Zusammenfassung....5
1. Einführung....9
2. Multilevel-Methoden....13
2.1. Modellproblem und Notation....14
2.2. Vorkonditionierer....16
3. Parallelisierung von Multilevel-Methoden....21
3.1. Theoretische Parallelrechner....22
3.2. Ansätze für Parallelisierungen....24
3.3. Das Kostenfunktional für Punktverteilungen....27
3.4. Abgeleitete Kostenfunktionale....30
4. Ein Beispielprogramm....31
4.1. Formulierung....32
4.2. Parallelisierung....32
4.3. Ergebnisse....33
5. Adaptive parallele Multilevel-Methoden....37
5.1. Fehlerschätzer....38
5.2. Gittermanipulation....38
5.3. Grobgitterlöser....41
5.3.1 Volle Grobgittermatrizen....41
5.3.2 Dünne Grobgittermatrizen....43
6. Lastverteilung adaptiver paralleler Multilevel-Methoden....47
6.1. Dynamische Ansätze....48
6.2. Der schlechteste Fall für statische Ansätze....49
6.2.1. Das Problem....50
6.2.2. Ansätze....50
6.2.3. Zusätzliche Voraussetzungen....50
6.3. Übersicht über statische Ansätze....51
6.3. Statistischer Ansatz mit geeigneter Komplexität....52
7. Experimentelle Ergebnisse....55
7.1. Ergebnisse auf Intel iPSC/2....56
7.2. Ergebnisse auf T800....60
7.3. Ergebnisse auf Alliant FX/2800....62
8. Schlußbemerkungen....65
Literatur....67

TECHNISCHE UNIVERSITÄT MÜNCHEN

Mathematisches Institut

Adaptive parallele Multilevel-Methoden
zur Lösung elliptischer Randwertprobleme

Diplomarbeit von
Gerhard W. Zumbusch

Aufgabensteller: Prof. Dr. Ronald Hoppe
Abgabedatum: 15.05.1992

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwendung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne vorherige Zustimmung des Autors unzulässig und strafbar. Das gilt besonders für Übersetzungen, Entnahmen von Abbildungen, Wiedergabe auf photomechanischem Weg und Speicherung in Datenverarbeitungsanlagen.
Alle Rechte vorbehalten.

© Gerhard W. Zumbusch 1991

Ich erkläre hiermit, daß ich Diplomarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe.

München, im Dezember 1991

(Gerhard Zumbusch)

Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme *

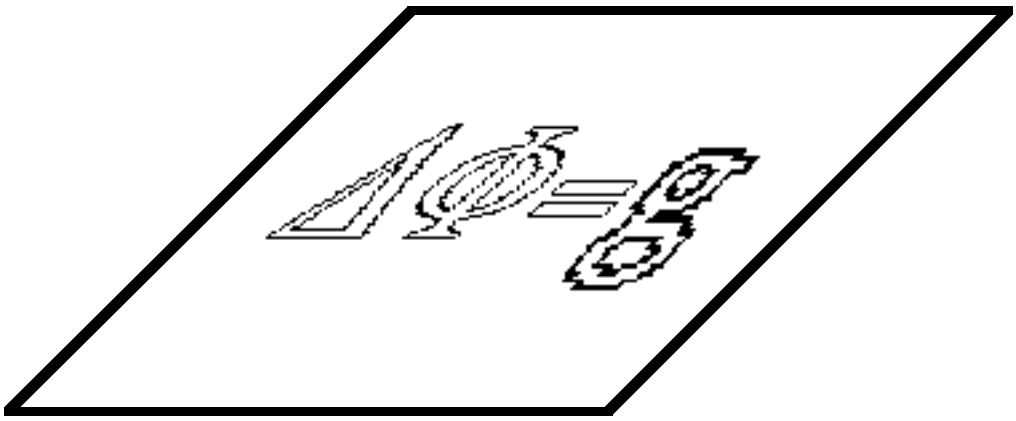
Gerhard W. Zumbusch
mit Unterstützung von Prof. Dr. Ronald Hoppe

Multilevel Methoden sind die zur Zeit effizientesten Verfahren zur Lösung großer symmetrischer schwachbesetzter linearer Gleichungssysteme, die aus der Diskretisierung selbstadjungierter elliptischer Randwertprobleme mit Finiten-Elementen entstehen. Im folgenden wird die Parallelisierung eines von Bramble, Pasciak und Xu vorgeschlagenen vorkonditionierten Verfahrens der konjugierten Gradienten, eingebettet in ein adaptives Finite-Elemente-Programm wie etwa Kaskade, diskutiert.

Dabei müssen zur effizienten Lastverteilung zusätzliche Forderungen an Triangulierungen, Finite-Elemente-Räume und Gittermanipulationsalgorithmen gestellt werden. Es werden Standardverfahren der Lastverteilung mit einem neuen Aufteilungsverfahren, das auf einem statistischen Ansatz beruht, verglichen. Mit einer hier vorgestellten gemischten Strategie der Aufteilung von Gitterpunkten und Dreiecken kann ein Gesamtverfahren von optimaler Ordnung erreicht werden.

Die experimentellen Ergebnisse auf einigen Parallelrechnern zeigen eine hohe Übereinstimmung mit einem hergeleiteten Kostenfunktional und eine ideale Parallelisierbarkeit des Verfahrens. Unterschiede zu anderen bekannten Multilevelverfahren werden in den einzelnen Abschnitten herausgestellt.

* Teile dieser Arbeit sind auch erschienen als: Gerhard W. Zumbusch, „Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme“, TU München, Institut für Informatik, TUM–I9127, SFB-Bericht Nr. 342/19/91 A (Juli 1991)



Kapitel 1
Einführung

Multilevel Methoden sind die zur Zeit effizientesten Verfahren zur Lösung großer linearer Gleichungssysteme, die aus der Diskretisierung elliptischer Randwertprobleme entstehen. Der Aufwand, mit Mehrgitterverfahren oder auch mit multilevel vorkonditionierten Verfahren der konjugierten Gradienten (CG) im symmetrisch positiv definiten Fall ein Gleichungssystem bis auf Diskretisierungsgenauigkeit zu lösen, ist unter geeigneten Voraussetzungen proportional zur Zahl der Unbekannten oder nur um einen logarithmischen Term höher.

Nach Standard-Mehrgitter V- und W-Zyklen [Hackbusch85] sind in letzter Zeit Verfahren mit hierarchischen Basen, als Mehrgitterverfahren [Bank, Dupont & Yserentant] und als Vorkonditionierer [Yserentant] in den Mittelpunkt des Interesses gerückt, da in den Konvergenzbeweisen auf stark einschränkende Regularitätsannahmen verzichtet werden kann. Für Randwertprobleme in drei Raumdimensionen sind diese Verfahren allerdings nicht mehr von quasioptimaler Ordnung wie im zweidimensionalen Fall, so daß sich als Alternative ein ähnlicher Ansatz von [Bramble, Pasciak & Xu] als Vorkonditionierer anbietet, dessen Mehrgittervariante der Standard V-Zyklus ist [Bramble, Pasciak, Wang & Xu]. Das entstehende Verfahren ist im Fall quasiuniformer Hierarchien von Triangulierungen von optimaler Ordnung [Oswald], unabhängig von der Raumdimension.

Die Ausführungsgeschwindigkeit kann durch adaptive Verfeinerungstechniken, die die Zahl der notwendigen Unbekannten reduzieren, drastisch erhöht werden. Dazu existieren vollständige Programmpakete, wie PLTMG [Bank] und Kaskade [Leinen], [Deuflhard, Leinen, Yserentant], [Bornemann, Erdmann, Roitzsch], die die Ordnung des eingebauten iterativen Lösers durch Gitterverwaltung, Verfeinerung und Gittermanipulation nicht verschlechtern.

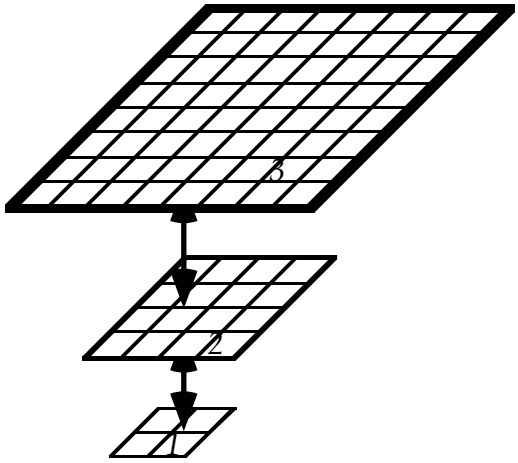
Die Ordnung des Lösungsverfahrens kann nur durch parallele Ausführung gesenkt werden. In Hinblick auf sehr große lineare Gleichungssysteme, wie sie insbesondere auch durch Randwertprobleme in drei Raumdimensionen entstehen, liegt es nahe, beide Techniken zu verbinden. Bei der Lastverteilung adaptiver also dynamisch erzeugter Strukturen können allerdings nicht mehr alle Voraussetzungen an die Finiten-Elemente-Räume und alle Algorithmen zur Gittermanipulation vom sequentiellen Programm übernommen werden.

Bekannte Ansätze, wie [Fox & Otto], [Berger & Bokhari] und [Bastian]

führen zu Gesamtverfahren, deren Komplexität höher als die des zugrundeliegenden iterativen Lösers ist, und nutzen die Multilevel-Struktur der Gitter nicht aus. Ansätze zur Parallelisierung von Standard-Mehrgitterverfahren wie [Briggs, Hart, McCormick & Quinlan] oder von adaptiven Mehrgitterverfahren wie [Mierendorff] können in dieser Form nicht auf adaptive Verfahren angewendet werden, obwohl sie für regulär verfeinerte Gitter gute Ergebnisse liefern.

Die Verfahren der Gebietszerlegung, wie [Bramble, Pasciak & Schatz], [Bjørstad & Widlund] oder [Bramble, Pasciak, Wang & Xu], bei denen die Parallelisierung die Konvergenzraten beeinflußt, weil die Kopplung der Daten zwischen den einzelnen Prozessoren anders organisiert wird als die Kopplung der Daten innerhalb der einzelnen Prozessoren, konnten bisher noch nicht die Effizienz von den Verfahren erreichen, die die numerischen Eigenschaften des sequentiellen Programms bei der Parallelisierung erreichen.

Wir werden im folgenden Parallelrechner mit verteiltem Speicher und Message-Passing-Kommunikation und Parallelrechner mit gemeinsamem Speicher und Semaphor-Synchronisation verwenden, um ein multilevel-vorkonditioniertes CG-Verfahren so zu implementieren, daß die Eigenschaften des sequentiellen Programms, soweit möglich, erhalten bleiben, und gleichzeitig eine effiziente Parallelisierung erreicht wird.



Kapitel 2
Multilevel-Methoden

Die im weiteren verwendeten Schreibweisen stammen aus der zitierten Literatur, wo auch weitergehende Bemerkungen über Problemstellung und Lösungsverfahren gemacht werden. Eine allgemeinere mathematische Darstellung der Methode der Finiten-Elemente findet man in [Ciarlet], eine Zusammenfassung klassischer Lösungsverfahren dafür in [Axelson & Barker], eine Einführung in Mehrgitterverfahren in [Hackbusch85] und einen Überblick über neuere Lösungsverfahren in [Hackbusch91]. Spezielle Multilevel-Verfahren sind bisher nur in den entsprechenden Orginalarbeiten beschrieben worden.

Wir führen die Aufgabenstellung, Lösung eines elliptischen Randwertproblems, und die dazugehörigen iterativen Lösungsverfahren ein, die wir in den weiteren Kapiteln dann parallelisieren werden. Zunächst notieren wir, was wir unter dem Modellproblem verstehen wollen.

2.1. Modellproblem und Notation

Gegeben sei folgendes Modellproblem: Es wird die Lösung u eines elliptischen Randwertproblems zweiter Ordnung auf einem polygonal berandeten Gebiet Ω aus $\mathbb{R}^2$ oder einem polyedrisch berandeten Gebiet aus $\mathbb{R}^3$ gesucht:

$$\begin{aligned} L\,u &= f && \text{in } \Omega \\ u &= 0 && \text{auf } \partial\Omega \end{aligned}$$

mit dem selbstadjungierten linearen skalarwertigen elliptischen Differentialoperator zweiter Ordnung L. Dieser soll sich schreiben lassen als

$$L\,u = -\sum_{i,l} \frac{\partial}{\partial x_i}\, \hat{a}_{il}\, \frac{\partial u}{\partial x_l} + \hat{a}u$$

mit uniform positiv definiter symmetrischer Matrix $\{\hat{a}_{il}(x)\}$, also dx-fast-überall mit gemeinsamen Schranken der Eigenwerte endlich und echt größer Null, und nicht negativem integrierbarem $\hat{a}(x)$ und $\hat{a}$, $\hat{a}_{il} \in L^\infty(\Omega)$. Damit ist L koerzitiv und beschränkt.

In der schwachen, variationellen Formulierung des Problems wird bei gegebenem $f \in L^2(\Omega)$ eine Funktion $u \in H_0^1(\Omega)$ gesucht, so daß gilt

(1) $\quad a(u,v) = (f,v) \quad \forall\, v \in H_0^1(\Omega)$

mit dem L^2-Skalarprodukt (.,.) und dem verallgemeinerten Dirichlet-Integral

$$a(u,v) = \sum_{i,l} \int_\Omega \hat{a}_{il}\, \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_l}\, dx + \int_\Omega \hat{a}uv\, dx.$$

Mit dem Lax-Milgram Lemma folgt aus der Beschränktheit und Koerzitivität von L die Existenz und Eindeutigkeit der Lösung u in dem Sobolevraum $H_0^1(\Omega)$ aller Funktionen, die als Grenzwerte beliebig oft stetig differenzierbarer Funktionen im Raum $H^1(\Omega)$ der Funktionen

darstellbar sind, die zusammen mit ihren ersten partiellen Ableitungen im Distributionensinn aus L^2 sind. Die Lösung erfüllt die Randbedingungen im Sinne des Spuroperators. Dabei minimiert die Lösung das quadratische Energie-Funktional

$$E(u) = \frac{1}{2}\, a(u,u) - (f,u).$$

Wir diskretisieren das kontinuierliche Problem mit linearen Finite-Elementen. Alle dargestellten Funktionen sind damit stetig und stückweise auf jedem Element linear. Der Interpolations- oder auch Diskretisierungsfehler ist von der Ordnung des jeweiligen Elementdurchmessers. Bei geeigneter Symmetrie der Elemente kann die Ordnung auch das Quadrat davon erreichen.

Für Probleme in zwei Raumdimensionen konstruieren wir die Finite-Elemente aus einer Folge von Triangulierungen $\tau_0 \subset \tau_1 \subset \ldots \subset \tau_j$ des Gebietes Ω, für die die Ausgangstriangulierung τ_0 ganz Ω mit Dreiecken überdeckt, und jedes Dreieck mit einem anderen eine Kante, einen Punkt oder nichts gemeinsam hat. Die Ausgangstriangulierung hat damit insbesondere den polygonalen Rand des Ausgangsproblems. Jede weitere Triangulierung entsteht aus der Vorhergehenden durch Zerlegen einzelner

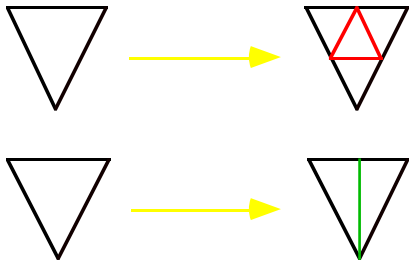


Abb 2.1.1. a,b. *Zerlegen von Dreiecken*

Dreiecke in 4 kongruente Teildreiecke oder durch Seitenhalbierung in zwei Dreiecke, die beide im weiteren aber nicht mehr verfeinert werden (Abbildungen 2.1.1.a,b). Die Mengen der so entstehenden Eckpunkte der Dreiecke im Inneren von Ω bezeichnen wir mit Ω_k .

Für Probleme in drei Raumdimensionen können auf eine ähnliche Weise Triangulierungen aus Tetraedern erzeugt werden. Bei der regulären Zerlegung in 8 Teiltetraeder muß allerdings der durch Kantenhalbierung entstehende Oktaeder entlang der längsten Raumdiagonale geteilt werden, um nicht zu stumpfwinklige Tetraeder zu erzeugen (nach Beweis von [Bänsch]), siehe Abbildung 2.1.2.

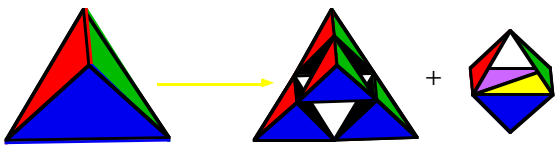


Abb 2.1.2. *Zerlegen von Tetraedern*

Wir konstruieren geschachtelte lineare Finite-Elemente-Räume V_k über den Triangulierungen τ_k mit

$0 \le k \le j$. In V_k diskretisiert, schreibt sich (1) über einer Knotenbasis $\{\Phi_i^k\}$ mit gesuchtem

$$u_k = (u_{k,i})_i \,,\ 1 \le i \le \dim V_k$$

$$(2) \qquad \sum_i a(\Phi_i^k, \Phi_l^k)\, u_{k,i} = (f, \Phi_l^k)$$

$$1 \le l \le \dim V_k.$$

Mit Hilfe der dünnbesetzten Steifigkeitsmatrix

$$A_k = (a(\Phi_i^k, \Phi_l^k))_{i,l}$$

und der rechten Seite

$$f_k = (f, \Phi_l^k)$$

kann man daraus ein lineares Gleichungssystem

$$(3) \qquad A_k u_k = f_k$$

konstruieren (Galerkin-Ansatz), wenn die Randbedingungen entsprechend diskretisiert und in das Lösungsverfahren eingefügt werden. Man kann (3) auch mit dem Ansatz von Ritz äquivalent als Minimierungsproblem formulieren, wobei u_k das diskrete quadratische Energie-Funktional

$$(4) \qquad E_k(u_k) = \tfrac{1}{2} u_k^t A_k u_k - f_k^t u_k$$

minimiert.

Im folgenden geht es um die effiziente Lösung des Gleichungssystems (3) der Verfeinerungsstufe j. Dabei wird die Lösung iterativ nur bis zu einer Genauigkeit von der Größenordnung des Diskretisierungsfehlers berechnet, um Rechenoperationen einzusparen.

2.2. Vorkonditionierer

Ausgehend von (4) kann man durch die Anwendung iterativer Minimierungsverfahren auf E_k Näherungslösungen für (3) berechnen. In dieser Interpretation wird der Fehler einer Startlösung bezüglich E_k in jedem Schritt des Minimierungsverfahrens um mindestens einen konstanten Faktor reduziert. Das Residuum, also die Differenz zwischen f_k und bisher berechnetem $A_k \hat{u}_k$, wird dabei als neue rechte Seite des zu lösenden Gleichungssystems im nächsten Minimierungsschritt verwendet.

Einzelschrittverfahren wie Richardson-, Jacobi- und Gauß-Seidel-Verfahren reduzieren den Fehler in jedem Iterationsschritt um mindestens einen konstanten Faktor, der allerdings polynomial von der Verfeinerungsstufe abhängt. Gedämpfte Versionen wie SOR und SSOR konvergieren zwar schneller, sind aber auch noch schrittweitenabhängig. Klassische Minimierer, wie die Gesamtschrittverfahren Gradientenmethode und konjugierte Gradientenmethode, sind genauso schrittweitenabhängig und im Vergleich langsam, können aber durch Vorkonditionierung, also Transformation auf eine geeignetere Basis, verbessert werden.

Die ersten Löser, deren Konvergenzrate für spezielle Probleme un-

abhängig von der Schrittweite ist, sind Mehrgitterverfahren gewesen. Dabei betrachtet man das Gleichungssystem (3) nicht nur für die Verfeinerungsstufe j, sondern für alle Stufen von 0 bis j gleichzeitig. Dem entspricht ein Minimierungsverfahren mit einer indefiniten Steifigkeitsmatrix angewendet auf Residuen, die über dem Erzeugendensystem der Knotenbasen aller Räume V_k dargestellt werden. Dazu werden ausgehend von A_j durch Anwenden von Restriktionsoperatoren sukzessive alle A_k gebildet und zu einer Blockdiagonalmatrix vom Rang von A_j zusammengesetzt und auf dieses System gewöhnliche Minimierungsverfahren angewendet.

Während das Jacobi-Verfahren in dieser Form im allgemeinen divergiert, liefert das symmetrische Gauß-Seidel-Verfahren mit geeigneter Anordnung der Punkte einen Mehrgitter V-Zyklus mit einem Vorglättungsschritt. Das Gesamtschrittverfahren der konjugierten Gradienten angewendet auf die indefinite Matrix führt zu einem BPX-vorkonditionierten CG-Verfahren [Griebel]. Damit kann man direkt die Konvergenzraten beider Verfahren vergleichen.

Die Interpretation als multiplikative und additive Schwarz-Iterationen führen zu einem anderen Zusammenhang zwischen Mehrgitterverfahren und dem BPX-Vorkonditionierer. Wir betrachten den diskreten Ritz-Galerkin-Operator A_k auf V_k, die diskrete L^2-Orthogonalprojektion Q_k von V_j nach V_k und die a-Orthogonalprojektion P_k von V_j nach V_k. Ein Mehrgitter V-Zyklus mit einem Vorglättungsschritt durch den symmetrischen Glätter $S_k = Id - R_k A_k$ läßt sich ausdrücken durch

$$(5) \qquad M_j^{MG} = \prod_{k=0}^{j} (Id - T_k)$$

$$\text{mit } T_0 := Id - P_0,$$

$$T_k := (Id - S_k) P_k = R_k A_k P_k = R_k Q_k A_j, \quad k > 0.$$

Unter Vernachlässigung aller Terme höherer Ordnung in T entsteht daraus ein additives Iterationsverfahren

$$M_j = Id - \sum_{k=0}^{j} T_k$$

[Bramble, Pasciak, Wang & Xu], das auf der Zerlegung

$$u_j = Q_0 u_j + \sum_{k=1}^{j} (Q_k - Q_{k-1}) u_j$$

und der Vernachlässigung der Terme Q_{k-1} gegenüber Q_k darin beruht.

Mit $R_0 := A_0^{-1}$ und $R_k := r_k Id$ für $k > 0$ und r_k^{-1} näherungsweise dem Spektralradius ρ von A_k für uniform verfeinerte Triangulierungen kann man daraus einen Vorkonditionierer für ein CG-Verfahren konstruieren,

(6) $B_j = \sum_{k=0}^{j} R_k Q_k.$

In der Orginalarbeit wurde gezeigt, daß unter der Voraussetzung, daß es ein C>0 gibt mit

$$\| (Id - Q_{k-1}) v \|^2 \leq C \rho(A_k)^{-1} a(v,v) \quad \forall v \in V_j,$$

für zu A_k^{-1} spektral äquivalente R_k die Kondition κ von $B_j A_j$ abgeschätzt werden kann durch

(7) $\kappa(B_j A_j) \leq O(j^2)$
[Bramble, Pasciak & Xu].

Unter der stärkeren Regularitätsannahme konnte dort gezeigt werden, daß wenn es ein C>0 und ein $\alpha \in (0,1]$ gibt mit

$$a((Id - P_{k-1}) v,v) \leq (C \rho(A_k)^{-1} (A_k v)^2)^{\alpha} \; a(v,v)^{1-\alpha;} \quad \forall v \in V_k$$

folgt

$\kappa(B_j A_j) \leq O(j^{1/\alpha})$
[Bramble, Pasciak & Xu].

Für quasiuniforme Hierarchien von Triangulierungen konnte als erster [Oswald] mit Mitteln der konstruktiven Funktionentheorie zeigen, daß sogar gilt

(8) $\kappa(B_j A_j) \leq O(1),$

während [Zhang] lediglich mit einer verschärften Cauchy-Schwarz Ungleichung

$\kappa(B_j A_j) \leq O(j)$

bewiesen hat.

Kondition	quasiuniform	adaptiv
Regularität	O(1)	O(j)
	[Zhang]	[Yserentant]
allgemein	O(1)	O(j)
	[Oswald]	[Bornemann]

Tab. 2.2.1 *Kondition des BPX-Vorkonditionierers*

[Yserentant] hat die Normierung der Basisfunktionen Φ_i^k, die zur Konstruktion von Q_k in (6) führen, mit dem Faktor $a(\Phi_i^k, \Phi_i^k)^{-1}$ auf den Fall allgemeinerer adaptiv verfeinerter Hierarchien von Triangulierungen ausgedehnt und damit ohne Regulariätsannahmen

$\kappa(B_j A_j) \leq O(j^2)$

bewiesen. Dieses Resultat konnte inzwischen auf

(9) $\kappa(B_j A_j) \leq O(j)$

verbessert werden, und experimentelle Ergebnisse legen sogar eine ähnliche Abschätzung wie im quasiuniformen Fall (8) nahe [Bornemann], siehe auch Tabelle 2.2.1.

Mit einer L^2-Orthonormalbasis $\{\Psi_i^k\}$ von V_k ist $Q_k u = \sum_i (u, \Psi_i^k) \Psi_i^k$. Für den praktischen Vorkonditionierer genügt die geeignet normierte Knotenbasis $\{\Phi_i^k\}$ von V_k. Damit ist

(10) $B_j^{BPX} u = \sum_{k=0}^{j} \sum_i (u,\Phi_i^k) \Phi_i^k.$

Die Terme $\{ (u,\Phi_i^k) \}_i$ lassen sich aus $\{ (u,\Phi_i^{k+1}) \}_i$ durch Linearkombinationen gewinnen, genauso wie die Knotenbasisdarstellungen in V_j der Terme $\{ \Phi_i^k \}_i$ aus den Darstellungen der $\{ \Phi_i^{k+1} \}_i$ durch Linearkombinationen folgen. Damit läßt sich die

Auswertung von $B_j^{BPX} u$ mehrgitterartig in $O(\sum_{k=0}^{j} \dim V_k)$ Operationen berechnen. Durch gitterpunktweises Umordnen der Terme kann dieser Wert in einer Implementierung wie ein additives Schwarz-Verfahren auf $O(\dim V_j)$ reduziert werden. Das gesamte iterative Lösungsverfahren, daß von einer Startlösung aus V_0 ausgehend eine Lösung des Randwertproblems aus V_j in Diskretisierungsgenauigkeit liefert, braucht damit insgesamt

$$O(\sum_{k=0}^{j} \dim V_k) \text{ oder}$$
$$O(\sum_{k=0}^{j} k \dim V_k)$$

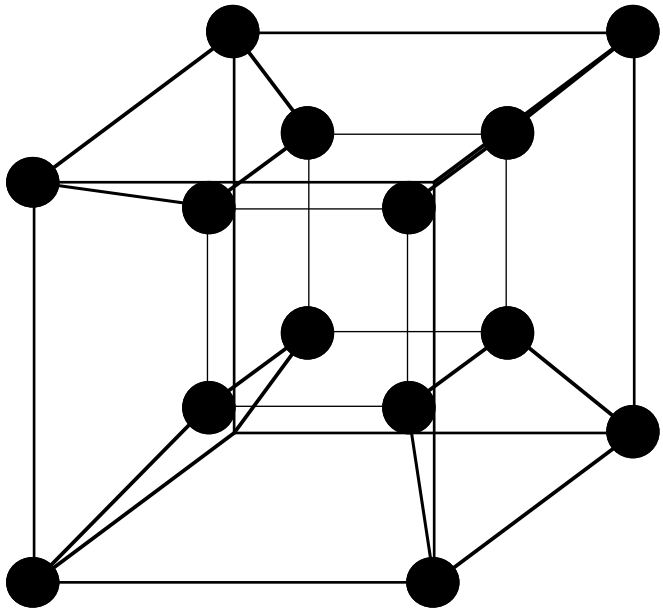
Rechenoperationen, je nach Abschätzung der Kondition. Im folgenden werden wir teilweise auch noch den Hierarchischen-Basis Vorkonditionierer

$$(11) \quad B_j^{hier} = \sum_{k=0}^{j} \sum_{\substack{i \text{ für die} \\ \Phi_i^k \in V_k \setminus V_{k-1}}} (u, \Phi_i^k) \Phi_i^k$$

betrachten, für den in zwei Dimensionen

$$(12) \quad \kappa(B_j^{hier} A_j) \le O(j^2)$$
[Yserentant]

in einer scharfen Abschätzung gilt. Eine Auswertung des hierarchischen Vorkonditionierers kostet $O(\dim V_j)$ Operationen. Für Ω aus $\mathbb{R}^3$ degeneriert die Abschätzung (12) zu $O(2^j)$. Im Gegensatz dazu bleiben die Abschätzungen (7)–(9) für den BPX-Vorkonditionierer in Räumen höherer Dimension erhalten.



Kapitel 3
Parallelisierung von Multilevel-Methoden

Wir werden im folgenden die Aufteilung der Rechenoperationen zur Auswertung von $B_j^{BPX}u$ und die dabei entstehenden Datenabhängigkeiten und damit die notwendigen Kommunikationsoperationen untersuchen. Das tun wir zunächst anhand eines theoretischen massivparallelen Rechners und dann durch Zusammenfassen einzelner Operationen für reale Rechnerarchitekturen. Wir untersuchen verschiedene Strategien der Parallelisierung und entwickeln dann ein Kostenfunktional, das später zur Optimierung der Aufteilung verwendet wird.

Wir gehen zunächst von einem geometrischen Wachstum der $\{\dim V_k\}$ aus,

(13) $\dim V_k \geq C \dim V_{k-1}$, $C > 1$, für $k>0$.

3.1. Theoretische Parallelrechner

Sei ein theoretischer paralleler Rechner mit genügend vielen Prozessoren gegeben zur Berechnung von $B_j^{BPX}u$. Alle Terme (u,Φ_i^k) können unabhängig voneinander berechnet werden. (u,Φ_i^k) ist Linearkombination von $O(C^k)$ Werten. Dafür muß also auf $O(C^k)$ Prozessoren in Baumanordnung (siehe Abbildung 3.1.) mindestens O(k log C) Rechenzeit aufgewendet werden.

Die Summe $\sum_{k=0}^{j} \sum_i (.,.)\,\Phi_i^k$ besteht in jedem Gitterpunkt aus höchstens O(j C) Summanden, benötigt also mit $O(j \dim V_j)$ Prozessoren O(log j) Rechenzeit, bzw. mit $O(\dim V_j)$ Prozessoren O(j) Zeit.

Insgesamt ist also durch massive Parallelisierung mit dem Einsatz von $O(\dim V_j)$ Prozessoren eine Reduktion der Ordnung von

(14) $O(\dim V_j)$ auf $O(j)$

erreichbar, das geometrische Wachstum der $\{\dim V_k\}$ vorausgesetzt.

Die gleiche Ordnung ist auch bei einem Mehrgitter V-Zyklus mit lokalem, vollständig parallelisierbarem Vorglätter, wie z.B. einer Schachbrett-Gauß-Seidel-Iteration, und auch bei Anwendung des hierarchischen Basis-Vorkonditionierers erreichbar.

Liegt schwächeres Wachstum der $\{\dim V_k\}$ vor, dann kann unter Einsatz von $O(j \dim V_j)$ Prozessoren der kleinere Wert

(15) $O(\log (j \dim V_j))$

für den BPX-Vorkonditionierer und den hierarchischen Basis Vorkonditionierer erreicht werden (siehe auch Tabelle 3.3.2.)

Für den Mehrgitter V-Zyklus ist eine solche Verbesserung von $O(j)$ auf einen kleineren Wert im allgemeinen nicht zu erwarten, da der

Glätter für den Wert an einem Gitterpunkt stets Nachbarwerte braucht, die aber nach obiger Reihenfolge der Berechnung teilweise erst später berechnet werden.

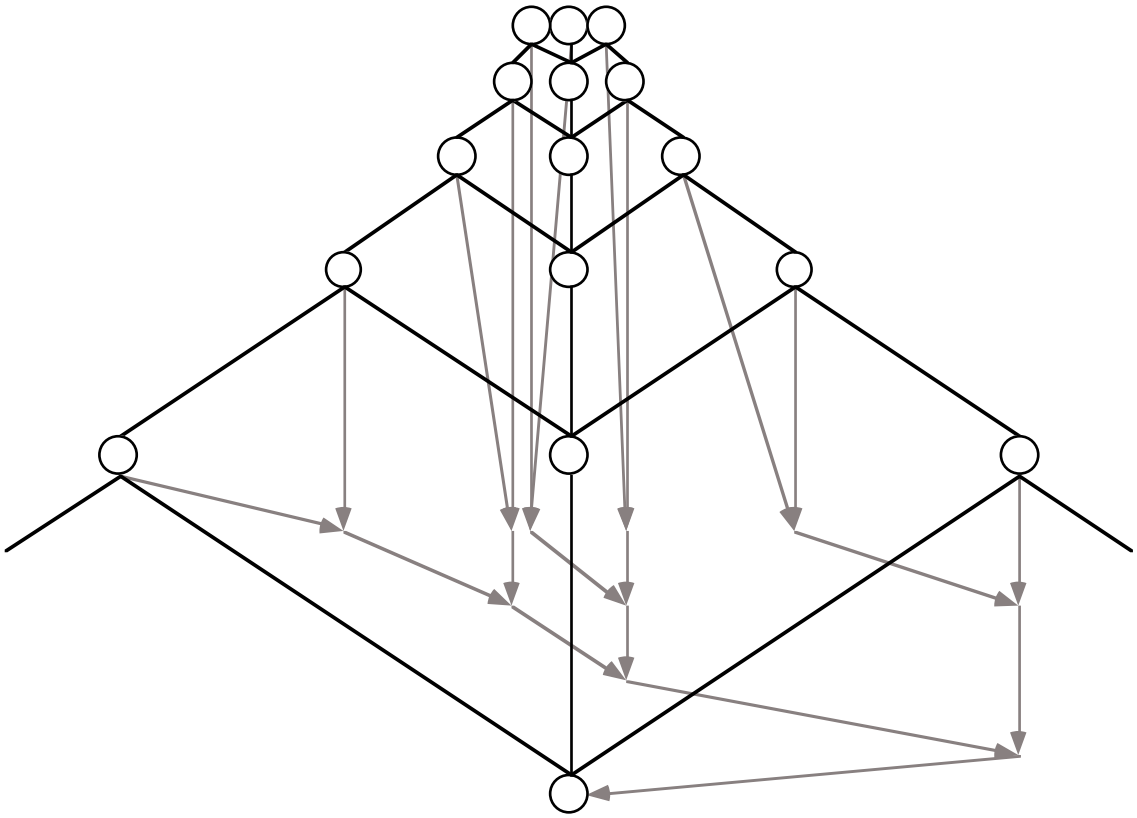


Abb 3.1. *Restriktion für einen Punkt mit binärem Baum*

3.2. Ansätze für Parallelisierungen

Wir betrachtem im folgenden Ansätze, diese Parallelität auf realen Parallelrechnern nutzbar zu machen.

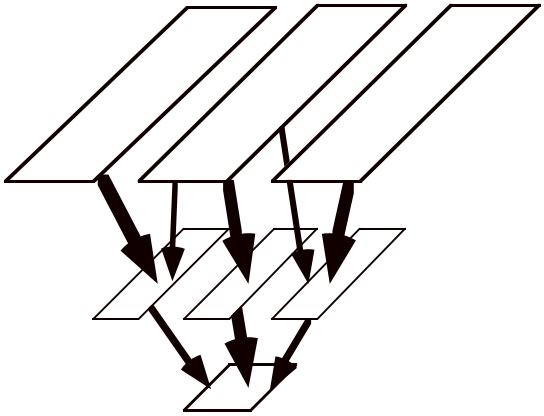


Abb 3.2.1. *Aufteilung von Gittern mit Datenabhängigkeiten*

Die Abbildung 3.2.1 zeigt eine mögliche Aufteilung von drei Verfeinerungsebenen und die möglichen Datenabhängigkeiten.
Bei realen MIMD-Rechnern hat man eine konstante Zahl von Prozessoren zur Verfügung, die im allgemeinen wesentlich kleiner als die Zahl der Unbekannten ist. Man muß also Rechenoperationen, die theoretisch parallel durchgeführt werden könnten, zu Gruppen zusammenfassen und einem Prozessor zuordnen. Damit wird es wichtig, gleiche Operationen nicht mehrfach durchzuführen, was bei genügend vielen Prozessoren kein Problem war.

Wir stellen uns eine Auswertung des Vorkonditionierers in einer sequentiellen mehrgitterartigen Implementierung mit Restriktionen, Lösung auf dem gröbsten Gitter und Prolongationen vor. Regelmäßige Aufteilungen können sowohl elementorientiert, als auch punktorientiert durchgeführt werden. Dabei werden die Rechenoperationen und Datenabhängigkeiten so aufgeteilt, daß entweder Dreiecke oder Gitterpunkte immer auf einem gemeinsamen Prozessor berechnet werden.

In der Abbildung 3.2.2 ist eine elementorientierte Aufteilung eingezeichnet, wobei die Zeilen drei verschiedene Verfeinerungsstufen, von der feinsten bis zur gröbsten darstellen. Die Punkte sind als Kugeln, die Datenabhängigkeiten sind als Linien gekennzeichnet.

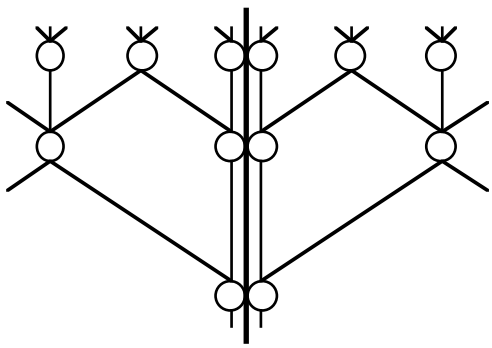


Abb 3.2.2. *Aufteilung von Elementen*

Die Aufteilung erfolgt entlang dem senkrechten Balken. Damit sind alle Punkte entlang der Trennungslinie doppelt vorhanden, wo-

bei sie jeweils nur Teilergebnisse enthalten und nicht, wie die inneren Punkte, konsistent sein müssen. Nach einem vollständigen Restriktionsschritt über alle Ebenen müssen für die Grobgitterlösung die Teilergebnisse an den Kanten zusammengefaßt werden, was einer lokalen Kommunikation entspricht. Damit werden auch diese Werte konsistent und stimmen auf beiden Nachbarprozessoren überein. Die Prolongationen anschließend arbeiten wieder mit inkonsistenten Rändern, die erst in einem abschließenden lokalen Kommunikationsschritt zusammengefaßt werden.

Eine Variante der Aufteilung ist die zeitliche Hintereinanderausführung der Operationen zweier Gebiete mit gemeinsamer Kante [Leinen]. Dabei kann eine Addition pro Gitterpunkt beim Zusammenfassen durch den lokalen Kommunikationsschritt eingespart werden, wobei allerdings eine Färbung der einzelnen Gebiete und die zeitliche Hintereinanderausführung der verschieden gefärbten Gebiete nötig wird. Weiterhin wird die Aufteilung in mindestens doppelt so viele (zwei Farben) Teile wie Prozessoren nötig, was die Lastverteilung behindern kann.

Eine Auswertung kann mit zwei lokalen Kommunikationsschritten durchgeführt werden. Das ist aber nur so lange richtig, wie die Aufteilung in Elemente bereits auf der gröbsten Triangulierung stattfindet. Für eine gute Lastverteilung der Prozessoren und damit große Beschleunigung durch die Parallelisierung ist aber im allgemeinen eine Aufteilung erst auf einem feineren Gitter möglich, was die Kommunikationsstruktur komplizierter macht.

Dazu betrachten wir nun die andere regelmäßige Aufteilung, die Verteilung der Gitterpunkte. In der Abbildung 3.2.3 ist die gleiche Aufteilung nur dieses Mal für Punkte gezeichnet.

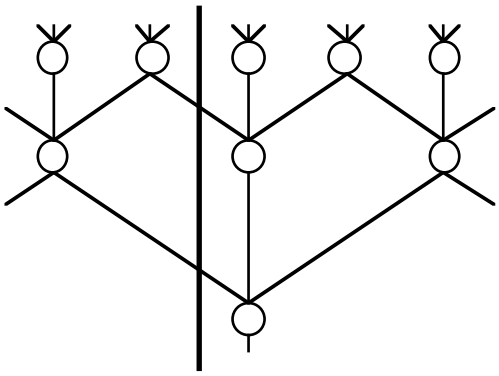


Abb 3.2.3. *Aufteilung von Gitterpunkten*

Die Werte der Gitterpunkte sind in jedem Schritt konsistent, weil sie nur genau einmal vorhanden sind. In jedem Restriktions- und Prolongationsschritt von einer zur nächsten Verfeinerungsebene muß jeweils eine lokale Kommunikation stattfinden, in diesem Fall allerdings

nur in einer Richtung. Der senkrechte Balken der Aufteilung auf die Prozessoren kann überall verlaufen, stets ist eine lokale Kommunikation in genau einer Richtung nötig.

Damit ist diese Aufteilung für eine gute Lastverteilung günstiger als die Verteilung der Elemente der gröbsten Ebene, weil die Punkte der feinsten Ebene frei verteilt werden können. Die Menge der zu transferierenden Daten ist bei beiden Methoden der Zerlegung gleich, nur die Zahl der Kommunikationsoperationen unterscheidet sich.

Für eine gute Lastverteilung, bei der eine bestimmte Effizienz der Parallelsierung erreicht werden soll, wird also zunächst eine Aufteilung und Kommunikation wie für Gitterpunkte durchgeführt, um ab einer festgelegten Verfeinerungsebene eine Aufteilung in Elemente zu verwenden und damit Kommunikationsoperationen zu sparen.

Wenn im folgenden von der Aufteilung von Gitterpunkten die Rede sein wird, so ist damit die Aufteilung der auf den gröberen Gittern als Punkte aufzufassenden Elemente und Hierarchien von Elementen auf dieser festgelegten Ebene zu verstehen.
Wie diese Ebene festgelegt werden sollte, wird in Kapitel 6.1. diskutiert.

Für die Parallelisierung von Mehrgitterverfahren ist eine Aufteilung in Elemente im allgemeinen nicht sinnvoll, da der Glätter konsistente Werte einer Gitterebene benötigt. Damit ist für jede Gitterebene ein lokaler Kommunikationsschritt nötig und eine Parallelisierung mittels Aufteilung der Gitterpunkte effizienter. Hier wird also bis zur feinsten Ebene punktorientiert verteilt.

Der im weiteren häufig verwendete Begriff der lokalen Kommunikation heißt in diesem Zusammenhang, daß jeder Prozessor nur mit einer beschränkten Zahl von Nachbarn Informationen austauscht. Das ist insbesondere für Abschätzungen für große Prozessorzahlen wichtig. Die Struktur der Verknüpfung der Prozessoren mit ihren Nachbarn wird auch als Prozessortopologie bezeichnet, wobei diese logische Struktur mit der physikalischen Struktur der Verknüpfungen der Prozessoren zusammenhängen sollte, um effizient kommunizieren zu können. Diese Strukturen sind sehr stark hardwareabhängig. In einem zweidimensionalen Netz von Prozessoren für ein Randwertproblem in zwei Raumdimensionen sollte jeder Prozessor mit mindestens 6 Nachbarn logisch verbunden sein, was aus der Struktur von Restriktion und Prolongation für innere Punkte hervorgeht.

Im folgenden gehen wir von einer mehrgitterartigen Implementierung von $B_j^{BPX}u$ oder $B_j^{hier}u$ sowie einer Aufteilung der Gitterpunkte auf p Prozessoren aus.

3.3. Das Kostenfunktional für Punktverteilungen

Wir betrachten für die Konstruktion des Kostenfunktionals eine Aufteilung der Gitterpunkte, die später durch die Minimierung des Funktionals optimiert werden soll. Jedes Gitter Ω_k wird also auf die p Prozessoren verteilt. Jeder Prozessor p erhält das Teilgebiet D_k^p aus Ω_k. Wir betrachten Aufteilungen mit

(16) $\bigcup_p D_k^p = \Omega_k$, die zunächst disjunkt seien,

(17) $D_k^i \cap D_k^j = \emptyset$ für $i \neq j$, woraus $\sum_{p=1}^{P} \#D_k^p = \#\Omega_k$ folgt.

Wir stellen ein Kostenfunktional für einen Modell-Parallelrechner auf, das die Zahl der wesentlichen Rechenoperationen und die Menge der Datenübertragungsoperationen bzw. die verlangsamende Benutzung des gemeinsamen Speichers, je nach Rechnerarchitektur, berücksichtigt und eine obere Schranke für die Rechenzeiten liefert. Im wesentlichen werden dabei Einsparungen durch Überlappung von Rechnung und Kommunikation vernachlässigt. Seien

(18) $supp(Op \mid D) :=$
$\bigcup \{ supp(f)$; mit
$supp(Op(f))$ aus D und
f aus Definitionsbereich von Op$\}$

die Träger der Urbildfunktionen von Gitterfunktionen unter dem Operator Op, deren Träger ganz in D aus dem Bildbereich des Gitteroperators Op enthalten sind, also genau die Menge der Gitterpunkte, die bei Anwendung von Op Werte der Gitterpunkte aus D beeinflussen.

Eine Restriktion mit dem Operator r von Ω_k nach Ω_{k-1} kostet dann

(19) $$W_k^R = \max_p \left\{ \#D_{k-1}^p \, T_R + \#\left(supp(r \mid D_{k-1}^p) \setminus D_k^p \right) C_R \right\}$$

mit einem Maß T_R für die wesentlichen Rechenoperationen und einem Maß C_R für Kommunikationsoperationen oder die Verzögerung bei Zugriffen auf den gemeinsamen Speicher. Die Prolongation mit $P = R^*$ von Ω_{k-1} nach Ω_k kostet entsprechend

(20) $$W_k^P = \max_p \left\{ \#D_k^p \; T_P + \#\left(supp(r^* \mid D_k^p) \setminus D_{k-1}^p \right) C_P \right\}.$$

Die exakte Lösung des Problems in V_0 kostet, mittels Cholesky-Zerlegung oder eines anderen direkten Lösers auf einem einzigen Prozessor mit maximalem $\#D_0^p$ durchgeführt,

$$(21)\quad W_0^E = (\#\Omega_0)^2\, T_E + (\#\Omega_0 - \max_p \#D_0^p)\, C_E.$$

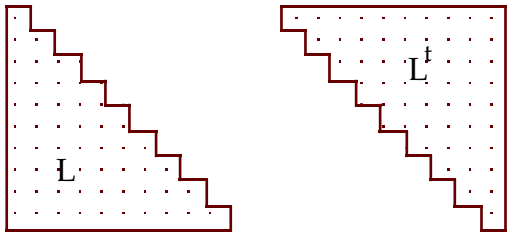


Abb 3.3.1 *Cholesky-Zerlegung für 9 Unbekannte*

Dabei ist die Zerlegung selbst, die zu Beginn des Lösungsverfahrens einmal bestimmt werden muß, höchstens von der Ordnung $(\#\Omega_0)^3$, wenn volle Matrizen verwendet werden (für schnellere Verfahren, siehe Kap. 5.3). Eine Anwendung von A_k, dem diskreten a-Operator, auf dem Gitter Ω_k kostet

$$(22)\quad W_k^A = \max_p \left\{ \#D_k^p\, T_A + \#(\,\mathrm{supp}(a \,|\, D_k^p) \setminus D_k^p\,)\, C_A \right\}.$$

Die für den Mehrgitter V-Zyklus verwendeten Glätter S_k auf Ω_k kosten, sofern sie lokal und vollständig parallelisierbar mittels Färbung {Col} der Gitterpunkte sind,

$$(23)\quad W_k^S = \sum_{\text{Col aus } \Omega_k^p} \max \left\{ \#(D_k^p \cap \mathrm{Col})\, T_S + \#(\,\mathrm{supp}(S \,|\, D_k^p \cap \mathrm{Col}) \setminus D_k^p\,)\, C_S \right\}.$$

Für das Schachbrett-Gauß-Seidel Verfahren ist {Col} beispielsweise gleich { {schwarze Punkte von Ω_k}, {rote Punkte von Ω_k} }. Die Vektoradditionen innerhalb des CG-Verfahrens oder des Mehrgitterverfahrens können ohne Kommunikation durchgeführt werden und kosten deshalb

$$(24)\quad W_k^V = \max_p \{\#D_k^p\}\, T_V.$$

Damit ergibt sich das Kostenfunktional für einen Mehrgitter V-Zyklus zu

$$(25)\quad W_j^{MG\text{-}V} = \sum_{k=1}^{j} (W_k^R + W_k^A + W_k^V + W_k^S + W_k^P) + W_0^E$$

Ein Skalarprodukt für das umgebende CG-Verfahren ist mit

$$(26)\quad W_k^{SC} = \max_p(\{\#D_k^p\}\, T_{SC} + \log p)\, C_{SC}$$

anzusetzen, wenn die Teilergebnisse in einer Baumstruktur mit p Prozessoren zusammengefaßt werden. Dies ist der einzige Teil des Verfahrens, in dem wegen der unterschiedlichen Anordnung der Teilsummen die sequentielle und die parallele Implementierung verschiedene numerische Ergebnisse liefern können.

Wenn die Prozessoren logisch als Hypercube vernetzt sind und jeder gleichzeitig senden und empfangen kann, so genügen $\log_2 p$ Kommunikationsschritte, um das Ergebnis parallel zu berechnen und zu verteilen. Wenn Sende- und Empfangs-

operationen hintereinander ausgeführt werden müssen, kann man einen binären Baum zum Berechnen des Skalarproduktes und denselben Baum in umgekehrter Richtung zum Verteilen des Ergebnisses verwenden, was zur doppelten Zahl von Kommunikationsschritten führt. Damit kostet insgesamt ein BPX-vorkonditionierter CG-Schritt

$$(27)\qquad W_j^{CG} = \sum_{k=1}^{j} (W_k^R + W_k^P) + W_0^E + W_j^A + 2\, W_j^{SC} + W_j^V,$$

wie auch aus dem Vergleich in Tabelle 3.3.2. zu sehen ist.

Um damit einen mit der hierarchischen Basis vorkonditionierten CG-Schritt abschätzen zu können, ersetzen wir in den Definitionen von W_k^R und W_k^P alle Terme der Form D_i^p durch $(D_i^p \setminus D_{i-1}^p)$ mit der Konvention $D_{-1}^p = \varnothing$.

Ordnung, 1 Iterationsschritt	skalar $p=1$	schwach parallel $p << n$	massiv parallel $p \approx n$
BPX, geometrisches Wachstum, $\log n \approx j$	$O(n)$	$O(n/p + \log n)$	$O(\log n)$
BPX, arithmetisches Wachstum, $n \approx j$	$O(n)$	$O(j) = O(n)$	$O(\log n)$ für $p \approx n^2$, sonst $O(n)$
2 Skalarprodukte	$O(n)$	$O(n/p)$	$O(\log n)$
Vektoradditionen	$O(n)$	$O(n/p)$	$O(1)$

Tab. 3.3.1 *Ordnung der Teile des Lösers*

3.4. Abgeleitete Kostenfunktionale

Wir werden nun einige Varianten des Kostenfunktionals diskutieren, die das Verhalten bestimmter Parallelrechner besser modellieren.

Für Parallelrechner mit gemeinsamem Speicher spielen die Konstanten C die Rolle von Verlangsamungen durch Benutzung des gemeinsamen Speichers und den daraus entstehenden Speicherzugriffskonflikten. Sie werden auch teilweise mit dem allgemeineren Begriff der Speicherbankkonflikte bezeichnet, die aber auch bei ungünstiger Ausrichtung der Daten im Speicher auftreten. Der bremsende Einfluß ist proportional zur Größe der Ränder der $\{ D_k^p \}$ und teilweise auch proportional zu p.

Für Parallelrechner mit verteiltem Speicher sind die Konstanten C die Zeit, die für Sende- und Empfangsoperationen für die Randdaten benötigt wird. Die reinen Übertragungskosten sind, von Einflüssen der kleinsten Übertragungseinheiten abgesehen, proportional zur Menge der Daten. Meistens sind aber auf realen Parallelrechnern die Initialisierungskosten höher als die auftretenden eigentlichen Übertragungskosten, so daß das Modell durch Ersetzen der Zahl der Randpunkte der Form

$$\#(\mathrm{supp}(Op \,|\, D_i^p) \setminus D_j^p)$$

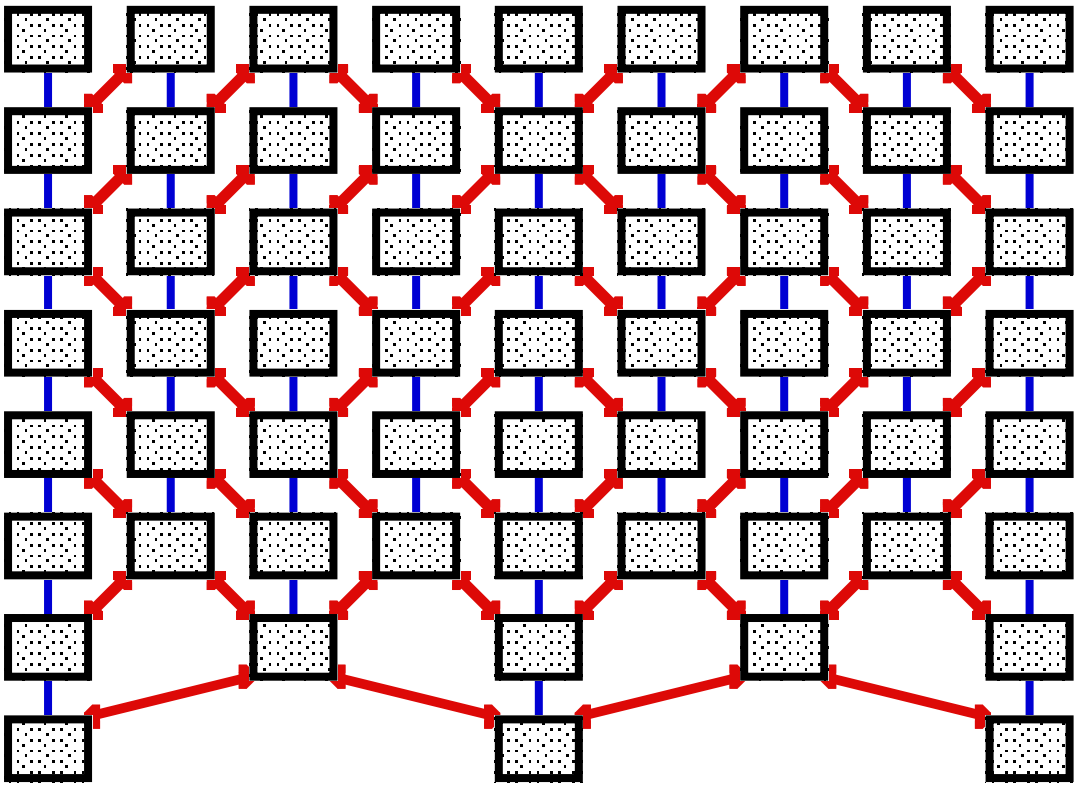
durch die Zahl der Nachbarprozessoren $\#N^p_{j,Op}$ mit

$$(28) \quad N^p_{j,Op} := \{\, p' : \text{mit } (\mathrm{supp}(Op \,|\, D_i^p) \setminus D_j^p) \cap D_j^{p'} \neq \varnothing \,\},$$

die für eine lokale Kommunikationsstruktur beschränkt ist, mit anderen C realistischer wird. Diese Zahl der Nachbarprozessoren wird für ein zweidimensionales Prozessornetz mindestens 6 sein. Wenn die Zerlegung der Ω_k nicht disjunkt ist, also

$$(29) \quad D_k^i \cap D_k^j = \varnothing \text{ für } i \neq j$$

verletzt ist, bleiben die obigen Ausdrücke weiterhin gültig. Das könnte insbesondere dann von Interesse sein, wenn auf den gröbsten Ω_k einige Werte schneller berechnet als gesendet und empfangen werden können. Dazu müssen natürlich dann auch die dazu nötigen Randdaten schnell verfügbar sein, was im allgemeinen aber nicht der Fall ist.



Kapitel 4
Ein Beispielprogramm

Bevor wir versuchen, die allgemeinen Kostenfunktionale W^{MG} und W^{CG} durch geeignete $\{D_k^p\}$ zu minimieren, folgt zunächst ein einfaches Beispiel für ein Randwertproblem und eine Folge von Gittern. Es wird eine bezüglich einfacher Kriterien optimale Aufteilung der Gitterpunkte und eine Näherung für das Kostenfunktional konstruiert, die in diesem Spezialfall das Laufzeitverhalten gut beschreibt.

4.1. Formulierung

Wir betrachten eine quasiuniforme regulär verfeinerte Triangulierung des Einheitsquadrates. Die Gebiete $\{D_k^p\}$ seien Rechtecke, deren eine Kantenlänge mit der des Quadrates übereinstimmt. Die Prozessoren sind also in einer Kette angeordnet, jeder Prozessor p mit $1<p<p$ hat die Nachbarn p–1 und p+1. Wir verwenden ein BPX-vorkonditioniertes CG-Verfahren, der diskrete Differentialoperator sei durch einen 3×3 Stern charakterisiert. Jede Triangulierung entsteht durch die Zerlegung jedes Dreiecks der vorhergehenden Triangulierung in jeweils 4 kongruente Teildreiecke. Die Triangulierung auf Ebene 0 entsteht dabei aus der Zerlegung eines in zwei Dreiecke geteilten Quadrates. Damit enthält Ω_0 eine Unbekannte. Alle Ω_k können als quadratische Gitter aufgefaßt werden, für die sich dann Restriktion und Prolongation als 3×3 Sterne ergeben.

4.2. Parallelisierung

Die Aufgabe, die rechteckigen $\{D_k^p\}$ zu finden, reduziert sich,wenn wir die Ränder mitzählen, auf die Aufteilung der Zeilen $\{0 ,\dots ,2^{k+1}\}$ von Ω_k in p Intervalle $\{d_k^p\}$ für jede Ebene k. Um die Zwischengitterkommunikation aus W_k^R und W_k^P gering zu halten, soll zwischen p und p+1 für Restriktion und Prolongation jeweils nur eine Zeile in genau einer Richtung übertragen werden. Damit folgt

$$(30)\qquad 2n \in d_k^p \Rightarrow n \in d_{k-1}^p \text{ für } k>0,$$

und damit wird die Kommunikation von Restriktion und Prolongation symmetrisch. Es bleibt noch die Aufteilung des feinsten Gitters Ω_j in $\{d_j^p\}$ zu finden. Zur Minimierung von W_j^A und damit W^{CG} muß gelten

$$(31)\qquad \#d_j^p \in \left(\frac{2^{j+1}+1}{p} - 1 \ , \ \frac{2^{j+1}+1}{p} + 1 \right).$$

Falls $\frac{2^{j+1}+1}{p}$ keine ganze Zahl ist, dann bleiben noch Freiheitsgrade, die aufgewendet werden sollten für

$$(32)\qquad \#d_j^1 \geq \#d_j^n \ \ \forall n, \text{ und}$$
$$\#d_j^p \geq \#d_j^n \ \ \forall n>1,$$

weil am Rand etwas weniger Arbeit für die Prozessoren sowohl an Kommunikation als auch an Rechenoperationen für die Randpunkte des Gebietes anfallen. Die übrigen überschüssigen Zeilen können gleichmäßig auf die inneren Prozessoren verteilt werden.

Aus (30) folgt, daß die Eigenschaften (31) und (32) für alle $\{d_k^p\}$ mit $k \leq j$ gelten.

Damit wird W^{CG} unter allen rechteckigen $\{D_k^p\}$ der Länge Eins minimiert, sobald $k > k_0$, das von p und C_{eff} / T_{eff} abhängt. Für kleinere k ist es sinnvoll, einige $d_k^p = \varnothing$ zu wählen.

4.3. Ergebnisse

Wir werden nun die entstehenden Parallelisierungen analysieren, sowohl die Aufteilungen der Gitter als auch die Werte für Beschleunigung und Effizienz, die in einem späteren Kapitel mit experimentellen Ergebnissen verglichen werden.

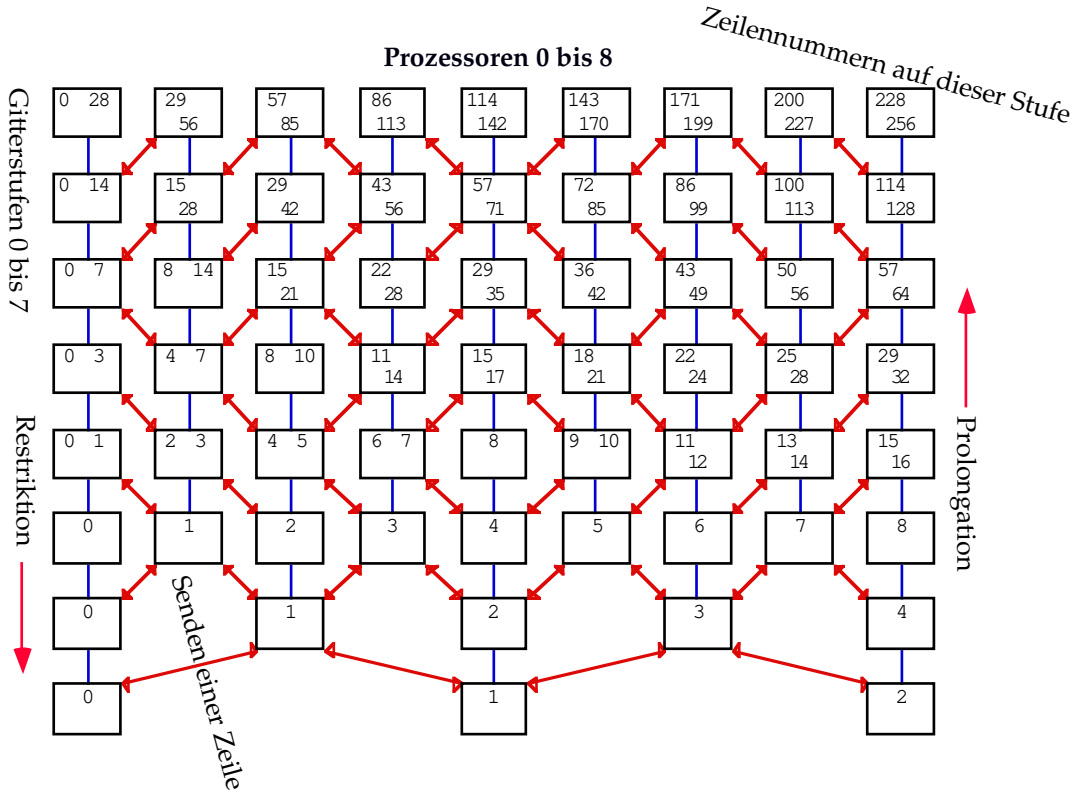


Abb 4.3.1. a *Kommunikation für eine Auswertung des Vorkonditionierers*

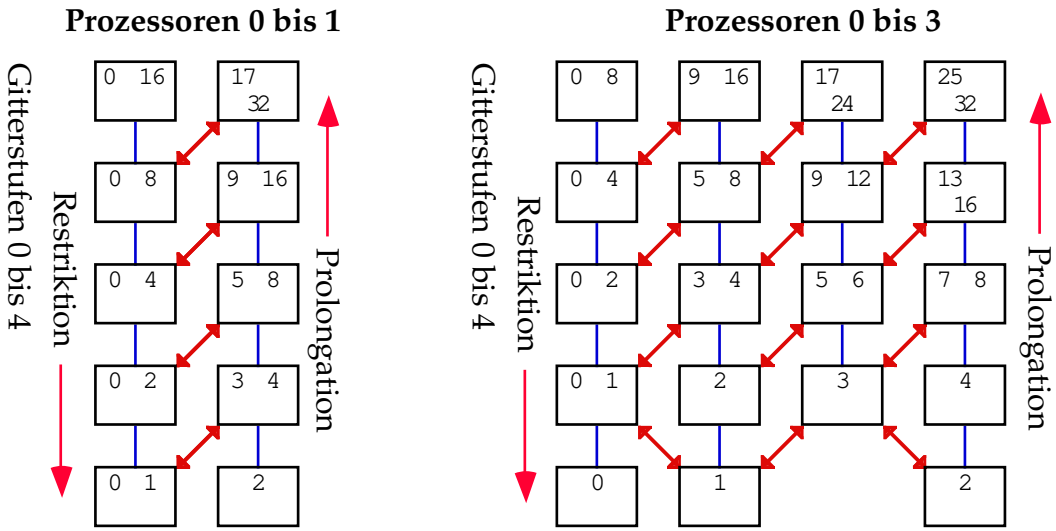


Abb 4.3.1. b,c *Kommunikation für 2 und 4 Prozessoren*

Es folgen Beispiele für solche Aufteilungen der Gebiete. Die Zeilen der Gitterstufen 0 bis 7 werden einmal auf 9 Prozessoren verteilt. Zusätzlich in der Abbildung 4.3.1.a eingezeichnet sind alle nötigen Kommunikationsoperationen für eine Auswertung von $B_7^{BPX}u$. Ein Pfeil zeigt dabei den Transfer einer Randzeile an. Deutlich zu sehen ist die Symmetrie von Restriktion und Prolongation und der Leerlauf einiger Prozessoren auf den beiden gröbsten Gitterebenen.

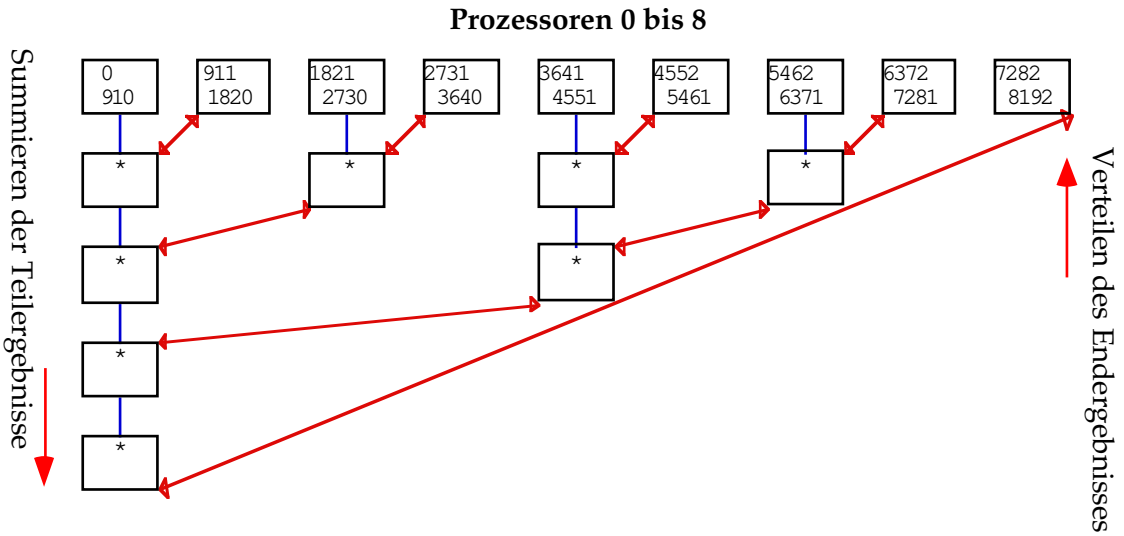


Abb 4.3.2. *Kommunikation für das Skalarprodukt mit binärem Baum*

In den Abbildungen 4.3.1.b,c wurden die Zeilen der Gitterstufen 0 bis 4 auf 2 und auf 4 Prozessoren verteilt. Man erkennt deutlich die für Zweierpotenzen von Prozessorzahlen auf feinen Gittern übliche, völlig

regelmäßige Kommunikationsstruktur.

Zum Vergleich dazu ist in Abbildung 4.3.2 die Kommunikationsstruktur für die Auswertung eines Skalarproduktes mit 9 Prozessoren und der Hilfe eines binären Baumes dargestellt. Auch hier ist der Leerlauf einiger Prozessoren während der Operation zu beobachten. Die einzelnen Zeilen deuten die Zeitschritte an, in denen die Kommunikation abläuft. Das Zusammenfassen der einzelnen Teilergebnisse auf einem Prozessor muß dabei nicht immer in der angegebenen Reihenfolge stattfinden und kann bei entsprechender Programmierung zu einem nichtdeterministischen Einfluß von Rundungsfehlern führen.

Für einen CG-Schritt sind auf einem Rechner mit verteiltem Speicher $(2j + 2 + 4\log_2 p)$ Sende- und Empfangsoperationen nötig. Es kann asymptotisch eine Geschwindigkeitssteigerung um

$$(33) \quad Sp^{CG}_{distr} = \frac{p}{1 + p(1 + j + 2\log_2 p)\,4^{-j}\,C_{eff} / T_{eff}}$$

und damit eine Effizienz

$$(34) \quad Eff^{CG}_{distr} = (1 + p(1 + j + 2\log_2 p)\,4^{-j}\,C_{eff} / T_{eff})^{-1}$$

erreicht werden. Die höchste Ausführungsgeschwindigkeit ergibt sich damit für eine Rechnung mit

$$(35) \quad p^{CG}_{opt} = 4^j \ln 2 \; T_{eff} / C_{eff}$$

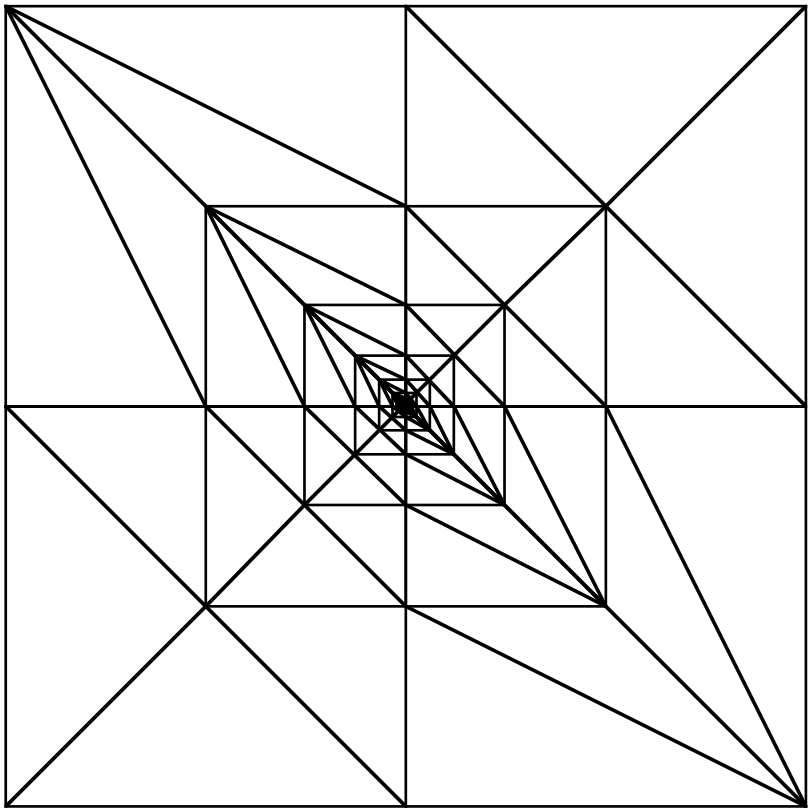
Prozessoren. Für einen Rechner mit gemeinsamem Speicher ergeben sich die Werte zu

$$(36) \quad Sp^{CG}_{shared} = \frac{p}{1 + p\,2^{-j}\,C_{eff} / T_{eff}}$$

und

$$(37) \quad Eff^{CG}_{shared} = (1 + p\,2^{-j}\,C_{eff} / T_{eff})^{-1}$$

für kleine p, solange der Einfluß von $(\log_2 p)$ der Skalarprodukte vernachlässigbar bleibt, was bei realen Rechnern mit gemeinsamem Speicher aber stets gewährleistet ist. Der Term $(p\,2^{-j})$ beschreibt dabei den Anteil der Ränder an $\{D^p_k\}$. Diese Werte, genauso wie die Werte der genaueren Modelle für W^{CG}_j, konnten in Experimenten mit guter Übereinstimmung gemessen werden, wie auch die Ergebnisse am Ende dieses Textes zeigen.



Kapitel 5
Adaptive parallele Multilevel-Methoden

Wir werden nun die übrigen Komponenten eines Finite-Elemente Programms und deren Parallelisierung, in das der iterative Löser eingebettet ist, analysieren.

Das gesamte Lösungsverfahren besteht aus

- Fehlerschätzer,
- Gittermanipulation und
- iterativem Löser mit
  - multilevel Vorkonditionierer und
  - direktem Grobgitterlöser.

Die Verwaltung der Gitter erzeugt die Datenstrukturen, mit denen der Löser arbeitet, und sollte nur einen geringen Einfluß auf den Gesamtaufwand des Verfahrens haben. Daher ist eine gute Parallelsierung dieser Komponenten für Rechner mit großen Prozessorzahlen wichtig.

5.1. Fehlerschätzer

Wir betrachten nun die Parallelisierung von Fehlerschätzern.

Das Finite-Elemente-Verfahren startet mit einer vorgegebenen Anfangstriangulierung τ_0. Die feineren Triangulierungen τ_k werden adaptiv erzeugt. Üblich sind lokale Fehlerschätzer, die einzelne Punkte, Kanten oder Elemente markieren, die dann anschließend verfeinert werden. Diese Fehlerschätzer arbeiten mit lokalen Schätzungen für Ableitungen, asymptotischen Fehlerentwicklungen für bestehende [Bank, Weiser] oder neu zu erzeugende Elemente [Babuska & Rheinboldt] oder Näherungslösungen in einem quadratischen Finite-Elemente-Ansatzraum [Deuflhard, Leinen, Yserentant] oder feineren linearen FE-Räumen. Alle genannten lokalen Fehlerschätzer sind mit ihrer lokalen Kommunikationsstruktur ähnlich einem Schritt des CG-Verfahrens parallelisierbar.

Um aus dem Fehlerschätzer einen Fehlerindikator zu gewinnen, der zu verfeinernde Punkte, Kanten oder Elemente markiert, wird häufig der lokale geschätzte Fehler mit einer Kombination aus globalem Maximum und Mittelwert aller geschätzten Fehler der aktuellen oder aller bisherigen Verfeinerungsstufen verglichen (siehe auch [Bornemann, Erdmann & Roitzsch]). Das erfordert eine globale Kommunikation, die die gleiche Komplexität $O(\log p)$ besitzt wie ein Skalarprodukt.

Insgesamt gilt damit $O(W^{Fehler}) \leq O(W^{CG})$.

5.2. Gittermanipulation

Die Erzeugung der neuen Gitter ist im allgemeinen nicht so problemlos

zu parallelsieren. Wir werden also auch Änderungen an bestehenden Finite-Elemente-Programmen in Betracht ziehen müssen.

Im Gittermanipulationsteil muß eine gültige neue Triangulierung τ_j erzeugt werden. Zweckmäßigerweise wird dabei für die statische Aufteilung der Gebiete auch die Verteilung der neuen und die Umverteilung der alten Elemente $\{D_j^p\}$ durchgeführt.

Legt man die Verfeinerungsregeln von [Bank] für die Gitterverfeinerung um eine Stufe zugrunde, so werden Dreiecke mit zwei zu verfeinernden Kanten in 4 kongruente Dreiecke (reguläre, rote Verfeinerung, Abbildung 5.2.1a), Dreiecke mit

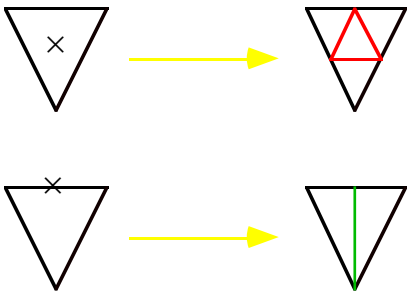


Abb 5.2.1 b. *rote und grüne Verfeinerung*

nur einer zu verfeinernden Kante in 2 Dreiecke zerlegt (grüne Verfeinerung, 5.2.1b). Auf einem sequentiellen Rechner läßt sich bei geeigneter lokaler Programmierung die neue Triangulierung τ_{j+1} in $O(\#\tau_{j+1} - \#\tau_j)$ erzeugen [Leinen, Kap. 3.3].

Auf einem parallelen Rechner mit verteiltem Speicher kann dagegen eine Situation wie in Abbildung 5.2.2 eintreten. Die zu verfeinernden Kanten sind in der Abbildung 5.2.2 mit „×“ gekennzeichnet. Abhängig von der Verfeinerung der Kante k werden damit alle abgebildeten Dreiecke halbiert oder in 4 kongruente Dreiecke zerlegt. Dazu müssen zwischen den Prozessoren p_1 und p_2 im schlimmsten Fall $\#\{D_j^{p1} \cap D_j^{p2}\}$ Botschaften ausgetauscht werden.

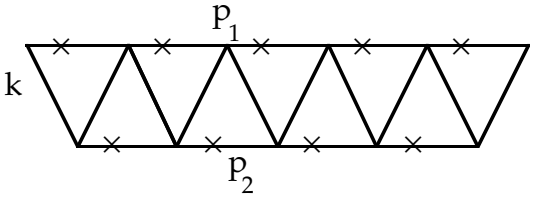


Abb 5.2.2. *Triangulierung mit markierten Kanten*

Auf Parallelrechnern mit gemeinsamem Speicher müssen entsprechend viele Synchronisierungen durchgeführt werden. Die gleichen Probleme entstehen auch mit der Strategie, Dreiecke mit markierter Kante als vollständig zu verfeinernd zu markieren und darauf Algorithmen zum regelkonformen Abschluß der Triangulierung anzuwenden. Die grünen Triangulierungen führen im ungünstigsten Fall zu

$$(38) \qquad W_j^{Gitter} \geq \max_p \{ \#D_{j+1}^p \, T_G \; +$$

$$\#(\mathrm{supp}(r \mid D_j^p) \setminus D_{j+1}^p) \, C_G^{init} \Big\}$$

mit den Initialisierungskosten C_G^{init} für eine Kommunikation, was auf realen Parallelrechnern um Ordnungen größer ist als der Kommunikationsaufwand von $W_j^{C\,G}$. Es muß daher nach anderen Verfahren gesucht werden, so daß nicht der Gitterverwaltungsteil die eigentliche Rechenzeit dominiert. Eine Möglichkeit, $W_j^{Gitter} << W_j^{C\,G}$ zu erhalten, ist, auf die grüne Triangulierung zu verzichten und statt dessen alle Verfeinerungen von Dreiecken regulär durchzuführen. Dabei können gebundene innere Knoten entstehen, deren Werte durch Interpolation bestimmt werden und die keinen Freiheitsgrad darstellen. Damit kann sehr einfach

$$(39) \qquad W_j^{Gitter} = \max_p \Big\{ \#D_{j+1}^p \, T_G + \#(\mathrm{supp}(r \mid D_j^p) \setminus D_{j+1}^p) \, C_G^{data} \Big\}$$

erreicht werden, mit den Kosten C_G^{data}, um Daten über die Randelemente den Nachbarprozessoren bekannt zu machen. Für Rechner mit verteiltem Speicher kann dieses mit einem Satz von lokalen Sendeoperationen durchgeführt werden.

Andere Verfeinerungsmethoden, wie etwa die gerichtete blaue Verfeinerung [Kornhuber & Roitzsch], die zwei Dreiecke zusammenfaßt und vier lange Dreiecke entlang von vermuteten Stromlinien erzeugt, siehe Abbildung 5.2.3., können dagegen gut auf Parallelrechnern eingesetzt werden.

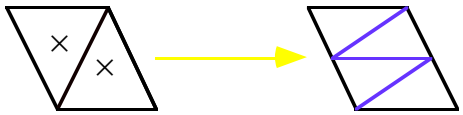


Abb 5.2.3. *blaue Verfeinerung*

In diesem Fall müssen für Nachbardreiecke, die auf verschiedenen Prozessoren liegen, auf genau einem Prozessor Kopien von beiden vorhanden sein. Dieser Prozessor entscheidet dann über die „Farbe“ der Verfeinerung von beiden Dreiecken. Die notwendige Kommunikation ist damit lokal. Wenn man auf die grüne Verfeinerung verzichtet, und nur mit roter und blauer Triangulierung und gebundenen inneren Knoten arbeitet, terminiert der Gittermanipulationsteil in jedem Fall. Damit entfallen die sonst für blaue Triangulierungen nötigen Hilfskonstruktionen zum Abbruch des Verfahrens [Roitzsch & Kornhuber, Anhang].

Vor dem Einsatz der blauen Verfeinerung muß allerdings die Konvergenz des iterativen Lösers sichergestellt sein, sei es durch die Konstruktion von geschachtelten Finite-Elemente-Räumen durch Manipulationen der gröberen Triangulierungen bei blauer Verfeinerung oder durch geeignete Gittertransferoperatoren.

5.3. Grobgitterlöser

Nachdem manche realen Probleme, die mit Finite-Elemente-Programmen behandelt werden, wegen komplexer geometrischer Daten sehr feine Grobgitter erfordern, bleibt die Frage, ob der zu diesen Gittern gehörende direkte Löser durch Parallelisierung auch beschleunigt werden kann. Eine ausführlichere Darstellung direkter Löser auf Skalarrechnern findet sich in [Axelson & Barker, Kap. 6] zusammen mit der dort zitierten Literatur, insbesondere auch [George & Liu].

Entscheidend für die Parallelisierung der direkten Lösungsverfahren ist die Symmetrie der Steifigkeitsmatrix auf dem Grobgitter. Damit entfällt die während der Zerlegung nötige Pivotsuche und das Vertauschen von Matrixzeilen oder -spalten. Die Reihenfolge der Unbekannten, der Rechenoperationen und der Kommunikationsoperationen kann vor der Matrixzerlegung festgelegt werden und damit können sowohl Lastverteilung, als auch Kommunikationsvolumen optimiert werden.

5.3.1 Volle Grobgittermatrizen

Bevor wir spezielle Verfahren für dünnbesetzte Matrizen betrachten, befassen wir uns zunächst mit der Cholesky-Zerlegung für vollbesetzte Matrizen, wie in Abbildung 5.3.1.1 angedeutet.

Für n Unbekannte kostet die Zerlegung auf einem Prozessor $n^3/6$ wesentliche Operationen. Die Lösung der entstandenen Gleichungssysteme in Dreiecksform zur Bestimmung der n Unbekannten hat dagegen nur noch die Ordnung n^2.

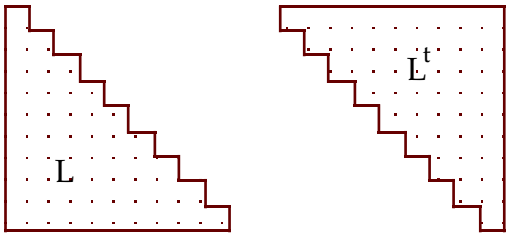


Abb 5.3.1.1. *Cholesky-Zerlegung für 9 Unbekannte*

Die blockweise Aufteilung der Unbekannten in ihrer natürlichen Reihenfolge auf p Prozessoren bei Spalten-Cholesky-Zerlegung wie in Abbildung 5.3.1.2 führt zu Verfahren, die bei jeweils $O(p-1)$ Kommunikationsschritten $n^3/(2p) + n^3/(6p^2)$ Operationen zur Zerlegung und $2n^2/p+n$ Operationen zur Lösung benötigen.

Dabei berechnet der erste Prozessor wie im skalaren Fall die ihm zugeordneten Werte, sendet diese an die übrigen Prozessoren und hat im weiteren nichts mehr zu tun. Die übrigen Prozessoren empfangen des-

sen Werte, führen alle mit den neuen Daten möglichen Operationen aus (frontaler Ansatz) und verfahren mit der verbleibenden Restmatrix, soweit vorhanden, so wie vorher alle Prozessoren mit der Gesamtmatrix. Für die Rücksubstitution während der Lösung sind die Operationen einmal beginnend mit dem letzten Prozessor und in umgekehrter Reihenfolge und anschließend in natürlicher Reihenfolge zu durchlaufen.

Die Daten müssen jeweils nur an den folgenden Prozessor sofort gesendet werden, die übrigen Prozessoren können diese Daten auch erst weitergereicht bekommen, ohne die Ordung der Zahl der Kommunikationsoperationen zu verschlechtern. Damit ist lediglich eine Prozessortopologie eines Ringes oder einer Kette nötig. Diese Topologie ist auf allen praktischen Parallelrechnern möglich. Der iterative Löser stellt dagegen stärkere Forderungen an die Prozessortopologie.

Nachdem die blockweise Aufteilung der Matrix zu einer unausgeglichenen Rechenlast führt und die ersten Prozessoren sofort mit der Arbeit aufhören, sind für größere n andere Aufteilungen nötig. Abbildung 5.3.1.3 zeigt die spaltenweise Aufteilung der Matrix, also die Verteilung der Unbekannten der Reihe nach auf die Prozessoren, mit Wiederholung, falls p größer n ist („wrap around“) [Bader & Gehrke], [Nölting].

Die Matrixoperationen benötigen in dieser Aufteilung zwar O(n) Kommunikationsoperationen, die entstehende Rechenlast ist aber so auf die Prozessoren verteilt, daß jeder Prozessor mindestens bis zur Zeile $(n–n/p)$ und mit einer Zahl von Matrixelementen arbeitet, die insgesamt um weniger als n kleiner der des ersten Prozessors ist.

Damit sind für die Cholesky-Zerlegung $n^3/(2p)$ Rechenoperationen und für die Lösung des Systems $n^2/p+2n$ Operationen nötig.

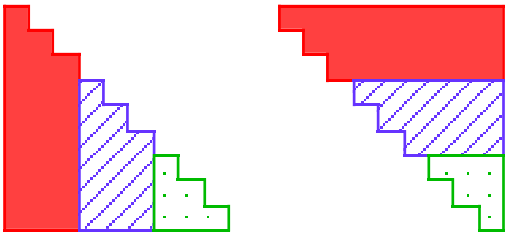


Abb 5.3.1.2. *Blockweise Parallelisierung der Cholesky-Zerlegung*

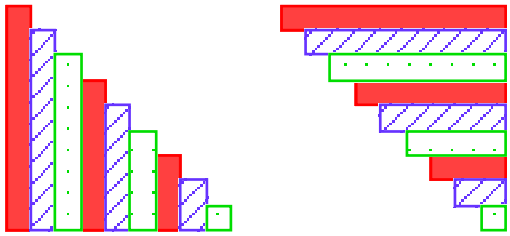


Abb 5.3.1.3. *Zeilenweise Parallelisierung mit 3 Prozessoren*

Die optimale Wahl von p zwischen 1 und n und der Blockgröße zwischen n/p und 1 hängt von dem verwendeten Parallelrechner, aber im wesentlichen vom Verhältnis von Kommunikationsleistung zu Rechenleistung, und der Zahl der Unbekannten ab. Grundsätzlich ist aber die besprochene Reihenfolge (Abbildungen 5.3.1. 1–3) für steigende Zahlen n von Unbekannten typisch. Für die effiziente Lösung des Gleichungssystems ist eine Blockgröße in der Größenordnung von

$$(40) \quad blk_{lsg} = \sqrt{2\, T_{eff} / C_{eff}}$$

sinnvoll, solange die Zahl der Blöcke kleiner als die Zahl der Prozessoren ist. Die Blöcke für die Cholesky-Zerlegung selber sollten in der Größenordnung von

$$(41) \quad blk_{ch} = \sqrt[3]{3\, T_{eff} / C_{eff}}$$

sein. Da die Matrix aber nur in einer Anordung aufgeteilt werden sollte, wird, abhängig von der Zahl der notwendigen Lösungen auf dem Grobgitter, die verwendete Blockgröße ein gewichtetes Mittel aus (40) und (41) sein. Wenn aus anderen Programmteilen so viele Prozessoren wie Matrixblöcke vorhanden sind, die sonst arbeitslos wären, dann sinkt die optimale Blockgröße mit steigender Zahl der Unbekannten n.

5.3.2 Dünne Grobgittermatrizen

Wenn die Zahlen der Unbekannten n sehr groß werden, können auch Verfahren für dünnbesetzte Matrizen, Hüllenmatrizen oder Bandmatrizen sinnvoll zum Einsatz kommen. Bei sequentieller Ausführung kann damit die Ordnung der Zerlegung auf bestenfalls $O(n^{3/2})$ und die des Lösers auf $O(n \log n)$ verbessert werden für Probleme in zwei Raumdimensionen.

	2 dim.	3 dim.
vollbesetzt	n^3	n^3
	n^2	n^2
Band	n^2	$n^{2\,1/3}$
	$n^{3/2}$	$n^{1\,2/3}$
nested-dissection	$n^{3/2}$	n^2
	$n \log n$	$n^{1\,1/3}$

Tab 5.3.2.1. *Ordnung direkter Löser, jeweils Zerlegung und Lösung*

Tabelle 5.3.2.1 gibt einen Überblick über den Aufwand für Matrixzerlegung und Lösung des entstehenden Gleichungssystems für ein quadratisches reguläres Gitter. Die Zeilen bezeichnen volle Matrizen, Matrizen mit minimaler Bandbreite und Matrizen mit Anordung der Unbekannten in der Reihenfolge von nested-disection. Die Werte sind für Randwertprobleme in 2 und in 3

Raumdimensionen angegeben (aus [Axelson & Barker, Kap. 7.5]).

Die angegebenen Werte sind typisch für reguläre Triangulierungen, bei denen die Verteilung der Unbekannten gleichmäßig in allen Raumdimensionen erfolgt. Wenn in einer Koordinatenrichtung mehr Zeilen von Dreiecken auftreten als in den übrigen, kann man das Gebiet senkrecht dazu in Gebiete mit kleineren Rändern zerlegen und damit effizientere Löser konstruieren.

Für die Parallelisierung analysieren wir die einzelnen Komponenten des direkten Lösers: Die Reihenfolge der Unbekannten wird aus der Geometrie des Problems oder dem Besetzungsmuster der Steifigkeitsmatrix bestimmt, so daß die Speicherung kompakt aber auch das „fill in“ gering ist, dann wird die dafür geeignete Datenstruktur gewählt, wie Bandmatrizen oder Hüllenmatrizen mit konstanten oder variablen Besetzungsmustern, und zuletzt wird die Aufteilung dieser Daten auf die Prozessoren festgelegt.

Die optimale Anordnung der Unbekannten hängt vom Ausgangsproblem ab. Sind geometrische Eigenschaften der Ausgangstriangulierung bekannt, so können diese für eine Zerlegung des Gebietes in Teilgebiete oder zur Bildung von Superelementen und einer entsprechenden Anordnung führen. Da damit die Reihenfolge der Unbekannten a priori feststeht, können sie vor der Rechnung auf die Prozessoren verteilt werden.

Durch die Parallelisierung wird dabei aber kein größerer Ordnungsgewinn als im Fall der Parallelisierung der vollbesetzten Matrizen erreicht werden. Diese Fassung des Grobgitterlösers ist trotzdem selbst massivparallel der besser zu parallelisierenden Fassung für vollbesetzte Matrizen vorzuziehen, da bei dünnbesetzten Matrizen weniger Daten übertragen werden müssen, und daher eine höhere Ausführungsgeschwindigkeit zu erwarten ist. Bei kleinen Prozessorzahlen können auch zusätzlich Rechenoperationen durch die dünnbesetzten Matrizen eingespart werden.

Für ein quadratisches Gitter in zwei Raumdimensionen kann beispielsweise eine Matrixzerlegung in der Reihenfolge von nested-disection auf p Prozessoren (Abbildung 5.3.2.2.) mit höchstens $O(\log n \log p)$ Kommunikationsoperationen, $183/4\ n$ übertragenen Zahlen und $O(n^{3/2}/p)$ Rechenoperationen je Prozessor ausgeführt werden [George, Heath, Liu & Ng], [George, Liu & Ng].

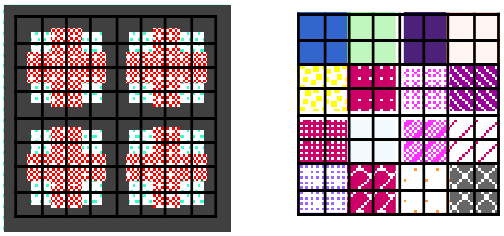
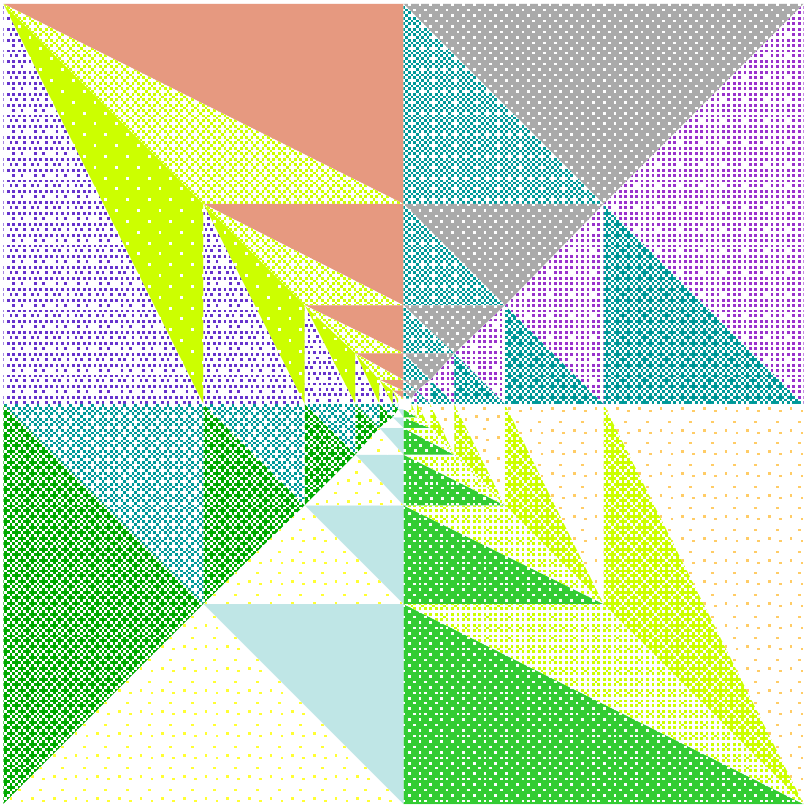


Abb 5.3.2.2. *nested-disection für 9×9 Gitter und 16 Prozessoren*

Verfahren, die erst dynamisch eine Anordnung der Unbekannten des Grobgitters aufbauen, werden dagegen grundsätzliche Probleme bei der Parallelisierung durch die notwendige Umordnung der Daten aufwerfen. Dabei ist zu überlegen, ob beim sequentiellen Einlesen der Grobgittergeometriedaten auf einem Prozessor nicht auch gleichzeitig Verfahren wie der umgekehrte Cuthill-McKee Algorithmus und das Verfahren von Gibbs, Poole und Stockmeyer [siehe Axelson & Barker, Kap. 6.1] ausgeführt werden können, so daß das Umordnen der Daten zwischen den Prozessoren entfällt. Die geordneten Daten werden dann auf die übrigen Prozessoren verteilt. Mit der so gewonnenen Anordnung der Unbekannten kann nun verfahren werden, als ob sie a priori bekannt wäre.

Insgesamt werden dann nur das Einlesen der Geometriedaten und die Bestimmung der Anordnung sequentiell bearbeitet, während der übrige Teil des Programms parallel abgearbeitet werden kann.



Kapitel 6
Lastverteilung adaptiver paralleler Multilevel-Methoden

Wir kommen nun zu verschiedenen Strategien, für allgemeinere Gebiete $\{\Omega_k\}$ gute Aufteilungen in $\{D_k^p\}$ zu finden, die die Ausführungszeiten des iterativen Lösers minimieren sollen. Dabei werden wir dynamische Scheduling-Algorithmen mit Verfahren für statische Aufteilungen mit den Mitteln der kombinatorischen Optimierung vergleichen. Wir werden eine Klasse von Gitterverfeinerungen konstruieren, die eine effiziente Verteilung auf mehr als eine beschränkte Zahl von Prozessoren unmöglich macht. Daraus leiten wir für große Prozessorzahlen Bedingungen an das Wachstum der Dimensionen der Finite-Elemente-Räume ab. Wenn diese nicht erfüllt werden, können wir bezüglich schwächerer Kriterien mit einigen Erweiterungen der Aufteilungsverfahren immer noch gute Aufteilungen erhalten. Zuletzt werden wir noch ein neues Lastverteilungsverfahren konstruieren, dessen Komplexität im Gegensatz zu den anderen vorgestellten Verfahren nicht größer als der des iterativen Lösers ist, das sich aber an die Strukturen verschiedener Parallelrechner und Löser anpaßt.

6.1. Dynamische Ansätze

Lastverteilungsverfahren, die die Aufteilung der Punkte dynamisch während der iterativen Lösung ständig neu erzeugen, beispielsweise durch Scheduling kleiner Gitterteile [Leinen], oder auch [Fox, Kolawa & Williams], sind für große Prozessorzahlen p ungeeignet, da der Scheduling-Prozeß die Lokalität der Kommunikationsstruktur durchbricht. Es muß also einen zentralen Prozeß geben, der mit allen anderen kommuniziert. Die Gitterteile müssen für den Mehrgitter V-Zyklus im allgemeinen für jede Ebene k getrennt bearbeitet werden. Dagegen können für die Vorkonditionierer auch ganze Hierarchien von Elementen zu einem Gitterteil zusammengefaßt werden, so daß für eine grobe Lastaufteilung teilweise schon eine Aufteilung in die Elemente der gröbsten Anfangstriangulierung genügt. Feinere Aufteilungen können auf einer der Prozessorzahl p angemessenen feineren Triangulierung durchgeführt werden, wobei auf den gröberen Gittern ein anderes Verteilungsverfahren zum Einsatz kommt.

Um die globale Kommunikation und den damit verbundenen zentralen Scheduler zu umgehen, bieten sich Hierarchien von Schedulern an, wie sie z.B. als Loadbalancer in CDL [Veer] üblich sind. Auf Parallelrechnern mit verteiltem Speicher ist

ein solches Verfahren ungünstig, weil die gesamten Daten des Problems über mehrere Zwischenstufen transportiert werden müssen, was den zentralen Scheduler-Prozeß nach wie vor zum Flaschenhals macht. Im übrigen müssen alle Daten im Speicher eines Prozessors Platz finden, was dem Prinzip der Datenparallelität widerspricht und was bei großen Problemen nicht der Fall sein muß.

Für Rechner mit gemeinsamem Speicher, für die p naturgemäß beschränkt ist und kein Datentransfer sondern nur Steuerinformation über die Gitterhierarchien nötig ist, ist dieses Verfahren aber durchaus möglich und liefert, abhängig von der Granularität der Unterteilung in Gitterteile, verschieden gute Werte. So wird man die Gitterebene k, auf der unterteilt wird, in Abhängigkeit von p und der zu erzielenden Effizienz wählen, so daß für die Zahl der Elemente der Triangulierung der Ebene k, oder in einer Näherung die Zahl der Gitterpunkte $\#\Omega_k$ der Ebene k, ungefähr gilt

$$(42) \qquad \#\tau_k \geq \frac{p}{1 - \text{Effizienz}}$$

mit der Effizienz <1.

6.2. Der schlechteste Fall für statische Ansätze

Nachdem für Rechner mit verteiltem Speicher realistischerweise nur noch eine statische Aufteilung der Gitter in Frage kommt, wollen wir untersuchen, ob das unter den bisher gemachten Voraussetzungen überhaupt sinnvoll ist.

Es bleibt also die Alternative einer statischen Aufteilung für eine Folge von Triangulierungen $\{ \tau_k \}_{k=0,\ldots,j}$. Wird durch Verfeinerung ein neues τ_{j+1} erzeugt, so muß ein neuer Satz von $\{ \overline{D}_k^p \}_{k=0,\ldots,j+1}$ erzeugt werden, der W_{j+1} näherungsweise minimiert. Für Rechner mit verteiltem Speicher und große p sind für gute Abschätzungen des Kostenfunktionals die oberen Schranken der Zahl der Nachbarprozessoren $\{\#N^P_{k,Op}\}$ definiert als

$$(43) \qquad N^p_{j,Op} := \{\, p' : \text{mit } (\text{supp}(\, Op \,|\, D^p_i \,) \setminus D^p_j \,) \cap D^{p'}_j \neq \varnothing \,\},$$
$$\#N^P_{k,Op} \leq N_{Op} \quad \forall p,k$$

wichtig, die auch als Grad der Graphen der jeweiligen Operatoren bezeichnet werden. Für kleine N_{Op} im Verhältniss zu p liegt eine lokale Kommunikationsstruktur vor. Sei diese im folgenden vorausgesetzt.

6.2.1. Das Problem

Wir betrachten folgende Verfeinerung: Alle Dreiecke von $\overline{D}_j^p$ werden

verfeinert, die übrigen Dreiecke von Ω_j dagegen nicht.

6.2.2. Ansätze

(i) Ist $\overline{D}^p_k = D^p_k$ für alle $k \le j$, so kann Ω_{j+1} höchstens auf N_r+1 Prozessoren verteilt werden. Für jeden dieser Prozessoren gilt

$$\#\,\overline{D}^p_{j+1} \approx \frac{3\,\#D^p_j}{N_r+1} + \#\,\overline{D}^p_j .$$

Mit Wiederholung dieser Verfeinerung, erhält man für $j\to\infty$ eine Aufteilung der Last auf höchstens N_r+1 Prozessoren, also eine maximale Beschleunigung $Sp^{CG} \le N_r+1$ unabhängig von p. Dabei wächst $\{\dim V_k\}$ höchstens geometrisch mit dem Faktor $(1+\frac{3}{N_r+1})$.

(ii) Für $\overline{D}^p_k \neq D^p_k$ wird bei optimaler Lastaufteilung auf Ω_{j+1}, $j\to\infty$ und globaler Kommunikation erreicht, daß $N_r^p \to p$. Für lokale Kommunikation kann die Last auf Ω_{j+1} auch noch optimal verteilt werden, die Aufteilungen auf $\{\Omega_k\}_{k=j,\dots,0}$ werden aber exponentiell schlechter, was im ungünstigsten Fall, für $p > O(N_r^2)$ und zusammenhängende Gebiete $\{D^p_k\}$ dazu führen kann, daß dann $W^{CG}_k \approx W^{CG}_{j+1}\ \forall k$ gilt.

Für Ω aus $\Re^3$ ist entsprechend $p > O(N_r^3)$ zu setzen.

Damit entstehen, unabhängig von der verwendeten Lösungsstrategie, denkbar schlechte Beschleunigungen, deren Wert unabhängig von der Zahl der eingesetzten Prozessoren nur von der Kommunikationsstruktur der Prozessoren abhängt.

6.2.3. Zusätzliche Voraussetzungen

Wir werden nun Möglichkeiten diskutieren, die die Problemstellung der statischen Lastverteilung doch noch sinnvoll machen.

Die diskutierte ungünstige Lastverteilung kann grundsätzlich ausgeschlossen werden, wenn für das geometrische Wachstum der $\{\dim V_k\}$ gilt

$$(44) \qquad \dim V_{k+1} > (1+\frac{3}{N_r+1})\ \dim V_k .$$

Für geringeres Wachstum bleibt noch die Möglichkeit, nicht zusammenhängende Gebiete $\{D^p_k\}$ zu verwenden, die einen größeren Kommunikationsaufwand für die Randdaten erfordern.

Die Topologie der logischen Verbindungen zwischen den einzelnen Prozessoren, die für die geforderte lokale Kommunikation üblicherweise ein regelmäßiges zweidimensionales Netz ist, kann auch so gestaltet werden, daß eine baumartige Struktur eingebettet wird. Damit verläuft der Abfall der Auslastung der Prozessoren auf den einzelnen Ebenen nicht mehr exponentiell.

Für dieses Vorgehen bieten sich quintäre Bäume an, die die Daten schneller übertragen können als durch Verfeinerung des Gitters je Ebene neue Punkte und damit Daten entstehen. Diese Struktur ist auch für beschränkte N_r, $P \to \infty$ möglich und erfordert je Prozessor höchstens 6 weitere Verbindungen zu anderen Prozessoren.

6.3. Übersicht über statische Ansätze

Unter der Voraussetzung, daß die Problemstellung der statischen Lastverteilung sinnvoll ist, wollen wir nun verschiedene Standardansätze betrachten und feststellen, daß diese für Multilevel-Verfahren nur bedingt geeignet sind.

Im Fall der hierarchischen Vorkonditionierung muß nicht $\{ D_k^p \}$ verteilt werden, sondern $\{D_k^p \setminus D_{k-1}^p \}$, was durch die Kopplung der einzelnen Gitter Ω_k durch lokale Kommunikation aber zu denselben Ergebnissen führt.

Sei im folgenden ein geometrisches Wachstum der $\{\dim V_k\}$ mit einem Faktor größer $(1 + \frac{3}{N_r+1})$ angenommen. Gesucht ist eine Verteilung

(45) $\{ \overline{D}_k^p \}_{k=0,\dots,j}$, die W_j^{CG} minimiert, mit der Nebenbedingung, daß N_r beschränkt ist für $j \to \infty$. Diese Verteilung sollte in

(46) $$O(W_j^{CG}) = O(\frac{\dim V_j}{p} + \log p)$$

Zeit auf einem Parallelrechner, einschließlich aller nötigen Datentransfers, geschehen können. Es ist zu teuer, die exakte Lösung des Minimierungsproblems zu berechnen, da das Problem im allgemeinen NP-vollständig ist.

Es bieten sich Heuristiken der kombinatorischen Optimierung zur näherungsweisen Minimierung an, die ausgehend von der Verteilung $\{D_k^p\}_{k=0,\dots,j-1}$ in einer Art von Diffusionsprozeß eine gute Näherung für die gesuchte Verteilung liefern, wie z.B. Ansätze mit „simulated annealing“ [Fox & Otto, Kap. 4] oder auch Metaheuristiken wie „tabu search“ [Glover] zur geschickten lokalen Minimierung von W_j^{CG}. Diese Ansätze liefern im allgemeinen Verfahren von deutlich höherer Ordnung als gesucht.

Andere Verfahren arbeiten mit rekursiver Bisektion des Gebietes Ω_j, das von zwei entgegengesetzt liegenden Gitterpunkten ausgehend bezüglich von Abstandsmaßen und Maßen für die Zahl von Verbindungen in zwei gleich große Teile geteilt wird, und verteilen dann in einem weiteren Schritt diese Teilgebiete möglichst gut auf die gegebene

Prozessortopologie [Bastian], [Berger & Bokhari], [Literatur zu MinCut]. Diese Verfahren sind mit einem Aufwand $\geq O(\log p \dim V_j)$ nicht nur zu teuer, sondern nehmen in dieser Form auch keine Rücksicht auf die Verbindung der verschiedenen Gitter Ω_k untereinander.

Ansätze, die auf einer von den Triangulierungen zunächst unabhängigen, a priori wählbaren Aufteilung der Gebiete beruht, scattered decomposition [Fox & Otto, Kap. 4.6], sind vom Aufwand her billiger. Wenn sich die Aufteilung an der Maschenweite des gröbsten Gitters orientiert, sind die Aufteilungen für große Prozessorzahlen im allgemeinen zu grob, so daß einzelne Prozessoren keine Gitterteile erhalten können. Orientiert sich die Verteilung aber an feineren Gittern, so kann an nicht verfeinerten Stellen des Gitters globale Kommunikation notwendig werden. Im übrigen ist die Aufteilung bezüglich der Gebietsränder normalerweise schlecht, während die Last für feine Maschenweiten gut verteilt werden kann. Für adaptive Multilevel-Methoden ist das Verfahren in dieser Form weniger geeignet, obwohl die Komplexität genügen würde.

6.3. Statistischer Ansatz mit geeigneter Komplexität

Nachdem die besprochenen Standardverfahren zur Parallelisierung für Multilevelverfahren nicht optimal geeignet sind, werden wir versuchen, ein an die Struktur mehrerer Gitter und an die Kommunikationsstruktur der Prozessoren angepaßtes Lastverteilungsverfahren zu konstruieren, das keine größere Komplexität hat als der iterative Löser selbst und ihn damit vom Aufwand her nicht dominiert.

Wir machen daher im folgenden Aufteilungsverfahren einen statistischen Ansatz. Wir gehen von einer Verteilung des gröbsten Gitters Ω_0 aus. Die Abbildung G ordnet jedem Gitterpunkt einen Prozessor zu. Wie solche Aufteilungen günstig gewählt werden, wurde in Kapitel 5.3 über direkte Grobgitterlöser behandelt.

Wir konstruieren nun induktiv die Punkteverteilungen für die feineren Gitter.

Ausgehend von der Aufteilung von Ω_{j-1} auf p Prozessoren und einer lokale Kommunikationsstruktur als symmetrische, reflexive Relation R zwischen den Prozessoren sei nun eine Abbildung G von $\Omega_j \setminus \Omega_{j-1}$ auf die Menge $\{1, \ldots, p\}$ gesucht. G ist bereits auf Ω_{j-1} definiert. Für ein $g \in \Omega_j \setminus \Omega_{j-1}$, das durch Halbierung einer Kante mit den Endpunkten g_1 und g_2 entstanden ist, gelte

(47) $$\mathbf{P}\left(G(g)=p\right)=\frac{w(p)}{\sum_{R'(p',G(p_1))\wedge R'(p',G(p_2))} w(p')}$$

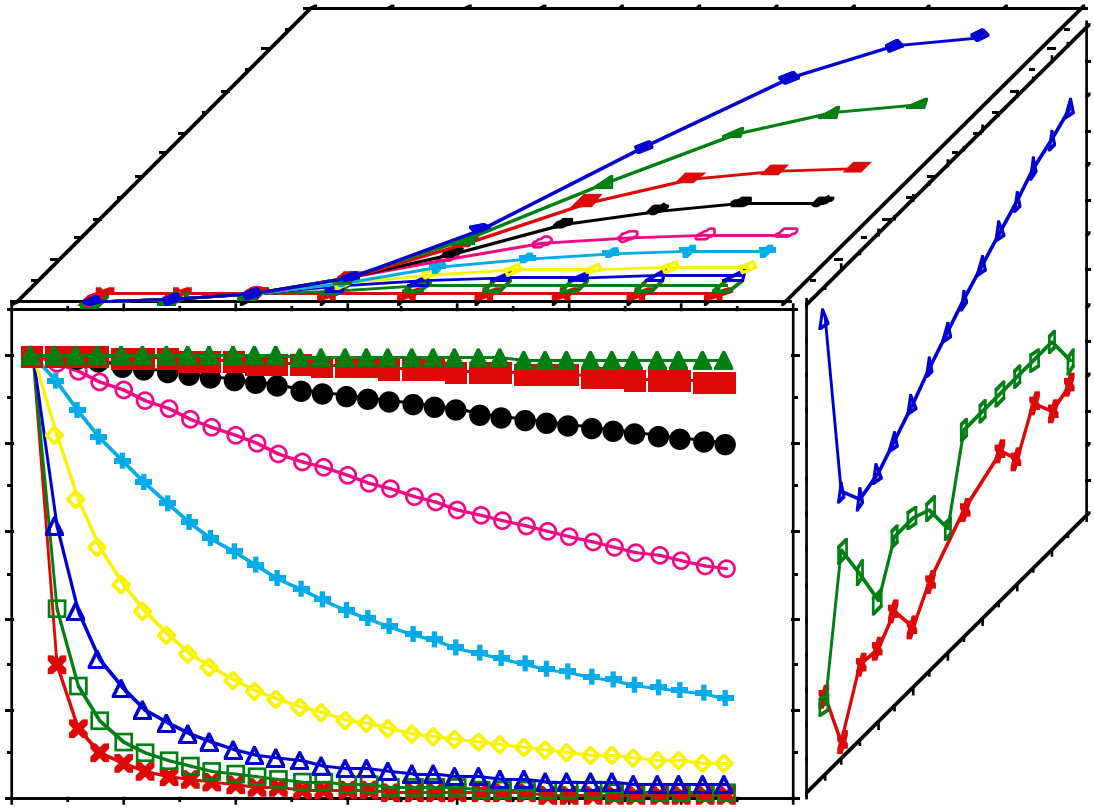
für alle p mit $R'(p,G(p_1))$ und $R'(p,G(p_2))$ und einer reflexiven Relation $R' \subset R$ und der Wahrscheinlichkeit $\mathbf{P}$.

G ist sowohl für R′ symmetrisch als auch für R′ total geordnet definiert. Für eine gute Lastverteilung muß nun das Maximum der Erwartungswerte der $\{\#D_j^p\}_p$ minimiert werden. Diese Gleichverteilung, in einer geeigneten Norm gemessen, führt für gegebenes oder heuristisch gewähltes R′ auf ein nichtlineares Gleichungssystem der Dimension p für die Terme w(p), das iterativ näherungsweise gelöst werden kann.

Das Gesamtverfahren hat damit keine höhere Ordnung als W_j^{BPX}. Aussagen über die Qualität der Lastverteilung können damit allerdings nur noch für große $\dim V_j$ gemacht werden.

Um sowohl die Aufteilung des Beispiels für das uniform verfeinerte Einheitsquadrat zu erreichen, als auch in allgemeineren Fällen gute Näherungen für das Minimum der W_j^{CG} zu erzielen, müssen die Ränder der Gebiete klein sein. Das kann man im allgemeinen dadurch erreichen, daß innere Punkte bereits gesondert vor der Berechnung der w(p) behandelt werden und daß Punkte g mit gleichen Nachbarprozessoren und gleichem G(g) in zusammenhängenden Gruppen angeordnet werden. Für stark adaptiv verfeinerte Gebiete, also nur langsam steigende $\{\dim V_j\}$, werden die entstehenden Ränder naturgemäß anteilig größer als für quasi-regulär verfeinerte Gebiete.

Es kann aber sowohl eine gute Lastverteilung als auch die Einhaltung der gegebenen lokalen Kommunikationsstruktur wie auch eine günstige Ordnung des Verteilungsverfahrens selbst erreicht werden.



Kapitel 7
Experimentelle Ergebnisse

Nachdem wir bisher auf der Basis des Kostenfunktionals optimiert haben, wollen wir überprüfen, in wie weit die Annahmen, die zu dem Funktional geführt haben, mit der Praxis übereinstimmen. Dazu vergleichen wir Meßwerte verschiedener Parallelrechner mit den Ergebnissen des Kapitel 4.

Die adaptiven Verfahren der Kapitel 5 und 6 wurden bisher noch nicht vollständig auf Parallelrechnern implementiert. Deren Ergebnisse werden aber in einer folgenden Arbeit vorgestellt werden.

Wir betrachten das nicht adaptive Beispiel des quasiuniformen regulär verfeinerten Einheitsquadrates. Der diskrete Differentialoperator wird durch einen 3×3 Stern charakterisiert. Jede Triangulierung entsteht durch die Zerlegung jedes Dreiecks der vorhergehenden Triangulierung in jeweils 4 kongruente Teildreiecke. Das Problem wird mit einem BPX-vorkonditionierten CG-Verfahren gelöst. Die in einer Kette angeordneten Prozessoren bearbeiten jeweils ein Rechteck der Kantenlänge des Quadrates auf jeder Verfeinerungsebene, also die Zeilen mit den Nummern $\{d_k^p\}$. Die Meßwerte werden verglichen mit dem allgemeinen Kostenfunktional W^{CG} und den speziell für dieses Beispiel hergeleiteten Näherungen (33)–(37) aus Kapitel 4.

7.1. Ergebnisse auf Intel iPSC/2

Die in den Abbildungen 7.1.1 bis 7.1.4 aufgeführten Messungen stammen von einem Rechner mit verteiltem Speicher (Intel iPSC/2), dessen Prozessoren in Hypercube-Topologie verbunden sind und asynchron kommunizieren. Die Übertragungszeiten sind dabei fast wegunabhängig. Die meisten Kommunikationen finden allerdings zwischen physikalischen Nachbarn statt, so daß sich dieser Effekt kaum auswirkt.

Die Meßwerte für die Beschleunigung der Rechnung durch den Einsatz mehrerer Prozessoren gegenüber einem Prozessor sind in Abbildung 7.1.1 gegenübergestellt dem Kostenfunktional W_j^{CG}, hier W geschrieben, das die Anzahl der Rechenoperationen und die Anzahl der Sende- und Empfangsoperationen berücksichtigt. Zum Vergleich sind noch die Werte der Formel (33) angegeben. Die Werte für die Beschleunigung sind angetragen gegen die feinste verwendete Verfeinerungsstufe. Die einzelnen Kurven stellen verschiedene Zahlen von Prozessoren dar.

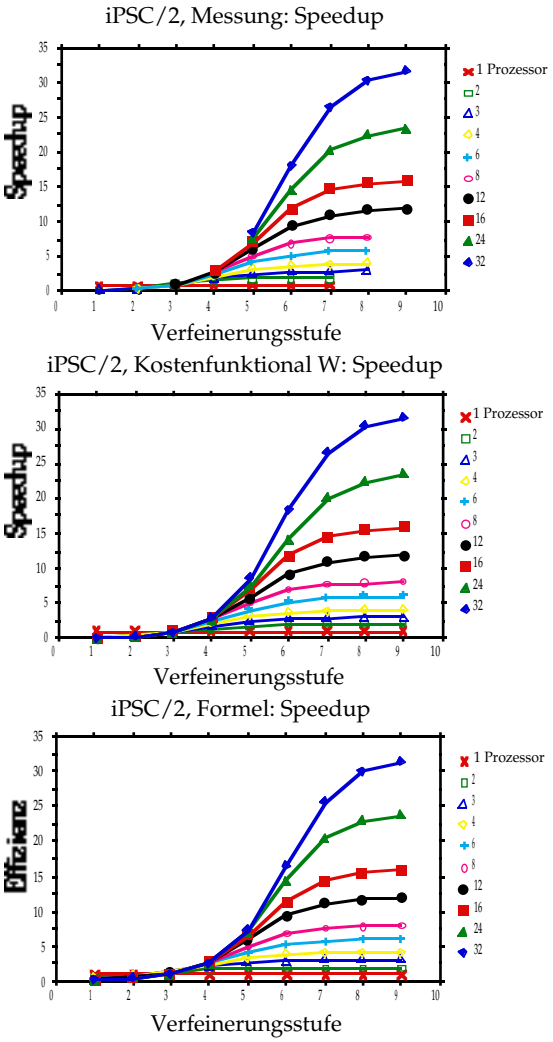


Abb 7.1.1 a–c. *Speedup: Messung, Kostenfunktional und Formel*

Die Meßwerte zeigen, daß für große Verfeinerungsstufen, also große Zahlen von Unbekannten, nahezu jede mögliche Beschleunigung erreicht werden kann. So können auch auf Verfeinerungsstufe 9 alle 32 Prozessoren sinnvoll eingesetzt werden und das Ergebnis in über 31.4-facher Geschwindigkeit berechnen gegenüber einem Prozessor der genügend Speicher hat. Die Vergleichswerte für einen Prozessor mit den Stufen 8 und 9 mußten extrapoliert werden, was bei der hohen Meßgenauigkeit und Güte der Modellrechnung aber sehr genau möglich war. Im übrigen ist eine gute Übereinstimmung der Meßwerte mit den beiden Modellen zu beobachten, obwohl hier als einziger rechnerabhängige Wert das Verhältnis von Rechengeschwindigkeit zu mittlerer Kommunikationszeit eingeht.

Die Grafiken in Abbildung 7.1.2 und 7.1.3 geben die Effizienz der Ausnutzung der Prozessoren wieder. Dabei sind in 7.1.2 jeweils Kurven für einzelne Prozessorzahlen abgebildet, wobei die erzielte Effizienz gegen die feinste Verfeinerungsstufe angetragen ist. In Abbildung 7.1.3 ist mit den Kurven für einzelne Verfeinerungsstufen die Effizienz gegen die Zahl der verwendeten Prozessoren angetragen. In den folgenden drei Zeilen sind zunächst die gemessenen Werte angegeben. Dann folgen die Werte, die mit dem Kostenfunktional für W_j^{CG} bestimmt wurden, und schließlich werden diese mit der Formel (34) verglichen.

Die Meßwerte zeigen, daß für große Verfeinerungsstufen Werte für die Effizienz von 96%, oft über 98%, bei kleinen Prozessorzahlen auch über 99%, erreicht werden können. Für kleinere Verfeinerungsstu-

fen ist deutlich der Abfall der Effizienzen bei steigenden Prozessorzahlen zu beobachten.

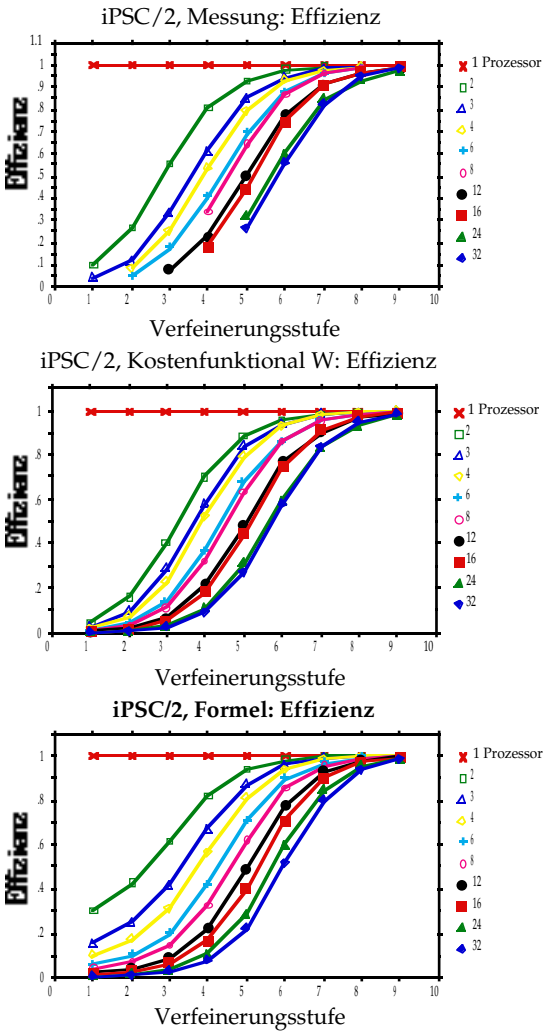


Abb 7.1.2 a–c. *Effizienz: Messung, Kostenfunktional und Formel*

Die beiden Modellrechnungen zeigen wiederum eine hohe Übereinstimmung mit den Meßwerten. Da die Formel (34) stetig differenzierbar in p und j ist, erscheinen in dieser einfachen Näherung keine Sprünge oder Zacken, obwohl die Werte qualitativ richtig sind.

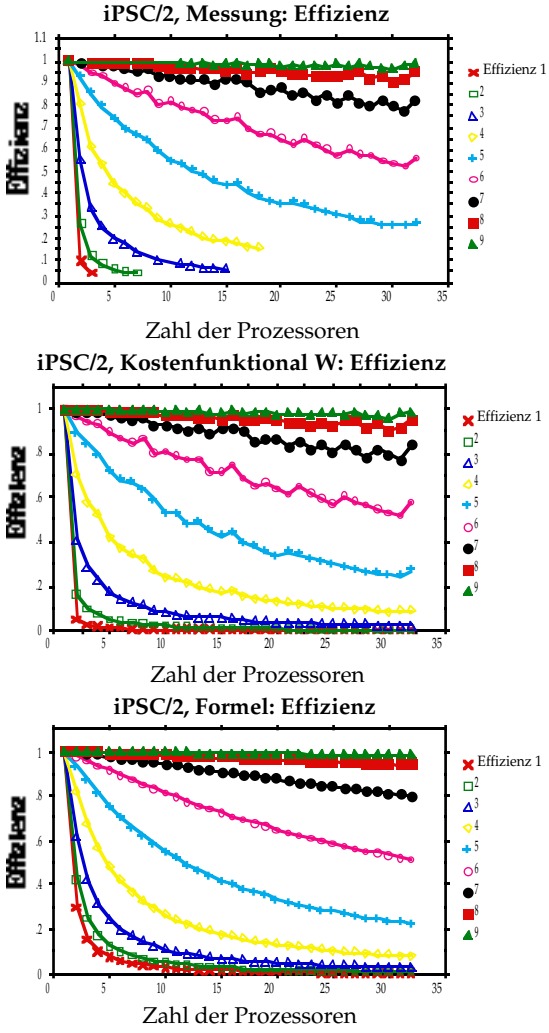


Abb 7.1.3 a–c. *Effizienz gegen Zahl der Prozessoren*

Das Kostenfunktional W_j^{CG} dagegen kann auch ungleiche Lastverteilungen, die durch die Division der Zeilenzahlen durch die Prozessorzahlen entstehen, modellieren und gibt daher selbst Sprünge und Zacken in den Kurven richtig wieder. So ist kein nennenswerter Un-

terschied zwischen den Meßwerten und dem Kostenfunktional zu beobachten, was die Bedeutung dieses Funktionals und seiner theoretischen Begründung zeigt.

Im folgenden wird die Frage untersucht, mit welcher Zahl von Prozessoren sich die kürzesten Ausführungszeiten ergeben. Diese Zahlen sind, nach Verfeinerungsstufen geordnet, für die aus (33) abgeleitete Formel (35) (gerundete Werte), das Kostenfunktional W_j^{CG} und die eigentlichen Meßwerte in Tabelle 7.1.4 angegeben.

Verfeinerungs-stufe	Formel (35)	W_j^{CG}	Messung
1	0.2	1	1
2	0.8	1	1
3	3.1	1	2
4	12.2	16	16
5	49.0	32 (max)	32 (max)
6	195.8 (>129)	32 (max)	32 (max)
7	783.2 (>257)	32 (max)	32 (max)
8	3132.8 (>513)	32 (max)	32 (max)
9	12531.3 (>1025)	32 (max)	32 (max)

Tab 7.1.4. *optimale Zahl von Prozessoren*

Ab Verfeinerungsstufe 5 reichen die vorhandenen 32 Prozessoren nicht mehr aus, um eine sinnvolle Aussage zu machen. Wir können lediglich sagen, daß alle vorhandenen Prozessoren zu einer Geschwindigkeitssteigerung beitragen und daher sinnvoll eingesetzt werden können.

Die Formel (35) zeigt aber, daß ab Verfeinerungsstufe 6 alle durch die zeilenweise Aufteilung des Gebietes möglichen Prozessorzahlen zu einer Beschleunigung führen werden. Auch in diesem Fall liefern beide Modellrechnungen Ergebnisse der gleichen Größenordnung wie die Messungen.

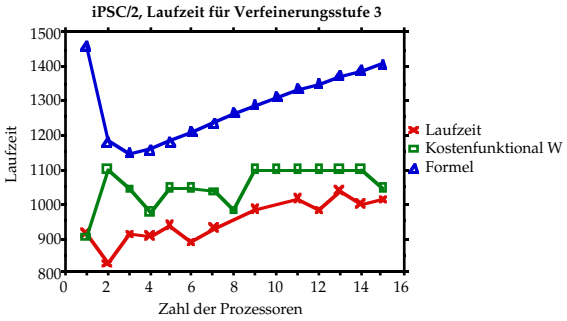


Abb 7.1.5. *Laufzeit: Messung, Kostenfunktional und Formel*

Die Abbildung 7.1.5 zeigt absolute Ausführungszeiten (in ms) für die beiden Modellrechnungen (33) und W^{CG}und die Messung, angetragen gegen die Zahl der Prozessoren für die Verfeinerungsstufe 3.

Deutlich ist die Übereinstimmung der Größenordnungen der drei Kurven zu erkennen, wobei für diese relativ geringe Verfeinerungsstufe Einflüsse der Gebietsränder und zeitliche Überlappungseffekte mit den gröberen Gittern die absoluten Werte noch stark beeinflussen. Die Werte stehen in Übereinstimmung mit der Tabelle für die optimalen Prozessorzahlen und illustrieren deren Ergebnisse für die Verfeinerungsstufe 3.

7.2. Ergebnisse auf T800

Die Werte der Abbildungen 7.2.1 und 7.2.2 sind Messungen mit einer Kette von Transputern (T800), also einem Rechner mit verteiltem Speicher. Durch das Betriebssystem Helios bedingt, sind die Initialisierungskosten für Sende- und Empfangsoperationen relativ zur etwa 9-fachen Rechengeschwindigkeit deutlich höher, was die Ergebnisse negativ beeinflußt.

Die Prozessoren konnten aus Hardware-Gründen nicht als Hypercube vernetzt werden, weshalb hier die Konfiguration als Kette gewählt wurde. Damit steigt der Aufwand für W^{SC} nicht mit $\log p$ sondern mit $p/2$, was aber für die kleinen Prozessorzahlen bis zu 8 und dem geringen Rechenzeitanteil der Skalarprodukte an der Gesamtrechenzeit vernachlässigbar bleibt.

Ein weiteres Problem entsteht auf den gröbsten Gittern, wenn einige Prozessoren keine Punkte mehr zu berechnen haben. Da die Kommunikationsstrukturen bereits während der Übersetzungsphase des Programmtextes festgelegt werden müssen, können die untätigen Prozessoren nicht einfach übersprungen werden, sondern müssen die Informationen weiterleiten. Damit werden Kommunikationsoperationen auf den gröbsten Gittern teurer als auf feineren Gittern, was aber keinen großen Einfluß auf die Gesamtergebnisse hat.

Es folgen in Abbildung 7.2.1 Meßwerte für die Beschleunigung der Rechnung durch den Einsatz mehrerer Prozessoren gegenüber einem Prozessor. In 7.2.1.a sind die Werte für die Beschleunigung angetragen gegen die feinste verwendete Verfeinerungsstufe. Die einzelnen Kurven stellen verschiedene Zahlen von Prozessoren dar. In 7.2.1.b sind die Meßwerte gegen die Zahl der Prozessoren angetragen, wobei für jede Verfeinerungsstufe eine Kurve eingezeichnet wurde.

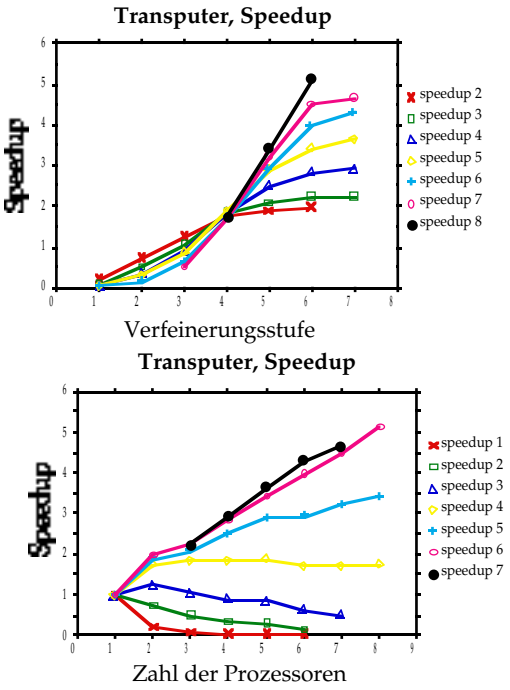


Abb 7.2.1 a,b. *Messung Speedup*

Die Meßwerte zeigen, daß für große Verfeinerungsstufen, also große Zahlen von Unbekannten, nahezu jede mögliche Beschleunigung erreicht werden kann. So können auch auf Verfeinerungsstufe 7 alle vorhandenen 8 Prozessoren sinnvoll eingesetzt werden und das Ergebnis in über 5.1-facher Geschwindigkeit berechnen gegenüber einem Prozessor. Im übrigen ist eine gute qualitative Übereinstimmung der Meßwerte mit den Werten des iPSC/2 zu beobachten, obwohl hier eine andere Prozessortopologie verwendet wurde.

Die Grafiken in Abbildung 7.2.2 geben die Effizienz der Ausnutzung der Prozessoren wieder. Dabei sind in 7.2.2.a jeweils Kurven für einzelne Prozessorzahlen abgebildet, wobei die erzielte Effizienz gegen die feinste Verfeinerungsstufe angetragen ist. In 7.2.2b ist mit den Kurven für einzelne Verfeinerungsstufen die Effizienz gegen die Zahl der verwendeten Prozessoren angetragen.

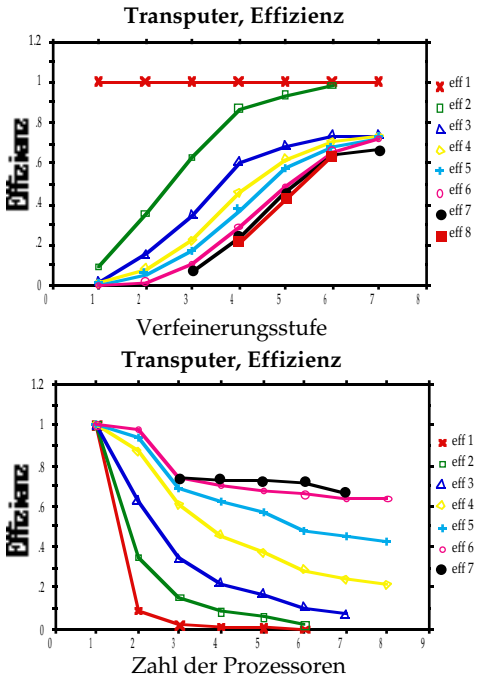


Abb 7.2.2 a,b. *Messung Effizienz*

Die Meßwerte zeigen, daß für große Verfeinerungsstufen Werte für die Effizienz von 64% bis 73%, bei zwei Prozessoren auch bis zu 98%, erreicht werden können. Für kleinere Verfeinerungsstufen ist deutlich der Abfall der Effizienzen bei steigenden Prozessorzahlen zu beobachten. Sobald mehr als zwei Prozessoren eingesetzt werden, wirkt die Kommunikation der inneren Prozessoren mit ihren jeweiligen Nachbarn sehr bremsend. Der

Datentransfer kann nur jeweils in eine bestimmte Richtung durchgeführt werden und wirkt damit synchronisierend auf die Prozesse, was zu den zu beobachtenden Leistungseinbußen führt.

Die Messungen zeigen wiederum eine qualitative Übereinstimmung mit den Meßwerten des iPSC/2.

7.3. Ergebnisse auf Alliant FX/2800

Die folgenden Messungen wurden mit einem Parallelrechner mit gemeinsamem Speicher und 8 Prozessoren durchgeführt (Alliant FX/2800). Die Messungen wurden mit dem vorhandenen Multitasking-Multiuser-Betriebssystem durchgeführt. Jeder Hierarchie von Teilgebieten $\{D_k^p\}_k$ wurde dabei ein Systemprozess zugeordnet. Die Abbildung der Prozesse auf Prozessoren wurde durch das Betriebssystem zur Laufzeit durchgeführt. Die Kommunikation der Prozesse erfolgt über die jeweiligen Randdaten der Gebiete, die im gemeinsamen Speicher angeordnet sind. Die inneren Daten sind aus Geschwindigkeitsgründen in privaten Speicherbereichen abgelegt. Die Rolle der Kommunikationsoperationen, also der Sende- und Empfangsoperationen, übernehmen hier Synchronisationsoperationen mit Semaphoren, die die Kohärenz der Daten sicherstellen. Dabei entstehen die gleichen Lastausgleichsprobleme wie für Rechner mit verteiltem Speicher.

Die Meßwerte hängen deutlich von der übrigen Belastung des Rechners ab und schwanken daher. Es wurde jeweils der beste Wert einer Reihe von Messungen bei möglichst unbelasteter Maschine verwendet.

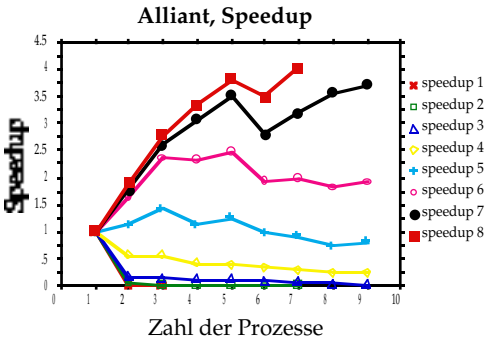


Abb 7.3.1. *Messung Speedup*

Es folgen in Abbildung 7.3.1 Meßwerte für die Beschleunigung der Rechnung durch den Einsatz mehrerer Prozessoren gegenüber einem Prozessor. Dabei bezieht sich die Beschleunigung hier auf die Gesamtlaufzeit der Rechnung, unabhängig davon, wieviel Rechenzeit den einzelnen Prozessen von dieser Echtzeit vom Betriebssystem zugeteilt werden. Die Meßwerte sind gegen die Zahl der Prozesse angetragen, wobei für jede Verfeinerungsstufe eine Kurve eingezeichnet wurde.

Der Meßwert für 9 Prozesse zeigt eine Laufzeitverbesserung gegenüber kleineren Werten durch eine

verbesserte Granularität des Gesamtprogramms. Die Meßwerte zeigen, daß für große Verfeinerungsstufen, also große Zahlen von Unbekannten, gute Beschleunigungen erreicht werden können. So können auch auf Verfeinerungsstufe 9 alle vorhandenen 8 Prozessoren sinnvoll eingesetzt werden und das Ergebnis in 4-facher Geschwindigkeit berechnen gegenüber einem Prozessor. Im übrigen ist eine gute qualitative Übereinstimmung der Meßwerte mit den theoretischen Werten zu beobachten. Der Abfall der Leistung beim Übergang von 5 auf 6 und 7 Prozesse kann mit den Hardware-Eigenschaften des Rechners erklärt werden. Die Datenpfade der Prozessoren zu den Speichermodulen sind so ausgelegt, daß die Hälfte der Prozessoren nahezu wechselwirkungsfrei auf privatem Speicher arbeiten kann. Arbeiten aber mehr Prozessoren mit ähnlichen Speicherzugriffen, so treten starke bremsende Bus- und Cache-Konflikte auf.

Die Grafiken in Abbildung 7.3.2 geben die Effizienz der Ausnutzung der Prozessoren wieder. Die Effizienz bezieht sich hier im Gegensatz zur Beschleunigung auf die maximale, von einem Prozeß verbrauchte Rechenzeit. Dabei sind in 7.3.2.a jeweils Kurven für einzelne Prozeßzahlen abgebildet, wobei die erzielte Effizienz gegen die feinste Verfeinerungsstufe angetragen ist. In 7.3.2.b ist mit den Kurven für einzelne Verfeinerungsstufen die Effizienz gegen die Zahl der verwendeten Prozesse angetragen.

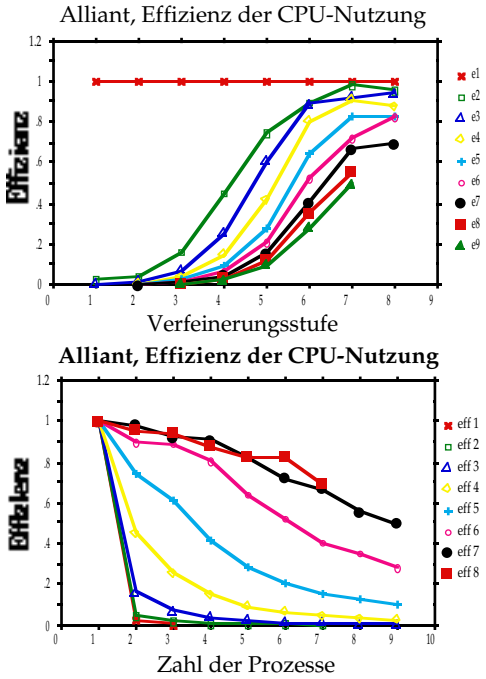


Abb 7.3.2 a,b. *Messung Effizienz*

Die Meßwerte zeigen, daß für große Verfeinerungsstufen Werte für die Effizienz von 70–85%, bei zwei Prozessen auch 98%, erreicht werden können. Für kleinere Verfeinerungsstufen ist deutlich der Abfall der Effizienzen bei steigenden Prozessorzahlen zu beobachten. Dieser Abfall ist aber für alle Verfeinerungsstufen zu erkennen. Das bedeutet, daß hier nicht nur Lastausgleichsprobleme und Zugriffe auf gemeinsame Randdaten bremsend wirken, sondern auch der unabhängige Ablauf des Programms auf

mehreren verschiedenen Prozessoren mit unabhängigen Daten. Dieser Effekt kann auch während des normalen Multiuser-Betriebs des Rechners beobachtet werden. Eine bestimmte Grundrechenlast auf einigen Prozessoren wirkt sich fatal auf die übrigen aus.

Die Messungen zeigen wiederum eine gute qualitative Übereinstimmung mit der Theorie (36) und (37), bis auf die Leistungsabfälle für innere Gitterpunkte bei großen Prozessorzahlen.

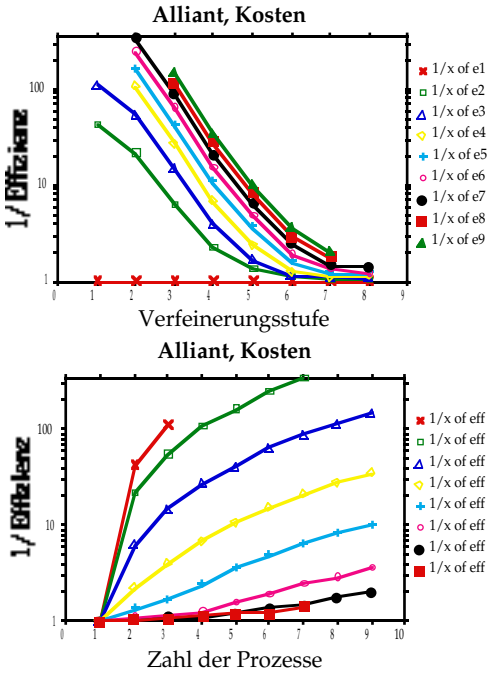
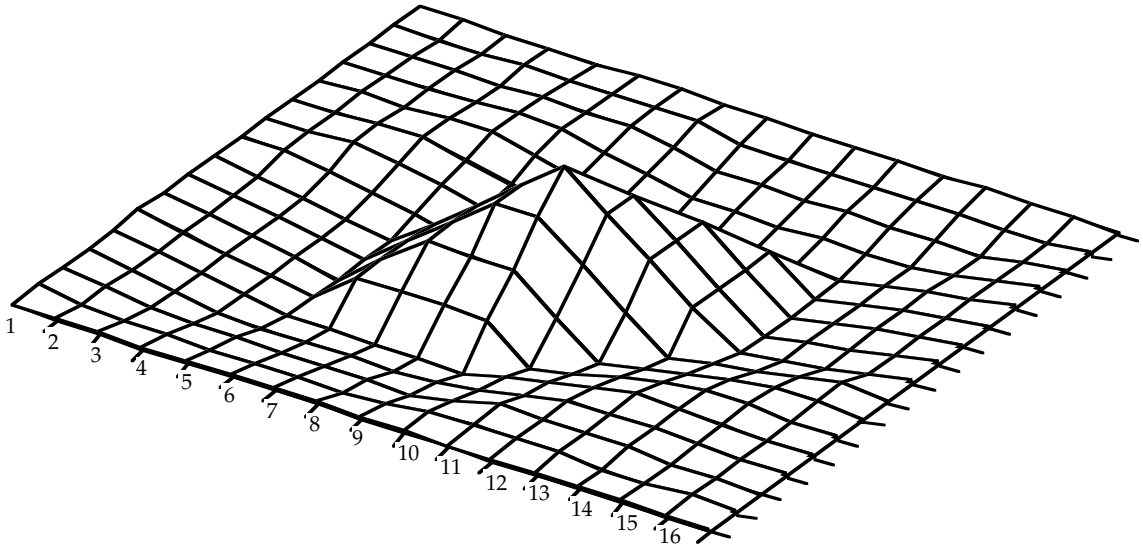


Abb 7.3.3 a,b. *Messung Kosten der Parallelisierung*

Die Abbildung 7.3.3 zeigt Grafiken, die analog zu den abgebildeten Effizienzen die reziproken Werte der Effizienzen, hier als Kosten der Parallelisierung bezeichnet, zeigen.

An den Kosten der Parallelisierung ist deutlich zu sehen, daß sich aufgrund der starken Beeinflussung der Prozessoren untereinander eine Parallelisierung einer derart auf Durchsatz und weniger auf Spitzenleistung hin optimierten Maschine, wie es die Werte nahelegen, nur für große Probleme und Verfeinerungsstufen lohnt. Wenn die Zahl der Unbekannten beispielsweise kleiner als Tausend ist, kostet die Parallelisierung mindestens einen Faktor 14 an Rechenzeit, bei zwei Prozessen aber zumindest den Faktor 6 mehr als die sequentielle Rechnung.

Bei diesen Vergleichen muß man aber weiterhin berücksichtigen, daß ein Prozessor der Alliant eine Größenordnung schneller arbeitet als ein Prozessor des iPSC/2, und damit immer noch Faktoren schneller als ein T800 Prozessor. Bei diesen Zuwächsen an Rechengeschwindigkeit wird es zusehend schwieriger, die Kommunikations- und Datentransferleistung der Rechner in ähnlichen Größenordnungen zu steigern.



Kapitel 8
Schlußbemerkungen

Wir haben die Parallelisierung der einzelnen Komponenten eines von Bramble, Pasciak und Xu vorgeschlagenen vorkonditionierten Verfahrens der konjugierten Gradienten, eingebettet in ein adaptives Finite-Elemente-Programm, diskutiert, die zusätzliche Forderungen an Triangulierungen, Finite-Elemente-Räume und Gittermanipulationsalgorithmen stellt. Nachdem Standardverfahren der Lastverteilung teils eine größere Komplexität als das Lösungsverfahren selbst besitzen und zum anderen Teil nicht genügend an Multilevelverfahren angepaßt sind, haben wir einen Ansatz für ein neues Aufteilungsverfahren gemacht, das auf einem statistischen Ansatz beruht. Mit einer gemischten Strategie der Aufteilung von Gitterpunkten und Dreiecken konnte ein Gesamtverfahren von optimaler Ordnung erreicht werden.

Die experimentellen Ergebnisse auf einigen Parallelrechnern zeigten eine hohe Übereinstimmung mit denen durch das Kostenfunktional vorhergesagten Werten, das die Grundlage unserer Analysen bildete. Für statische Aufteilungen konnte eine ideale Parallelisierbarkeit des Verfahrens gezeigt werden. Die Unterschiede zu Parallelisierungen anderer bekannter Multilevelverfahren waren gering, konnten aber im Modell erfaßt werden.

Die nächste Aufgabe besteht in einer praktischen Umsetzung der Verfahren im adaptiven Fall. Entsprechende Ergebnisse werden in einer folgenden Arbeit vorgestellt werden. Ziele weiterer Untersuchungen könnten Experimente mit vollständig parallelisierten Finite-Elemente-Programmen oder auch genauere Untersuchungen des vorgeschlagenen Lastverteilungsverfahrens sein. Die Abstimmung des Verfahrens wird aber mit Sicherheit stark vom verwendeten Parallelrechner, vom umgebenden iterativen Löser und den Ansprüchen an die Qualität der Verteilung abhängen. Eine andere Frage ist, ob trotz der höheren Komplexität in der Praxis nicht doch ein teureres Lastverteilungsverfahren zu günstigeren Meßwerten führt.

An dieser Stelle möchte ich dem Lehrstuhl Prof. Dr. Ritter für die Erlaubnis zur Benutzung des verwendeten Transputersystems danken.

Literatur

O. Axelson, V.A. Barker: *Finite Element Solution of Boundary Value Problems: Theory and Computation,* Academic Press, Orlando (1984)

I. Babuska, W.C. Rheinboldt: *Error Estimates for Adaptive Finite Element Computations,* SIAM J. Numer. Anal. 4 (1978), 736–754

G. Bader, E. Gehrke: *Leistungsmessung auf massiv parallelen Transputersystemen,* Vortrag, GAMM Seminar 31.05.91 Univ. Heidelberg (1991)

R.E. Bank: *PLTMG. A Software Package for Solving Elliptic Partial Differential Equations: Users' Guide 6.0,* SIAM (1990)

R.E. Bank, T.F. Dupont, H. Yserentant: *The Hierarchical Basis Multigrid Method*, Numer. Math. 52 (1988), 427–458

R.E. Bank, A. Weiser: *Some a posteriori Error Estimators for Elliptic Partial Differential Equations,* Math. Comp. 44 (1985), 283–301

E. Bänsch: *Local Mesh Refinement in 2 and 3 Dimensions,* Univ. Bonn, SFB 256, Report 6 (1989)

P. Bastian: *Kommunikation in Transputernetzen und Lastverteilung unstrukturierter Gitter*, Vortrag, GAMM Seminar 31.05.91 Univ. Heidelberg (1991)

M.J. Berger, S. Bokhari: *A Partitioning Strategy for Nonuniform Problems on Multiprocessors,* IEEE Trans. Comput., vol. C–36, no. 5 (1987), 570–580

P.E. Bjørstad, O.B. Widlund: *To overlap or not to overlap: A note on a domain decomposition method for elliptic problems*, SIAM J. Sci. Stat. Comput. 10 (1989), 1053–1061

F.A. Bornemann: *A Sharpened Condition Number Estimate for the BPX Preconditioner of Elliptic Finite Element Problems on Highly Nonuniform Triangulations,* Konrad-Zuse-Zentrum für Informationstechnik Berlin, Preprint SC 91–9 (1991)

F.A. Bornemann, B. Erdmann, R. Roitzsch: *KASKADE – Numerical Experiments,* Konrad-Zuse-Zentrum für Informationstechnik Berlin, Techn. Report TR 91–1 (1991)

J.H. Bramble, J.E. Pasciak, A.H. Schatz: *The construction of preconditioners for elliptic problems by substructuring, I*, Math. Comp. 47 (1986), 103–134 & *…, II,* Math. Comp. 49 (1987), 1–16 & *…, III,* Math. Comp. 51 (1988), 415–430 & *…, IV,* Math. Comp. 53 (1989), 1–24

J.H. Bramble, J.E. Pasciak, J. Wang, J. Xu: *Convergence estimates for product iterative methods with applications to domain decomposition and multigrid*, Math. Sciences Inst. Cornell Univ., Techn. Report 90–39 (1990)

J.H. Bramble, J.E. Pasciak, J. Xu: *Parallel Multilevel Preconditioners*, Math. Comp. 55 (1990), 1–22

B. Briggs, L. Hart, S.F. McCormick, D. Quinlan: *Multigrid Methods on a Hypercube*, in S.F. McCormick (Ed.), Proc. 3rd Copper Montain Conf. Multigrid Methods, Marcel Dekker, New York (1988), 63–83

P.G. Ciarlet: *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam (1978)

P. Deuflhard, P. Leinen, H. Yserentant: *Concepts of an Adaptive Hierarchical Finite Element Code*, IMPACT of Computing in Science and Engineering 1 (1989), 3–35

G.C. Fox, A. Kolawa, R. Williams: *The Implementation of a Dynamic Load Balancer*, in M.T. Heath (Ed.), Proc. 2nd Conf. Hypercube Multiprocessors 1986, SIAM (1987), 114–121

G.C. Fox, S.W. Otto: *Concurrent Computation and the Theory of Complex Systems*, in M.T. Heath (Ed.), Proc. 1st Conf. Hypercube Multiprocessors 1985, SIAM (1986), 244–268

A. George, M.T. Heath, J.W. Liu, E.G-Y. Ng: *Sparse Cholesky Factorization on a Local-Memory Multiprocessor*, Oak Ridge National Laboratory, Oak Ridge, Techn. Report ORNL/TM–9962 (1986)

A. George, J.W. Liu: *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs (1981)

A. George, J.W. Liu, E.G-Y. Ng: *Comunication Reduction in Parallel Sparse Cholesky Factorization on a Hypercube*, in M.T. Heath (Ed.), Proc. 2nd Conf. Hypercube Multiprocessors 1986, SIAM (1987), 576–586

F. Glover: *Tabu Search*, University of Colorado, Boulder, CAAI Report 88–3 (1988)

M. Griebel: *Multilevel Algorithms considered as Iterative Methods on Indefinite Systems*, TU München, Institut für Informatik, SFB-Bericht 342/29/91A (1991)

W. Hackbusch: *Multi-grid methods and applications*, Springer, Berlin (1985)

W. Hackbusch: *Iterative Lösung großer schwachbesetzter Gleichungssysteme*, Teubner, Stuttgart (1991)

R. Kornhuber, R. Roitzsch: *On Adaptive Grid Refinement in the Presence of Internal or Boundary Layers*, IMPACT of Computing in Science and Engineering 2 (1990), 40–72

P. Leinen: *Ein schneller adaptiver Löser für elliptische Randwertprobleme auf Seriell- und Parallelrechnern,* Dissertation, Dortmund (1990)

H. Mierendorff: *Parallelization of Multigrid Methods with Local Refinements for a Class of Nonshared Memory Systems*, in S.F. McCormick (Ed.), Proc. 3rd Copper Montain Conf. Multigrid Methods, Marcel Dekker, New York (1988), 449–465

S. Nölting: *Nonlinear Adaptive Finite Element Systems on Distibuted Memory Computers*, in A. Bode (Ed.), Proc. 2nd European Conf. Distributed Memory Computing, Springer, Berlin (1991), 283–293

P. Oswald: *On discrete norm estimates related to multilevel preconditioners in the finite element method*., Erscheint in: Proc. Int. Conf. Constr. Theory of Functions Varna '91

R. Roitzsch, R. Kornhuber: *BOXES – a Program to Generate Triangulations from a Rectangular Domain Description,* Konrad-Zuse-Zentrum für Informationstechnik Berlin, Techn. Report TR 90–9 (1990)

B. Veer:*The CDL Guide*, Distributed Software Ltd. (1990)

J. Xu: *Theory of Multilevel Methods,* PhD thesis, Cornell Univ. & Department of Mathematics, Pennsylvania State Univ, Report AM 48 (1989)

H. Yserentant: *Two Preconditioners Based on the Multi-Level Splitting of Finite Element Spaces*., Numer. Math. 58 (1990), 379–412

X. Zhang: *Multilevel Additive Schwarz Methods,* Department of Computer Science, Courant Institute, New York, Techn. Report 582 (1991)

———

——— 36

1 37

2 38

3 39

4 40

5 41

6 42

7 43

8 44

9 45

10 46

11 47

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35