\end{aligned}$$

Damit ist nach Definition 4.1.16 $(x^p)_{p \in \mathcal{P}}$ eine konsistente Darstellung des Vektors Aw und rechtfertigt somit die Bezeichnung $\bar{x} := \bar{A}\bar{w}$.

b) $\hat{y} := \bar{D}^\top \bar{x}$

Es wird lokal berechnet (Matrixmultiplikation komponentenweise betrachtet):

$$\bar{D}^p \hat{x}^p = \big(\underbrace{\bar{D}^p_{ii} \hat{x}^p_i}_{y^p_i} \big)_{i \in J^p} =: y^p. \tag{4.29}$$

Für dieses lokale Ergebnis ist nun zu zeigen:

$$\forall i \in J: \sum_{p \in \alpha(i)} y^p_i = y_i := (Dx)_i.$$

Es gilt:

$$\begin{aligned}
\forall i \in J: \quad & \sum_{p\in\alpha(i)} y_i^p && \text{Definition } y_i^p, \text{ (4.29)}\\
&= \sum_{p\in\alpha(i)} \bar{D}_{ii}^p \hat{x}_i^p && D \text{ konsistent, (4.12c)}\\
&= D_{ii} \sum_{p\in\alpha(i)} \hat{x}_i^p && x \text{ additiv, (4.8)}\\
&= D_{ii} x_i = (Dx)_i.
\end{aligned}$$

Damit ist nach Definition 4.1.16 $(y^p)_{p\in\mathcal{P}}$ eine additive Darstellung des Vektors Dx und rechtfertigt somit die Bezeichnung $\hat{y} := \bar{D}\hat{x}$.

c) $\hat{w} := \bar{A}^\top \hat{x}$

Es wird lokal berechnet (Matrixmultiplikation komponentenweise betrachtet):

$$\bar{A}^{p\top}\hat{x}^p = \Bigg(\underbrace{\sum_{i\in\mathcal{N}^p(j)} \bar{A}_{ji}^{p\top} \hat{x}_i^p}_{w_j^p} \Bigg)_{j\in J'^p} =: w^p. \tag{4.30}$$

Für dieses lokale Ergebnis ist nun zu zeigen:

$$\forall j \in J': \sum_{p\in\alpha'(j)} w_j^p = w_j := (A^\top x)_j.$$

Es gilt:

$$\begin{aligned}
\forall j \in J': \quad & \sum_{p\in\alpha'(j)} w_j^p && \text{Definition } w_j^p, \text{ (4.30)}\\
&= \sum_{p\in\alpha'(j)} \sum_{i\in\mathcal{N}^p(j)} \bar{A}_{ji}^{p\top} \hat{x}_i^p && A_{ji}^\top = A_{ij}
\end{aligned}$$

Diese Summation stellt eine Enumeration der Menge $\{\,(p,i) \in \mathcal{P}\times J \mid p \in \alpha(i) \ \wedge\ p \in \alpha'(j)\,\}$ dar. Beachte $A_{ij}^p = 0$, falls $j \notin \mathcal{N}^p(i)$. Eine andere Enumeration ergibt

$$\begin{aligned}
&= \sum_{i\in J} \sum_{p\in\alpha(i)\cap\alpha'(j)} \bar{A}_{ij}^p \hat{x}_i^p && A \text{ konsistent, (4.12b)}\\
&= \sum_{i\in J} \sum_{p\in\alpha(i)} \bar{A}_{ij}^p \hat{x}_i^p && A \text{ konsistent, (4.12c)}\\
&= \sum_{i\in J} \sum_{p\in\alpha(i)} A_{ij} \hat{x}_i^p && A_{ij} = A_{ji}^\top \text{ unabhängig von } p\\
&= \sum_{i\in J} A_{ji}^\top \sum_{p\in\alpha(i)} \hat{x}_i^p && x \text{ additiv, (4.8)}\\
&= \sum_{i\in J} A_{ji}^\top x_i = (A^\top x)_j.
\end{aligned}$$

Damit ist nach Definition 4.1.16 $(w^p)_{p\in\mathcal{P}}$ eine additive Darstellung des Vektors Ax und rechtfertigt somit die Bezeichnung $\hat{w} := \bar{A}^\top \hat{x}$.

d) $\hat{x} := \tilde{A}\check{w}$

Es wird lokal berechnet (Matrixmultiplikation komponentenweise betrachtet):

$$\tilde{A}^p \check{w}^p = \Bigg(\underbrace{\sum_{j\in\mathcal{N}^p(i)} \tilde{A}^p_{ij} \check{w}^p_j}_{x^p_i} \Bigg)_{i\in J^p} =: x^p. \tag{4.31}$$

Für dieses lokale Ergebnis ist nun zu zeigen:

$$\forall i \in J : \sum_{p\in\alpha(i)} x^p_i = x_i := (Aw)_i.$$

Es gilt $\forall j \in J$:

$$\begin{aligned}
&\sum_{p\in\alpha(i)} x^p_i && \text{Definition } x^p_i, \text{ (4.31)}\\
&= \sum_{p\in\alpha(i)} \sum_{j\in\mathcal{N}^p(i)} \tilde{A}^p_{ij} \check{w}^p_j
\end{aligned}$$

Diese Summation stellt eine Enumeration der Menge $\{\, (p,j) \in \mathcal{P}\times J' \mid p \in \alpha(i) \ \wedge\ p \in \alpha'(j) \,\}$ dar. Beachte $A^p_{ij} = 0$, falls $j \notin \mathcal{N}^p(i)$. Eine andere Enumeration ergibt

$$\begin{aligned}
&= \sum_{j\in J'} \sum_{p\in\alpha(i)\cap\alpha'(j)} \tilde{A}^p_{ij} \check{w}^p_j && A \text{ teilweise konsistent, (4.13c)}\\
&= \sum_{j\in J'} \sum_{p\in\alpha(i)\cap\alpha'(j)} A_{ij} \check{w}^p_j && \text{disjunkte Aufteilung von } \alpha'(j)\\
&= \sum_{j\in J'} \sum_{p\in\alpha(i)\cap\{\mu'(j)\}} A_{ij} \check{w}^p_j\\
&\quad + \sum_{j\in J'} \sum_{p\in\alpha(i)\cap(\alpha'(j)\setminus\{\mu'(j)\})} \underbrace{A_{ij} \check{w}^p_j}_{\equiv 0}\\
&&& (w \text{ eindeutig, (4.9)})
\end{aligned}$$

A teilweise konsistent, Bedingung an α, (4.13b)

$$\begin{aligned}
&= \sum_{j\in J'} A_{ij} \check{w}^{\mu'(j)}_j && w \text{ eindeutig, (4.9)}\\
&= \sum_{j\in J'} A_{ij} w_j = x_i = (Aw)_i.
\end{aligned}$$

Damit ist nach Definition 4.1.16 $(x^p)_{p\in\mathcal{P}}$ eine additive Darstellung des Vektors Aw und rechtfertigt somit die Bezeichnung $\hat{x} := \tilde{A}\check{w}$.

e) $\hat{x} := \hat{A}\bar{w}$

Es wird lokal berechnet (Matrixmultiplikation komponentenweise betrachtet):

$$\hat{A}^p\bar{w}^p = \Bigg(\underbrace{\sum_{j\in\mathcal{N}^p(i)} \hat{A}^p_{ij}\bar{w}^p_j}_{x^p_i}\Bigg)_{i\in J^p} =: x^p. \tag{4.32}$$

Für dieses lokale Ergebnis ist nun zu zeigen:

$$\forall i \in J : \sum_{p\in\alpha(i)} x^p_i = x_i := (Aw)_i.$$

Es gilt:

$$\begin{aligned} \forall i \in J : \quad & \sum_{p\in\alpha(i)} x^p_i && \text{Definition } x^p_i, \text{ (4.32)} \\ & = \sum_{p\in\alpha(i)} \sum_{j\in\mathcal{N}^p(i)} \hat{A}^p_{ij}\bar{w}^p_j \end{aligned}$$

Diese Summation stellt eine Enumeration der Menge $\{\,(p,j) \in \mathcal{P}\times J' \mid p \in \alpha(i) \ \wedge\ p \in \alpha'(j)\,\}$ dar. Beachte $A^p_{ij} = 0$, falls $j \notin \mathcal{N}^p(i)$. Eine andere Enumeration ergibt

$$\begin{aligned} & = \sum_{j\in J'} \sum_{p\in\alpha(i)\cap\alpha'(j)} \hat{A}^p_{ij}\bar{w}^p_j && w \text{ konsistent, (4.7)} \\ & = \sum_{j\in J'} \sum_{p\in\alpha(i)\cap\alpha'(j)} \hat{A}^p_{ij} w_j && \text{A additiv, (4.14b)} \\ & = \sum_{j\in J'} \sum_{p\in\alpha(i)\cap\alpha'(j)} A_{ij} w_j = x_i = (Aw)_i. \end{aligned}$$

Damit ist nach Definition 4.1.16 $(x^p)_{p\in\mathcal{P}}$ eine additive Darstellung des Vektors Aw und rechtfertigt somit die Bezeichnung $\hat{x} := \hat{A}\bar{w}$. □

Bemerkung 4.1.39 ($\bar{x} := \bar{A}\bar{w}$)
Was in [Wie00b] noch als definierende Forderung an eine konsistente Matrix gestellt wurde, läßt sich nun als Folgerung herleiten: $\bar{x} := \bar{A}\bar{w}$. Es gehen in die Definition 4.1.25 nur noch unmittelbar an der Matrix feststellbare Eigenschaften ein.

Bemerkung 4.1.40 ($\tilde{A}\hat{x}$ unmöglich)
Teil d) im letzten Beweis läßt auch gut erkennen, warum eine Operation der Art $\tilde{A}\hat{x}$ für einen allgemeinen additiven Vektor $\hat{x}$ *nicht* korrekt ablaufen kann: Im Endergebnis müßte $\forall i \in \mathcal{N}(j)$ und $\forall p \in \alpha'(j)$ ein Anteil $A^p_{ij}\hat{x}^p_j$ enthalten sein. Da aber im allgemeinen $p \notin \alpha(i)$ sein kann, würden im Endergebnis entsprechende Anteile fehlen. Vergleiche dazu auch Bemerkung 4.1.29.

Bemerkung 4.1.41
Alle linearen Algebra Operationen können *lokal* ausgeführt werden; nur das Skalarprodukt und die Norm erfordern abschließend das globale Aufaddieren aller lokalen Anteile. Bis auf diese beiden Operationen erfordert also die Parallelisierung zunächst keinen Mehraufwand. Allerdings müssen, wie schon mehrmals festgestellt, unter Umständen die parallele Darstellungsform der Operanden transformiert werden; das verursacht dann erheblichen Zusatzaufwand in der parallelen Ausführung, kann aber durch geschickte Anordnung der einzelnen linearen Algebra Operationen nach Möglichkeit gering gehalten werden: Man versucht, eine Darstellungsform des Ergebnisses zu erzielen, die für die nächsten Operationen auch gleich als Darstellungsform der Operanden zulässig ist. Die optimale Darstellungsform für jeden Schritt kann aber nur aus der Betrachtung des gesamten Programmablaufs ermittelt werden, wobei einzelne Algorithmusteile ihrem Zeitanteil an der Gesamtlaufzeit entsprechend gewichtet werden müssen. Die daraus resultierenden Konflikte zwischen Modularität im Algorithmusaufbau und Laufzeiteffizienz werden wir im Zweifelsfall zugunsten der Modularität entscheiden.

4.2 Paralleles geometrisches Mehrgitterverfahren

In diesem Abschnitt sollen die bisher erläuterten Grundlagen dazu benutzt werden, die einfachste Form eines parallelen Mehrgitterverfahrens für unstrukturierte Gitter zu entwickeln. Die vielfältigen Erweiterungsmöglichkeiten sollen hier nur angedeutet werden. Dieser Algorithmus stellt sowohl die Ausgangsbasis für die Überlegungen zur Parallelisierung des FAMG im nächsten Kapitel dar, als auch den bestehenden Rahmen, in den sich dessen Implementierung im Sinne des Abschnitts 2.3.3.5 eingliedern muß.

Das Ergebnis wird das multiplikative Mehrgitterverfahren für unstrukturierte Gitter sein, wie es derzeit auch im Programmbaukasten *UG* verwendet wird [Bas96b] [BBJ+97] [BBJ+99] [Wie00a] [Wie00b] [BBJ+00] [BJL+01]. Von dem Verfahren wollen wir hier aber alle Aspekte weglassen, die nicht direkt für den Rest der Arbeit wichtig sind: Adaptivität, Lokalität, dynamische Lastverteilung, nicht–konforme Finite Elemente. Auf dieser Ebene muß noch nicht zwischen 2– und 3–dimensionalen Rechengebieten unterschieden werden. Das sind drastische Einschränkungen; trotzdem werden die charakteristischen Schwierigkeiten, die auch beim parallelen FAMG auftreten werden, schon erkennbar. Alternative Algorithmen für vereinfachte Situationen sind z. B. in [Haa99b] dargestellt.

Die Beschreibung soll modular erfolgen, um die Struktur klar herauszuarbeiten und ein allgemein tragfähiges Gerüst aufzubauen. Eine Folge wird sein, daß für jede Funktion die parallele Darstellungsform ihrer Operanden beim Aufruf und im gelieferten Ergebnis gleich ist; so kann der Funktionsaufruf ohne weitere Anpassungen immer dort eingeschoben werden, wo die Voraussetzungen erfüllt sind. In konkreten Fällen werden so allerdings manche Code–Optimierungen nicht mehr möglich sein.

Das resultierende parallele Gesamtverfahren wird am Ende dieses Abschnitts zusammenfassend dargestellt. In **Fettdruck** gesetzt sind jeweils die Erweiterungen und Änderungen bezüglich der seriellen Algorithmen in Abschnitt 2.3.3, die durch die Parallelisierung bedingt sind.

4.2.1 Parallelrechner und Programmiermodell

Da das Ziel dieser Arbeit die Entwicklung von Lösungsverfahren für sehr große Gleichungssysteme ist, sollten die Rechner, für die die Verfahren konzipiert werden, zur höchsten Leistungsklasse gehören. Neben den ASCI Rechnern mit einigen tausend Prozessoren sind das im Moment die Modelle Cray T3E, IBM SP und Hitachi SR8000 mit bis zu mehreren hundert Prozessoren und bis zu einigen hundert MB lokalem Speicher [Top]. Ihnen allen ist gemeinsam, daß sie nach Flynn's Klassifikation MIMD (multiple instruction multiple data [Fly66] [Fly72]) Rechner sind, d. h. jeder Prozessor hat sein eigenes Programm und arbeitet auf seinen eigenen Daten. Im Bereich der Numerik bietet sich die Unterklasse SPMD (same program multiple data [Lew91]) an, weil auf den verteilten Daten im wesentlichen überall das gleiche Programm abläuft. Die Kopplung sehr vieler Prozessoren läßt kein echtes shared memory mehr zu, so daß nur noch distributed memory (oder allenfalls virtual shared memory) Applikationen effizient sind; d. h. die Kommunikation zwischen den Prozessoren geschieht durch expliziten (oder impliziten) Nachrichtenaustausch. Dafür hat sich inzwischen MPI (message passing interface) oder auch noch PVM (parallel virtual machine) als Programmierschnittstelle durchgesetzt. Daneben gibt es verschiedene Erweiterungen bekannter Hochsprachen, die versuchen, den expliziten Nachrichtenaustausch vor dem Anwender zu verbergen (z. B. HPF (high performance fortran [HPF93]) oder Fortran90). Da diese automatische Parallelisierung üblicherweise nur auf strukturierten Daten (arrays) funktioniert, scheidet sie für unsere Anwendungen auf unstrukturierten Gittern aus. Eine Einführung in das Gebiet der Parallelrechner und ihrer Programmiermodelle gibt z. B. [Brä93] [KGGK94]. Wir verwenden expliziten Nachrichtenaustausch, der über die Bibliothek PPIF (parallel programming interface [Bas93]) abstrahiert wird, die sich ihrerseits auf alle üblichen Programmiermodelle wie MPI, PVM, shared memory usw. aufsetzen läßt. Dadurch erreicht die Anwendung eine sehr gute Portabilität.

Eine der Grundaufgaben des Programms wird sein, einen sehr komplexen Graphen bestehend aus Elementen, Kanten, Knoten, Vektoren, Matrizen usw. — verteilt über alle Level — aufzubauen, zu ändern und Datenaustauschoperationen darauf durchzuführen. Als sehr mächtiges Werkzeug wird dafür DDD (dynamic distributed data) verwendet [Bir97] [Bir98]. Auf der Abstraktionsebene, die in dieser Arbeit vorwiegend eingehalten wird, sollte man davon allerdings kaum etwas sehen.

Nachdem nun die Programmierumgebung definiert ist, können wir uns der eigentlichen Numerik zuwenden.

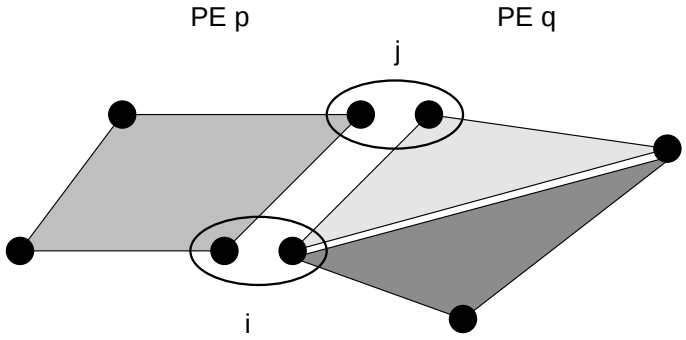


Abbildung 4.2: Partitionierung von 3 Elementen

4.2.2 Datenpartitionierung

Ausgangspunkt für ein geometrisches Mehrgitterverfahren ist eine grobe Triangulierung des Rechengebiets in Elemente (siehe Abschnitt 2.3.3.3). Erste Aufgabe ist es nun, diese Elemente auf die Prozessoren zu verteilen. Das ist auch gleich einer der Hauptschwierigkeiten, weil das Gitter unstrukturiert ist. Da unsere numerischen Verfahren immer nur die Daten in einer gewissen Nachbarschaft benötigen, sollten diese Nachbarn nach Möglichkeit auf dem gleichen Prozessor vorhanden sein, um den Kommunikationsaufwand gering zu halten; andererseits ist eine gleichmäßige Verteilung der Elemente auf die Prozessoren wünschenswert, um die Rechenlast im Gleichgewicht zu halten. Weitere schwere Anforderungen stellt ein lokales Mehrgitterverfahren: Zusammen mit einem adaptiven Verfeinern und Vergröbern müssen im Lauf der Berechnung unter Umständen ganze Element–Hierarchien umverteilt werden, um eine gute Lastverteilung aufrecht zu erhalten [Lan93] [Bas96a] [Bas96b] [Lan00]. Hier kommt die Bibliothek Chaco [HL93] zum Einsatz; ein anderes bekanntes Werkzeug dafür ist Metis [KK95].

Wir führen eine eindeutige Aufteilung der Elemente auf die Prozessoren durch. Da keine Elementkopien angelegt werden, entsteht keine Überlappung der Elemente. Abbildung 4.2 zeigt einen typischen Ausschnitt eines verteilten Gitters. Das Viereckselement werde auf Prozessor p verteilt, die beiden Dreiecke auf Prozessor q. Trotz der Überlappungsfreiheit auf Elementebene tritt eine Überlappung der Eckpunkte auf. Nach Bemerkung 4.1.11 gibt es 2 Möglichkeiten, die Knoten für die lineare Algebra nach Definition 4.1.3 festzulegen. In beiden Fällen führt die Überlappung der Elementkanten am Prozessorrand auf eine Überlappung der Knoten. An dieser Stelle muß nun *eine* Instanz zum Masterknoten, alle anderen zu Borderknoten definiert werden; alle anderen Knoten sind eindeutig einem Prozessor zugeteilt und werden zu Masterknoten erklärt. Damit sind dann die Abbildungen μ und α und Mengen J^p, $\mathcal{M}^p$ und $\mathcal{B}^p$ in Definition 4.1.3 ebenfalls festgelegt.

Werden im Laufe der Rechnung verfeinerte Gitter angelegt, sollen sie für unsere Zwecke hier immer vollständig und regelmäßig verfeinert werden. Auch das ist eine

starke Einschränkung gegenüber des Funktionsumfangs von *UG*. Die verfeinerten Kindelemente sollen auf demselben Prozessor verbleiben wie ihre Elternelemente. Da wir vollständig verfeinern wollen, bleibt das Lastgleichgewicht erhalten, es ist *keine* dynamische Lastumverteilung nötig und es gibt keinen Grund, die Kindelemente auf einem anderen Prozessor zu plazieren. Daher gilt in natürlicher Weise, daß jedes Element sein Elternelement auf demselben Prozessor hat.

4.2.3 Wahl der parallelen Darstellungsform

Nachdem die Grundlagen der parallelen linearen Algebra geschaffen sind, muß man sich entscheiden, in welchen parallelen Darstellungsformen die einzelnen Vektoren und Matrizen erzeugt werden sollen. Wünschenswert ist es, daß die Form im Verlauf der Rechnung im wesentlichen unverändert bleiben kann. Die Erfahrungen zeigen, daß es so ein Schema gibt [Bas96b] [Haa99b]. Wir wollen hier vorwegnehmen, was eine detailierte Betrachtung als günstig erweisen wird:

$\hat{A}$	Steifigkeitsmatrix additiv
$\tilde{W}$	Glättungsmatrix teilweise konsistent
$\bar{u}$	Lösungsvektor konsistent
$\hat{d}$	rechte Seite bzw. Defekt additiv
$\bar{R}, \bar{P}$	Restriktion– und Prolongationsmatrix konsistent.

Andere Aufteilungen erfordern mehr Kommunikation [Haa99b].

4.2.4 Vorbereitungsschritt

Hier sollen die Voraussetzungen zusammengefaßt werden, die gelten müssen, bevor der eigentliche Mehrgitteralgorithmus ausgeführt werden kann. Die Motivation rührt von weiter unten zu analysierenden Algorithmusteilen her und wird unter Umständen erst im Gesamtzusammenhang verständlich.

Zunächst müssen für alle Level die Steifigkeitsmatrizen bereitstehen. Das kann durch explizite Assemblierung auf jedem Level geschehen oder durch eine Galerkin–Assemblierung, die zu gegebener Feingittermatrix ${}^{\ell}\hat{A}$ mittels

$$ {}^{\ell-1}\hat{A} := {}^{\ell}\bar{R}\, {}^{\ell}\hat{A}\, {}^{\ell}\bar{P} $$

eine Grobgittermatrix konstruiert. Wir wollen uns hier vom ersten Vorgehen leiten lassen.

Um die Gittertransferoperationen durchführen zu können, müssen explizite Matrizen ${}^{\ell}\bar{R}$ und ${}^{\ell}\bar{P}$ für $\ell = \ell_{min+1}, \ldots, \ell_{max}$ erstellt werden oder entsprechende Funktionen bereitgestellt werden, die die entsprechenden Ergebnisse berechnen.

Für die Glätter benötigt man eine aus der Steifigkeitsmatrix abgeleitete, teilweise konsistent gemachte Matrix ${}^{\ell}\tilde{W}$ für $\ell = \ell_{min}, \ldots, \ell_{max}$. Da es recht aufwendig ist, eine (Kopie der Steifigkeits–)Matrix teilweise konsistent zu machen, ist es günstig, diese Operation nur einmal auszuführen und das Ergebnis für mehrmalige Verwendung abzuspeichern.

Falls man auf dem gröbsten Level einen direkten Löser verwendet, sollte dieser Level auf einem Prozessor zusammengezogen werden, die sogenannte Grobgitteragglomeration (siehe Abschnitt 4.2.10).

4.2.5 Assemblierung

Wenigstens auf dem feinsten Gitter muß die Steifigkeitsmatrix assembliert werden. Das wird üblicherweise elementweise durchgeführt. Die so entstehenden *Elementsteifigkeitsmatrizen* werden dann auf die gesamte Steifigkeitsmatrix addiert [Bra97].

Da die Elemente eindeutig und vollständig auf Prozessoren verteilt wurden, kann *jede* Elementsteifigkeitsmatrix lokal berechnet werden und auf den lokal vorhandenen Teil der Steifigkeitsmatrix aufaddiert werden, weil die Datenpartitionierung sichergestellt hat, daß alle zu einem Element gehörigen Knoten ebenfalls vorhanden sind. Die globale Steifigkeitsmatrix liegt also ihrerseits *additiv* verteilt vor und soll es auch bleiben, denn nach Lemma 4.1.27 kann es gar keine konsistente Verteilung dafür geben. Da jeder Matrixeintrag in der globalen Matrix aus einer Elementsteifigkeitsmatrix stammt, folgt, daß unter der gewählten Datenaufteilung auch sichergestellt ist, daß die Aufteilungsabbildung den Matrixgraphen respektiert.

Da die rechte Seite ebenfalls elementweise assembliert wird, gilt analog, daß dieser Vektor zunächst in *additiv* verteilter Form vorliegt.

Treten Dirichlet–Randbedingungen auf, muß darauf geachtet werden, daß *alle* Instanzen dieser Komponenten den gleichen Wert erhalten [Wie00b].

Für nicht-konforme Finite Elemente geschieht die Assemblierung im wesentlichen genauso. Allerdings sind entscheidende Änderungen im anschließenden Mehrgitterlöser notwendig [Wie00b].

4.2.6 Defektberechnung

Aufgrund der festgelegten parallelen Darstellungsformen kann der Defekt direkt und *ohne* Kommunikation nach Gleichung (4.27) und (4.18) berechnet werden:

$$\hat{d} := \hat{f} - \hat{A}\,\bar{u}.$$

Für die Defektkorrektur $\hat{d}$ zu einer Lösungskorrektur $\bar{t}$ in $\bar{u} := \bar{u} + \bar{t}$ gilt dementsprechend

$$\hat{d} := \hat{d} - \hat{A}\,\bar{t}.$$

4.2.7 Parallele Glätter

Bei den Glättern werden Änderungen und auch Abstriche zu deren serieller Formulierung nötig sein. Die allgemeine Formulierung ist:

Algorithmus 4.4 (Iter($\bar{c}$, $\hat{d}$))
(1) Löse näherungsweise $A\hat{t} = \hat{d}$
(2) $\bar{\boldsymbol{t}}$ **:= MakeAdditiveVectorConsistent** ($\hat{\boldsymbol{t}}$)
(3) $\hat{d} := \hat{d} - \hat{A}\,\bar{t}$ nach (4.27) und (4.18)
(4) $\bar{c} := \bar{c} + \bar{t}$ nach (4.17)

Der einfachste Fall ist das Jacobiverfahren (serielle Fassung siehe Abschnitt 2.3.2) mit $W := \mathrm{diag}(A)$. Man möchte W zu einer konsistent verteilten Matrix transformieren. Die Forderungen (4.12b) sind für W bereits in trivialer Weise erfüllt. Es müssen nur noch die Matrixeinträge konsistent gemacht werden. Dazu faßt man W als einen Vektor auf, macht diesen mittels MakeAdditiveVectorConsistent (Algorithmus 4.1) konsistent. In diesem Sinne kann der parallele Jacobiglätter als eine Funktion der Klasse Iter formuliert werden, indem man als näherungsweise Lösung in Schritt (1) verwendet:

(1) $\hat{t} := \omega\,\bar{W}^{-1}\,\hat{d}$ nach (4.24)

Für alle anderen Glätter muß man noch mehr ändern. Das Problem ist, daß in allen anderen Verfahren die approximierte Inverse W^{-1} nicht mehr als teilweise konistente Matrix darstellbar ist. Man behilft sich, indem man eine Kopie W der additiven Steifigkeitsmatrix $\hat{A}$ teilweise konsistent macht (MakeMatrixPartlyConsistent, Algorithmus 4.3) und dann nur auf jedem Prozessor lokal mit der durch das Glättungsverfahren bedingten Matrix $(\tilde{W}^p)^{-1}$ arbeitet; das unterschlägt nun in diesem Schritt die Kopplungen zwischen den Prozessoren und kann deshalb als globales Block–Jacobi–Verfahren betrachtet werden. Man hat keine Kontrolle, wie groß der dabei begangene Fehler ist. Der Vektor muß laut (4.26) dazu noch in eindeutiger Form verteilt sein. Damit läßt sich ein allgemeiner Glätter der Klasse Iter formulieren, indem in Schritt (1) die Lösung näherungsweise so berechnet wird:

(1) $\check{\boldsymbol{d}}$ **:= MakeAdditiveVectorUnique** ($\hat{\boldsymbol{d}}$)
$\hat{t} := (\tilde{W})^{-1}\,\check{d}$ nach (4.26)

Ein ganz anderer Zugang für Gauß–Seidel ähnliche Verfahren besteht darin, die Knotenmenge in bezüglich des Matrixgraphen vollkommen unzusammenhängende Teilmengen zu zerlegen. Bei strukturierten Gittern führt das zu Schachbrett–Numerierungen oder red–black Numerierungen [Fro90] [Haa99b], bei unseren unstrukturierten Gittern wäre das ein erheblicher Aufwand, der mehr als 2 Teilmengen erfordert. Die Knoten werden nach Farben getrennt bearbeitet. Innerhalb eines Schritts beeinflussen die Ergebnisse sich nicht gegenseitig; nach jedem Schritt werden die Ergebnisse

kommuniziert, insgesamt so oft, wieviele Farben benötigt wurden. Dieses Vorgehen erfordert daher auch in der Lösungsphase einen höheren Kommunikationsaufwand.

Bei all diesen Parallelisierungsstrategien gehen die Robustheitseigenschaften der seriellen Glätter weitgehend verloren [Bas96b]. Daher wird es wichtig sein, daß das parallele FAMG nicht unter diesem Mangel leidet.

4.2.8 Restriktion und Prolongation

Da in unserem Modell immer Eltern– und Kindelemente auf einem Prozessor gemeinsam vorhanden sind, kann die Prolongation streng lokal durchgeführt werden. Für die Prolongationsmatrizen gelten damit die Bedingungen (4.12a) und (4.12b). Die Einträge der Restriktions– und Prolongationsmatrizen R und P sind nach Abschnitt 2.3.3 die Interpolationsgewichte. Diese Gewichte sind allein aus den geometrischen Verhältnissen eines jeden Elements bestimmbar, also streng lokale Größen und unabhängig von der Nachbarschaft und dem Prozessor. Daher erfüllt P auch die Bedingung (4.12c). Daraus folgt, daß die Matrix P in konsistenter Darstellungsform $\bar{P}$ vorliegt. Damit kann (nach Gleichung (4.23)) die konsistente Lösungskorrektur prolongiert werden:

$$^{\ell}\bar{t} := {}^{\ell}\bar{P}\,{}^{\ell-1}\bar{c}.$$

Da hier immer $R := P^{\top}$ gelten soll, kann damit auch der additive Defekt nach Gleichung (4.25) restringiert werden:

$$^{\ell-1}\hat{d} := {}^{\ell}\bar{R}\,{}^{\ell}\hat{d}.$$

Auch hier passen die gewählten parallelen Darstellungsformen so gut zusammen, daß keine Kommunikation benötigt wird.

4.2.9 Grobgitterkorrektur

Durch die einheitliche Wahl der parallelen Darstellungsformen kann man den Mehrgitterlöser ohne irgendwelche Zusätze rekursiv aufrufen.

4.2.10 Baselevellöser

Um die numerische Güte des Gesamtverfahrens zu gewährleisten, muß auf dem gröbsten Level genau genug gelöst werden; der Defekt sollte um mehrere Größenordnungen reduziert werden. Selbst wenn auf dem gröbsten Level das Gleichungssystem so klein ist, daß man einen direkten Löser einsetzen kann, besteht dafür noch eine große Schwierigkeit: Die Matrix ist über alle Prozessoren verteilt und parallele, direkte Verfahren sind sehr ineffizient für große Prozessorzahlen (Abschnitt 2.3.5).

Man wird daher bestrebt sein, *alle* Masterknoten auf *einem* Prozessor, dem sogenannten Masterprozessor, zusammen zu ziehen; die bisherigen Masterknoten werden auf allen anderen Prozessoren zu Borderknoten. Formal bedeutet das eine Änderung der Abbildungen ${}^{\ell_{\min}}\alpha$ und ${}^{\ell_{\min}}\mu$. Eine sehr wichtige Feststellung ist, daß nach diesem Vorgang weiterhin *alle* Bedingungen an den Matrixgraphen und die Aufteilungsabbildung in der Definition 4.1.25 der parallelen Darstellungsform für Matrizen erfüllt sind. Dieser **Grobgitteragglomeration** (engl. **baselevel agglomeration**) genannte Prozeß kann einmalig im Vorbereitungsschritt durchgeführt werden. Auf dem Masterprozessor kann dann ein gewöhnlicher, serieller, direkter Löser verwendet werden. Natürlich muß in der Lösungsphase bei Erreichen des gröbsten Levels der Defekt von allen Prozessoren eingesammelt, auf dem Masterprozessor gelöst und das Ergebnis wieder an alle ausgeteilt werden. Wenn man einen direkten Löser einsetzen will, muß man immer alle Aspekte gemeinsam betrachten.

Die Alternative dazu besteht im Einsatz eines iterativen Verfahrens, das auf der verteilt vorliegenden Baselevel Matrix arbeitet: also ein Glätter eventuell mit Krylovbeschleunigung oder eben ein paralleles, algebraisches Mehrgitterverfahren. Gerade bei einem der üblichen Glättungsverfahren werden viele Iterationsschritte benötigt, um die geforderte Genauigkeit zu erzielen, und in jedem sind Kommunikationen enthalten (siehe Abschnitt 4.2.7), erhöht durch die Skalarproduktberechnungen im Krylovbeschleuniger. Als Verfahren eignen sich vorkonditionierte Krylovverfahren, wie z. B. Bi–CGSTAB mit Gauß–Seidel oder ILU Vorkonditionierer. Daher darf man erwarten, daß dieses Vorgehen erst für größere Baselevel Matrizen Vorteile gegenüber dem ersten Ansatz bringt.

Eine dritte Möglichkeit besteht darin, den gröbsten Level zu agglomerieren, dann aber ein optimales, iteratives Verfahren — also ein Mehrgitterverfahren — zu verwenden. Gegenüber dem ersten Ansatz sollte sich dieser Weg bei großen Baselevel Matrizen bewähren. Umgekehrt versucht man aber, durch ein paralleles Verfahren möglichst lange zu vergröbern, bevor man auf dem Baselevel die Parallelität aufgibt. Einen Kompromiß stellt eine schrittweise Verringerung des Parallelisierungsgrades nach jeweils einigen Gitterebenen dar (Zwischenagglomeration).

4.2.11 Gesamtverfahren

Nachdem nun alle Komponenten bereitgestellt sind, können sie zum Prototyp eines parallelen, geometrischen Mehrgitterverfahrens zusammengefügt werden. Bemerkenswert ist die geringe Zahl an Kommunikationsschritten, die durch die günstige Wahl der parallelen Darstellungsformen erzielt werden kann. In **Fettdruck** gesetzt sind die Änderungen gegenüber den seriellen Algorithmen aus Abschnitt 2.3.

Algorithmus 4.5 (LinearSolver($\bar{u}, \hat{d}$))

(1) $\check{\boldsymbol{d}}$ **:= MakeAdditiveVectorUnique(**$\hat{\boldsymbol{d}}$**)**
(2) $\delta := ||\check{d}||$ nach (4.21)
(3) $\epsilon := \max\{ \delta\epsilon_{rel}, \epsilon_{abs} \}$
(4) while $\delta > \epsilon$
(5) $\quad \bar{c} := 0$ nach (4.15)
(6) $\quad$ Iter$(\bar{c}, \check{d})$ $\check{d}$ ist ein spezielles $\hat{d}$, Def. 4.1.16
(7) $\quad \bar{u} := \bar{u} + \bar{c}$ nach (4.17)
(8) $\quad \check{\boldsymbol{d}}$ **:= MakeAdditiveVectorUnique(**$\hat{\boldsymbol{d}}$**)**
(9) $\quad \delta := ||\check{d}||$ nach (4.21)

Aufgerufen wird ein paralleler linearer Löser mit LinearSolver($\bar{0}$, $\hat{f}$).

Ein Glätter ist eine spezielle Funktion vom Typ Iter. Üblicherweise wird darin die Gleichung $A\hat{t} = \hat{d}$ nur sehr grob gelöst:

$$\hat{t} := \tilde{W}^{-1}\hat{d} \qquad \text{nach Algorithmus 2.2 Iter, Zeile (1).}$$

Auch das Mehrgitterverfahren ist ein spezielles Verfahren der Klasse Iter; vergleiche mit Algorithmus 2.3 MG.

Algorithmus 4.6 (pMG($\bar{c}$, $\hat{d}$, ℓ))

(1) if($\ell = \ell_{min}$)
(2) $\quad$ **BaseSolver(**${}^{\ell}\bar{\boldsymbol{c}}, {}^{\ell}\hat{\boldsymbol{d}}$**)**
else
(3) $\quad$ for $i := 1$ to ν_1: **Smooth(**${}^{\ell}\bar{\boldsymbol{c}}, {}^{\ell}\hat{\boldsymbol{d}}$**)**
(4) $\quad {}^{\ell-1}\hat{d} := {}^{\ell}\bar{P}^{\top}\, {}^{\ell}\hat{d}$ nach (4.25)
(5) $\quad {}^{\ell-1}\bar{c} := 0$ nach (4.15)
(6) $\quad$ for $i := 1$ to γ: **MG(**$\bar{\boldsymbol{c}}, \hat{\boldsymbol{d}}, \boldsymbol{\ell - 1}$**)**
(7) $\quad {}^{\ell}\bar{t} := {}^{\ell}\bar{P}\, {}^{\ell-1}\bar{c}$ nach (4.23)
(8) $\quad {}^{\ell}\hat{d} := {}^{\ell}\hat{d} - {}^{\ell}\hat{A}\, {}^{\ell}\bar{t}$ nach (4.27), (4.18)
(9) $\quad {}^{\ell}\bar{c} := {}^{\ell}\bar{c} + {}^{\ell}\bar{t}$ nach (4.17)
(10) $\quad$ for $i := 1$ to ν_2: **Smooth(**${}^{\ell}\bar{\boldsymbol{c}}, {}^{\ell}\hat{\boldsymbol{d}}$**)**

4.2.12 Krylovbeschleuniger

Krylovverfahren können sowohl als Beschleuniger für die Baselevel Iteration als auch zur Verbesserung der Konvergenz des gesamten Mehrgitterverfahrens eingesetzt werden. Sie sind als spezielle LinearSolver einzusetzen und verwenden als Vorkonditionierer einen Schritt eines iterativen Lösers von der Klasse Iter, z. B. einen Mehrgitterschritt MG. Ihre Parallelisierung beruht im wesentlichen auf der Parallelisierung der Skalarprodukte und ist damit ohne neue Konzepte möglich [Wie00b]. Für symmetrische, positiv definite Gleichungen verwendet man das klassische CG-Verfahren:

Algorithmus 4.7 (CG($\bar{u}, \hat{d}$))

(1) $\check{d} :=$ MakeAdditiveVectorUnique($\hat{d}$)
(2) $\delta := \|\check{d}\|$ nach (4.21)
(3) $\epsilon := \max\{\delta\, \epsilon_{rel}, \epsilon_{abs}\}$
(4) $\varrho_0 := 1$
(5) $\bar{y} := 0$ nach (4.15)
(6) while $\delta > \epsilon$
(7) $\quad \check{t} := \check{d}$
(8) $\quad \bar{c} := 0$ nach (4.15)
(9) $\quad$ Preconditioner($\bar{c}, \check{t}$)
(10) $\quad \varrho_1 := \bar{c}^\top\, \check{d}$ nach (4.20)
(11) $\quad \bar{y} := \bar{c} + \frac{\varrho_1}{\varrho_0}\, \bar{y}$ nach (4.17)
(12) $\quad \hat{s} := \hat{A}\, \bar{y}$ nach (4.27)
(13) $\quad \varrho_0 := \varrho_1$
(14) $\quad \beta := \frac{\varrho_0}{\bar{p}^\top\, \hat{s}}$ nach (4.20)
(15) $\quad \hat{d} := \check{d} - \beta\, \hat{s}$ $\check{d}$ ist ein spezielles $\hat{d}$, nach (4.27)
(16) $\quad \bar{u} := \bar{u} + \beta\, \bar{y}$

Für allgemeine Gleichungen kann man das Bi–CGSTAB–Verfahren nach [vdV92] verwenden. Es läßt sich analog parallelisieren.

5 Parallelisierung des FAMG

Dieses Kapitel bildet den Kern der vorliegenden Arbeit. Mit der in Abschnitt 4.1 eingeführten parallelen linearen Algebra wird nun die Parallelisierung des in Kapitel 3 vorgestellten filternden algebraischen Mehrgitterverfahrens FAMG entwickelt. Eine Vielzahl einzelner Aspekte muß geklärt werden, ehe der endgültige Algorithmus pFAMG formuliert werden kann.

Der Überlapp muß Mindestanforderungen erfüllen, die zu analysieren sein werden, und sie müssen realisiert werden. Die parallele lineare Algebra muß daraufhin erweitert werden. Die Verarbeitungsreihenfolge im Markierungsalgorithmus muß kritisch überdacht und ein völlig neues Vorgehen eingeführt werden. Ein paralleler Graphfärbungsalgorithmus mußte hierzu entwickelt werden. Schließlich muß die Galerkin–Assemblierung auf ihre Parallelisierbarkeit hin untersucht und ein praktikables Vorgehen entwickelt werden. Eine Diskussion über Möglichkeiten, das Gleichungssystem auf dem gröbsten Level zu lösen, schließt die Entwicklung ab.

5.1 Grundüberlegungen

Das hier zu entwickelnde Verfahren soll in das bestehende Programmpaket *UG* integriert werden, weil es so sowohl als Löser für schwere Problemklassen als auch als Baselevellöser für große Anwendungen mit geometrischem Mehrgitterverfahren verwendet werden kann und *UG* eine sehr leistungsfähige Plattform zur Realisierung paralleler Multilevelverfahren ist. Allerdings beruht dort die Datenaufteilung auf Elementen (Abschnitt 4.2.2), die wir im algebraischen Verfahren gar nicht mehr haben. Wir müssen mit einer knotenbasierten Datenaufteilung arbeiten. Glücklicherweise stellt die parallele lineare Algebra in Abschnitt 4.1 keine dahingehenden Anforderungen, sondern ist auf beide Modelle anwendbar. Da wir keine lokale Gitteradaption vornehmen werden und die Last nicht umverteilen wollen, benötigen wir — bis auf bei der Grobgitteragglomeration — keine Ghostknoten; daher wurde die parallele lineare Algebra allein mit Master– und Borderknoten vorgestellt. Im Gegensatz zum geometrischen Mehrgitter spielen hier die Borderknoten eine stärkere Rolle, dienen aber weiterhin nur als Hilfsknoten für eine effiziente Berechnung. Daher kann der Übergang zwischen geometrischen und algebraischen Leveln nahtlos erfolgen.

Nach den Erfahrungen von P. Bastian in [Bas96b] lohnt sich der Einsatz eines additiven Mehrgitterverfahrens gegenüber einem multiplikativen nicht, weil die bessere parallele Effizienz durch die schlechtere numerische Güte überkompensiert wird. Wir betrachten daher nur die multiplikative Variante.

Es soll ein tragfähiges Gerüst für die Realisierung paralleler AMG nach der Selektionsmethode geschaffen werden. In diesen Rahmen sollten sich die Komponenten des FAMG aus Kapitel 3 gut integrieren lassen, ohne daß sie neu erstellt werden müssen. Das wird für das Aufstellen der Elternpaarlisten ideal möglich sein. Während der Markierungsablauf entscheidend umgestaltet werden muß und vom Rahmenprogramm gesteuert wird; das serielle Verfahren muß dazu feingranularere Funktionalitäten bereitstellen, als dies für das serielle Verfahren nötig war.

5.2 Voraussetzungen

Hier sollen die für die Anwendung des Verfahrens benötigten Voraussetzungen zusammengestellt werden. Zuerst wird jedoch noch der Begriff der Struktursymmetrie für eine Matrix eingeführt.

Definition 5.2.1 (Matrix struktursymmetrisch)
Eine Matrix $A \in \mathbb{R}^{n \times n'}$ heißt **struktursymmetrisch**, wenn

$$\forall i \in J \ \forall j \in J' : \quad A_{ij} \neq 0 \Leftrightarrow A_{ji} \neq 0. \tag{5.1}$$

Lemma 5.2.2 (Matrix symmetrisch $\Rightarrow$ struktursymmetrisch)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n \times n}$. Wenn sie symmetrisch ist, so ist sie auch struktursymmetrisch.

Beweis: Wenn $\forall i, j \in J$ gilt $A_{ij} = A_{ji}$ folgt sofort $A_{ij} \neq 0 \Leftrightarrow A_{ji} \neq 0$. □

Bemerkung 5.2.3 (Struktursymmetrie)
Die Struktursymmetrie einer Matrix ist also eine schwächere Eigenschaft als die Symmetrie. Wenn ein Matrixeintrag A_{ij} vorhanden ist, muß der transponierte Eintrag A_{ji} nur auch vorhanden sein, darf aber einen anderen Wert annehmen.

Die Matrix liege bereits verteilt vor. Dies annähernd optimal und unter den zu beachtenden Nebenbedingungen zu erreichen, ist eine schwierige Aufgabe [HL93] [Lan93] [KK95] [Bas96b], die wir von *UG* durchführen lassen; wir verwenden die so erzeugte Matrix. Folgende Bedingungen müssen dabei eingehalten werden. Das sind aber größtenteils natürliche Forderungen und leicht zu erfüllen:

$$\alpha \text{ respektiert Matrixgraph } \mathcal{G}(A) \tag{5.2}$$

$$A \text{ ist struktursymmetrisch} \tag{5.3}$$

$$A \text{ liegt additiv verteilt vor} \tag{5.4}$$

5.3 Überlapp

Alle Operationen zur Herstellung der Mehrgitterhierarchie und des Lösers arbeiten primär auf den Masterknoten und bei jedem Knoten auf einer gewissen Nachbarschaft. Damit das auch für Masterknoten, die im Bereich des Prozessorrandes liegen, möglich ist, müssen an diesen Rändern zusätzliche Knotenkopien angelegt werden, die den sogenannten **Überlapp** bilden. Da sich der gesamte Vorgang an der Nachbarschaft bezüglich der Steifigkeitsmatrix orientieren wird, soll sie nicht immer explizit erwähnt werden. Zur Strukturierung des Überlapps werden einige Begriffe eingeführt.

Die Anforderungen an die Größe des Überlapps rühren von den darauf arbeitenden Operationen her. Sie werden zwar erst im weiteren Verlauf des Kapitels behandelt; hier sollen aber schon die die Überlappgröße bestimmenden Ergebnisse zusammengestellt werden. Das entspricht auch dem algorithmischen Ablauf: Zuerst sollte der Überlapp und damit die Matrix erzeugt sein, bevor darauf gearbeitet wird; andernfalls müßte vor jeder Operation jedesmal die Voraussetzungen an den Matrixgraphen überprüft und er gegebenenfalls erweitert werden.

Wenn dann die Anforderungen an den Überlapp definiert sind, muß ein Algorithmus entwickelt werden, der die benötigte Erweiterung des Überlapps herstellt. Wichtig dabei wird sein, daß alle formulierten Bedingungen *lokal* entscheidbar sind, um Kommunikation zu vermeiden. Die auftretenden Verteilungssituationen können sehr komplex werden. Um auch dann korrekte Ergebnisse zu erhalten, sind Beweise der Korrektheit nötig, da es sich mehrfach im Laufe der Entwicklungsarbeit gezeigt hat, daß ein Vorgehen, das aus dem Gefühl und der Anschauung einfacher Situationen heraus entstanden ist, für komplexere Fälle versagte. Die Bedeutung der Beweise geht hier also weit über das rein akademische Interesse hinaus.

5.3.1 Begriffe und Eigenschaften

Um verschiedene Teile des Überlapps unterscheiden zu können, führen wir einige Begriffe ein und klären wichtige Eigenschaften von ihnen.

Definition 5.3.1 (Teile des Überlapps)
Im folgenden sei immer $p \in \mathcal{P}$.
Die Menge der Masterknoten $\mathcal{M}^p$ soll hier auch als **Kernpartition** bezeichnet werden. Der Vollständigkeit halber kann man sie auch Überlapp der Tiefe 0 (${}_0\mathcal{U}^p$) nennen.
Die Masterknoten des **Prozessorinterfaces** (oder **Prozessorrandes**)

$$\mathcal{I}^p := \{\, i \in \mathcal{M}^p \mid \mathcal{N}^p(i) \cap \mathcal{B}^p \neq \emptyset \,\} \tag{5.5}$$

bilden die Abgrenzung der Kernpartition zu den Nachbarprozessoren.
Alle Borderknoten, die in der Steifigkeitsmatrix eine direkte Verbindung zu einem Masterknoten besitzen, bilden den **Randbereich der Tiefe 1**

$${}_1\mathcal{R}^p := \{\, i \in \mathcal{B}^p \mid \mathcal{N}^p(i) \cap \mathcal{M}^p \neq \emptyset \,\} \tag{5.6}$$

und zusammen mit den Masterknoten den **Überlapp der Tiefe 1**

$$ {}_1\mathcal{U}^p := \mathcal{M}^p \,\dot\cup\, {}_1\mathcal{R}^p, \tag{5.7} $$

alle Borderknoten, die zu ${}_1\mathcal{U}^p$ eine direkte Verbindung haben, bilden den **Randbereich der Tiefe 2**

$$ {}_2\mathcal{R}^p := \{\, i \in \mathcal{B}^p \setminus {}_1\mathcal{R}^p \mid \mathcal{N}^p(i) \cap {}_1\mathcal{U}^p \neq \emptyset \,\} \tag{5.8} $$

und zusammen mit dem ${}_1\mathcal{U}$ den **Überlapp der Tiefe 2**

$$ {}_2\mathcal{U}^p := {}_1\mathcal{U}^p \,\dot\cup\, {}_2\mathcal{R}^p. \tag{5.9} $$

Analog ließen sich weitere Randbereiche in fortschreitender Tiefe definieren.

Alle Knoten, die noch eine weitere Instanz besitzen, bilden das **Interface**. Die Anzahl dieser Knoten mit ihrer Vielfachheit gezählt

$$ \sum_{i \in \{i \in J^p:\ |\alpha(i)|>1\}} |\alpha(i)| $$

ergibt die **Interfacelänge**, die eine wichtige Kennzahl für den Zeit– und Speicherbedarf einer Kommunikation darstellt.

Definition 5.3.2 (Vollständigkeit eines Überlapps)
Der Überlapp der Tiefe t heiße **vollständig**, wenn

$$ \forall p \in \mathcal{P}\ \ \forall i \in {}_{t-1}\mathcal{U}^p\ \ \forall q \in \alpha(i)\ \ \forall j \in \mathcal{N}^q(i): \quad j \in {}_t\mathcal{U}^p \ \ \wedge\ \ j \in \mathcal{N}^p(i). \tag{5.10} $$

Bemerkung 5.3.3 (Vollständigkeit eines Überlapps)
Die Vollständigkeit eines Überlapps erfordert also gewisse Knotenkopien — oder anders ausgedrückt, eine gewisse Größe der Bilder $\alpha(i)\ \forall i \in J$ (in vorstehender Definition die Bedingung $j \in {}_t\mathcal{U}^p$) — und darauf eine Vollständigkeit der Struktur der Steifigkeitsmatrix (Bedingung $j \in \mathcal{N}^p(i)$): Für den Knoten i muß die *gesamte globale Matrixzeile* auf Prozessor p vorhanden sein.

Abbildung 5.1 veranschaulicht die eingeführten Begriffe anhand eines einfachen Beispiels eines zweidimensionalen Gebiets, das auf zwei Prozessoren verteilt wird. Dabei wird deutlich, wie die Randstreifen zunehmender Tiefe die Kernpartition schichtweise einhüllen. Zur besseren Visualisierung wird der Matrixgraph in Form des ursprünglichen, zweidimensionalen Gitters dargestellt: Die Knoten werden (ihren ehemaligen geometrischen Positionen entsprechend) im Grundgebiet plaziert und Kanten dazwischen stellen ein Paar von Matrixeinträgen A_{ij} und A_{ji} dar.

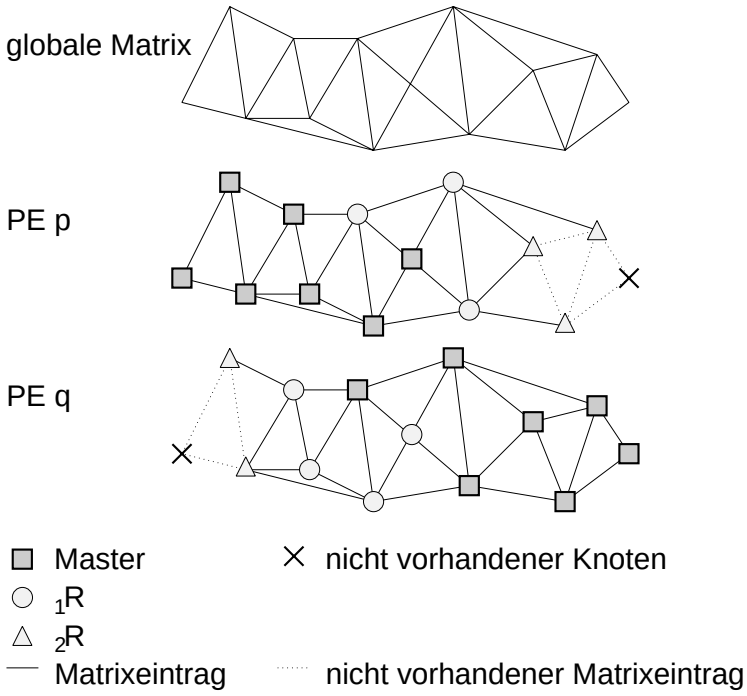


Abbildung 5.1: Darstellung eines Überlapps der Tiefe 2 zwischen 2 Prozessoren

5.3.2 Erweiterung der parallelen linearen Algebra

Die Konsistenz der parallelen Darstellungsform soll in geeigneter Weise auf den Überlapp der Tiefe t eingeschränkt werden.

Definition 5.3.4 (t–konsistent)
Die Einschränkung der Aufteilungsabbildung α auf den Überlapp der Tiefe t sei definiert als

$$ {}_t\psi(j) := \{p \in \mathcal{P} \mid j \in {}_t\mathcal{U}^p\}, \tag{5.11} $$

ψ sei die Abkürzung von ${}_1\psi$.
Ein Vektor $x \in \mathbb{R}^n$ ist in **t–konsistenter** Form ${}_t\ddot{\ddot{x}} = ({}_t\ddot{\ddot{x}}^p)_{p\in\mathcal{P}}$ verteilt, wenn

$$ \forall j \in J \ \forall p \in {}_t\psi(j) : {}_t\ddot{\ddot{x}}_j = x_j, \tag{5.12} $$

in Kurzschreibweise

$$ \ddot{\ddot{x}} \mid_{{}_t\mathcal{U}} = x. $$

Eine Matrix $A \in \mathbb{R}^{n \times n'}$ ist in **t–konsistenter** Form ${}_t\ddot{\ddot{A}} = ({}_t\ddot{\ddot{A}}^p)_{p \in \mathcal{P}}$ verteilt, wenn

$$\begin{aligned}
&\text{die eingeschränkte Aufteilungsabbildung } {}_t\psi \text{ und } \alpha' \\
&\text{den Matrixgraph } \mathcal{G}(A) \text{ respektieren,} && (5.13a)\\
&\text{für } {}_t\psi, \alpha' \text{ gilt: } \forall i \in J\ \forall j \in \mathcal{N}_A(i) :\ {}_t\psi(i) \subseteq \alpha'(j) \text{ und} && (5.13b)\\
&\ddot{\ddot{A}}^p_{ij} = A_{ij} \quad \forall i \in J,\ \forall j \in \mathcal{N}_A(i)\ \forall p \in {}_t\psi(i). && (5.13c)
\end{aligned}$$

Lemma 5.3.5 (t–konsistent ⇒ s–konsistent $\forall$s < t)
Ist ein Vektor oder eine Matrix t–konsistent, so auch s-konsistent $\forall s < t$.

Beweis: An ${}_s\psi$ werden geringere Anforderungen gestellt als an ${}_t\psi$ und die Respektion des Matrixgraphen bleibt durch die Vollständigkeit des Überlapps gesichert. □

Lemma 5.3.6 (${}_t\mathcal{U}$ vollständig ⇒ ${}_{t-1}\ddot{\ddot{A}}$)
Der Überlapp der Tiefe t sei vollständig. Dann ist eine nach Definition 4.1.25 teilweise konsistente Matrix $\bar{A}$ $(t-1)$–konsistent.

Beweis: (5.13a) ist schon durch (4.13a) sichergestellt und (5.13b) durch die Vollständigkeit des ${}_t\mathcal{U}$ (5.10). Weiterhin sichert die Vollständigkeit des ${}_t\mathcal{U}$

$$\forall i \in {}_{t-1}\mathcal{U}\ \forall j \in \mathcal{N}_A(i) : \alpha(i) \subseteq \alpha'(j).$$

Daher gilt

$$\forall p \in {}_t\psi(i) : p \in \alpha(i) \cap \alpha'(j)$$

und (4.13c) liefert (5.13c). □

Lemma 5.3.7 (${}_1\mathcal{U}$ vollständig ⇒ α läßt teilweise konsistente Matrix zu)
Wenn der Überlapp der Tiefe 1 vollständig ist, erfüllt die damit verbundene Aufteilungsabbildung α auch beide Bedingungen (4.13a) und (4.13b), die für die Existenz einer teilweise konsistenten Matrix $\bar{W}$ nötig sind.

Beweis: Daß die gewählte Datenaufteilung den Graph der Steifigkeitsmatrix respektiert, wurde bereits in Abschnitt 4.2.5 nachgewiesen; das ist genau die Bedingung (4.13a).
Die Vollständigkeit des ${}_1\mathcal{U}$ bedeutet nach (5.10):

$$\forall p \in \mathcal{P}\ \forall j \in \mathcal{M}^p\ \forall q \in \alpha(j)\ \forall i \in \mathcal{N}^q(j) : i \in \mathcal{N}^p(j),$$

d. h. $\forall i, j \in J$: W_{ji} ist auf Prozessor $\mu(j)$ vorhanden
$\Rightarrow \mu(j) \in \alpha(i)$.
Da W nach Voraussetzung (5.3) struktursymmetrisch ist, gilt diese Aussage auch für W_{ij}, womit (4.13b) bewiesen ist. □

Um einen t–konsistenten Vektor auf die allgemein übliche konsistente Darstellungsform zu transformieren, müssen die Knoten in ${}_t\mathcal{U}$ ihre Werte an die restlichen Borderknoten schicken. Da aber die Werte in ${}_t\mathcal{U} \setminus \mathcal{M}$ identisch mit denen in $\mathcal{M}$ sind, kann auf schon bestehende Kommunikationsstrukturen zurückgegriffen werden und deshalb können einfach die Masterwerte an *alle* Borderknoten geschicken werden. Für $t = 0$ sind beide Varianten identisch.

Algorithmus 5.1 (MaketconsistentVectorConsistent($\ddot{x}$))
$Q^p := \{ q \in \mathcal{P} \setminus \{p\} \mid \mathcal{B}^q \cap \mathcal{M}^p \neq \emptyset \}$
(1) Gather: $\forall q \in Q^p \ \forall j \in \mathcal{B}^q \cap \mathcal{M}^p$: kopiere $\ddot{x}_j^p$ in Nachrichtenpuffer $M_{p,q}$
(2) Datenaustausch: $\forall q \in Q^p$: sende $M_{p,q}$ an q und empfange $M_{q,p}$ von q
(3) Scatter: $\forall q \in Q^p \ \forall j \in \mathcal{B}^p \cap \mathcal{M}^q$: hole $\ddot{x}_j^q$ aus $M_{q,p}$ und setze

$$\ddot{\ddot{x}}_j^p := \ddot{x}_j^q$$

Am Ende der Operation steht in $\ddot{\ddot{x}}^p$ der konsistent verteilte Vektor und muß folglich als $\bar{x}^p$ bezeichnet werden.

Beweis: In einem konsistenten Vektor muß nach (4.7) zu allen Komponenten der globale Wert x_j stehen. Für alle Masterkomponenten $\ddot{x}_j^p$ gilt das bereits nach (5.12). Folglich benötigen nur noch die Borderkomponenten ihren richtigen Wert. Im Gatherschritt schickt dazu jeder Master seinen Wert an alle seine Borderkopien, die umgekehrt im Scatterschritt diesen Wert bei sich eintragen. □

Damit lassen sich folgende Matrix–Vektor–Multiplikationen definieren:

Sei $\ddot{\ddot{A}} \in \mathbb{R}^{n \times n'}$ t–konsistent und $x \in \mathbb{R}^n, w \in \mathbb{R}^{n'}$.

$$ {}_t\ddot{x} := {}_t\ddot{\ddot{A}}\bar{w} \tag{5.14}$$

$$ \hat{w} := {}_t\ddot{\ddot{A}}^\top \check{x} \tag{5.15}$$

Beweis:

a) ${}_t\ddot{\ddot{x}} := {}_t\ddot{\ddot{A}}\bar{w}$
Wenn man den Beweis zu (4.23) ($\bar{x} := \bar{A}\bar{w}$) auf $p \in {}_t\psi(i)$ einschränkt statt $p \in \alpha(i)$, wird genau die zu zeigende Aussage bewiesen: Da ${}_t\ddot{\ddot{A}} \ \forall i \in {}_t\mathcal{U}^p$ die vollständige und konsistente Matrixzeile enthält und $\bar{w}$ ebenfalls global konsistent ist, kann auch das konsistente Ergebnis auf ${}_t\mathcal{U}^p$ berechnet werden. Die restlichen Borderknoten tragen im Ergebnis undefinierte Werte.

b) $\hat{w} := {}_t\ddot{\ddot{A}}^\top \check{x}$
Man kann im Beweis zu (4.25) ($\hat{w} := \bar{A}^\top \hat{x}$) statt $p \in \alpha(i)$ nur $p = \mu(i) \subseteq {}_t\psi(i)$

einsetzen, da $\check{x}_j^p = 0 \ \forall p \in \alpha(i) \setminus \mu(i)$ (Eigenschaft des eindeutig verteilten Vektors (4.9)); damit ist nach (4.8)

$$\sum_{p \in \alpha(i)} \hat{x}_i^p = \check{x}_i^{\mu(p)}$$

und der angegebene Beweis sichert auch die hier getroffene Aussage.

□

Bemerkung 5.3.8 (nur Knoten in ${}_t\mathcal{U}$ auf feinem Level beachten)
Da das Ergebnis nur auf den Knoten in ${}_t\mathcal{U}$ benötigt wird, muß nicht die vollständige Matrixmultiplikation durchgeführt werden. Analoges gilt für die transponierte Matrix.

5.3.3 Anforderungsanalyse an den Überlapp

Hier sollen alle Anforderungen, die nachfolgende Algorithmusteile an die Größe des Überlapps stellen, zusammengefaßt werden. Besonders ist auf Lemma 5.3.6 hinzuweisen, da dort eine gewisse Überlappgröße die Konsistenz einer Matrix in gewissen Teilen des Überlapps sichert. Die hier verwendete Tiefe darf nicht mit der elementweisen Überlapptiefe für geometrische Mehrgitterverfahren verwechselt werden.

Operation	Kap.	Erläuterung	Tiefe
Aufteilungsabbildung respektiert den Matrixgraph	4.2.5	Bei der beschriebenen Finiten Element Diskretisierung ist die Bedingung schon erfüllt	< 1
Elternliste erstellen	5.4	Glätter benötigt konsistente Matrix für $\mathcal{M}^p$ und $\mathcal{N}^p(\mathcal{M}^p)$	2
F-/C-Markierung	5.5	Entscheidung F/C in $\mathcal{M}^p$ involviert nur direkte Nachbarn $\mathcal{N}^p(\mathcal{M}^p)$	1
Update nach Markierung	5.5	Das Neubewerten der Eltern betrifft die gleichen Knoten wie das ursprüngliche Aufstellen der Elternlisten	2
Graphfärbung	5.6	äquivalente, aber lokal entscheidbare Bedingung, ob 2 Prozessoren im Graph verbunden sind	$\boxed{2}$
Prolongation und Restriktion	5.7	benötigen nur $\mathcal{N}^p(\mathcal{M}^p)$, aber die Galerkinassemblierung benötigt ${}_1\ddot{P}$	2
Galerkin–Assemblierung	5.8	trickreiches Vorgehen; ansonsten wäre sogar Tiefe 3 erforderlich	$\boxed{2}$
Löser	4.2	rechnet nur auf $\mathcal{M}^p$. Nur für Matrixmultiplikationen wird noch Verbindung zu direkten Nachbarn $\mathcal{N}^p(\mathcal{M}^p)$ benötigt	1

Die maximal benötigte Überlapptiefe beträgt folglich 2. Eine eingerahmte Ziffer $\boxed{2}$ besagt, daß diese Forderung zwingend ist; andere Tiefe 2 Anforderungen könnten mit wahrscheinlich nur kleinen Qualitätseinbußen auf Tiefe 1 abgemildert werden. Dieses Ergebnis ist zwar schmerzlich, aber noch vertretbar. Gerade auf gröberen Leveln und vielen Prozessoren wird der Überlapp durch die Tiefe 2 einen beträchtlichen Teil der gesamten Knoten darstellen und damit erhöhte Kommunikationskosten durch größere Nachrichtenlängen und einen dichteren Prozessor–Nachbarschaftsgraphen erzeugen. Wenn schon der große Überlapp nötig ist, ist es andererseits aber erfreulich, daß er an mehreren Stellen gewinnbringend eingesetzt werden kann.

5.3.4 Algorithmus zum Herstellen des benötigten Überlapps

Zum Herstellen des benötigten Überlapps der Tiefe 2 wird in zwei Schritten vorgegangen: Zunächst wird der vollständige Überlapp der Tiefe 1 hergestellt. Darauf aufbauend wird der Randbereich der Tiefe 2 vervollständigt. Die Vergrößerung muß dabei vollkommen *verteilt* geschehen, ohne eine übergeordnete Kontrolle und ohne das Wissen um den globalen Matrixgraph, damit der Algorithmus mit der Anzahl der Prozessoren skalierbar bleibt. Dabei darf die Additivität der Steifigkeitsmatrix nicht verloren gehen.

Das Vorgehen wird sein, daß für einzelne Knoten i die zugehörigen lokalen Matrixeinträge $A_{ij} \ \forall j \in \mathcal{N}^p(i)$ auf einem Prozessor p in geeignete Nachrichten verpackt und an einen Nachbarprozessor q verschickt werden. Wichtig ist, daß der Knoten i auf q schon vorhanden ist, da sonst nicht zu bestimmen ist, wohin die Informationen gehören. Das begründet auch das inkrementelle Vorgehen: Man kann immer nur 1 Schicht weiterbauen, weil man an bestehende Knoten anknüpfen muß. Auf q werden dann die Nachrichten den Knoten i zugeordnet, für die Matrixeinträge A_{ij}, für die der Knoten j noch nicht vorhanden ist, muß j erzeugt werden; dann wird auch der Matrixeintrag selbst erzeugt und der Wert A_{ij} gesetzt bzw. auf den bereits bestehenden aufaddiert. Da all dies sehr implementierungsspezifisch ist, soll hier nicht weiter darauf eingegangen werden. Das Konsistenthalten der verteilten Datenstruktur ist dabei eine sehr komplexe Aufgabe, weil über die neuentstandenen Knoten neben den unmittelbar beteiligten Prozessoren p und q auch unbeteiligte dritte Prozessoren informiert werden müssen, so daß auch sie weiterhin korrekte Abbildungen $\alpha(j)$ besitzen. Dieser Mechanismus wird in [Bir98, Transfermodul] beschrieben und durch die verwendete Bibliothek DDD bereitgestellt.

5.3.4.1 Erzeugung des vollständigen Überlapps der Tiefe 1

Aufgabe ist also zunächst, die Vollständigkeit des Randbereichs der Tiefe 1 ${}_1\mathcal{R}$ auf jedem Prozessor herzustellen. Nach (5.10) müssen dazu alle (Border)Nachbarn aller Masterknoten bezüglich der *globalen* Steifigkeitsmatrix vorhanden sein. Fehlende

Nachbarn müssen als Borderkopien erzeugt werden. Das Problem ist, daß ein Masterknoten i auf einem Prozessor p *nicht* das *globale* Wissen um all seine Nachbarn j hat. Folglich muß die Zuständigkeit anders geregelt werden: Jeder Borderknoten i auf Prozessor $q \neq p$ muß seinem Master all seine Nachbarn schicken; so wird beim Master letztendlich die globale Information aufgesammelt. Ein Beweis wird klären müssen, ob die Vollständigkeit des ${}_1\mathcal{U}$ damit gesichert ist.

Algorithmus 5.2 (CompleteU1(A))
Voraussetzung: α respektiert $\mathcal{G}(A)$ und A ist struktursymmetrisch.
(1) Gather: $\forall i \in \mathcal{B}^p \quad \forall j \in \mathcal{N}^p(i)$: halbiere Werte der lokalen A^p_{ij} und A^p_{ji} und stelle sie dann in Nachrichtenpuffer $M_{p,\mu(i)}$
(2) Datenaustausch: sende alle $M_{p,q}$ und empfange alle $M_{q,p}$
(3) Scatter: empfange alle Nachrichten $M_{q,p}$ und
$\forall i \in \mathcal{M}^p \ \forall q \in \alpha(i) \setminus \{p\} \ \forall A^q_{ij}, A^q_{ji} \in M_{q,p}$: erzeuge j falls nicht vorhanden und setze Werte $A^p_{ij} := A^q_{ij}, A^p_{ji} := A^q_{ji}$, ansonsten addiere A^q_{ij}, A^q_{ji} auf bestehende Werte.

Bemerkung 5.3.9
In der Gather Operation wird der Wert des Matrixeintrags halbiert; die eine Hälfte verbleibt im lokalen Speicher, die andere wird verschickt. So bleibt die Additivität der Steifigkeitsmatrix erhalten und der Matrixgraph kann dennoch vergrößert werden. Sollte das Datenmodell auch Matrixeinträge mit Wert 0 zulassen (wie es im hier verwendeten *UG* der Fall ist), kann auf das Halbieren verzichtet werden, und statt dessen ein 0-Eintrag verschickt werden.
Hier ist ein Kommunikationsmechanismus nötig, bei dem der Empfänger a priori nicht weiß, von welchen Prozessoren er Nachrichten bekommen wird.

Satz 5.3.10 (Korrektheit des Algorithmus CompleteU1)
Nach Ausführen des Algorithmus CompleteU1 ist die Steifigkeitsmatrix weiterhin struktursymmetrisch und additiv und der Überlapp der Tiefe 1 ist vollständig.

Beweis:

a) Da alle Einträge paarweise erzeugt werden (A_{ij} und A_{ji}), bleibt die Struktursymmetrie (5.1) erhalten.

b) Da ein Eintrag im Gatherschritt halbiert wird, und eine Hälfte lokal verbleibt und die andere verschickt und im Scatterschritt aufaddiert wird, bleibt die globale Summe der Matrixeinträge unverändert erhalten, Bedingung (4.13c) gilt also weiterhin. Da die Aufteilungsabbildung α nur erweitert wird, gelten auch (4.13a) und (4.13b) weiterhin.

c) Widerspruchsannahme: ${}_1\mathcal{U}$ sei nicht vollständig, d. h.

$$\exists p \in \mathcal{P}\ \exists i \in \mathcal{M}^p\ \exists q \in \alpha(i)\ \exists j \in \mathcal{N}^q(i): \tag{5.16a}$$
$$j \notin {}_1\mathcal{U}^p \quad \vee \tag{5.16b}$$
$$j \notin \mathcal{N}^p(i) \tag{5.16c}$$

Fall 1: $q = p$

$j \in \mathcal{N}^p(i)$ nach (5.16a). Damit kann (5.16c) nicht zutreffen.

$\Longrightarrow \exists j \in \mathcal{N}^p(i) : j \notin {}_1\mathcal{U}^p$

Für $j \in J^p$ gilt nach (4.4) genau einer der beiden Fälle:

1.1: $j \in \mathcal{M}^p$

$\overset{(5.7)}{\Longrightarrow} j \in {}_1\mathcal{U}^p$. Widerspruch zu (5.16b).

1.2: $j \in \mathcal{B}^p$

$j \in \mathcal{N}^p(i)$ nach Voraussetzung $\overset{(4.11)}{\Longrightarrow} i \in \mathcal{N}^p(j)$ und nach Voraussetzung (5.16a) ist $i \in \mathcal{M}^p$

$\Longrightarrow i \in \mathcal{N}^p(j) \cap \mathcal{M}^p$, d. h. $\mathcal{N}^p(j) \cap \mathcal{M}^p \neq \emptyset \overset{(5.6)}{\Longrightarrow} j \in {}_1\mathcal{R}^p$. Widerspruch zur Annahme (5.16b).

Damit kann nur zutreffen:

Fall 2: $q \neq p$

Nach Voraussetzung (5.16a) ist $i \in \mathcal{M}^p$ und $i \in J^q$. Die Eindeutigkeit der Master nach Lemma 4.1.7 impliziert $i \in \mathcal{B}^q$.

Weiterhin gilt nach Voraussetzung (5.16a) $j \in \mathcal{N}^q(i)$. Damit wurden im Algorithmusschritt (1) die zugehörigen Werte A_{ij} und A_{ji} an Prozessor p geschickt und im Schritt (3) auf p in die Matrix A^p eingetragen.

$\overset{(4.10)}{\Longrightarrow} j \in \mathcal{N}^p(i)$. Widerspruch zu (5.16c).

$\overset{(4.11)}{\Longrightarrow} i \in \mathcal{N}^p(j)$ und $i \in \mathcal{M}^p$ nach Voraussetzung (5.16a)

$\Longrightarrow i \in \mathcal{N}^p(j) \cap \mathcal{M}^p$; d. h. $\mathcal{N}^p(j) \cap \mathcal{M}^p \neq \emptyset$.

$\overset{(5.6)}{\Longrightarrow} j \in {}_1\mathcal{U}^p$. Widerspruch zu (5.16b).

Die Annahme war also falsch und damit ist die Aussage bewiesen.

□

5.3.4.2 Erzeugung des vollständigen Überlapps der Tiefe 2

Nachdem der Überlapp der Tiefe 1 vollständig vorhanden ist, kann im zweiten Schritt der Randbereich ${}_2\mathcal{R}$ erzeugt werden. Wieder müssen fehlende Borderknoten erzeugt werden, diesmal um Bedingung 5.10 für ${}_2\mathcal{U}$ zu erfüllen. Bedarf danach haben diejenigen Borderknoten, die in ${}_1\mathcal{U}^q$ liegen; sie brauchen diejenigen Nachbarn, die eigentlich in ${}_2\mathcal{R}^q$ liegen sollten. Auch hier haben sie selbst nicht die Möglichkeit, das Fehlen

festzustellen. In die Bresche springen kann jedoch ihr Masterknoten: Er hat bereits *alle* Nachbarn und kann diese Nachbarschaft an seine Borderinstanzen in ${}_1\mathcal{U}^q$ verschicken. Erst ein Beweis wird sicherstellen, daß damit die Vollständigkeit des ${}_2\mathcal{U}$ hergestellt ist.

Algorithmus 5.3 (CompleteU2(A))
Voraussetzung: ${}_1\mathcal{U}$ ist vollständig.

(1) Gather: $\forall i \in \mathcal{M}^p \ \forall q \in \{q \in \alpha(i) \setminus \{p\} \mid \ \exists j \in \mathcal{N}^p(i) : q = \mu(j)\}$ mache: $\forall k \in \mathcal{N}^p(i)$: halbiere Wert der lokalen A^p_{ik} und A^p_{ki} und stelle sie dann in Nachrichtenpuffer $M_{p,q}$

(2) Datenaustausch: sende alle $M_{p,q}$ und empfange alle $M_{q,p}$

(3) Scatter: empfange alle Nachrichten $M_{q,p}$ und $\forall i \in {}_1\mathcal{R}^p, q := \mu(i) \ \forall A^q_{ij}, A^q_{ji} \in M_{q,p}$: erzeuge j falls nicht vorhanden und setze Werte $A^p_{ij} := A^q_{ij}, A^p_{ji} := A^q_{ji}$, ansonsten addiere A^q_{ij}, A^q_{ji} auf bestehende Werte.

Bemerkung 5.3.11 (Ausführbarkeit)
Kritisch ist das Feststellen auf Prozessor p, ob eine Masterinstanz von i auf einem anderen Prozessor $q \in \alpha(i) \setminus \{p\}$ genau in ${}_1\mathcal{R}^q$ liegt. Das geht aber, da

$$\begin{aligned}
i \in {}_1\mathcal{R}^q \quad &\overset{(5.6)}{\Longleftrightarrow} \quad i \in \mathcal{B}^q \wedge \mathcal{N}^q(i) \cap \mathcal{M}^q \neq \emptyset \\
&\overset{(4.4)}{\Longleftrightarrow} \quad q \in \alpha(i) \setminus \{p\} \wedge \mathcal{N}^q(i) \cap \mathcal{M}^q \neq \emptyset \\
&\overset{(5.10) \text{ für } {}_1\mathcal{U}^p}{\Longleftrightarrow} \quad q \in \alpha(i) \setminus \{p\} \wedge \mathcal{N}^p(i) \cap \mathcal{M}^q \neq \emptyset \\
&\Longleftrightarrow \quad q \in \alpha(i) \setminus \{p\} \wedge \ \exists j \in \mathcal{N}^p(i) \mid \mu(j) = q.
\end{aligned}$$

Das alles sind Eigenschaften, die mit Hilfe der lokal auf Prozessor p vorhandenen Informationen feststellbar sind und in Schritt (1) verwendet werden können. Der letzte Schritt ist auch derjenige, der sich *nicht* auf Überlapptiefen größer 2 übertragen läßt!

Satz 5.3.12 (Korrektheit des Algorithmus CompleteU2)
Nach Ausführen des Algorithmus CompleteU2 ist die Steifigkeitsmatrix weiterhin struktursymmetrisch und der Überlapp der Tiefe 2 ist vollständig.

Beweis:

a) und b) können unverändert aus dem Beweis zu Satz 5.3.10 übernommen werden.

c) Widerspruchsannahme: ${}_2\mathcal{U}$ sei nicht vollständig, d. h.

$$\exists p \in \mathcal{P} \ \exists i \in {}_1\mathcal{U}^p \ \exists q \in \alpha(i) \ \exists j \in \mathcal{N}^q(i) : \tag{5.17a}$$
$$j \notin {}_2\mathcal{U}^p \quad \vee \tag{5.17b}$$
$$j \notin \mathcal{N}^p(i) \tag{5.17c}$$

Für Prozessor $r := \mu(i)$ gilt wegen der Vollständigkeit des ${}_1\mathcal{U}$ (nach Voraussetzung (5.10))

$$j \in {}_1\mathcal{U}^r \wedge j \in \mathcal{N}^r(i). \tag{5.18}$$

Für die Annahme $i \in {}_1\mathcal{U}^p$ trifft genau einer der beiden Fälle zu (nach (5.7)):

Fall 1: $i \in \mathcal{M}^p$
d. h. $r = p \Rightarrow$ (5.18) steht im Widerspruch zu (5.17b) und (5.17c).

Fall 2: $i \in {}_1\mathcal{U}^p \setminus \mathcal{M}^p \overset{(5.7)}{=} {}_1\mathcal{R}^p$
Abbildung 5.2 illustriert den folgenden Konstruktionsprozeß.
Nach Bemerkung 5.3.11 folgt:

$$p \in \alpha(i) \setminus \{r\} \quad \wedge \quad \exists k \in \mathcal{N}^r(i) \mid \mu(k) = p.$$

Im Scatterschritt (3) wird auf Prozessor p folglich von Prozessor r eine Nachricht, die die gesamte Nachbarschaft $\mathcal{N}^r(i)$ enthält, empfangen und $\forall k \in \mathcal{N}^r(i)\ A^r_{ik}, A^r_{ki}$ in A^p eingebaut. Speziell wurde also auch A_{ij}, A_{ji} in A^p eingebaut (falls noch nicht vorhanden).

$\overset{(4.10)}{\Longrightarrow}$ $j \in \mathcal{N}^p(i)$. Widerspruch zu (5.17c).

$\overset{(4.11)}{\Longrightarrow}$ $i \in \mathcal{N}^p(j)$ und $i \in {}_1\mathcal{U}^p$, d. h. $i \in \mathcal{N}^p(j) \cap {}_1\mathcal{U}^p \neq \emptyset$

$\overset{(5.8)}{\Longrightarrow}$ $j \in {}_2\mathcal{R}^p \overset{(5.9)}{\subseteq} {}_2\mathcal{U}^p$. Widerspruch zu (5.17b).

Die Annahme war also falsch und damit ist die Aussage bewiesen.

□

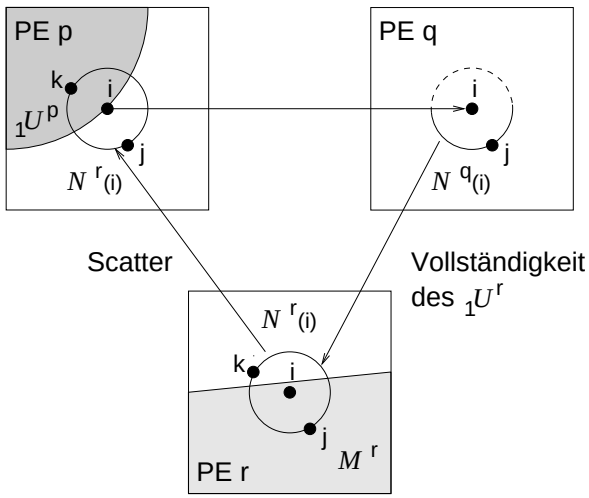


Abbildung 5.2: Vollständigkeit des ${}_2\mathcal{U}^p$ wird über Umweg über Prozessor r gesichert

5.3.4.3 Entfernen überflüssiger Knoten aus dem Überlapp

Wenn wir rekursiv einen bereits vergröberten Level bearbeiten, haben die Erfahrungen in der Praxis gezeigt, daß folgende Situation oft eintritt: Knoten am Rand der Kernpartition des feineren Levels sind fein markiert worden, wodurch sie auf diesem Level nicht mehr auftreten. Dadurch kann sich die Entfernung einiger grobmarkierter Borderknoten von der Kernpartition auf mehr als 2 vergrößern. Um den Speicher– und Zeitaufwand klein zu halten, wird man diese, nun nicht mehr benötigten Borderknoten, entfernen wollen. Dabei können zwei spezielle Situationen auftreten. Zum einen kann ein Borderknoten ohne jegliche Nebendiagonalmatrixeinträge existieren; er ist also von der Kernpartition aus *nicht* mehr erreichbar. Der Algorithmus muß darauf ausgelegt sein. Zum anderen kann es vorkommen, daß ein solcher Borderknoten ein Elternknoten zu einem feinen Masterknoten ist, der genau am Rand der Kernpartition des feineren Levels liegt. In diesem Fall darf er *nicht* entfernt werden, weil sonst die 0–Konsistenz der Restriktions– und Prolongationsmatrix verloren geht (vergleiche Abschnitt 5.7). Damit die Additivität der Steifigkeitsmatrix nicht verloren geht, dürfen die an den Vektoren hängenden Matrixeinträge nicht einfach weggeworfen werden. Vielmehr muß dafür gesorgt werden, daß diese Einträge verschickt und auf eine sicher weiterbestehende Instanz dieses Eintrags addiert wird; das ist z. B. der Eintrag, der am Masterknoten des zu entfernenden Knotens hängt. Bei diesem Vorgang ist auch der transponierte Matrixeintrag mitzuschicken, da auch er nicht mehr ohne den Knoten existieren kann. Insgesamt ergibt sich damit folgender Algorithmus zum Entfernen des nicht mehr benötigten Überlapps.

Algorithmus 5.4 (CleanOverlap($^{\ell}A$))

(1) $\forall i \in {}^{\ell}\mathcal{B}^p: \;\; i.Flag1 := 0; i.Flag2 := 0$
(2) $\forall j \in {}^{\ell}\mathcal{M}^p: \;\; \forall i \in \mathcal{N}^p(j) \cap {}^{\ell}\mathcal{B}^p: \;\; i.Flag1 := 1$
(3) $\forall j \in {}^{\ell}\mathcal{B}^p: \;\; \text{if } (j.Flag1 = 1) \quad \forall i \in \mathcal{N}^p(j) \cap {}^{\ell}\mathcal{B}^p: \;\; i.Flag2 := 1$
(4) $\forall j \in {}^{\ell-1}\mathcal{M}^p \;\; \forall i' \in {}^{\ell}\mathcal{B}^p \mid R^p_{i'j} \neq 0: \;\; i'.Flag2 := 1$
(5) $\forall i \in {}^{\ell}\mathcal{B}^p \mid i.Flag1 = 0 \wedge i.Flag2 = 0:$
(6) $\quad {}^{\ell}\mathcal{B}^p := {}^{\ell}\mathcal{B}^p \setminus \{i\}$
(7) $\quad \forall j \in \mathcal{N}^p(i)$ move A^p_{ij}, A^p_{ji} to $A^{\mu(i)}_{ij}, A^{\mu(i)}_{ji}$

Ohne dieses Ausdünnen des Überlapps wären manche der in Kapitel 6 vorgestellten Rechnungen aus Speicherplatzgründen nicht möglich gewesen; die Prozessorinterfaces wären so groß geworden, daß die benötigten Nachrichtenpuffer nicht mehr angelegt hätten werden können.

5.3.4.4 Gesamtalgorithmus zur Konstruktion des benötigten Überlapps

Die getroffenen Voraussetzungen in Abschnitt 5.2 erlauben eine direkte Anwendung des Algorithmus CompleteU1, der wiederum die Voraussetzungen für die Anwendbar-

keit von Algorithmus CompleteU2 schafft. Nach Satz 5.3.12 ist danach die Vollständigkeit des Überlapps der Tiefe 2 hergestellt. Die Gesamtaufgabe wird durch nachstehenden Algorithmus CompleteU gelöst. Das Ergebnis entspricht genau der Anforderung nach Abschnitt 5.3.3.

Algorithmus 5.5 (CompleteU($^{\ell}A$))
Voraussetzung: α respektiert $\mathcal{G}(^{\ell}A)$ und A ist struktursymmetrisch.
(1) CompleteU1($^{\ell}A$)
(2) CompleteU2($^{\ell}A$)
(3) CleanOverlap($^{\ell}A$)

Damit können jetzt die eigentlichen Berechnungen des parallelen FAMG ausgeführt werden.

5.4 Berechnung der Elternlisten

Da die F/C–Markierung nur für Masterknoten erfolgen muß, müssen nur für diese die vollständigen, dem seriellen Fall entsprechenden Elternmengen erzeugt werden. Den stärksten Anspruch an den Überlapp stellt dabei der Glätter (3.10) und (3.11). Er muß für alle $i \in \mathcal{M}^p$ angewendet werden und benötigt dafür Werte aus $\acute{\mathcal{N}}(\acute{\mathcal{N}}(i)) \subseteq \mathcal{N}(\mathcal{N}(i))$. Nach (5.10) ist dazu Überlapptiefe 2 notwendig und hinreichend, und wird durch Algorithmus CompleteU2 hergestellt.

Der Glätter im Auswahlprozeß soll seriell wie parallel lokal der gleiche sein. Für die parallele Rechnung benötigt man daher die 1–konsistente Matrix ${}_1\ddot{\ddot{A}}$; nach Lemma 5.3.6 ist aufgrund der Vollständigkeit des ${}_2\mathcal{U}$ dafür die teilkonsistente Matrix $\tilde{A}$ hinreichend. Die relevante Umgebung eines Masterknotens ist somit seriell wie parallel identisch und führt deshalb zu identischen Elternmengen.

5.5 Paralleler Markierungsalgorithmus

Nachdem für alle Knoten eine Liste mit den möglichen Elternpaaren angelegt wurde, muß diese Information nun dazu benutzt werden, die Knoten in einer guten Reihenfolge zu markieren. Im seriellen Algorithmus geschieht dies, indem alle Knoten in die Verarbeitungsliste $\mathcal{L}$ gestellt werden und danach elementweise einer mit einem besten Elternpaar verarbeitet wird; vergleiche Algorithmus 3.1 Label.

Hier ist nun wichtig festzustellen, daß dort der Update–Schritt (Zeile (7) im Unteralgorithmus 3.2 EliminateFNode bzw. Zeile (4) im Unteralgorithmus 3.3 EliminateCNode) auch die Sortierung der Liste verändert (auch die Liste der Besten) und somit die Auswahlgrundlage im darauffolgenden Schritt (4) beeinflußt. Der Markierungsprozeß ist dadurch *streng sequentiell* und muß für einen parallelen Ablauf

grundlegend umgestaltet werden. Dies ist das Hauptproblem der ganzen Parallelisierung, die nun entwickelt werden soll. Inwieweit damit Verluste eintreten, muß noch diskutiert werden.

5.5.1 Parallele Markierungsstrategien

Grundsätzlich werden nur Masterknoten in die Liste $\mathcal{L}$ gestellt. So ist eine eindeutige Zuständigkeit gewährleistet. Trotzdem treten 2 Probleme auf. Zunächst hat jeder Prozessor nur einen Teil J^p der gesamten Knoten J. Damit ist die Liste der Besten auf jedem Prozessor eine andere und unterscheidet sich in allgemeinen von derjenigen im seriellen Fall. Nicht nur die besten Knoten differieren, sondern auch das beste Gewicht. Kein Prozessor kennt die global Besten. Diese Aussagen setzen sich auf alle folgenden Markierungsschritte fort. Wie soll nun von jedem Prozessor entschieden werden, welcher Knoten zu markieren ist?

Das zweite Problem ist noch entscheidender. Obwohl alle Knoten in der Liste aus der Kernpartition $\mathcal{M}^p$ stammen, können ihre Elternknoten dennoch aus dem gesamten Überlapp der Tiefe 1 stammen. Wird ein Knoten grob oder fein markiert, so muß dies auch allen seinen Instanzen mitgeteilt werden, um die Konsistenz der Datenstruktur zu gewährleisten. Ist eine seiner Instanzen im selben Verarbeitungsschritt ebenfalls markiert worden, aber mit der anderen Marke, so ist dieser Konflikt nicht mehr zu lösen, denn eine einmal getroffene Entscheidung kann nicht mehr revidiert werden: Sie hat schon die Datenstruktur verändert und alle danach erfolgten Markierungen mitentschieden. Dieser Konflikt muß also schon in der Entstehung verhindert werden, indem das Markieren in geeigneter Weise synchronisiert wird.

Bei der Parallelisierung anderer AMG Verfahren ist teilweise eine einfache Konfliktlösung möglich. So kann in [Wie00b] einfach eine Grobmarkierung eine schon vorhandene Feinmarkierung „überstimmen“. Auch [KS99] sprechen diese Möglichkeit an; sie haben allerdings schlechte Vergröberungsraten festgestellt, die zu einer schlechten Operatorkomplexität führen und deshalb ist dieses stark vereinfachte Vorgehen zu vermeiden. Weitere Überlegungen sind in [WJ99] zu finden.

In unserem Verfahren muß die Strategie aber darauf hinauslaufen, zu verhindern, daß in einem Verarbeitungsschritt 2 Instanzen eines Knotens markiert werden *können*.

Es besteht die Möglichkeit, den seriellen Ablauf nachzuvollziehen: In jedem Verarbeitungsschritt wird in einer globalen Kommunikation bestimmt, was das beste derzeit mögliche Gewicht ist. Aus den Prozessoren, deren bestes Gewicht dem globalen Optimum entspricht, muß dann genau einer ausgewählt werden, der seine Elternauswahl tatsächlich durchführt; das entspricht der Auswahl eines Knotens aus der Bestenliste. Dieser Ansatz ist allerdings zu feingranular, als daß er gute Effizienz erwarten ließe.

Eine ganze Schar von Strategien eröffnet eine prozessorweise Betrachtung: Ein Prozessor p kann all seine Knoten in die Liste $\mathcal{L}^p$ stellen und zu markieren versuchen,

wenn sichergestellt ist, daß im selben Verarbeitungsschritt kein anderer Prozessor q einen Knoten in seine Liste $\mathcal{L}^q$ stellt, so daß ein markierter Knoten eine Instanz auf dem ersten Prozessor hat. Es muß also gelten

$$\forall p \in \mathcal{P} \ \forall q \in \mathcal{P} \setminus \{p\} : \alpha(\mathcal{N}^p(\mathcal{L}^p)) \cap \alpha(\mathcal{N}^q(\mathcal{L}^q)) = \emptyset. \tag{5.19}$$

Am einfachsten ist diese Bedingung zu erreichen, wenn die Prozessoren einzeln nacheinander arbeiten. Zusätzliche Parallelisierung der Arbeit ist aber zu erzielen, wenn in jedem Schritt möglichst viele zulässige Prozessoren aktiv sind. Ein Graphfärbungsverfahren hierzu wird in Abschnitt 5.6 entwickelt. Es liefert eine Liste $\mathcal{T}$ von maximal unabhängigen Teilmengen von Prozessoren, die jeweils gleichzeitig arbeiten dürfen.

Als Vereinfachung soll noch der Algorithmus CoarsenRemainingNodes eingeführt werden, der die unmarkierten Knoten zwangsweise vergröbert und dabei die Elternliste konsistent hält. Im Gegensatz zum seriellen Algorithmus können hier die Gewichte $w(\mathbb{P}_J)$ nicht inkonsistent gelassen werden (vergleiche Zeile (6) in Algorithmus 3.1 Label). $\mathcal{D}$ ist die Menge der noch nicht markierten Knoten aus Algorithmus 3.1.

Algorithmus 5.6 (CoarsenRemainingNodes)

```
(1)  ∀i ∈ D
(2)      C := C ∪ {i}
(3)      aktualisiere alle betroffenen Gewichte w(P_J)
```

Damit lassen sich die bisherigen Überlegungen zu einem vorläufigen Algorithmus formulieren:

Algorithmus 5.7 (pLabel_1, erste Fassung**)**

```
(1)  ∀C ∈ T
(2)      if p ∈ C
(3)          Label(M^p)
(4)          CoarsenRemainingNodes
(5)      Synchronize
```

Wichtig ist Schritt (5), der eine globale Synchronisierung der einzelnen Verarbeitungsschritte erzwingt.

Eine weitere Erhöhung des Parallelisierungsgrades kann durch eine Verringerung der Anzahl der Knoten erreicht werden, die in solchen teilsequentialisierten Verarbeitungsschritten bearbeitet werden. Synchronisiert werden muß eigentlich nur die Bearbeitung der Masterknoten in den Prozessorinterfaces $\mathcal{I}^p$ nach Definition 5.3.1; ein Knoten in $\mathcal{M}^p \setminus \mathcal{I}^p$ markiert höchstens Knoten in $\mathcal{N}^p(\mathcal{M}^p \setminus \mathcal{I}^p) = \mathcal{M}^p$ als grob und beeinflußt damit nicht andere Prozessoren — erzeugt also keine Konflikte — und kann deshalb vollkommen parallel bearbeitet werden.

Am Ende eines jeden Verarbeitungsschrittes müssen alle noch nicht markierten Knoten $\mathcal{D}$ aus der Liste zwangsweise vergröbert werden, weil sie für eine Verfeinerung ein zu schlechtes Gewicht haben (sonst wären sie ja jetzt nicht noch übrig!) und die Liste für den nächsten Schritt wieder benötigt wird; eine Vergröberung an dieser Stelle läßt jedoch ein Mitbenutzen als Elternknoten an anderer Stelle zu, was sogar angestrebt wird.

Gebietsrandknoten mit Dirichlet–Randbedingung können bei Bedarf in allen Fällen vorweg fein markiert werden, weil sie dazu keine Elternknoten benötigen und somit keine Konflikte erzeugen.

Zusammenfassend ergibt sich also folgender Algorithmus, der den höchsten Parallelisierungsgrad besitzt:

Algorithmus 5.8 (pLabel_2, zweite Fassung**)**

```
(1)  Label(M^p ∩ Dirichletrand)
(2)  Label(M^p \ I^p)
(3)  ∀C ∈ T
(4)      if p ∈ C
(5)          Label(I^p)
(6)          CoarsenRemainingNodes
(7)      Synchronize
```

Prinzipiell kann Schritt (2) auch nach den restlichen Schritten erfolgen; dann ist noch ein CoarsenRemainingNodes nötig. Welche Anordnung besser ist, müssen numerische Experimente zeigen.

Die 7 beschriebenen Strategien zur Parallelisierung der Markierungsreihenfolge sind in Tabelle 5.1 zusammengefaßt und stellen die weißen Felder dar. Die dunklen Felder sind nicht mögliche Kombinationen. Die parallele Effizienz der einzelnen Methoden konnte schon recht gut abgeschätzt werden. Die Gesamteffizienz wird aber auch wesentlich von der numerischen Güte der erzeugten Grobgittermatrizen bestimmt. Diese Eigenschaft kann nicht theoretisch abgeschätzt werden und bedarf numerischer Experimente. Es ist zu erwarten, daß sich die Güte entgegengesetzt zur Parallelisierbarkeit verhält und es damit einen mittleren Bereich geben wird, in dem die Gesamteffizienz optimal ist. Daß die Güte nicht zu stark leidet ist zu erwarten, da die serielle Auswahlreihenfolge kein Optimum darstellt, sondern lediglich durch Heuristiken motiviert ist; Abweichungen davon *müssen nicht* schlechter sein. Gestützt wird diese Hoffnung durch die beobachtete relative Unempfindlichkeit des seriellen Verfahrens von der Markierungsreihenfolge. Eine endgültige Beurteilung kann daher erst im Kapitel 6 erfolgen.

5.5.2 Kommunikation der Fein–/Grobmarkierung

Die Markierungsstrategie vermeidet, daß in einem Verarbeitungsschritt mehr als eine Instanz eines Knotens fein oder grob markiert wird. Trotzdem müssen die Markierun-

		Knoten je Verarbeitungsschritt			
		alle Masterknoten	Master im Prozessorinterface		ein Masterknoten mit bestem Gewicht
			Interfaceknoten zuerst markieren	Interfaceknoten nach restlichen Masterknoten markieren	
beteiligte Prozessoren je Verarbeitungsschritt	einer nach dem anderen				
	maximal unabhängige Menge je Schritt				
	einer mit bestem Gewicht				

Tabelle 5.1: Übersicht der parallelen Markierungsstrategien

gen an alle Knoteninstanzen kommuniziert werden, damit die Datenstruktur wieder konsistent wird: Denn diese anderen Instanzen müssen die eben geschaffene Vergröberungssituation den Nachbarprozessoren als Ausgangssituation für deren Markierungsprozeß bereitstellen. Neben der eigentlichen Fein–/Grob–Information muß auf den Nachbarprozessoren vor allem die Datenstruktur der markierten Knoten und ihrer Nachbarschaft so verändert werden, als wäre die Markierung auf diesem Prozessor selbst verursacht worden.

Die Nachricht muß alle Informationen enthalten, um die nötigen Veränderungen zu beschreiben. Ein grober Knoten muß nichts weiteres als diese Tatsache an sich mitteilen. Ein feiner Knoten muß daneben noch mitteilen, welche Eltern er verwendet hat und mit welchen Interpolationsgewichten er aus diesen berechnet werden möchte,

da diese Informationen den anderen Instanzen nicht vorliegen beziehungsweise dort nicht selbst berechnet werden können. Da das verwendete Modul DDD erfordert, daß immer alle Knoten an der Kommunikation teilnehmen, benötigen wir noch einen Nachrichtentyp, der besagt, daß sich am Markierungszustand nichts geändert hat; das vermeidet insbesondere für feine Knoten ein unnötiges Einsammeln und Einpacken der Informationen.

Damit ein Knoten in der Kommunikationsphase bei der Gatheroperation feststellen kann, ob seine Markierung im eben durchgeführten Verarbeitungsschritt oder in einem früheren Schritt, dessen Markierung schon kommuniziert wurde, entstanden ist, benötigt er ein entsprechendes Flag, das vor dem ersten Verarbeitungsschritt mit „unverändert“ initialisiert wird, beim Fein–/Grobmarkieren auf „neumarkiert“ gesetzt wird und am Ende der Kommunikationsphase wieder auf „unverändert“ zurückgesetzt wird.

Auf Empfängerseite muß beim Einbau der angekommenen Informationen dafür gesorgt werden, daß die genau gleichen Veränderungen an den Datenstrukturen vorgenommen werden, als wenn die Markierung lokal vorgenommen worden wäre; am besten werden folglich dieselben Funktionen verwendet. Man muß allerdings darauf achten, daß die Funktion zum Grobmarkieren robust gegen das bereits Grobsein des Knotens ist: Der Knoten kann zwar nicht in der selben Kommunikationsphase von einem anderen Prozessor ebenfalls grob gesetzt werden — dafür sorgt die Markierungsstrategie —, aber das Anwenderprogramm hat keine Kontrolle darüber, in welcher Reihenfolge die Nachrichten eines feinen Knotens und seiner Eltern eintreffen; jede der beiden Nachrichten (für Kindknoten und den Elternknoten) beschreibt eine Vergröberung des Elternknotens. Diese Erschwernis läßt sich *nicht* dadurch vermeiden, daß nur Nachrichten für feine Knoten geschickt werden (die die Elterninformation eigentlich schon beinhalten), weil ein Elternknoten durchaus auf einem Prozessor vorhanden sein kann, auf dem der Kindknoten nicht existiert, er muß aber ebenfalls die Nachricht über die Vergröberung bekommen.

Auf dem ersten Blick mag man erschrecken, daß zu viele unverändert–Nachrichten in jedem Schritt versandt werden. Da man aber aus Gründen einer guten Parallelisierung bestrebt sein wird, viele Prozessoren in einem Verarbeitungsschritt arbeiten zu lassen, wird es nicht allzuviele Knoten geben, die weder neu markiert wurden, noch eine solche Nachricht erhalten müssen, also von der Kommunikation ausgespart werden könnten.

Zusammenfassend ergibt sich der folgende Algorithmus für die Kommunikation der Fein–/Grobmarkierungen.

Algorithmus 5.9 (CommunicateNodeStatus)

(1) Gather: $\forall q \in \alpha(i)$: erzeuge im Nachrichtenpuffer $\mathcal{M}_{p,q}$ folgende Nachricht:
    if i is newmarked
        if i is fine
            Fine($\mathbb{P}_i$)
        else Coarse()
    else Unchanged()
(2) Datenaustausch: sende alle $M_{p,q}$ und empfange alle $M_{q,p}$
(3) Scatter: $\forall q \in \alpha(i) \setminus \{p\}$ hole Nachricht N aus Puffer $\mathcal{M}_{q,p}$
    case $N.type$
        fine: $\mathbb{P}_{\mathcal{L}} := \mathbb{P}_{\mathcal{L}} \setminus \mathbb{P}_i^+$
            $\mathbb{P}_{\mathcal{L}} := \mathbb{P}_{\mathcal{L}} \cup N.\mathbb{P}_i$
            EliminateFNode($N.\mathbb{P}_i$)
        coarse: EliminateCNode(i)
        unchanged: ist nichts zu tun.
(4) Markierungsflag für i zurücksetzen.

Damit kann der parallele Markierungsalgorithmus vollständig formuliert werden. Es ist keine explizite Synchronisierung der einzelnen Verarbeitungsschritte (Zeilen (5)-(7)) nötig, weil diese Aufgabe durch CommunicateNodeStatus in Zeile (8) implizit miterledigt wird.

Algorithmus 5.10 (pLabel($\mathcal{M}$), endgültige Fassung)

(1) Label($\mathcal{M}^p \cap Dirichletrand$)
(2) Label($\mathcal{M}^p \setminus \mathcal{I}^p$)
(3) CommunicateNodeStatus
(4) $\forall C \in \mathcal{T}$
(5)   if $p \in C$
(6)   Label($\mathcal{I}^p$)
(7)   CoarsenRemainingNodes
(8)   CommunicateNodeStatus

5.5.3 Speziallösungen für strukturierte Gitter?

Ein strukturierter Matrixgraph sollte eventuell auch bessere Speziallösungen erlauben, verglichen mit dem allgemeinen Fall eines unstrukturierten Graphen. Eine offensichtliche Idee ist in Abbildung 5.3 skizziert: Man kann in einem Schritt 2 Prozessorränder auf einmal markieren, weil sich dieses Schema auf alle Prozessoren konsistent erweitern läßt. Der Vorteil ist zunächst, daß nur 2 Kommunikationsschritte benötigt werden. Er relativiert sich aber, weil je 2 Eckpunkte ausgespart bleiben müssen. Ein exaktes Vorgehen erfordert dafür allein 2 weitere Kommunikationsschritte. Alternativ können sie entweder zwangsweise vergröbert werden, was die Reduzierbarkeit des Baselevelgitters limitiert, oder man führt die Zusammenfassung

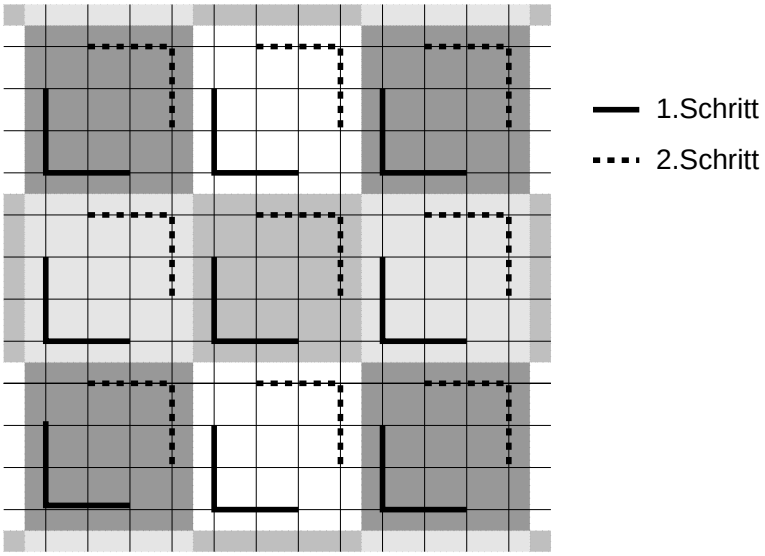


Abbildung 5.3: Spezielle Markierungsverfahren für strukturierte Matrixgraphen

der Interfaceseiten alternierend von Level zu Level durch (⌜⌟ abwechselnd mit ⌞⌝). In allen Fällen ist Zusatzaufwand nötig.

Als müßig erweisen sich all diese Überlegungen, weil sie nicht rekursiv anwendbar sind: Die entstehende Grobgittermatrix wird im allgemeinen *nicht* mehr diese Strukturiertheit besitzen, so daß spätestens ab Level -1 eine allgemeine Markierungsstrategie angewendet werden muß. Auch auf der gegebenen Feingittermatrix wird das allgemeine Verfahren mit 4 Schritten auskommen, so daß auch dort keine wesentlichen Vorteile — verglichen mit dem Zusatzaufwand — gezogen werden können.

5.6 Graphfärbung

Im vorangehenden Abschnitt 5.5 wurde die Notwendigkeit erklärt, daß verschiedene Prozessoren während des F/C–Markierens im Bereich des Prozessorinterfaces nicht die gleichen Knoten bearbeiten dürfen, um Inkonsistenzen zu vermeiden. Hier entwickeln wir nun schrittweise den Algorithmus Coloring zur Lösung dieses Problems. Dazu beschreiben wir die vorliegende Abhängigkeit der Prozessoren in Form eines Graphen und erstellen ein Verfahren zur Färbung dieses Graphen. Daraus lassen sich dann die benötigten, maximal unabhängigen Prozessorteilmengen $\mathcal{T}$ bestimmen.

5.6.1 Definition des Färbungsgraphen

Der Färbungsgraph soll die Abhängigkeit der Prozessoren während des Markierungsalgorithmus Label modellieren. Jeder Prozessor wird als ein Knoten im Graphen dar-

gestellt. Eine Kante (p, q) wird definiert als:

> ein Knoten i auf Prozessor p kann beim Markieren einen Knoten j auf Prozessor q beeinflussen.

Das Markieren eines Knotens i zieht nach Zeile (6) des Algorithmus 3.2 EliminateFNode auch das Markieren der Elternknoten nach sich. Da nach (3.14) die Elternknoten aus der direkten Nachbarschaft $\mathcal{N}(i)$ eines Knotens stammen, beeinflußt folglich das Markieren eines Knotens i auch seine Nachbarschaft $\mathcal{N}(i)$ bzgl. der F/C–Markierung, wie es in (5.19) schon formuliert wurde. Genau dieser Einfluß muß im Färbungsgraphen ausgedrückt werden.

Definition 5.6.1 (Färbungsgraph $\mathcal{G}_A^F$)
Der **Färbungsgraph** $\mathcal{G}_A^F := (\mathcal{V}, \mathcal{E})$ zu einer verteilten Matrix A und einer Aufteilungsabbildung α sei definiert als:

$$\begin{aligned} \mathcal{V} &:= \mathcal{P} \\ \mathcal{E} &:= \{(p, q) \in \mathcal{V} \times \mathcal{V} \mid {}_1\mathcal{U}^p \cap {}_1\mathcal{U}^q \neq \emptyset\}. \end{aligned} \tag{5.20}$$

5.6.2 Konstruktion des Färbungsgraphen

Für eine algorithmische Formulierung ist die Mengenschreibweise in (5.20) nicht sehr brauchbar. Daher geben wir eine prädikatenlogische Form, die sich an einem Abarbeiten der Knoten einer Indexmenge orientiert.

Definition 5.6.2 (Prädikate „p und q beeinflussen sich“)
Es seien folgende Prädikate auf der Menge $\mathcal{P} \times \mathcal{P}$ definiert:

$$C(p, q) := \exists i \in J : i \in {}_1\mathcal{R}^p \wedge i \in \mathcal{M}^q \tag{5.21}$$
$$D(p, q) := \exists i \in J : i \in {}_1\mathcal{R}^p \wedge i \in {}_1\mathcal{R}^q \tag{5.22}$$
$$E(p, q) := \exists i \in J : i \in \mathcal{M}^p \wedge i \in {}_1\mathcal{R}^q \tag{5.23}$$
$$H(p, q) := \exists i \in J : i \in {}_2\mathcal{R}^p \wedge i \in \mathcal{M}^q. \tag{5.24}$$

Damit definieren wir zwei Formulierungen des Prädikats „p und q beeinflussen sich“:

$$B(p, q) := C(p, q) \vee D(p, q) \vee E(p, q) \tag{5.25}$$
$$\tilde{B}(p, q) := C(p, q) \vee H(p, q) = \exists i \in J : i \in {}_1\mathcal{R}^p \cup {}_2\mathcal{R}^p \wedge i \in \mathcal{M}^q. \tag{5.26}$$

Bemerkung 5.6.3 (Unterschied der Prädikate B und $\tilde{B}$)
Jeder Prozessor $p \in \mathcal{P}$ wird für alle $i \in J^p$ die Prädikate bewerten müssen. In dem hier verwendeten Modell der parallelen Datenaufteilung aus Abschnitt 4.1.1 ist für jeden Knoten $i \in J^p$ auf Prozessor p mittels der Aufteilungsabbildung α bekannt, auf welchen anderen Prozessoren der Knoten i weitere Instanzen besitzt, und mittels

der Masterabbildung μ, auf welchem davon i als Master vorliegt. Es ist jedoch *nicht* möglich zu entscheiden, in welcher Überlapptiefe ein Borderknoten auf einem anderen Prozessor liegt. Prädikat B benötigt aber in Teilprädikat D und E das Wissen, ob ein Knoten auf einem anderen Prozessor ein Borderknoten genau in Tiefe 1 ist. Daher ist eine andere Formulierung notwendig, die lokal entscheidbar ist.

Prädikat $\tilde{B}$ benötigt nur Informationen, die lokal auf jedem Prozessor vorhanden sind und ist damit algorithmisch verwendbar. Ob der mittels $\tilde{B}$ konstruierte Graph identisch zu dem Färbungsgraph in Definition 5.6.1 ist, ist nicht offensichtlich und muß erst bewiesen werden.

Wir werden in zwei Schritten vorgehen. Zuerst wird gezeigt, daß der mittels B konstruierte Graph genau der Färbungsgraph ist. Danach beweisen wir die Äquivalenz der Prädikate B und $\tilde{B}$.

Lemma 5.6.4 (Prädikat B erzeugt den Färbungsgraph $\mathcal{G}_{\mathbf{A}}^{\mathbf{F}}$)
Der Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ mit

$$\begin{aligned} \mathcal{V} :=& \mathcal{P} \\ \mathcal{E} :=& \{(p,q) \in \mathcal{V} \times \mathcal{V} \mid B(p,q)\} \end{aligned} \tag{5.27}$$

ist äquivalent zum Färbungsgraphen nach Definition 5.6.1.

Beweis: Die Knotenmengen beider Graphen sind identisch. Für die Kantenmenge gilt:

$$\begin{aligned} (p,q) \in \mathcal{E} \quad &\overset{(5.20)}{\Longleftrightarrow} \quad {}_1\mathcal{U}^p \cap {}_1\mathcal{U}^q \neq \emptyset \\ &\Longleftrightarrow \quad \exists i \in J: \ \ i \in {}_1\mathcal{U}^p \wedge i \in {}_1\mathcal{U}^q \\ &\overset{(5.7)}{\Longleftrightarrow} \quad \exists i \in J: \ \ i \in (\mathcal{M}^p \cup {}_1\mathcal{R}^p) \wedge i \in (\mathcal{M}^q \cup {}_1\mathcal{R}^q) \\ &\Longleftrightarrow \quad \exists i \in J: \ \ i \in \mathcal{M}^p \cap \mathcal{M}^q \vee i \in \mathcal{M}^p \cap {}_1\mathcal{R}^q \\ &\qquad\qquad\qquad\quad \vee i \in {}_1\mathcal{R}^p \cap \mathcal{M}^q \vee i \in {}_1\mathcal{R}^p \cap {}_1\mathcal{R}^q \end{aligned}$$

wegen der Eindeutigkeit des Masters (4.6) gilt $\mathcal{M}^p \cap \mathcal{M}^q = \emptyset$

$$\begin{aligned} &\overset{(5.21)-(5.23)}{\Longleftrightarrow} \quad C(p,q) \vee D(p,q) \vee E(p,q) \\ &\overset{(5.25)}{\Longleftrightarrow} \quad B(p,q). \end{aligned}$$

□

Lemma 5.6.5 (Äquivalenz von B und $\tilde{B}$)
Es sei der Überlapp der Tiefe 2 vollständig. Dann sind die beiden Prädikate B und $\tilde{B}$ äquivalent.

Beweis: Da die Argumente der Prädikate immer (p,q) sein werden, sollen sie in diesem Beweis zur Vereinfachung weggelassen werden.

Es ist nach Definition 5.6.2 zu zeigen

$$C \vee D \vee E \iff C \vee H.$$

Wir gehen in drei Schritten vor:

(1) $D \Longrightarrow C \vee H$

(2) $E \Longrightarrow C$

(3) $H \Longrightarrow C \vee D \vee E.$

Aus (1) und (2) folgt dann $C \vee D \vee E \Longrightarrow C \vee H$, d. h. $B \Rightarrow \tilde{B}$.
Aus (3) folgt $C \vee H \Longrightarrow C \vee D \vee E$, d. h. $\tilde{B} \Rightarrow B$.
Insgesamt ist die Äquivalenz damit bewiesen.

Schritt (1): $D \Longrightarrow C \vee H$

$D \overset{(5.22)}{\iff} \exists i \in J : i \in {}_1\mathcal{R}^p \wedge i \in {}_1\mathcal{R}^q$

$\overset{(5.6)}{\Longrightarrow} \exists j \in \mathcal{M}^q : j \in \mathcal{N}^q(i)$

Voraussetzung (Vollständigkeit von ${}_2\mathcal{U}^p$ (5.10))

$\Longrightarrow \exists j \in \mathcal{M}^q : j \in \mathcal{N}^p(i) \wedge j \in {}_2\mathcal{U}^p$

da ${}_2\mathcal{U}^p \overset{(5.9)}{=} {}_2\mathcal{R}^p \cup {}_1\mathcal{R}^p \cup \mathcal{M}^p$ und wegen der Eindeutigkeit des Masters (4.6) ($j \in \mathcal{M}^q \Longrightarrow j \notin \mathcal{M}^p$)

$\Longrightarrow \exists j \in \mathcal{M}^q : j \in {}_1\mathcal{R}^p \cup {}_2\mathcal{R}^p$

$\iff \exists j \in J : (j \in {}_1\mathcal{R}^p \wedge j \in \mathcal{M}^q) \vee (j \in {}_2\mathcal{R}^p \wedge j \in \mathcal{M}^q)$

$\overset{(5.21),(5.24)}{\iff} C \vee H.$

Schritt (2): $E \Longrightarrow C$

$E \overset{(5.23)}{\iff} \exists i \in J : i \in \mathcal{M}^p \wedge i \in {}_1\mathcal{R}^q$

$\overset{(5.6)}{\Longrightarrow} \exists j \in \mathcal{M}^q : i \in \mathcal{N}^q(j)$

Voraussetzung (Vollständigkeit von ${}_1\mathcal{U}^p$ (5.10))

$\Longrightarrow \exists j \in \mathcal{M}^q : j \in {}_1\mathcal{U}^p$

da ${}_1\mathcal{U}^p \overset{(5.7)}{=} \mathcal{M}^p \cup {}_1\mathcal{R}^p$ und wegen der Eindeutigkeit des Masters (4.6) ($j \in \mathcal{M}^q \Longrightarrow j \notin \mathcal{M}^p$)

$\Longrightarrow \exists j \in \mathcal{M}^q : j \in {}_1\mathcal{R}^p$

$\overset{(5.21)}{\iff} C.$

Schritt (3): $H \Longrightarrow C \vee D \vee E$

$$
\begin{aligned}
H \quad &\overset{(5.24)}{\Longleftrightarrow} && \exists j \in J : j \in {}_2\mathcal{R}^p \wedge j \in \mathcal{M}^q \\
&\overset{(5.8)}{\Longrightarrow} && \exists i \in J : i \in \mathcal{N}^p(j) \wedge i \in {}_1\mathcal{U}^p \wedge j \in \mathcal{M}^q \\
&\text{Voraussetzung (Vollständigkeit } {}_1\mathcal{U}^q \text{ (5.10))} \\
&\Longrightarrow && \exists i \in J : i \in \mathcal{N}^p(j) \wedge i \in {}_1\mathcal{U}^p \wedge i \in \mathcal{N}^q(j) \wedge i \in {}_1\mathcal{U}^q \\
&\Longrightarrow && \exists i \in J : i \in {}_1\mathcal{U}^p \wedge i \in {}_1\mathcal{U}^q \\
&\overset{(5.7)}{\Longleftrightarrow} && \exists i \in J : (i \in {}_1\mathcal{U}^p \wedge i \in \mathcal{M}^q) \vee (i \in {}_1\mathcal{U}^p \wedge i \in {}_1\mathcal{R}^q) \\
&\overset{(5.7)}{\Longleftrightarrow} && \exists i \in J : (i \in \mathcal{M}^p \wedge i \in \mathcal{M}^q) \vee (i \in {}_1\mathcal{R}^p \wedge i \in \mathcal{M}^q) \\
& && \quad \vee (i \in \mathcal{M}^p \wedge i \in {}_1\mathcal{R}^q) \vee (i \in {}_1\mathcal{R}^p \wedge i \in {}_1\mathcal{R}^q) \\
&\text{wegen der Eindeutigkeit des Masters (4.6) } (\mathcal{M}^p \cap \mathcal{M}^q = \emptyset) \\
&\overset{(5.21),(5.22),(5.23)}{\Longleftrightarrow} && C \vee D \vee E.
\end{aligned}
$$

□

Satz 5.6.6 (Lokal ausführbare Konstruktion des Färbungsgraphen)
Es sei der Überlapp der Tiefe 2 vollständig. Dann läßt sich mittels Prädikat $\tilde{B}$ rein lokal der Färbungsgraph $\mathcal{G}_A^F$ konstruieren.

Beweis: Der Beweis ergibt sich aus Lemma 5.6.4 und 5.6.5. □

Bemerkung 5.6.7 (Symmetrie verletzt?)
Die Formulierung des Prädikats $\tilde{B}$ in (5.26) ist unsymmetrisch in p und q. Das ist auch nötig, weil die nicht verfügbare Information auf dem Nachbarprozessor durch weitere lokale Informationen ersetzt werden muß. Aufgrund der Strukturssymmetrie der Steifigkeitsmatrix nach Voraussetzung (5.3) ist $\tilde{B}$ dennoch symmetrisch. Die (statische) Matrixstruktur und der (statische) Überlapp helfen somit (dynamische) Kommunikation zu sparen.

Für die algorithmische Umsetzung (Algorithmus 5.11 ConstructColoringGraph) muß die Größe des Überlapps berücksichtigt werden. Algorithmus CompleteU2 sichert die Vollständigkeit der Tiefe 2 zu, kann und darf aber manche schon bestehende Borderknoten einer größeren Tiefe nicht entfernen. Wenn man über alle Borderknoten gehen würde, sind auch solche von Tiefen größer 2 enthalten. Das muß verhindert werden. Einerseits würde das einen unnötig dichten Prozessorgraphen ergeben, der eventuell mehr Farben benötigt und damit mehr Verarbeitungsschritte beim Markieren der Prozessorinterfaces erzwingt.

Mag man diesen Effizienzverlust noch tolerieren, kann man den essentiellen zweiten Grund nicht umgehen: Größere Tiefen sind nicht mehr vollständig, d. h. benachbarte Prozessoren haben nicht mehr die gleichen Borderknoten. Durch diesen Symmetriebruch bedingt, kann man dann nicht mehr auf Kommunikation verzichten, wie in Bemerkung 5.6.7 gezeigt. Als Ausweg werden wir die Borderknoten der Tiefe 2 durch einen Schalen–Markierungsalgorithmus identifizieren (Zeilen (2)-(4)) und nur noch für sie das Prädikat $\tilde{B}$ prüfen (Zeile (5) im Algorithmus ConstructColoringGraph).

Algorithmus 5.11 (ConstructColoringGraph)

(1) $\mathcal{E}^p := \emptyset$
(2) $\forall i \in \mathcal{B}^p : i.Flag := 0$
(3) $\forall i \in \mathcal{B}^p : if(\mathcal{N}^p(i) \cap \mathcal{M}^p \neq \emptyset) \quad i.Flag := 1$
(4) $\forall i \in \mathcal{B}^p : if(i.Flag = 1 \vee \ \exists j \in \mathcal{N}^p(i) : \ \ j.Flag = 1)$
(5) $\qquad \mathcal{E}^p := \mathcal{E}^p \cup \{(p, \mu(i))\}$

Der Färbungsgraph $\mathcal{G}_A^F := (\mathcal{P}, \bigcup_{p \in \mathcal{P}} \mathcal{E}^p)$ liegt nun verteilt vor und soll gefärbt werden.

5.6.3 Der parallele Graphfärbungsalgorithmus

Ziel einer Graphfärbung ist es, jedem Knoten eine Farbe derart zuzuweisen, daß keine durch eine Kante direkt verbundenen Knoten die gleiche Farbe besitzen. Dabei sollen möglichst wenige Farben benutzt werden. Die Knoten einer Farbe bilden damit unabhängige Teilmengen bzgl. dieses Graphen.

Man könnte den verteilten Färbungsgraphen durch Kommunikation auf einem Prozessor zusammensammeln, dort lokal einen seriellen Färbungsalgorithmus anwenden und das Ergebnis wieder an alle Prozessoren versenden.

Um die parallele Skalierbarkeit nicht zu gefährden, wählen wir aber einen anderen Weg, der ein vollkommen paralleles Verfahren ergibt. Da die Graphfärbung ein NP–hartes Problem ist [GJ79], ist man grundsätzlich auf heuristische Algorithmen angewiesen, die nur eine Näherungslösung finden werden. Die Parallelisierung muß also nicht einen Algorithmus erhalten, der die exakte Lösung berechnet, sondern kann selbst heuristisch vorgehen.

Für einen allgemeinen, verteilten Graphen haben R. Gjertsen, M. Jones und P. Plassmann eine Klasse verteilt arbeitender Graphfärbungsalgorithmen entwickelt und erprobt [JP93][GJP96]. Wir haben den Spezialfall vorliegen, daß jeder Prozessor nur einen Knoten des Graphen enthält. Daher haben wir eine speziell an diese Situation angepaßte Variante entwickelt, die nun vorgestellt wird.

Die bekannten seriellen Algorithmen sind vom Greedy–Typ, also streng sequentiell arbeitend. Ähnlich wie bei Lösungsverfahren, die auf der Gauß–Elimination beruhen, muß dieser sequentielle Ablauf aufgebrochen werden.

Der Ansatz ist, die Knotenmenge in unabhängige Teilmengen aufzuspalten und diese unabhängig voneinander konsistent zu färben. M. Luby hat einen ersten, verteilt aber synchron arbeitenden Algorithmus angegeben [Lub86], der eine Monte Carlo Methode ist. Er konnte von M. Jones und P. Plassmann zu einem asynchronen und damit effizienteren Verfahren verbessert werden [JP93]. Wir entwickeln daraus einen auf unsere Situation angepaßte Variante. Dabei bedienen wir uns der message passing Kommunikationsroutinen des Moduls PPIF und setzten damit unterhalb der DDD-Schicht an.

Es sei $\deg(p) := |\{(r,q) \in \mathcal{E} \mid r = p\}|$ der Knotengrad und $\mathsf{rand}(p)$ eine Pseudo–Zufallszahl im Intervall $(0,1) \subset \mathbb{R}$ zum Startwert p. Dann sei über

$$w(p) := \deg(p) + \mathsf{rand}(p) \tag{5.28}$$

eine **Gewichtsfunktion** w definiert. Durch die Zufallsfunktion werden Knotengewichte mit gleichem Grad randomisiert und so (fast sicher) unterscheidbar gemacht. Die üblichen Realisierungen einer Funktion rand liefern zu einem festen Startwert (englisch *seed*) immer das gleiche Ergebnis; man spricht daher von einer Pseudo–Zufallszahlenfunktion. Diese Eigenschaft werden wir später noch ausnutzen.

Die Gewichte induzieren eine irreflexible, partielle Ordnung auf der Knotenmenge:

$$p \succ q \;:\Longleftrightarrow\; (p,q) \in \mathcal{G}_A^F \;\wedge\; w(p) > w(q).$$

Alle noch ungefärbten Knoten $p \in \mathcal{V}$, die bzgl. dieser partiellen Ordnung maximal sind, sind unabhängig bzgl. der Nachbarschaft im Färbungsgraphen und können sich gleichzeitig die kleinste Farbe geben, die von ihren bereits gefärbten Nachbarknoten noch nicht verwendet wurde. Es ist also gelungen, die gesamte Knotenmenge so aufzuteilen, daß in jedem Schritt des Greedy–Färbungsalgorithmus eine parallel und unabhängig von einander zu bearbeitende Teilmenge von Knoten vorliegt. Nach jedem Schritt muß ein Datenaustausch mit den noch ungefärbten Nachbarprozessoren durchgeführt werden, in dem die eben gewählte Farbe mitgeteilt wird. Damit können wir den parallelen Färbungsalgorithmus 5.12 CalculateColors3 für den Graphen $\mathcal{G}_A^F$ angeben. Die verschiedenen in dieser Arbeit entwickelten und getesteten Varianten werden durchnumeriert.

Das ist die Variante, deren Heuristik nach [GJP96] im Mittel die geringste Anzahl Farben erzeugen sollte. Dafür ist allerdings nötig, daß am Anfang (Zeilen (4), (7)–(15) und (18)) das eigene Gewicht den Nachbarn mitgeteilt werden muß. Variante 2 versucht, ohne diese Kommunikation auszukommen und verwendet als Gewichtsfunktion nur $w(p) := \mathsf{rand}(p)$. Diesen Wert kann jeder Prozessor für seine Nachbarn selbst ausrechnen, sofern eine Pseudo–Zufallszahlenfunktion wie oben beschrieben verwendet wird. Diese Heuristik entspricht dem Vorgehen in [JP93]. Damit ergibt sich Algorithmus 5.13 CalculateColors2.

Eine triviale dritte Version soll noch eingeführt werden, bei der jeder Prozessor seine Nummer als Farbe erhält. Das läßt zwar keine Parallelisierung der Randbehandlung mehr zu, vermeidet aber den Aufwand für die Graphfärbung und führt vor allem zu einer lexikographischen Verarbeitungsreihenfolge der Teilgebiete für tensorproduktförmige Gebietsaufteilungen, die die geringste Beeinträchtigung für den Markierungsvorgang im Vergleich zum seriellen zu sein verspricht. Das ist der Algorithmus 5.14 CalculateColors1.

Algorithmus 5.12 (CalculateColors3($\mathcal{G}_A^F$))

```
(1)  Nb^p := {q ∈ P | (p,q) ∈ G_A^F}
(2)  w_p := w(p)
(3)  ∀q ∈ Nb^p
(4)        SendASync(q, w_p)
(5)        ReceiveASync(q, w_q)
(6)  R := Nb^p; Q := ∅; S := ∅
(7)  while R ≠ ∅
(8)        ∀q ∈ R
(9)              if Received(q, w_q)
(10)                   R := R \ {q}
(11)                   if w_p < w_q
(12)                         ReceiveASync(q, color_q)
(13)                         Q := Q ∪ {q}
(14)                   else
(15)                         S := S ∪ {q}
(16) ∀q ∈ Q : wait for Received(q, color_q)
(17) color_p := smallest color ∉ color_Q
(18) ∀q ∈ Nb^p : wait for Sent(q, w_p)
(19) ∀q ∈ S : SendASync(q, color_p)
(20) ∀q ∈ S : wait for Sent(q, color_p)
```

Algorithmus 5.13 (CalculateColors2($\mathcal{G}_A^F$))

```
(1)  Nb^p := {q ∈ P | (p,q) ∈ G_A^F}
(2)  w_p := rand(p); Q := ∅; S := ∅
(3)  ∀q ∈ Nb^p
(4)        w_q := rand(q)
(5)        if w_p < w_q
(6)              Q := Q ∪ {q}
(7)              ReceiveASync(q, color_q)
(8)        else
(9)              S := S ∪ {q}
(10) ∀q ∈ Q : wait for Received(q, color_q)
(11) color_p := smallest color ∉ color_Q
(12) ∀q ∈ S : SendASync(q, color_p)
(13) ∀q ∈ S : wait for Sent(q, color_p)
```

Algorithmus 5.14 (CalculateColors1($\mathcal{G}_A^F$))

```
(1)  color_p := p
```

Experimente haben gezeigt, daß selbst Algorithmus CalculateColors3 im Verhältnis zum Rest des Programms nur sehr wenig Zeit benötigt, andererseits durch das heuristische Vorgehen bedingt, nicht die minimal erforderliche Anzahl Farben gefunden wird. Unter diesen Umständen sollte man einige Färbungen mit unterschiedlichen

Gewichtsverteilungen durchführen und eine mit minimaler Farbenzahl letztendlich wählen. Erfahrungen zeigen, daß für unseren Einsatzzweck 10 Färbungen ausprobiert werden sollten. Damit kann man im Vergleich zur 1. Färbung durchschnittlich 1–3 Farben einsparen; dieses Ergebnis streut allerdings sehr breit.

Um im k–ten Versuch unterschiedliche Gewichtsverteilungen zu erhalten, wählen wir für Variante 3

$$w(p) := \deg(p) + \mathsf{rand}((p+1)k)$$

und für Variante 2

$$w(p) := \mathsf{rand}((p+1)k).$$

Damit ergibt sich der Gesamtalgorithmus Coloring zu:

Algorithmus 5.15 (Coloring($type$))

```
(1)  if type = 1
(2)      CalculateColors1
(3)      b_p := color_p
(4)  else
(5)      ConstructColoringGraph G_A^F
(6)      color_max := ∞; b_p := 0
(7)      ∀k = 1, ..., 10
(8)          CalculateColors<type>(G_A^F)
(9)              if max_{p∈P} color_p < color_max
(10)                 b_p := color_p
(11) T := ⋃_{color∈color_P} {p ∈ P | color_p = color}
```

In Zeile (11) wird das Ergebnis, das für die Behandlung der Markierung im Interfacebereich benötigt wird, erzeugt. Jedes Element von $\mathcal{T}$ ist eine Menge von Prozessoren, die in ihrem Interface gleichzeitig den Markierungsalgorithmus ausführen dürfen, weil sie sich nicht gegenseitig beeinflussen können.

Als Bezeichnung werden wir cm1 bis cm3 für die 3 Varianten Coloring(1) bis Coloring(3) verwenden.

Die Laufzeit hängt nicht mehr nur vom gegebenen Graphen ab, sondern entscheidend auch von der Verteilung der Zufallszahlen auf den Knoten. Es sind daher nur noch statistische Aufwandsabschätzungen möglich. Nach [JP93] beträgt der Erwartungswert für die Laufzeit $\mathcal{O}(\log(|\mathcal{P}|)/\log\log(|\mathcal{P}|))$.

5.7 Prolongation und Restriktion

Das Erzeugen der Matrizen erfordert Aufmerksamkeit sowohl hinsichtlich der Erzeugung der parallelen linearen Algebra Datenstruktur für den gröberen Level als

auch bei der erzeugten parallelen Darstellungsform. Schließlich sollen auch noch die Anwendung der Transfermatrizen innerhalb der Restriktion und Prolongation angegeben werden, weil im Parallelen zusätzliche Kommunikationen erforderlich sind, auch wegen des Jacobischrittes für die feinen Knoten.

5.7.1 Erzeugen der parallelen Transfermatrizen

Beim Feinmarkieren eines Knotens wird immer das beste Paar aus seiner Elternliste verwendet, um Einträge in den Transfermatrizen zu erzeugen, mit deren Hilfe der Wert des feinen Knotens aus den Werten von groben Knoten interpoliert werden kann: Das tritt beim Markieren selbst auf (Algorithmus 5.10 pLabel) und im Scatter Callback des Algorithmus 5.9 CommunicateNodeStatus. Im seriellen Fall brauchten wir uns um das Anlegen der Matrixeinträge als Implementierungsdetail nicht weiter zu kümmern. Im Parallelen benötigen wir jedoch grundsätzliche Zusatzüberlegungen.

Zunächst ist das Problem zu lösen, daß im Bereich des Prozessorrandes entstehende Instanzen eines Groblevelknotens wieder als ein verteiltes Objekt zusammengeführt werden müssen. Dazu ist Kommunikation nötig. Würde diese Kommunikation für jede Komponente einzeln ausgeführt, ergäbe sich ein zu feingranularer Ablauf mit schlechter Effizienz. Das Bestreben sollte sein, mit 1 Kommunikation auszukommen.

Der Identifikationsmechanismus in DDD erlaubt dies [Bir98, Indentify Modul]: Nachdem alle lokalen Objekte i' erzeugt sind, werden sie in einem Schritt zu verteilten Objekten zusammengeführt, d. h. es werden die Abbildungen ${}^{\ell-1}\mu$ und ${}^{\ell-1}\alpha$ erstellt, wobei die Einteilung Master/Border vom Elternknoten geerbt wird. Der dazu benötigte Identifikator muß eine Information sein, die genau die jeweils zusammengehörigen Knoten auszeichnet. Im Algorithmus kann dazu der bereits verteilte Elternknoten i benutzt werden.

Algorithmus 5.16 (IdentifyCoarse(ℓ))
(1) $\forall i \in {}^{\ell}\mathcal{C}\ \forall q \in \alpha(i) \setminus \{p\}:$ Identify(i', q, i)

Nun ist zu klären, in welcher parallelen Darstellungsform die Matrizen P und R vorliegen.

Lemma 5.7.1 (${}_1\ddot{\ddot{R}}^{\top}$ und ${}_1\ddot{\ddot{P}}$ sind 1–konsistent verteilt)
Nachdem alle Knoten markiert und alle CommunicateNodeStatus Aufrufe ausgeführt sind, liegen die Transformationsmatrizen in 1–konsistenter Darstellungsform vor.

Beweis: Es sind die 3 Bedingungen (5.13) nachzuweisen.
Sei ein beliebiges $i \in J$ gegeben. $\forall p \in {}_1\psi(i)$ gilt genau einer der beiden Fälle:

a) $i \in \mathcal{F}^p$:
Die Markierung wird mit Elternknoten durchgeführt. Nach (3.14) sind Elternknoten $j \in \mathcal{N}_A(i)$. Da der Überlapp ${}_2\mathcal{U}$ vollständig ist (Algorithmus 5.5 CompleteU) und $i \in {}_1\mathcal{U}^p$, gilt nach (5.10)

$$j \in {}_2\mathcal{U}^p \wedge j \in \mathcal{N}^p(i).$$

j' sei der Knoten j auf dem gröberen Level. Außerdem werden Matrixeinträge $P^p_{ij'}$ angelegt, die für i auch die einzigen sind, weil jeder Knoten nur einmal markiert wird. Damit gilt

$$p \in \alpha(i) \cap \alpha'(j') \neq \emptyset.$$

b) $i \in \mathcal{C}^p$
Für ein $q \in \mathcal{P}$ wurde i grob markiert und auf alle $\mathcal{P} \setminus \{q\}$ über CommunicateNodeStatus kommuniziert. Jeder grob markierte Knoten legt nur einen Matrixeintrag $P^{\alpha(i)}_{ii'} = 1$ zu seinem Groblevelknoten i' an. Damit gilt für $\{j' \in J' \mid P_{ij'} \neq 0\} = \{i'\}$ und

$$\alpha(i) \subseteq \alpha(i) \cap \alpha(i') \neq \emptyset.$$

In beiden Fällen sind somit die Bedingung (5.13b) und wegen Definition 4.1.23 auch (5.13a) nachgewiesen.

Da ein einmal erzeugter Matrixeintrag unverändert an alle Kopien mittels CommunicateNodeStatus verschickt wird, bzw. für grobe Knoten konstant 1 ist, gilt auch die dritte Bedingung (5.13c). Da $R^\top$ per Konstruktion den gleichen Matrixgraph wie P hat, gelten alle Aussagen auch für $R^\top$. □

Bemerkung 5.7.2 (Zu viel oder zu wenig getan?)
Die Matrixeinträge werden im Transfer des Mehrgitteralgorithmus nur bis einschließlich Überlapptiefe 1 benötigt, d. h. es würden 0–konsistente Transfermatrizen genügen. Trotzdem ist die Mehrarbeit, die Transfermatrizen auf der gesamten Knotenmenge zu erzeugen, nicht überflüssig, weil diese zusätzlichen Einträge später bei der Galerkin–Assemblierung noch erforderlich sein werden.

Umgekehrt stellt sich die Frage, ob die Matrixeinträge nicht für die Knoten fehlen, die erst beim CompleteU nach dem rekursiven Aufruf auf dem jetzigen gröberen Level erzeugt werden. Nach dieser Erweiterung benötigt die Zerlegungsphase die angesprochenen Transfermatrizen aber nicht mehr, denn die Galerkin–Assemblierung ist schon abgeschlossen. Erst in der Lösungsphase werden diese Matrizen wieder benutzt, dort reicht aber ihre Existenz auf ${}_1\mathcal{U}$ (wird durch Algorithmus 5.4 CleanOverlap sichergestellt).

5.7.2 Ausführung des Leveltransfers

Für das FAMG besteht der Leveltransfer von Vektoren neben der eigentlichen Interpolation noch aus einem speziellen Glättungsschritt. Beide Teile benötigen parallele Erweiterungen. Der Jacobiglätter für die F–Knoten (seriell siehe Algorithmus 3.5 JacobiFine) wird nach Abschnitt 4.2.7 wie der übliche Jacobiglätter parallelisiert. Die teilweise konsistente Steifigkeitsmatrix, die für die Berechnung der Elternknoten benötigt wurde (Abschnitt 5.4), kann auch in diesem Jacobiglätter verwendet werden, weil ihre Diagonale nach Lemma 4.1.30 konsistent ist.

Algorithmus 5.17 (pJacobiFine($\hat{d}$))

$$(1)\quad \hat{t}_i := \begin{cases} 0 & \text{wenn } i \in \mathcal{C}, \\ \bar{D}_{ii}^{-1}\hat{d}_i & \text{wenn } i \in \mathcal{F}. \end{cases} \qquad \text{nach (4.24)}$$

Das Hauptproblem ist, daß die Transfermatrix *nicht* konsistent, sondern nur 1–konsistent (Lemma 5.7.1) ist. Man könnte zwar versuchen, die fehlenden Borderknoten auf dem gröberen Level beim Anlegen der Transfermatrix zu erzeugen, aber dafür existieren keine Elternknoten auf dem feinen Level, die quasi als Ankerpunkt dienen könnten; also ist diese Art der Erweiterung nicht möglich. Dieses Manko der fehlenden Konsistenz muß durch zusätzliche Kommunikation während der linearen Algebra Berechnungen kompensiert werden. Trotzdem benötigt man noch die Voraussetzung, daß die Transfermatrix wenigstens in 0–konsistenter Form vorliegt, was nach Lemma 5.7.1 und Lemma 5.3.5 hier zutrifft. Das Mehr von 0-konsistent zu 1-konsistent kann hier leider nicht gewinnbringend eingesetzt werden.

Zunächst wollen wir die Restriktion des additiven Defektvektors betrachten. Vergleiche dazu auch Algorithmus 5.18 und Abbildung 5.4. Da nicht zu allen fein markierten Knoten alle Elternknoten auf dem gröberen Level vorhanden sind (in der Abbildung fehlt z. B. dem Knoten j auf q sein Elternknoten j'), würden bei der üblichen Matrixmultiplikation ${}^{\ell-1}\hat{d} := {}^{\ell}R\,{}^{\ell}\hat{d}$ die entsprechenden Anteile (hier $R_{i'j}d_j$) verloren gehen. Man kann ohne Kommunikation auskommen und diesen Fehler akzeptieren, aber er ist schwer zu kontrollieren, weil er sehr stark von der konkreten Verteilungssituation abhängt. Wir bemühen uns um ein exaktes Vorgehen: Der additive Vektor $\hat{d}$ wird in die eindeutige Darstellungsform $\check{d}$ transformiert (Schritt (5)). Danach steht der globale Wert d_i in $\check{d}_i^p$ für $p = \mu(i)$. Da für i auf seinem Masterprozessor aber *alle* Elternknoten vorhanden sind, fehlen im additiven Ergebnis ${}^{\ell-1}\hat{d} := {}^{\ell}_{0}\ddot{R}\,{}^{\ell}\check{d}$ keine Anteile mehr (Schritt (6)). Formal ist die Korrektheit durch Lemma 5.7.1 und Operation 5.15 gesichert.

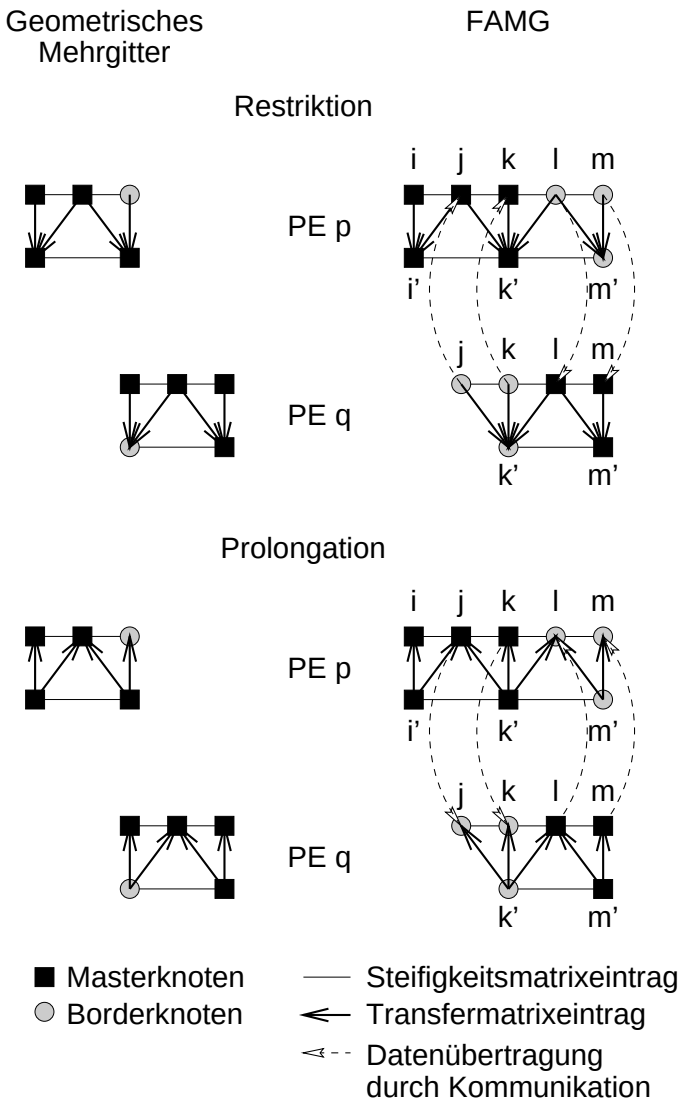


Abbildung 5.4: Vergleich des Transfers beim geometrischen Mehrgitterverfahren und beim parallelen FAMG

Algorithmus 5.18 (pFAMGRestriction($^{\ell}\hat{d}$, $^{\ell-1}\hat{d}$, $^{\ell}\bar{c}$))

(1) $^{\ell}\hat{t} := \mathsf{pJacobiFine}(^{\ell}\hat{d})$

(2) $^{\ell}\bar{\boldsymbol{t}} :=$ **MakeAdditiveVectorConsistent**$(^{\ell}\hat{\boldsymbol{t}})$

(3) $^{\ell}\hat{d} := {}^{\ell}\hat{d} - {}^{\ell}\hat{A}\,{}^{\ell}\bar{t}$ nach (4.27) und (4.18)

(4) $^{\ell}\bar{c} := {}^{\ell}\bar{c} + {}^{\ell}\bar{t}$ nach (4.17)

(5) $^{\ell}\check{\boldsymbol{d}} :=$ **MakeAdditiveVectorUnique**$(^{\ell}\hat{\boldsymbol{d}})$

(6) $^{\ell-1}\hat{\boldsymbol{d}} := {}^{\ell}_{0}\ddot{\boldsymbol{R}}\,{}^{\ell}\check{\boldsymbol{d}}$ nach Lemma 5.7.1und (5.15)

Auch bei der Prolongation der konsistenten Lösungskorrektur erzwingt die nur 0–konsistente Transfermatrix zusätzlichen Aufwand. Vergleiche dazu auch Algorithmus 5.19 und Abbildung 5.4. Auch hier fehlen die gleichen Knoten wie bei der Restriktion besprochen. Die Lösung hier wird sein, die Groblevel Lösungskorrektur ${}^{\ell-1}\bar{t}$ zunächst nur auf die Masterknoten des feineren Levels ${}^{\ell}\ddot{\ddot{t}}$ zu interpolieren (Schritt(1)). Das geht vollständig, weil für diese Knoten *alle* Elternknoten vorhanden sind. Für den Borderknoten j auf q würde wieder der Anteil $P_{ji'}\, t_{i'}$ fehlen. Die Ergebnisse auf den Borderknoten (soweit überhaupt berechnet, siehe Bemerkung 5.3.8) sind unbrauchbar und müssen mittels Kommunikation durch die korrekten Werte auf ihren Mastern ersetzt werden (Schritt (2)). Lemma 5.7.1 und Operation 5.14 sichern formal die Korrektheit des Ergebnisses.

Algorithmus 5.19 (pFAMGProlongation(${}^{\ell-1}\bar{c}$, ${}^{\ell}\bar{t}$, ${}^{\ell}\hat{d}$, ${}^{\ell}\bar{c}$ **))**

(1)	$\boldsymbol{{}^{\ell}\ddot{\ddot{t}} := {}^{\ell}_{0}\ddot{\boldsymbol{P}}\, {}^{\ell-1}\bar{\boldsymbol{c}}}$	nach (5.14)
(2)	$\boldsymbol{{}^{\ell}\bar{t}}$ **:= MakeMasterconsistentVectorConsistent(** $\boldsymbol{{}^{\ell}\ddot{\ddot{t}}}$ **)**	
(3)	${}^{\ell}\hat{d} := {}^{\ell}\hat{d} - {}^{\ell}\hat{A}\, {}^{\ell}\bar{t}$	nach (4.27) und (4.18)
(4)	${}^{\ell}\bar{c} := {}^{\ell}\bar{c} + {}^{\ell}\bar{t}$	nach (4.17)
(5)	${}^{\ell}\hat{t} :=$ pJacobiFine(${}^{\ell}\hat{d}$)	
(6)	$\boldsymbol{{}^{\ell}\bar{t}}$ **:= MakeAdditiveVectorConsistent(** $\boldsymbol{{}^{\ell}\hat{t}}$ **)**	
pMG(8)	${}^{\ell}\hat{d} := {}^{\ell}\hat{d} - {}^{\ell}\hat{A}\, {}^{\ell}\bar{t}$	
pMG(9)	${}^{\ell}\bar{c} := {}^{\ell}\bar{c} + {}^{\ell}\bar{t}$	

Im Gegensatz zum geometrischen Mehrgitterverfahren benötigen wir hier also pro Transferschritt zwei Kommunikationen.

5.8 Galerkin–Assemblierung

Die Grobgittermatrix wird mittels Galerkin–Assemblierung berechnet

$$ {}^{\ell-1}A := {}^{\ell}_{1}\ddot{R}\, {}^{\ell}A\, {}^{\ell}_{1}\ddot{P}. \tag{5.29} $$

Fraglich ist noch, ob für ${}^{\ell}A$ die additive oder die teilweise konsistente Darstellungsform gewählt werden soll. Es wird sich zeigen, daß die teilweise konsistente Form $\tilde{A}$ die effizientere Wahl ist. Man könnte auch ${}^{\ell}A$ in die eindeutig additive Form transformieren und dafür dann eine vollständige Matrixmultiplikation durchführen; dazu wird aber eine zusätzliche Kommunikation benötigt, die das hier von uns entwickelte Vorgehen einspart. Klar ist auf jeden Fall, daß das Ergebnis ${}^{\ell-1}A$ additiv sein soll, weil es die Steifigkeitsmatrix für den gröberen Level ist (Voraussetzung (5.4)).

Um den Levelindex weglassen zu können, definieren wir für diesen Abschnitt

$$R := {}^{\ell}_{1}\dddot{R} \tag{5.30a}$$
$$A := {}^{\ell}\hat{A} \tag{5.30b}$$
$$P := {}^{\ell}_{1}\dddot{P} \tag{5.30c}$$
$$G := {}^{\ell-1}\hat{A}. \tag{5.30d}$$

Da alle Matrizen dünn besetzt sind, bietet es sich an, das Galerkinprodukt (5.29) in folgender Form komponentenweise zu formulieren

$$\forall i', j' \in J' : \; G_{i'j'} := \sum_{s \in \mathcal{N}_R(i')} R_{i's} H_{sj'} \tag{5.31}$$

mit

$$H_{sj'} := \sum_{t \in \mathcal{N}_A(s) \wedge P_{tj'} \neq 0} A_{st} P_{tj'}. \tag{5.32}$$

Anschaulich bedeutet das, daß man über alle *gerichteten* Wege im Matrixgraph $\mathcal{G}(R) \cup \mathcal{G}(A) \cup \mathcal{G}(P)$ summiert, die von $i' \in J'$ nach $j' \in J'$ führen und über genau 2, nicht notwendigerweise verschiedene Punkte $s, t \in J$ verlaufen und entlang denen die Matrixeinträge multipliziert werden. Ein solcher Weg ist in Abbildung 5.5 dargestellt. Entscheidend für die Korrektheit der Parallelisierung ist nun, daß jeder solche Weg (i', s, t, j') genau einmal im verteilten Ergebnis G vorhanden ist, damit G eine additive Darstellung ist.

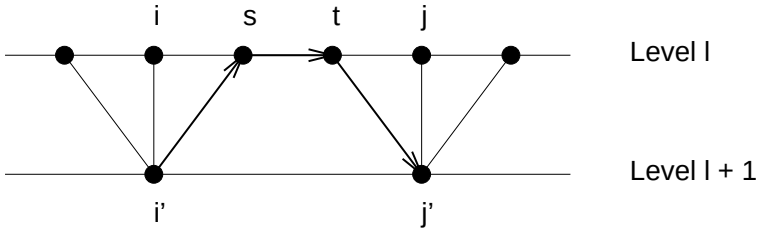


Abbildung 5.5: Weg, auf dem ein Eintrag in die Galerkinmatrix erzeugt wird

Die Idee ist, jedem Weg einen eindeutigen Knoten zuzuordnen, der als Verantwortlicher dafür betrachtet werden kann, daß dieser Weg im Ergebnis $\hat{G}$ enthalten ist. Da jeder Knoten des Weges im allgemeinen mehrere Kopien besitzen kann, aber nicht muß, ist es naheliegend, den Masterknoten (der auf jeden Fall existiert) zu wählen.

Betrachtet man die übliche Programmierung eines Matrixproduktes, wird die Wahl des Verantwortlichen auf einen der beiden Endknoten eines Weges fallen. Sei das

o. B. d. A. der Anfangsknoten i'. Er liegt in der Kernpartition, da er eindeutig sein muß. Damit liegt der Teilweg (i,s) in $\mathcal{N}(\mathcal{M}) = {}_1\mathcal{U}$, (i,s,t) ist in $\mathcal{N}({}_1\mathcal{U}) = {}_2\mathcal{U}$ enthalten und (i',s,t,j') wäre in $\mathcal{N}({}_2\mathcal{U}) = {}_3\mathcal{U}$ enthalten. Da aber ${}_3\mathcal{U}$ mit vertretbarem Aufwand nicht erzeugt werden kann (Abschnitt 5.3.3), ist dieses Vorgehen unbrauchbar.

Als Verantwortlicher kann aber der zweite Knoten eines Weges definiert werden: s ist wegen der Eindeutigkeit als Master zu wählen. Dann ist der Teilweg (i,s,t) in $\mathcal{N}(\mathcal{M}) = {}_1\mathcal{U}$ und der gesamte Weg ist in $\mathcal{N}({}_1\mathcal{U}) = {}_2\mathcal{U}$ enthalten, also existiert er wegen der Vollständigkeit des Überlapps der Tiefe 2 und der 1–Konsistenz der Transfermatrix nach Lemma 5.7.1.

Aus diesen Überlegungen ergibt sich folgender Algorithmus:

Algorithmus 5.20 (pAssembleGalerkinMatrix(ℓ))
Voraussetzung: ${}_2\mathcal{U}$ vollständig, $\tilde{A}$ teilweise konsistent, ${}_1\ddot{\ddot{P}}$ und ${}_1\ddot{\ddot{R}}$ 1–konsistent
(1) $\forall s \in {}^{\ell}\mathcal{M}^p$
(2) $\forall i' \in \mathcal{N}_R^p(s)$
(3) $\forall t \in \mathcal{N}_A^p(s)$
(4) $\forall j' \in \mathcal{N}_P^p(t)$
(5) ${}^{\ell-1}\hat{A}_{i'j'} := {}^{\ell-1}\hat{A}_{i'j'} + {}^{\ell}_1\ddot{\ddot{R}}_{i's}\,{}^{\ell}\tilde{A}_{st}\,{}^{\ell}_1\ddot{\ddot{P}}_{tj'}$

Ob dieser Algorithmus wirklich eine additive Darstellungsform der Galerkinmatrix berechnet, ist nur durch einen Beweis festzustellen.

Beweis: Nachzuweisen ist, daß das Ergebnis $({}^{\ell-1}A)^p_{p\in\mathcal{P}}$ die 3 Bedingungen (4.14) erfüllt. Es sollen wieder die vereinfachten Bezeichnungen aus (5.30) verwendet werden.

a) **Behauptung:** Der parallele Algorithmus bearbeitet genau die gleichen Anfangswegstücke (i,s) wie der serielle Algorithmus in (5.31).

Nach (5.31) enthält das globale Ergebnis

$$\forall i' \in J' : \sum_{s\in\mathcal{N}_R(i')} R_{i's} H_{sj'}$$

genau die Anfangswegstücke aus

$$\mathcal{W} := \{(i',s) \in J' \times J \mid s \in \mathcal{N}_R(i')\}. \tag{5.33}$$

Da nach Lemma 4.1.20 $\mathcal{N}$ symmetrisch ist, kann $\mathcal{W}$ äquivalent als

$$\mathcal{W} = \{(i',s) \in J' \times J \mid i' \in \mathcal{N}_R(s)\}$$

geschrieben werden. Auf Grund der Eindeutigkeit des Masterknotens nach Lemma 4.1.7 können alle $s \in J$ durch

$$\forall p\ \forall s \in {}^{\ell}\mathcal{M}^p$$

aufgezählt werden. Das geschieht in Zeile (1).

Da nach Lemma 5.7.1 R 1–konsistent und damit nach Lemma 5.3.5 0–konsistent ist und nach Voraussetzung $s \in \mathcal{M}$ ist, d. h.

$$p \in {}_0\psi(s), \tag{5.34}$$

besitzt s nach (5.13c) *alle* globalen Nachbarn auf Prozessor p

$$\mathcal{N}_R^p(s) = \mathcal{N}_R(s).$$

Daher ist Schritt (1) und (2)

$$\forall p \ \forall s \in {}^{\ell}\mathcal{M}^p \ \forall i' \in \mathcal{N}_R^p(s)$$

eine parallele, lokal ausführbare Enumeration der Menge $\mathcal{W}$ und Behauptung a) ist bewiesen.

b) **Behauptung:** Der parallele Algorithmus bearbeitet genau die gleichen Restwegstücke (s, t, j') wie der serielle Algorithmus in (5.31).

Da nach Teilalgorithmus 5.2 CompleteU1 in Algorithmus 5.5 CompleteU der Überlapp der Tiefe 1 vollständig ist, und $s \in \mathcal{M}^p$ nach a), hat s auf Prozessor p nach (5.10) *alle* Nachbarn t

$$\mathcal{N}_A^p(s) = \mathcal{N}_A(s), \tag{5.35}$$

d. h. alle Wegstücke (s, t) sind seriell wie parallel in Schritt (3) identisch.

Nach (5.6) und (5.7) liegt t in ${}_1\mathcal{U}^p$, d. h.

$$p \in {}_1\psi(t). \tag{5.36}$$

Da P nach Lemma 5.7.1 1–konsistent ist, folgt mit (5.13c)

$$\mathcal{N}_P^p(t) = \mathcal{N}_P(t),$$

d. h. in Schritt (4) besitzen alle t auf Prozessor p *alle* Nachbarn j'. Deshalb sind seriell wie parallel alle Endwegstücke (t, j') identisch und Behauptung b) ist bewiesen.

c) **Behauptung:** Seriell wie parallel werden genau die gleichen Wege (i', s, t, j') beschritten, die ihren jeweiligen Beitrag $R_{i's}A_{st}P_{tj'}$ zum Endergebnis beitragen.

Das ergibt sich aus der Zusammenfassung von a) und b).

d) **Behauptung:** Seriell wie parallel werden die gleichen Beiträge $R_{i's}A_{st}P_{tj'}$ berechnet.

R und P sind nach Lemma 5.7.1 1–konsistent, und da nach (5.34) $p \in {}_0\psi(s) \subset {}_1\psi(s)$ und nach (5.36) $p \in {}_1\psi(t)$ gilt, folgt aus der 1–Konsistenz nach Definition 5.3.4 (5.13c), daß in Schritt (5)

$$\begin{aligned} R^p_{i's} &= R_{i's} \text{ und} \\ P^p_{tj'} &= P_{tj'} \end{aligned}$$

jeweils den globalen, konsistenten Wert besitzt. Wählt man nun noch für A die teilweise konsistente Darstellungsform (wie sie schon für die Auswahl der Elternknoten in Abschnitt 5.4 benötigt wurde), so gilt wegen der Vollständigkeit des Überlapps der Tiefe 2 nach Algorithmus 5.5 CompleteU und $p \in \alpha(s) \cap \alpha(t)$ (siehe (5.35)) nach (4.13c) in Schritt (5)

$$\tilde{A}^p_{st} = A_{st},$$

d. h. auch die Anteile der Steifigkeitsmatrix im Galerkinprodukt sind seriell wie parallel identisch. Damit ist Behauptung d) bewiesen.

e) **Behauptung:** Das Endergebnis $\hat{G}$ ist eine additive Darstellung des Galerkinprodukts ${}^\ell R\, {}^\ell A\, {}^\ell P$.

Es müssen die beiden Bedingungen (4.14) in Definition 4.1.25 nachgeprüft werden. Sei $i', j' \in J'$ so, daß $G_{i'j'} \neq 0$. Dann existiert mindestens ein Weg (i', s, t, j'), entlang dem ein Anteil zu $G_{i'j'}$ berechnet wird. Nach Teil c) gibt es einen Prozessor p, der diesen Weg enthält (speziell also $p \in \alpha'(i') \cap \alpha'(j')$) und einen Eintrag $G_{i'j'}$ anlegt. Daher respektiert α' per Konstruktion den Matrixgraphen $\mathcal{G}(G)$ und (4.14a) ist bewiesen.

Jeder Eintrag $G_{i'j'}$ des Galerkinprodukts besteht nach (5.31) aus der Summe von Anteilen, die entlang eines jeweils eindeutigen Weges gebildet werden. Nach Teil c) existiert zu jedem solchen Weg *genau ein* Prozessor p, der für ihn verantwortlich ist, und nach (5) wird dort dieses Teilergebnis in die Matrix G^p eingetragen, das nach d) den gleichen Wert wie im globalen Ergebnis besitzt; es gibt keine andere Möglichkeit, wie ein Matrixeintrag $G^p_{i'j'}$ erzeugt wird. Daher wird in

$$\sum_{p \in \alpha'(i') \cap \alpha'(j')} G^p_{i'j'}$$

genau über die verschiedenen Wege und damit Teileinträge summiert wie bei serieller Rechnung und das globale, konsistente Endergebnis $G_{i'j'}$ erhalten, was (4.14) beweist.

□

5.9 Grobgitter

Um das Verfahren rekursiv anwenden zu können, muß sichergestellt sein, daß die Voraussetzungen aus Abschnitt 5.2 auch für den eben erzeugten, gröberen Level wieder zutreffen. Die durch die Galerkin–Assemblierung erzeugte Matrix besitzt nun schon die nötigen Eigenschaften (vergleiche Abschnitt 5.8), so daß hier nichts mehr zu tun ist.

5.10 Baselevellöser

Nachdem die Hierarchie von Matrizen bis zur gewünschten Grobheit erzeugt ist, muß nun vorausschauend an das später benötigte Lösen des Gleichungssystems auf dem gröbsten Level gedacht werden. Es stehen prinzipiell die zwei Möglichkeiten zur Auswahl, direkt oder iterativ zu lösen, wie in Abschnitt 4.2.10 diskutiert.

Als iterative Löser kommen nur solche in Frage, die auf einem einzigen Level arbeiten, um die Problemstellung eines parallelen Mehrgitterverfahrens nicht rekursiv wieder aufzuwerfen (vergleiche die Ausführungen in Abschnitt 2.3.5). Erfahrungen mit einem seriellen Mehrgitterverfahren auf dem agglomerierten Baselevel liegen noch nicht vor.

Als Alternative stehen direkte Löser zur Verfügung. Wir nutzen die vergleichsweise kleine Knotenzahl aus und agglomerieren den Baselevel, um so auf dem Masterprozessor einen seriellen, direkten Löser einsetzen zu können.

Bei der Entscheidung über die geeignete Methode spielt natürlich die Knotenzahl und das Besetzungsmuster der Matrix eine erhebliche Rolle. Eine Verringerung der Knotenzahl beschleunigt den Löser auf dem gröbsten Level, erfordert aber dafür entsprechend mehr Vergröberungsschritte. Die Entscheidung hängt sehr stark von der zur Verfügung stehenden Implementierung ab und kann nur experimentell vorgenommen werden, da wir uns im *nicht–asymptotischen* Bereich befinden, in dem theoretische Aussagen nicht anwendbar sind.

5.11 Übersicht

Nun ist die Parallelisierung aller Komponenten des FAMG vorgestellt worden. Hier sollen sie zum Gesamtalgorithmus zusammengestellt werden.

5.11.1 Aufbau der Gitterhierarchie

Algorithmus 5.21 (pFAMGConstruct(^0A))

(1) for $\ell := 0$ step -1
(2) $f :=$ lokales Abbruchkriterium
(3) **$f_g :=$ Communicate(f)**
(4) if(f_g) break
(5) **CompleteU($^\ell\hat{A}$)**
(6) **Coloring(cm_type)**
(7) **$^\ell\hat{W} := {}^\ell\hat{A}$**
(8) **$^\ell\tilde{W} :=$ MakeMatrixPartlyConsistent($^\ell\hat{W}$)**
(9) $\forall i \in {}^\ell\mathcal{M}^p$ Calculate($\mathbb{P}_i^+$)
(10) **pLabel($^\ell\mathcal{M}$)**
(11) **IdentifyCoarse(ℓ)**
(12) **pAssembleGalerkinMatrix(ℓ)**
(13) **Baselevelagglomeration($^{\ell_{min}}\hat{A}$)**

Bis auf die Bestimmung der möglichen Elternknoten in Schritt (9), dem zentralen Algorithmus des FAMG Verfahrens, mußten alle anderen Komponenten für die Parallelisierung angepaßt werden.

5.11.2 Mehrgitterlöser

Algorithmus 5.22 (pFAMG($\bar{c}$, $\hat{d}$, ℓ))

(1) if($\ell = \ell_{min}$)
(2) **BaseSolver($^\ell\bar{c}$, $^\ell\hat{d}$)**
else
(3) for $i := 1$ to ν_1: **Smooth($^\ell\bar{c}$, $^\ell\hat{d}$)**
(4) **pFAMGRestriction($^\ell\hat{d}$, $^{\ell-1}\hat{d}$, $^\ell\bar{c}$)**
(5) $^{\ell-1}\bar{c} := 0$ nach (4.15)
(6) for $i := 1$ to γ: **pFAMG($\bar{c}, \hat{d}, \ell - 1$)**
(7) **pFAMGProlongation($^{\ell-1}\bar{c}$, $^\ell\bar{t}$, $^\ell\hat{d}$, $^\ell\bar{c}$)**
(8) $^\ell\hat{d} := {}^\ell\hat{d} - {}^\ell\hat{A}\,{}^\ell\bar{t}$ nach (4.27), (4.18)
(9) $^\ell\bar{c} := {}^\ell\bar{c} + {}^\ell\bar{t}$ nach (4.17)
(10) for $i := 1$ to ν_2: **Smooth($^\ell\bar{c}$, $^\ell\hat{d}$)**

6 Numerische Experimente

6.1 Ziele

Nachdem das gesamte parallele Verfahren entwickelt ist, interessiert uns nun, wie effizient es ist. Da theoretische Untersuchungen dazu keinen Beitrag leisten können, müssen gezielte, numerische Experimente durchgeführt werden, um wenigstens Anhaltspunkte über das Verhalten des Verfahrens zu gewinnen.

Grundsätzlich unterscheiden wir zwei Aspekte: die *numerische Güte*, die sich als Anzahl benötigter Iterationsschritte bzw. als Konvergenzrate messen läßt, und die benötigte *Laufzeit*, insbesondere ihre *Skalierbarkeit*.

Die untersuchten Einflußfaktoren werden sein: der Gleichungstyp und wesentliche Parameter, die Anzahl der Knoten der gegebenen Ausgangsmatrix (Gitterfeinheit), die Anzahl der verwendeten Prozessoren, die Gebietsaufteilung und die Graphfärbungsmethode.

Ziel ist es, die Robustheit der Güte gegen Gleichungstyp, Gleichungsparameter, Gitterfeinheit und Prozessoranzahl exemplarisch nachzuweisen.

Wir beschränken uns auf die Untersuchung von Modellproblemen, weil anhand dieser einzelne Aspekte herausgearbeitet werden können und nur so ein Vergleich mit anderen Verfahren möglich ist. Dazu sind realitätsnahe, komplexe Anwendungen nicht geeignet.

6.2 Kennzahlen

Zur Vereinfachung werden die Symbole der Tabelle 6.1 verwendet.

Dabei gelten folgende Definitionen:

Definition 6.2.1 (Komplexitäten)
Es seien folgende Komplexitäten definiert.
Operatorkomplexität

$$c_A := \frac{\sum_\ell {}^\ell m}{{}^0 m} \tag{6.1}$$

h	Gitterweite des feinsten, gegebenen Gitters
${}^{\ell}n$	Anzahl der Masterknoten auf Level ℓ, summiert über alle Prozessoren
${}^{\ell}b$	Anzahl der Borderknoten auf Level ℓ, summiert über alle Prozessoren
${}^{\ell}m$	Anzahl der Matrixeinträge auf Level ℓ, summiert über alle Prozessoren
N	Anzahl der Masterknoten auf dem feinsten gegebenen Gitter ${}^{0}n$
N_{Ba}	Anzahl der Masterknoten auf dem gröbsten Level ${}^{\ell_{min}}n$
ϵ	Anisotropieparameter der Poissongleichung bzw. Diffusionsstärke in der Konvektions–Diffusions–Gleichung
PEC	Pecletzahl für die Konvektions–Diffusions–Gleichung
IT	Anzahl der Iterationsschritte für eine relative Defektreduktion von 10^{-8}
k	durchschnittliche Konvergenzrate für eine relative Defektreduktion von 10^{-8} nach Definition 2.3.1
T_{FA}	Zeit für den Aufbau der FAMG Hierarchie
T_{IT}	Zeit für 1 Iterationsschritt des Mehrgitterlösers
T_{SOL}	Zeit des Mehrgitterlösers ($= IT * T_{IT}$)
T_{Ges}	Gesamtlaufzeit ($= T_{FA} + T_{SOL}$)
c_A	Operatorkomplexität
c_G	Gitterkomplexität
$c_{\ddot{U}}$	Überlappkomplexität
B/M	Verhältnis Border– zu Masterknoten
S	Speedup
$\bar{E}$	Speedup Effizienz
E	Scaleup Effizienz
T_X^*	Zeit aufgrund der Modellannahme (6.8)
E_X^*	Effizienz aufgrund der Modellannahme (6.8)

Tabelle 6.1: Verwendete Symbole in Kapitel 6

Gitterkomplexität

$$c_G := \frac{\sum_{\ell} {}^{\ell}n}{{}^{0}n} \tag{6.2}$$

Überlappkomplexität

$$c_{\ddot{U}} := \frac{\sum_{\ell} {}^{\ell}b}{{}^{0}b} \tag{6.3}$$

Verhältnis Border– zu Masterknoten

$$B/M := \frac{\sum_{\ell} {}^{\ell}b}{\sum_{\ell} {}^{\ell}n} \tag{6.4}$$

Die Gitterkomplexität c_G drückt den Speichermehraufwand aus, den das Mehrgitterverfahren im Vergleich zu einem 1-Gitterverfahren verursacht. Man kann auch sagen,

daß es der Speicheraufwand ist, den das Mehrgitterverfahren über die pure Speicherung der Lösung hinaus benötigt, um eine optimale Laufzeit zu gewährleisten. Sie sollte für eine parallele Rechnung nicht wesentlich höher als für eine serielle sein, weil sich die Vergröberungsrate durch die Parallelisierung nicht verschlechtern sollte.

Die Überlappkomplexität $c_{\ddot{U}}$ gibt an, wie stark der Überlapp auf den gröberen Leveln ansteigt. Sie wird deutlich über 1 liegen, weil das Verhältnis Oberfläche zu Volumen einer Partition für gröbere Level immer schlechter wird.

Die Operatorkomplexität c_A faßt die beiden Effekte der Vergröberungsleistung und des Überlappwachstums zusammen und stellt damit eine starke inhaltliche Erweiterung gegenüber dem seriellen Fall dar. Sie ist besonders wichtig, weil der Großteil des Speichers von den Matrixeinträgen beansprucht wird, sie also der limitierende Faktor ist. Eine Operatorkomplexität nicht größer als 2 ist nach [Stü99b, Seite 17] auch für die Industrietauglichkeit ausschlaggebend. Darüber hinaus bestimmt die Anzahl der Matrixeinträge auch die Laufzeit entscheidend mit, weil die zeitaufwendigsten Algorithmusteile neben der Kommunikation die Matrixoperationen sind.

Schließlich drückt B/M den durch die Parallelisierung direkt verursachten Speichermehrbedarf aus.

Um die Effizienz eines Verfahrens anzugeben, bedienen wir uns folgender, weit verbreiteter Effizienzbegriffe.

Definition 6.2.2 (Effizienz, Speedup, Scaleup)
Es sei $T_X(p,n)$ die Zeit, die ein Verfahren oder eine Verfahrenskomponente X benötigt, um auf p Prozessoren die Aufgabe der Größe n zu lösen. Dann heißt

$$S_X(p,n) := \frac{T_X(1,n)}{T_X(p,n)} \tag{6.5}$$

der **Speedup**,

$$\bar{E}_X(p,n) := \frac{S_X(p)}{p} = \frac{T_X(1,n)}{pT_X(p,n)} \tag{6.6}$$

die **Speedup Effizienz** und

$$E_X(p,n) := \frac{T_X(1,n)}{T_X(p,pn)} \tag{6.7}$$

die **Scaleup Effizienz**.

Die **Speedup** Betrachtung geht von einer fixen Problemgröße aus und bewertet, wieviel schneller dieses Problem auf mehreren Prozessoren gelöst werden kann. Skaliert das Verfahren ideal, d. h. ist $T_X(\cdot,n)$ linear, ergibt sich eine Effizienz $\bar{E}_X(p,n) = 1$. In der Praxis sind aber mit dem Einsatz mehrerer Prozessoren zusätzliche Kommunikationszeiten, Mehrberechnungen im Überlapp und Lastungleichgewichte verbunden, die

zu einer unterproportionalen Zeitreduktion führen, weil die größte Teillast das Zeitverhalten dominiert. Dadurch sinkt die Effizienz unter 1. Das **Amdahlsche Gesetz** läßt gar den Einsatz vieler Prozessoren entsprechend dieser Sichtweise als unsinnig erscheinen [Amd67].

Die **Scaleup** Betrachtung geht davon aus, daß man mit mehreren Prozessoren auch größere Probleme lösen möchte und hält daher die Last pro Prozessor konstant. Es treffen die gleichen Ungunstfaktoren wie beim Speedup zu; jedoch wirken sie sich nicht so stark aus, weil sich das Verhältnis Rechenarbeit zu Kommunikationszeit wesentlich weniger stark verschlechtert. Die Scaleup Betrachtung entspricht auch dem überwiegenden Einsatzzweck von parallelen Differentialgleichungslösern: Die enorm großen Knotenzahlen können nur durch Parallelrechner bewältigt werden. Zu beachten ist, daß aufgrund der regelmäßigen Verfeinerung zur Erzeugung der großen Rechenbeispiele in 2D die Prozessorzahl ebenfalls mit einem Faktor 4 skaliert. Daher stehen nur für 4, 16, 64 Prozessoren serielle Vergleichszeiten zur Verfügung. Für 8, 32 und 128 Prozessoren beziehen wir uns immer auf die Zeiten, die 2 Prozessoren benötigen.

6.3 Parameter für die Rechnungen

Für alle nachfolgend präsentierten Rechnungen werden, soweit nichts anderes erwähnt wird, folgende Einstellungen verwendet.

Die Gebietsaufteilung für strukturierte Gitter wird mit einer speziellen Lastverteilung vorgenommen, um Box– und Streifenaufteilungen gezielt zu erzeugen (siehe auch Abschnitt 6.4.1). Unstrukturierte Gitter werden mit dem RCB Verfahren (recursiv coordinate bisection) verteilt.

Das FAMG wird mit folgenden Einstellungen ausgeführt:

σ	$= 0.1$	Abschneideschwelle für Matrixeinträge in (3.9)
ω	$= 0.7$	Dämpfungsparameter für Glätter zur Elternauswahl in (3.10) und (3.11)
θ	$= 0.9$	Schwelle für sehr gute Elternpaare in (3.16)
α^e	$= 1$	Wichtungsfaktor für Anteil der neuerzeugten Matrixeinträge in (3.17)
α^c	$= 10$	Wichtungsfaktor für Anteil der neuerzeugten Grobgitterknoten in (3.17)
δ	$= 10^{20}$	für 2–elementige Elternmengen, d. h. alle Vorschläge werden bzgl. (3.15) akzeptiert
δ	$= 10^{-10}$	für 1-elementige Elternmengen, d. h. praktisch keine solchen Mengen werden bzgl. (3.15) akzeptiert
cm3		Färbungsmethode, die im Mittel am wenigsten Farben erzeugt.

Als Testvektor wird der Vektor $t := (1, 1, \ldots, 1)^\top$ verwendet. Es wird keine weitere Vergröberung berechnet, wenn auf einem Level weniger als 5000 Masterknoten

existieren ($N_{Ba} \leq 5000$) oder die über die Prozessoren maximierte mittlere Vergröberungsrate unter 1.25 sinkt.

Für den Mehrgitterlöser (Algorithmus 5.22) wird ein V–Zyklus ($\gamma := 1$ in Zeile (6)) mit einem Vor– und einem Nachglättungsschritt ($\nu_1 := \nu_2 := 1$ in Zeile (3) und (10)) eines mit $\omega = 0.85$ gedämpften Jacobiglätters verwendet. Als Baselevellöser wird ein direkter Löser mit Bandbreitenoptimierung eingesetzt; dazu ist eine Grobgitteragglomeration nach Abschnitt 5.10 nötig. Dieses Vorgehen in Verbindung mit einem Vergröberungsende bei weniger als 5000 Knoten erwies sich in hier nicht dokumentierten Tests als das Schnellste. Die Mehrgitteriteration wird beendet, wenn der Anfangsdefekt um 10^{-8} reduziert wurde.

Zwar lassen sich für jede einzelne Gleichung noch bessere Parameterwerte finden, aber mit den hier vorgestellten ließen sich gleichmäßig gute Ergebnisse erzielen. Die Einheitlichkeit entspricht auch dem Wunsch, einen „black box“ Löser zu definieren.

Als Parallelrechner wurde eine IBM RS/6000 SP verwendet. Es ist ein MIMD Rechner und im Regelbetrieb standen bis zu 128 Prozessoren zur Verfügung. Ein Rechenknoten ist mit 512 MB Hauptspeicher ausgerüstet, mit 120 MHz getaktet und kommuniziert mittels US–Protokoll. Für die Rechnungen war die exklusive Nutzung der Rechenknoten sichergestellt, so daß die Zeitmessungen nicht verfälscht wurden. Trotzdem weisen die Laufzeiten Schwankungen von bis zu 10% auf. Neben dem nicht deterministischen Ablaufverhalten der einzelnen Nachrichten trägt dazu auch die Tatsache bei, daß jeder einzelne Rechenknoten mit einem vollständigen UNIX Betriebssystem betrieben wird, das ebenfalls nicht deterministisch interne Prozesse ausführt.

6.4 Isotroper Laplace Operator

Ziel dieser ersten Untersuchung soll es sein, grundlegende Eigenschaften der entwickelten Parallelisierung darzustellen. Als Modellproblem wird der isotrope Laplace Operator verwendet, weil er als ein Musterbeispiel einer elliptischen partiellen Differentialgleichung weit verbreitet ist und dafür das geometrische Mehrgitterverfahren optimal geeignet ist. Gerade deshalb ist der isotrope Laplace Operator aber ein unerwartet schwieriges Testproblem für jedes algebraische Mehrgitterverfahren: Beim geometrischen Mehrgitterverfahren stellt die regelmäßige Gitterverfeinerung umgekehrt automatisch die optimalen Grobgitter bereit — die möglichst regelmäßigen Grobgitter muß allerdings ein algebraisches Verfahren selbst konstruieren, will es vergleichbar gute Konvergenzeigenschaften erzielen.

Da jeweils Rechengebiete ohne einspringende Ecken, als Rechte Seite 0 und glatte Dirichletrandbedingungen verwendet werden, besitzen die Lösungen volle Regularität und stellen dem Löser daher keine spezifischen Probleme entgegen, so daß wir uns vollständig auf die parallelen Aspekte konzentrieren können: Welchen Einfluß haben die Lastverteilung, die Graphfärbungsmethoden und der Überlapp auf die numerische Güte (d. h. Konvergenzeigenschaften) und die parallele Performance?

6.4.1 Strukturiertes Gitter

Das Modellproblem

$$\begin{aligned} -\Delta u &= 0 && u \text{ auf } \Omega = (0,1) \times (0,1) \\ u(x,y) &= \tfrac{1}{2}(x+y) && \text{für } (x,y) \in \partial\Omega \end{aligned}$$

wird auf einem strukturierten Dreiecksgitter mit P1 Elementen diskretisiert. Die Steifigkeitsmatrix besitzt damit einen 5–Punkt Stern. Aber auch strukturierte Vierecksgitter oder andere genügend glatte Randbedingungen liefern sehr ähnliche Ergebnisse. Da die rechte Seite keinen Einfluß auf die uns interessierenden Eigenschaften der Gleichung hat, wählen wir sie o. B. d. A. 0, sprechen aber trotzdem synonym von einer Laplace- bzw. Poissongleichung.

Zunächst betrachten wir den Einfluß der Lastverteilung. Es wird ein spezieller Lastverteiler verwendet, der das gegebene strukturierte Gitter so auf die Prozessoren aufteilt, daß eine tensorproduktförmige Prozessoraufteilung entsteht. Hier wollen wir zwei extreme Verteilungen miteinander vergleichen, die die gesamte Spannweite der Möglichkeiten eingrenzen: Die *Streifenaufteilung* ist die einfachste Methode, die aber relativ lange Prozessorinterfaces erzeugt. Dahingehend optimiert ist die *Boxaufteilung* mit minimaler Interfacelänge. Beide Fälle sind in Abbildung 6.1 veranschaulicht. Als Kurzbezeichnung werden sie mit „$h \times v$“ gekennzeichnet, wobei h die Anzahl der Prozessoren in horizontaler, v die in vertikaler Richtung angibt. $h \cdot v$ gibt somit die Gesamtzahl der verwendeten Prozessoren an. Wenn nichts anderes gesagt ist, wird $h \geq v$ gewählt, was bei der Streifenaufteilung folglich zu vertikalen Streifen führt.

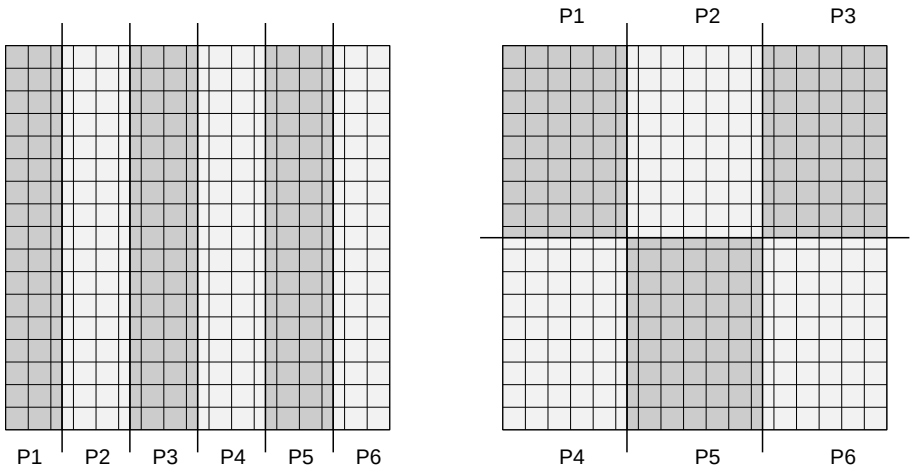


Abbildung 6.1: (Vertikale) Streifen– und Boxaufteilung für strukturierte Gitter; Beispiel 6×1 und 3×2

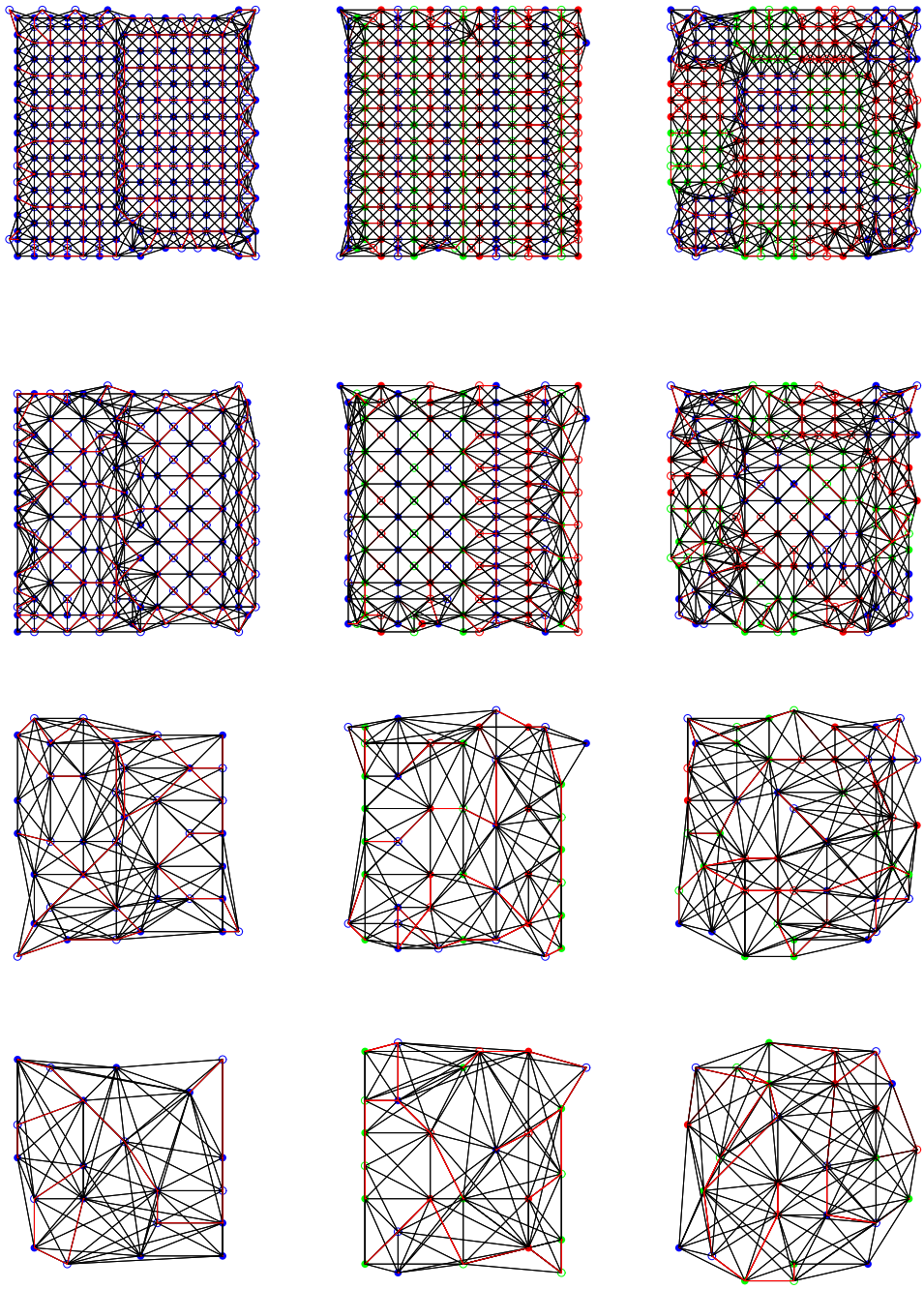


Abbildung 6.2: Isotroper Laplace Operator; Startgitterweite 1/32; Matrixgraphen der Level $-2, -3, -5, -6$ (von oben nach unten) für serielle Berechnung (links), 16×1 Streifenaufteilung (mitte) und 4×4 Boxaufteilung (rechts)

PE	h 1/128 N 16 641	1/256 66 049	1/512 263 169	1/1024 1 050 625	1/2048 4 198 401	1/4096 16 785 409
1×1	7 0.055	7 0.056				
2×1	7 0.055	7 0.056	7 0.056			
2×2	7 0.055	7 0.056	7 0.056			
4×2	7 0.055	7 0.056	7 0.059	7 0.058		
4×4	7 0.054	7 0.056	7 0.056	7 0.056		
8×4	7 0.054	7 0.056	7 0.056	7 0.066	7 0.062	
8×8	7 0.054	7 0.056	7 0.056	7 0.061	7 0.072	
16×8	7 0.055	7 0.056	7 0.056	7 0.065	7 0.064	8 0.085
1×1	7 0.055	7 0.056				
2×1	7 0.055	7 0.056	7 0.056			
4×1	7 0.055	7 0.056	7 0.056			
8×1	7 0.055	7 0.056	7 0.056	7 0.058		
16×1	7 0.055	7 0.056	7 0.056	7 0.059		
32×1	7 0.053	7 0.057	7 0.062	7 0.069	7 0.067	
64×1	7 0.055	7 0.057	8 0.085	8 0.083	8 0.096	
128×1	8 0.090	7 0.056	7 0.062	8 0.090	8 0.089	8 0.098

Tabelle 6.2: Laplace Operator; Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessoraufteilung (Box- und Streifenaufteilung)

Abbildung 6.2 soll einen ersten Eindruck von der Struktur der Graphen der vergröberten Matrizen geben. Für die graphische Darstellung werden auch auf den gröberen Leveln die Koordinaten der entsprechenden Elternknoten auf dem gegebenen Ausgangsgitter verwendet, um den Knoten im Bild plazieren zu können. Diese Information wird nur zum Plotten bereitgestellt und wird während der Rechnung nicht verwendet. An der Knotenposition wird für einen grobmarkierten Knoten ein gefüllter Kreis, für einen feinmarkierten eine Kreislinie gezeichnet. Matrixeinträge werden als Verbindungslinie zwischen den Knoten gezeichnet. Für parallele Rechnungen werden nur Masterknoten gezeichnet und als Farbe ein Grauton entsprechend der Prozessornummer verwendet.

Die Matrixgraphen entsprechen nur teilweise und lokal einer idealen regelmäßigen Vergröberung, wie sie bei einem geometrischen Mehrgitterverfahren vorliegen würde. Die Störung tritt aber schon bei seriellen Rechnungen auf (linke Spalte im Bild) und sollte daher die Parallelisierung nicht zusätzlich belasten. Auf gröberen Leveln ist nicht mehr zwischen serieller und paralleler Rechnung zu unterscheiden.

Die spaltenweise Betrachtung von Tabelle 6.2 zeigt, daß die Anzahl benötigter Mehrgitter Iterationsschritte fast konstant 7 beträgt, unabhängig von der Anzahl an Prozessoren und auch der Lastverteilung; d. h. es ist gelungen, daß die Parallelisierung im wesentlichen gleichwertige Grobgittermatrizen erstellt, unabhängig davon, wie stark die Feingittermatrix zerteilt ist. Die Leitidee für die Parallelisierung — an den Prozessorrändern sowenig Fehler wie möglich zu machen — hat sich also erfolgreich in die Praxis umsetzen lassen.

Betrachtet man Tabelle 6.2 zeilenweise, erkennt man, daß die Konvergenzgüte nahezu unabhängig von der Gitterweite, d. h. der Anzahl der Knoten auf dem Startgitter ist. Das ist eine wichtige Voraussetzung, um sehr große Gleichungen effizient lösen zu können; ansonsten könnte ein Laufzeitgewinn durch Einsatz vieler Prozessoren durch erhöhte Iterationszahlen zunichte gemacht werden. Damit ist es gelungen, die für den Laplace Operator theoretisch erzielbare Gitterweitenunabhängigkeit der Konvergenzgüte auch in der Praxis für sehr viele Prozessoren zu realisieren.

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	insgesamt 66 049 Knoten			
1	2.11	1.92	0.00	0.00
2	2.10	1.92	3.07	0.02
4	2.12	1.93	3.13	0.05
8	2.15	1.93	3.13	0.10
16	2.18	1.94	3.09	0.15
32	2.26	1.95	3.16	0.26
64	2.34	1.95	3.27	0.38
128	2.46	1.95	3.33	0.61

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	insgesamt 4 198 401 Knoten			
32	2.19	2.00	3.77	0.04
64	2.21	2.00	3.84	0.05
128	2.25	2.01	3.92	0.08

Tabelle 6.3: Laplace Operator ($\varepsilon = 1$); Speicherkomplexität für insgesamt 66 049 und 4 198 401 Knoten (Speedup)

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.16	1.98	3.51	0.01
8	2.19	2.00	3.75	0.03
32	2.19	2.00	3.77	0.04
128	2.20	2.00	3.88	0.04
	32 768 Knoten/PE			
2	2.10	1.92	3.07	0.02
8	2.20	1.98	3.50	0.06
32	2.23	2.00	3.70	0.07
128	2.25	2.01	3.92	0.08

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.11	1.92	0.00	0.00
4	2.18	1.98	3.49	0.03
16	2.20	2.00	3.68	0.04
64	2.21	2.00	3.84	0.05
	16 384 Knoten/PE			
1	1.85	1.73	0.00	0.00
4	2.12	1.93	3.13	0.05
16	2.21	1.98	3.44	0.08
64	2.26	2.00	3.79	0.10

Tabelle 6.4: Laplace Operator ($\varepsilon = 1$); Speicherkomplexität für volle bis achtel Last (Scaleup)

Nun stellt sich die Frage, wie hoch die Kosten für diese Güte sind, in zweierlei Hinsicht: Speicheraufwand und Laufzeit. Zunächst soll der Speicheraufwand betrachtet werden. Tabelle 6.3 zeigt die Situation für 2 feste Problemgrößen und Tabelle 6.4 für feste Problemgrößen je Prozessor, die in Boxaufteilung auf unterschiedliche Prozessorzahlen verteilt werden.

Die Gitterkomplexität c_G ist außerordentlich konstant ca. 2,0 und zeigt, daß unabhängig von der Prozessoranzahl und Problemgröße immer ähnlich große Grobgitter relativ zum gegebenen Gitter erzeugt werden, der Vergröberungsfaktor also im wesentlichen konstant 2 ist.

Der Überlappanteil B/M ist für kleine Probleme auf vielen Prozessoren ungeheuer groß (bis zu 61%), weil mit sinkender Größe der Kernpartition das Verhältnis von Oberfläche zu Volumen stark ansteigt. Ist dagegen die Kernpartition vernünftig groß (rechte Tabelle 6.3), so macht der Überlapp nur wenige Prozent der Kernpartition aus, und der daraus resultierende Mehraufwand sollte tolerierbar sein.

Die Überlappkomplexität $c_{\ddot{U}}$ zeigt, daß der Großteil des Überlapps erst auf den algebraisch vergröberten Leveln entsteht, d. h. nicht genau im selben Maße kleiner wird wie die Kernpartition ($c_G \approx 2{,}0$). Gut ist, daß $c_{\ddot{U}}$ nur schwach von der Prozessorzahl abhängt, d. h. die parallele Skalierbarkeit nicht einschränkt.

Die Operatorkomplexität c_A in Spalte 1 der Tabellen 6.3 und 6.4 steigt für zu kleine Probleme etwas an, ist jedoch für ausreichend große Problemgrößen fast konstant 2,2, d. h. die Matrizen werden nicht dichter, sowohl bzgl. der Prozessorzahl als auch der Problemgröße. Die Operatorkomplexität ist nicht nur für den Hauptspeicherbedarf der Rechnung ein entscheidender Einflußfaktor, sondern auch für den Rechenaufwand. Auch diese wichtige Größe skaliert gut mit der Prozessorzahl.

Alle angeführten Kennzahlen lassen erkennen, daß Rechnungen auch für große Prozessorzahlen effizient durchführbar sind. Mit wachsender Problemgröße nimmt der mit der Parallelisierung verbundene Mehraufwand sogar ab, das asymptotische Verhalten ist folglich optimal.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	32.69	2.51	17.59	53.70	1.00	1.00	1.00	1.00	31.24	1.00	1.00
2	20.61	1.36	9.51	31.70	0.79	0.93	0.93	0.85	18.57	0.84	0.87
4	13.87	0.78	5.49	22.82	0.59	0.80	0.80	0.59	11.34	0.69	0.73
8	10.90	0.50	3.52	17.08	0.37	0.62	0.62	0.39	6.77	0.58	0.59
16	7.51	0.46	3.23	14.17	0.27	0.34	0.34	0.24	4.31	0.45	0.40
32	5.71	0.41	2.85	12.13	0.18	0.19	0.19	0.14	3.06	0.32	0.26
64	5.91	0.56	3.90	13.33	0.09	0.07	0.07	0.06	2.60	0.19	0.12
128	5.13	0.18	1.28	10.27	0.05	0.11	0.11	0.04	2.02	0.12	0.12
	insgesamt 4 198 401 Knoten										
32	162.49	5.80	40.63	208.41	1.00	1.00	1.00	1.00	86.14	1.00	1.00
64	119.10	3.19	22.32	147.20	0.68	0.91	0.91	0.71	55.93	0.77	0.81
128	73.79	1.76	12.30	89.67	0.55	0.83	0.83	0.58	28.41	0.76	0.78

Tabelle 6.5: Laplace Operator ($\varepsilon = 1$); Speedup für insgesamt 66 049 und 4 198 401 Knoten

Nachdem die verschiedenen Komplexitätsmaße optimales Verhalten anzeigen, muß jetzt noch die Laufzeit betrachtet werden. Tabelle 6.5 gibt die entscheidenden Zeiten

in einer *Speedup* Betrachtung an: Die globale Problemgröße beträgt konstant 66 049 bzw. 4 198 401 Knoten. Ausgehend von vollausgelasteten Prozessoren ergeben sich über wenige Prozessorzahlverdopplungen hinweg gute Effizienzen. Ansonsten macht die zu schwache Auslastung der Prozessoren verbunden mit den zunehmenden Kommunikationskosten die Effizienz zunichte. Bei extremer Unterauslastung erhöhen sich die Zeiten sogar teilweise. Die Ausführungszeiten reagieren somit wesentlich stärker als die Komplexitäten auf das Mißverhältnis von Kernpartition zu Überlapp. Einen großen Anteil daran haben Fixkosten in Datenaustauschroutinen und Effekte, die in Abschnit 6.4.2 genauer untersucht werden.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	80.65	5.32	37.26	121.04	1.00	1.00	1.00	1.00	73.86	1.00	1.00
8	127.39	5.66	39.59	170.85	0.63	0.94	0.94	0.71	81.97	0.90	0.91
32	162.49	5.80	40.63	208.41	0.50	0.92	0.92	0.58	86.14	0.86	0.88
128	200.97	5.93	47.41	252.52	0.40	0.90	0.79	0.48	90.70	0.81	0.80
	32 768 Knoten pro Prozessor										
2	20.61	1.36	9.51	31.70	1.00	1.00	1.00	1.00	18.57	1.00	1.00
8	37.03	1.58	11.08	51.75	0.56	0.86	0.86	0.61	23.51	0.79	0.81
32	51.95	1.74	12.17	69.74	0.40	0.78	0.78	0.45	26.12	0.71	0.73
128	73.79	1.76	12.30	89.67	0.28	0.77	0.77	0.35	28.41	0.65	0.69
	65 536 Knoten pro Prozessor										
1	32.69	2.51	17.59	53.70	1.00	1.00	1.00	1.00	31.24	1.00	1.00
4	53.55	2.88	20.16	80.87	0.61	0.87	0.87	0.66	43.04	0.73	0.77
16	79.94	2.98	20.84	105.99	0.41	0.84	0.84	0.51	44.94	0.70	0.74
64	119.10	3.19	22.32	147.20	0.27	0.79	0.79	0.36	55.93	0.56	0.62
	16 384 Knoten pro Prozessor										
1	6.36	0.57	3.98	14.20	1.00	1.00	1.00	1.00	5.71	1.00	1.00
4	13.87	0.78	5.49	22.82	0.46	0.72	0.72	0.62	11.34	0.50	0.58
16	24.43	0.91	6.36	33.96	0.26	0.63	0.63	0.42	13.59	0.42	0.49
64	39.30	1.45	10.18	54.61	0.16	0.39	0.39	0.26	17.95	0.32	0.34

Tabelle 6.6: Laplace Operator ($\varepsilon = 1$); Scaleup für volle bis achtel Last

Die für die Anwendung relevantere *Scaleup* Betrachtung liegt der Tabelle 6.6 zugrunde. Die Zeiten sind von voller Last ausgehend für jede Halbierung bis zu einem Achtel angegeben. Man sieht eine deutliche Verschlechterung für wachsende Prozessorzahlen, die für 128 Prozessoren für die Aufbauphase in einer Effizienz E_{FA} von unter 50% endet; der Mehrgitterlöser selbst skaliert wesentlich besser (E_{IT} bis 90%). Die Gesamteffizienz wird durch die Aufbauphase dominiert.

Den Hauptgrund für dieses ungünstige Verhalten wollen wir nun analysieren. Abbildung 6.3 oben zeigt zunächst, daß die Knotenanzahlen mit den zunehmend gröber werdenden algebraischen Leveln exponentiell abnehmen. Dies entspricht einer konstanten Vergröberungsrate, die auch als Zusammenfassung in Tabelle 6.4 erkennbar war. Der Überlapp nimmt ebenfalls — mit leichter Ausnahme für die gröbsten Level — exponentiell ab, allerdings mit einer etwas geringeren Rate, so daß auf den

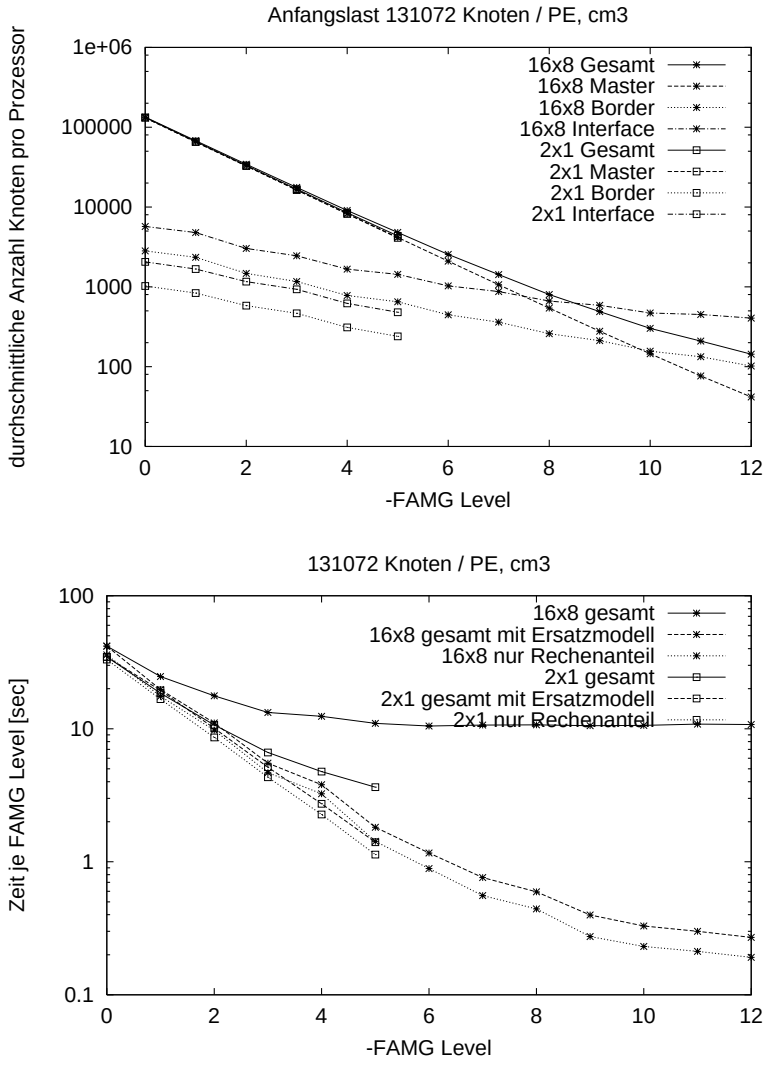


Abbildung 6.3: Skalierbarkeit des puren Gitteraufbaus ohne DDD_Xfer

gröbsten Leveln die Borderknoten die Masterknoten überwiegen. Die untere Graphik zeigt verschiedene Zeiten je Level. Zunächst sieht man, daß die Gesamtzeit je Level *nicht* gegen 0 sondern gegen eine fixe Zeit von mehreren Sekunden strebt, also *nicht* mit der Knotenzahl skaliert.

Um die Ursache ausfindig zu machen, wurde der Algorithmus zur Bearbeitung eines Levels in 11 Teile gegliedert und von jedem einzeln die Laufzeit gemessen. Die Ergebnisse sind in Abbildung 6.4 für das größte Beispiel, bei dem die Effekte am besten sichtbar werden, dargestellt. Das oberste Bild zeigt für einen Prozessor das Zeitverhalten, die unteren Bilder die Übersicht über alle Prozessoren für Level 0 und -12.

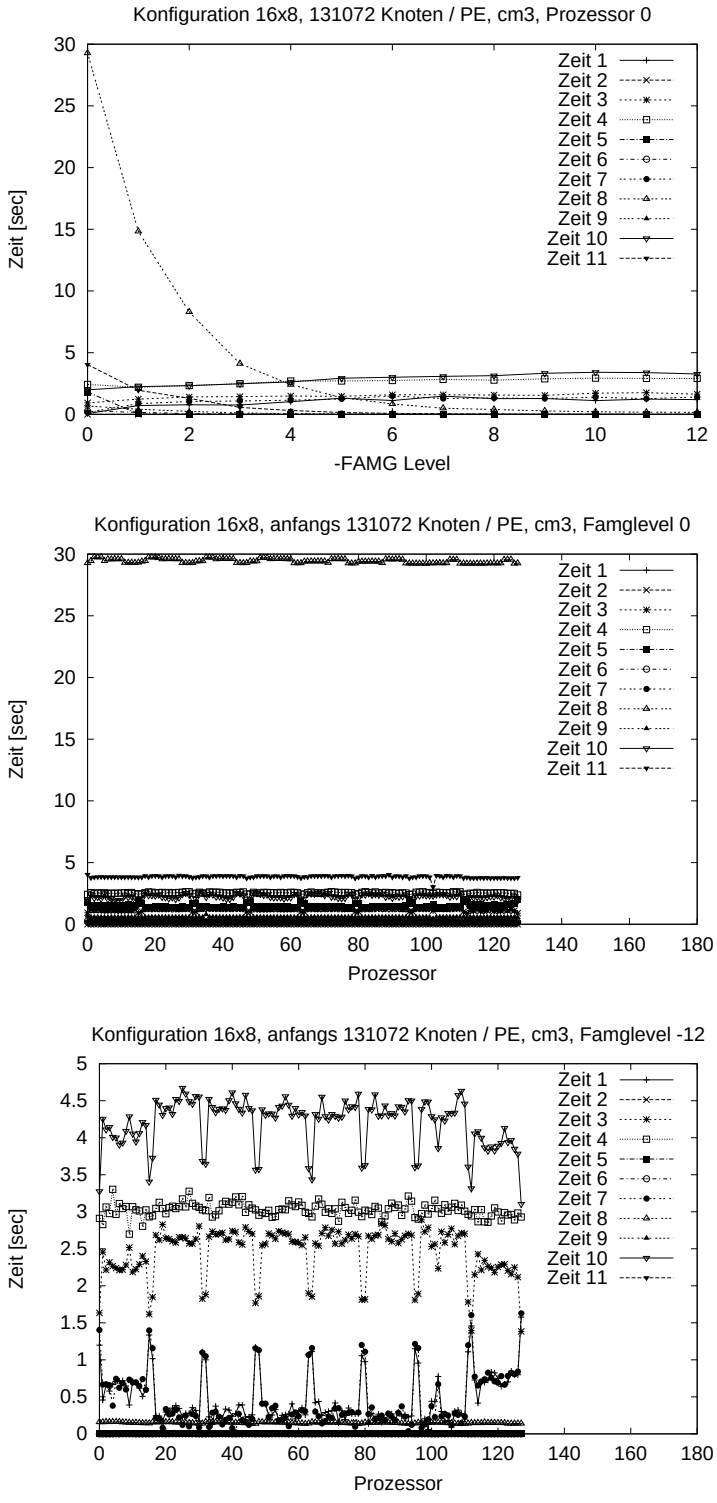


Abbildung 6.4: Beispiel bei einer freien Zeitmessung

Zunächst fällt vor allem auf gröberen Leveln das regelmäßige Schwanken der Zeiten in Abhängigkeit von der Prozessornummer auf. Das ist der Einfluß des Randes des Rechengebiets, an dem die Prozessoren entsprechend weniger Nachbarprozessoren besitzen. Auf feinen Leveln wird die Laufzeit von Algorithmusteil (8) dominiert, dem Bewerten und Erstellen der guten Elternmengen. Das entspricht der Erwartung. Die Probleme treten auf gröberen Leveln auf. Es sind nur 5 Algorithmusteile, die nicht mit der Knotenzahl skalieren:

- die Synchronisierung des Vergröberungsendes in Zeile (3) Algorithmus 5.21 pFAMGConstruct (Zeit 1),
- die Datenübertragung und Modifikation der DDD internen Datenstrukturen (DDD_XferEnd) in CompleteU1, CompleteU2, IdentifyCoarse (Zeit 3, 4, 10) und
- das Herstellen der teilweisen Konsistenz für die Steifigkeitsmatrix in Zeile (8) Algorithmus 5.21 pFAMGConstruct (Zeit 7).

Vor allem das Verhalten von Teil 1 und 7 ist nicht erklärbar, da es sich jeweils nur um das Bilden eines globalen Maximums bzw. um eine einfache Datenaustauschkommunikation handelt. Daher wurden nochmals Zeitmessungen durchgeführt, aber diesmal wurde vor jedem Algorithmusteil explizit synchronisiert; die Ergebnisse sind in Abbildung 6.5 zu sehen. Die Vermutung hat sich bestätigt: Zeiten 1 und 7 sind so unerklärlich hoch, weil sie durch synchronisierend wirkende Kommunikation in früheren Algorithmusteilen verfälscht wurden. Der Ablauf ist in Abbildung 6.6 schematisch dargestellt. Unerwartet weitreichend ist die Wirkung der Kommunikation in Teil 10 für Level $\ell - 1$ auf Teil 1 für Level ℓ.

Die Laufzeit wird also nur von den Algorithmusteilen 3, 4 und 10 beeinträchtigt. Es ist genau die Datenübertragung und Änderung der internen Datenstruktur des Moduls DDD, auf die wir keinen Einfluß haben. Die Ursache dafür ist wiederum, daß DDD nur einen einzigen Graphen handhaben kann, keine Hierarchie von Graphen. Eine Änderung auf einem sehr groben Level, d. h. einem kleinen Teilgraphen, muß als Änderung des Gesamtgraphen, der alle Level umfaßt, verarbeitet werden. Hierbei verursacht vor allem das Sortieren der Objekte im Prozessorinterface den wesentlichen Aufwand. Sobald die Interfaces wieder konsistent bestehen, kann ein purer Datenaustausch, d. h. eine Kommunikation ohne Strukturänderung, auf einzelnen Leveln zeitoptimal durchgeführt werden.

Da wir auf die Verwendung von DDD angewiesen sind, um unser Verfahren in das Gesamtsystem *UG* integrieren zu können, schlagen wir folgende Abhilfe vor. DDD müßte auch bei Strukturänderungen levelweise vorgehen können. Daß das keine unerfüllbare Anforderung ist, zeigte P. Bastian mit dem Prototypen *UGp* des hier verwendeten Programmsystems, in dem er genau dieses Vorgehen realisiert hat [Bas96b]. Um unseren Algorithmus realistisch zu bewerten, werden wir auch Zeiten betrachten, die eine noch nicht vorliegende, nach Art von *UGp* verbesserte Implementierung des Moduls

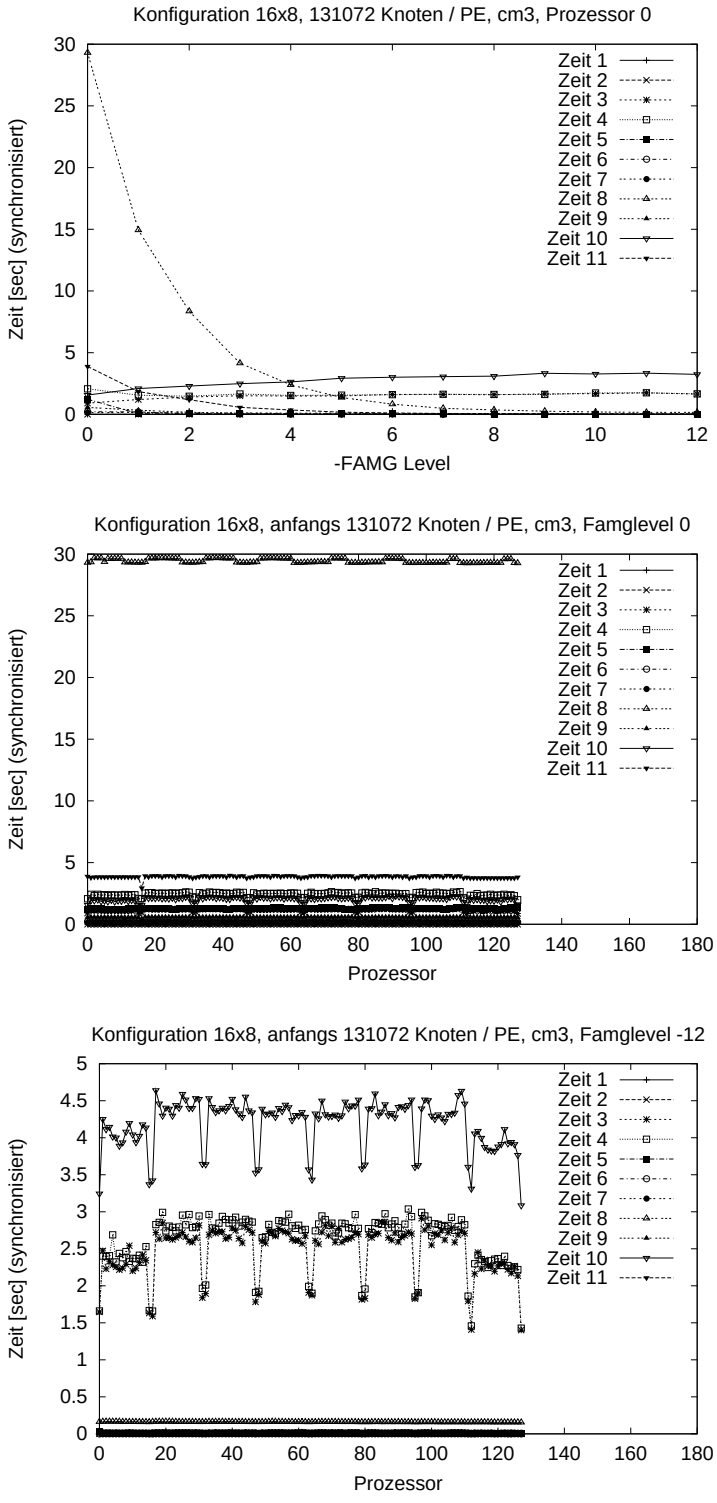


Abbildung 6.5: Beispiel bei einer synchronisierten Zeitmessung

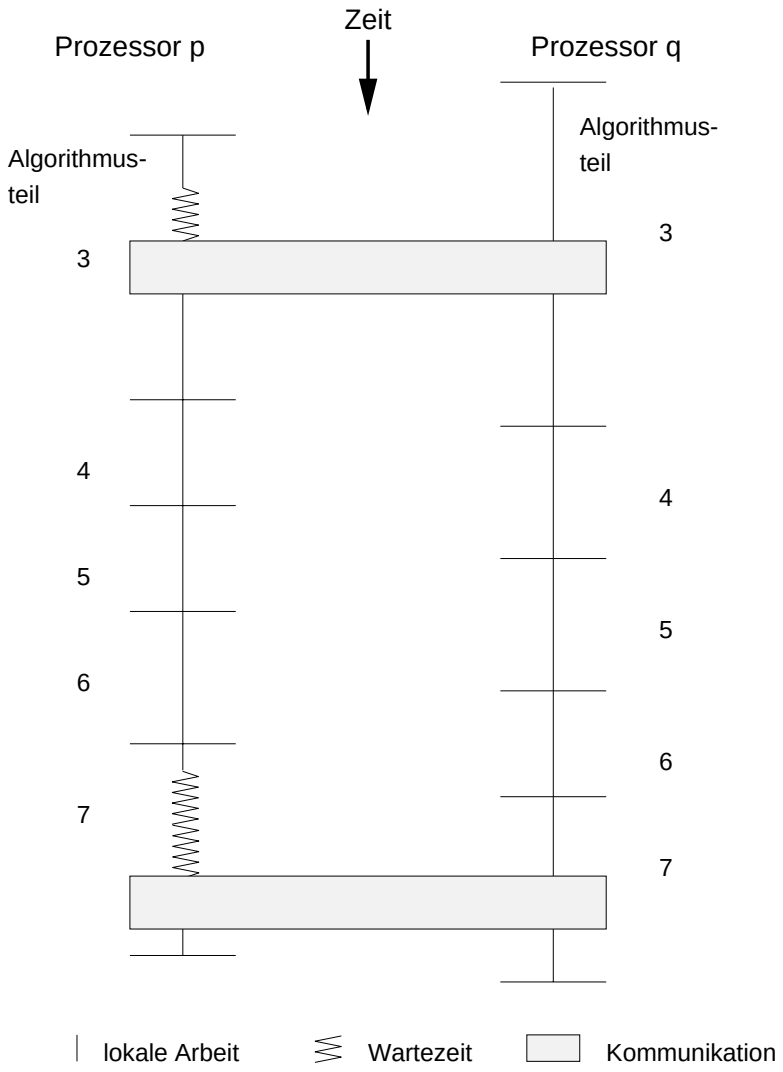


Abbildung 6.6: Kommunikation in Algorithmusteil 3 verfälscht Zeitmessung in Teil 7

DDD annimmt. Dazu messen wir die Zeit unserer Algorithmusteile 1 bis 2, 5 bis 9 und 11, also alle Anteile ohne Veränderungen der Graphstruktur, und addieren dazu die über die Prozessoren maximierte Zeit, die das verbesserte DDD haben würde.

Das Laufzeitmodell für dieses verbesserte DDD entspricht dem des *UGp*. Es berücksichtigt 2 Anteile. Das ist zum einen die Zeit für das Erstellen der Nachricht $\mathcal{O}({}^{\ell}N)$, wozu einmal *alle* ${}^{\ell}N$ Knoten auf Level ℓ des Prozessors bearbeitet werden. Zum anderen ist das die Zeit zum Sortieren der neuen Interfacelisten $\mathcal{O}({}^{\ell}n \log {}^{\ell}n)$. Dazu werden ${}^{\ell}n$ Objekte sortiert, wobei ${}^{\ell}n$ die Zahl der Interfaceobjekte ist, gezählt mit der Vielfachheit ihrer Kopien. $\mathcal{O}(n \log n)$ entspricht dem statistisch zu erwartenden

Aufwand eines optimalen Sortierverfahrens [OW90]. Das ergibt das Modell

$$ {}^{\ell}T^{p} = \alpha\, {}^{\ell}N^{p} + \beta\, {}^{\ell}n^{p} \log {}^{\ell}n^{p} \tag{6.8} $$

für die zu erwartende Laufzeit einer der Schritte 3, 4 und 7. Als Berechnungsgrundlagen werden die Werte ${}^{\ell}N$ und ${}^{\ell}n$ während des Programmlaufs gewonnen. Die Zeitkonstanten α und β werden anhand der Daten von Level 0 (der schon einer optimalen Implementierung entspricht, da noch keine unnötigen Graphenteile existieren) durch das Lösen eines Ausgleichsproblems bestimmt [Sto89] und für die Berechnungen auf den gröberen Leveln verwendet. Das Laufzeitmodell wird also für jedes Gleichungssystem individuell geeicht; sollte das Ausgleichsproblem keine zulässige Lösung finden, werden die Zeitkonstanten eines vergleichbaren Falls verwendet.

Die derartig bestimmten Laufzeiten werden hier mit einem Stern markiert, ebenso die darauf beruhenden Effizienzzahlen, z. B.

$$ T^{*}_{FA} \text{ und } E^{*}_{FA}. \tag{6.9} $$

Den Erfolg sieht man in Abbildung 6.3 unten als gestrichelte Linie: Die Laufzeit für die Bearbeitung eines Levels skaliert nun wie erwartet mit der Knotenanzahl. Dadurch ergeben sich auch realistischere Werte in der Speedup Tabelle 6.5. Allerdings gilt die Aussage immer noch, daß eine gewisse Mindestauslastung eines Prozessors gegeben sein muß, um einen noch vertretbaren Speedup zu erhalten. Wesentlich besser skaliert die Laufzeit, wenn man eine Scaleup Betrachtung zugrunde legt: Tabelle 6.6 zeigt durchwegs sehr gute Effizienzen von über 80% bei 128 Prozessoren und voller Last. Bis 1/4 der vollen Last liegt die Effizienz immer noch deutlich über 50%. Bei zu geringer Auslastung der Prozessoren (ab 1/8 der vollen Last) sinkt die Effizienz allerdings spürbar unter 50%.

Obwohl die Skalierungseigenschaften sehr gut sind, wollen wir noch den Baselevellöser betrachten. Vorzuziehen ist, wie in Abschnitt 5.10 motiviert, ein direkter Löser, für den das Grobgitter allerdings agglomeriert werden muß. Die Skalierbarkeit dieses Vorgehens ist nicht gesichert und muß daher experimentell nachgewiesen werden. Die Daten sind in Tabelle 6.7 aufgeführt. Die Abkürzungen für die einzelnen Programmteile sind *Ba* für die gesamten Aktionen auf dem Baselevel, die sich aus den 3 Teilen *Ag* für die Baselevelagglomeration, *De* für die *LU* Zerlegung und *Sol* für das Lösen aller Gleichungssysteme auf dem gröbsten Level zusammensetzen. Da das Agglomerieren eine Strukturänderung des Graphen bedeutet, leidet es ebenfalls unter der derzeitigen Implementierung des Moduls DDD. Daher ist es auch hier sinnvoll, das verbesserte Laufzeitmodell nach (6.8) zu berücksichtigen. Die daraus resultierenden Effizienzen von über 50% entsprechen denen des Gesamtverfahrens. Daher ist dieses Vorgehen geeignet und man muß nicht zu verteilt, aber iterativ vorgehenden Varianten übergehen, die die numerische Gesamtgüte gefährden könnten bzw. die Laufzeit durch zu viele Iterationen bei steigender Prozessorzahl belasten würden. Da alle Rechnungen in dieser Arbeit das gleiche, gute Verhalten für den Baselevellöser zeigen, wird im weiteren nicht mehr darauf eingegangen.

PE	T_{Ba}	T_{Ag}	T_{De}	T_{Sol}	E_{Ba}	E_{Ag}	E_{De}	E_{Sol}	N_{cg}	T^*_{Ba}	E^*_{Ba}
	131 072 Knoten pro Prozessor										
2	3.91	1.14	2.36	0.40	1.00	1.00	1.00	1.00	4132	3.47	1.00
8	5.89	2.39	3.04	0.46	0.66	0.48	0.78	0.89	4346	4.24	0.82
32	8.36	3.38	4.43	0.55	0.47	0.34	0.53	0.74	4613	6.08	0.57
128	8.14	4.50	3.21	0.43	0.48	0.25	0.74	0.94	2957	5.04	0.69
	32 768 Knoten pro Prozessor										
2	2.43	0.69	1.39	0.35	1.00	1.00	1.00	1.00	4160	2.44	1.00
8	5.13	1.25	3.40	0.47	0.47	0.55	0.41	0.75	4238	4.61	0.53
32	6.60	1.83	4.23	0.53	0.37	0.38	0.33	0.66	4605	5.85	0.42
128	6.42	2.81	3.24	0.37	0.38	0.25	0.43	0.94	2905	4.91	0.50
	65 536 Knoten pro Prozessor										
1	3.65	0.15	3.06	0.44	1.00	1.00	1.00	1.00	4034	3.51	1.00
4	8.62	1.25	6.75	0.62	0.42	0.12	0.45	0.71	4150	8.45	0.41
16	6.24	1.94	3.81	0.49	0.59	0.08	0.80	0.91	4334	5.03	0.70
64	9.03	3.35	5.10	0.58	0.40	0.04	0.60	0.77	4724	6.99	0.50
	16 384 Knoten pro Prozessor										
1	4.35	0.07	3.82	0.46	1.00	1.00	1.00	1.00	4032	4.28	1.00
4	4.59	0.80	3.33	0.46	0.95	0.09	1.15	1.02	4161	4.90	0.87
16	4.58	1.12	3.01	0.44	0.95	0.06	1.27	1.05	4299	4.19	1.02
64	7.78	2.34	4.89	0.54	0.56	0.03	0.78	0.85	4689	6.74	0.64

Tabelle 6.7: Laplace Operator ($\varepsilon = 1$); Scaleup der Basesolver Komponenten für volle bis achtel Last

Der Einfluß der Färbungsmethode wird in Abschnitt 6.5.2 im Zusammenhang mit dem anisotropen Laplace Operator untersucht.

Zusammenfassend haben wir gesehen, daß für den Laplace Operator auf einem strukturierten Gitter und einer Boxaufteilung die numerische Güte bei der Parallelisierung vollkommen unverändert bleibt und die Skalierbarkeit auch hohen Erwartungen voll entspricht, wenn man das Laufzeitmodell nach (6.8) unterstellt.

6.4.2 Unstrukturiertes Gitter

Das Modellproblem

$$\begin{aligned} -\Delta u &= 0 && u \text{ auf } \Omega \\ u(x,y) &= \tfrac{1}{2}(x+y) && \text{für } (x,y) \in \partial\Omega \end{aligned}$$

wird auf einem unstrukturierten Gitter berechnet, das eine durch einen Gittergenerator erzeugte Triangulierung des Gebiets des Wolfgangsees in Österreich ist. Zur Diskretisierung werden P1 Elemente verwendet. Nachdem das Startgitter (7006 Elemente, 3690 Knoten) auf die beteiligten Prozessoren verteilt ist, wird zunächst solange regelmäßig verfeinert, bis die gewünschte Gitterfeinheit für die Berechnung hergestellt ist. Dann werden allerdings alle gröberen Gitter entfernt und das FAMG auf dem feinsten Level gestartet.

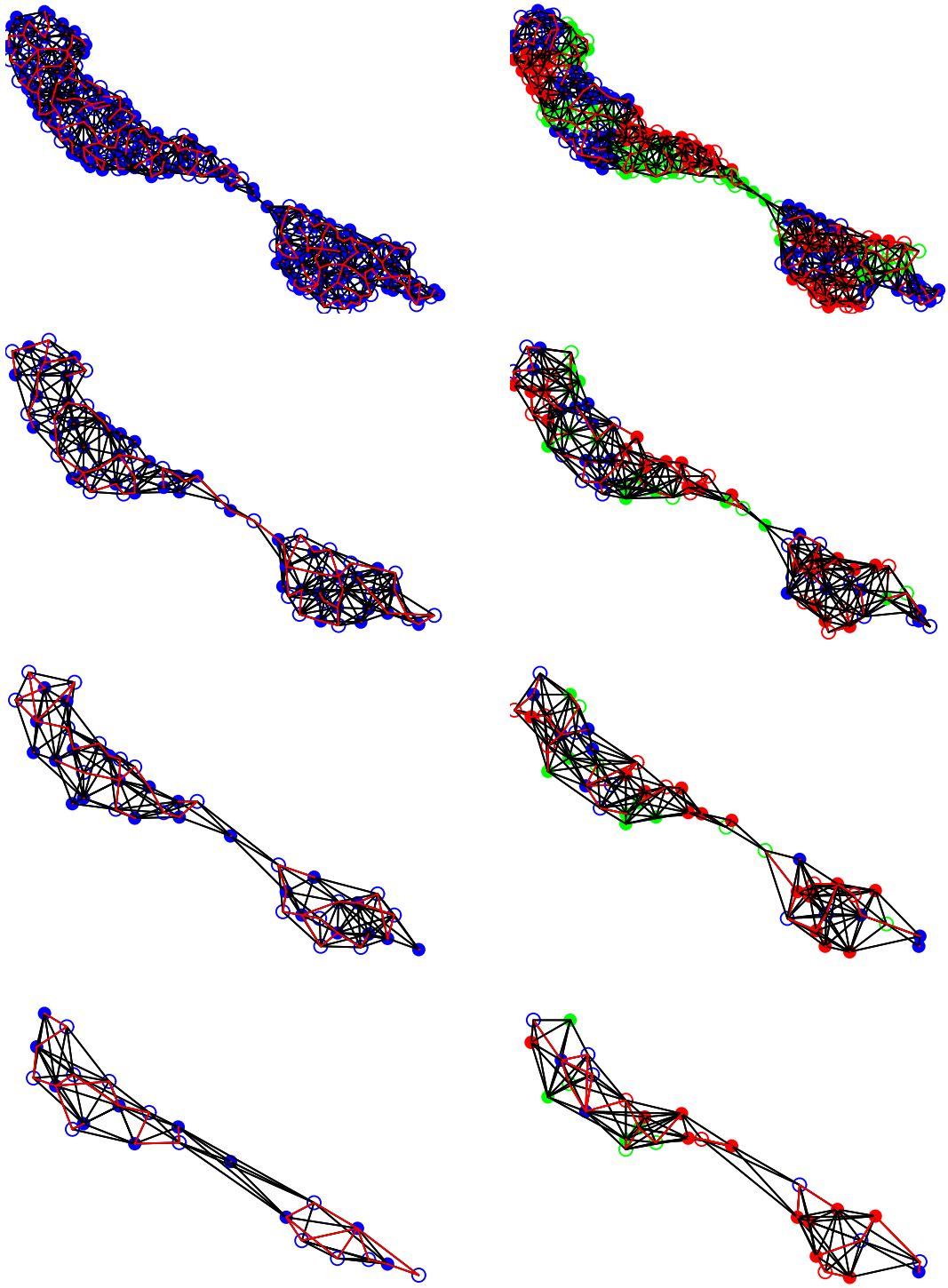


Abbildung 6.7: Isotroper Laplace Operator auf Wolfgangsee; 994 Feinlevelknoten; Matrixgraphen der Level -2, -4, -5, -6 (von oben nach unten) für serielle Rechnung (links) und 16 Prozessoren (rechts)

Abbildung 6.7 zeigt für ein sehr kleines Beispiel mit 994 Feingitterknoten 2 Folgen von vergröberten Matrixgraphen. Ausgefüllte Kreisscheiben stellen grobmarkierte Knoten dar. Für die parallele Rechnung entspricht der Grauton der Prozessornummer. Es zeigt sich kein relevanter Unterschied zwischen serieller und paralleler Berechnung. Detailierte Analysen folgen nun für Berechnungen mit größeren Knotenzahlen.

PE	N 14 385	56 793	225 681	899 745	3 593 025	14 360 193
1	7 0.063	9 0.105				
2	7 0.061	8 0.095	10 0.137			
4	7 0.061	8 0.098	9 0.122			
8	7 0.058	8 0.079	9 0.121	11 0.175		
16	7 0.056	8 0.093	9 0.117	11 0.170		
32	7 0.056	9 0.109	9 0.117	11 0.182	12 0.207	
64	7 0.056	9 0.116	9 0.119	11 0.181	12 0.206	
128	7 0.058	8 0.080	10 0.156	12 0.197	12 0.210	12 0.214

Tabelle 6.8: Isotroper Laplace Operator auf Wolfgangsee; Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessorzahl

Tabelle 6.8 zeigt, daß die Konvergenzgüte (fast) unabhängig von der Prozessoranzahl ist. Allerdings verschlechtert sich die Güte leicht in logarithmischem Verhältnis zur Knotenzahl, eine Eigenschaft, die schon am seriellen Verfahren zu beobachten ist [Wag00a] und hier in der ersten Zeile andeutungsweise zu sehen ist. Für sehr große Knotenzahlen bleibt die Konvergenzrate konstant. Leider lassen sich derzeit keine größeren Probleme rechnen, um das Verhalten noch besser beurteilen zu können.

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	insgesamt 56 793 Knoten			
1	2.33	1.98	0.00	0.00
2	2.34	1.98	3.75	0.01
4	2.39	1.98	3.57	0.06
8	2.43	1.98	3.56	0.11
16	2.54	1.99	3.60	0.22
32	2.62	1.99	3.62	0.32
64	2.77	2.00	3.63	0.54
128	2.92	2.00	3.73	0.78

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	insgesamt 3 593 025 Knoten			
32	2.48	1.99	4.82	0.06
64	2.54	1.99	4.85	0.09
128	2.60	1.99	4.91	0.13

Tabelle 6.9: Isotroper Laplace Operator auf Wolfgangsee; Speicherkomplexität für insgesamt 56 793 und 3 593 025 Knoten (Speedup)

Die Tabellen 6.9 und 6.10 zeigen die Komplexitäten. Die Gitterkomplexität c_G ist wieder nahezu konstant. Auch der Anteil B/M des Überlapps beträgt in den relevanten Fällen nur einige Prozent; dies liegt nur leicht über dem Idealfall des strukturierten

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	112 840 Knoten/PE			
2	2.47	2.04	4.86	0.01
8	2.49	2.01	4.79	0.04
32	2.48	1.99	4.82	0.06
128	2.44	1.97	4.81	0.06
	28 396 Knoten/PE			
2	2.34	1.98	3.75	0.01
8	2.58	2.04	4.56	0.07
32	2.62	2.02	4.81	0.11
128	2.60	1.99	4.91	0.13

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	56 793 Knoten/PE			
1	2.33	1.98	0.00	0.00
4	2.51	2.04	4.43	0.04
16	2.56	2.02	4.78	0.07
64	2.54	1.99	4.85	0.09
	14 385 Knoten/PE			
1	1.82	1.68	0.00	0.00
4	2.39	1.98	3.57	0.06
16	2.68	2.04	4.56	0.14
64	2.74	2.02	4.86	0.18

Tabelle 6.10: Isotroper Laplace Operator auf Wolfgangsee; Speicherkomplexität für volle bis achtel Last (Scaleup)

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 56 793 Knoten										
1	37.94	2.43	21.88	61.51	1.00	1.00	1.00	1.00	36.52	1.00	1.00
2	20.70	1.25	9.98	32.12	0.92	0.97	1.10	0.96	20.08	0.91	0.97
4	15.15	0.73	5.84	22.39	0.63	0.83	0.94	0.69	11.72	0.78	0.83
8	10.50	0.42	3.34	15.01	0.45	0.73	0.82	0.51	6.80	0.67	0.72
16	10.15	0.30	2.41	13.85	0.23	0.50	0.57	0.28	5.81	0.39	0.44
32	7.01	0.21	1.88	10.32	0.17	0.36	0.36	0.19	3.85	0.30	0.32
64	6.29	0.17	1.57	9.20	0.09	0.22	0.22	0.10	3.20	0.18	0.19
128	11.75	0.39	3.09	17.52	0.03	0.05	0.06	0.03	3.48	0.08	0.07
	insgesamt 3 593 025 Knoten										
32	222.54	5.47	65.67	290.62	1.00	1.00	1.00	1.00	106.61	1.00	1.00
64	175.90	3.16	37.93	215.59	0.63	0.87	0.87	0.67	67.92	0.78	0.81
128	100.96	1.77	21.23	124.35	0.55	0.77	0.77	0.58	38.68	0.69	0.72

Tabelle 6.11: Isotroper Laplace Operator auf Wolfgangsee; Speedup für insgesamt 56 793 und 3 593 025 Knoten

Gitters. Die Überlappkomplexität $c_{\ddot{U}}$ zeigt, daß auf den gröberen Leveln merklich mehr Knotenkopien angelegt werden. Das belastet auch die Operatorkomplexität c_A, die im Durchschnitt mit 2,5 schon deutlich über 2,0 liegt.

Tabelle 6.11 zeigt ein ähnliches Speedup Verhalten wie beim strukturierten Gitter. Der Hauptgrund für die schlechten Werte ist die äußerst geringe Auslastung verbunden mit Einflüssen, die im folgenden analysiert werden. Wichtiger ist der Scaleup, dargestellt in Tabelle 6.12. Zunächst betrachten wir die Zeiten für eine Iteration T_{IT} und deren Effizienz E_{IT}. Verglichen mit dem strukturierten Gitter (Tabelle 6.6) sind die Werte nur um wenige Prozentpunkte schlechter. Die dafür entscheidenden Gründe sind für alle Zeitbetrachtungen ausschlaggebend und sollen nun ausführlich analysiert werden.

Zunächst wirkt sich eine leichte Erhöhung des Überlapps stärker aus, als man gemein-

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	112 840 Knoten pro Prozessor										
2	92.60	5.02	50.23	144.20	1.00	1.00	1.00	1.00	90.14	1.00	1.00
8	152.01	5.25	57.74	211.76	0.61	0.96	0.87	0.68	100.80	0.89	0.89
32	222.54	5.47	65.67	290.62	0.42	0.92	0.76	0.50	106.61	0.85	0.81
128	282.43	5.70	68.36	353.47	0.33	0.88	0.73	0.41	120.26	0.75	0.74
	28 396 Knoten pro Prozessor										
2	20.70	1.25	9.98	32.12	1.00	1.00	1.00	1.00	20.08	1.00	1.00
8	42.18	1.48	13.28	56.38	0.49	0.85	0.75	0.57	26.20	0.77	0.76
32	74.18	1.64	18.06	93.67	0.28	0.76	0.55	0.34	32.93	0.61	0.59
128	100.96	1.77	21.23	124.35	0.20	0.70	0.47	0.26	38.68	0.52	0.50
	56 793 Knoten pro Prozessor										
1	37.94	2.43	21.88	61.51	1.00	1.00	1.00	1.00	36.52	1.00	1.00
4	63.00	2.67	24.00	88.02	0.60	0.91	0.91	0.70	46.74	0.78	0.83
16	128.02	3.00	33.01	162.63	0.30	0.81	0.66	0.38	59.34	0.62	0.63
64	175.90	3.16	37.93	215.59	0.22	0.77	0.58	0.29	67.92	0.54	0.55
	14 385 Knoten pro Prozessor										
1	7.06	0.48	3.34	10.92	1.00	1.00	1.00	1.00	6.67	1.00	1.00
4	15.15	0.73	5.84	22.39	0.47	0.65	0.57	0.49	11.72	0.57	0.57
16	40.96	0.92	8.30	50.19	0.17	0.52	0.40	0.22	18.99	0.35	0.37
64	61.88	1.01	11.09	74.35	0.11	0.47	0.30	0.15	23.14	0.29	0.29

Tabelle 6.12: Isotroper Laplace Operator auf Wolfgangsee; Scaleup für volle bis achtel Last

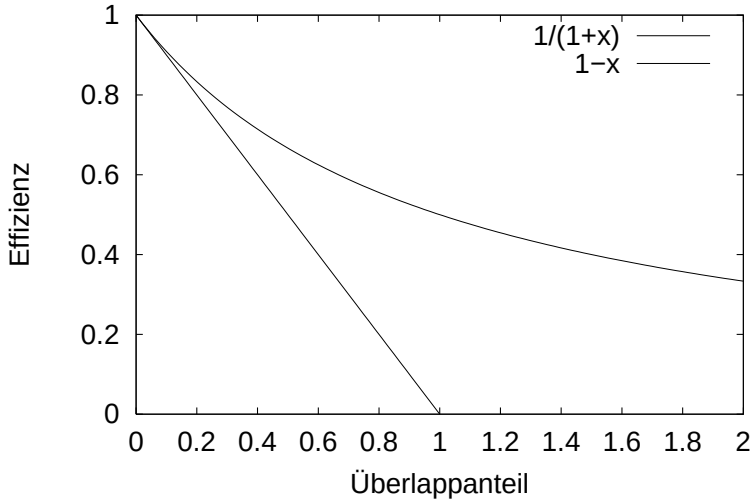


Abbildung 6.8: Effizienzreduktion durch Erhöhung der Knotenzahl

hin denken möchte. Unter der Annahme, daß die Ausführungszeit direkt proportional zur Anzahl der Knoten ist, wirkt sich ein Überlappanteil von ε folgendermaßen auf

die skalierte Effizienz aus:

$$E_\epsilon = \frac{T(1)}{\frac{1}{p}T(p(1+\epsilon))} = \frac{T(1)}{\frac{1}{p}T(p)(1+\epsilon)} = E \cdot \frac{1}{1+\epsilon} \approx E \cdot (1-\epsilon).$$

Diese Näherung gilt für kleine ϵ. Eine Vorstellung gibt Abbildung 6.8. Kleine Erhöhungen der Knotenzahl im Überlapp verringern also direkt proportional die Effizienz. Der erste Eindruck, daß die geringe Erhöhung des Überlapps unbedenklich sei, trog damit.

Level	n	b	$n+b$	m	B/M
	2 Prozessoren				
0	1.00	1.00	1.00	1.00	0.01
-1	1.00	1.15	1.00	1.00	0.01
-2	1.00	1.05	1.00	1.00	0.02
-3	1.00	1.10	1.00	1.00	0.03
-4	1.00	1.00	1.00	1.00	0.04
	8 Prozessoren				
0	1.00	1.25	1.00	1.00	0.02
-1	1.00	1.35	1.01	1.01	0.03
-2	1.01	1.39	1.01	1.02	0.03
-3	1.01	1.40	1.03	1.04	0.05
-4	1.02	1.34	1.04	1.04	0.07
	32 Prozessoren				
0	1.00	1.20	1.00	1.00	0.02
-1	1.00	1.35	1.01	1.02	0.03
-2	1.02	1.44	1.02	1.02	0.04
-3	1.02	1.37	1.03	1.04	0.06
-4	1.02	1.30	1.04	1.05	0.09
	128 Prozessoren				
0	1.01	1.09	1.00	1.00	0.02
-1	1.00	1.27	1.01	1.01	0.04
-2	1.02	1.23	1.02	1.02	0.04
-3	1.02	1.40	1.04	1.05	0.07
-4	1.03	1.32	1.04	1.05	0.09

Level	n	b	$n+b$	m	B/M
	2 Prozessoren				
0	1.00	1.01	1.00	1.00	0.00
-1	1.00	1.02	1.00	1.00	0.00
-2	1.00	1.10	1.00	1.01	0.01
-3	1.00	1.09	1.00	1.01	0.01
-4	1.00	1.11	1.00	1.01	0.02
	8 Prozessoren				
0	1.01	1.26	1.00	1.00	0.02
-1	1.01	1.26	1.01	1.02	0.03
-2	1.01	1.25	1.02	1.02	0.04
-3	1.02	1.20	1.03	1.03	0.07
-4	1.02	1.22	1.03	1.03	0.10
	32 Prozessoren				
0	1.01	1.57	1.01	1.01	0.02
-1	1.05	1.59	1.04	1.06	0.04
-2	1.06	1.54	1.05	1.06	0.06
-3	1.07	1.52	1.06	1.08	0.10
-4	1.07	1.53	1.07	1.09	0.15
	128 Prozessoren				
0	1.01	1.55	1.02	1.01	0.03
-1	1.06	1.52	1.07	1.09	0.05
-2	1.08	1.48	1.09	1.14	0.07
-3	1.10	1.45	1.12	1.15	0.12
-4	1.12	1.46	1.15	1.18	0.17

Tabelle 6.13: Isotroper Laplace Operator; Vergleich des Lastungleichgewichts bei voller Last (links strukturiertes Gitter mit Boxaufteilung, rechts Wolfgangsee)

Ein weiterer Grund ist das Lastungleichgewicht wie in Tabelle 6.13 dargestellt. In den ersten vier Spalten ist jeweils das Verhältnis von maximaler zu durchschnittlicher Anzahl der Masterknoten (n), Borderknoten (b), Knoten insgesamt ($n+b$) und Matrixeinträge (m) angegeben. Je größer dieses Mißverhältnis ist, desto mehr erhöht sich die Laufzeit bzgl. des Idealzustands, weil der Prozessor mit der größten Last an jedem Synchronisationspunkt alle anderen aufhält. Pro Level gibt es mehrere solcher Synchronisationspunkte. Ausschlaggebend sind nur die ersten Level, da sie die

größte Anzahl Knoten und Matrizen enthalten und alle Laufzeiten proportional zur Anzahl der Objekte sind. Die Lastungleichheit für unstrukturierte Gitter (rechte Tabelle von 6.13) ist zwar nur einige Prozent höher als für strukturierte Gitter (linke Tabelle), aber wir haben eben gesehen, daß das schon merkliche Auswirkungen auf die Effizienz haben kann.

Abhilfe kann hier eine levelweise Neuverteilung der Matrix liefern. Da das eine aufwendige Maßnahme ist, soll sie zukünftigen Arbeiten vorbehalten bleiben. Man könnte auch versuchen, den Vergröberungsprozess so zu steuern, daß das initiale Lastgleichgewicht erhalten bleibt. Diese Eigenschaft ist aber nicht direkt beeinflußbar und würde eine Abweichung von der derzeit optimalen Markierungsreihenfolge erfordern; über das Ausmaß der Konvergenzverschlechterung kann a priori keine Aussage getroffen werden. Die Anzahl der Nachbarprozessoren bleibt bei der gewählten Lastverteilung für unstrukturierte Gitter unverändert gegenüber einem strukturierten Gitter, d. h. in einem Datenaustauschschritt sind ungefähr gleichviele Nachrichten zu versenden. Für beide Gitterarten ist der Prozessornachbarschaftsgraph also ähnlich dicht und die Färbungsheuristik erzeugt auch gleich viele Farben, leidet also unter der Unstrukturiertheit nicht. Spalte 5 in Tabelle 6.13 zeigt, daß im unstrukturierten Fall bei der Vergröberung bis zu doppelt soviele Borderknoten erzeugt werden müssen; als Ursache ist die unregelmäßigere Berandung der einzelnen Partitionen anzunehmen.

Zusammenfassend haben wir die Belastungsfaktoren der Effizienz einer Iteration erkannt: leicht zunehmender Überlapp und das Lastungleichgewicht.

Die Gitteraufbauphase (T^*_{FA} bzw. E^*_{FA} in Tabelle 6.12) leidet ebenfalls unter den beiden Faktoren. Sie wird im vergleichbaren Maße wie die Zeit T_{IT} beeinträchtigt; es gibt keine Veranlassung, weitere Einflußfaktoren aufzuspüren.

Die Gesamtzeit des Mehrgitterlösers (T_{SOL} bzw. E_{SOL} in Tabelle 6.12) ist natürlich auf der einen Seite durch die Effizienz E_{IT} eines einzelnen Schritts beschränkt. Auf der anderen Seite ist zu bedenken, daß einer skalierten Betrachung wie dieser hier für große Prozessorzahlen ein entsprechend großes Anfangsgitter zugrunde liegt. Da wir für unstrukturierte Gitter aber eine mit der Problemgröße leicht ansteigende Iterationsanzahl benötigen (Tabelle 6.2), sinkt somit auch die numerische Effizienz und E_{SOL} fällt damit geringer aus als E_{IT}.

Da trotz leicht erhöhter Iterationsanzahl für große Prozessorzahlen die Aufbauphase länger als die Lösungsphase dauert, ist auch die Gesamteffizienz E_{Ges} verständlich. Der so erreichbare Wert von 75% für 128 Prozessoren bzgl. der Zeit von 2 Prozessoren unter voller Last ist sehr gut. Auch für geringere Auslastungen bleibt die Gesamteffizienz über 50%.

Die Auswirkung eines unstrukturierten Gitters ist also eine leichte Abhängigkeit der Konvergenzgüte von der Problemgröße und ein leicht vergrößerter Überlappbereich. Dementsprechend sinkt die Effizienz.

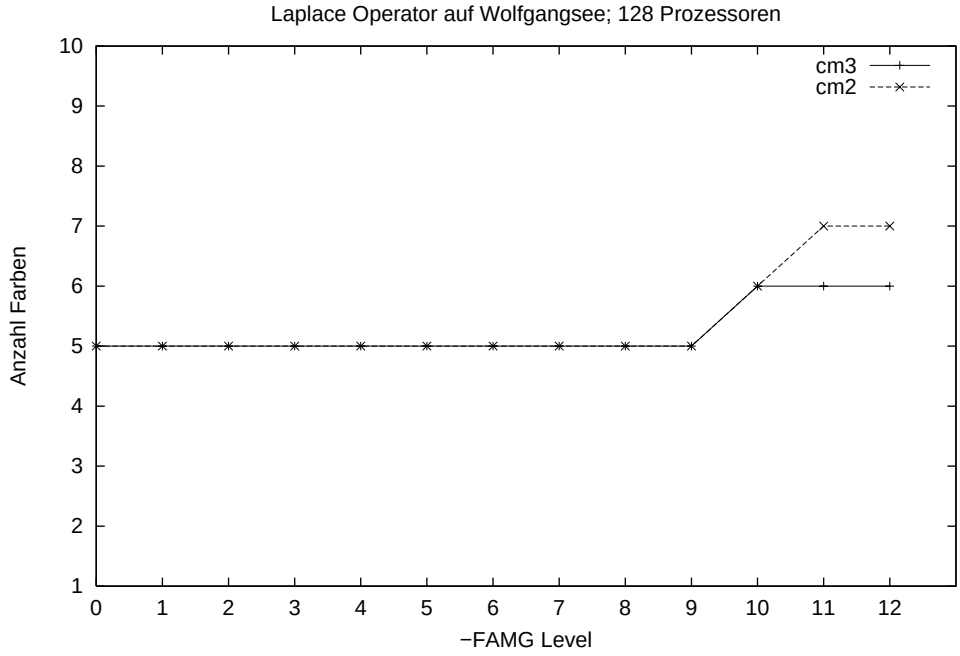


Abbildung 6.9: Isotroper Laplace Operator auf Wolfgangsee; volle/halbe Last; Anzahl benötigter Farben in Abhängigkeit der Färbungsmethode

Nun wollen wir den Einfluß der Graphfärbungsmethode aus Abschnitt 5.6 untersuchen. Die Anzahl der benötigten Farben ist in Abbildung 6.9 für die größten Rechungen dargestellt. Die beiden Methoden cm2 und cm3 unterscheiden sich kaum und erzeugen erstaunlich wenig Farben. Abbildung 6.10 zeigt im oberen Bild, daß alle 3 Färbungsmethoden gleich gute Konvergenz bewirken. Die Zeiten für den Gitteraufbau (mittleres Bild) für kleinere Prozessorzahlen sind für Methode cm2 erwartungsgemäß am kleinsten. Überraschend ist, daß die Kommunikation der Gewichte in cm3 (verbunden mit der synchronisierenden Wirkung) merklich länger dauert als der Zeitverlust aus dem seriellen Abarbeiten der Prozessorinterfaces im Markierungsalgorithmus für cm1. Erst für große Prozessorzahlen überwiegt der sequentialisierende Effekt von cm1 die Kommunikationskosten von cm3. Dementsprechend verhält sich auch die Gesamtzeit (unterstes Bild).

Die Komplexitäten sind nahezu unabhängig von der Färbungsmethode und sind hier nicht wiedergegeben. Die parallele Skalierbarkeit von cm2 ist sehr ähnlich der von cm3 und daher nicht aufgeführt. Für cm1 unterscheidet sich der Speedup ebenfalls kaum, weil das Zeitverhalten von der geringen Prozessorauslastung dominiert wird. Beim Scaleup (vergleiche Tabelle 6.14 mit 6.12) zeigt sich ebenfalls das erwartete Verhalten: Der Mehrgitterlöser ist von der Färbung unberührt und der Gitteraufbau skaliert für kleine Prozessorzahlen etwas besser, aber für große Prozessorzahlen bis zu 25% schlechter.

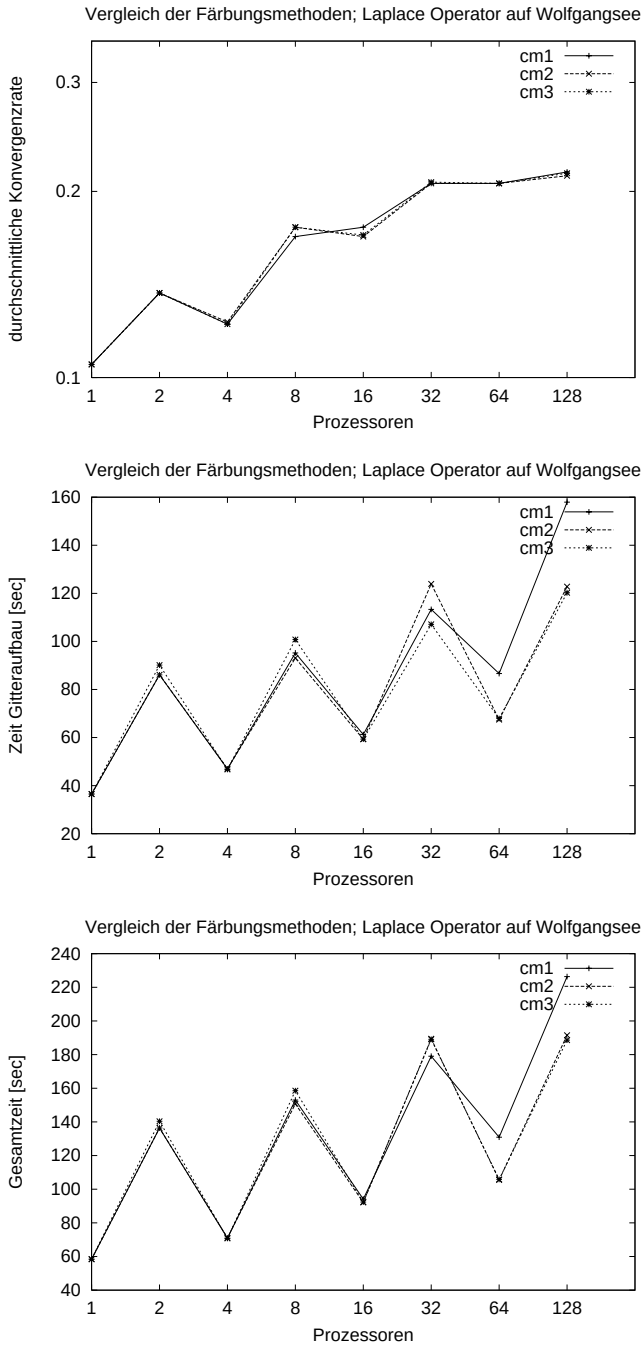


Abbildung 6.10: Isotroper Laplace Operator auf Wolfgangsee; volle Last; Abhängigkeit der durchschnittlichen Konvergenzrate, Gitteraufbauzeit T^*_{FA} und der Gesamtzeit T^*_{Ges} von der Färbungsmethode

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	112 840 Knoten pro Prozessor										
2	88.51	5.03	50.32	140.21	1.00	1.00	1.00	1.00	86.06	1.00	1.00
8	143.14	5.25	57.78	202.68	0.62	0.96	0.87	0.69	95.21	0.90	0.89
32	229.39	5.47	65.69	297.37	0.39	0.92	0.77	0.47	113.24	0.76	0.76
128	319.82	5.70	68.44	390.89	0.28	0.88	0.74	0.36	157.95	0.54	0.60
	28 396 Knoten pro Prozessor										
2	24.41	1.25	9.98	35.84	1.00	1.00	1.00	1.00	23.74	1.00	1.00
8	42.76	1.46	13.17	56.89	0.57	0.85	0.76	0.63	26.67	0.89	0.85
32	77.60	1.59	17.51	96.47	0.31	0.78	0.57	0.37	36.26	0.65	0.63
128	119.79	1.72	20.68	142.86	0.20	0.72	0.48	0.25	57.41	0.41	0.43
	56 793 Knoten pro Prozessor										
1	37.93	2.43	21.88	61.47	1.00	1.00	1.00	1.00	36.52	1.00	1.00
4	63.13	2.67	23.99	88.15	0.60	0.91	0.91	0.70	46.93	0.78	0.82
16	128.84	2.99	32.88	163.23	0.29	0.81	0.67	0.38	61.24	0.60	0.62
64	209.21	3.69	44.27	256.12	0.18	0.66	0.49	0.24	86.61	0.42	0.45
	14 385 Knoten pro Prozessor										
1	6.87	0.48	3.35	10.73	1.00	1.00	1.00	1.00	6.47	1.00	1.00
4	15.14	0.71	5.65	22.19	0.45	0.68	0.59	0.48	11.75	0.55	0.56
16	41.78	0.90	8.12	50.90	0.16	0.53	0.41	0.21	19.59	0.33	0.35
64	68.02	1.01	11.15	80.71	0.10	0.47	0.30	0.13	29.27	0.22	0.24

Tabelle 6.14: Isotroper Laplace Operator auf Wolfgangsee; Färbungsmethode cm1; Scaleup für volle bis achtel Last

6.5 Anisotrope Differentialgleichung

Als Modell einer anisotropen Differentialgleichung verwenden wir den anisotropen Laplace Operator

$$\nabla \cdot \left(\begin{pmatrix} \varepsilon & 0 \\ 0 & 1 \end{pmatrix} \nabla u \right) = 0 \qquad u \text{ auf } \Omega = (0,1)^2,\ \varepsilon \in \mathbb{R}_0^+ \tag{6.10}$$

$$u(x,y) = \tfrac{1}{2}(x+y) \qquad \text{für } (x,y) \in \partial\Omega.$$

Die Diskretisierung erfolgt auf einem Dreiecksgitter mit P1 Elementen, die einen 5–Punkt Stern liefert. Klassischerweise wird die Robustheit gegenüber Variation des Parameters ε untersucht. Für ein paralleles Verfahren muß darüber hinaus auch die Robustheit bzgl. der Anzahl Prozessoren und der Datenaufteilung analysiert werden. Die dabei auftretenden Probleme und bisher verwendeten Löser sind bereits in Abschnitt 2.3.6 diskutiert worden. Auch das FAMG löst die anisotrope Differentialgleichung durch Halbvergröberung in Richtung der starken Kopplung. Die Aufgabe ist so gestellt, daß die Linien der Halbvergröberung über das ganze Gebiet reichen. Diese globale Eigenschaft kann einem parallelen Verfahren grundsätzlich Schwierigkeiten bereiten. Uns interessiert bei der Parallelisierung nun besonders, wie solche Linien der Halbvergröberung über Prozessorgrenzen hinaus fortgesetzt werden können.

Die Abbildungen 6.11 und 6.12 geben einen ersten Eindruck der erzeugten Groblevelmatrizen. Besonders interessant ist die rechte Spalte in Abbildung 6.11, die das Er-

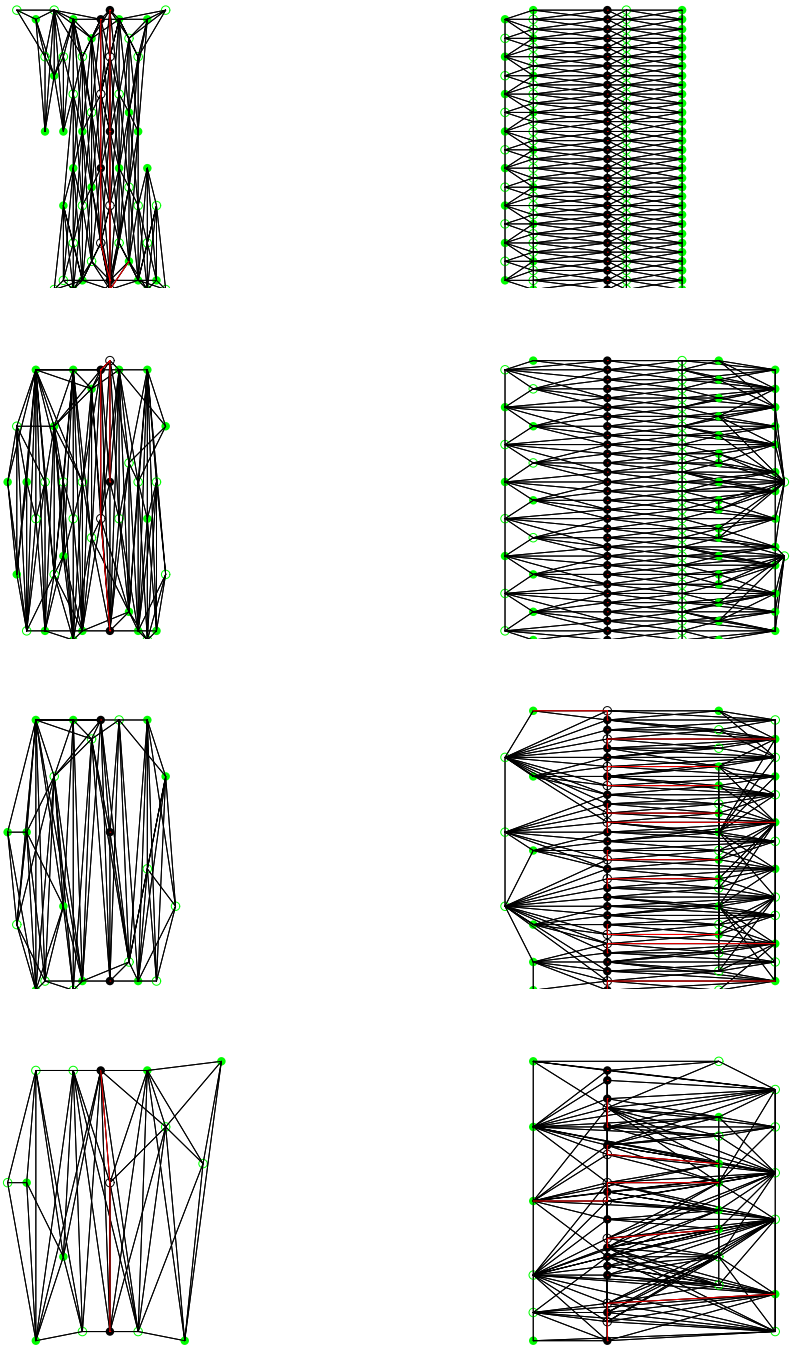


Abbildung 6.11: Matrixgraphen der Level -3 bis -6; Anisotroper Laplace Operator; 16×1, PE 5; links $\varepsilon = 10^{-6}$, rechts $\varepsilon = 10^{+6}$

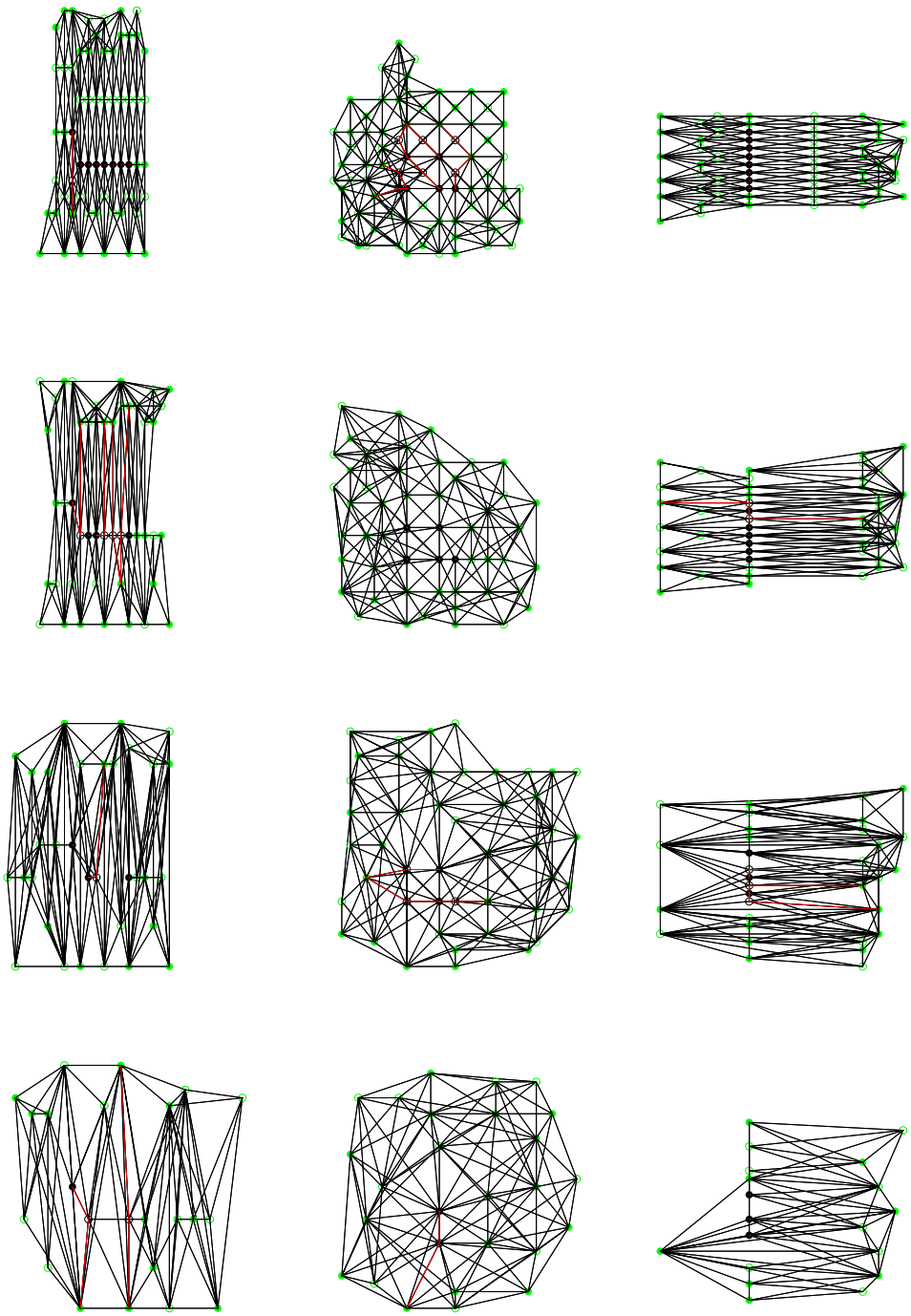


Abbildung 6.12: Matrixgraphen der Level -3 bis -6; Anisotroper Laplace Operator; 4×4, PE 5; links $\varepsilon = 10^{-6}$, mitte $\varepsilon = 1$, rechts $\varepsilon = 10^{+6}$

PE	h / N	1/128 16 641	1/256 66 049	1/512 263 169	1/1024 1 050 625	1/2048 4 198 401	1/4096 16 785 409
1×1		1 $4.8\cdot10^{-13}$	1 $6.9\cdot10^{-11}$				
2×1		1 $5.9\cdot10^{-13}$	1 $3.3\cdot10^{-11}$	1 $2.6\cdot10^{-9}$			
2×2		1 $6.6\cdot10^{-13}$	1 $5.3\cdot10^{-11}$	1 $2.3\cdot10^{-9}$			
4×2		1 $6.6\cdot10^{-13}$	1 $5.3\cdot10^{-11}$	1 $2.3\cdot10^{-9}$	2 $4.7\cdot10^{-6}$		
4×4		1 $6.5\cdot10^{-13}$	1 $5.2\cdot10^{-11}$	1 $2.2\cdot10^{-9}$	2 $4.8\cdot10^{-6}$		
8×4		1 $6.5\cdot10^{-13}$	1 $5.2\cdot10^{-11}$	1 $2.2\cdot10^{-9}$	2 $2.2\cdot10^{-5}$	3 $9.2\cdot10^{-4}$	
8×8		1 $6.2\cdot10^{-13}$	1 $5.2\cdot10^{-11}$	1 $5.7\cdot10^{-9}$	2 $2.2\cdot10^{-5}$	3 0.001	
16×8		1 $6.1\cdot10^{-13}$	1 $5.2\cdot10^{-11}$	1 $5.9\cdot10^{-9}$	2 $2.2\cdot10^{-5}$	3 $9.2\cdot10^{-4}$	3 0.002
1×1		1 $4.8\cdot10^{-13}$	1 $6.9\cdot10^{-11}$				
2×1		1 $5.9\cdot10^{-13}$	1 $3.3\cdot10^{-11}$	1 $2.6\cdot10^{-9}$			
4×1		1 $6.1\cdot10^{-13}$	1 $5.6\cdot10^{-11}$	1 $2.5\cdot10^{-9}$			
8×1		1 $6.2\cdot10^{-13}$	1 $6.4\cdot10^{-11}$	1 $2.5\cdot10^{-9}$	2 $5.3\cdot10^{-6}$		
16×1		1 $6.2\cdot10^{-13}$	1 $6.1\cdot10^{-11}$	1 $2.4\cdot10^{-9}$	2 $5.0\cdot10^{-6}$		
32×1		1 $6.1\cdot10^{-13}$	1 $4.2\cdot10^{-11}$	1 $2.4\cdot10^{-9}$	2 $5.3\cdot10^{-6}$	3 $8.9\cdot10^{-4}$	
64×1		1 $6.4\cdot10^{-13}$	1 $3.1\cdot10^{-11}$	1 $2.2\cdot10^{-9}$	2 $5.4\cdot10^{-6}$	3 $8.9\cdot10^{-4}$	
128×1		1 $6.3\cdot10^{-13}$	1 $2.9\cdot10^{-11}$	1 $2.2\cdot10^{-9}$	2 $5.7\cdot10^{-6}$	3 $8.9\cdot10^{-4}$	3 0.002

Tabelle 6.15: Anisotroper Laplace Operator ($\varepsilon = 10^{-6}$); Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessoraufteilung (Box- und Streifenaufteilung)

gebniss einer Gleichung mit starker horizontaler Kopplung auf einer vertikalen Streifenaufteilung zeigt. Durch den Markierungsablauf werden überwiegend vollständige vertikale Knotenlinien entfernt, so daß ein sehr regelmäßiges Muster entsteht.

Als erstes soll die numerische Güte betrachtet werden. Tabellen 6.15 und 6.16 zeigen für einige charakteristische Werte des Anisotropieparameters die Konvergenzraten. Abbildung 6.13 zeigt einen Überblick über die Werte unter voller bzw. halber Auslastung der Prozessoren. Man erkennt die Robustheit: Die Konvergenzrate ist sowohl bezüglich ε als auch der Prozessorzahl und –aufteilung gleichmäßig durch 0,16 nach oben beschränkt. Die Symmetrie bezüglich ε läßt sich leicht durch eine Koordinatentransformation in (6.10) erklären. Wie von anderen robusten Lösungsverfahren schon bekannt ist, ist die Gleichung für Parameterwerte um $10^{\pm1}$ bis $10^{\pm2}$ am schwierigsten zu lösen. Daher ist ein solcher Fall in Tabelle 6.16 herausgestellt. Ebenso ist auch das Verhalten für extreme Parameterwerte bekannt.

Offensichtlich stellt das Verfahren für extreme Parameterwerte und nicht zu viele Unbekannte einen exakten Löser dar, da in einem Iterationsschritt die Lösung bis (fast) auf Maschinengenauigkeit genau gelöst wird. Die Erklärung durch Entartung zu eindimensionalen Teilproblemen nach Wittums Robustheitstheorie aus Abschnitt 2.3.6.2 greift hier ebenso, wenn auch nicht direkt erkennbar: Die wesentliche Komponente ist der Feinknoten–Jacobiglätter JacobiFine aus Algorithmus 3.5. Abbildung 6.14 stellt den wesentlichen Abschnitt aus Algorithmus 5.22 dar. Ausgangspunkt ist, daß auf dem gröbsten Gitter mittels direktem Löser eine exakte Grobgitterkorrektur berechnet wird. Die exakte Korrektur auf Level $\ell - 1$ wird nun auf den feineren Level ℓ

PE	h 1/128 N 16 641	1/256 66 049	1/512 263 169	1/1024 1 050 625	1/2048 4 198 401	1/4096 16 785 409
1×1	6 0.042	7 0.053				
2×1	6 0.042	7 0.060	7 0.064			
2×2	6 0.044	7 0.059	7 0.068			
4×2	6 0.044	7 0.061	7 0.068	8 0.095		
4×4	6 0.044	7 0.057	7 0.067	9 0.109		
8×4	6 0.044	7 0.056	8 0.083	9 0.120	9 0.127	
8×8	6 0.045	7 0.055	8 0.087	9 0.119	9 0.127	
16×8	6 0.043	7 0.062	8 0.100	9 0.110	9 0.127	10 0.153
1×1	6 0.042	7 0.053				
2×1	6 0.042	7 0.060	7 0.064			
4×1	6 0.043	7 0.059	7 0.064			
8×1	6 0.043	7 0.056	7 0.069	8 0.088		
16×1	6 0.041	7 0.065	7 0.070	8 0.100		
32×1	6 0.036	7 0.060	8 0.091	8 0.093	9 0.120	
64×1	6 0.041	6 0.044	8 0.091	8 0.099	9 0.123	
128×1	6 0.042	7 0.052	7 0.063	9 0.121	10 0.148	10 0.150

Tabelle 6.16: Anisotroper Laplace Operator ($\varepsilon = 10^{+1}$); Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessoraufteilung

prolongiert. Abbildung 6.15 veranschaulicht die Situation. Auf den groben Knoten (i,k) liegt der exakte Wert bereits vor ($p_{i'i} \equiv 1$), auf den feinen (j) nur eine Näherung. JacobiFine ändert die groben Werte nicht, auf den feinen wird c_j aus

$$a_{jj}c_j + a_{ji}c_i + a_{jk}c_k = d_j$$

berechnet. Da c_i und c_k exakt sind, wird auch für c_j der exakte Wert berechnet. Wichtig ist festzustellen, daß durch die durchgeführte Halbvergröberung *alle* Nachbarknoten eines feinen Knotens für den Grenzfall $\varepsilon = 0$ grobe Knoten sind, und somit *jeder* feine Knoten seinen exakten Wert erhält, insgesamt hinterläßt JacobiFine auf Level ℓ die exakte Korrektur. Da Glätter üblicherweise konsistente Iterationsverfahren sind (speziell das hier verwendete Jacobiverfahren), wird die exakte Korrektur im Nachglättungsschritt nicht mehr verändert. Rekursiv angewandt zeigt diese Betrachtung, daß im asymptotischen Fall $\varepsilon = 0$ das FAMG Verfahren zu einem exakten Löser wird.

Für eindimensionale Probleme tritt diese Eigenschaft auch bei einem **Hierarchische Basis Mehrgitterverfahren** auf, bei dem überhaupt nur die feinen Knoten geglättet werden. Asymptotisch für $\varepsilon \to 0$ entartet die anisotrope zweidimensionale Gleichung zu einer Menge eindimensionaler Gleichungen, so daß ebenfalls alle Nachbarn eines feinen Knotens grob sind (siehe [BDY88] und [BW93]).

Abbildung 6.16 zeigt den Einfluß der Gitterweite h auf die Robustheit. Für nahezu isotrope Situationen ist die Konvergenzrate h–unabhängig. Für mittlere Parameterwerte verschlechtert sich die Konvergenzrate nur sehr langsam mit h und bleibt auch

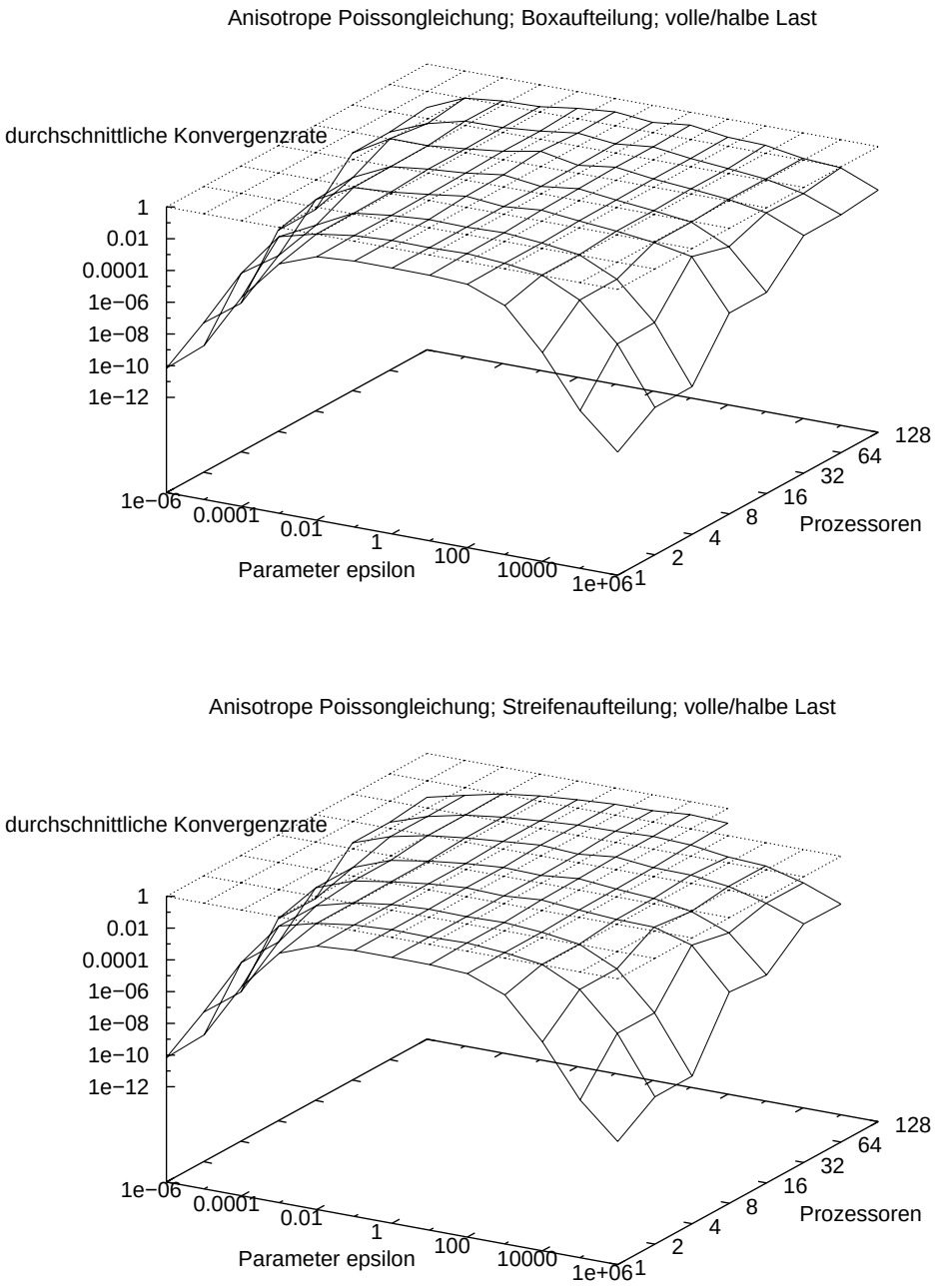


Abbildung 6.13: Anisotroper Laplace Operator; oben Boxaufteilung, unten Streifenaufteilung; volle bzw. halbe Last; Abhängigkeit der mittleren Konvergenzrate vom Anisotropieparameter ε und der Prozessorzahl

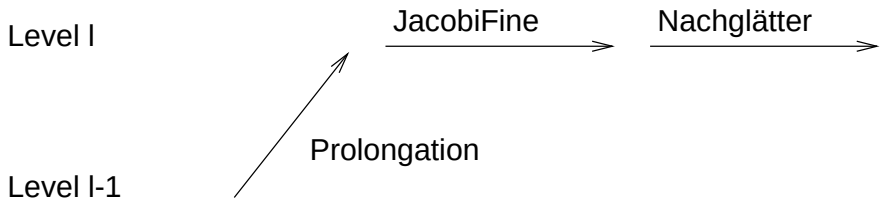


Abbildung 6.14: Detailsituation der Grobgitterkorrektur

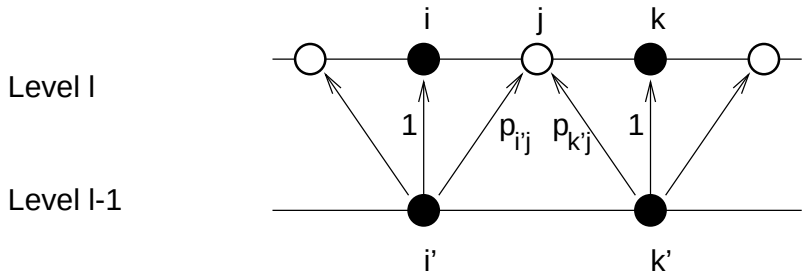


Abbildung 6.15: Prolongierte Lösungskorrektur

für sehr feine Gitter durch den sehr guten Wert 0,16 beschränkt. Für extreme Parameterwerte geht die oben analysierte Eigenschaft eines exakten Lösers für kleiner werdendes h langsam verloren, weil auf sehr groben Leveln durch die Halbvergröberung die Anisotropie kompensiert wird und die Matrizen einer isotropen Poissongleichung entsprechen. Die Konvergenzrate nähert sich folglich dem isotropen Fall, bleibt also auch durch einen sehr guten Wert nach oben beschränkt. Die Robustheit gilt also h–unabhängig für Box– und Streifenaufteilung.

Abbildung 6.17 zeigt im Überblick, daß die Konvergenzrate (fast) unabhängig von der Prozessorzahl ist, d. h. die parallele Skalierbarkeit der numerischen Güteeigenschaften ist gesichert.

Betrachtet man die Speicherkomplexität der anisotropen Modellfälle in den Tabellen 6.17 und 6.18 im Vergleich zum isotropen Fall in Tabelle 6.4, so bleibt die Gitterkomplexität c_G unverändert bei ihrem sehr guten Wert 2; d. h. die Vergröberungsleistung ist in allen Fällen optimal. Der Überlappanteil B/M wächst, je weiter ε von 1 entfernt ist. Das ist auf die in diesen Fällen deutlich erhöhte Überlappkomplexität $c_{\ddot{U}}$ zurückzuführen, die das gleiche Verhalten aber deutlicher ausgeprägt zeigt. Ursache hierfür ist die Halbvergröberung in Verbindung mit der Vollständigkeit des Überlapps. Da der Vergröberungsfaktor in allen Fällen ca. 2 beträgt, liegt auf gleichen Leveln jeweils eine im wesentlichen konstante Knotendichte vor. Durch die Halbvergröberung verbinden die Matrixeinträge Knoten miteinander, deren Entfernung sich mit jedem Vergröberungsschritt verdoppelt. Dadurch wächst für starke Anisotropie

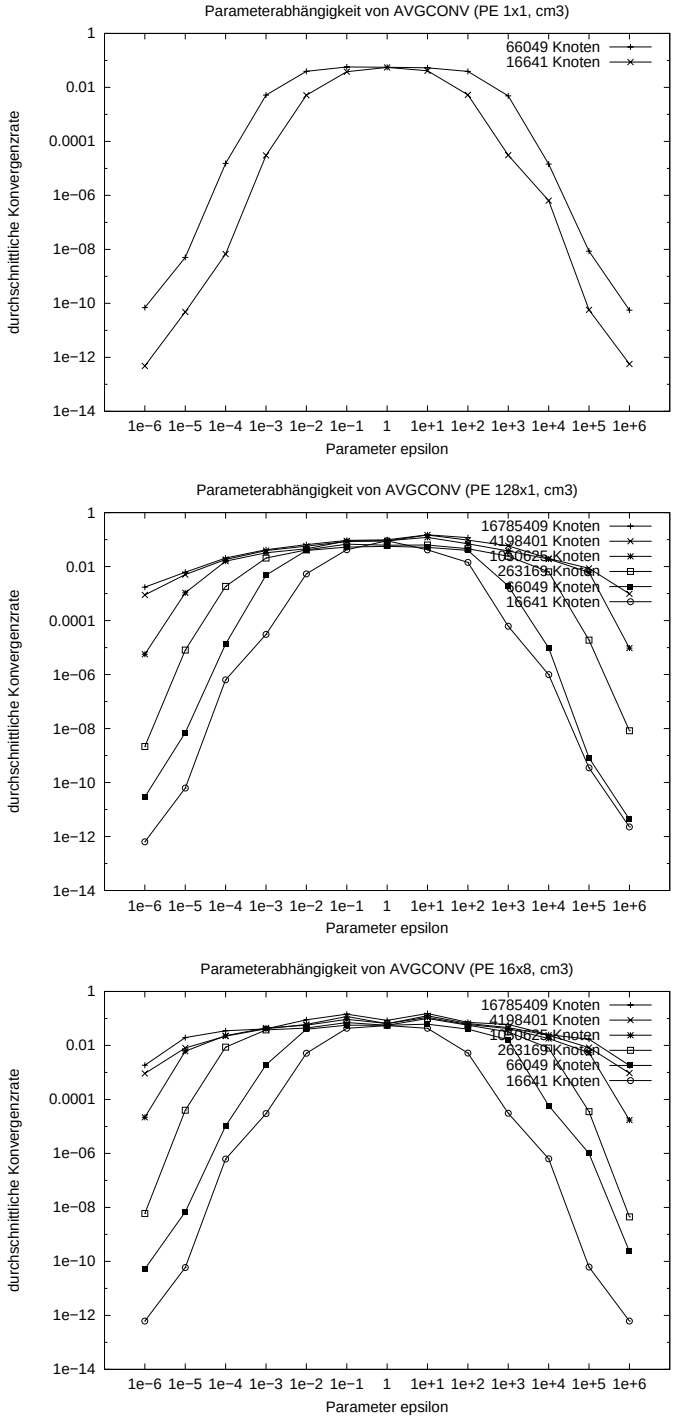


Abbildung 6.16: Anisotroper Laplace Operator; Abhängigkeit der mittleren Konvergenzrate vom Anisotropieparameter ε; verschiedene Knotenzahlen zu den Prozessoraufteilungen 1×1, 128×1 und 16×8

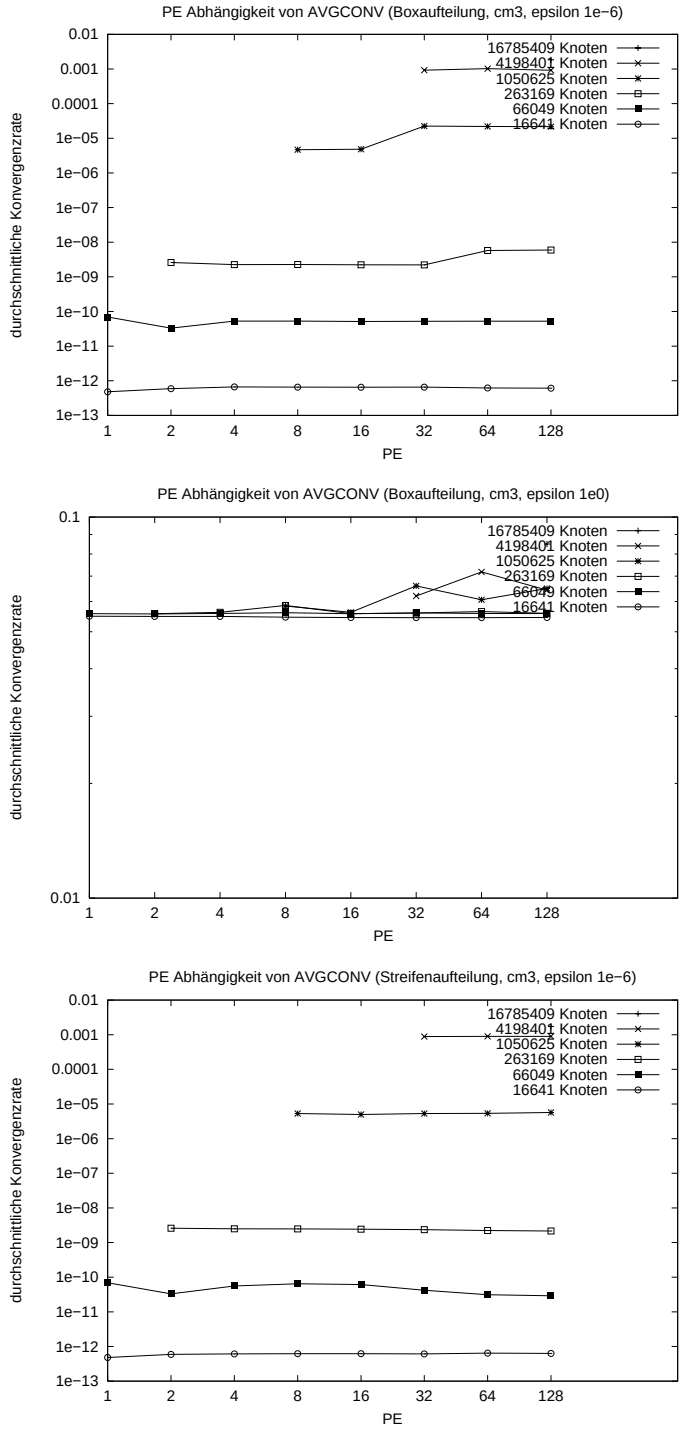


Abbildung 6.17: Anisotroper Laplace Operator; Abhängigkeit der mittleren Konvergenzrate von der Prozessorzahl; verschiedene Gitterweiten zu ($\varepsilon = 1$, Box), ($\varepsilon = 10$, Box) und ($\varepsilon = 10^{-6}$, Streifen)

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.28	1.98	2.00	0.01
8	2.34	1.99	4.26	0.03
32	2.37	2.00	5.05	0.05
128	2.38	2.01	5.46	0.06
	32 768 Knoten/PE			
2	2.20	1.93	1.94	0.02
8	2.34	1.98	3.56	0.06
32	2.41	2.00	4.58	0.09
128	2.45	2.01	5.13	0.11

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.20	1.93	0.00	0.00
4	2.33	1.98	5.04	0.04
16	2.39	2.00	5.69	0.07
64	2.44	2.01	6.75	0.09
	16 384 Knoten/PE			
1	1.94	1.74	0.00	0.00
4	2.24	1.93	3.27	0.05
16	2.41	1.99	4.67	0.11
64	2.53	2.01	6.01	0.16

Tabelle 6.17: Laplace Operator ($\varepsilon = 10^{-6}$); Speicherkomplexität für volle bis achtel Last (Scaleup)

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.36	1.98	8.87	0.03
8	2.40	2.00	7.75	0.06
32	2.45	2.01	8.74	0.09
128	2.46	2.01	8.94	0.10
	32 768 Knoten/PE			
2	2.29	1.93	5.82	0.05
8	2.42	1.99	6.17	0.10
32	2.54	2.01	7.65	0.15
128	2.58	2.02	8.19	0.18

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.22	1.93	0.00	0.00
4	2.35	1.98	5.20	0.04
16	2.40	2.00	5.85	0.07
64	2.44	2.01	6.80	0.09
	16 384 Knoten/PE			
1	1.95	1.74	0.00	0.00
4	2.26	1.93	3.50	0.06
16	2.42	1.99	4.82	0.11
64	2.54	2.01	6.12	0.17

Tabelle 6.18: Laplace Operator ($\varepsilon = 10^{+6}$); Speicherkomplexität für volle bis achtel Last (Scaleup)

die von einem Punkt aus erreichbare Fläche wesentlich stärker als bei einer isotropen Vergröberung, bei der sich die Entfernungszunahme gleichmäßig auf alle Raumdimensionen verteilen muß. Durch die Auffächerung der Verbindungen quer zur stark gekoppelten Richtung, wie in den Abbildungen 6.11 und 6.12 zu sehen ist, wird ein Punkt mit allen anderen innerhalb der erreichbaren Fläche gekoppelt. Durch die Vollständigkeitsforderung für den Überlapp der Tiefe 2 werden so übermäßig große Flächen als Überlapp erzeugt mit allen darin befindlichen Knoten. Dadurch wächst die Größe des Überlapps beträchtlich. Da sich aber die durchschnittliche Sterngröße nicht wesentlich erhöht, wächst die Operatorkomplexität c_A nur schwach an, d. h. der Rechenaufwand skaliert gut mit ε.

Für eine Boxaufteilung mit quadratischen Partitionen, d. h. die Prozessorzahlen 1, 4, 16, 64 gelten die Aussagen vollkommen symmetrisch für $\varepsilon = 10^{+e}$ und $\varepsilon = 10^{-e}$. Für die anderen Prozessorzahlen (2, 8, 32, 128) sind die Boxen bereits Rechtecke mit dem Seitenverhältnis 1:2; dabei wirken sich schon Effekte merklich aus, die für den Extremfall einer Streifenaufteilung detailiert in Abschnitt 6.5.1 analysiert werden, in

dem die verstärkte Zunahme erklärt wird.

Das Lastungleichgewicht ist nur leicht erhöht und beeinträchtigt die Skalierbarkeit nicht merklich.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	58.13	5.44	5.44	64.44	1.00	1.00	1.00	1.00	53.85	1.00	1.00
8	104.58	5.84	11.69	117.23	0.56	0.93	0.47	0.55	62.18	0.87	0.80
32	164.97	6.18	18.53	184.57	0.35	0.88	0.29	0.35	70.95	0.76	0.66
128	211.30	6.33	19.00	231.45	0.28	0.86	0.29	0.28	82.84	0.65	0.58
	32 768 Knoten pro Prozessor										
2	15.07	1.36	1.36	16.91	1.00	1.00	1.00	1.00	13.86	1.00	1.00
8	30.29	1.62	1.62	32.37	0.50	0.84	0.84	0.52	17.95	0.77	0.78
32	54.19	1.74	3.48	58.02	0.28	0.78	0.39	0.29	22.45	0.62	0.59
128	75.82	1.86	5.58	81.91	0.20	0.73	0.24	0.21	29.72	0.47	0.43
	65 536 Knoten pro Prozessor										
1	22.38	2.53	2.53	25.53	1.00	1.00	1.00	1.00	21.35	1.00	1.00
4	42.81	2.91	2.91	46.34	0.52	0.87	0.87	0.55	32.22	0.66	0.68
16	78.60	3.19	6.37	85.60	0.28	0.79	0.40	0.30	38.33	0.56	0.53
64	120.68	3.50	10.49	131.88	0.19	0.72	0.24	0.19	49.06	0.44	0.40
	16 384 Knoten pro Prozessor										
1	4.38	0.55	0.55	5.39	1.00	1.00	1.00	1.00	4.16	1.00	1.00
4	11.34	0.78	0.78	12.63	0.39	0.71	0.71	0.43	9.10	0.46	0.48
16	24.15	0.93	0.93	25.48	0.18	0.59	0.59	0.21	12.81	0.32	0.34
64	41.71	1.09	2.19	44.24	0.10	0.50	0.25	0.12	17.23	0.24	0.24

Tabelle 6.19: Laplace Operator ($\varepsilon = 10^{-6}$); Scaleup für volle bis achtel Last

Wie in den Tabellen 6.19 und 6.20 gezeigt wird, sind die parallelen Skalierungseigenschaften der Zeiten ähnlich denen im isotropen Fall (vergleiche Tabelle 6.6).

Die Effizienz für einen Schritt E_{IT} verschlechtert sich aufgrund der wachsenden Interfacelänge bei steigender Anisotropie etwas. Die Anzahl der Iterationen erhöht sich leicht mit der Prozessoranzahl, weil in einer Scaleup Betrachtung damit eine entsprechende Knotenzahlerhöhung einhergeht und dafür wenige Iterationen zusätzlich benötigt werden (siehe Tabelle 6.15). Für starke Anisotropie bedeutet diese kleine absolute Erhöhung bezogen auf die Basis von einer Iteration für wenige Prozessoren aber eine Vervielfachung; die Auswirkungen auf die Effizienz E_{SOL} sind dementsprechend stark.

Auch die Gitteraufbauphase leidet unter der übermäßig wachsenden Interfacelänge. Damit geht auch ein dichterer Färbungsgraph einher, so daß die benötigte Anzahl Farben merklich steigt und die teilsequentialisierte Färbung im Bereich des Prozessorinterfaces länger dauert.

Abbildung 6.18 faßt das eben besprochene Zeitverhalten zusammen.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	71.73	5.72	5.72	78.47	1.00	1.00	1.00	1.00	59.95	1.00	1.00
8	137.65	6.25	12.51	151.22	0.52	0.91	0.46	0.52	71.74	0.84	0.78
32	222.07	6.66	19.97	243.13	0.32	0.86	0.29	0.32	90.73	0.66	0.59
128	317.24	8.98	26.94	345.56	0.23	0.64	0.21	0.23	120.37	0.50	0.45
	32 768 Knoten pro Prozessor										
2	18.02	1.45	1.45	19.98	1.00	1.00	1.00	1.00	15.51	1.00	1.00
8	39.14	1.71	1.71	41.38	0.46	0.85	0.85	0.48	21.35	0.73	0.74
32	70.61	1.90	3.80	74.84	0.26	0.76	0.38	0.27	29.57	0.52	0.51
128	111.83	2.15	6.44	118.89	0.16	0.67	0.22	0.17	45.21	0.34	0.33
	65 536 Knoten pro Prozessor										
1	22.96	2.54	2.54	26.08	1.00	1.00	1.00	1.00	21.93	1.00	1.00
4	43.67	2.93	2.93	47.23	0.53	0.87	0.87	0.55	31.24	0.70	0.72
16	79.02	3.19	6.38	86.02	0.29	0.80	0.40	0.30	38.47	0.57	0.55
64	120.81	3.46	10.39	131.91	0.19	0.73	0.24	0.20	49.06	0.45	0.41
	16 384 Knoten pro Prozessor										
1	4.41	0.55	0.55	5.48	1.00	1.00	1.00	1.00	4.19	1.00	1.00
4	11.44	0.78	0.78	12.75	0.39	0.71	0.71	0.43	8.55	0.49	0.51
16	24.07	1.17	1.17	25.65	0.18	0.47	0.47	0.21	12.77	0.33	0.34
64	42.01	1.05	2.11	44.45	0.11	0.52	0.26	0.12	17.77	0.24	0.24

Tabelle 6.20: Laplace Operator ($\varepsilon = 10^{+6}$); Scaleup für volle bis achtel Last

6.5.1 Einfluß der Gebietsaufteilung

Der Einfluß der Gebietsaufteilung soll nun untersucht werden. Von Vorteil ist dabei das strukturierte Gitter auf einem sehr einfachen Grundgebiet, weil so verschiedene Aufteilungen der (Elemente und damit) Knoten sehr einfach realisiert werden können. Die zwei extremen Formen (Box– und Streifenaufteilung) wurden bereits eingeführt und werden hier gegenübergestellt. Abbildung 6.13 zeigte bereits, daß die Konvergenzgüte unabhängig von der Aufteilung ist.

Im folgenden wird die Hauptunterscheidung immer sein, ob die Richtung der starken Kopplung entlang der Gebietsstreifen (das entspricht $\varepsilon < 1$ in unserem Modellproblem) oder senkrecht dazu verläuft (das entspricht $\varepsilon > 1$). In diesem Sinne wird die Boxaufteilung, v. a. für die Fälle mit rechteckigen Partitionen, eine Zwischenstellung einnehmen.

Die Komplexitäten sind in den Tabellen 6.21 bis 6.23 angegeben. Die Gitterkomplexität c_G beträgt auch hier unverändert ca. 2 und zeigt damit eine anhaltend gute Vergröberungsleistung an. Problematisch ist der Überlapp. Beide Indikatoren ($c_{\ddot{U}}$ und B/M) zeigen, daß mit zunehmender Anisotropiestärke senkrecht zur Streifenrichtung der Überlapp sehr stark anwächst. Die Ursache ist die gleiche, wie oben bereits ausgeführt: In Richtung der Halbvergröberung steigt die Zahl der Borderknoten stark an. Je größer die Prozessorinterfacelänge ist, über die hinweg eine starke Anisotropie vorliegt, desto stärker ist die überlappvergrößernde Wirkung. Für $\varepsilon = 10^{+6}$ und schmale

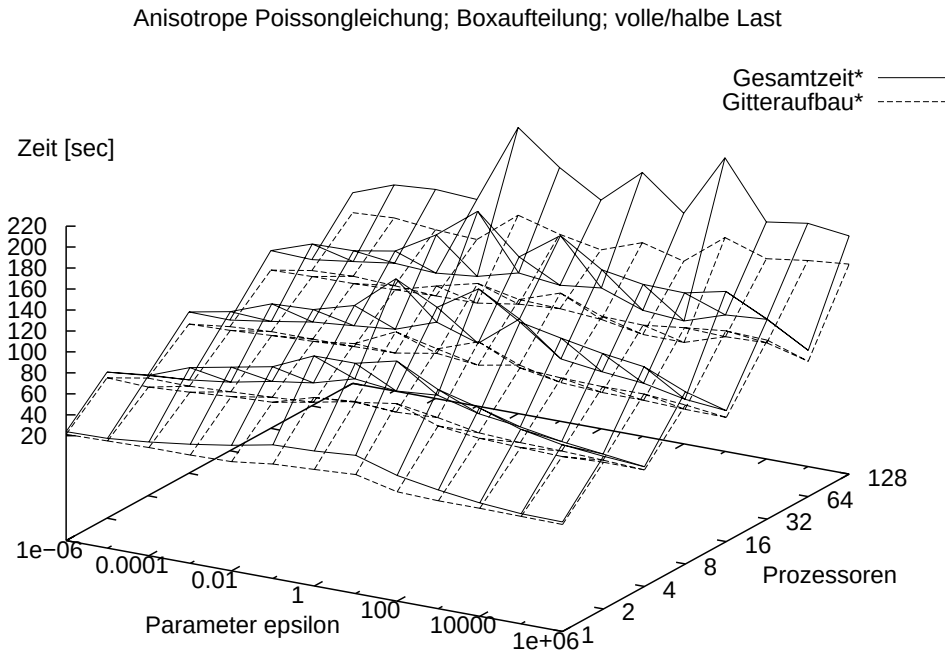


Abbildung 6.18: Anisotroper Laplace Operator; Boxaufteilung; volle bzw. halbe Last; Abhängigkeit der Laufzeiten vom Anisotropieparameter ε und der Prozessorzahl

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.28	1.98	2.00	0.01
8	2.31	1.99	2.01	0.03
32	2.32	2.00	2.01	0.06
128	2.33	2.00	2.01	0.12
	32 768 Knoten/PE			
2	2.20	1.93	1.94	0.02
8	2.28	1.98	2.00	0.06
32	2.31	1.99	2.01	0.12
128	2.33	2.00	2.01	0.25

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.20	1.93	0.00	0.00
4	2.28	1.98	2.00	0.02
16	2.31	1.99	2.01	0.06
64	2.32	2.00	2.01	0.12
	16 384 Knoten/PE			
1	1.94	1.74	0.00	0.00
4	2.20	1.93	1.94	0.05
16	2.29	1.98	2.00	0.12
64	2.31	1.99	2.01	0.25

Tabelle 6.21: Anisotroper Laplace Operator ($\varepsilon = 10^{-6}$); Speicherkomplexität für volle bis achtel Last (Scaleup) für die Streifenaufteilung

Streifen (d. h. 128 Prozessoren) existieren mehr Border– als Masterknoten und lassen eine äußerst schlechte parallele Skalierbarkeit befürchten. Der Speichermehraufwand wird dort so groß, daß bei voller Ausgangslast die nötigen Kommunikationspuffer

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.16	1.98	3.51	0.01
8	2.20	2.00	3.58	0.05
32	2.30	2.01	3.82	0.12
128	2.41	2.01	3.84	0.24
	32 768 Knoten/PE			
2	2.10	1.92	3.07	0.02
8	2.22	1.99	3.40	0.09
32	2.39	2.01	3.71	0.22
128	2.62	2.02	3.82	0.47

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.11	1.92	0.00	0.00
4	2.19	1.98	3.49	0.04
16	2.26	2.00	3.58	0.10
64	2.42	2.01	3.74	0.23
	16 384 Knoten/PE			
1	1.85	1.73	0.00	0.00
4	2.13	1.93	3.07	0.07
16	2.30	1.99	3.37	0.20
64	2.60	2.03	3.68	0.45

Tabelle 6.22: Laplace Operator ($\varepsilon = 1$); Speicherkomplexität für volle bis achtel Last (Scaleup) für die Streifenaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.36	1.98	8.87	0.03
8	2.58	2.02	10.86	0.15
32	2.95	2.05	11.84	0.35
	32 768 Knoten/PE			
2	2.29	1.93	5.82	0.05
8	2.68	2.02	8.51	0.23
32	3.31	2.08	10.19	0.59
128	4.20	2.12	10.80	1.26

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.22	1.93	0.00	0.00
4	2.43	1.99	7.83	0.09
16	2.84	2.04	10.45	0.30
64	3.45	2.08	11.34	0.67
	16 384 Knoten/PE			
1	1.95	1.74	0.00	0.00
4	2.36	1.94	5.10	0.12
16	3.12	2.05	8.74	0.50
64	4.05	2.10	10.04	1.17

Tabelle 6.23: Anisotroper Laplace Operator ($\varepsilon = 10^{+6}$); Speicherkomplexität für volle bis achtel Last (Scaleup) für die Streifenaufteilung

nicht mehr angelegt werden konnten und die Rechnung daher nicht durchgeführt werden konnte.

Die Interfacelängen sind in den Abbildungen 6.19 bis 6.21 quantitativ und in der Abbildung 6.22 als qualitative Übersicht angegeben und geben einen Eindruck vom Ausmaß der Vergrößerung. Umgekehrt geht für $\varepsilon \ll 1$ und Streifenaufteilung keine starke Kopplung über einen Prozessorrand. Der Überlapp ist daher kleiner als für eine Boxaufteilung. Da der Überlapp vollständig erzeugt wird, verhält sich die Anzahl der Matrixeinträge dementsprechend und bestimmt die Operatorkomplexität c_A.

Die parallele Skalierbarkeit ist aus den Tabellen 6.24 bis 6.26 ersichtlich. Für starke Kopplung entlang der Streifen ($\varepsilon \ll 1$) skaliert die Zeit für eine Iteration (E_{IT}) ähnlich wie für die Boxaufteilung (vergleiche Tabelle 6.19). Die Belastung durch die größere Interfacelänge bei Streifenaufteilung entspricht in etwa dem vergrößerten Überlapp über die (schmalere) Boxseite. Die Effizienz des Mehrgitterlösers E_{SOL} wird durch den absolut betrachtet geringen Anstieg von 1 auf 3 Iterationsschritte dominiert. Die

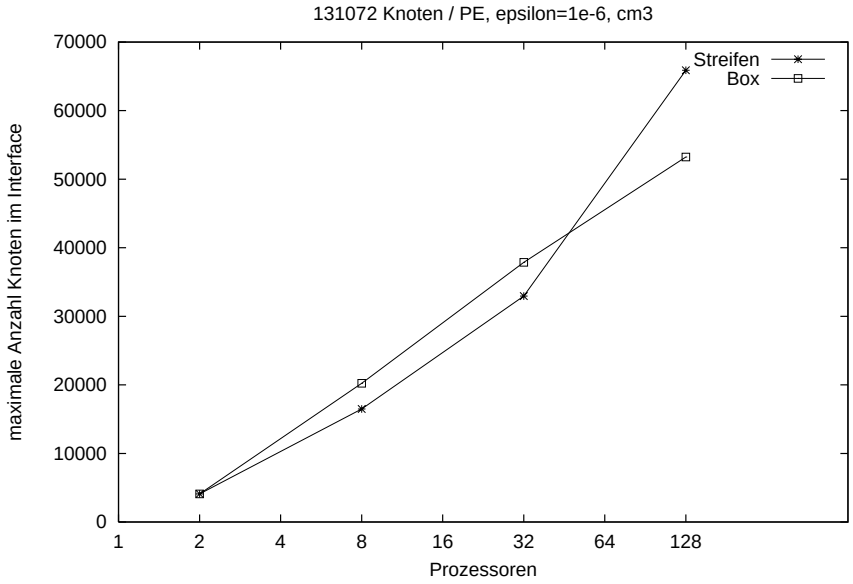


Abbildung 6.19: Anisotroper Laplace Operator ($\varepsilon = 10^{-6}$); volle Last; Abhängigkeit der Interfacelänge von der Gebietsaufteilung und der Prozessorzahl

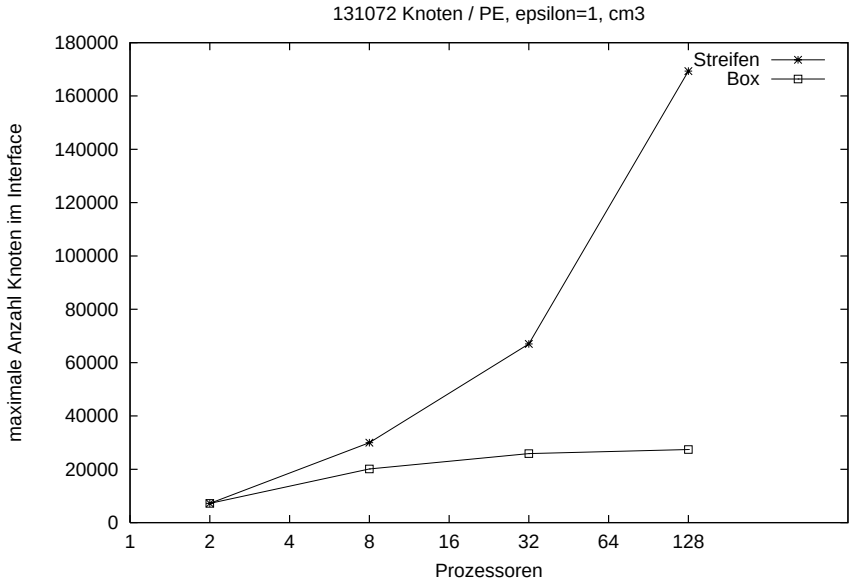


Abbildung 6.20: Anisotroper Laplace Operator ($\varepsilon = 1$); volle Last; Abhängigkeit der Interfacelänge von der Gebietsaufteilung und der Prozessorzahl

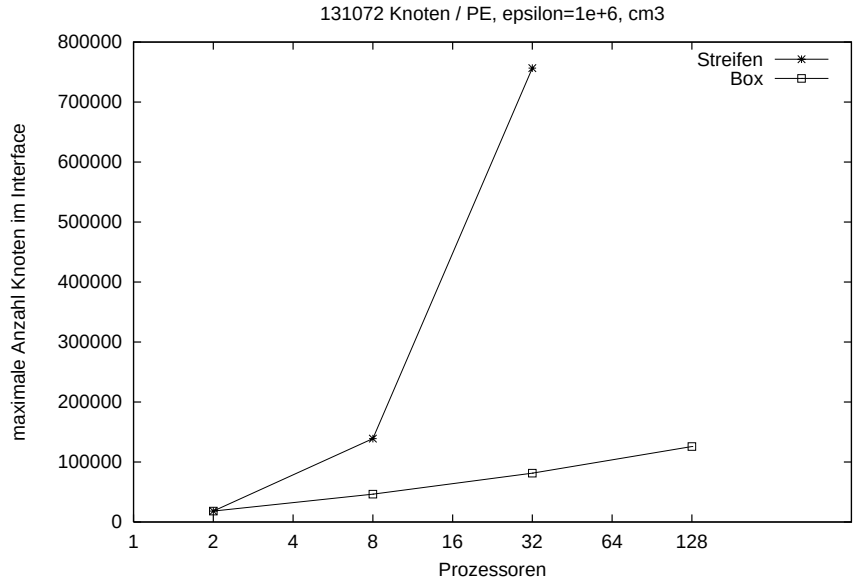


Abbildung 6.21: Anisotroper Laplace Operator ($\varepsilon = 10^{+6}$); volle Last; Abhängigkeit der Interfacelänge von der Gebietsaufteilung und der Prozessorzahl

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	58.13	5.44	5.44	64.44	1.00	1.00	1.00	1.00	53.85	1.00	1.00
8	120.60	5.80	11.60	133.14	0.48	0.94	0.47	0.48	64.23	0.84	0.78
32	235.72	6.17	18.50	255.28	0.25	0.88	0.29	0.25	81.92	0.66	0.59
128	560.52	6.78	20.35	582.11	0.10	0.80	0.27	0.11	123.32	0.44	0.41
	32 768 Knoten pro Prozessor										
2	15.07	1.36	1.36	16.91	1.00	1.00	1.00	1.00	13.86	1.00	1.00
8	37.18	1.56	1.56	39.08	0.41	0.87	0.87	0.43	19.58	0.71	0.72
32	83.39	1.76	3.52	87.28	0.18	0.78	0.39	0.19	27.77	0.50	0.49
128	199.17	2.10	6.29	205.91	0.08	0.65	0.22	0.08	47.41	0.29	0.28
	65 536 Knoten pro Prozessor										
1	22.38	2.53	2.53	25.53	1.00	1.00	1.00	1.00	21.35	1.00	1.00
4	48.33	2.85	2.85	51.70	0.46	0.89	0.89	0.49	30.71	0.70	0.71
16	95.10	3.10	6.19	101.84	0.24	0.82	0.41	0.25	40.09	0.53	0.52
64	210.04	3.43	10.28	220.95	0.11	0.74	0.25	0.12	57.66	0.37	0.35
	16 384 Knoten pro Prozessor										
1	4.38	0.55	0.55	5.39	1.00	1.00	1.00	1.00	4.16	1.00	1.00
4	13.96	0.77	0.77	15.13	0.31	0.71	0.71	0.36	9.19	0.45	0.47
16	31.80	0.91	0.91	32.98	0.14	0.60	0.60	0.16	13.68	0.30	0.32
64	78.17	1.09	2.18	80.66	0.06	0.50	0.25	0.07	22.34	0.19	0.19

Tabelle 6.24: Anisotroper Laplace Operator ($\varepsilon = 10^{-6}$); Scaleup für volle bis achtel Last für die Streifenaufteilung

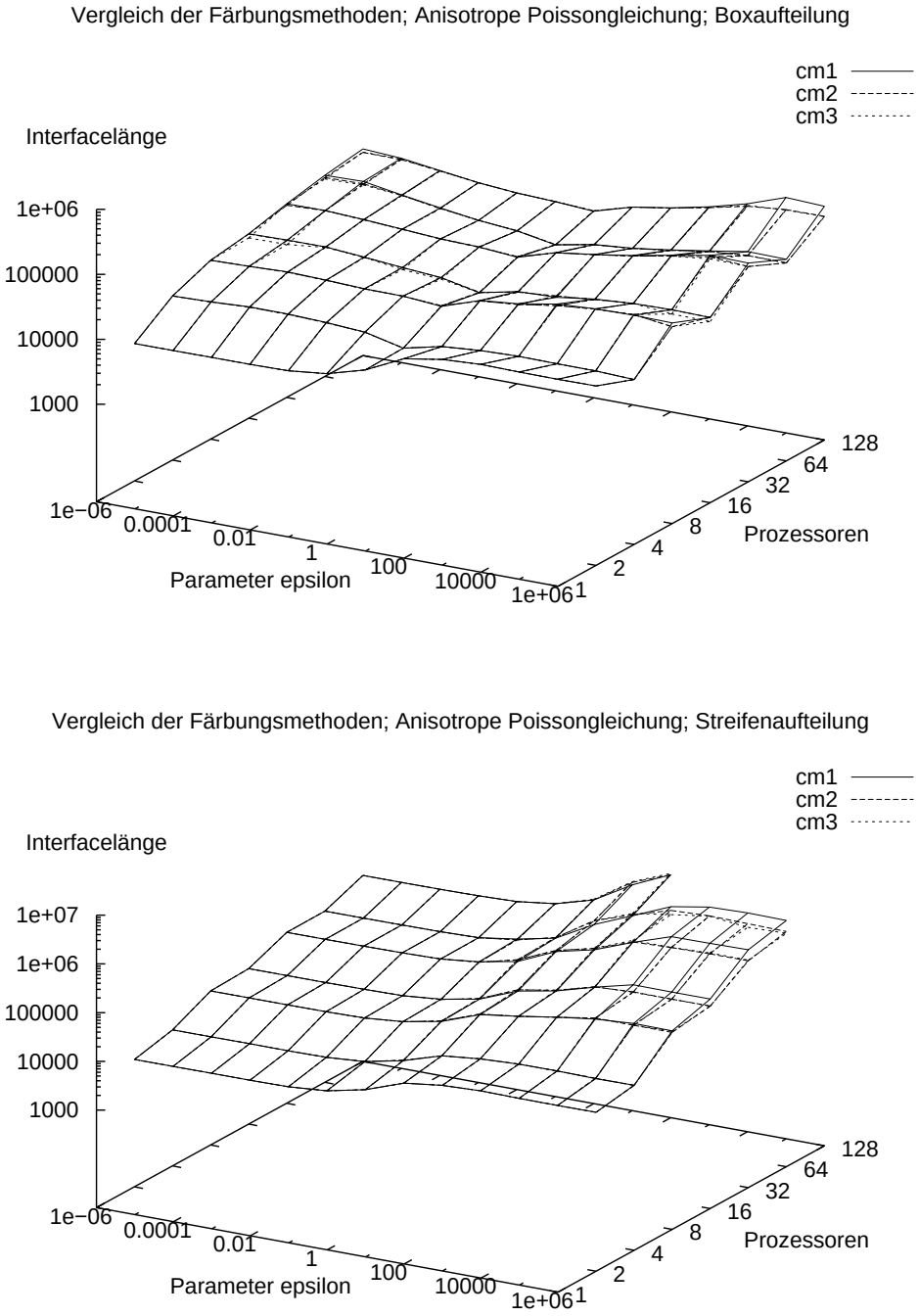


Abbildung 6.22: Anisotroper Laplace Operator; volle/halbe Last; Boxaufteilung (oben), Streifenaufteilung (unten); Abhängigkeit der Interfacelänge von der Färbungsmethode

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	80.65	5.32	37.26	121.04	1.00	1.00	1.00	1.00	73.86	1.00	1.00
8	164.41	5.87	41.12	209.75	0.49	0.91	0.91	0.58	88.85	0.83	0.85
32	355.65	6.75	47.22	408.37	0.23	0.79	0.79	0.30	120.10	0.61	0.66
128	838.10	8.01	64.09	912.47	0.10	0.66	0.58	0.13	205.91	0.36	0.41
	32 768 Knoten pro Prozessor										
2	20.61	1.36	9.51	31.70	1.00	1.00	1.00	1.00	18.57	1.00	1.00
8	49.09	1.69	11.84	64.67	0.42	0.80	0.80	0.49	27.15	0.68	0.72
32	125.17	2.04	14.27	141.80	0.16	0.67	0.67	0.22	41.94	0.44	0.50
128	437.46	2.87	22.96	468.26	0.05	0.47	0.41	0.07	107.79	0.17	0.21
	65 536 Knoten pro Prozessor										
1	32.69	2.51	17.59	53.70	1.00	1.00	1.00	1.00	31.24	1.00	1.00
4	65.96	2.93	20.50	90.30	0.50	0.86	0.86	0.59	43.12	0.72	0.77
16	127.44	3.31	23.14	155.10	0.26	0.76	0.76	0.35	56.71	0.55	0.61
64	391.63	5.43	43.42	440.20	0.08	0.46	0.41	0.12	107.68	0.29	0.32
	16 384 Knoten pro Prozessor										
1	6.36	0.57	3.98	14.20	1.00	1.00	1.00	1.00	5.71	1.00	1.00
4	17.78	0.81	5.66	25.40	0.36	0.70	0.70	0.56	12.08	0.47	0.55
16	41.01	1.07	7.47	51.35	0.16	0.53	0.53	0.28	19.33	0.30	0.36
64	121.61	1.93	15.45	143.44	0.05	0.29	0.26	0.10	37.79	0.15	0.18

Tabelle 6.25: Laplace Operator ($\varepsilon = 1$); Scaleup für volle bis achtel Last für die Streifenaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	71.73	5.72	5.72	78.47	1.00	1.00	1.00	1.00	59.95	1.00	1.00
8	264.40	7.21	14.43	280.06	0.27	0.79	0.40	0.28	119.91	0.50	0.49
32	1351.52	12.25	36.75	1391.80	0.05	0.47	0.16	0.06	536.04	0.11	0.11
	32 768 Knoten pro Prozessor										
2	18.02	1.45	1.45	19.98	1.00	1.00	1.00	1.00	15.51	1.00	1.00
8	74.85	2.03	2.03	77.33	0.24	0.71	0.71	0.26	35.49	0.44	0.45
32	345.55	4.11	8.22	355.56	0.05	0.35	0.18	0.06	161.95	0.10	0.10
128	3155.09	18.51	55.53	3217.24	0.01	0.08	0.03	0.01	1375.38	0.01	0.01
	65 536 Knoten pro Prozessor										
1	22.96	2.54	2.54	26.08	1.00	1.00	1.00	1.00	21.93	1.00	1.00
4	73.36	3.24	3.24	77.32	0.31	0.79	0.79	0.34	40.87	0.54	0.55
16	279.10	4.87	9.74	289.92	0.08	0.52	0.26	0.09	121.97	0.18	0.19
64	2012.48	11.93	35.80	2051.31	0.01	0.21	0.07	0.01	893.15	0.02	0.03
	16 384 Knoten pro Prozessor										
1	4.41	0.55	0.55	5.48	1.00	1.00	1.00	1.00	4.19	1.00	1.00
4	17.81	0.87	0.87	19.25	0.25	0.63	0.63	0.28	11.55	0.36	0.38
16	80.73	1.53	1.53	82.74	0.05	0.36	0.36	0.07	38.95	0.11	0.12
64	537.35	4.85	9.71	548.59	0.01	0.11	0.06	0.01	280.02	0.01	0.02

Tabelle 6.26: Anisotroper Laplace Operator ($\varepsilon = 10^{+6}$); Scaleup für volle bis achtel Last für die Streifenaufteilung

Gitteraufbauphase E^*_{FA} leidet recht stark unter der großen Prozessorinterfacelänge durch die Streifenaufteilung. Da für große Prozessorzahlen die Gebietsstreifen sehr schmal werden, steigt der Anteil an Knoten, die durch die Randbehandlung teilsequentialisiert bearbeitet werden müssen.

Verläuft die starke Kopplung senkrecht zur Streifenrichtung ($\varepsilon \gg 1$), vergrößert sich, wie bereits festgestellt, der Anteil an Borderknoten enorm. Dadurch reduzieren sich alle Effizienzen sehr stark. Besonders ausgeprägt ist die Verschlechterung der Gitteraufbauzeit. Wie bei allen Streifenaufteilungen sind die teilsequentialisiert zu bearbeitenden Knotenmengen sehr groß. Hier kommt noch erschwerend hinzu, daß durch die große Reichweite des Überlapps von jedem Prozessor aus sehr viele Nachbarprozessoren erreicht werden. Daher benötigt man mehr Farben bei der Graphfärbung, die wiederum mehr Einzelschritte bei der Randbearbeitung bedingen (siehe Abbildung 6.25). Der nicht voll parallelisierbare Zeitanteil steigt hier also besonders stark an. Der Einsatz dieses Verfahrens für solche Konstellationen aus Gebietsaufteilung und Richtung der starken Kopplung muß strikt vermieden werden. Die annähernd isotropen Fälle ($\varepsilon \approx 1$) ordnen sich entsprechend ihrer Zwischenstellung monoton zwischen den Extremfällen ein.

6.5.2 Einfluß der Graphfärbungsmethode

In Abschnitt 5.6 wurden drei Methoden (cm1, cm2, cm3) zur Graphfärbung entwickelt. Sie sollen hier am Beispiel der anisotropen Poissongleichung verglichen werden. cm3 ist die teuerste Methode, weil die Knotengewichte explizit kommuniziert werden müssen. cm2 vermeidet diese Kommunikation, kann dafür nur eine sehr einfache Heuristik für die Knotengewichte verwenden, die keine Eigenschaften der Knoten einbezieht. Dieses Vorgehen kann mehr Farben zur Folge haben. Schließlich ist cm1 die Formalisierung des sequentiellen Abarbeitens der einzelnen Prozessoren. Diese Methode benötigt keine Zeit für die Graphfärbung, läßt dafür aber auch keine Parallelisierung bei der Bearbeitung des Randbereichs zu. Interessant ist sie nur deshalb, weil durch das sequentielle Vorgehen die Möglichkeit besteht, daß strukturiertere Grobgitter mit besseren Konvergenzeigenschaften erzeugt werden. Ob das zutrifft und ob die schlechtere parallele Skalierbarkeit dadurch kompensiert werden kann, soll nun untersucht werden.

Abbildungen 6.23 und 6.24 geben einen Überblick. Alle 3 Methoden unterscheiden sich bezüglich der Konvergenzrate kaum. Selbst für die Streifenaufteilung und $\varepsilon \gg 1$ bringt cm1 keine weitere Verbesserung der ohnehin schon sehr guten Konvergenz. Die Hoffnung, durch ein strukturierteres Vorgehen noch bessere numerische Ergebnisse zu erhalten, hat sich nicht bewahrheitet.

Die Anzahl benötigter Farben ist in Abbildung 6.25 für die größten Rechnungen dargestellt. Für die ersten algebraischen Level wird höchstens 1 Farbe mehr, als minimal nötig wäre, erzeugt. Für gröbere Level weiß man a priori nichts über die zu

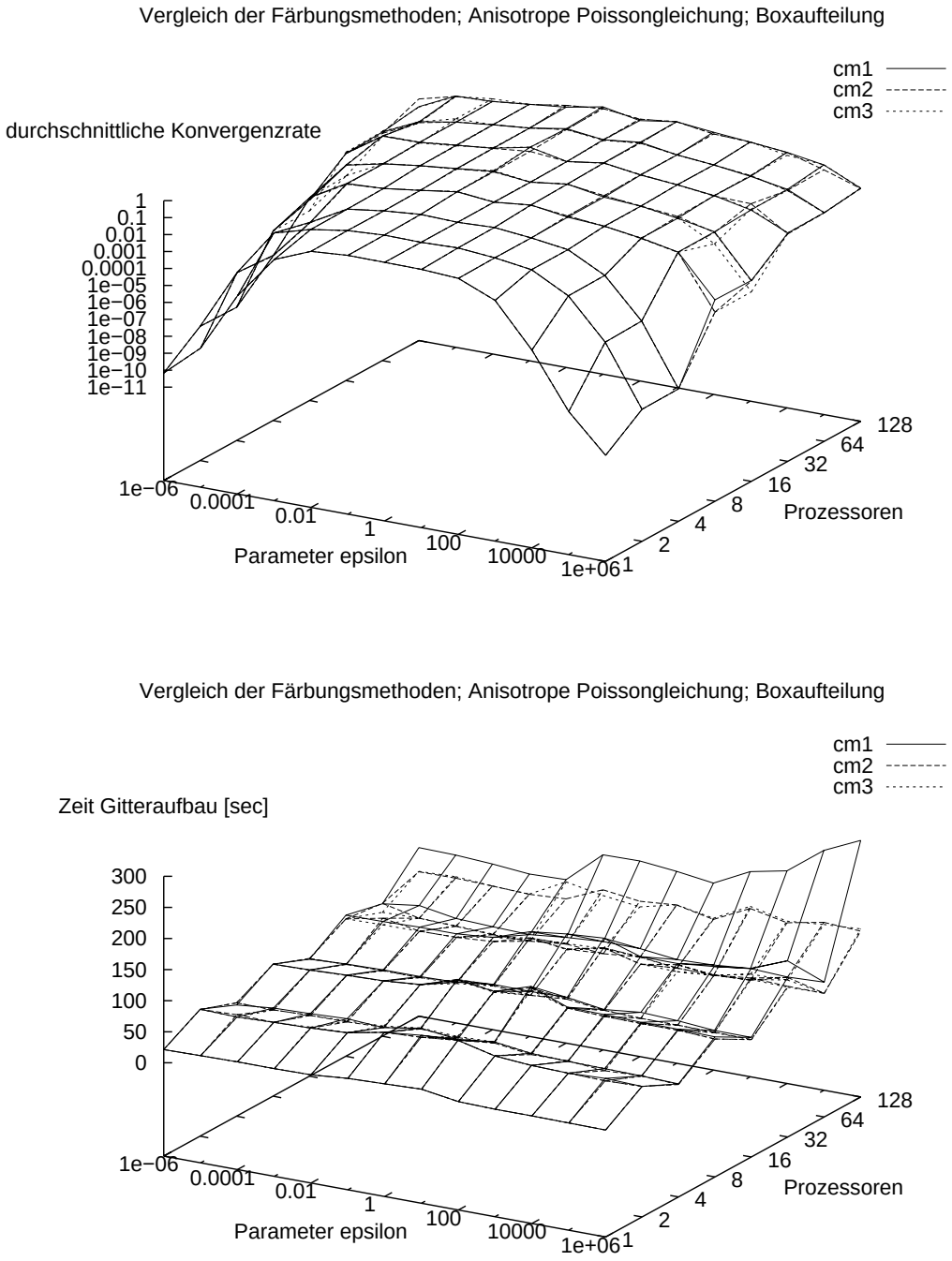
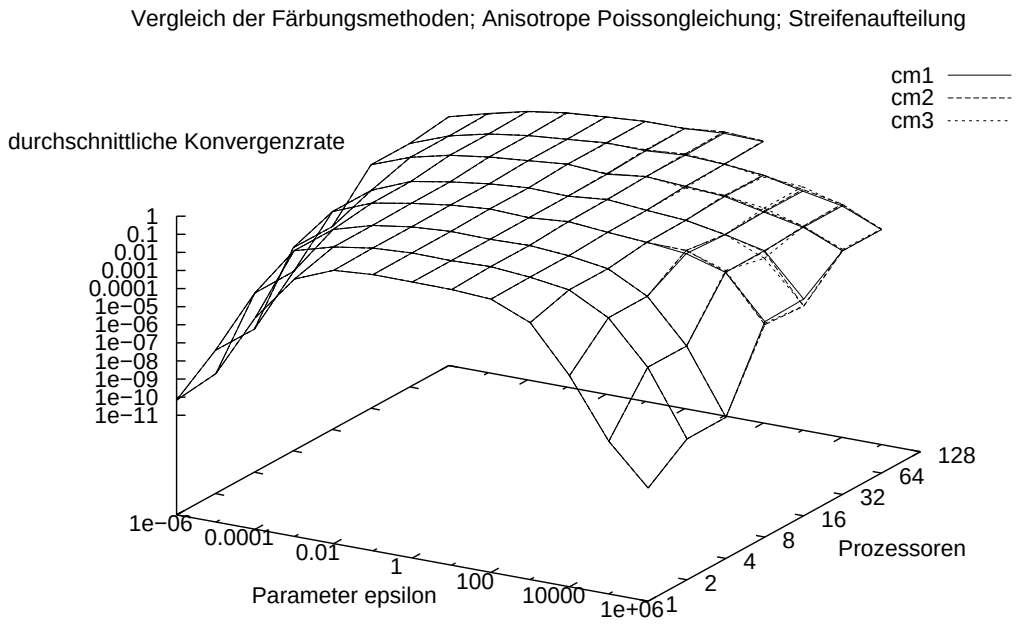


Abbildung 6.23: Anisotroper Laplace Operator; volle/halbe Last; Boxaufteilung; Abhängigkeit der durchschnittlichen Konvergenzrate und Gitteraufbauzeit T^*_{FA} von der Färbungsmethode



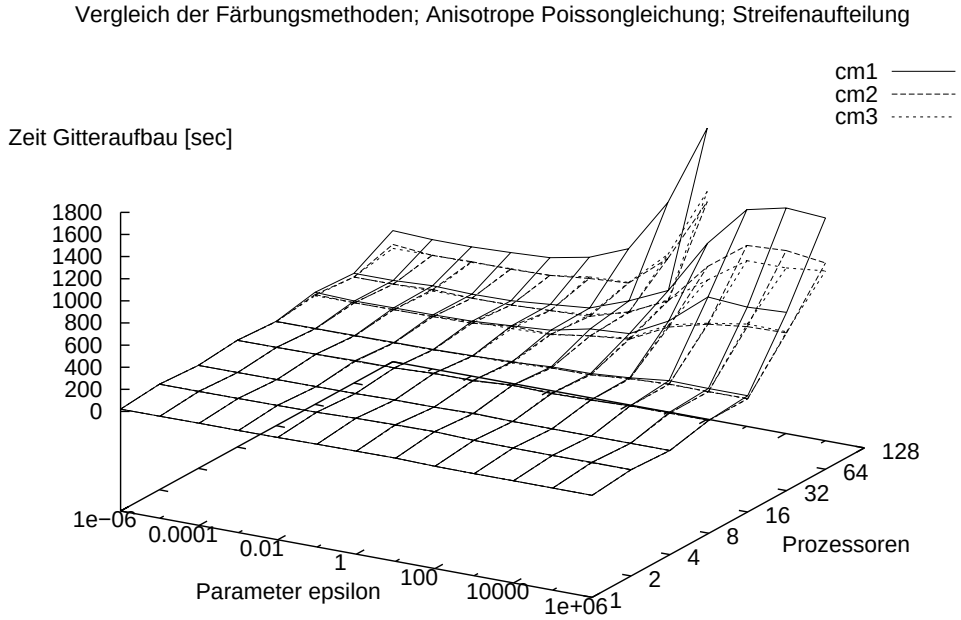


Abbildung 6.24: Anisotroper Laplace Operator; volle/halbe Last; Streifenaufteilung; Abhängigkeit der durchschnittlichen Konvergenzrate und Gitteraufbauzeit T^*_{FA} von der Färbungsmethode

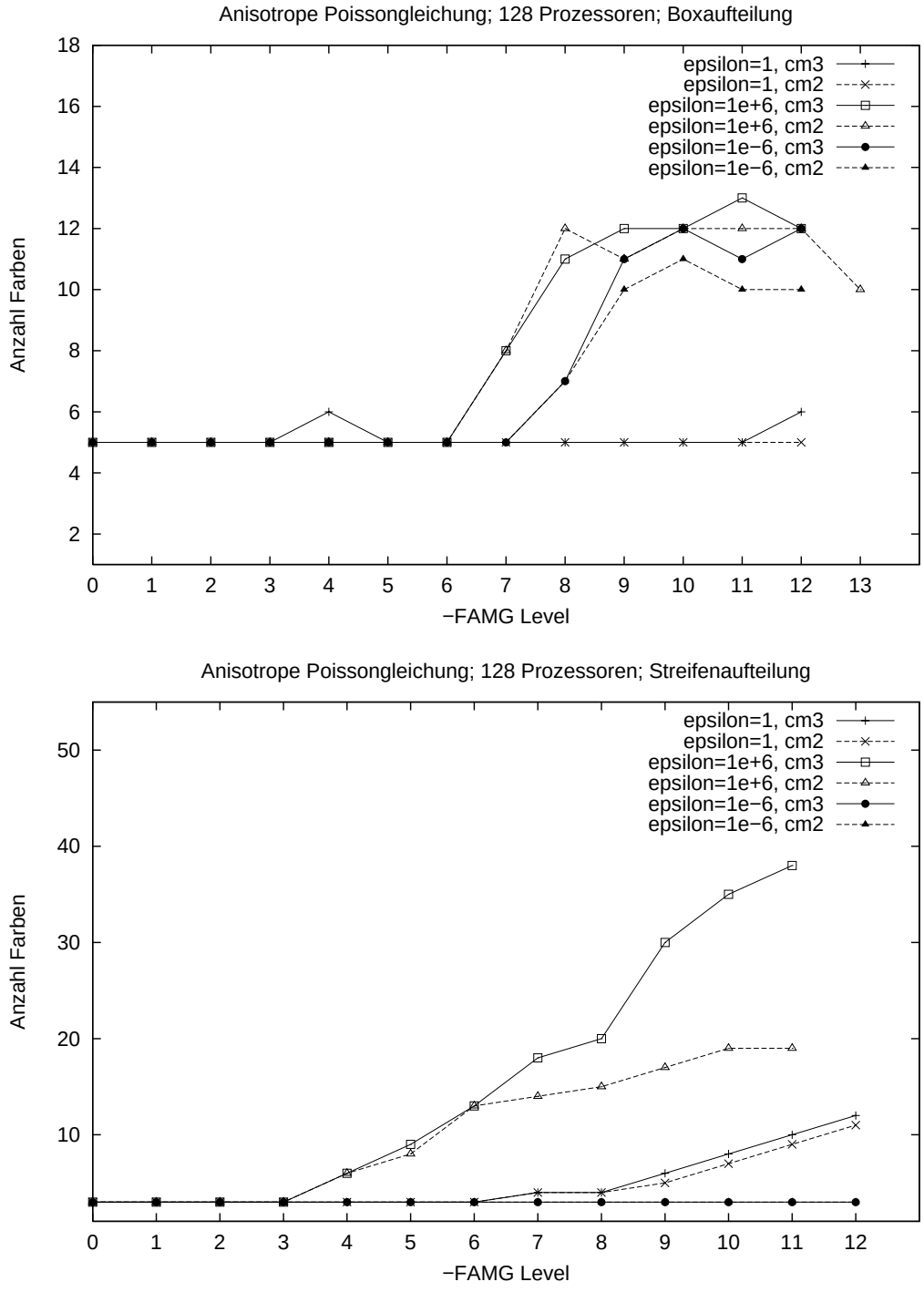


Abbildung 6.25: Anisotroper Laplace Operator; volle/halbe Last; Anzahl benötigter Farben in Abhängigkeit der Färbungsmethode

erwartende Farbenanzahl. Für die Boxaufteilung unterscheiden sich cm2 und cm3 nicht signifikant. Durch die größere Reichweite ist der Prozessornachbarschaftsgraph in Richtung der starken Kopplung wesentlich dichter und es werden erheblich mehr Farben benötigt (unteres Bild). In diesem Fall erzeugt cm3 teilweise deutlich mehr Farben als cm2, d. h. das aufwendigere Verfahren ist schlechter. In diesem Beispiel decken sich unsere Resultate nicht mit den Ergebnissen in [GJP96]. Die Wirkung auf die Färbungszeit und die Zeit für die Randmarkierung kann hier nicht dargestellt werden, weil diese Zeiten zu stark von weiteren Einflüssen und verschleppter Synchronisation (vergleiche Seite 116) geprägt werden.

Dagegen erkennt man die erwartete schlechtere parallele Skalierbarkeit des Gitteraufbaus bei cm1 sehr deutlich. Die Verlangsamung nimmt erwartungsgemäß, wie bei 128 Prozessoren besonders deutlich zu erkennen ist, für größer werdende Borderknotenzahlen (d. h. $\varepsilon \gg 1$) noch deutlich zu. Der Vergleich von cm2 und cm3 zeigt, daß beide Verfahren im wesentlichen die gleiche Zeit benötigen. Die Zeit für die zusätzliche Kommunikation beträgt weniger als 1 Sekunde und ist im Verhältnis zur gesamten Gitteraufbauzeit verschwindend gering. Sekundäre Auswirkungen der Färbungsmethoden sind teilweise entscheidender. Lediglich für die Streifenaufteilung, große Prozessorzahlen und $\varepsilon \gg 1$ ist cm3 merklich besser als die *beiden* anderen Methoden.

Da die Konvergenzrate im wesentlichen unabhängig von der Färbungsmethode ist, benötigt die Lösungsphase nur eine konstante Zusatzzeit, so daß die Gesamtzeit entsprechend der Gitteraufbauzeit skaliert und deshalb nicht eigens dargestellt ist.

Abbildung 6.22 zeigt die Interfacelängen im Vergleich. Wieder hebt sich nur cm1 etwas ab. Die Verschlechterung rührt von Störungen der Grobgittermatrizen am Gebietsrand her, die immer zu beobachten sind. Allerdings wird bei cm1 diese Störung immer von Prozessor 0 ins Gebietsinnere weitergetragen, wohingegen bei cm2 und cm3 die Prozessoren eher vom ungestörten Inneren zum Rand hin abgearbeitet werden und so kaum eine Verschleppung solcher Störungen auftritt. Deutlich erkennbar ist der Einfluß des Anisotropieparameters ε. Für quadratische Boxen ist die Interfacelänge vollkommen symmetrisch bezüglich $\varepsilon = 1$, für rechteckige Boxen nimmt die Länge bei starker Kopplung über die längere Boxseite stärker zu und für Streifenaufteilungen ist dieser Effekt noch stärker. Bei steigender Prozessorzahl steigt das Wachstum nochmals deutlich an.

Tabellen 6.27 bis 6.29 zeigen für cm1 quantitativ einige charakteristische Scaleup Effizienzen für Boxaufteilungen und Tabelle 6.30 den schlechtesten Fall für die Streifenaufteilung. Die Werte entsprechen nach den obigen Ausführungen vollständig den Erwartungen.

Zusammenfassend betrachtet gibt es keinen Grund, cm1 zu verwenden. cm2 und cm3 sind für die relevante Gebietsaufteilung (Boxaufteilung) praktisch gleichwertig. Wir bevorzugen daher cm3, weil durch die bessere Heuristik tendenziell weniger Farben benötigt werden und bei komplexeren Anwendungen dieser Vorteil überwiegen dürfte.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	58.23	5.45	5.45	64.55	1.00	1.00	1.00	1.00	53.94	1.00	1.00
8	105.29	5.81	11.61	117.87	0.55	0.94	0.47	0.55	62.27	0.87	0.80
32	171.18	6.21	18.65	190.80	0.34	0.88	0.29	0.34	76.79	0.70	0.62
128	251.81	6.51	19.52	272.50	0.23	0.84	0.28	0.24	121.14	0.45	0.42
	32 768 Knoten pro Prozessor										
2	15.09	1.36	1.36	16.93	1.00	1.00	1.00	1.00	13.87	1.00	1.00
8	30.59	1.65	1.65	32.70	0.49	0.83	0.83	0.52	18.04	0.77	0.77
32	56.17	1.73	3.47	60.11	0.27	0.79	0.39	0.28	26.11	0.53	0.52
128	95.60	1.87	5.61	101.70	0.16	0.73	0.24	0.17	48.73	0.28	0.28
	65 536 Knoten pro Prozessor										
1	22.38	2.53	2.53	25.53	1.00	1.00	1.00	1.00	21.35	1.00	1.00
4	42.87	2.98	2.98	46.52	0.52	0.85	0.85	0.55	28.68	0.74	0.75
16	92.51	4.93	9.86	102.92	0.24	0.51	0.26	0.25	38.03	0.56	0.50
64	135.67	3.54	10.61	146.94	0.16	0.71	0.24	0.17	62.79	0.34	0.33
	16 384 Knoten pro Prozessor										
1	4.38	0.55	0.55	5.39	1.00	1.00	1.00	1.00	4.16	1.00	1.00
4	11.20	0.77	0.77	12.49	0.39	0.71	0.71	0.43	7.54	0.55	0.57
16	24.82	0.95	0.95	26.18	0.18	0.58	0.58	0.21	13.56	0.31	0.32
64	48.57	1.09	2.18	51.10	0.09	0.50	0.25	0.11	23.85	0.17	0.18

Tabelle 6.27: Anisotroper Laplace Operator ($\varepsilon = 10^{-6}$); Scaleup für volle bis achtel Last für die Boxaufteilung und Färbungsmethode cm1

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	84.27	5.33	37.34	124.72	1.00	1.00	1.00	1.00	77.18	1.00	1.00
8	129.69	5.64	39.52	172.90	0.65	0.94	0.94	0.72	83.52	0.92	0.93
32	201.78	7.85	54.94	261.89	0.42	0.68	0.68	0.48	111.35	0.69	0.69
128	272.86	5.92	41.42	318.45	0.31	0.90	0.90	0.39	164.67	0.47	0.56
	32 768 Knoten pro Prozessor										
2	20.95	1.36	9.51	32.08	1.00	1.00	1.00	1.00	18.99	1.00	1.00
8	38.09	1.58	11.07	52.81	0.55	0.86	0.86	0.61	23.92	0.79	0.81
32	56.95	2.16	15.12	77.61	0.37	0.63	0.63	0.41	31.01	0.61	0.62
128	95.86	1.82	12.75	112.85	0.22	0.75	0.75	0.28	54.28	0.35	0.43
	65 536 Knoten pro Prozessor										
1	32.69	2.51	17.59	53.70	1.00	1.00	1.00	1.00	31.24	1.00	1.00
4	51.73	2.86	19.98	76.93	0.63	0.88	0.88	0.70	38.63	0.81	0.83
16	88.22	3.42	23.95	116.12	0.37	0.73	0.73	0.46	51.10	0.61	0.65
64	115.67	3.13	25.06	145.71	0.28	0.80	0.70	0.37	67.02	0.47	0.53
	16 384 Knoten pro Prozessor										
1	6.36	0.57	3.98	14.20	1.00	1.00	1.00	1.00	5.71	1.00	1.00
4	13.83	0.78	5.49	22.39	0.46	0.72	0.72	0.63	10.29	0.55	0.61
16	25.41	0.91	6.34	34.70	0.25	0.63	0.63	0.41	14.91	0.38	0.46
64	39.41	0.98	6.85	51.02	0.16	0.58	0.58	0.28	21.80	0.26	0.34

Tabelle 6.28: Laplace Operator ($\varepsilon = 1$); Scaleup für volle bis achtel Last für die Boxaufteilung und Färbungsmethode cm1

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	71.62	5.79	5.79	78.43	1.00	1.00	1.00	1.00	59.85	1.00	1.00
8	156.27	6.34	12.68	169.94	0.46	0.91	0.46	0.46	75.12	0.80	0.75
32	235.54	6.73	20.18	256.93	0.30	0.86	0.29	0.31	107.14	0.56	0.52
128	520.00	7.23	21.68	543.07	0.14	0.80	0.27	0.14	262.55	0.23	0.23
	32 768 Knoten pro Prozessor										
2	17.72	1.44	1.44	19.67	1.00	1.00	1.00	1.00	15.21	1.00	1.00
8	39.44	1.70	1.70	41.64	0.45	0.84	0.84	0.47	21.62	0.70	0.71
32	78.01	1.95	3.89	82.35	0.23	0.74	0.37	0.24	36.98	0.41	0.41
128	224.14	4.91	14.74	239.59	0.08	0.29	0.10	0.08	123.99	0.12	0.12
	65 536 Knoten pro Prozessor										
1	22.96	2.54	2.54	26.08	1.00	1.00	1.00	1.00	21.93	1.00	1.00
4	43.52	2.96	2.96	47.10	0.53	0.86	0.86	0.55	31.74	0.69	0.71
16	96.44	4.17	8.35	105.35	0.24	0.61	0.30	0.25	42.32	0.52	0.48
64	140.30	3.45	10.35	151.32	0.16	0.74	0.25	0.17	66.85	0.33	0.32
	16 384 Knoten pro Prozessor										
1	4.41	0.55	0.55	5.48	1.00	1.00	1.00	1.00	4.19	1.00	1.00
4	11.30	0.78	0.78	12.61	0.39	0.70	0.70	0.43	8.51	0.49	0.51
16	25.23	1.79	1.79	27.43	0.17	0.31	0.31	0.20	13.23	0.32	0.32
64	50.08	1.07	2.14	52.55	0.09	0.51	0.26	0.10	25.57	0.16	0.17

Tabelle 6.29: Anisotroper Laplace Operator ($\varepsilon = 10^{+6}$); Scaleup für volle bis achtel Last für die Boxaufteilung und Färbungsmethode cm1

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	71.62	5.79	5.79	78.43	1.00	1.00	1.00	1.00	59.85	1.00	1.00
8	265.30	7.15	14.30	280.80	0.27	0.81	0.40	0.28	120.54	0.50	0.49
32	1431.45	12.25	36.74	1470.83	0.05	0.47	0.16	0.05	710.71	0.08	0.09
	32 768 Knoten pro Prozessor										
2	17.72	1.44	1.44	19.67	1.00	1.00	1.00	1.00	15.21	1.00	1.00
8	76.04	2.02	2.02	78.50	0.23	0.71	0.71	0.25	36.72	0.41	0.43
32	475.04	4.30	8.61	485.59	0.04	0.33	0.17	0.04	265.84	0.06	0.06
128	4339.74	12.86	38.59	4383.69	0.00	0.11	0.04	0.00	2645.60	0.01	0.01
	65 536 Knoten pro Prozessor										
1	22.96	2.54	2.54	26.08	1.00	1.00	1.00	1.00	21.93	1.00	1.00
4	72.45	3.25	3.25	76.47	0.32	0.78	0.78	0.34	41.34	0.53	0.55
16	330.29	5.23	10.46	341.83	0.07	0.49	0.24	0.08	150.19	0.15	0.15
64	2503.82	12.08	36.23	2543.22	0.01	0.21	0.07	0.01	1371.22	0.02	0.02
	16 384 Knoten pro Prozessor										
1	4.41	0.55	0.55	5.48	1.00	1.00	1.00	1.00	4.19	1.00	1.00
4	17.70	0.86	0.86	19.10	0.25	0.64	0.64	0.29	11.71	0.36	0.38
16	92.39	1.61	1.61	94.43	0.05	0.34	0.34	0.06	50.10	0.08	0.09
64	645.92	6.21	12.41	660.08	0.01	0.09	0.04	0.01	420.80	0.01	0.01

Tabelle 6.30: Anisotroper Laplace Operator ($\varepsilon = 10^{+6}$); Scaleup für volle bis achtel Last für die Streifenaufteilung und Färbungsmethode cm1

6.6 Konvektions–Diffusions–Gleichung

In diesem Abschnitt wird die Konvektions–Diffusions–Gleichung (2.4)

$$\begin{aligned} -\varepsilon\Delta u + v\cdot\nabla u &= f && u \text{ auf } \Omega = (0,1)^2, \varepsilon \in \mathbb{R}_0^+ \\ u(x,y) &= \begin{cases} 1 & \text{für } y < 1/2 \\ 0 & \text{sonst} \end{cases} && \text{für } (x,y) \in \partial\Omega \end{aligned} \tag{6.11}$$

zu einem gegebenen Geschwindigkeitsfeld $v \in [\mathcal{C}^1(\mathbb{R}^2)]^2$ genauer untersucht. Das Gebiet wird mit einem strukturierten Viereckgitter vernetzt. Die Diskretisierung erfolgt mit einem baryzentrischen *Finiten Volumen Verfahren*; der Konvektionsterm wird mittels *full–upwinding* stabilisiert. Details sind z. B. in [Joh00b] ausgeführt.

Die dimensionslose Kennzahl, die auf diskreter Ebene die Stärke von Konvektion zu Diffusion beschreibt, ist die sogenannte Gitterpecletzahl.

Definition 6.6.1 (Pecletzahl)
Die Zahl

$$PEC := \frac{\bar{h}\,\bar{v}}{\varepsilon}$$

heißt **Gitterpecletzahl**. Da wir keine weiteren Pecletzahlen betrachten werden, soll sie in dieser Arbeit nur **Pecletzahl** genannt werden. Dabei sind $\bar{h}$ und $\bar{v}$ geeignet gemittelte Werte für die Gitterweite h und die Konvektionsstärke $\|v\|$.

Bemerkung 6.6.2 (Wahl von $\bar{h}$ und $\bar{v}$)
Da wir mit einem strukturierten Gitter starten, ist die Wahl $\bar{h} := h$ als Gitterweite trivial. Uns interessiert das Verhalten des Lösungsverfahrens gegenüber der Pecletzahl für ein fest vorgegebenes Strömungsfeld v. Da die von uns betrachteten Strömungsfelder im Mittel ungefähr Norm 1 haben, können wir hier $\bar{v} := 1$ wählen und erhalten für unsere Betrachtungen

$$PEC := \frac{h}{\varepsilon}$$

für das Startgitter.

Diese Definition trägt der Erfahrung Rechnung, daß die Schwierigkeit der Gleichung sich üblicherweise aus dem Verhältnis von Konvektions– zu Diffusionsstärke und nicht aus dem absoluten Wert einer dieser Zahlen ergibt.

Die zu erwartenden Schwierigkeiten wurden bereits in Abschnitt 2.3.6.2 besprochen. Wir wollen in 2 Stufen vorgehen. Zuerst wird anhand einer horizontal verlaufenden Strömung der Einfluß der Parallelisierung auf den globalen Informationsfluß und damit auf die Konvergenzeigenschaften geprüft. Danach kommt bei der Kreisströmung das Problem geschlossener Charakteristiken hinzu. Das ist die schwierigste der hier betrachteten Gleichungen und stellt eine große Herausforderung für parallele, robuste Lösungsverfahren dar.

6.6.1 Horizontale Strömung

Als erstes wird eine horizontal verlaufende Strömung betrachtet. In (6.11) wird dazu

$$v(x,y) := \begin{pmatrix} 1 \\ 0 \end{pmatrix}^\top$$

gewählt. Dieses Modellproblem dient dazu, um die typische Eigenschaft der Konvektions–Diffusions–Gleichung zu studieren: Der Informationsfluß verläuft hauptsächlich entlang der Charakteristiken des reduzierten Problems, d. h. der Gleichung für $\varepsilon = 0$. Diese Linien heißen auch Stromlinien. Dabei wird für die Parallelisierung von besonderem Interesse sein, wie stark der Einfluß der Gebietsaufteilung ist. Wir werden ein Tensorproduktgitter für das Gebiet des Einheitsquadrats benutzen, um so auch extreme Konstellationen herbeiführen zu können: Bei einer horizontalen Streifenaufteilung liegt jede Stromlinie *vollständig* auf einem Prozessor — das ist für den Informationsfluß die optimale Situation. Genau dieser Fluß wird bei einer vertikalen Streifenaufteilung größtmöglich gestört — jede Stromlinie ist über *alle* Prozessoren verteilt. Da eine Streifenaufteilung aber sehr große Interfaces bedingt, die ihrerseits erhöhte Kommunikations– und Speicherkosten nach sich ziehen, soll auch eine Boxaufteilung erprobt werden, die diese Art der Kosten minimiert; der Einfluß auf die Konvergenzgüte muß allerdings genau beobachtet werden.

Zunächst soll eine Vorstellung der erzeugten Grobgittermatrizen gegeben werden. Für kleine Pecletzahlen unterscheiden sie sich kaum von denen des isotropen Laplace Operators in Abbildung 6.2. Daher werden in Abbildung 6.26 nur Bilder für eine starke Strömung (Pecletzahl 10^6) präsentiert. Eine horizontale Streifenaufteilung (2. Zeile) erzeugt fast die gleichen Grobgittermatrizen wie der serielle Vergleichsfall (1. Zeile). Auch Box– und vertikale Streifenaufteilung liefern ähnliche Ergebnisse, wobei überraschenderweise gerade die vertikale Streifenaufteilung die regelmäßigsten, verteilten Graphen erzeugt. Die Ursache dafür ist, daß die Auswahl zunächst innerhalb der Prozessoren mit der ersten Farbe erfolgt. Da die Situation für alle Knoten auf einer vertikalen Linie abgesehen von Randeffekten identisch ist und diese Knoten aber fast entkoppelt sind, so daß sie sich nicht gegenseitig beeinflussen, werden alle Knoten auf einer solchen Linie die gleiche Markierung erhalten. Das Muster kann sich dann schrittweise von Prozessor zu Prozessor fortsetzen. Für eine gute Konvergenz ist eine solche vollkommene Regelmäßigkeit jedoch nicht notwendig, wie der serielle Fall zeigt, und muß daher nicht angestrebt werden. Diese kleinen Beispiele geben einen ersten Eindruck. Nun sollen Ergebnisse mit größeren Knotenanzahlen diskutiert werden.

Die Abbildungen 6.27 bis 6.29 zeigen, daß die mittlere Konvergenzrate fast unabhängig von der Prozessorzahl und auch der Gebietsaufteilung ist. Das ist nach den obigen Überlegungen sehr bemerkenswert. Auch gegenüber der Pecletzahl ist das Verhalten vollkommen robust.

Die Abhängigkeit der Konvergenzgüte von der Gitterweite ist in den Tabellen 6.31 und 6.32 für 2 typische Pecletzahlen bezüglich der Prozessoranzahl und der Gebiets-

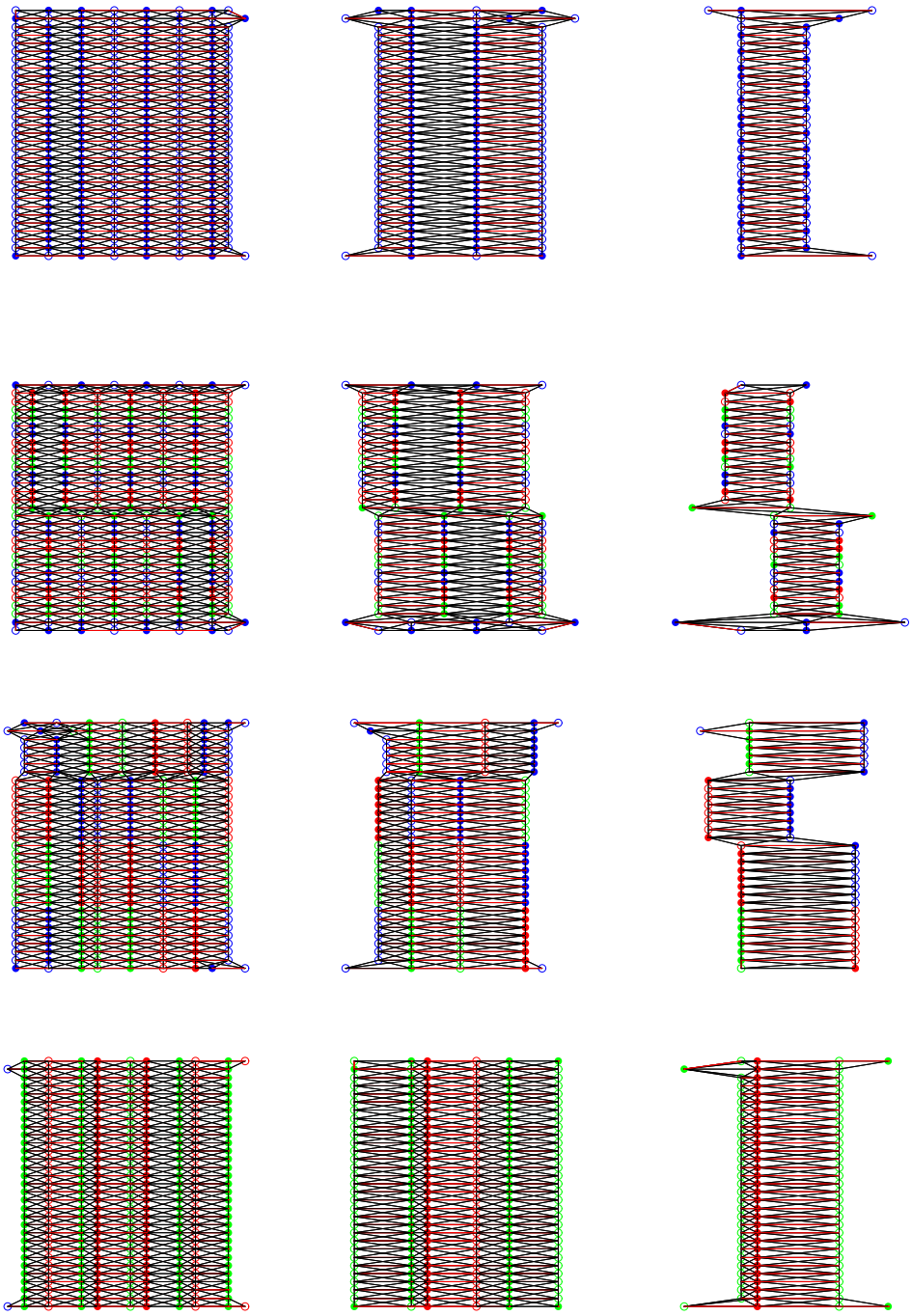


Abbildung 6.26: Horizontale Strömung; $PEC = 10^6$; Startgitterweite 1/32; Matrixgraphen der Level $-2, -3, -4$ (v. l. n. r.); für 1×1, 1×16, 4×4, 16×1 Prozessoren (v. o. n. u.)

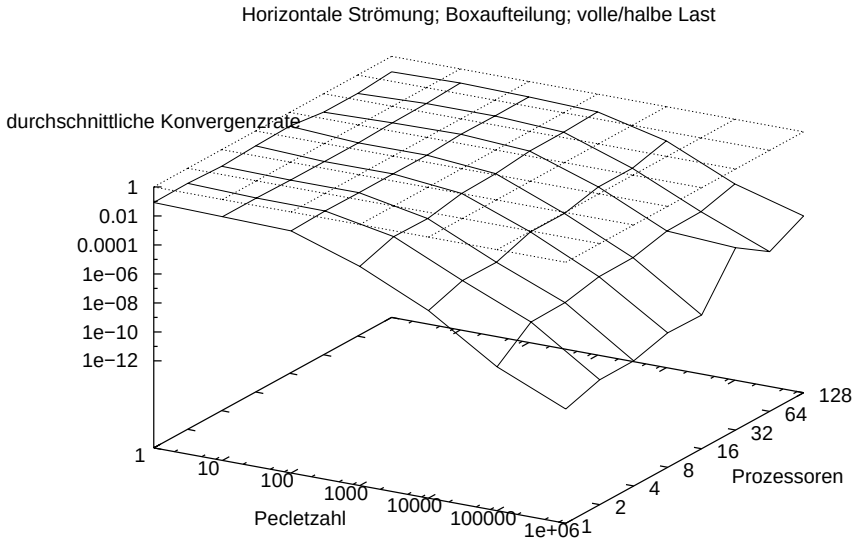


Abbildung 6.27: Horizontale Strömung; Boxaufteilung; volle bzw. halbe Last; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl und der Prozessorzahl

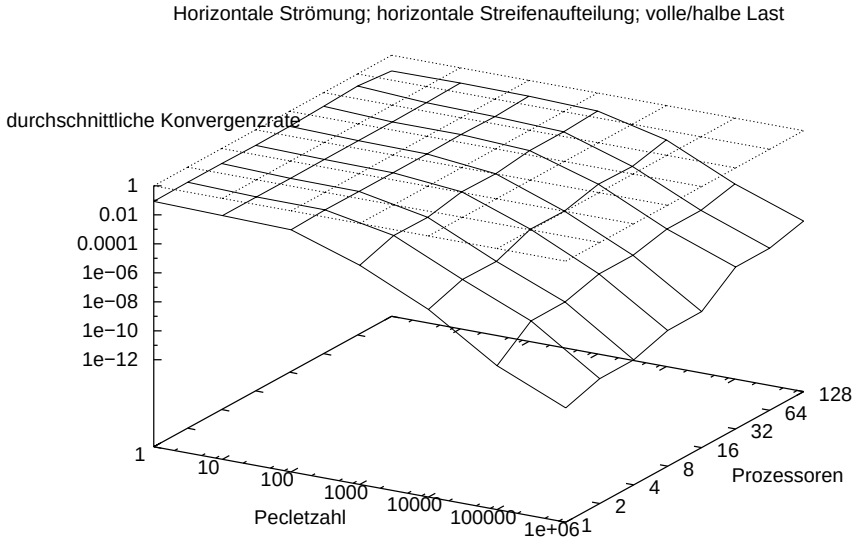


Abbildung 6.28: Horizontale Strömung; horizontale Streifenaufteilung; volle bzw. halbe Last; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl und der Prozessorzahl

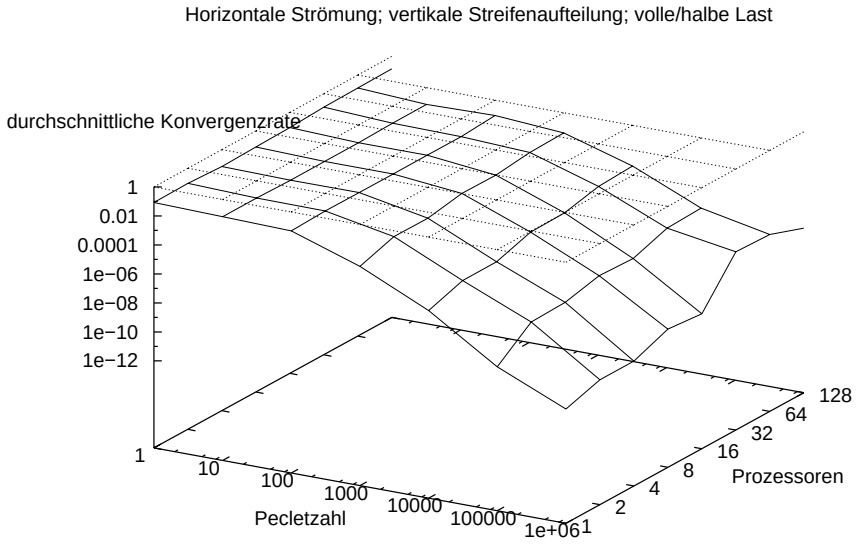


Abbildung 6.29: Horizontale Strömung; vertikale Streifenaufteilung; volle bzw. halbe Last; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl und der Prozessorzahl

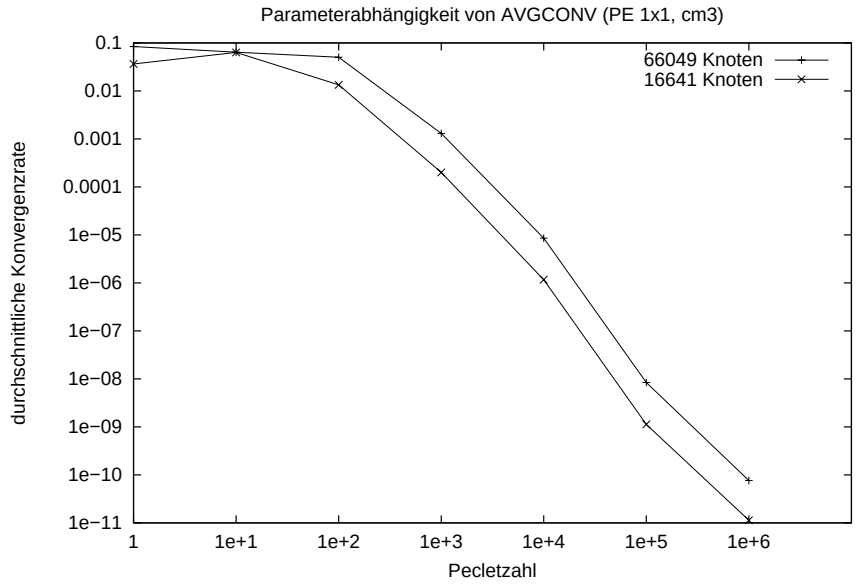


Abbildung 6.30: Horizontale Strömung; seriell; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl; serielle Rechnung; verschiedene Knotenzahlen

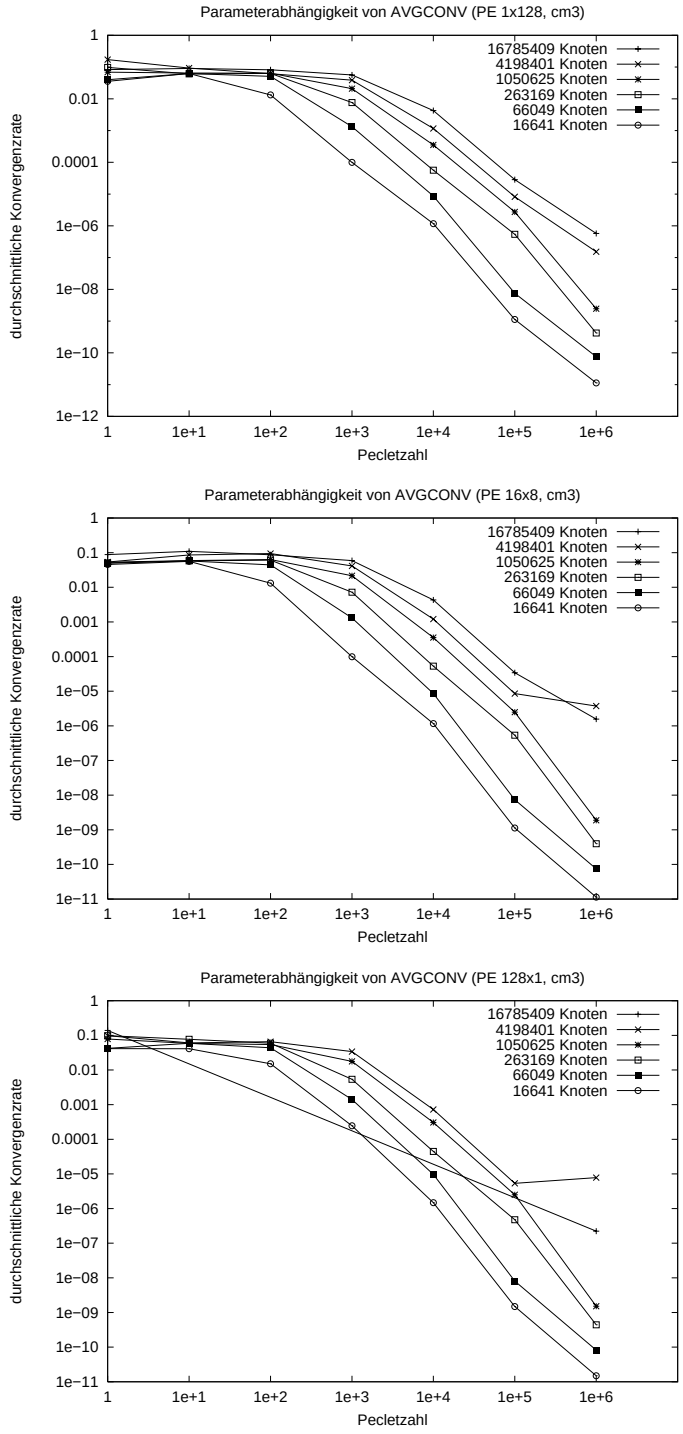


Abbildung 6.31: Horizontale Strömung; parallel; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl; verschiedene Knotenzahlen zu den Prozessoraufteilungen 1×128, 16×8 und 128×1

PE	h 1/128 N 16 641	1/256 66 049	1/512 263 169	1/1024 1 050 625	1/2048 4 198 401	1/4096 16 785 409
1×1	6 0.036	8 0.084				
2×1	7 0.052	7 0.054	7 0.068			
2×2	7 0.050	7 0.056	7 0.070			
4×2	7 0.050	7 0.056	7 0.063	8 0.093		
4×4	7 0.049	7 0.051	7 0.059	9 0.111		
8×4	6 0.046	7 0.050	7 0.051	7 0.066	7 0.053	
8×8	6 0.046	7 0.049	7 0.051	7 0.064	7 0.054	
16×8	6 0.046	7 0.050	7 0.051	7 0.053	7 0.054	8 0.088
1×1	6 0.036	8 0.084				
1×2	6 0.037	6 0.040	8 0.091			
1×4	6 0.037	6 0.040	8 0.092			
1×8	6 0.037	6 0.041	8 0.085	8 0.085		
1×16	6 0.037	6 0.040	8 0.091	8 0.091		
1×32	6 0.037	9 0.114	7 0.068	8 0.089	9 0.113	
1×64	6 0.038	6 0.045	7 0.066	9 0.126	10 0.146	
1×128	6 0.036	6 0.040	8 0.098	7 0.069	11 0.171	8 0.083
1×1	6 0.036	8 0.084				
2×1	7 0.052	7 0.054	7 0.068			
4×1	7 0.052	7 0.054	7 0.068			
8×1	7 0.053	7 0.054	7 0.061	8 0.079		
16×1	6 0.041	7 0.051	8 0.088	8 0.089		
32×1	7 0.051	7 0.049	8 0.097	8 0.087	9 0.118	
64×1	6 0.036	7 0.057	9 0.122	8 0.079	9 0.128	
128×1	6 0.041	6 0.042	8 0.098	8 0.078	8 0.098	10 0.139

Tabelle 6.31: Horizontale Strömung, $PEC = 1$; Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessoraufteilung (Box- und beide Streifenaufteilungen)

aufteilung und in den Abbildungen 6.30 bis 6.31 für charakteristische Gebietsaufteilungen bezüglich der Pecletzahl dargestellt. Für kleine Pecletzahlen verschlechtert sich die Konvergenzrate kaum mit wachsender Gitterfeinheit, für große Pecletzahlen wird das Verfahren asymptotisch zu einem exakten Löser. Bei sehr kleiner Startgitterweite (nur für große Prozessoranzahl möglich) werden die gröbsten Gitter — wie bei der Poissongleichung auf Seite 135 diskutiert — annähernd isotrop, weil durch die Halbvergröberung die anfängliche Anisotropie ausgeglichen wird. Die Konvergenzraten schwächen sich dadurch leicht ab.

Das Verfahren ist also robust gegen die Pecletzahl, Knotenanzahl, Prozessoranzahl und die Gebietsaufteilung.

Der Speicheraufwand ist für 2 charakteristische Pecletzahlen in den Tabellen 6.33 und 6.34 dargestellt. Für alle Fälle ist die Gitterkomplexität c_G ziemlich konstant 2, d. h. die Vergröberung verläuft in allen Fällen sehr gut. Das Verhältnis Border– zu Masterknoten B/M ist für Streifenaufteilungen erwartungsgemäß um ein Vielfaches höher als für Boxaufteilungen; das ist eine direkte Folge der erheblich längeren Prozessor-

PE	h / N	1/128 16 641	1/256 66 049	1/512 263 169	1/1024 1 050 625	1/2048 4 198 401	1/4096 16 785 409
2×1		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$			
2×2		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$			
4×2		1 $1.1\cdot10^{-11}$	1 $7.4\cdot10^{-11}$	1 $4.1\cdot10^{-10}$	1 $2.2\cdot10^{-9}$		
4×4		1 $1.1\cdot10^{-11}$	1 $7.4\cdot10^{-11}$	1 $4.2\cdot10^{-10}$	1 $2.2\cdot10^{-9}$		
8×4		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.1\cdot10^{-10}$	1 $2.0\cdot10^{-9}$	1 $7.5\cdot10^{-9}$	
8×8		1 $1.1\cdot10^{-11}$	1 $7.4\cdot10^{-11}$	1 $3.8\cdot10^{-10}$	1 $1.6\cdot10^{-9}$	2 $3.5\cdot10^{-6}$	
16×8		1 $1.1\cdot10^{-11}$	1 $7.4\cdot10^{-11}$	1 $4.0\cdot10^{-10}$	1 $2.1\cdot10^{-9}$	2 $4.8\cdot10^{-6}$	2 $1.0\cdot10^{-6}$
1×2		1 $1.1\cdot10^{-11}$	1 $7.2\cdot10^{-11}$	1 $4.1\cdot10^{-10}$			
1×4		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$			
1×8		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$	1 $2.3\cdot10^{-9}$		
1×16		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$	1 $2.4\cdot10^{-9}$		
1×32		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$	1 $2.4\cdot10^{-9}$	2 $1.5\cdot10^{-7}$	
1×64		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$	1 $2.4\cdot10^{-9}$	2 $1.5\cdot10^{-7}$	
1×128		1 $1.1\cdot10^{-11}$	1 $7.6\cdot10^{-11}$	1 $4.2\cdot10^{-10}$	1 $2.5\cdot10^{-9}$	2 $1.5\cdot10^{-7}$	2 $5.8\cdot10^{-7}$
2×1		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $4.2\cdot10^{-10}$			
4×1		1 $1.1\cdot10^{-11}$	1 $7.3\cdot10^{-11}$	1 $4.1\cdot10^{-10}$			
8×1		1 $1.1\cdot10^{-11}$	1 $7.3\cdot10^{-11}$	1 $4.0\cdot10^{-10}$	1 $6.1\cdot10^{-9}$		
16×1		1 $1.1\cdot10^{-11}$	1 $7.5\cdot10^{-11}$	1 $6.4\cdot10^{-10}$	1 $2.7\cdot10^{-9}$		
32×1		1 $1.1\cdot10^{-11}$	1 $1.2\cdot10^{-10}$	1 $6.9\cdot10^{-10}$	1 $3.3\cdot10^{-9}$	2 $8.3\cdot10^{-6}$	
64×1		1 $1.9\cdot10^{-11}$	1 $7.4\cdot10^{-11}$	1 $5.7\cdot10^{-10}$	1 $2.3\cdot10^{-9}$	2 $2.8\cdot10^{-6}$	
128×1		1 $1.1\cdot10^{-11}$	1 $1.2\cdot10^{-10}$	1 $4.9\cdot10^{-10}$	1 $2.6\cdot10^{-9}$	2 $1.3\cdot10^{-6}$	2 $2.1\cdot10^{-7}$

Tabelle 6.32: Horizontale Strömung, $PEC = 10^6$; Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessoraufteilung (Box- und beide Streifenaufteilungen)

interfaces. Dieser Einfluß verstärkt sich mit abnehmender Prozessorauslastung. Das Wachstum des Überlapps auf den gröberen Leveln $c_{\ddot{U}}$ nimmt jedoch von der horizontalen Streifenaufteilung über die Boxaufteilung bis zur vertikalen Streifenaufteilung monoton zu.

Die Ursache dafür ist in den Bildfolgen in den Abbildungen 6.32 und 6.33 zu erkennen. Gezeigt ist für verschiedene Gebietsaufteilungen und 2 charakteristische Pecletzahlen der Matrixgraph einzelner Prozessoren auf verschiedenen Leveln. Schwarz gezeichnete Knoten liegen in der Kernpartition, graue im Überlapp; grobmarkierte Knoten sind als gefüllte Kreisscheiben, feinmarkierte als Kreislinien dargestellt. Am deutlichsten tritt der Effekt für große Pecletzahlen in Erscheinung (Abbildung 6.33). Für alle Gebietsaufteilungen sieht man klar die Überlapptiefe 2. Am dichtesten besetzt ist der Überlapp für die vertikale Streifenaufteilung (im Bild rechte Spalte), da der Überlapp aus kompletten Linien des Anfangsgitters besteht. Für die horizontale Streifenaufteilung (linke Spalte im Bild) entstehen gleich viele Linien im Überlapp, die aber auf Grund der Halbvergröberung stark ausgelichtet sind. Gemessen in Anzahl Knoten wird der Überlapp somit von Level zu Level kleiner, woraus die günstige Überlappkomplexität $c_{\ddot{U}}$ resultiert. Für eine Boxaufteilung mischen sich beide Effekte (mittlere Bildspalte) und die Überlappkomplexität $c_{\ddot{U}}$ liegt damit zwischen den

horizontale Streifenaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.14	1.95	2.69	0.01
8	2.24	1.98	2.94	0.04
32	2.30	2.00	2.98	0.09
128	2.26	1.96	2.76	0.17
	32 768 Knoten/PE			
2	2.13	1.92	2.47	0.02
8	2.19	1.96	2.71	0.08
32	2.36	2.01	2.95	0.18
128	2.40	1.98	2.81	0.35

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.11	1.93	0.00	0.00
4	2.17	1.96	2.69	0.03
16	2.29	1.99	2.95	0.09
64	2.40	2.02	3.03	0.18
	16 384 Knoten/PE			
1	1.85	1.74	0.00	0.00
4	2.13	1.92	2.42	0.06
16	2.26	1.98	2.70	0.16
64	2.47	2.00	2.94	0.36

Boxaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.19	1.99	4.79	0.02
8	2.22	1.99	4.93	0.04
32	2.15	1.97	4.78	0.05
128	2.11	1.96	4.75	0.05
	32 768 Knoten/PE			
2	2.11	1.94	3.63	0.03
8	2.23	1.99	4.44	0.07
32	2.19	1.97	4.50	0.09
128	2.16	1.96	4.59	0.10

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.11	1.93	0.00	0.00
4	2.19	1.98	4.01	0.03
16	2.16	1.97	4.25	0.05
64	2.11	1.96	4.21	0.06
	16 384 Knoten/PE			
1	1.85	1.74	0.00	0.00
4	2.12	1.93	3.26	0.05
16	2.17	1.97	3.87	0.09
64	2.16	1.96	4.05	0.11

vertikale Streifenaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.19	1.99	4.79	0.02
8	2.28	2.00	5.27	0.07
32	2.50	2.01	5.69	0.17
128	2.79	2.01	5.79	0.36
	32 768 Knoten/PE			
2	2.11	1.94	3.63	0.03
8	2.30	1.99	4.52	0.12
32	2.65	2.01	5.29	0.32
128	3.15	2.00	5.47	0.68

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.11	1.93	0.00	0.00
4	2.23	1.99	4.78	0.06
16	2.45	2.01	5.48	0.16
64	2.76	2.01	5.67	0.35
	16 384 Knoten/PE			
1	1.85	1.74	0.00	0.00
4	2.16	1.94	3.80	0.09
16	2.55	2.01	4.91	0.29
64	3.03	2.06	5.10	0.61

Tabelle 6.33: Horizontale Strömung, $PEC = 1$; verschiedene Gebietsaufteilungen; Speicherkomplexität für volle bis achtel Last (Scaleup)

horizontale Streifenaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	1.97	1.98	1.98	0.01
8	1.99	1.99	2.00	0.03
32	2.00	2.00	2.00	0.06
128	2.00	2.00	2.00	0.12
	32 768 Knoten/PE			
2	1.92	1.93	1.93	0.02
8	1.97	1.98	1.98	0.05
32	1.99	1.99	2.00	0.12
128	2.00	2.00	2.00	0.25

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	1.92	1.93	0.00	0.00
4	1.97	1.98	1.98	0.02
16	1.99	1.99	2.00	0.06
64	2.00	2.00	2.00	0.12
	16 384 Knoten/PE			
1	1.73	1.73	0.00	0.00
4	1.92	1.93	1.93	0.05
16	1.97	1.98	1.98	0.12
64	1.99	1.99	2.00	0.25

Boxaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.00	1.98	6.50	0.03
8	2.05	1.99	6.84	0.05
32	2.09	2.00	8.04	0.08
128	2.10	2.00	8.23	0.09
	32 768 Knoten/PE			
2	1.95	1.93	4.49	0.04
8	2.06	1.98	5.38	0.09
32	2.15	2.00	7.01	0.14
128	2.17	2.00	7.31	0.16

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	1.92	1.93	0.00	0.00
4	2.00	1.98	4.25	0.03
16	2.05	2.00	5.26	0.06
64	2.09	2.00	6.28	0.09
	16 384 Knoten/PE			
1	1.73	1.73	0.00	0.00
4	1.95	1.93	3.23	0.05
16	2.06	1.98	4.28	0.10
64	2.13	2.00	5.24	0.15

vertikale Streifenaufteilung

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.00	1.98	6.50	0.03
8	2.15	2.00	9.08	0.12
32	2.33	2.01	9.98	0.30
128	2.57	2.02	9.20	0.57
	32 768 Knoten/PE			
2	1.95	1.93	4.49	0.04
8	2.17	1.99	6.48	0.18
32	2.52	2.01	8.50	0.51
128	2.93	2.04	8.72	1.06

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	1.92	1.93	0.00	0.00
4	2.06	1.99	6.48	0.08
16	2.29	2.00	8.87	0.26
64	2.54	2.01	8.75	0.53
	16 384 Knoten/PE			
1	1.73	1.73	0.00	0.00
4	2.02	1.94	4.48	0.11
16	2.43	2.00	7.33	0.43
64	2.81	2.02	7.54	0.92

Tabelle 6.34: Horizontale Strömung, $PEC = 10^6$; verschiedene Gebietsaufteilungen; Speicherkomplexität für volle bis achtel Last (Scaleup)

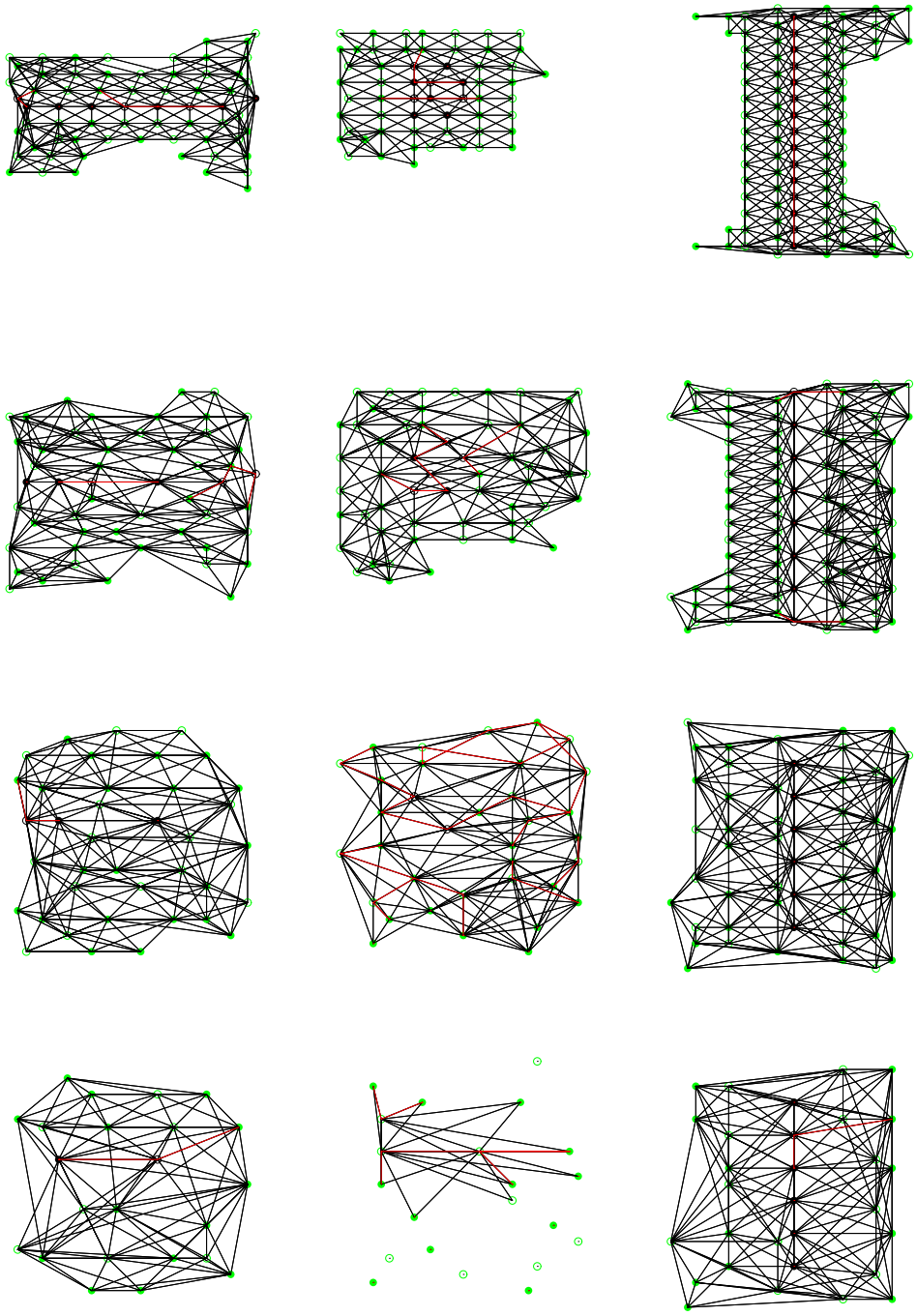


Abbildung 6.32: Matrixgraphen der Level -3 bis -6 (v. o. n. u.); horizontale Strömung; $PEC = 1$; links 1×16, PE 9; mitte 4×4, PE 9; rechts 16×1, PE 7

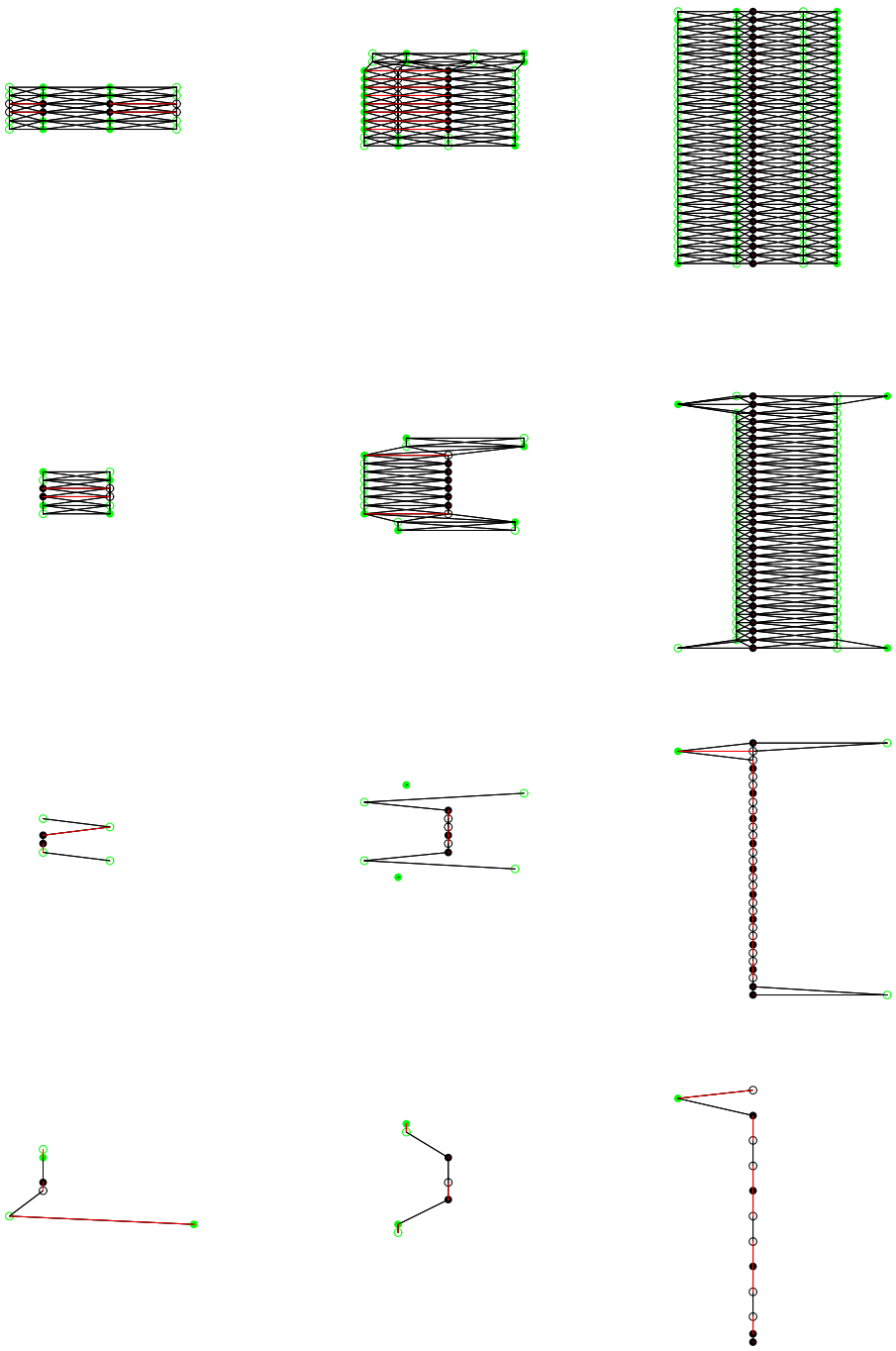


Abbildung 6.33: Matrixgraphen der Level -3 bis -6 (v. o. n. u.); horizontale Strömung; $PEC = 10^6$; links 1×16, PE 9; mitte 4×4, PE 9; rechts 16×1, PE 7

Werten für die extremen Streifenaufteilungen. Für schwächer ausgeprägte Strömungen sind die Unterschiede erwartungsgemäß weniger stark; Tabelle 6.33 bestätigt dies. Die beobachteten Erscheinungen sind also durch die durchgeführte Halbvergröberung in Strömungsrichtung bedingt. Die scheinbar überflüssigen Knoten im mittleren Bild der letzten Zeile in Abbildung 6.32 sind Knoten in Überlapptiefe > 2, die aber aufgrund der in Abschnitt 5.3.4.3 besprochenen Abhängigkeiten *nicht* entfernt werden dürfen.

Die Effekte sind genau die gleichen, wie sie schon bei der anisotropen Poissongleichung beobachtet und erklärt wurden (vergleiche die Tabellen 6.4, 6.17, 6.18, 6.21 und 6.23). Für die Einschätzung der Stärke des Einflusses ist die Skalierung der Diffusionsstärke $\varepsilon = \frac{h}{PEC}$ mit der Gitterweite zu beachten, die zu noch stärkerer Anisotropie in den Matrizen führt. Daher tritt auch für die größten hier untersuchten Pecletzahlen auf den feinsten Startgittern noch keine so stark die Anisotropie ausgleichende Wirkung ein.

Das Zusammenwirken aller bisher analysierten Eigenschaften bestimmt die erzielbare Effizienz. Die Tabellen 6.35 und 6.36 zeigen wieder, daß eine Speedup Betrachtung über zu viele Prozessoren hinweg nicht angemessen ist, weil bei Einsatz zu vieler Prozessoren jedem viel zu wenig Knoten bleiben. Solange aber jeder Prozessor wenigstens zu einigen Prozent ausgelastet ist und die Strömung nicht senkrecht zu den Gebietsstreifen verläuft, liegt der Speedup über 50% und ist damit akzeptabel. Das gilt auch für den Übergang von 32 auf 128 Prozessoren, der für realistische Supercomputeranwendungen von Bedeutung ist.

Interessanter ist wieder die Scaleup Betrachtung (Tabellen 6.37 bis 6.42). Die Effizienz einer Iteration E_{IT} wird im wesentlichen vom Lastungleichgewicht und der Interfacelänge bestimmt. Dementsprechend ist die Boxaufteilung am besten, dicht gefolgt von der horizontalen Streifenaufteilung und danach folgt deutlich schlechter die vertikale Streifenaufteilung. Für sehr starke Strömungen wird der Überlapp fast nur in Strömungsrichtung erzeugt, bei horizontaler Streifenaufteilung also fast keiner. Daher ist dort das gesamte Prozessorinterface sogar etwas kleiner als für eine Boxaufteilung (vergleiche auch die Abbildungen 6.34 und 6.35). Da die Anzahl benötigter Iterationen konstant oder nur sehr leicht mit der Prozessorzahl ansteigend ist, verhält sich die Effizienz E_{SOL} des Mehrgitterlösers dementsprechend. Da bei großer Pecletzahl normalerweise 1 Iterationsschritt benötigt wird, für die größten berechneten Probleme jedoch 2 Schritte, wirkt sich das auf eine relative Größe wie die Effizienz übermäßig stark aus.

Die Skalierbarkeit der Gitteraufbauphase E^*_{FA} ist für eine Boxaufteilung und kleinere Pecletzahlen sehr ähnlich den Werten für die Poissongleichung (siehe Tabelle 6.6) und damit sehr gut. Die absoluten Zeiten liegen hier aber deutlich höher. Ursache ist der größere durchschnittliche Matrixstern: Hat jeder Knoten jetzt ca. 10 Nachbarn, waren es bei der Poissongleichung nur ca. 7,5. Für große Pecletzahlen gehen die Zeiten zurück, weil beim rechenzeitintensiven Elternauswahlprozeß nur die starken Matrixeinträge betrachtet werden und so für starke Strömungen eine starke Vorauswahl

horizontale Streifenaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
2	35.73	1.45	8.69	45.70	0.82	0.96	1.28	0.91	33.27	0.86	0.95
4	24.22	0.81	4.87	30.63	0.60	0.85	1.14	0.68	19.21	0.75	0.82
8	16.55	0.52	3.15	21.02	0.44	0.66	0.88	0.49	11.62	0.62	0.67
16	12.94	0.34	2.06	16.81	0.28	0.51	0.67	0.31	7.89	0.45	0.50
32	11.64	0.30	2.71	16.32	0.16	0.29	0.26	0.16	6.30	0.28	0.28
64	11.87	0.22	1.35	14.91	0.08	0.19	0.26	0.09	6.30	0.14	0.16
128	15.64	0.25	1.49	18.98	0.03	0.09	0.12	0.03	6.91	0.06	0.07
	insgesamt 4 198 401 Knoten										
32	362.56	7.02	63.15	429.48	1.00	1.00	1.00	1.00	184.98	1.00	1.00
64	323.84	4.17	41.69	366.95	0.56	0.84	0.76	0.59	120.38	0.77	0.77
128	271.55	2.55	28.09	302.61	0.33	0.69	0.56	0.35	90.00	0.51	0.53

Boxaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
2	35.37	1.49	10.43	48.91	0.83	0.93	1.06	0.85	30.71	0.93	0.97
4	21.15	0.83	5.82	29.98	0.69	0.83	0.95	0.69	18.30	0.78	0.82
8	15.06	0.53	3.70	22.44	0.49	0.65	0.75	0.46	10.97	0.65	0.68
16	9.40	0.31	2.15	13.32	0.39	0.56	0.64	0.39	6.22	0.58	0.59
32	6.72	0.22	1.53	9.93	0.27	0.40	0.45	0.26	3.94	0.45	0.45
64	4.65	0.15	1.05	7.09	0.20	0.29	0.33	0.18	2.52	0.35	0.35
128	4.70	0.13	0.94	7.20	0.10	0.16	0.18	0.09	2.15	0.21	0.20
	insgesamt 4 198 401 Knoten										
32	252.11	6.59	46.11	302.90	1.00	1.00	1.00	1.00	170.77	1.00	1.00
64	138.23	3.42	23.96	164.82	0.91	0.96	0.96	0.92	89.27	0.96	0.96
128	92.83	1.94	13.60	109.22	0.68	0.85	0.85	0.69	52.63	0.81	0.82

vertikale Streifenaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
2	35.37	1.49	10.43	48.91	0.83	0.93	1.06	0.85	30.71	0.93	0.97
4	25.90	0.88	6.14	37.25	0.57	0.79	0.90	0.56	19.58	0.73	0.77
8	18.85	0.57	3.99	28.72	0.39	0.61	0.69	0.36	12.46	0.57	0.60
16	18.92	0.40	2.81	27.54	0.19	0.43	0.49	0.19	11.45	0.31	0.35
32	17.82	0.33	2.34	24.10	0.10	0.26	0.30	0.11	9.39	0.19	0.21
64	16.22	0.42	2.98	20.27	0.06	0.10	0.12	0.06	8.52	0.11	0.11
128	29.71	0.40	2.39	36.28	0.02	0.05	0.07	0.02	12.23	0.04	0.04
	insgesamt 4 198 401 Knoten										
32	558.83	10.62	95.62	658.05	1.00	1.00	1.00	1.00	246.50	1.00	1.00
64	554.05	5.62	50.60	609.00	0.50	0.94	0.94	0.54	228.94	0.54	0.61
128	651.90	4.57	36.53	692.34	0.21	0.58	0.65	0.24	275.60	0.22	0.27

Tabelle 6.35: Horizontale Strömung, $PEC = 1$, verschiedene Gebietsaufteilungen; Speedup für insgesamt 64 049 und 4 198 401 Knoten

horizontale Streifenaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
2	30.61	1.38	1.38	32.36	0.88	0.93	0.93	0.88	28.72	0.92	0.92
4	21.33	0.75	0.75	22.36	0.63	0.86	0.86	0.64	17.06	0.78	0.78
8	14.72	0.45	0.45	15.41	0.46	0.72	0.72	0.46	10.45	0.63	0.64
16	11.42	0.28	0.28	11.92	0.29	0.58	0.58	0.30	7.21	0.46	0.46
32	10.28	0.20	0.20	10.72	0.16	0.40	0.40	0.17	5.79	0.29	0.29
64	10.09	0.17	0.17	10.49	0.08	0.24	0.24	0.08	5.51	0.15	0.15
128	12.26	0.29	0.29	12.95	0.03	0.07	0.07	0.03	5.51	0.07	0.07
	insgesamt 4 198 401 Knoten										
32	321.98	6.21	12.41	335.46	1.00	1.00	1.00	1.00	159.17	1.00	1.00
64	235.09	3.44	6.88	242.61	0.68	0.90	0.90	0.69	90.77	0.88	0.88
128	221.95	2.19	4.37	226.84	0.36	0.71	0.71	0.37	65.42	0.61	0.61

Boxaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
2	31.88	1.42	1.42	33.67	0.84	0.91	0.91	0.85	27.47	0.96	0.96
4	18.79	0.77	0.77	19.84	0.72	0.83	0.83	0.72	16.43	0.81	0.81
8	13.76	0.49	0.49	14.53	0.49	0.66	0.66	0.49	10.21	0.65	0.65
16	8.78	0.27	0.27	9.30	0.38	0.59	0.59	0.38	6.07	0.54	0.55
32	6.83	0.19	0.19	7.28	0.25	0.41	0.41	0.24	4.40	0.38	0.38
64	4.71	0.14	0.14	5.11	0.18	0.29	0.29	0.17	2.89	0.29	0.29
128	5.12	0.13	0.13	5.57	0.08	0.15	0.15	0.08	2.78	0.15	0.15
	insgesamt 4 198 401 Knoten										
32	271.19	6.53	13.05	285.37	1.00	1.00	1.00	1.00	152.34	1.00	1.00
64	148.17	3.38	6.77	155.59	0.92	0.96	0.96	0.92	80.62	0.94	0.95
128	116.33	2.08	4.15	120.98	0.58	0.79	0.79	0.59	56.74	0.67	0.68

vertikale Streifenaufteilung

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
2	31.88	1.42	1.42	33.67	0.84	0.91	0.91	0.85	27.47	0.96	0.96
4	24.73	0.84	0.84	25.91	0.54	0.76	0.76	0.55	19.14	0.69	0.69
8	19.51	0.54	0.54	20.36	0.34	0.59	0.59	0.35	12.86	0.51	0.52
16	16.50	0.37	0.37	17.16	0.20	0.44	0.44	0.21	10.33	0.32	0.32
32	22.12	0.92	0.92	23.47	0.08	0.09	0.09	0.08	13.65	0.12	0.12
64	28.10	0.37	0.37	28.75	0.03	0.11	0.11	0.03	17.02	0.05	0.05
128	36.68	0.43	0.43	37.55	0.01	0.05	0.05	0.01	15.14	0.03	0.03
	insgesamt 4 198 401 Knoten										
32	1187.04	10.17	20.35	1209.37	1.00	1.00	1.00	1.00	440.52	1.00	1.00
64	1345.33	7.92	15.84	1362.93	0.44	0.64	0.64	0.44	460.70	0.48	0.48
128	1712.08	7.71	15.43	1729.52	0.17	0.33	0.33	0.17	612.62	0.18	0.18

Tabelle 6.36: Horizontale Strömung, $PEC = 10^6$, verschiedene Gebietsaufteilungen; Speedup für insgesamt 64 049 und 4 198 401 Knoten

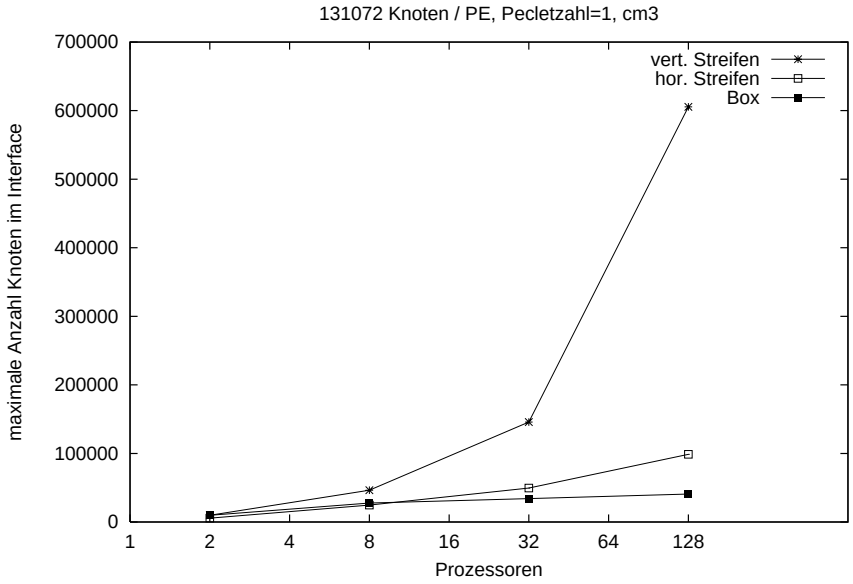


Abbildung 6.34: Horizontale Strömung $PEC = 1$; volle Last; Abhängigkeit der Interfacelänge von der Gebietsaufteilung und der Prozessorzahl

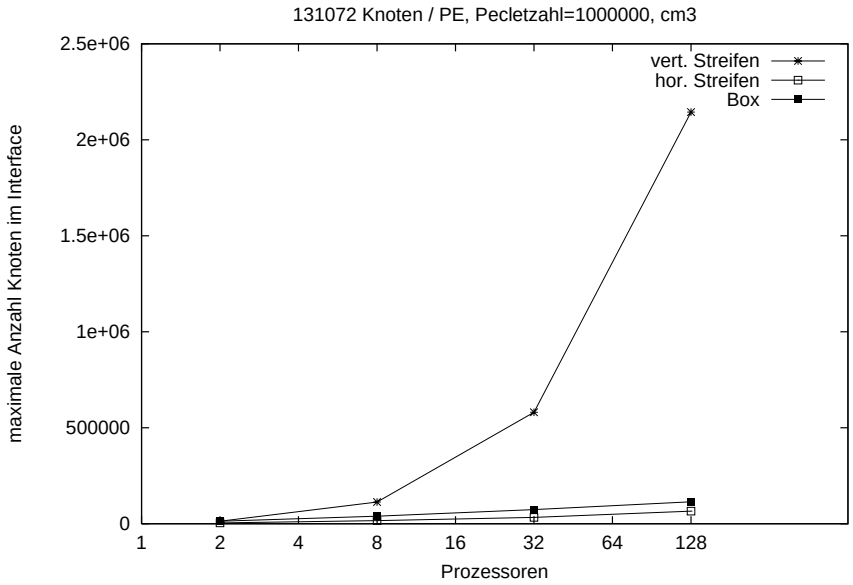


Abbildung 6.35: Horizontale Strömung $PEC = 10^6$; volle Last; Abhängigkeit der Interfacelänge von der Gebietsaufteilung und der Prozessorzahl

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	150.08	6.03	42.24	197.86	1.00	1.00	1.00	1.00	134.65	1.00	1.00
8	215.17	6.50	51.96	273.53	0.70	0.93	0.81	0.72	163.71	0.82	0.82
32	252.11	6.59	46.11	302.90	0.60	0.92	0.92	0.65	170.77	0.79	0.82
128	286.56	6.76	54.10	344.21	0.52	0.89	0.78	0.57	178.78	0.75	0.76
	32 768 Knoten pro Prozessor										
2	35.37	1.49	10.43	48.91	1.00	1.00	1.00	1.00	30.71	1.00	1.00
8	57.65	1.78	12.45	75.56	0.61	0.84	0.84	0.65	42.12	0.73	0.75
32	74.86	1.87	13.12	92.58	0.47	0.79	0.79	0.53	46.89	0.65	0.69
128	92.83	1.94	13.60	109.22	0.38	0.77	0.77	0.45	52.63	0.58	0.62
	65 536 Knoten pro Prozessor										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
4	87.68	3.08	21.56	113.42	0.67	0.90	1.03	0.73	77.07	0.74	0.81
16	120.21	3.35	30.14	154.82	0.49	0.83	0.74	0.53	84.01	0.68	0.70
64	138.23	3.42	23.96	164.82	0.42	0.81	0.93	0.50	89.27	0.64	0.70
	16 384 Knoten pro Prozessor										
1	10.96	0.60	3.58	17.02	1.00	1.00	1.00	1.00	10.43	1.00	1.00
4	21.15	0.83	5.82	29.98	0.52	0.72	0.62	0.57	18.30	0.57	0.58
16	33.63	0.95	6.67	43.77	0.33	0.63	0.54	0.39	22.45	0.46	0.48
64	43.70	1.04	7.31	53.58	0.25	0.57	0.49	0.32	26.01	0.40	0.42

Tabelle 6.37: Horizontale Strömung, $PEC = 1$, Boxaufteilung; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	147.34	5.85	46.82	199.29	1.00	1.00	1.00	1.00	137.93	1.00	1.00
8	222.30	6.43	51.44	276.51	0.66	0.91	0.91	0.72	160.47	0.86	0.87
32	362.56	7.02	63.15	429.48	0.41	0.83	0.74	0.46	184.98	0.75	0.74
128	733.63	7.72	61.78	797.53	0.20	0.76	0.76	0.25	248.72	0.55	0.60
	32 768 Knoten pro Prozessor										
2	35.73	1.45	8.69	45.70	1.00	1.00	1.00	1.00	33.27	1.00	1.00
8	61.24	1.71	13.67	78.44	0.58	0.85	0.64	0.58	42.16	0.79	0.75
32	128.42	2.07	16.54	146.13	0.28	0.70	0.53	0.31	56.65	0.59	0.57
128	271.55	2.55	28.09	302.61	0.13	0.57	0.31	0.15	90.00	0.37	0.36
	65 536 Knoten pro Prozessor										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
4	96.28	3.13	25.03	123.52	0.61	0.89	0.89	0.67	76.64	0.75	0.78
16	151.81	3.53	28.26	182.79	0.39	0.78	0.78	0.45	89.45	0.64	0.67
64	323.84	4.17	41.69	366.95	0.18	0.66	0.53	0.23	120.38	0.48	0.49
	16 384 Knoten pro Prozessor										
1	10.96	0.60	3.58	17.02	1.00	1.00	1.00	1.00	10.43	1.00	1.00
4	24.22	0.81	4.87	30.63	0.45	0.74	0.74	0.56	19.21	0.54	0.58
16	44.38	1.00	8.03	54.61	0.25	0.60	0.45	0.31	25.67	0.41	0.42
64	104.59	1.38	12.41	120.04	0.10	0.43	0.29	0.14	42.26	0.25	0.26

Tabelle 6.38: Horizontale Strömung, $PEC = 1$, horizontale Streifenaufteilung; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	150.08	6.03	42.24	197.86	1.00	1.00	1.00	1.00	134.65	1.00	1.00
8	256.52	6.79	54.32	316.59	0.59	0.89	0.78	0.62	172.10	0.78	0.78
32	558.83	10.62	95.62	658.05	0.27	0.57	0.44	0.30	246.50	0.55	0.52
128	1779.62	18.24	182.40	1999.03	0.08	0.33	0.23	0.10	756.24	0.18	0.19
	32 768 Knoten pro Prozessor										
2	35.37	1.49	10.43	48.91	1.00	1.00	1.00	1.00	30.71	1.00	1.00
8	69.98	1.90	13.27	90.43	0.51	0.79	0.79	0.54	45.61	0.67	0.70
32	182.34	2.54	20.33	205.12	0.19	0.59	0.51	0.24	80.39	0.38	0.41
128	651.90	4.57	36.53	692.34	0.05	0.33	0.29	0.07	275.60	0.11	0.13
	65 536 Knoten pro Prozessor										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
4	103.23	3.28	22.95	133.84	0.57	0.85	0.97	0.62	78.26	0.73	0.78
16	209.44	3.92	31.40	242.91	0.28	0.71	0.71	0.34	108.64	0.53	0.57
64	554.05	5.62	50.60	609.00	0.11	0.49	0.44	0.14	228.94	0.25	0.28
	16 384 Knoten pro Prozessor										
1	10.96	0.60	3.58	17.02	1.00	1.00	1.00	1.00	10.43	1.00	1.00
4	25.90	0.88	6.14	37.25	0.42	0.68	0.58	0.46	19.58	0.53	0.54
16	67.05	1.48	11.83	80.89	0.16	0.40	0.30	0.21	34.26	0.30	0.30
64	221.88	2.14	17.11	245.32	0.05	0.28	0.21	0.07	85.81	0.12	0.14

Tabelle 6.39: Horizontale Strömung, $PEC = 1$, vertikale Streifenaufteilung; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	129.75	5.67	5.67	136.35	1.00	1.00	1.00	1.00	112.96	1.00	1.00
8	190.86	6.20	6.20	198.11	0.68	0.91	0.91	0.69	133.81	0.84	0.85
32	271.19	6.53	13.05	285.37	0.48	0.87	0.43	0.48	152.34	0.74	0.72
128	353.72	7.33	14.66	370.01	0.37	0.77	0.39	0.37	174.73	0.65	0.63
	32 768 Knoten pro Prozessor										
2	31.88	1.42	1.42	33.67	1.00	1.00	1.00	1.00	27.47	1.00	1.00
8	52.01	1.67	1.67	54.03	0.61	0.85	0.85	0.62	36.22	0.76	0.76
32	82.63	1.95	1.95	84.99	0.39	0.73	0.73	0.40	44.60	0.62	0.62
128	116.33	2.08	4.15	120.98	0.27	0.68	0.34	0.28	56.74	0.48	0.47
	65 536 Knoten pro Prozessor										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
4	72.30	2.92	2.92	75.75	0.74	0.88	0.88	0.75	62.57	0.85	0.85
16	108.20	3.24	3.24	112.07	0.50	0.79	0.79	0.51	70.81	0.75	0.75
64	148.17	3.38	6.77	155.59	0.36	0.76	0.38	0.37	80.62	0.66	0.64
	16 384 Knoten pro Prozessor										
1	10.72	0.56	0.56	11.83	1.00	1.00	1.00	1.00	10.51	1.00	1.00
4	18.79	0.77	0.77	19.84	0.57	0.73	0.73	0.60	16.43	0.64	0.64
16	30.91	0.94	0.94	32.09	0.35	0.60	0.60	0.37	20.12	0.52	0.53
64	42.13	0.98	0.98	43.45	0.25	0.58	0.58	0.27	24.49	0.43	0.43

Tabelle 6.40: Horizontale Strömung, $PEC = 10^6$, Boxaufteilung; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	122.65	5.56	5.56	129.13	1.00	1.00	1.00	1.00	115.32	1.00	1.00
8	181.10	5.86	5.86	187.94	0.68	0.95	0.95	0.69	128.51	0.90	0.90
32	321.98	6.21	12.41	335.46	0.38	0.90	0.45	0.38	159.17	0.72	0.70
128	734.52	6.93	13.85	749.61	0.17	0.80	0.40	0.17	213.78	0.54	0.53
	32 768 Knoten pro Prozessor										
2	30.61	1.38	1.38	32.36	1.00	1.00	1.00	1.00	28.72	1.00	1.00
8	52.15	1.60	1.60	54.08	0.59	0.86	0.86	0.60	35.17	0.82	0.82
32	96.68	1.84	1.84	98.89	0.32	0.75	0.75	0.33	44.10	0.65	0.66
128	221.95	2.19	4.37	226.84	0.14	0.63	0.32	0.14	65.42	0.44	0.43
	65 536 Knoten pro Prozessor										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
4	79.82	2.85	2.85	83.19	0.67	0.90	0.90	0.68	62.95	0.84	0.84
16	124.88	3.08	3.08	128.53	0.43	0.83	0.83	0.44	72.16	0.73	0.74
64	235.09	3.44	6.88	242.61	0.23	0.75	0.37	0.23	90.77	0.58	0.57
	16 384 Knoten pro Prozessor										
1	10.72	0.56	0.56	11.83	1.00	1.00	1.00	1.00	10.51	1.00	1.00
4	21.33	0.75	0.75	22.36	0.50	0.75	0.75	0.53	17.06	0.62	0.62
16	38.41	1.05	1.05	39.73	0.28	0.54	0.54	0.30	21.54	0.49	0.49
64	81.71	1.03	1.03	83.01	0.13	0.55	0.55	0.14	30.40	0.35	0.35

Tabelle 6.41: Horizontale Strömung, $PEC = 10^6$, horizontale Streifenaufteilung; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	129.75	5.67	5.67	136.35	1.00	1.00	1.00	1.00	112.96	1.00	1.00
8	343.96	6.89	6.89	351.99	0.38	0.82	0.82	0.39	170.50	0.66	0.67
32	1187.04	10.17	20.35	1209.37	0.11	0.56	0.28	0.11	440.52	0.26	0.26
128	5738.10	27.98	55.96	5850.14	0.02	0.20	0.10	0.02	1602.51	0.07	0.07
	32 768 Knoten pro Prozessor										
2	31.88	1.42	1.42	33.67	1.00	1.00	1.00	1.00	27.47	1.00	1.00
8	78.26	1.92	1.92	80.93	0.41	0.74	0.74	0.42	44.87	0.61	0.62
32	295.25	3.14	3.14	299.09	0.11	0.45	0.45	0.11	139.28	0.20	0.20
128	1712.08	7.71	15.43	1729.52	0.02	0.18	0.09	0.02	612.62	0.04	0.05
	65 536 Knoten pro Prozessor										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
4	98.63	3.18	3.18	102.42	0.55	0.81	0.81	0.56	71.60	0.74	0.74
16	308.92	4.29	4.29	314.03	0.17	0.60	0.60	0.18	135.99	0.39	0.40
64	1345.33	7.92	15.84	1362.93	0.04	0.32	0.16	0.04	460.70	0.11	0.12
	16 384 Knoten pro Prozessor										
1	10.72	0.56	0.56	11.83	1.00	1.00	1.00	1.00	10.51	1.00	1.00
4	24.73	0.84	0.84	25.91	0.43	0.67	0.67	0.46	19.14	0.55	0.55
16	85.02	1.29	1.29	86.60	0.13	0.44	0.44	0.14	41.69	0.25	0.26
64	340.86	2.70	2.70	344.36	0.03	0.21	0.21	0.03	147.79	0.07	0.07

Tabelle 6.42: Horizontale Strömung, $PEC = 10^6$, vertikale Streifenaufteilung; Scaleup für volle bis achtel Last

getroffen werden kann. Da aber für größere Prozessorzahlen die Kommunikationskosten deutlich zunehmen und unabhängig von der Größe der Matrixeinträge sind, skalieren die Zeiten etwas schlechter als bei schwacher Strömung.

Da selbst für $PEC = 1$ immer noch eine Strömung anliegt und so eine Halbvergröberungsrichtung vorgibt, fallen die Unterschiede zwischen horizontaler und vertikaler Streifenaufteilung recht deutlich aus. Für alle Pecletzahlen sind die Effizienzen für Boxaufteilungen sehr gut und für Streifenaufteilungen, die mit der Strömung ausgerichtet sind, gut. Nur für Aufteilungen senkrecht zur Strömungsrichtung sind die Effizienzen erwartungsgemäß sehr schlecht.

Insgesamt verhält sich die Konvektions–Diffusions–Gleichung mit horizontaler Strömung sehr ähnlich einer anisotropen Poissongleichung. Das entspricht auch den Erwartungen an die Parallelisierung.

Auch hier sollen die drei Färbungsmethoden gegenübergestellt werden. Besonders interessiert uns, ob cm1 durch das sequentielle Vorgehen in Strömungsrichtung vor allem bei vertikaler Streifenaufteilung einen Vorteil bringt.

Die Anzahl benötigter Farben ist in Abbildung 6.36 für die größten Rechnungen wiedergegeben. Bis zur Hälfte der algebraischen Level wird (fast) nur die minimal notwendige Anzahl Farben verwendet. Durch dichter werdende Prozessornachbarschaftsgraphen werden für gröbere Level mehr Farben benötigt. Daß für die vertikale Streifenaufteilung und $\varepsilon = 10^6$ nur so wenige Farben benötigt werden, liegt am Markierungsalgorithmus. Die Verhältnisse der Matrixeinträge sind so eindeutig, daß für alle Knoten auf einer vertikalen Linie immer die gleiche Markierungsentscheidung getroffen wird. Unterstützt wird das durch die vertikale Streifenaufteilung (vergleiche Abbildung 6.33, rechte Bildspalte). So reichen zwar die Matrixeinträge über große Entfernungen, aber alle Einträge einer Tiefe verbinden nur Knoten zweier Prozessoren (unter Umständen werden dazwischen mehrere bereits leere Prozessoren übersprungen). So wird der Prozessornachbarschaftsgraph im Vergleich zum Ausgangsgitter kaum dichter und es reichen stets wenige Farben aus. Die Heuristik cm3 erzeugt entgegen der Erwartung nach [GJP96] teilweise mehr Farben als cm2.

Ein wesentlicher Einflußfaktor wird die Interfacelänge sein. In Abbildung 6.37 sind die charakteristischen Fälle dargestellt. Das Bild oben zeigt für die horizontale Streifenaufteilung, daß bei steigender Pecletzahl die Interfacelänge etwas abnimmt. Die Ursache dafür ist, daß bei starker Strömung keine Vergröberung senkrecht zur Strömung durchgeführt wird und sich in dieser Richtung also der Überlapp nicht mehr vergrößert. Für die vertikale Streifenaufteilung (im Bild unten) sieht man für cm1 einen ausgeprägten Buckel für größere Prozessorzahlen; für cm2 und cm3 einen ausgeglichenen Verlauf. Als Ursache für die Verschlechterung bei cm1 sind wieder Störungen am (linken) Gebietsrand anzunehmen, die schrittweise weiter ins Gebietsinnere getragen werden. Für starke Strömungen ergibt sich, wie bei der Poissongleichung besprochen, eine große Reichweite der Matrixeinträge, die aber quer zur Strömungsrichtung nur

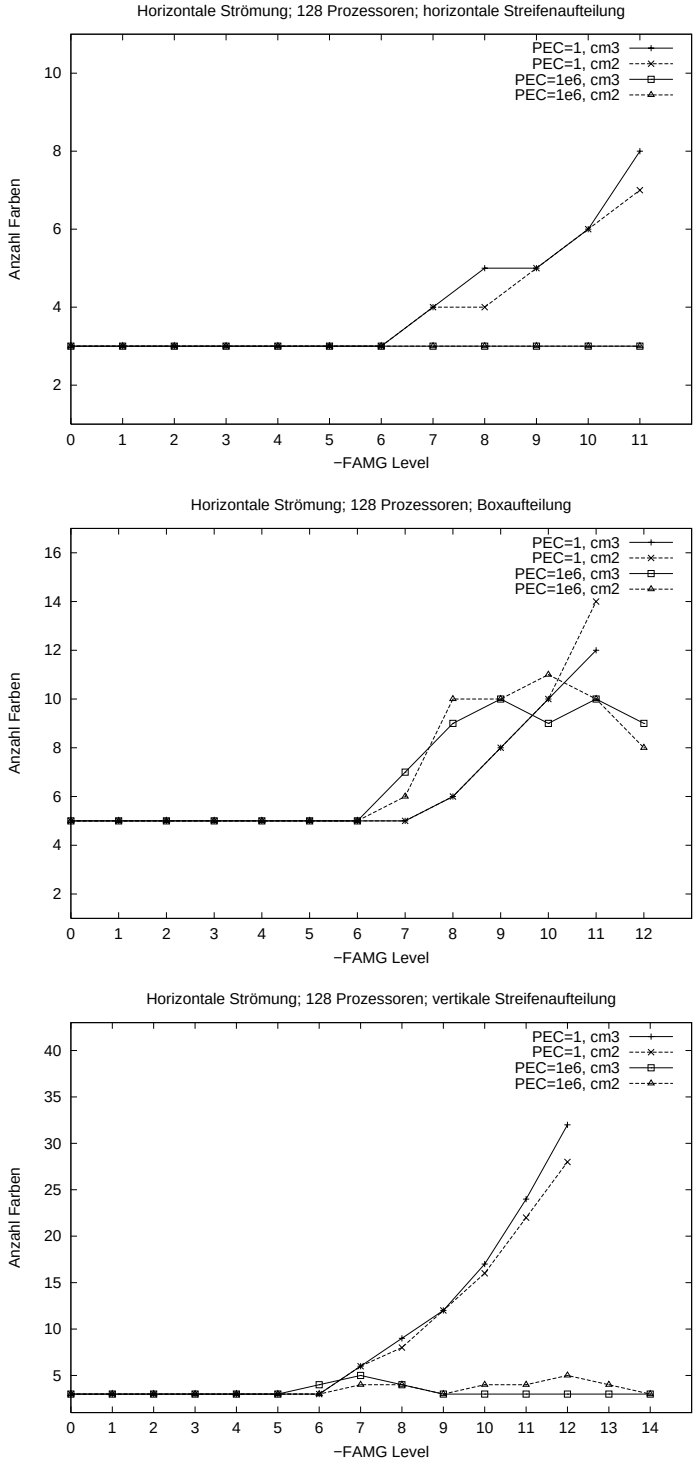


Abbildung 6.36: Horizontale Strömung; volle/halbe Last; Anzahl benötigter Farben in Abhängigkeit der Färbungsmethode

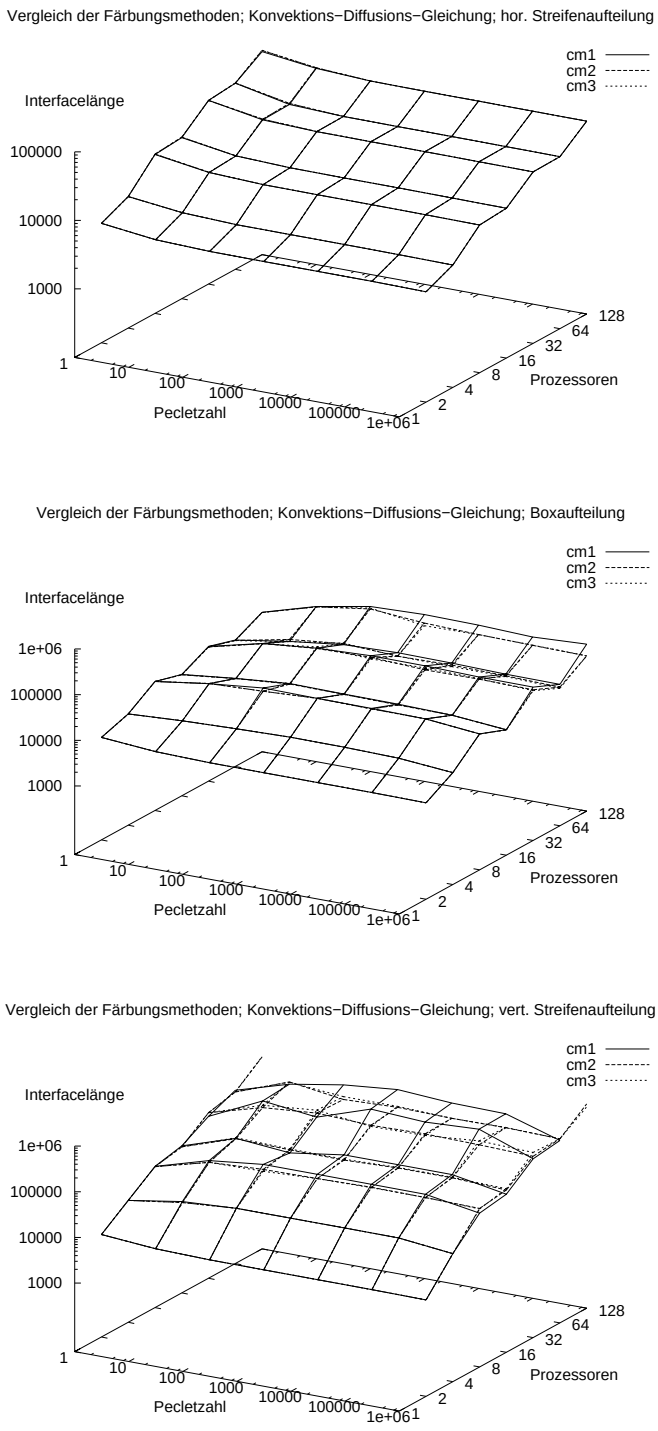


Abbildung 6.37: Horizontale Strömung; alle Gebietsaufteilungen; volle/halbe Last; Abhängigkeit der Interfacelänge von der Färbungsmethode

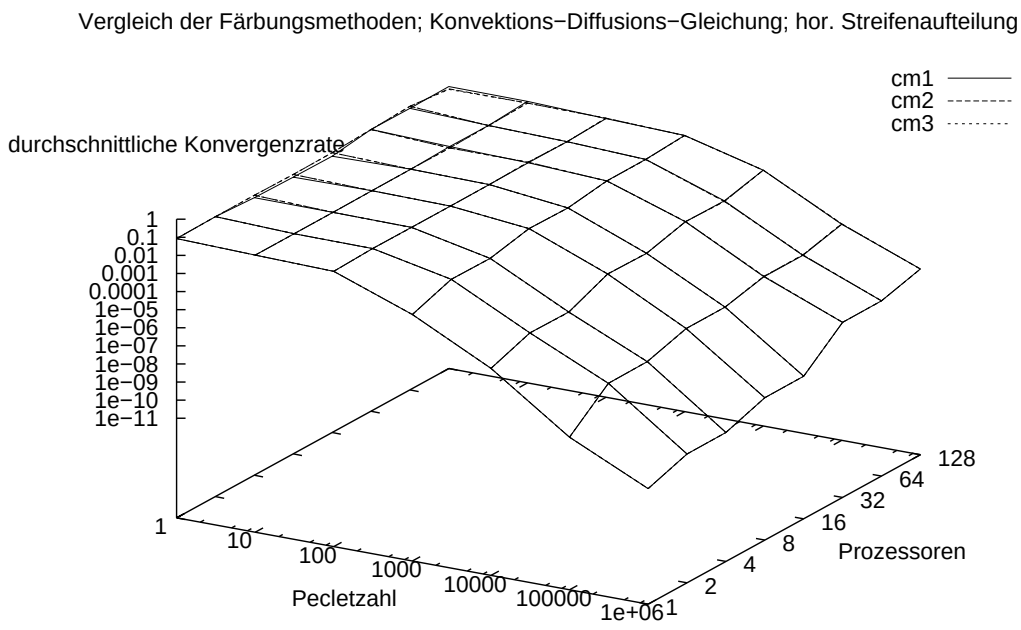
wenig auffächern. Für mittlere Pecletzahlen addieren sich beide Effekte (große Reichweite durch merkliche Strömungsstärke, aber immer noch erhebliche Auffächerung durch einen gewissen isotropen Anteil) und vergrößern die Interfacelänge deutlich. Die Boxaufteilung nimmt wieder eine Mittelstellung zwischen den beiden extremen Streifenaufteilungen ein.

In den Abbildungen 6.38 bis 6.40 sind die Ergebnisse für den Vergleich der Färbungsmethoden zusammengefaßt. Die Konvergenzrate (jeweils die obere Graphik) zeigt vollkommene Robustheit für alle Färbungsmehtoden. Auffällig sind einzelne Störungen für $PEC = 10^6$ und viele Prozessoren, die die Robustheit aber in keiner Weise gefährden. Die Zeiten werden im wesentlichen durch die Interfacelänge beeinflußt. Durch das serielle Bearbeiten der bei Einsatz vieler Prozessoren großen Randbereiche bei cm1 wirkt sich die Interfacelänge besonders stark aus. Daneben wächst die Zeit für die Gitterstrukturänderungen, bedingt durch die enthaltenen Sortiervorgänge, überproportional an.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	149.98	6.04	42.28	197.85	1.00	1.00	1.00	1.00	134.64	1.00	1.00
8	216.59	6.51	52.05	277.38	0.69	0.93	0.81	0.71	165.16	0.82	0.81
32	266.29	6.62	46.34	317.23	0.56	0.91	0.91	0.62	185.32	0.73	0.76
128	406.34	6.72	47.02	457.69	0.37	0.90	0.90	0.43	283.91	0.47	0.53
	32 768 Knoten pro Prozessor										
2	35.40	1.49	10.41	48.92	1.00	1.00	1.00	1.00	30.70	1.00	1.00
8	58.29	1.80	12.57	77.04	0.61	0.83	0.83	0.63	42.73	0.72	0.74
32	80.53	1.87	13.08	98.77	0.44	0.80	0.80	0.50	52.18	0.59	0.63
128	181.81	3.06	21.44	205.98	0.19	0.49	0.49	0.24	122.36	0.25	0.29
	65 536 Knoten pro Prozessor										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
4	87.53	3.15	22.05	112.99	0.67	0.88	1.01	0.73	77.05	0.74	0.80
16	123.41	3.31	26.47	154.43	0.47	0.84	0.84	0.54	86.92	0.66	0.70
64	155.74	3.47	24.32	183.00	0.38	0.80	0.91	0.45	105.38	0.54	0.61
	16 384 Knoten pro Prozessor										
1	10.96	0.60	3.58	17.02	1.00	1.00	1.00	1.00	10.43	1.00	1.00
4	21.18	0.93	6.53	30.71	0.52	0.64	0.55	0.55	18.29	0.57	0.56
16	35.27	0.96	6.69	44.80	0.31	0.63	0.54	0.38	23.57	0.44	0.46
64	50.66	1.18	8.28	61.53	0.22	0.51	0.43	0.28	32.49	0.32	0.34

Tabelle 6.43: Horizontale Strömung, $PEC = 1$, Boxaufteilung; Färbungsmethode cm1; Scaleup für volle bis achtel Last

Wir wollen einige Werte quantitativ betrachten (siehe Tabellen 6.43 bis 6.48). Da sich cm2 und cm3 kaum unterscheiden, beschränken wir uns hier auf cm1. Für alle gezeigten Fälle ist die Effizienz eines Iterationsschritts E_{IT} unverändert gegenüber cm3. Die Färbungsmethode wirkt sich also auf die Lastbalance kaum aus. Die Zeit für den Gitteraufbau T^*_{FA} ist für 2 Prozessoren identisch zu cm3, dann wächst die Laufzeit



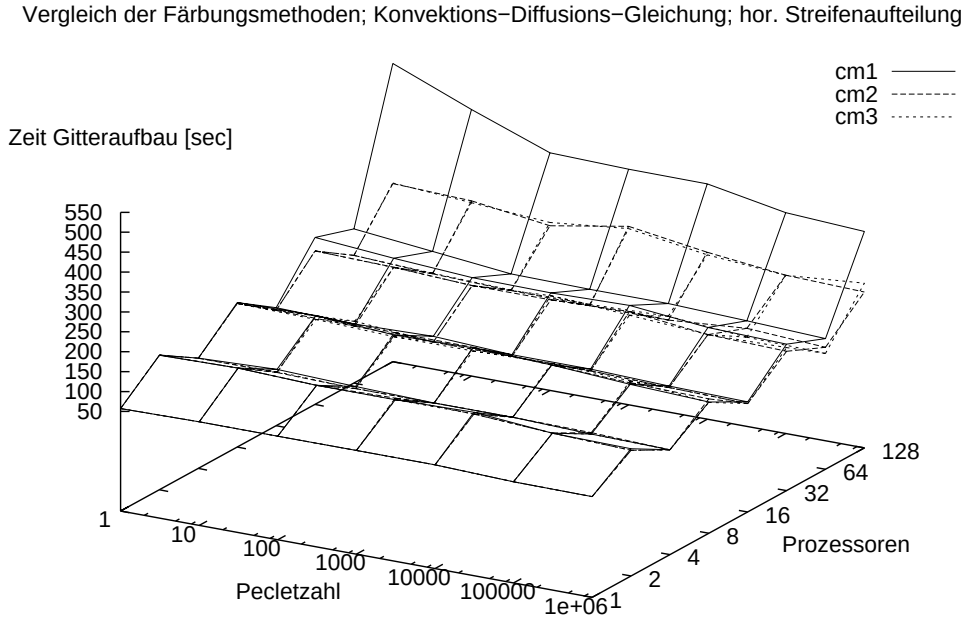
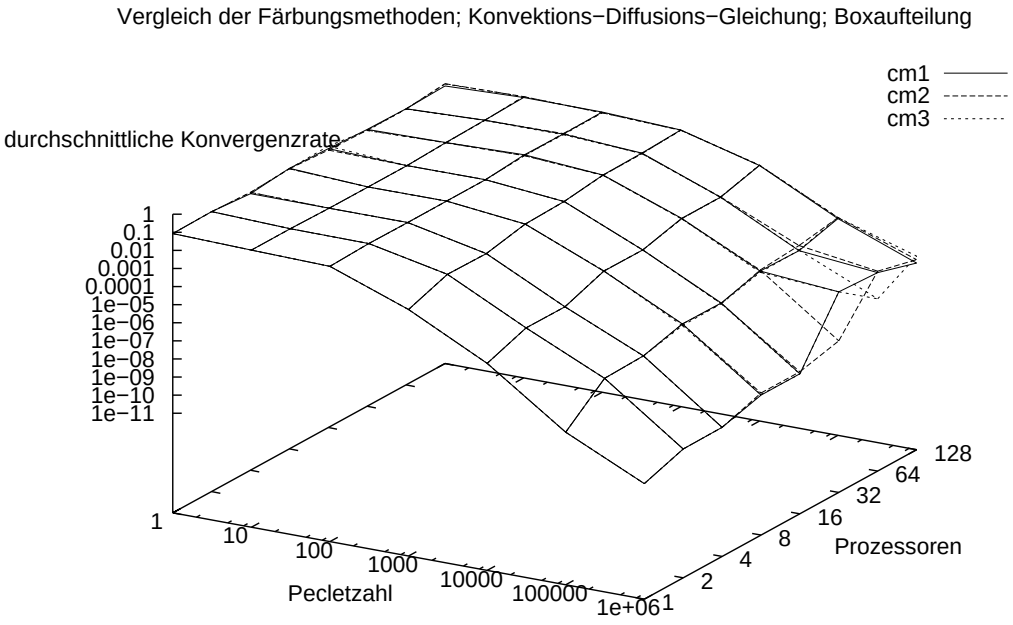


Abbildung 6.38: Horizontale Strömung; horizontale Streifenaufteilung; volle/halbe Last; Abhängigkeit der durchschnittlichen Konvergenzrate und Gitteraufbauzeit T^*_{FA} von der Färbungsmethode



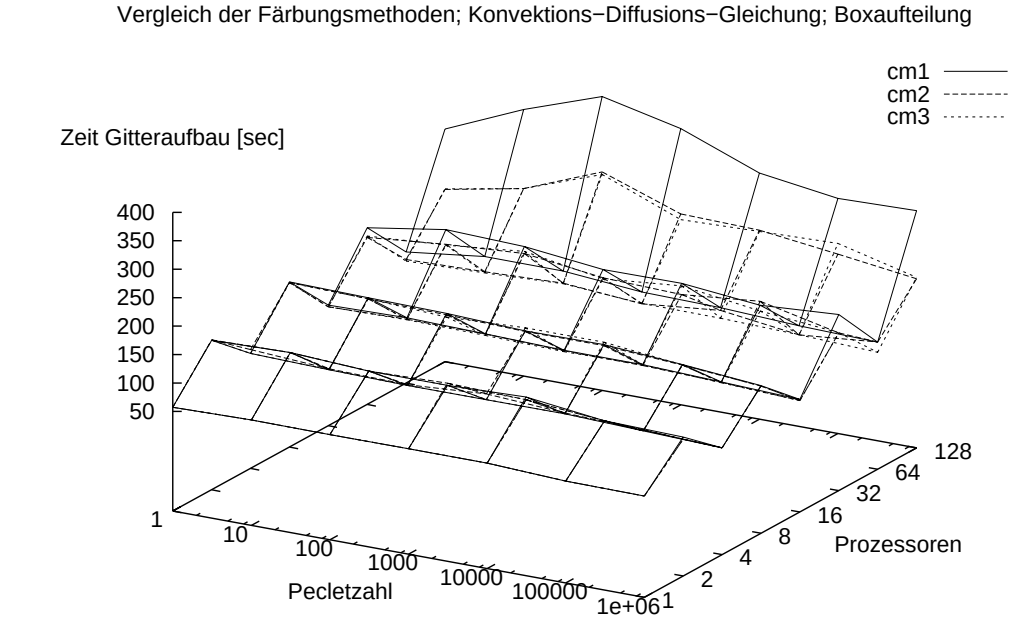
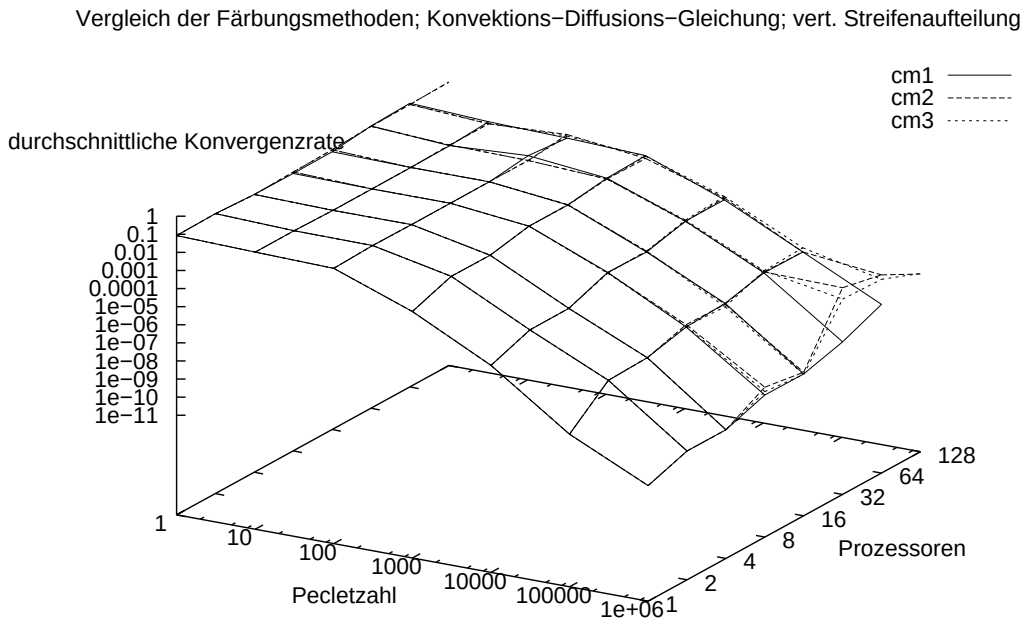


Abbildung 6.39: Horizontale Strömung; Boxaufteilung; volle/halbe Last; Abhängigkeit der durchschnittlichen Konvergenzrate und Gitteraufbauzeit T^*_{FA} von der Färbungsmethode



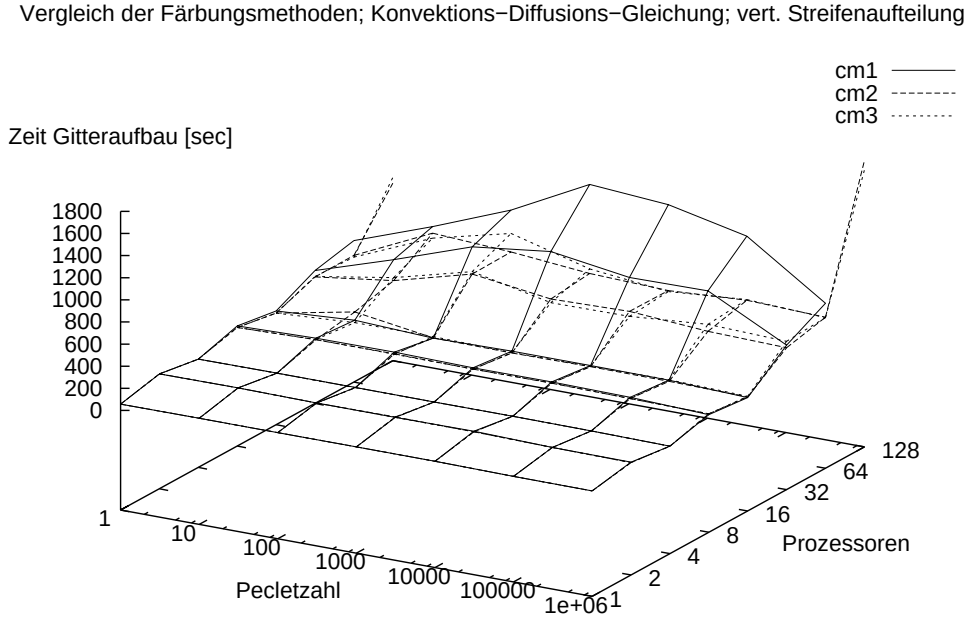


Abbildung 6.40: Horizontale Strömung; vertikale Streifenaufteilung; volle/halbe Last; Abhängigkeit der durchschnittlichen Konvergenzrate und Gitteraufbauzeit T^*_{FA} von der Färbungsmethode

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	146.75	5.85	46.82	198.70	1.00	1.00	1.00	1.00	137.63	1.00	1.00
8	224.49	6.50	45.54	272.86	0.65	0.90	1.03	0.73	162.91	0.84	0.88
32	395.48	7.02	63.15	462.37	0.37	0.83	0.74	0.43	219.17	0.63	0.65
128	989.63	7.70	69.27	1061.38	0.15	0.76	0.68	0.19	548.77	0.25	0.30
	32 768 Knoten pro Prozessor										
2	35.61	1.45	8.69	45.56	1.00	1.00	1.00	1.00	33.18	1.00	1.00
8	69.37	1.73	12.12	84.02	0.51	0.84	0.72	0.54	48.61	0.68	0.69
32	143.82	2.08	14.55	159.63	0.25	0.70	0.60	0.29	72.89	0.46	0.48
128	424.88	2.50	17.50	444.69	0.08	0.58	0.50	0.10	247.29	0.13	0.16
	65 536 Knoten pro Prozessor										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
4	96.39	3.12	21.81	120.35	0.61	0.89	1.02	0.69	76.91	0.74	0.80
16	157.19	3.53	24.69	184.88	0.37	0.79	0.90	0.45	96.39	0.59	0.66
64	390.07	4.19	37.72	429.77	0.15	0.66	0.59	0.19	187.11	0.31	0.35
	16 384 Knoten pro Prozessor										
1	10.96	0.60	3.58	17.02	1.00	1.00	1.00	1.00	10.43	1.00	1.00
4	24.26	0.81	4.89	30.39	0.45	0.73	0.73	0.56	19.33	0.54	0.58
16	48.47	1.18	8.28	59.93	0.23	0.51	0.43	0.28	29.52	0.35	0.37
64	157.40	1.34	12.05	171.08	0.07	0.45	0.30	0.10	88.86	0.12	0.14

Tabelle 6.44: Horizontale Strömung, $PEC = 1$, horizontale Streifenaufteilung; Färbungsmethode cm1; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	149.98	6.04	42.28	197.85	1.00	1.00	1.00	1.00	134.64	1.00	1.00
8	273.15	6.81	47.69	326.97	0.55	0.89	0.89	0.61	185.19	0.73	0.76
32	551.83	7.88	70.88	627.06	0.27	0.77	0.60	0.32	300.46	0.45	0.48
	32 768 Knoten pro Prozessor										
2	35.40	1.49	10.41	48.92	1.00	1.00	1.00	1.00	30.70	1.00	1.00
8	76.01	1.88	13.14	97.18	0.47	0.79	0.79	0.50	49.13	0.62	0.66
32	221.39	2.87	22.95	246.24	0.16	0.52	0.45	0.20	116.64	0.26	0.29
128	1140.19	8.26	82.59	1227.52	0.03	0.18	0.13	0.04	713.43	0.04	0.05
	65 536 Knoten pro Prozessor										
1	58.59	2.77	22.17	82.82	1.00	1.00	1.00	1.00	57.26	1.00	1.00
4	104.02	3.27	22.87	134.78	0.56	0.85	0.97	0.61	78.55	0.73	0.78
16	232.42	3.99	27.92	262.10	0.25	0.69	0.79	0.32	128.00	0.45	0.51
64	758.97	5.56	55.64	817.66	0.08	0.50	0.40	0.10	378.06	0.15	0.18
	16 384 Knoten pro Prozessor										
1	10.96	0.60	3.58	17.02	1.00	1.00	1.00	1.00	10.43	1.00	1.00
4	26.30	0.87	6.10	37.23	0.42	0.69	0.59	0.46	19.98	0.52	0.54
16	72.04	1.24	9.88	85.52	0.15	0.48	0.36	0.20	39.86	0.26	0.28
64	232.13	1.91	15.24	249.60	0.05	0.31	0.24	0.07	126.60	0.08	0.10

Tabelle 6.45: Horizontale Strömung, $PEC = 1$, vertikale Streifenaufteilung; Färbungsmethode cm1; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	129.06	5.74	5.74	135.73	1.00	1.00	1.00	1.00	112.28	1.00	1.00
8	191.58	6.21	6.21	198.85	0.67	0.93	0.93	0.68	134.50	0.83	0.84
32	316.79	9.27	18.55	337.54	0.41	0.62	0.31	0.40	184.41	0.61	0.58
128	497.11	7.27	14.54	513.00	0.26	0.79	0.39	0.26	291.88	0.38	0.39
	32 768 Knoten pro Prozessor										
2	35.67	1.42	1.42	37.47	1.00	1.00	1.00	1.00	30.48	1.00	1.00
8	52.36	1.67	1.67	54.38	0.68	0.85	0.85	0.69	36.48	0.84	0.84
32	87.03	2.04	2.04	89.48	0.41	0.70	0.70	0.42	49.82	0.61	0.62
128	139.84	2.09	4.18	144.62	0.26	0.68	0.34	0.26	88.96	0.34	0.34
	65 536 Knoten pro Prozessor										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
4	72.16	2.94	2.94	75.64	0.75	0.87	0.87	0.75	62.45	0.85	0.85
16	120.17	3.21	3.21	124.02	0.45	0.80	0.80	0.46	72.82	0.73	0.73
64	165.34	3.44	6.88	172.87	0.33	0.75	0.37	0.33	98.25	0.54	0.53
	16 384 Knoten pro Prozessor										
1	10.72	0.56	0.56	11.83	1.00	1.00	1.00	1.00	10.51	1.00	1.00
4	18.86	0.77	0.77	19.91	0.57	0.73	0.73	0.59	16.50	0.64	0.64
16	31.35	0.90	0.90	32.50	0.34	0.62	0.62	0.36	20.57	0.51	0.52
64	48.23	0.98	0.98	49.52	0.22	0.57	0.57	0.24	30.19	0.35	0.36

Tabelle 6.46: Horizontale Strömung, $PEC = 10^6$, Boxaufteilung; Färbungsmethode cm1; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	126.56	5.61	5.61	133.09	1.00	1.00	1.00	1.00	119.21	1.00	1.00
8	182.23	5.86	5.86	189.07	0.69	0.96	0.96	0.70	129.73	0.92	0.92
32	315.91	6.19	12.38	329.35	0.40	0.91	0.45	0.40	167.79	0.71	0.69
128	797.35	10.87	21.74	821.80	0.16	0.52	0.26	0.16	343.42	0.35	0.34
	32 768 Knoten pro Prozessor										
2	30.54	1.42	1.42	32.33	1.00	1.00	1.00	1.00	28.68	1.00	1.00
8	52.90	2.25	2.25	55.62	0.58	0.63	0.63	0.58	35.92	0.80	0.79
32	105.55	1.71	1.71	107.62	0.29	0.83	0.83	0.30	53.15	0.54	0.55
128	296.36	2.11	4.21	301.03	0.10	0.67	0.34	0.11	141.35	0.20	0.21
	65 536 Knoten pro Prozessor										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
4	80.08	2.85	2.85	83.45	0.67	0.90	0.90	0.68	63.14	0.84	0.84
16	128.67	3.10	3.10	132.34	0.42	0.83	0.83	0.43	76.41	0.69	0.70
64	273.23	3.50	7.00	280.88	0.20	0.73	0.37	0.20	128.19	0.41	0.41
	16 384 Knoten pro Prozessor										
1	10.72	0.56	0.56	11.83	1.00	1.00	1.00	1.00	10.51	1.00	1.00
4	21.29	0.75	0.75	22.32	0.50	0.75	0.75	0.53	17.06	0.62	0.62
16	40.10	0.86	0.86	41.19	0.27	0.65	0.65	0.29	23.47	0.45	0.45
64	100.21	1.06	1.06	101.53	0.11	0.53	0.53	0.12	49.01	0.21	0.22

Tabelle 6.47: Horizontale Strömung, $PEC = 10^6$, horizontale Streifenaufteilung; Färbungsmethode cm1; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	129.06	5.74	5.74	135.73	1.00	1.00	1.00	1.00	112.28	1.00	1.00
8	302.31	6.70	6.70	310.14	0.43	0.86	0.86	0.44	163.35	0.69	0.69
32	876.84	10.58	10.58	891.42	0.15	0.54	0.54	0.15	405.03	0.28	0.28
	32 768 Knoten pro Prozessor										
2	35.67	1.42	1.42	37.47	1.00	1.00	1.00	1.00	30.48	1.00	1.00
8	79.40	1.83	1.83	81.63	0.45	0.77	0.77	0.46	46.05	0.66	0.67
32	261.36	2.82	2.82	264.81	0.14	0.50	0.50	0.14	131.80	0.23	0.24
128	1479.71	6.46	12.93	1494.48	0.02	0.22	0.11	0.03	880.54	0.03	0.04
	65 536 Knoten pro Prozessor										
1	53.84	2.57	2.57	56.97	1.00	1.00	1.00	1.00	52.92	1.00	1.00
4	98.72	3.18	3.18	102.49	0.55	0.81	0.81	0.56	71.65	0.74	0.74
16	246.01	4.01	4.01	250.77	0.22	0.64	0.64	0.23	124.08	0.43	0.43
64	1126.94	7.70	15.40	1144.19	0.05	0.33	0.17	0.05	591.46	0.09	0.09
	16 384 Knoten pro Prozessor										
1	10.72	0.56	0.56	11.83	1.00	1.00	1.00	1.00	10.51	1.00	1.00
4	24.80	0.83	0.83	25.98	0.43	0.68	0.68	0.46	19.19	0.55	0.55
16	71.25	1.18	1.18	72.72	0.15	0.48	0.48	0.16	38.65	0.27	0.28
64	313.89	2.44	2.44	318.55	0.03	0.23	0.23	0.04	169.50	0.06	0.06

Tabelle 6.48: Horizontale Strömung, $PEC = 10^6$, vertikale Streifenaufteilung; Färbungsmethode cm1; Scaleup für volle bis achtel Last

durch die Sequentialisierung der Randmarkierung stark an und erhöht die Zeiten spürbar. Damit sinkt die Effizienz E^*_{FA} deutlich mit wachsender Prozessorzahl. Für 128 Prozessoren halbiert sich die Effizienz in etwa. Die Aussagen gelten in ähnlicher Weise auch für die Gesamteffizienz E^*_{Ges}, weil die Konvergenzgüte im wesentlichen unverändert ist.

Das strukturiertere Vorgehen durch Streifenaufteilung und/oder cm1 bringt also keine weitere Verbesserung der numerischen Güte, skaliert aber — wie erwartet — entschieden schlechter. Für beide Varianten gibt es also auch bei der Konvektions–Diffusions–Gleichung keine Rechtfertigung.

6.6.2 Kreisströmung

Hier soll nun die schwierigste der in dieser Arbeit untersuchten Gleichungen betrachtet werden. Bei der Konvektions–Diffusions–Gleichung (6.11) wird durch die Wahl

$$v(x,y) := \begin{pmatrix} 4x(x-1)(1-2y) \\ -4y(y-1)(1-2x) \end{pmatrix}^\top$$

eine kreisförmige Strömung um den Mittelpunkt $(0.5, 0.5)$ des Rechengebiets erzwungen. Die Charakteristiken sind geschlossene Stromlinien. Die damit einhergehenden Probleme werden ausführlich in [Joh00b] diskutiert und wurden hier in Ab-

schnitt 2.3.6.2 zusammengefaßt. Diese Gleichung wird auch als Modellproblem für Transportprozesse in turbulenter Strömung verwendet.

Der Informationsfluß, der entlang den Stromlinien verläuft, führt nicht nur über Prozessorgrenzen hinweg, sondern ist auch noch zyklisch. Im vorigen Abschnitt sahen wir, daß das erste Problem gut gelöst wird. Fraglich ist, ob auch die zyklische Rückkopplung ausreichend gut erfolgt und nicht durch zu viele Prozessorgrenzen zu ungenau wird (etwa durch ein Aufweiten der Stromlinien oder ein Verschieben der Linien).

Einen ersten Eindruck der erzeugten groben Matrizen geben Abbildungen 6.41 und 6.42 und Abbildung 6.43 für einen einzelnen Prozessor. Mit zunehmender Strömungsstärke, d. h. wachsender Pecletzahl, sieht man deutlich die Halbvergröberung entlang konzentrischer Kreise um den Gebietsmittelpunkt. Die Ausbreitung der Halbvergröberung wird auch durch Prozessorgrenzen nicht behindert. Ob die Rückkopplung auch ausreichend gut gewährleistet ist, müssen nun numerische Experimente erweisen.

	h 1/128	1/256	1/512	1/1024	1/2048	1/4096
PE	N 16 641	66 049	263 169	1 050 625	4 198 401	16 785 409
1×1	6 0.036	6 0.040				
2×1	6 0.036	6 0.041	6 0.045			
2×2	6 0.036	6 0.041	6 0.046			
4×2	6 0.036	6 0.040	7 0.050	8 0.090		
4×4	6 0.036	6 0.041	7 0.052	8 0.081		
8×4	6 0.037	6 0.041	6 0.046	8 0.083	9 0.124	
8×8	6 0.037	6 0.040	6 0.045	7 0.062	9 0.112	
16×8	6 0.037	6 0.040	7 0.054	7 0.069	9 0.122	11 0.174

Tabelle 6.49: Kreisströmung, $PEC = 1$; Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessorzahl (Boxaufteilung)

Abbildung 6.44 und Tabellen 6.49 und 6.50 zeigen, daß die Konvergenzgüte für alle Pecletzahlen und Prozessorzahlen sehr gut ist. Lediglich für kleine Pecletzahlen ist eine geringe Konvergenzverschlechterung für sehr feine Startgitter erkennbar. Selbst jedoch für 16,7 Millionen Unbekannte auf 128 Prozessoren sind 11 Iterationen noch sehr gut und es zeichnet sich keine rapide Verschlechterung für feinere Gitter ab. Die Konvergenzrate ist unabhängig von der Prozessorzahl. Das Verfahren konvergiert für alle Konfigurationen der Kreisströmung also sehr gut und es liegt wiederum robustes Verhalten vor.

Abbildung 6.45 zeigt die Robustheit gegenüber der Pecletzahl in Abhängigkeit von der Größe des Startgitters. Daß für Parameterwerte von 10 bis 100 die Konvergenz relativ gesehen am schlechtesten ist, kennt man von vielen singulär gestörten Differentialgleichungen. Abbildung 6.46 zeigt die außerordentliche Konstanz der Konvergenzrate bezüglich der Prozessoranzahl.

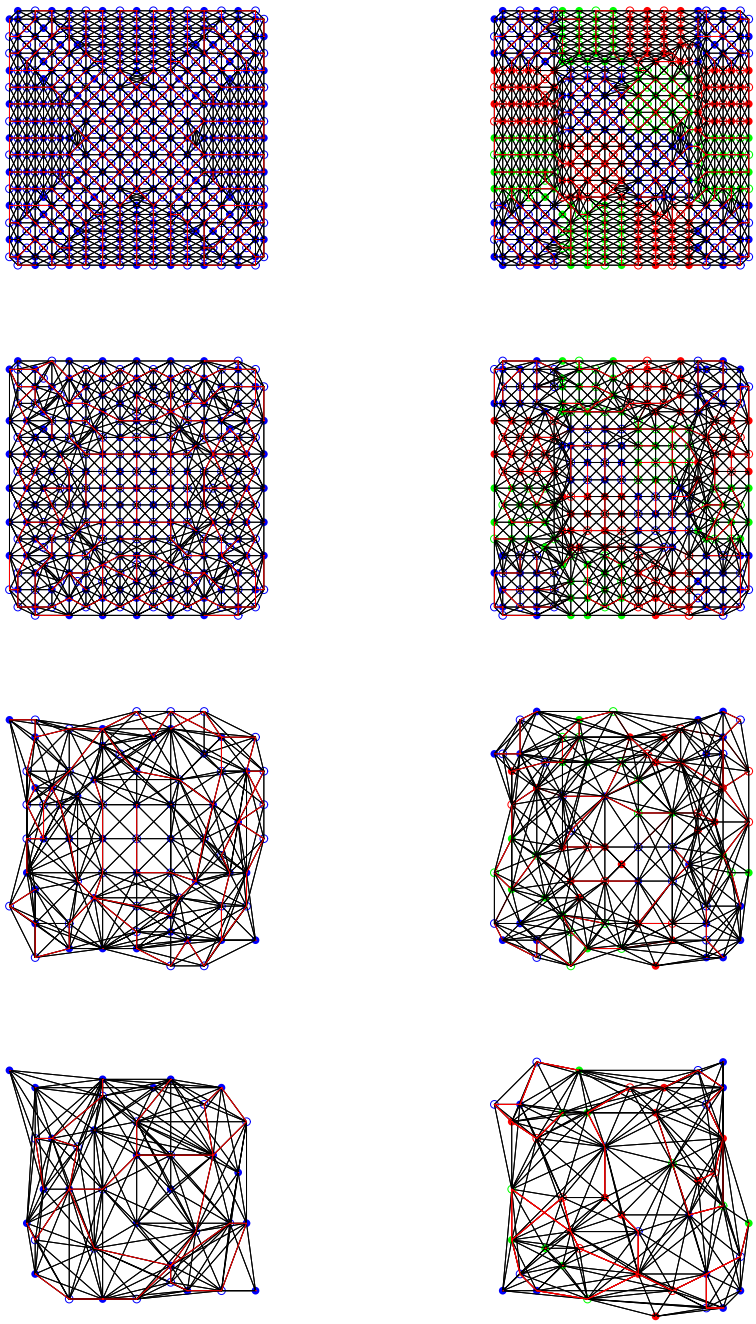


Abbildung 6.41: Kreisströmung; $PEC = 1$; Startgitterweite 1/32; Matrixgraphen der Level $-1, -2, -4, -5$ (von oben nach unten) für serielle (linke Spalte) und parallele Berechnung (4×4 Prozessorkonfiguration; rechts)

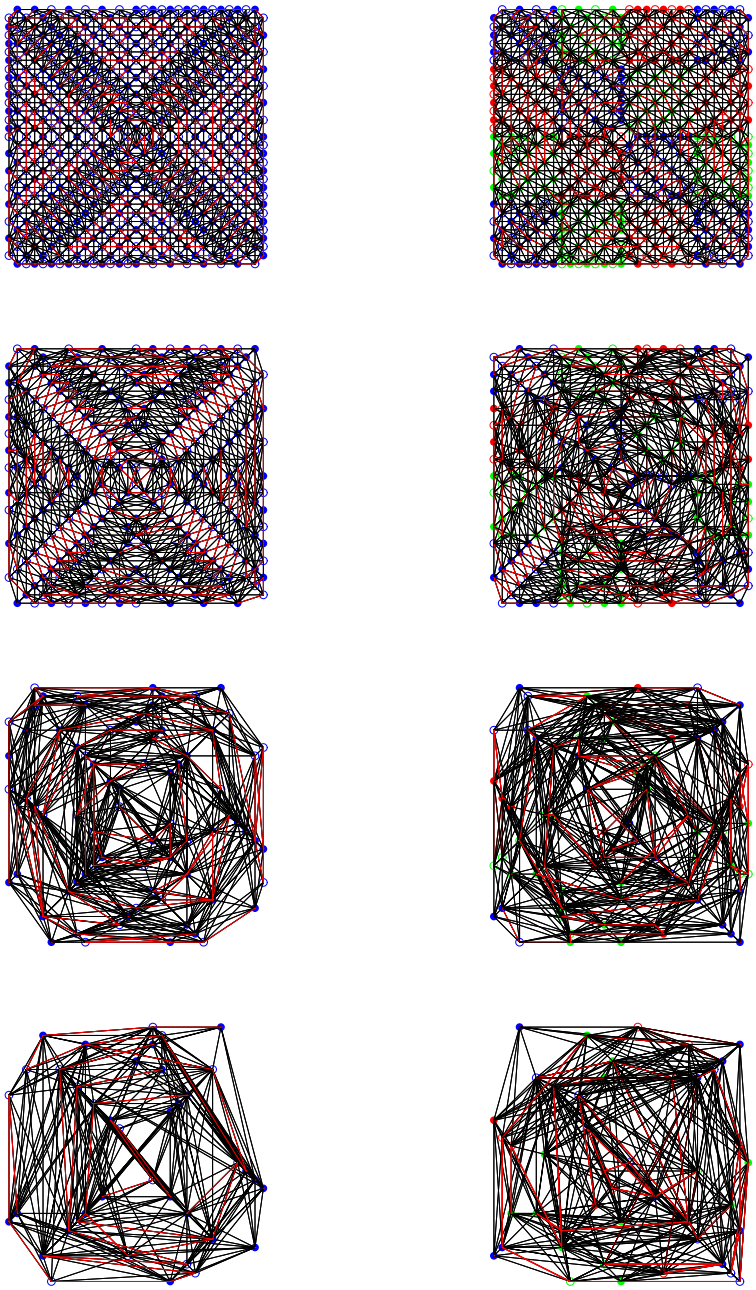


Abbildung 6.42: Kreisströmung; $PEC = 10^6$; Startgitterweite 1/32; Matrixgraphen der Level $-1, -2, -4, -5$ (von oben nach unten) für serielle (linke Spalte) und parallele Berechnung (4×4 Prozessorkonfiguration; rechts)

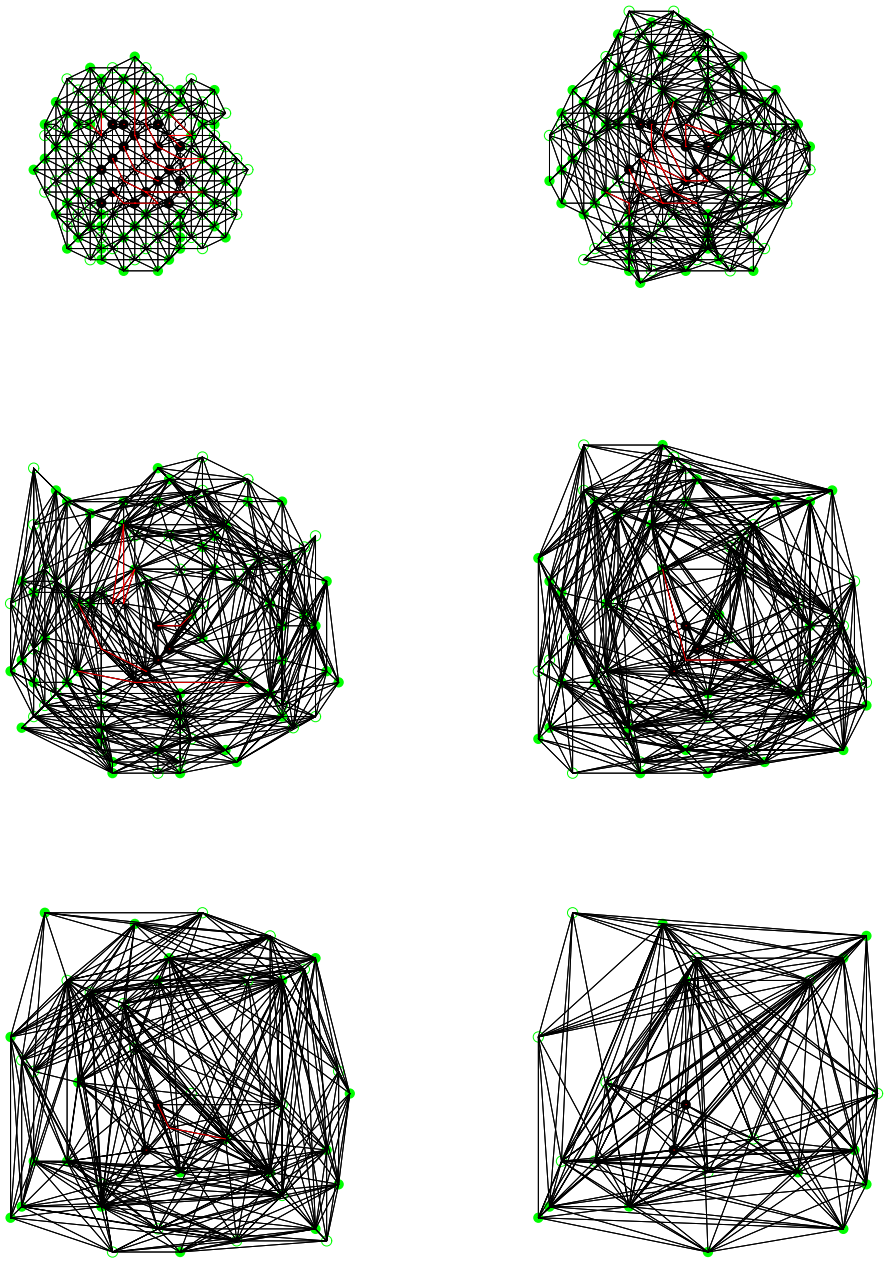


Abbildung 6.43: Matrixgraphen der Level -1 bis -6 (links oben bis rechts unten); Kreisströmung; $PEC = 10^6$; 4×4, PE 5

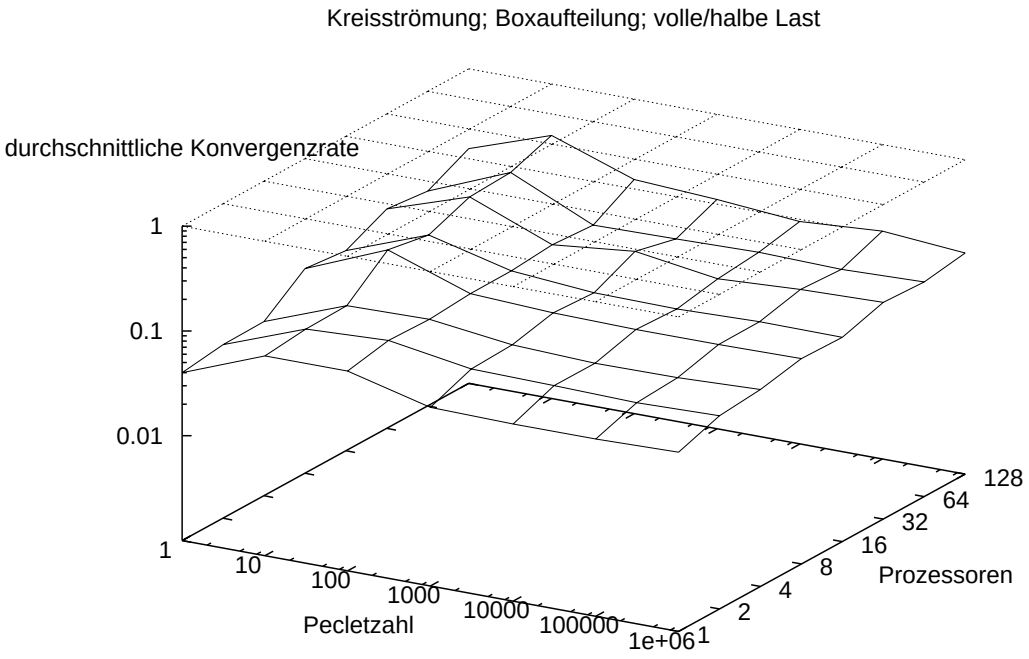


Abbildung 6.44: Kreisströmung; Boxaufteilung; volle bzw. halbe Last; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl und der Prozessorzahl

PE	h 1/128 N 16 641	1/256 66 049	1/512 263 169	1/1024 1 050 625	1/2048 4 198 401	1/4096 16 785 409
1×1	5 0.018	5 0.051				
2×1	5 0.019	6 0.053	6 0.070			
2×2	5 0.020	6 0.056	6 0.076			
4×2	5 0.021	6 0.055	6 0.072	6 0.091		
4×4	5 0.022	6 0.054	6 0.070	6 0.089		
8×4	5 0.023	6 0.053	6 0.069	6 0.087	6 0.117	
8×8	5 0.022	6 0.059	6 0.070	6 0.095	6 0.112	
16×8	5 0.024	6 0.052	6 0.067	6 0.083	6 0.134	6 0.129

Tabelle 6.50: Kreisströmung, $PEC = 10^6$; Konvergenzgüte (Anzahl Iterationen, durchschnittliche Konvergenzrate) in Abhängigkeit der Gitterweite und der Prozessorzahl (Boxaufteilung)

Als nächstes betrachten wir den Speicherbedarf. Die Tabellen 6.51 und 6.52 geben für zwei extreme Pecletzahlen die Kennzahlen dafür in einer Scaleup Betrachtung wieder. Die Gitterkomplexität c_G ist mit 2,0 für schwache Strömungen gut und für

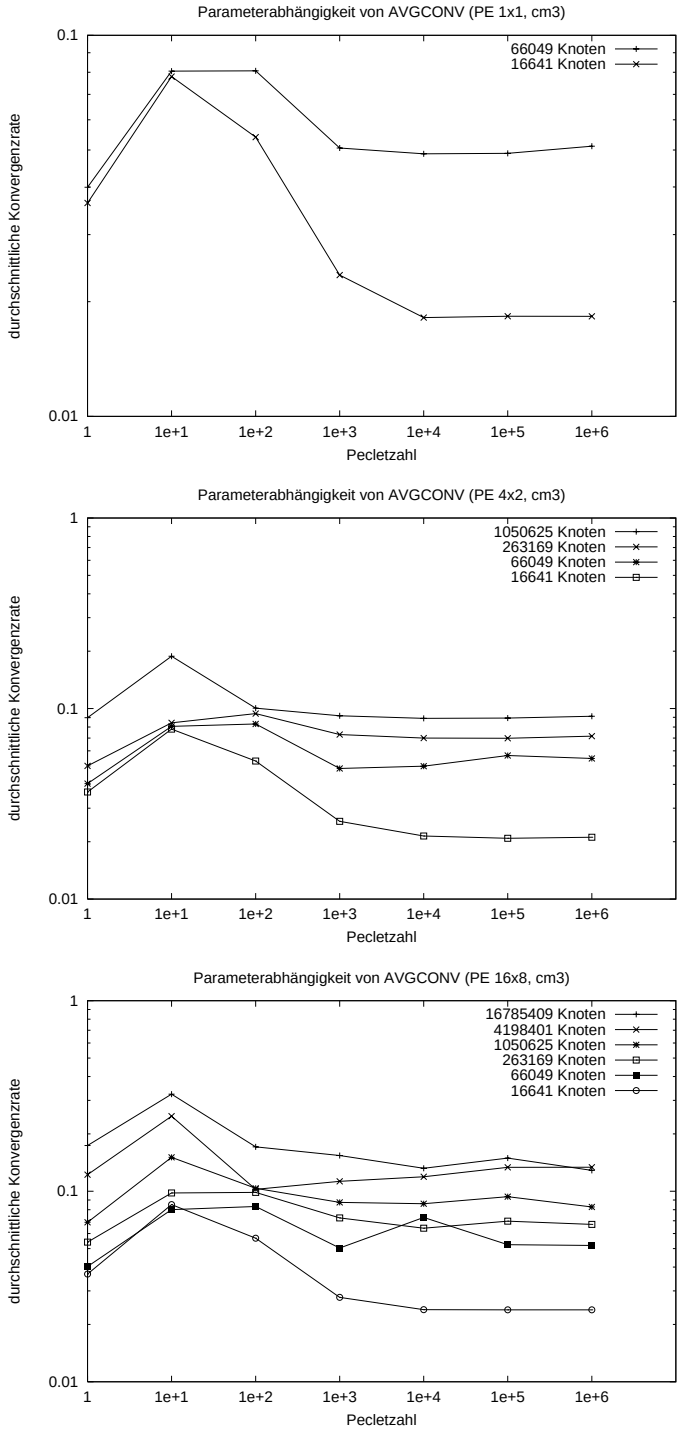


Abbildung 6.45: Kreisströmung; Abhängigkeit der mittleren Konvergenzrate von der Pecletzahl; verschiedene Knotenzahlen zu den Prozessoraufteilungen 1×1, 4×2 und 16×8

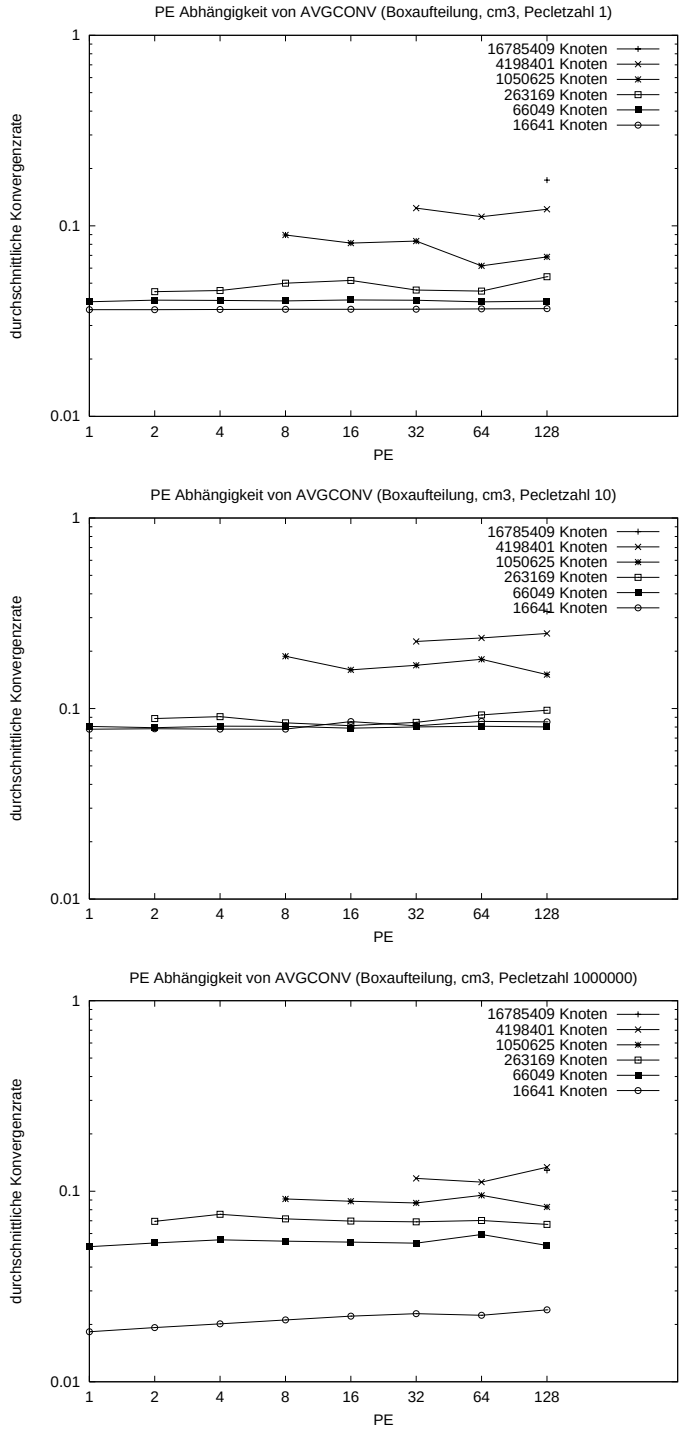


Abbildung 6.46: Kreisströmung; Boxaufteilung; Abhängigkeit der mittleren Konvergenzrate von der Prozessorzahl; verschiedene Gitterweiten zu den Pecletzahlen 1, 10, 10^6

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.11	1.99	3.84	0.02
8	2.16	2.01	4.54	0.04
32	2.17	2.01	4.67	0.05
128	2.15	2.00	4.66	0.05
	32 768 Knoten/PE			
2	2.03	1.94	3.21	0.03
8	2.16	1.99	3.94	0.06
32	2.23	2.01	4.56	0.09
128	2.22	2.01	4.62	0.10

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.02	1.94	0.00	0.00
4	2.12	1.99	3.82	0.03
16	2.19	2.01	4.54	0.05
64	2.20	2.01	4.69	0.06
	16 384 Knoten/PE			
1	1.77	1.74	0.00	0.00
4	2.04	1.94	3.20	0.05
16	2.24	2.01	4.29	0.10
64	2.28	2.01	4.60	0.13

Tabelle 6.51: Kreisströmung, $PEC = 1$; Boxaufteilung; Speicherkomplexität für volle bis achtel Last (Scaleup)

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	131 072 Knoten/PE			
2	2.60	1.91	7.24	0.03
8	2.62	1.79	8.01	0.07
32	2.65	1.75	8.22	0.09
128	2.66	1.72	8.41	0.11
	32 768 Knoten/PE			
2	2.45	1.87	4.86	0.04
8	2.65	1.77	6.54	0.12
32	2.83	1.74	7.45	0.17
128	2.96	1.72	8.09	0.20

PE	c_A	c_G	$c_{\ddot{U}}$	B/M
	65 536 Knoten/PE			
1	2.45	1.93	0.00	0.00
4	2.58	1.83	7.22	0.06
16	2.69	1.77	7.68	0.10
64	2.74	1.73	8.12	0.13
	16 384 Knoten/PE			
1	2.00	1.74	0.00	0.00
4	2.44	1.80	4.87	0.08
16	2.76	1.76	6.36	0.17
64	3.02	1.73	7.48	0.24

Tabelle 6.52: Kreisströmung, $PEC = 10^6$; Boxaufteilung; Speicherkomplexität für volle bis achtel Last (Scaleup)

starke Strömungen mit 1,7 sogar sehr gut, d. h. die Halbvergröberungsstrategie kann für alle Fälle erfolgreich durchgeführt werden. Die Überlappkomplexität $c_{\ddot{U}}$ liegt nur leicht über der der horizontalen Strömung mit Boxaufteilung (vergleiche Tabellen 6.33 und 6.34). Die Werte sind unbefriedigend; die Gründe hierfür wurden im vorigen Abschnitt analysiert. Die Operatorkomlexität c_A, die im wesentlichen den Rechenaufwand bestimmt, ist für schwache Kreisströmungen wie bei der horizontalen Strömung sehr gut. Nur für starke Kreisströmungen liegt der Wert mit 2,7 bis 3,0 unangenehm hoch. Ursache dafür ist das Anwachsen der durchschnittlichen Größe des Matrixsterns von ca. 9 auf ca. 13 bedingt durch ein Auffächern der Matrixeinträge quer zur Strömungsrichtung, wie in Abbildung 6.43 zu erkennen. Dieses Verhalten wurde auch für das serielle Verfahren beobachtet [Wag00a]. Einen weiteren Niederschlag finden die dichteren Matrizen in der Tatsache, daß für eine starke Kreisströmung etwas mehr Borderknoten angelegt werden (Verhältnis B/M).

Es ist somit für starke Kreisströmungen eine höhere Speicher– und damit Zeitbelastung gegenüber der einfachen horizontalen Strömung zu erwarten.

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	58.76	2.73	16.37	81.26	1.00	1.00	1.00	1.00	57.35	1.00	1.00
2	36.12	1.49	8.92	50.81	0.81	0.92	0.92	0.80	33.34	0.86	0.87
4	20.45	0.85	5.11	31.39	0.72	0.80	0.80	0.65	16.98	0.84	0.83
8	14.01	0.54	3.22	22.78	0.52	0.64	0.64	0.45	10.09	0.71	0.69
16	9.68	0.36	2.13	18.38	0.38	0.48	0.48	0.28	6.22	0.58	0.55
32	6.87	0.45	2.70	15.79	0.27	0.19	0.19	0.16	4.00	0.45	0.34
64	5.25	0.23	1.38	12.98	0.17	0.19	0.19	0.10	2.66	0.34	0.29
128	5.38	0.37	2.23	13.03	0.09	0.06	0.06	0.05	2.24	0.20	0.13
	insgesamt 4 198 401 Knoten										
32	255.24	6.56	59.02	323.10	1.00	1.00	1.00	1.00	167.64	1.00	1.00
64	151.09	3.48	31.31	191.19	0.84	0.94	0.94	0.84	91.73	0.91	0.92
128	99.14	2.06	18.52	126.57	0.64	0.80	0.80	0.64	53.76	0.78	0.78

Tabelle 6.53: Kreisströmung, $PEC = 1$, Boxaufteilung; Speedup für insgesamt 64049 und 4 198 401 Knoten

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	$\bar{E}_{FA}$	$\bar{E}_{IT}$	$\bar{E}_{SOL}$	$\bar{E}_{Ges}$	T^*_{FA}	$\bar{E}^*_{FA}$	$\bar{E}^*_{Ges}$
	insgesamt 66 049 Knoten										
1	51.47	3.01	15.04	82.52	1.00	1.00	1.00	1.00	49.87	1.00	1.00
2	31.77	1.66	9.95	55.79	0.81	0.91	0.76	0.74	27.45	0.91	0.87
4	19.78	0.94	5.63	36.22	0.65	0.80	0.67	0.57	15.08	0.83	0.78
8	14.90	0.62	3.70	28.81	0.43	0.61	0.51	0.36	9.61	0.65	0.61
16	11.10	0.42	2.53	23.70	0.29	0.45	0.37	0.22	6.20	0.50	0.47
32	8.77	0.31	1.85	21.74	0.18	0.30	0.25	0.12	4.59	0.34	0.31
64	7.07	0.42	2.55	22.66	0.11	0.11	0.09	0.06	3.16	0.25	0.18
128	8.13	0.28	1.70	26.23	0.05	0.08	0.07	0.02	3.06	0.13	0.11
	insgesamt 4 198 401 Knoten										
32	323.91	7.74	46.46	376.08	1.00	1.00	1.00	1.00	201.46	1.00	1.00
64	197.35	4.20	25.21	227.87	0.82	0.92	0.92	0.83	117.49	0.86	0.87
128	167.73	2.81	16.88	191.56	0.48	0.69	0.69	0.49	92.84	0.54	0.56

Tabelle 6.54: Kreisströmung, $PEC = 10^6$, Boxaufteilung; Speedup für insgesamt 64 049 und 4 198 401 Knoten

Tabelle 6.53 und 6.54 zeigen das Speedup Verhalten. Wie bisher skaliert die Zeit gut, solange jeder Prozessor wenigstens zu einigen Prozent ausgelastet ist. Auch von 32 auf 128 Prozessoren ist eine Effizienz von 50% bis 75% sehr gut.

Die Laufzeiten in Scaleup Betrachtung sind in den Tabellen 6.55 und 6.56 wiedergegeben. Für nicht zu starke Kreisströmungen sind die Effizienzen ebenso gut wie bei den bisher betrachteten Testfällen. Dieses Ergebnis ist sehr erstaunlich, gab es für diese Gleichung bisher doch überhaupt kein effizientes paralleles Verfahren. Für starke Kreisströmungen wurde der dichter werdende Matrixgraph und der sehr stark in Strömungsrichtung wachsende Überlapp als Belastungsfaktoren festgestellt. Dementsprechend leidet die Effizienz spürbar. Die Werte von unter 55% für 128 Prozessoren

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	141.25	5.97	35.81	189.58	1.00	1.00	1.00	1.00	131.30	1.00	1.00
8	202.36	6.22	49.75	259.19	0.70	0.96	0.72	0.73	147.60	0.89	0.85
32	255.24	6.56	59.02	323.10	0.55	0.91	0.61	0.59	167.64	0.78	0.74
128	307.31	6.77	74.51	391.37	0.46	0.88	0.48	0.48	188.37	0.70	0.64
	32 768 Knoten pro Prozessor										
2	36.12	1.49	8.92	50.81	1.00	1.00	1.00	1.00	33.34	1.00	1.00
8	52.94	1.74	12.21	77.07	0.68	0.85	0.73	0.66	38.32	0.87	0.84
32	78.20	1.99	15.90	101.16	0.46	0.75	0.56	0.50	46.66	0.71	0.68
128	99.14	2.06	18.52	126.57	0.36	0.72	0.48	0.40	53.76	0.62	0.58
	65 536 Knoten pro Prozessor										
1	58.76	2.73	16.37	81.26	1.00	1.00	1.00	1.00	57.35	1.00	1.00
4	80.64	3.10	18.63	109.54	0.73	0.88	0.88	0.74	68.84	0.83	0.84
16	126.03	3.46	27.71	161.36	0.47	0.79	0.59	0.50	84.24	0.68	0.66
64	151.09	3.48	31.31	191.19	0.39	0.78	0.52	0.42	91.73	0.63	0.60
	16 384 Knoten pro Prozessor										
1	11.34	0.59	3.55	18.42	1.00	1.00	1.00	1.00	10.77	1.00	1.00
4	20.45	0.85	5.11	31.39	0.55	0.70	0.70	0.59	16.98	0.63	0.65
16	37.10	0.97	6.78	48.73	0.31	0.61	0.52	0.38	23.10	0.47	0.48
64	48.68	1.38	9.65	64.81	0.23	0.43	0.37	0.28	26.79	0.40	0.39

Tabelle 6.55: Kreisströmung, $PEC = 1$, Boxaufteilung; Scaleup für volle bis achtel Last

PE	T_{FA}	T_{IT}	T_{SOL}	T_{Ges}	E_{FA}	E_{IT}	E_{SOL}	E_{Ges}	T^*_{FA}	E^*_{FA}	E^*_{Ges}
	131 072 Knoten pro Prozessor										
2	149.56	6.55	39.28	199.40	1.00	1.00	1.00	1.00	134.11	1.00	1.00
8	237.04	7.19	43.12	288.34	0.63	0.91	0.91	0.69	165.33	0.81	0.83
32	323.91	7.74	46.46	376.08	0.46	0.85	0.85	0.53	201.46	0.67	0.70
128	475.76	8.54	51.23	532.46	0.31	0.77	0.77	0.37	265.45	0.51	0.55
	32 768 Knoten pro Prozessor										
2	31.77	1.66	9.95	55.79	1.00	1.00	1.00	1.00	27.45	1.00	1.00
8	63.44	2.04	12.24	84.23	0.50	0.81	0.81	0.66	42.87	0.64	0.68
32	99.81	2.34	14.06	121.45	0.32	0.71	0.71	0.46	59.67	0.46	0.51
128	167.73	2.81	16.88	191.56	0.19	0.59	0.59	0.29	92.84	0.30	0.34
	65 536 Knoten pro Prozessor										
1	51.47	3.01	15.04	82.52	1.00	1.00	1.00	1.00	49.87	1.00	1.00
4	89.18	3.44	20.61	119.17	0.58	0.88	0.73	0.69	71.19	0.70	0.71
16	163.20	4.30	25.80	195.47	0.32	0.70	0.58	0.42	104.89	0.48	0.50
64	197.35	4.20	25.21	227.87	0.26	0.72	0.60	0.36	117.49	0.42	0.45
	16 384 Knoten pro Prozessor										
1	8.31	0.62	3.08	14.99	1.00	1.00	1.00	1.00	7.78	1.00	1.00
4	19.78	0.94	5.63	36.22	0.42	0.66	0.55	0.41	15.08	0.52	0.52
16	42.54	1.40	8.41	60.47	0.20	0.44	0.37	0.25	26.23	0.30	0.31
64	65.98	1.40	8.37	81.20	0.13	0.44	0.37	0.18	37.70	0.21	0.24

Tabelle 6.56: Kreisströmung, $PEC = 10^6$, Boxaufteilung; Scaleup für volle bis achtel Last

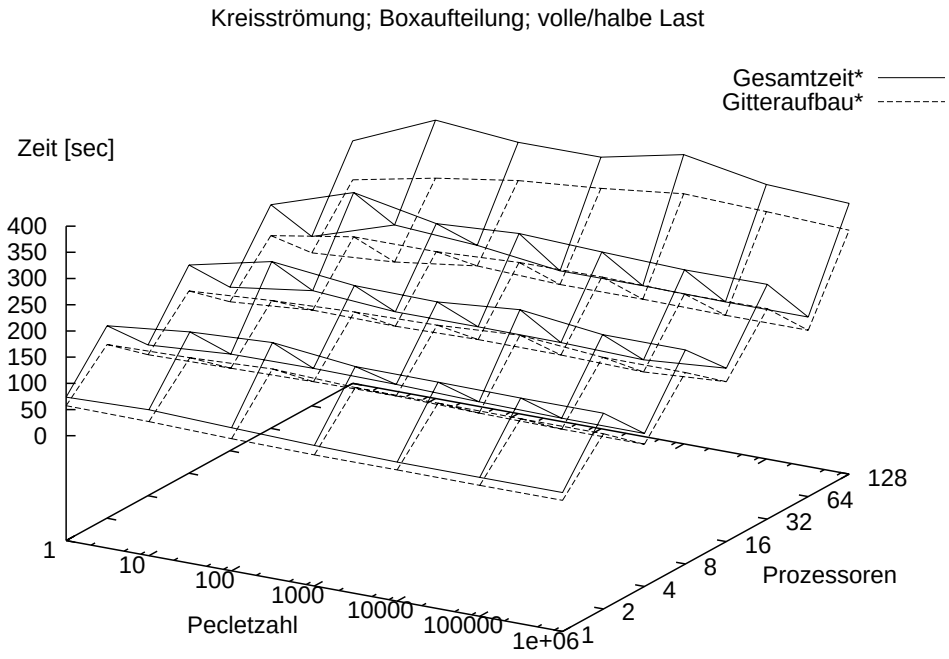


Abbildung 6.47: Kreisströmung; Boxaufteilung; volle bzw. halbe Last; Abhängigkeit der Laufzeiten von der Pecletzahl und der Prozessorzahl

sind angesichts der Schwierigkeit des Problems tolerierbar, eine Verbesserung wäre jedoch wünschenswert.

Abbildung 6.47 gibt nochmals eine Übersicht über das Zeitverhalten. Gerade für große Prozessorzahlen sieht man eine deutliche Zeitzunahme für wachsende Pecletzahlen, die aber durch etwas besser werdende Konvergenzraten teilweise kompensiert werden können. Ansonsten sind die Werte sehr gut.

Bei der horizontalen Strömung konnte schon durch eine speziell an die Gleichung angepaßte Gebietsaufteilung — damals die horizontale Streifenaufteilung — keine Verbesserung der Konvergenzgüte erzielt werden (vergleiche die Ergebnisse des vorigen Abschnitts). Daher soll hier der gleiche Versuch nicht nochmals unternommen werden. Die hier denkbare Spezialaufteilung wären konzentrische Gebietsringe. Ein solcher Ansatz wäre auch nicht für praxisrelevante Problemstellungen verallgemeinerungsfähig.

Nach den Erfahrungen bei der horizontalen Strömung ist hier von anderen Färbungsmethoden ebenfalls keine Verbesserung der sehr guten Ergebnisse zu erwarten. Die Färbungsmethoden cm2 und cm3 unterscheiden sich kaum in der Anzahl der benötigten Farben (siehe Abbildung 6.48). Erfreulich ist, daß auch für dieses schwierige Pro-

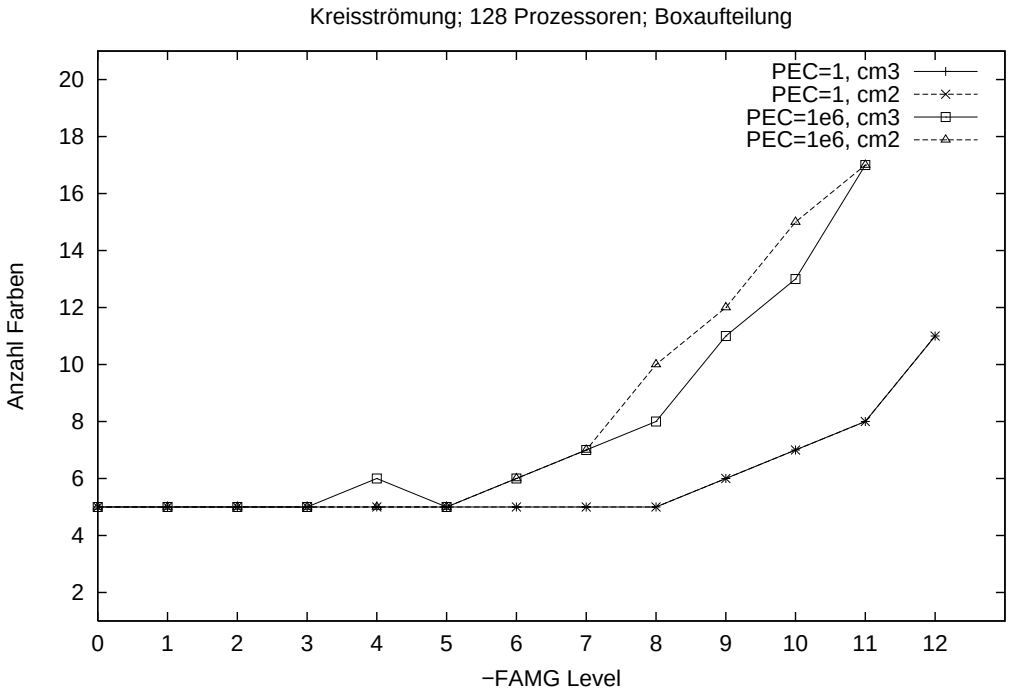


Abbildung 6.48: Kreisströmung; volle/halbe Last; Anzahl benötigter Farben in Abhängigkeit der Färbungsmethode

blem nur relativ wenige Farben benötigt werden. Abbildung 6.49 zeigt, daß die Interfacelänge vollkommen unabhängig von der Färbungsmethode ist. In Abbildung 6.50 ist oben zu sehen, daß die Konvergenzrate kaum von der Färbungsmethode abhängt und sich die Gitteraufbauzeit (untere Graphik), wie von der horizontalen Strömung bekannt, ändert. Die Experimente bestätigen die Erwartung: cm2 und cm3 sind gleich gut, cm1 bringt keine numerischen Vorteile und wegen der wesentlich schlechteren parallelen Skalierbarkeit sollte diese Methode nicht verwendet werden.

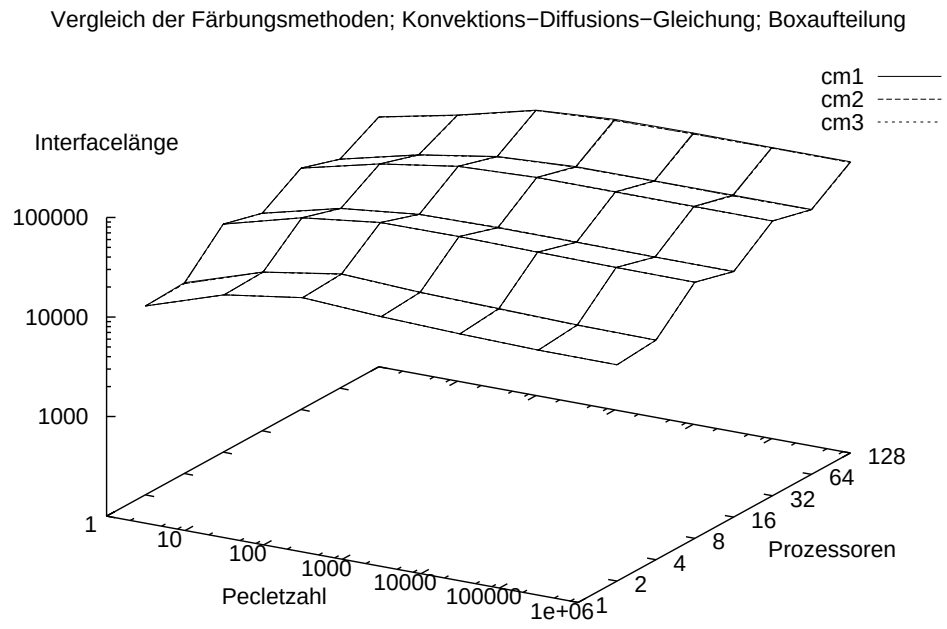


Abbildung 6.49: Kreisströmung; Boxaufteilung; volle/halbe Last; Abhängigkeit der Interfacelänge von der Färbungsmethode

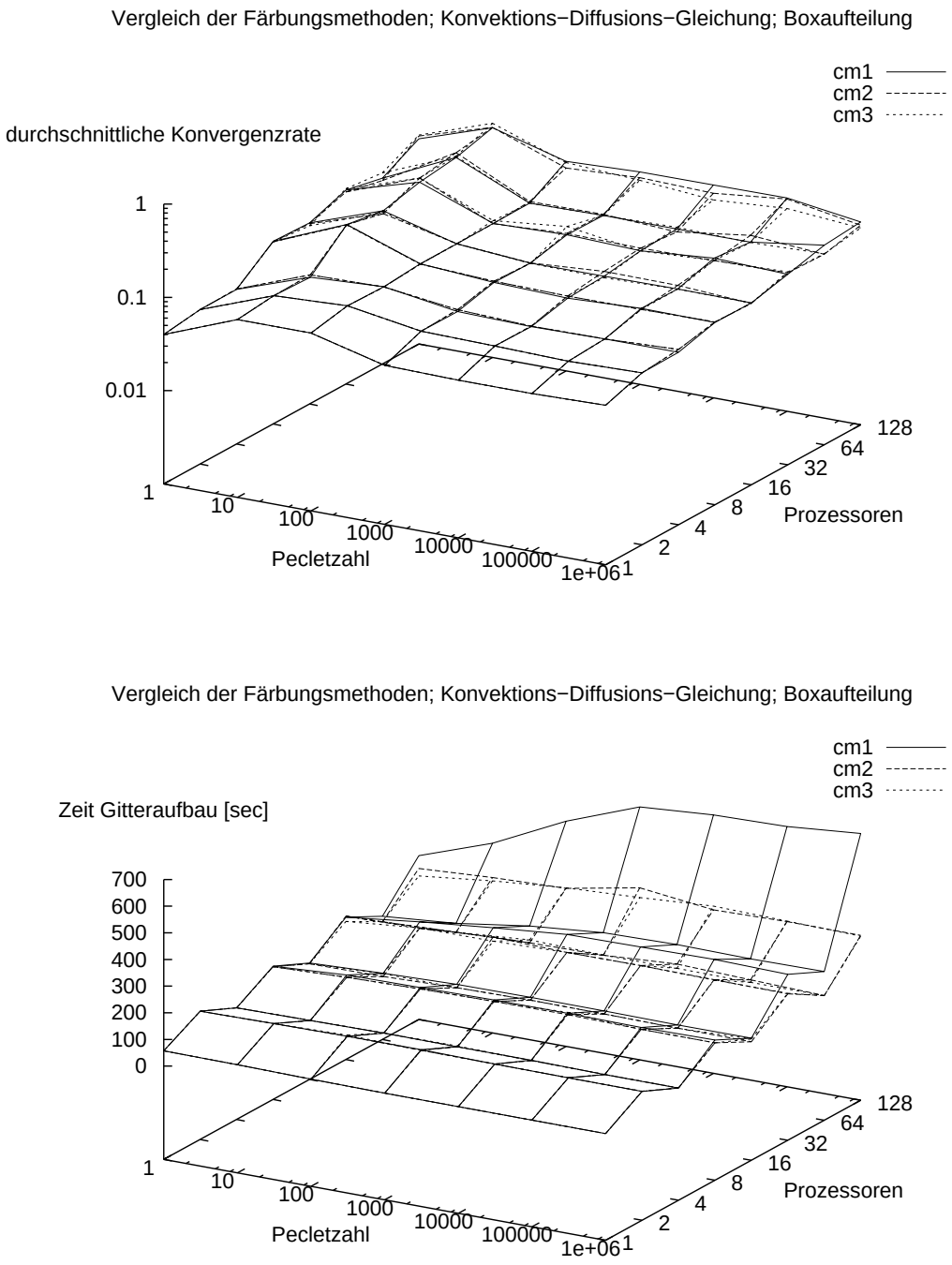


Abbildung 6.50: Kreisströmung; Boxaufteilung; volle/halbe Last; Abhängigkeit der durchschnittlichen Konvergenzrate und Gitteraufbauzeit T^*_{FA} von der Färbungsmethode

7 Zusammenfassung und Ausblick

Ziel dieser Arbeit war die Entwicklung eines parallelen Lösungsverfahrens für partielle Differentialgleichungen, das für eine möglichst große Klasse von Anwendungsproblemen effizient und robust ist. Zu Beginn der Arbeit waren einige serielle Verfahren zur robusten Lösung der anisotropen Poissongleichung und der Konvektions–Diffusions–Gleichung bekannt. Aber die meisten lassen sich prinzipiell nicht oder nur sehr ineffizient parallelisieren. Um dem Bedarf nach parallelen und robusten Lösungsverfahren gerecht zu werden, wurde das FAMG ausgewählt, weil dessen Parallelisierung aussichtsreich erschien (Abschnitt 3.1). Das Ziel, eine effiziente Parallelisierung zu entwickeln, die die Robustheit für die beiden angegebenen Klassen von Differentialgleichungen erhält, konnte in dieser Arbeit vollständig erreicht werden.

Nachdem die wichtigsten Grundlagen zusammengestellt waren, konnte das parallele FAMG schrittweise entwickelt werden. Zunächst mußte die parallele lineare Algebra erweitert werden (Abschnitte 4.1 und 5.3.2). Uns ist damit gelungen, *alle* auftretenden Matrixtypen einheitlich darzustellen und zu bearbeiten. Erstmals liegen nun auch Beweise für die Korrektheit der Operationen vor. Der Überlapp als ein zentraler Begriff wurde spezifiziert und eine Anforderungsanalyse für seine Größe durchgeführt (Abschnitt 5.3). Für die Erstellung zusätzlicher und das Entfernen nicht mehr benötigter Knotenkopien wurden verteilt arbeitende Algorithmen entwickelt; das gelang nur durch Korrektheitsbeweise.

Neben der Konzeption des parallelen Gesamtablaufs war die Parallelisierung des Markierungsalgorithmus die wesentliche parallele Zusatzkomponente, die es zu entwickeln galt (Abschnitt 5.5). Dazu wurde eine Matrix von Varianten systematisch entworfen und diskutiert. Damit gelang es, einen allgemeinen Rahmen für die Parallelisierung von algebraischen Mehrgitterverfahren nach der Selektionsmethode zu entwerfen und zu realisieren. Das Kernstück des seriellen FAMG Verfahrens — die Bestimmung der guten Elternpaare — konnte so im Sinne des SPMD Modells vollkommen unverändert übernommen werden.

Als eine Teilaufgabe wurde ein Graphfärbungsproblem extrahiert (Abschnitt 5.6). Die Konstruktion des Färbungsgraphen erforderte eine Umformulierung des ursprünglichen Prädikats, so daß es lokal entscheidbar geworden ist. Die Äquivalenz beider Prädikate sicherten wir durch Beweise ab. Der eigentliche Färbungsalgorithmus konnte unter Zuhilfenahme anderer Arbeiten vollkommen verteilt arbeitend formuliert werden. Die erweiterte parallele lineare Algebra ermöglichte die Beschreibung der parallelen Darstellungsform der Gittertransfermatrizen und die Ableitung der korrekten

Algorithmen zu ihrer Anwendung (Abschnitt 5.7), wobei charakteristische Unterschiede zwischen der in *UG* bestehenden elementbasierten und der hier benötigten knotenbasierten Datenaufteilung sichtbar wurden. Schließlich mußte bei der parallelen Galerkin–Assemblierung sichergestellt werden, daß jede Matrixeintragskomponente genau einmal im Ergebnis erscheint (Abschnitt 5.8).

Damit gelang es, den gegebenen seriellen FAMG Algorithmus so zu parallelisieren, daß möglichst wenig Abstriche von seiner numerischen Güte hingenommen werden mußten. Daß sich dieser Aufwand gelohnt hat, zeigten die durchgeführten und nachfolgend zusammengefaßten numerischen Experimente.

Als Modellprobleme wurden in Kapitel 6 die anisotrope Poissongleichung auf einem strukturierten Ausgangsgitter, der Laplace Operator auf einem unstrukturierten Gitter auf dem Gebiet des Wolfgangsees und die Konvektions–Diffusions–Gleichung auf einem strukturierten Ausgangsgitter sowohl mit einer horizontalen Strömung als auch mit einer Kreisströmung getestet. Alle Beispiele zeigten robustes Verhalten und sehr gute parallele Skalierbarkeit. Als entscheidende Einflußfaktoren wurden die Interfacelänge und die Dichte des Prozessor–Nachbarschaftsgraphen erkannt. Deshalb sollte man eine Gebietsaufteilung mit geringer Interfacelänge und eine Graphfärbungsmethode, die wenige Farben erzeugt, verwenden. Da die Parallelisierung strikt auf der algebraischen Ebene angesiedelt ist, können Differentialgleichungen auch in 3 Raumdimensionen parallel gelöst werden. Schon das serielle FAMG benötigte für den Gitteraufbau länger als für die eigentliche Lösungsphase. Dieses Mißverhältnis verstärkte sich durch die Parallelisierung leider nochmals.

Die auf Praxiserfahrungen beruhende Arbeitshypothese, daß durch eine Streifenaufteilung des Rechengebiets und/oder eine strukturiertere Abarbeitung der Prozessorränder im Markierungsalgorithmus eine bessere Effizienz erzielbar wäre, hat sich nicht bestätigt. Man braucht also keine problemangepaßten Speziallösungen in Betracht zu ziehen, deren Verallgemeinerungsfähigkeit darüber hinaus fraglich wäre. Sollten in der Praxis Konvergenzprobleme auftreten, ist der Einsatz von Krylovraummethoden (wie z. B. Bi–CGSTAB) zur Beschleunigung möglich und erfolgversprechend.

Die Robustheit und parallele Skalierbarkeit dieses neuen Verfahrens ist an Hand der angeführten Modellprobleme erwiesen und es ist daher sinnvoll, die Anwendbarkeit des Verfahrens auf weitere Gleichungsklassen auszudehnen.

Die Übertragung auf Systeme von partiellen Differentialgleichungen ist ohne konzeptionelle Erweiterungen möglich. Es müssen nur noch einzelne Stellen im Programmcode angepaßt werden, weil das serielle Verfahren schon für Differentialgleichungssysteme erweitert und implementiert worden ist [Wag00a]. Der Einfluß verschiedener Markierungsstrategien in Abhängigkeit von der Differentialgleichung sollte systematisch untersucht werden. Dringend erforderlich wäre ein Ausweiten der Funktionalität des Moduls DDD, wie es in Abschnitt 6.4.1 dargestellt wurde. Dazu sind aber erhebliche konzeptionelle Erweiterungen nötig.

Unabhängig davon brächte eine Zwischenagglomeration während des Gitteraufbaus eine Beschleunigung: Wenn auf gröberen Leveln die Auslastung der Prozessoren zu stark sinkt, sollten die Knoten einiger benachbarter Prozessoren jeweils auf einem Prozessor zusammengefaßt werden. Dadurch entfallen die ehemals zwischen ihnen existierenden Prozessorinterfaces, der Speicher– und Kommunikationsaufwand nimmt ab, dafür reduziert sich aber auch der Parallelisierungsgrad. Es sind Heuristiken zu entwickeln, die den widerstrebenden Anforderungen gerecht werden. Elementbasierte Lastumverteilungsverfahren für parallele Mehrgitteralgorithmen sind in [Bas96b] und [Lan00] entwickelt worden und können auf den knotenweisen Verteilungsansatz übertragen werden. Dazu müssen Ghostknoten in die hier weiterentwickelte parallele lineare Algebra entsprechend [Wie00a] eingeführt werden.

Eine weitere Beschleunigung könnte eventuell durch ein vorsichtiges Weglassen von Kommunikation erzielt werden. Dadurch weicht der Ablauf weiter vom ursprünglich seriellen ab. Jeder solche Schritt ist aber heuristisch und muß durch umfangreiche Experimente in seiner Gesamtwirkung abgesichert werden. Denkbar wäre, den Überlapp der Tiefe 2 nur für starke Matrixeinträge zu fordern, weil kleine Einträge bei der Elternauswahl keine Rolle spielen, wohl aber in anderen Algorithmusteilen zu Verschlechterungen führen können. In [Stü99a] werden Techniken beschrieben, sehr kleine Einträge in Gittertransfermatrizen zu streichen, um so weniger dichte Grobgittermatrizen zu erhalten. Eigene Versuche haben für unser Verfahren keinen großen Erfolg gezeigt, könnten aber noch ausführlicher untersucht werden. Für Selektionsverfahren wurden noch weitere Verbesserungsmaßnahmen entwickelt [Stü99a]. Ihre Anwendbarkeit auf unser Verfahren sollte geprüft werden.

Für die Anwendung auf nichtlineare und/oder zeitabhängige partielle Differentialgleichungen sind sehr viele, aber ähnliche lineare Gleichungssysteme zu lösen. Da bei uns die Gitteraufbauphase sehr zeitaufwendig ist, könnten Techniken übertragen werden, die für Newtonverfahren wegen des aufwendigen Aufbaus der Jacobimatrix entwickelt worden sind [Sto89]. Dabei kann die Jacobimatrix (bei uns die Gitterhierarchie) einige Schritte lang unverändert verwendet werden, ehe Indikatoren anzeigen, daß ein Neuaufbau effizienter wäre.

Es sind zwei Stufen von Verbesserungen denkbar. Zum einen kann man die Gitterhierarchie und –struktur unverändert lassen, aber die Steifigkeitsmatrixeinträge aller Level durch Galerkin–Assemblierung in jedem Schritt neu berechnen. Etwas aufwendiger ist das Vorgehen, in jedem Schritt auch die Einträge der Gittertransfermatrizen neu zu berechnen. Es müssen aber nicht mehr alle möglichen Nachbarknotenpaare neu bewertet werden, sondern es genügt, die Minimierungsaufgabe (3.13) für das eine bereits existierende Elternpaar erneut zu lösen. In allen Fällen bleibt die zeitaufwendige Strukturänderung des verteilten Matrixgraphen erspart.

Zusammenfassend betrachtet erscheint die Erweiterung des hier entwickelten parallelen FAMG auf größere Anwendungsklassen machbar. Damit steht ein effizientes und robustes paralleles Lösungsverfahren mit einer breiten Anwendbarkeit zur Verfügung.

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Allgemein können einem Objekt folgende Attribute hinzugefügt werden:

$$ {}^{\ell}_{t}X^{p}_{i}, $$

wobei ℓ der Gitterlevel, t die Tiefe des Überlapps, $p \in \mathcal{P}$ die Prozessornummer und $i \in J$ der Index ist.

Für Symbole, die in dieser Arbeit definiert werden, ist die Seitennummer ihrer Definition soweit möglich angegeben. Die Reihenfolge der Auflistung orientiert sich an der Aussprache der Symbole.

A	(globale) Matrix; speziell Steifigkeitsmatrix	
$\hat{A}$	additiv (oder inkonsistent) verteilte Matrix	37
$\bar{A}$	konsistent verteilte Matrix	37
$\tilde{A}$	teilweise konsistent verteilte Matrix	37
${}_{t}\ddot{A}$	t–konsistent verteilte Matrix	65
α	Aufteilungsabb. Knoten $\mapsto$ Prozessoren mit Knotenkopie	32
$\forall$	Allquantor	
$\mathcal{B}$	Menge der Borderknoten	32
B/M	Verhältnis Border- zu Masterknoten	104
$\square$	Ende des Beweises	
$\mathcal{C}$	Menge der grobmarkierten Knoten (C–Knoten)	23
c_A	Operatorkomplexität	103
c_G	Gitterkomplexität	104
$c_{\ddot{U}}$	Überlappkomplexität	104
$\mathcal{D}$	Liste der noch zu verarbeitenden Knoten von $\mathcal{L}$	28
$\mathrm{diag}(A)$	Diagonalmatrix von A	
δ_{ij}	Kroneckersymbol	
E_X	Scaleup Effizienz für Komponente X	105
$E^*{}_X$	Scaleup Effizienz für Komponente X nach Modell (6.8)	119
$\bar{E}_X$	Speedup Effizienz für Komponente X	105
ε	Parameter in der Differentialgleichung	129

$\mathcal{E}$	Kantenmenge eines Graphen	36
$\exists$	Existenzquantor	
$\exists_1$	es existiert *genau* ein	
f	rechte Seite der Gleichung	5
$\mathcal{F}$	Menge der feinmarkierten Knoten (F–Knoten)	23
$\mathcal{G}, \mathcal{G}(A)$	Graph, Matrixgraph	36
i, j	Knoten; Elemente der Indexmenge	32
I	Einheitsmatrix	
$\mathcal{I}$	Masterknoten im Interface eines Prozessors	63
J	globale Indexmenge	32
J^p	lokale Indexmenge auf Prozessor p	32
k	durchschnittliche Konvergenzrate	7
$\mathcal{L}$	Liste zu verarbeitender Knoten	28
$\mathcal{M}$	Menge der Masterknoten, Kernpartition	32
$M_{p,q}$	Nachrichtenpuffer von Prozessor p an q	
μ	Abb. Knoten $\mapsto$ Prozessor für Masterinstanz	32
$\mathcal{N}_A^p(j)$	Nachbarschaft um Knoten j in Matrix A auf PE p	36
n	Vektorlänge	33
n_j	Anzahl der Freiheitsgrade im Knoten j	33
ω	Dämpfungsparameter eines Glätters	25
$\mathcal{P}$	Menge der Prozessoren	32
$\mathbb{P}_i$	Elternknotenmenge (–paar) des Knotens i	23
$\mathbb{P}_i^+$	Menge sehr guter Elternknotenmengen zum Knoten i	27
PEC	Gitterpecletzahl	154
$\mathfrak{P}$	Potenzmenge	
${}^\ell P$	Prolongation Level $\ell - 1 \to \ell$	10
${}_\iota\psi$	Einschränkung der Aufteilungsabbildung α auf ${}_\iota\mathcal{U}$	65
$\mathbb{R}$	Körper der reellen Zahlen	
${}_1\mathcal{R}, {}_2\mathcal{R}$	Rand in Tiefe 1 bzw. 2	63
${}^\ell R$	Restriktion Level $\ell \to \ell - 1$	10
ϱ	asymptotische Konvergenzrate; Spektralradius	17
S	Speedup	105
σ	Abschneideschwelle für Matrixeinträge	25
T_X	Laufzeit für Komponente X	
${T^*}_X$	Laufzeit für Komponente X nach Modell (6.8)	119
$\mathcal{T}$	Liste unabhängiger Prozessorteilmengen	90
θ	Toleranz für „gute Elternpaare“	27
$x^\top, A^\top$	transponiertes Objekt	
u	Lösung der Gleichung	5
${}_1\mathcal{U}, {}_2\mathcal{U}$	Überlapp der Tiefe 1 bzw. 2	63
$\dot\cup$	disjunkte Vereinigung	
$\mathcal{V}$	Knotenmenge eines Graphen	36
$w(\mathbb{P})$	Gewicht einer Elternmenge	27

x	(globaler) Vektor	35
$\hat{x}$	additiv (oder inkonsistent) verteilter Vektor	35
$\check{x}$	eindeutig (additiv) verteilter Vektor	35
$\bar{x}$	konsistent verteilter Vektor	35
${}_t\ddot{\ddot{x}}$	t–konsistent verteilter Vektor	65
$x := y$	Zuweisungs– bzw. Definitionsoperator	

Literaturverzeichnis

[ABDP81] R. E. Alcouffe, A. Brandt, J. E. Dendy, and J. W. Painter: *The multi-grid method for the diffusion equation with strongly discontinous coefficients.* SIAM J. Sci. Stat. Comput., 2:430–454, 1981.

[Amd67] G. M. Amdahl: *Validity of the single–processor approach to achieving large scale computing capabilities.* In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[Bas93] Peter Bastian: *Parallel adaptive multigrid methods.* Preprint 93 - 60, Interdisziplinäres Zentrum für wissenschaftliches Rechnen, Universität Heidelberg, October 1993.

[Bas96a] Peter Bastian: *Load Balancing for Adaptive Multigrid Methods.* Institute for Computer Applications 3, Universität Stuttgart, Report N96/1, January 1996.

[Bas96b] Peter Bastian: *Parallele adaptive Mehrgitterverfahren.* Teubner Skripten zur Numrik. Teubner–Verlag, 1996.

[Bas00] Peter Bastian: *Paralleles Rechnen I.* Vorlesungsskript, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg, März 2000.

[BBJ+97] Peter Bastian, Klaus Birken, Klaus Johannsen, Stefan Lang, Nicolas Neuss, Henrik Rentz-Reichert, and Christian Wieners: *UG - a flexible software toolbox for solving partial differential equations.* Computation and Visualization in Science, 1:27–40, 1997.

[BBJ+99] Peter Bastian, Klaus Birken, Klaus Johannsen, Stefan Lang, Volker Reichenberger, Christian Wieners, Gabriel Wittum, and Christian Wrobel: *A Parallel Software-Platform for Solving Problems of Partial Differential Equations using Unstructured Grids and Adaptive Multigrid Methods*, pages 326–339. In Krause, E. and W. Jäger [KJ99], 1999.

[BBJ+00] Peter Bastian, Klaus Birken, Klaus Johannsen, Stefan Lang, Volker Reichenberger, Christian Wieners, Gabriel Wittum, and Christan Wrobel:

Parallel Solutions of Partial Differential Equations with Adaptive Multigrid Methods on Unstructured Grids, pages 496–508. In Krause, E. and W. Jäger [KJ00], 2000.

[BDY88] R. E. Bank, T. F. Dupont, and H. Yserentant: *The hierarchical basis multigrid method.* Numer. Math., 52:427–458, 1988.

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick: *A Multigrid Tutorial.* SIAM, second edition, 2000.

[Bir97] Klaus Birken: *Dynamic Distibuted Data (DDD) — A Software Tool for Distributed Memory Parallelisation.* University of Stuttgart, 1997.

[Bir98] Klaus Birken: *Ein Modell zur effizienten Parallelisierung von Algorithmen auf komplexen, dynamischen Datenstrukturen.* Doktorarbeit, Universität Stuttgart, Rechenyentrum der Universität Stuttgart, 1998.

[BJL+01] Peter Bastian, Klaus Johannsen, Stefan Lang, Sandra Nägele, Volker Reichenberger, Christian Wieners, Gabriel Wittum, and Christan Wrobel: *Advances in High-Performance Computing: Multigrid Methods for Partial Differential Equations and its Applications.* In Krause, E. and W. Jäger [KJ01], 2001. submitted.

[BLE96] Peter Bastian, Stefan Lang, and Knut Eckstein: *Parallel Adaptive Multigrid Methods in Structural Mechanics.* Institute for Computer Applications 3, Universität Stuttgart, Report N96/2, January 1996.

[BMR82] A. Brandt, S. F. McCormick, and J. Ruge: *Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations.* Technical report, Institute for Computational Studies, POB 1852, Fort Collins, Colorado, 1982.

[BPS86] James H. Bramble, Joseph E. Pasciak, and Alfred H. Schatz: *The construction of preconditioners for elliptic problems by substructuring, I.* Mathematics of Computation, 47(175):103–134, 1986.

[BPX90] James H. Bramble, Joseph E. Pasciak, and Jinchao Xu: *Parallel multilevel preconditioners.* Mathematics of Computation, 55(191):1–22, July 1990.

[Bra73] Achi Brandt: *Multi-level adaptive technique for fast numerical solution to boundary value problems.* In *3rd International Conference on Numerical Methods in Fluid Mechanics, Paris*, volume 18 of *Lecture Notes in Physics*, pages 83–89, Berlin, 1973. Springer–Verlag.

[Brä93] Thomas Bräunl: *Parallele Programmierung.* Vieweg Verlag, 1993.

[Bra95] Dietrich Braess: *Towards algebraic multigrid methods for elliptic problems of second order.* Computing, 55:379–393, 1995.

[Bra97] Dietrich Braess: *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie.* Springer Lehrbuch. Springer–Verlag Berlin Heidelberg, zweite, überarbeitete Auflage, 1997.

[BS97] Randolph E. Bank and R. Kent Smith: *The incomplete factorization multigraph algorithm.* Technical report, Department of Mathematics, University of California at San Diego, La Jolla, CA 92093, 1997.

[Buz98] A. Buzdin: *Tangentiale Zerlegung.* Computing, 61:257–276, 1998.

[BW93] Peter Bastian and Gabriel Wittum: *On robust and adaptive multi-grid methods.* In *4th European Multigrid Conference, Amsterdam*, July 1993.

[BW97] Jürgen Bey and Gabriel Wittum: *Downwind numbering: robust multigrid for convection-diffusion problems.* Applied Numerical Mathematics, 23:177–192, 1997.

[BW99] R. E. Bank and C. Wagner: *Multilevel ILU decomposition.* Numerische Mathematik, 82:543–576, 1999.

[CM94] Tony F. Chan and Tarek P. Mathew: *Domain Decomposition Algorithms*, pages 61–144. In Iserles, A. [Ise94], 1994.

[Cop00] University of Colorado and Front Range Scientific Computations Inc.: *Sixth Copper Mountain Conference on Iterative Methods*, April 2000. virtual proceedings on http://www.mgnet.org/mgnet-cm2000.html.

[dF94] E.F. Van de Felde: *Concurrent Scientific Computing*, volume 16 of *Texts in Applied Mathematics.* Springer–Verlag, Berlin, Heidelberg, 1994.

[dZ90] P. M. de Zeeuw: *Matrix–dependent prolongations and restrictions in a blackbox multigrid solver.* Journal of Computational and Applied Mathematics, 33:1–27, 1990.

[Fed61] R. P. Fedorenko: *A relaxation method for solving elliptic difference equations (russian).* USSR Comput. Math. and Phys., 1(5):1092–1096, 1961.

[Fed64] R. P. Fedorenko: *The speed of convergence of one iterative process.* USSR Comput. Math. and Phys., 4(3):227–235, 1964.

[Fly66] Michael Flynn: *Very high–speed computing systems.* In *Proceedings of the IEEE*, volume 54, pages 1901–1909, December 1966.

[Fly72] Michael Flynn: *Some computer organizations and their effectiveness.* IEEE Trans. Computers, 21(9):948–960, 1972.

[Fro90] Andreas Frommer: *Lösung linearer Gleichungssysteme auf Parallelrechnern.* Vieweg Verlag, 1990.

[GJ79] M. R. Garey and D. S. Johnson: *Computers and Interactability.* W. H. Freeman, New York, 1979.

[GJP96] Robert K. Gjertsen Jr., Mark T. Jones, and Paul E. Plassmann: *Parallel heuristics for improved, balanced graph colorings.* Journal of Parallel and Distributed Computing, 37:171–186, 1996.

[GS95] William Gropp and Barry Smith: *Parallel Domain Decomposition Software*, chapter 6, pages 97–106. In Keyes, David E. et al. [KST95], 1995.

[Haa99a] Gundolf Haase: *A parallel AMG for overlapping and non-overlapping domain decomposition.* In *Proceedings of Copper Mountain Conference*, April 26, 1999.

[Haa99b] Gundolf Haase: *Parallele Lösungen von partiellen Differentialgleichungen.* Preprint 99-11, Mathematisches Institut, Universität Stuttgart, Juli 1999.

[Haa99c] Gundolf Haase: *Parallelisierung numerischer Algorithmen für partielle Differentialgleichungen.* Teubner Verlag, Berlin, Heidelberg, 1999.

[Hac76] Wolfgang Hackbusch: *Ein iteratives Verfahren zur schnellen Auflösung elliptischer Randwertprobleme.* Technical Report 76–12, Universität Köln, 1976.

[Hac84] Wolfgang Hackbusch: *Multigrid convergence for a singular pertubation problem.* Linear algebra and its applications, 58:125–145, 1984.

[Hac85] Wolfgang Hackbusch: *Multi–Grid Methods and Applications.* Springer–Verlag, Berlin, Heidelberg, 1985.

[Hac86] Wolfgang Hackbusch: *Theorie und Numerik elliptischer Differentialgleichungen.* Teubner Studienbücher Mathematik, Stuttgart, 1986.

[Hac89a] Wolfgang Hackbusch: *The frequency decomposition multi–grid method; Part I: Application to anisotropic equations.* Numerische Mathematik, 56:229–245, 1989.

[Hac89b] Wolfgang Hackbusch: *On first and second order box schemes.* Computing, 41:277–296, 1989.

[Hac91] Wolfgang Hackbusch: *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Teubner Studienbücher Mathematik, Stuttgart, 1991.

[Hac92] Wolfgang Hackbusch: *The frequency decomposition multi–grid method; Part II: Convergence analysis based on the additive Schwarz method.* Numerische Mathematik, 63:433–453, 1992.

[Hal98] Sabine Halder: *Frequenzfilternde Zerlegungen zur numerischen Lösung der Stokes-Gleichungen.* Diplomarbeit, Universität Stuttgart, Institut für Computeranwendungen 3, September 1998.

[Hel96] Doris Helzle: *Tangential frequenzfilternde Zerlegungen zur Lösung partieller Differentialgleichungen in drei Dimensionen.* Diplomarbeit, Universität Stuttgart, Institut für Computeranwendungen 3, Februar 1996.

[HL93] B. Hendrickson and R. Leland: *The chaco user's guide version 1.0.* Report SAND93–2339, Sandia National Laboratories, Albuquerque, NM, 1993.

[HPF93] High Performance Fortran Forum: *High Performance Fortran*, May 1993. language specification.

[HT81] Wolfgang Hackbusch and Ulrich Trottenberg (editors): *Lecture Notes in Mathematics*, volume 960. Springer–Verlag, Berlin, Heidelberg, 1981.

[Ise94] A. Iserles (editor): *Acta Numerica 1994.* Cambridge University Press, 1994.

[Joh00a] Klaus Johannsen: *A robust smoother for convection–diffusion problems with closed characteristics.* Computing, 2000. accepted.

[Joh00b] Klaus Johannsen: *Robuste Mehrgitterverfahren für die Konvektions-Diffusions-Gleichung mit wirbelbehafteter Konvektion.* Teubner-Verlag, Stuttgart, Leipzig, 2000.

[JP93] Mark T. Jones and Paul E. Plassmann: *A parallel graph coloring heuristic.* SIAM Journal on Scientific Computing, 14:654–669, 1993.

[Jun94] Dieter Jungnickel: *Graphen, Netzwerke und Algorithmen.* BI–Wissenschaftsverlag, Mannheim, dritte überarbeitete und erweiterte Auflage, 1994.

[Ket81] R. Kettler: *Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods*, pages 502–534. Volume 960 of Hackbusch, Wolfgang and Ulrich Trottenberg [HT81], 1981.

[KGGK94] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis: *Introduction to Parallel Computing — Design and Analysis of Algorithms.* The Benjamin/Cummings Publication Company, Inc, Redwood City, California, 1994.

[KJ99] E. Krause and W. Jäger (editors): *High Performance Computing in Science and Engineering '98.* Springer Verlag, 1999.

[KJ00] E. Krause and W. Jäger (editors): *High Performance Computing in Science and Engineering '99.* Springer Verlag, 2000.

[KJ01] E. Krause and W. Jäger (editors): *High Performance Computing in Science and Engineering '00.* Springer Verlag, 2001.

[KK95] G. Karypis and V. Kumar: *METIS — unstructured graph partitioning and sparse matrix ordering systems.* Technical report, University of Minnesota, Minneapolis, MN, 1995.

[KS99] Arnold Krechel and Klaus Stüben: *Parallel algebraic multigrid based on subdomain blocking.* GMD report no. 71, GMD — Forschungszentrum Informationstechnik GmbH, Schloß Birlinghoven, D–53754 Sankt Augustin, December 1999.

[KST95] David E. Keyes, Youcef Saad, and Donald G. Truhlar (editors): *Domain–Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering.* SIAM, Philadelphia, 1995.

[Lan93] Stefan Lang: *Lastverteilung für parallele, adaptive Numerische Mehrgitterberechnungen.* Diplomarbeit, Universität Erlangen–Nürnberg, Institut für Mathematische Maschinen und Datenverarbeitung, Juni 1993.

[Lan99] Stefan Lang: *UG - a parallel software tool for unstructured adaptive multigrids.* In E. H. D'Hollander, G. R. Joubert, S. J. Peters, and H. J. Sips (editors): *Parallel Computing — Fundamentals and Applications. Proceedings of ParCo '99*, pages 324–332, Delft, Netherlands, 1999.

[Lan00] Stefan Lang: *Eine parallele Software–Plattform für adaptive unstrukturierte Mehrgitterverfahren.* Doktorarbeit, Universität Heidelberg, Interdisziplinäres Zentrum für wissenschaftliches Rechnen, 2000.

[Lew91] T. Lewis: *Data parallel computing: An alternative for the 1990s.* In *IEEE Computer, Viewpoints*, volume 24, pages 110–111, September 1991.

[LMR+98] P. Luksch, U. Maier, S. Rathmayer, M. Weidmann, F. Unger, P. Bastian, V. Reichenberger, and A. Haas: *SEMPA — Software Engineering Methods for Parallel Applications in Scientific Computing. Project Report*, volume 12 of *Research Report Series / Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM), Technische Universität München.* Shaker Verlag, Aachen, 1998.

[Lub86] M. Luby: *A simple parallel algorithm for the maximal independent set problem.* SIAM Journal on Computing, 4:1036–1053, 1986.

[Mav98] Dimitri J. Mavriplis: *Directional agglomeration multigrid techniques for high-Reynolds number viscous flows.* NASA/CR-1998-206911, ICASE Report No. 98-7, Institute for Computer Applications in Science and

Engineering, NASA Langley Research Center, Hampton, VA, January 1998.

[McC87] Stephen McCormick (editor): *Multigrid Methods.* Frontiers in Applied Mathematics. SIAM, Philadelphia, 1987.

[Meu99] Gerard Meurant: *Computer Solution of Large Linear Systems*, volume 28 of *Studies in Mathematics and its Applications.* North Holland, 1999.

[Mul89] Wim A. Mulder: *A new multigrid approach to convection problems.* Journal of Computational Physics, 83:303–323, 1989.

[NvR93] N. H. Naik and J. van Rosendale: *The improved robustness of multigrid elliptic solvers based on multiple semicoarsened grids.* SIAM Num. Anal., 30:215–229, 1993.

[OW90] Thomas Ottmann und Peter Widmayer: *Algorithmen und Datenstrukturen.* Reihe Informatik. BI–Wissenschaftsverlag, Mannheim, Wien, Zürich, 1990.

[OW95] C. W. Oosterlee und P. Wesseling: *On the robustness of a Multiple Semicoarsened Grid method.* Zeitschrift Angew. Math. und Mech., 75:251–257, 1995.

[Ren96] Henrik Rentz-Reichert: *Robuste Mehrgitterverfahren zur Lösung der inkompressiblen Navier-Stokes Gleichung.* Doktorarbeit, Universität Stuttgart, Institut für Computeranwendungen 3, Bericht N96/6, Dezember 1996.

[Reu94] Arnold Reusken: *Multigrid with matrix–dependent transfer operators for convection–diffusion problems.* In P. Hemker and P. Wesseling (editors): *7th International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 269–280. Birkhäuser, Basel, 1994.

[Rob91] Guy Robinson: *A simple parallel algebraic multigrid.* In *Occam and the Transputer*, pages 62–75. IOS Press, 1991.

[Rob93] Guy Robinson: *Parallel computational fluid dynamics on unstructured meshes using algebraic multigrid.* In A. Ecer R. B. Pelz and J. Häuser (editors): *Parallel Computational Fluid Dynamics '92: proceedings of the conference on Parallel CFD'92*, pages 359–370. Elsevier Science Publishers B. V., 1993.

[RS87] J. W. Ruge and Klaus Stüben: *Algebraic Multigrid*, chapter 4, pages 73–130. In McCormick, Stephen [McC87], 1987.

[SBG96] Barry Smith, Petter Bjørstad, and William Gropp: *Domain Decomposition — Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, Cambridge, first edition, 1996.

[Sch96] Dirk Schöllkopf: *Algebraische Mehrgitterverfahren auf Frequenzfilterbasis.* Diplomarbeit, Universität Stuttgart, Institut für Computeranwendungen 3, Mai 1996.

[Sti52] E. Stiefel: *Über einige Methoden der Relaxationsrechnung.* Zeitschrift für angewandte Mathematik und Physik, 3:1–33, 1952.

[Sto89] Josef Stoer (Herausgeber): *Numerische Mathematik 1.* Springer Lehrbuch. Springer–Verlag, Berlin, Heidelberg, 1989.

[Str00] *International Workshop on Algebraic Multigrid Methods.* Federal Institute for Adult Education (FIAE) in St. Wolfgang (Austria) organized by the Johannes Kepler Univerity Linz, Institute of Analysis and Computational Mathematics, June 2000. virtual proceedings on http://www.numa.uni-linz.ac.at.

[Stü83] Klaus Stüben: *Algebraic multigird (AMG): Experiences and comparisons.* Applied Mathematics and Computation, 13:419–451, 1983.

[Stü99a] Klaus Stüben: *Algebraic Multigrid (AMG): An Introduction with Applications.* In [TOS99], 1999.

[Stü99b] Klaus Stüben: *A review of algebraic multigrid.* GMD report no. 69, GMD — Forschungszentrum Informationstechnik GmbH, Schloß Birlinghoven, D–53754 Sankt Augustin, November 1999. to appear in Journal of Computational and Applied Mathematics, 2000.

[Top] Top500. Internetseite `http://www.top500.org`.

[TOS99] U. Trottenberg, C. W. Oosterlee, and A. Schüller: *"Multigrid" with guest contribution by K. Stüben, P. Oswald, A. Brand.* Academic Press, 1999.

[VBM98] Petr Vaněk, Marian Brezina, and Jan Mandel: *Convergence of algebraic multigrid based on smoothed aggregation.* Technical report 126, Center for Computational Mathematics, University of Colorado at Denver, February 1998.

[vdV92] H. van der Vorst: *Bi-CGSTAB: A fast and smoothly converging variant of bicg for the solution of nonlinear systems.* SIAM J. Sci. Stat. Comp., 13:631–644, 1992.

[VMB96] P. Vaněk, J. Mandel, and M. Brezina: *Algebraic multigrid by smoothed aggregation for second and forth order elliptic problems.* Computing, 56:179–196, 1996.

[Wag95] Christian Wagner: *Frequenzfilternde Zerlegungen für unsymmetrische Matrizen und Matrizen mit stark variierenden Koeffizienten.* Doktorarbeit, Universität Stuttgart, Institut für Computeranwendungen 3, Bericht N95/7, Juli 1995.

[Wag98] Christian Wagner: *Introduction to algebraic multigrid.* Lecture Notes of an Algebraic Multigrid Course at the University of Heidelberg in the Wintersemester 1998/99, 1998.

[Wag00a] Christian Wagner: *Ein Algebraisches Multilevelverfahren — Entwicklung und Anwendung auf einen Sanierungsfall.* Habilitationsschrift, Universität Heidelberg, Interdisziplinäres Zentrum für wissenschaftliches Rechnen, 2000.

[Wag00b] Christian Wagner: *On the algebraic construction of multilevel transfer operators.* Computing 2000, 2000. to appear.

[Wes81] Pieter Wesseling: *A robust and efficient multigrid method*, pages 614–630. Volume 960 of Hackbusch, Wolfgang and Ulrich Trottenberg [HT81], 1981.

[Wes92] Pieter Wesseling: *An Introduction to Multigrid Methods.* John Wiley & Sons, Chichester, 1992.

[Wie00a] Christian Wieners: *Parallel linear algebra and the application to multigrid methods.* In W. Hackbusch and G. Wittum (editors): *Notes on Numerical Fluid Mechanics, vol. 56.* Vieweg Verlag, 2000. to appear.

[Wie00b] Christian Wieners: *Theorie und Numerik der Prandtl-Reuß Plastizität.* Universität Heidelberg, 2000. Habilitationsschrift.

[Wit89] Gabriel Wittum: *On the robustness of ILU–smoothing.* SIAM J. Sci. Stat. Comput., 10:699–717, 1989.

[Wit92] Gabriel Wittum: *Filternde Zerlegungen — Schnelle Löser für große Gleichungssysteme*, Band 1 der Reihe *Teubner Skripten zur Numerik.* B. G. Teubner, Stuttgart, 1992.

[WJ99] Katina Warendorf and Wayne D. Joubert: *Parallel coarsening of algebraic multigrid.* Technical report, Los Alamos National Laboratory, Los Alamos, NM, August 1999.

[WKW97] Christian Wagner, Wolfgang Kinzelbach, and Gabriel Wittum: *Schur-complement multigrid — a robust method for groundwater flow and transport problems.* Numerische Mathematik, 75:523–545, 1997.

[WO98] T. Washio and C. W. Oosterlee: *Flexible multiple semicoarsening for 3d singularly perturbed problems.* SIAM J. Sci. Comput., 19:1646–1666, 1998.

[WW97] Christian Wagner and Gabriel Wittum: *Adaptive filtering.* Numerische Mathematik, 75:305–328, 1997.

Index

Lebenslauf

Persönliche Daten

Name	Christian Wrobel
Adresse	Christofstraße 39 A, 71332 Waiblingen
Geburtsort	Augsburg
Geburtstag	17.2.1967
Staatsangehörigkeit	deutsch
Familienstand	ledig

Ausbildung

1973 – 1977	Grundschule Donauwörth
1977 – 1986	Gymnasium Donauwörth
	Abitur
1986 – 1987	Grundwehrdienst
1987 – 1994	Studium an der Universität Augsburg
	Stipendium nach dem bayr. Begabtenförderungsgesetz
	Vordiplom in Mathematik
	Siemens Studentenkreis
	Diplom in Mathematik
1994 – 1997	Universität Stuttgart
	Stipendiat im Graduiertenkolleg „Modellierung und Diskretisierungsverfahren für Kontinua und Strömungen“
	Mitglied im Siemens Doktorandenkreis

Anstellungen

1989 – 1992	Werkstudent im Programm „T“ bei Siemens AG, München
1997 – 1998	wissenschaftlicher Angestellter der Universität Stuttgart
1998 – 2000	wissenschaftlicher Angestellter der Universität Heidelberg
seit 2000	Software Entwicklungsingenieur bei ETAS GmbH