Adaptive Semi-Lagrange-Finite-Elemente-Methode zur Lösung der Flachwassergleichungen: Implementierung und Parallelisierung

Adaptive semi-Lagranigian finite element method for the solution of the shallow water equations: Implementation and parallelization

---

Jörn Behrens

Ber. Polarforsch. 217(1996)
ISSN 0176 - 5027

Jörn Behrens
Alfred-Wegener-Institut für Polar- und Meeresforschung
Institute for Polar and Marine Research
Postfach 12 01 61
27515 Bremerhaven, Germany
jbehrens@awi-bremerhaven.de

Die vorliegende Arbeit ist die inhaltlich unveränderte Fassung
einer Dissertation zur Erlangung des Doktorgrades des
Fachbereichs Mathematik und Informatik an der
Universität Bremen, die am 13.12.1995 eingereicht wurde

Inhaltsverzeichnis

Einleitung 7

I Die Finite-Elemente-Methode (FEM) für elliptische Probleme 11

1 Einführung in die FEM 13

1.1 Übersicht 13

1.2 Modellprobleme und ihre Variationsformulierung 14

1.2.1 Die Poissongleichung und einige ihrer Eigenschaften 15

1.2.2 Verschiedene Randbedingungen 17

1.2.3 Ein allgemeinerer Differentialoperator 18

1.3 Diskretisierung 19

1.3.1 Das Ritz-Galerkin-Verfahren 19

1.3.2 Die Finite-Elemente-Methode 21

2 Implementierung der FEM 24

2.1 Übersicht 24

2.2 Gittergenerierung und Datenstrukturen 25

2.2.1 Notationen 26

2.2.2 Ansätze zur Gittergenerierung 27

2.2.3 Gitterverfeinerung 29

2.2.4 Datenstrukturen 34

2.3 Aufstellung der Steifigkeitsmatrix 36

2.3.1 Berechnung der Elementsteifigkeitsmatrizen 37

2.3.2 Assemblierung der Steifigkeitsmatrix 38
2.3.3 Sparsame Speicherung spärlich besetzter Matrizen 39
2.4 Lösungsverfahren für das lineare Gleichungssystem 43
2.5 Präkonditionierung für iterative Verfahren 43
2.5.1 Präkonditionierung mit hierarchischen Basen 44
2.6 Randbedingungen 48

3 Parallelisierung der FEM **50**
3.1 Übersicht 50
3.2 Parallelisierung der iterativen Lösungsverfahren 51
3.3 Parallelisierung mit Farbindizierung 52
3.4 Parallelisierung mit Hilfe von Index-Arrays 55
3.5 Probleme und Bemerkungen 56

4 Ergebnisse für die FEM **58**
4.1 Übersicht 58
4.2 Modellprobleme und Konvergenztests 59
4.3 Effizienz der parallelen Verfahren 61
4.3.1 Ergebnisse für die Matrixassemblierung 62
4.3.2 Ergebnisse für das Lösungsverfahren 64
4.4 Zusammenfassung 66

II Die Semi-Lagrange-Methode (SLM) für die Advektionsgleichung **69**

5 Einführung in die SLM **71**
5.1 Übersicht 71
5.2 Das Advektionsproblem 72
5.3 Semi-Lagrange-Form der Advektionsgleichung 73
5.3.1 Das Drei-Schritt-Verfahren 73
5.3.2 Das Zwei-Schritt-Verfahren 74
5.4 Berechnung der Trajektorien 75
5.5 Interpolation der stromaufwärts gelegenen Werte 76
5.5.1 Bedeutung der Interpolationsordnung 76
5.5.2 Quasi-monotone Interpolation 76
5.5.3 Quasi-konservative Interpolation 78

6 Implementierung der SLM **80**

6.1 Übersicht 80

6.2 Algorithmus und adaptive Strategie 81

6.3 Gitterroutinen 83

6.3.1 Elementsuche im Dreiecksgitter 83

6.4 Interpolation 85

6.4.1 Berechnung der ersten partiellen Ableitungen 85

6.4.2 Berechnung der kubischen Spline-Interpolation 86

6.4.3 Quasi-monotone und quasi-konservative Interpolation 87

6.5 Diagnose 87

7 Parallelisierung der SLM **89**

7.1 Übersicht 89

7.2 Parallelisierung der Operationen an Gitterpunkten 89

7.3 Datenpartitionierung 90

7.4 Probleme bei der Parallelisierung 91

8 Ergebnisse für die SLM **93**

8.1 Übersicht 93

8.2 Numerische Eigenschaften der SLM 93

8.3 Eigenschaften des adaptiven Gitters 99

8.4 Absolute Stabilität der SLM 102

8.5 Parallele Leistung der SLM 103

8.6 Zusammenfassung 104

III Die Adaptive Semi-Lagrange-Finite-Elemente-Methode (ASLM) für die Flachwassergleichungen **107**

9 Einführung in die ASLM **109**

9.1 Übersicht 109

9.2 Die Flachwassergleichungen 110

9.3 Diskretisierung der Flachwassergleichungen 111

9.4 Dimensionslose Gleichungen 115

9.5 Modellproblem und numerisches Referenzmodell 117

10 Implementierung der ASLM **119**

10.1 Übersicht 119

10.2 Trajektorienberechnung 120

10.3 Berechnung der rechten Seite des elliptischen Problems 121

10.4 Das elliptische Problem 122

10.5 Fehlerschätzer und Adaptivität 123

10.6 Abgeleitete Größen 125

11 Parallelisierung der ASLM **126**

11.1 Übersicht 126

11.2 Parallelisierung des Semi-Lagrange-Teils 126

11.3 Parallelisierung des Fehlerschätzers 127

11.4 Parallelisierung des Lösers 127

12 Ergebnisse für die ASLM **129**

12.1 Übersicht 129

12.2 Numerische Ergebnisse und Referenzläufe 132

12.2.1 Ergebnisse des Referenzmodells 132

12.2.2 Ergebnisse der ASLM 133

12.3 Parallele Effizienz 135

12.4 Zusammenfassung 138

A Begriffe der Parallelisierung **141**

A.1 Begriffe der Parallelverarbeitung 141

A.2 Leistungsbestimmung bei parallelen Algorithmen 143

B Architekturmerkmale **146**

B.1 Architekturen von Parallelrechnern 146

B.2 Virtueller gemeinsamer Hauptspeicher 148

B.3 Details der KSR-1 Architektur 149

B.4 Details der Alliant FX/2800 Architektur 150

Bezeichnungen **152**

Abbildungsverzeichnis **154**

Tabellenverzeichnis **157**

Literaturverzeichnis **158**

Zusammenfassung

Die Flachwassergleichungen dienen als Fallbeispiel für die Entwicklung, Implementierung und Parallelisierung der neuen adaptiven Semi-Lagrange-Finite-Elemente-Methode (ASLM). In drei Teilen werden Datenstrukturen, Algorithmen und Verfahren für die Implementierung der parallelen ASLM entwickelt.

Der erste Teil ist der Finite-Elemente-Methode (FEM) gewidmet. Eine Einführung wird gegeben, die wichtigsten Techniken für die Implementierung vorgestellt, neue Datenstrukturen für die adaptive Gittergenerierung entwickelt, die Parallelisierung der verschiedenen Programmsegmente vorgenommen und experimentelle Ergebnisse vorgestellt. Zwei neue Parallelisierungsstrategien für die Aufstellung der Diskretisierungsmatrix, die tabellierte *Farbindizierung* und die *Index-Array-Methode*, erweisen sich als effizient und skalierbar. Die Index-Array-Methode wird in allen späteren Parallelisierungsansätzen in jeweils angepaßter Form wiederverwendet. Präkonditionierte CG-ähnliche iterative Verfahren werden als Löser für die Gleichungssysteme eingesetzt.

Im zweiten Teil wird die Semi-Lagrange-Methode (SLM) für die passive Advektionsgleichung auf einem adaptiven Gitter eingeführt. Besondere Sorgfalt muß bei der Implementierung der Interpolation und des adaptiven Algorithmus aufgewendet werden. Die Parallelisierung der SLM ist möglich, weil alle Operationen, die Gitterpunkten zugeordnet sind, unabhängig voneinander vorgenommen werden können. Alle SLM-Teile des Verfahrens lassen sich sehr effizient parallelisieren. Die numerischen Ergebnisse der bikubischen Spline-Interpolation auf Dreieckselementen entsprechen anderen in der Literatur genannten Ergebnissen auf regulären Rechtecksgittern.

Die Methoden aus den Teilen I und II werden im dritten Teil zur ASLM für die Flachwassergleichungen zusammengefügt und erweitert. Das gekoppelte System von Differentialgleichungen wird zunächst in dimensionslose Form überführt. Eine semi-implizite mit Hilfe der SLM diskretisierte Form der Gleichungen wird dann auf einem adaptiven Gitter unter Verwendung des elliptischen FEM Lösers aus dem ersten Teil implementiert. Die Parallelisierung erfolgt mit den zuvor entwickelten Strategien. Ein speziell angepaßter Fehlerschätzer wird zur Steuerung der Adaptivität eingesetzt.

Die Testrechnungen werden mit den Ergebnissen eines Referenzmodells, das auf einer Finite-Differenzen-Diskretisierung basiert, verglichen. Die Güte der Ergebnisse ist überzeugend. Durch Parallelisierung erreicht die ASLM schon für das glatte und regulär berandete Modellproblem der Flachwassergleichungen die gleiche zeitliche Ordnung wie das numerisch einfache Referenzmodell. Anhand des Modellproblems kann der Nachweis erbracht werden, daß die Kombination von Semi-Lagrange-Zeitdiskretisierung und adaptiver Finite-Elemente-Methode ein effizientes, flexibles und genaues Verfahren zur Lösung von geophysikalischen Strömungsproblemen ergibt.

Abstract

Shallow-water-equations are used as an example for the development, implementation and parallelization of the new adaptive semi-Lagrangian finite element method (ASLM). Within three parts of this thesis datastructures, algorithms and methods for implementing the parallel ASLM are described.

The first part is devoted to the finite element method (FEM). An introduction is given, the most important techniques for the implementation are introduced, new datastructures for adaptive grid generation are developed, parallelization of the different program segments is performed, and experimental results are shown. Two new parallelization strategies for assembling the discretization matrix, the *color-indexing* method based on color-tables, and the *index-array method* prove to be efficient and scalable. The index-array method is reused and adopted in the sequel in all parallelizations. Preconditioned CG-like iterative methods are used as solvers for the linear systems of equations.

In the second part the semi-Lagrangian method (SLM) for the passive advection equation on an adaptive grid is introduced. Special care has to be taken when implementing the interpolation within an adaptive algorithm. Parallelization of the SLM is easily possible, because all operations corresponding to grid-points are independent. The routines, representing the SLM, are parallelized efficiently. Numerical results of a bi-cubic spline interpolation on triangular elements correspond to results for regular and rectangular grids, published elsewhere.

The methods derived in parts I and II are merged and enhanced in the third part to obtain the ASLM for the shallow-water-equations. The coupled system of differential equations is transformed into a dimensionless form. A semi-implicit semi-Lagrangian discrete form of the equations is then implemented, using the elliptic solver from part I. Parallelization is achieved by the use of the strategies derived before. A specially trimmed a posteriori error estimator controls adaptivity of the algorithm.

Test calculations are compared with results from a reference model, which is based on a finite-difference discretization. The qualitiy of these results is convincing. By parallelization the ASLM features the same order of time-complexity as the much simpler reference model, even with a smooth and regularly bounded model problem of the shallow-water-equations. The model problem shows that the combination of semi-Lagrangian time-discretization with adaptive finite-element space-discretization yields an efficient, flexible and accurate method for solving geophysical fluid dynamics problems.

Einleitung

Einer der Arbeitsschwerpunkte des *Alfred-Wegener-Instituts für Polar- und Meeresforschung* (AWI) in Bremerhaven, an dem diese Arbeit erstellt wurde, ist die numerische Simulation der Zirkulation des Südlichen Ozeans. Ozeanmodellierung ist eine der sogenannten „grand challenges", der großen Herausforderungen des wissenschaftlichen Rechnens. Zu den „grand challenges" gehören Probleme aus der Moleküldynamik, der Strömungsdynamik und Konstruktion, die mit der verfügbaren Hardware und Numerik der neunziger Jahre noch nicht befriedigend gelöst werden können [123].

Um den großen Anforderungen an Modellierungssoftware begegnen zu können, existiert am AWI eine Arbeitsgruppe „Parallelisierung von Ozeanmodellen". Die Gruppe hat zwei Arbeitsschwerpunkte: Die Parallelisierung vorhandener Modellierungssoftware und die Entwicklung neuer paralleler numerischer Methoden zur Verwendung in Ozeanmodellen [23, 71, 116]. Das Ziel der vorliegenden Arbeit ist die Entwicklung, Implementierung und Parallelisierung neuer numerischer Methoden für die Ozeanmodellierung.

Die Simulation von ozeanischen oder atmosphärischen Zirkulationen ist eine junge Wissenschaftsdisziplin. Die frühen Ansätze zur Modellierung atmosphärischer Prozesse waren von dem Bestreben motiviert, das Wetter vorherzusagen. Einer der ersten Artikel, die sich mit der mathematischen Formulierung der atmosphärischen Bewegungsgleichungen beschäftigen, wurde von Bjerknes Anfang dieses Jahrhunderts veröffentlicht [26]. Das in der Arbeit von 1904 vorgeschlagene graphische Lösungsverfahren für die analytisch nicht lösbaren partiellen Differentialgleichungen erwies sich als ungeeignet.

Ein erster Ansatz zur numerischen Lösung von atmosphärischen Bewegungsgleichungen mit Hilfe von finiten Differenzen stammt von Richardson 1922 [111]. Aufgrund der Verletzung eines Stabilitätskriteriums, das erst 1928 von Courant, Friedrichs und Lewy [36] entdeckt wurde, waren die Berechnungen aber so weit von den gemessenen Werten entfernt, daß die numerische Wettervorhersage vorerst wieder verworfen wurde.

Die erfolgreiche Modellierung von Ozean und Atmosphäre setzte die Entwicklung speicherprogrammierbarer Computer voraus. Das erste echte Wettermodell wurde folgerichtig im Umfeld John von Neumanns, einem der „Väter des Computers", entwickelt [31]. Das von Charney et al. 1950 auf dem „Electronic Numerical Integrator and Computer" (ENIAC) implementierte Modell löst die barotrope Vorticitygleichung in einem begrenzten Rechengebiet. Ein globales Zirkulationsmodell (GZM), dem die quasi-geostrophischen Gleichungen zugrunde liegen und das in zwei Schichten rechnet, wurde von Phillips in der zweiten Hälfte der fünfziger Jahre vorgeschlagen [103].

Smagorinsky et al. implementierten 1965 das erste globale Atmosphärenmodell, dem die „Primitiven Gleichungen" (PG) zugrunde liegen [124]. Wesentliche numerische Verbesserungen wurden durch die Entwicklung der schnellen Fourier-Transformation (FFT) zur Lösung der auftretenden elliptischen Probleme und der spektralen Transformationsmethode zur Entkopplung des Gleichungssystems Ende der sechziger Jahre eingeführt [34, 100]. In den siebziger und achziger Jahren wurden die Modelle vor allem bezüglich der physikalischen Darstellung bestimmter Prozesse verbessert [140, 136].

Ähnlich wie die Entwicklung von numerischen Atmosphärenmodellen hat sich – mit leichter zeitlicher Verzögerung – auch die der Ozeanmodelle vollzogen. Erste Ozeanmodelle wurden Anfang der sechziger Jahre vorgestellt [28, 115]. Das erste globale dreidimensionale PG-Modell des Ozeans wurde von Bryan und Cox 1967 entwickelt [29]. Es gibt eine ganze Familie verschiedener Weiterentwicklungen dieses Modells. Semtner und Cox haben wesentliche Erweiterungen in den siebziger und achtziger Jahren zugefügt, die sich auf die Darstellung der Topographie und der physikalischen Prozesse beziehen [37, 120].

Von Holland et al. stammt ein Ozeanmodell, dessen Implementierung auf den quasi-geostrophischen (QG) Gleichungen beruht [64]. Auch von diesem Modell gibt es einige Derivate, unter anderem stammt das am AWI verwendete Modell des Südlichen Ozeans [75] von Hollands Modell ab. Eine neue Implementierung der QG-Gleichungen wurde von Wolff vorgenommen [144].

Eine Übersicht über die zur Zeit aktuellen globalen Zirkulationsmodelle zu geben, ist an dieser Stelle nicht möglich. Jedes Institut erweitert vorhandene Software nach den eigenen Bedürfnissen, so daß die Taxonomie erheblich erschwert wird. Wesentlichen Einfluß auf die Gemeinde der ModelliererInnen haben zur Zeit die folgenden Modelle:

- MOM (Modular Ocean Model) von Pacanowski et al. am *Geophysical Fluid Dynamics Laboratory* (GFDL) in Princeton [101].
- SPEM (Semi-Spectral Primitive Equation Model) von Haidvogel et al., *Johns Hopkins Universität* in Baltimore [56].
- POP (Parallel Ocean Program), eine parallele und umformulierte Version des Bryan-Cox-Modells vom *Los Alamos National Laboratory* (LANL) [125].
- FRAM (Fine Resolution Antarctic Model); bei FRAM handelt es sich eher um ein Projekt, das verwendete Modell basiert auf MOM, hat aber eine freie Oberfläche [72, 133].

Finite-Elemente-Methoden (FEM) sind in den globalen Zirkulationsmodellen des Ozeans bisher kaum vertreten. Erste Versuche, die Flexibilität der Methode für dieses Anwendungsfeld zu erschließen, wurden zwar schon Mitte der siebziger Jahre von Fix unternommen [42] und weitere Untersuchungen stammen von le Provost [79]. Aufgrund des relativ hohen numerischen Aufwandes konnten sich diese Ansätze jedoch nicht in globalen Modellen durchsetzen. Die vorliegende Arbeit zielt darauf ab, mit Hilfe neuer numerischer Verfahren und unter Ausnutzung paralleler Hochleistungscomputer die Grenzen, die der FEM durch den numerischen Aufwand gesetzt sind, zu erweitern.

Viele der genannten Ozeanmodelle enthalten numerische Verfahren der sechziger und siebziger Jahre. Es handelt sich dabei um Finite-Differenzen-Verfahren oder Spektral-Methoden, die auf regulären rechteckigen Gittern arbeiten [101]. Lange Zeit wurde vor allem Arbeit in bessere Parametrisierungen der physikalischen Prozesse investiert. Numerische Verbesserungen wurden erst in den letzten Jahren begonnen [41, 125].

Um zukünftigen Aufgaben der Ozeanmodellierung gewachsen zu sein, muß flexible und moderne Modellierungssoftware zur Verfügung gestellt werden. Einige Methoden zur Berechnung ozeanischer Zirkulation werden in dieser Arbeit neu entwickelt oder kombiniert. Dabei stehen mehrere

Überlegungen am Anfang der Entwicklung. Zunächst soll die Finite-Elemente-Methode als eine gegenüber Finite-Differenzen-Verfahren wesentlich flexiblere Methode zur Lösung von partiellen Differentialgleichungen in die Ozeanmodellierung Eingang finden. Erfahrungen mit Finite-Elemente-Verfahren legen es nahe, gleichzeitig sehr effiziente Verfahren zur Gleichungslösung, zur Zeitdiskretisierung und zur Parallelisierung zu verwenden, um die Flexibilität der finiten Elemente nicht mit einer erheblichen Verlängerung der Rechenzeit erkaufen zu müssen [58, 79].

Diese Anforderung führt zur Wahl der Semi-Lagrange-Zeitdiskretisierung. Einerseits erscheint die potentiell gute Parallelisierbarkeit sehr aussichtsreich. Andererseits erlaubt die unbedingte Stabilität des Verfahrens große Zeitschritte, so daß Rechenzeit gespart werden kann. Zusätzlich erscheint es ratsam, adaptive Strukturen einzuführen, so daß bei hoher (lokaler) Auflösung der relevanten physikalischen Prozesse moderate Anforderungen an Speicherplatz und Rechenaufwand erhalten bleiben.

Die vorliegende Arbeit beschreibt die Entwicklung, Implementierung und Parallelisierung eines Verfahrens zur Lösung der Flachwassergleichungen. Dieses gekoppelte Gleichungssystem für die horizontalen Geschwindigkeitskomponenten und den Druck (bzw. die geopotentielle Höhe) ähnelt den barotropen Gleichungen in PG-Modellen. Damit lassen sich die gewonnenen Ergebnisse auf komplexere Modelle übertragen.

Der Schwerpunkt der Arbeit liegt auf der Implementierung und Parallelisierung. Dazu werden einige neue Datenstrukturen entworfen, und neue Parallelisierungsstrategien entwickelt. Die Kombination der adaptiven Finite-Elemente-Methode mit Semi-Lagrange-Zeitdiskretisierung auf Parallelrechnern ist dabei vollkommen neu. Ansätze für die Parallelisierung von Semi-Lagrange-Verfahren sind bei Thomas und Côté zu finden [134]. Die Autoren verwenden aber kein adaptives Verfahren und auch keine Finite-Elemente-Methode. Ein Semi-Lagrange-Verfahren auf einem unstrukturierten Dreiecksgitter mit möglichen lokalen Verfeinerungen wird zur Zeit von Le Roux et al. entwickelt, von Adaptivität und Parallelität ist jedoch keine Rede [80].

Insgesamt ist die Forschungslandschaft auf dem Gebiet der numerischen Verfahren für ozeanische oder atmosphärische Zirkulation in der letzten Zeit in Bewegung geraten. Am *Europäischen Zentrum für Mittelfristige Wettervorhersage* (ECMWF) wird mit Dreiecksgittern experimentiert [97], am *Centre de Recherche en Calcul Appliqué* in Quebec wird an Semi-Lagrange-Verfahren auf unstrukturierten Gittern gearbeitet, der Bedarf an effizienten Lösungsverfahren ist vorhanden [145, 149], flexible Randbehandlung wird verlangt [141] und neue numerische Advektionsverfahren werden entwickelt [129, 152]. Gleichzeitig sind einige Arbeiten zur Verifikation und zum Vergleich der Modelle erschienen [99, 33]. Es ist daher zu erwarten, daß einige neue angepaßte Programme erscheinen werden, die parallelisiert sind oder adaptiv rechnen oder sich durch erhöhte numerische Stabilität und Genauigkeit auszeichnen. In diese Strömung paßt das hier vorgestellte Verfahren.

Aufbau der Arbeit

Die Arbeit ist in drei Teile gegliedert. Teil I ist der Behandlung der Finite-Elemente-Methode gewidmet. Teil II beschreibt die Semi-Lagrange-Methode für die Advektionsgleichung und in Teil III wird durch die Kombination der Methoden ein Modell für die Flachwassergleichungen eingeführt.

Alle drei Hauptteile sind (fast) unabhängig voneinander zu verstehen. Natürlich gibt es Querverweise, wenn beispielsweise im Semi-Lagrange-Verfahren auf die Gittergenerierung Bezug genommen wird, welche im Teil über finite Elemente ausführlich beschrieben ist. Aber es ist versucht worden, nur die wesentlichen und spezifischen Eigenschaften jedes Verfahrens im zugehörigen Teil zu behandeln. Dadurch sollen Wiederholungen vermieden werden.

Jeder der drei Teile ist in gleicher Weise in vier Kapitel untergliedert. Im ersten Kapitel wird eine Einführung in das Thema gegeben und die Methode wird vorgestellt. Auf Einzelheiten bei der Implementierung des jeweiligen numerischen Verfahrens wird im zweiten Kapitel eingegangen. Das dritte Kapitel behandelt die Parallelisierung des Verfahrens. Im letzten Kapitel jedes Teils werden Ergebnisse aus Testrechnungen präsentiert.

Die einführenden Kapitel erklären die jeweilige Methode, nennen Modellprobleme, zählen verschiedene Verfahrensvarianten auf und legen theoretische Grundlagen.

In den Kapiteln zur Implementierung wird auf die für die Programmierung wichtigen Details der Verfahren eingegangen. Neben eigenen Lösungsansätzen werden einige selten veröffentlichte Techniken vorgestellt, die die Datenstruktur oder die Berechnung einiger häufig gebrauchter Größen (die Dreiecksfläche oder die ersten partiellen Ableitungen) betreffen.

In den jeweils dritten Kapiteln werden die Ansätze zur Parallelisierung der Methode und die parallele Implementierung dargestellt. Zum Teil beschränken sich die Ansätze darauf, eine effiziente Aufteilung der Daten zu erreichen, zum Teil werden aber auch Algorithmen modifiziert, so daß sie sich besser für die Parallelisierung eignen.

Ergebnisse werden jeweils im vierten Kapitel jedes Teils vorgestellt. Diverse Fragestellungen, die jeweils in der Übersicht erläutert sind, werden anhand von Testrechnungen behandelt. Dazu gehören die numerische Genauigkeit oder die parallele Effizienz der Verfahren.

Jedes Kapitel beginnt mit einem Abschnitt, der einen Überblick über die Motivation gibt, mit der die jeweiligen Themen gewählt wurden, der in die Themen einführt, und kurz die grundsätzlichen Ansätze und Lösungen darstellt. Diese *Übersicht* genannten Abschnitte erleichtern den verschiedenen Leserkreisen das Verständnis der Arbeit. OzeanographInnen können im Teil über finite Elemente leicht einen einführenden Zugang zur Methode gewinnen, wenn sie lediglich die Übersichten lesen. Umgekehrt wird es MathematikerInnen erleichtert, in die Grundbegriffe der Ozeanmodellierung einzusteigen, wenn sie die Übersichten der beiden letzten Teile lesen.

Die Kapitel über Parallelisierung setzen einige Begriffe aus der Theorie paralleler Systeme voraus. Die entsprechenden Notationen und Definitionen sind im Anhang zusammengefaßt. Dabei kann selbstverständlich keine umfassende Einführung in parallele Systeme gegeben werden. Grundlegend unterschiedliche Architekturen und deren Eigenschaften werden vorgestellt, einige Programmiermodelle eingeführt und Parametrisierungen definiert.

Danksagungen

Besonders danken möchte ich Andrew Staniforth (*Recherche en prévision numérique, Environnement Canada, Québec*), der mir den Weg zur Entwicklung von Semi-Lagrange-Verfahren zeigte und mir immer wieder hilfreiche Hinweise zur Fehlersuche gab. Andreas Hense (*Meteorologisches Institut der Universität Bonn*) gab mir wichtige Ratschläge, wenn es um die physikalische Interpretation der Ergebnisse ging. Wolfgang Hiller (*Alfred-Wegener-Institut*) danke ich für die Chancen, die er mir gab. Den MitarbeiterInnen am Alfred-Wegener-Institut danke ich für zahllose Gespräche und Diskussionen, die mir die Physik des Ozeans und der Atmosphäre näher brachten.

Teil I

Die Finite-Elemente-Methode (FEM) für elliptische Probleme

Kapitel 1

Einführung in die Finite-Elemente-Methode

1.1 Übersicht

Viele physikalische Probleme lassen sich in Form von (partiellen) Differentialgleichungen ausdrücken. Um diese Gleichungen mit Hilfe eines Computers lösen zu können, müssen sie *diskretisiert* werden, das heißt die kontinuierliche Gleichung mit unendlich vielen Freiheitsgraden wird in eine diskrete Gleichung mit endlich vielen Freiheitsgraden überführt.

Die gebräuchlichste Methode der Diskretisierung von Differentialgleichungen ist die *Finite-Differenzen-Methode* (FDM). Dabei werden Ableitungen durch Differenzenquotienten ersetzt [87]. Eine andere Methode ist die *Finite-Elemente-Methode* (FEM). Sie soll im folgenden vorgestellt werden. Die FEM bietet gegenüber der FDM einige wesentliche Vorteile, die sich vor allem auf die Behandlung der Ränder beziehen, und die diese Methode gerade für die Simulation von Ozeanströmungen geeignet erscheinen lassen.

Um eine Differentialgleichung

$$Du = f \text{ in } \mathcal{G} \tag{1.1}$$

mit einem Differentialoperator D und geeigneten Randbedingungen mit Hilfe der FEM zu diskretisieren, formuliert man zunächst eine schwache Formulierung der Gleichung:

$$\text{Finde } u \in \mathcal{V}, \text{ so daß } a(u,v) = f(v)\ \forall v \in \mathcal{V}. \tag{1.2}$$

Dabei ist $\mathcal{V}$ ein Funktionenraum, $a(\cdot,\cdot)$ eine Bilinearform $a : \mathcal{V} \times \mathcal{V} \to \mathcal{R}$, f eine Linearform $f : \mathcal{V} \to \mathcal{R}$. Im Falle elliptischer Gleichungen ist die Variationsformulierung (1.2) äquivalent zu einem Minimierungsproblem.

$$\text{Finde } u \in \mathcal{V}, \text{ so daß } \mathcal{F}(u) \leq \mathcal{F}(v)\ \forall v \in \mathcal{V}, \tag{1.3}$$

wobei $\mathcal{F} : \mathcal{V} \to \mathcal{R}$ ein Funktional ist (siehe z.B. [117]). Prinzipiell wird die schwache Formulierung mit Hilfe von Testfunktionen und einem Skalarprodukt hergeleitet. Die Herleitung der schwachen Formulierung für ein Modellproblem ist im Abschnitt 1.2 angegeben.

Die theoretischen Grundlagen der FEM (Sobolev-Räume, Variationsrechnung) wurden schon gegen Ende des letzten Jahrhunderts gelegt. Die Methode selbst wurde zu Anfang der 60er Jahre von Ingenieuren für die Berechnung von strukturmechanischen Problemen entwickelt [45]. In den 70er Jahren dieses Jahrhunderts wurden die Verfahren verfeinert und erweitert [32]. Heute findet die FEM in vielen weiteren Bereichen wie der Strömungsdynamik oder der Modellierung von plasto-elastischen Verformungen Anwendung [151].

Das Problem (1.2) wird diskretisiert, indem statt des unendlichdimensionalen Funktionenraumes $\mathcal{V}$ ein endlichdimensionaler Raum $\mathcal{V}_h$ gewählt wird, der in einem zu definierenden Sinne den Raum $\mathcal{V}$ annähert. Mit Hilfe der endlich vielen Basisfunktionen des Raumes $\mathcal{V}_h$ kann dann ein diskretes Minimierungsroblem oder ein (großes) lineares oder nichtlineares Gleichungssystem formuliert werden. FEM-Basisfunktionen erden so gewählt, daß die Diskretisierungsmatrix schwach besetzt ist. Solche Basisfunktionen existieren auf einer Unterteilung des Gebietes in kleine (finite) Elemente, so daß ihr Träger jeweils nur auf wenigen Elementen definiert ist.

Die Attraktivität der FEM liegt vor allem in der Flexibilität der Methode. Durch geeignete Wahl von Basisfunktionen lassen sich auch komplizierte Probleme gut abbilden. Randbedingungen werden durch die Wahl des diskreten Raumes elegant dargestellt. Komplizierte Geometrien sind leicht zu diskretisieren. Diesen Vorteilen stehen die notwendige Herleitung des Variationsproblems und der meist höhere numerische Aufwand der Methode (verglichen mit einfachen FDM) gegenüber.

Im Prinzip sind vier Schritte notwendig, um eine gegebene Differentialgleichung mit Hilfe der FEM zu lösen [67]:

1. Herleitung der Variationsformulierung zum gegebenen Problem.
2. Diskretisierung der Variationsformulierung.
3. Lösung des diskreten Problems.
4. Implementierung der Methode auf einem Computer.

Der Schwerpunkt der Darstellung in diesem Teil wird auf den Schritten 3 und 4 liegen.

Die weiteren Abschnitte dieses Kapitels widmen sich der Definition einiger Modellprobleme und ihrer Diskretisierung (Abschnitt 1.2). Der Einfluß verschiedener Randbedingungen wird ebenso dargestellt wie die Behandlung verschiedener Differentialoperatoren. Schließlich wird die Definition der finiten Elemente vorgenommen und die FEM-Basisfunktionen werden eingeführt. Dabei werden ausschließlich elliptische Probleme behandelt. Motiviert ist diese Wahl durch das Auftreten eines elliptischen Problems bei der semi-impliziten Semi-Lagrange-Diskretisierung der Flachwassergleichungen im Teil III.

1.2 Modellprobleme und ihre Variationsformulierung

Das klassische elliptische Modellproblem ist durch die *Potentialgleichung* oder *Laplace-Gleichung* gegeben:

$$\triangle u = 0 \text{ im Gebiet } \mathcal{G} \in \mathcal{R}^d. \tag{1.4}$$

$\mathcal{G}$ sei dabei ein zusammenhängendes Gebiet im $\mathcal{R}^d$ mit $d = 1, 2, 3$, der Laplace-Operator $\triangle$ ist durch

$$\triangle u = \sum_{k=1}^{d} \frac{\partial^2 u}{\partial x_k^2}$$

definiert. Für die Eindeutigkeit der Lösung müssen noch Randwerte

$$u = g \text{ auf dem Rand } \Gamma = \partial\mathcal{G}, \tag{1.5}$$

mit $g \in C^0(\Gamma)$, vorgegeben werden. Die Potentialgleichung ist (mit entsprechenden Randbedingungen) eindeutig lösbar. Die Lösungen besitzen eine Mittelwerteigenschaft und und erfüllen ein Maximum-Minimum-Prinzip, das heißt sie hängen stetig und eindeutig von den Randwerten ab (zu den Eigenschaften der Potentialgleichung siehe beispielsweise [55]).

Der Differentialoperator und die Randbedingungen von (1.4) werden in diesem Abschnitt in verschiedenen Varianten vorgestellt. Die Potentialgleichung mit nichthomogener rechter Seite heißt *Poissongleichung*. Für die Poissongleichung wird die Herleitung der Variationsformulierung prinzipiell nachvollzogen. Für die theoretischen Grundlagen wird auf die Literatur verwiesen (z.B. [32, 67, 117, 139]). Die Variationsformulierungen der anderen Modellprobleme werden meist nur angegeben.

1.2.1 Die Poissongleichung und einige ihrer Eigenschaften

Die *Poissongleichung* mit homogenen *Dirichlet-Randbedingungen* ist gegeben durch:

$$\begin{aligned} -\triangle u &= f \quad \text{in } \mathcal{G}, \\ u &= 0 \quad \text{auf } \Gamma. \end{aligned} \tag{1.6}$$

Zur Vereinfachung sei $\mathcal{G} \in \mathcal{R}^2$.

Ein physikalisches Problem, das von (1.6) beschrieben wird, ist die Auslenkung u einer elastischen Membran, die am Rand fest eingespannt ist und auf die eine Kraft f wirkt. Andere mögliche Interpretationen für u sind das elektro-magnetische Potential, wobei f die Ladungsdichte ist, oder die Temperatur mit einer Energiedichte f.

Aus (1.6) soll nun die Variationsformulierung hergeleitet werden. Dazu sei $u \in \mathcal{V} = C^2(\mathcal{G}) \cap H_0^1(\mathcal{G})$ klassische Lösung von (1.6). Dabei bezeichne $C^2(\mathcal{G})$ die zweimal stetig differenzierbaren Funktionen auf $\mathcal{G}$ und $H_0^1(\mathcal{G})$ den *Sobolev-Raum* der Lebesgue-meßbaren Funktionen $v \in L^2(\mathcal{G})$ deren erste schwache Ableitungen ebenfalls in $L^2(\mathcal{G})$ liegen und deren Träger kompakt in $\mathcal{G}$ enthalten sind. $\mathcal{V}$ ist mit dem Skalarprodukt

$$(u, v) := (u, v)_{L^2(\mathcal{G})} := \int_{\mathcal{G}} u\, v \, d\mathcal{G} \tag{1.7}$$

ein Hilbert-Raum. Wähle eine beliebige *Testfunktion* $v \in C_0^\infty(\mathcal{G})$ bzw. $v \in \mathcal{V}$ und skalarmultipliziere (1.6) auf beiden Seiten:

$$(-\triangle u, v) = \int_{\mathcal{G}} -\triangle u\, v \, d\mathcal{G} = \int_{\mathcal{G}} f\, v \, d\mathcal{G} = (f, v).$$

Es gilt (partielle Integration bzw. *Greensche Formel*):

$$\int_{\mathcal{G}} \nabla u \cdot \nabla v \, d\mathcal{G} = \int_{\Gamma} u \, \frac{\partial v}{\partial n} \, d\Gamma - \int_{\mathcal{G}} \triangle u\, v \, d\mathcal{G}, \tag{1.8}$$

wobei $\frac{\partial v}{\partial n}$ die Normalenableitung von v ist. Unter Berücksichtigung der Randwerte $u = 0$ auf Γ folgt aus (1.8):

$$\int_{\mathcal{G}} \nabla u \cdot \nabla v \, d\mathcal{G} = \int_{\mathcal{G}} -\triangle u\, v \, d\mathcal{G}.$$

Da $v \in \mathcal{V}$ beliebig gewählt und mit der Notation aus (1.7) folgt:

$$(\nabla u, \nabla v) = (f, v) \; \forall v \in \mathcal{V}. \tag{1.9}$$

Das heißt, wenn u eine Lösung von (1.6) ist, dann ist u auch eine Lösung von (1.2) mit den Notationen

$$\begin{aligned} \mathcal{V} &= H_0^1(\mathcal{G}), \\ a(u, v) &= \int_{\mathcal{G}} \nabla u \cdot \nabla v \, d\mathcal{G}, \\ f(v) &= \int_{\mathcal{G}} f\, v \, d\mathcal{G}. \end{aligned} \tag{1.10}$$

Die umgekehrte Folgerung ist – unter gewissen Annahmen an die Regularität der Funktion u – analog möglich.

Damit läßt sich zusammenfassen:

Bemerkung 1.1 (Poissonproblem mit homogenen Dirichlet-Randbedingungen)

Sei die Poissongleichung (1.6) gegeben. Dann lautet die zugehörige Variationsformulierung:

$$\textit{Finde } u \in H_0^1(\mathcal{G}), \textit{ so daß } a(u,v) = f(v)\ \forall v \in H_0^1(\mathcal{G}),$$

mit

$$\begin{aligned} a(u,v) &= \int_{\mathcal{G}} \nabla u \cdot \nabla v \, d\mathcal{G}, \\ f(v) &= \int_{\mathcal{G}} f\, v \, d\mathcal{G}. \end{aligned}$$

Es bleibt noch nachzutragen, wie das Minimierungsproblem (1.3) aus der Variationsformulierung (1.2) folgt. Dazu wird zunächst die Form $\mathcal{F}$ mit der Notation aus (1.10) definiert als:

$$\mathcal{F}(v) := \frac{1}{2}\, a(v,v) - f(v). \tag{1.11}$$

Sei $u \in \mathcal{V}$ Lösung des Variationsproblems (1.2). Sei weiter $v \in \mathcal{V}$ und setze $w = v - u$, so daß $w \in \mathcal{V}$. Dann folgt:

$$\begin{aligned} \mathcal{F}(v) &= \mathcal{F}(u+w) \\ &= \frac{1}{2}\, a(u+w, u+w) - f(u+w) \\ &= \frac{1}{2}\, a(u,u) - f(u) + a(u,w) - f(w) + \frac{1}{2}\, a(w,w) \\ &\geq \mathcal{F}(u), \end{aligned}$$

da wegen (1.9) $a(u,w) - f(w) = 0$ und $a(w,w) \geq 0$. Umgekehrt sei u eine Lösung von (1.3). Dann gilt für jedes $v \in \mathcal{V}$ und jede reelle Zahl ϵ:

$$\mathcal{F}(u) \leq \mathcal{F}(u + \epsilon v).$$

Definiere die differenzierbare Funktion

$$\begin{aligned} g(\epsilon) &:= \mathcal{F}(u + \epsilon v) \\ &= \frac{1}{2}\, a(u,u) + \epsilon a(u,v) + \frac{\epsilon^2}{2}\, a(v,v) - f(u) - \epsilon f(v). \end{aligned}$$

g hat ein Minimum in $\epsilon = 0$, d.h. $g'(0) = 0$. Da

$$g'(0) = \frac{1}{2}\, a(u,v) - f(v),$$

folgt, daß u Lösung von (1.2) ist.

In den weiteren Abschnitten wird nur noch die Variationsformulierung (1.2) verwendet. Das Minimierungsproblem sollte aber ebenfalls beschrieben werden, weil es in manchen Fällen die adäquatere Formulierung ist.

1.2.2 Verschiedene Randbedingungen

Eine Stärke der FEM ist die elegante Behandlung der Randbedingungen. Diese Eigenschaft beruht auf der Wahl der Testfunktionen bzw. auf den Eigenschaften der Variationsformulierung. Folgende Randbedingungen zur Poissongleichung (1.6) und deren Formulierung als Variationsproblem sollen beschrieben werden:

1. Inhomogene Dirichlet-Randbedingungen: $u = g$ auf Γ.
2. Neumann-Randbedingungen: $\frac{\partial u}{\partial n} = q$ auf Γ.
3. Gemischte Randbedingungen: $u = g$ auf $\gamma \subset \Gamma$ und $\frac{\partial u}{\partial n} = q$ auf $\Gamma \backslash \gamma$.

Die inhomogene Dirichlet-Randbedingung aus 1. läßt sich abbilden, indem die Lösung u jetzt im Raum $\mathcal{V} = H^1(\mathcal{G})$ mit $v \in \mathcal{V}$ so, daß $v|_\Gamma = g$ gesucht wird. Zusammengefaßt:

Bemerkung 1.2 (Poissonproblem mit inhomogenen Dirichlet-Randbedingungen)

Sei die Poissongleichung mit inhomogenen Dirichlet-Randbedingungen gegeben:

$$\begin{aligned} -\triangle u &= f \quad in\ \mathcal{G}, \\ u &= g \quad auf\ \Gamma. \end{aligned}$$

Dann lautet das zugehörige Variationsproblem:

$$Finde\ u \in H^1(\mathcal{G}),\ u|_\Gamma = g,\ so\ daß\ a(u,v) = f(v)\ \forall v \in H^1(\mathcal{G}),$$

mit

$$\begin{aligned} a(u,v) &= \int_{\mathcal{G}} \nabla u \cdot \nabla v\, d\mathcal{G}, \\ f(v) &= \int_{\mathcal{G}} f\, v\, d\mathcal{G}. \end{aligned}$$

Falls es ein $u_0 \in H^1(\mathcal{G})$ gibt mit $u_0|_\Gamma = g$, dann läßt sich die Lösung $u = w + u_0, w|_\Gamma = 0$ des Variationsproblems auch durch das folgende Problem mit homogenen Dirichlet-Randbedingungen charakterisieren:

$$a(w,v) = f'(v) =: f(v) - a(u_0,v), w, v \in H_0^1(\mathcal{G}) \tag{1.12}$$

Neumann-Randbedingungen werden in die Variationsformulierung in folgender Weise integriert: In der Linearform $f(v)$ tritt ein weiteres (Rand-)Integral auf:

$$f(v) = \int_{\mathcal{G}} f\, v\, d\mathcal{G} + \int_\Gamma q\, \frac{\partial v}{\partial n}\, d\Gamma. \tag{1.13}$$

Eine homogene Neumann-Randbedingung $\frac{\partial u}{\partial n} = q = 0$ auf Γ ändert also die Variationsformulierung aus (1.1) nicht. Lediglich der Raum $\mathcal{V}$ ist jetzt $H^1(\mathcal{G})$. Zusammenfassend gilt also für Neumann-Randbedingungen:

Bemerkung 1.3 (Poissonproblem mit Neumann-Randbedingungen)

Sei das Poissonproblem mit Neumann-Randbedingungen

$$\begin{aligned} -\triangle u &= f \quad in\, \mathcal{G}, \\ \frac{\partial u}{\partial n} &= q \quad auf \Gamma, \end{aligned}$$

wobei $q \equiv 0$ zugelassen ist, gegeben. Dann ist eine klassische Lösung dieser Gleichung auch Lösung des Variationsproblems

$$\textit{Finde } u \in H^1(\mathcal{G}), \textit{ so daß } a(u,v) = f(v)\ \forall v \in H^1(\mathcal{G}),$$

mit

$$\begin{aligned} a(u,v) &= \int_{\mathcal{G}} \nabla u \cdot \nabla v \, d\mathcal{G}, \\ f(v) &= \int_{\mathcal{G}} f\, v \, d\mathcal{G} + \int_{\Gamma} q \frac{\partial v}{\partial n} \, d\Gamma, \end{aligned}$$

und umgekehrt.

Gemischte Randbedingungen wie in 3. sind mit den Ergebnissen in Bemerkungen 1.2 und 1.3 leicht zu behandeln. Der Raum der möglichen Lösungsfunktionen muß modifiziert werden. Gesucht ist also ein $u \in H^1_{[\gamma]}(\mathcal{G})$ mit

$$H^1_{[\gamma]}(\mathcal{G}) := \left\{ v \in H^1(\mathcal{G}) : \ u|_\gamma = g \right\}.$$

Außerdem ändert sich die Linearform $f(v)$ zu

$$f(v) = \int_{\mathcal{G}} f\, v \, d\mathcal{G} + \int_{\Gamma \setminus \gamma} q \frac{\partial v}{\partial n} \, d\Gamma.$$

Zusammenfassend kann festgehalten werden, daß Randbedingungen in der Variationsformulierung durch einfache Integraldarstellungen abgebildet werden. Mögliche Lösungsverfahren bleiben davon unberührt. Das steht im Gegensatz zur FDM, wo die Randbedingungen (insbesondere Neumann-Randbedingungen) explizit diskretisiert werden müssen.

1.2.3 Ein allgemeinerer Differentialoperator

Im Teil III wird bei der Diskretisierung der Flachwassergleichungen ein allgemeinerer Differentialoperator auftreten. Daher soll hier die Variationsformulierung für einen elliptischen Differentialoperator

$$D := \sum_{k=1}^{d} \frac{\partial}{\partial x_k} \left(\sum_{l=1}^{d} a_{k,l}(\mathbf{x}) \frac{\partial}{\partial x_l} \right) \tag{1.14}$$

angegeben werden.

Damit D elliptisch ist, muß für die $a_{k,l}$ gelten:

$$\sum_{k,l} a_{k,l}(\mathbf{x}) \xi_k \xi_l \geq 0 \ \forall\, \mathbf{x} \in \mathcal{G}, 0 \neq \xi_l, \xi_k \in \mathcal{R}^d.$$

Zur Vereinfachung der Darstellung seien homogene Dirichlet-Randbedingungen gegeben. Dann lautet die Variationsformulierung zum Differentialoperator (1.14):

$$\text{Finde } u \in H^1_0(\mathcal{G}), \text{ so daß } a(u,v) = f(v)\ \forall v \in H^1_0(\mathcal{G}),$$

mit $f(v)$ wie in (1.10) und

$$a(u,v) = \int_{\mathcal{G}} \left[\sum_{k=1}^{d} \left(\sum_{l=1}^{d} a_{k,l}(\mathbf{x}) \frac{\partial}{\partial x_l} \right) u \cdot \frac{\partial}{\partial x_k} v \right] d\mathcal{G}. \tag{1.15}$$

In (1.15) ist besonders zu beachten, daß die Koeffizienten $a_{k,l}$ wegen der partiellen Integration nicht mehr in der differenzierten Form auftreten.

1.3 Diskretisierung

Die im Abschnitt 1.2 angegebenen Variationsformulierungen sollen mit Hilfe eines Computers gelöst werden. Dazu müssen sie diskretisiert werden. Der erste Schritt der Diskretisierung führt zum sogenannten *Ritz-Galerkin-Verfahren.* Die Idee besteht darin, die Funktionenräume $\mathcal{V} = H^1_\nu(\mathcal{G}), \nu = 0, [\gamma], [\,]$, durch endlichdimensionale Räume $\mathcal{V}_h$ zu approximieren.

Die Lösung des Variationsproblems muß dann wegen der endlich vielen Basen von $\mathcal{V}_h$ nur noch in endlich vielen Variationen gesucht werden. Mit Hilfe der Finite-Elemente-Methode werden die Basisfunktionen so gewählt, daß die Lösung effizient mit Hilfe eines numerischen Verfahrens gefunden werden kann.

1.3.1 Das Ritz-Galerkin-Verfahren

Gegeben sei nun ein Funktionenraum $\mathcal{V}$ (Banach-Raum mit einer Norm $\|\cdot\|_\mathcal{V}$), eine elliptische Bilinearform $a(\cdot,\cdot)$ und eine Linearform $f(\cdot)$ auf $\mathcal{V}$ sowie ein Variationsproblem:

$$\text{Finde } u \in \mathcal{V}, \text{ so daß } a(u,v) = f(v)\ \forall v \in \mathcal{V}. \tag{1.16}$$

Dabei ist $\mathcal{V}$ insbesondere einer der Räume $H^1_0(\mathcal{G})$ oder $H^1(\mathcal{G})$ aus dem Abschnitt 1.2. Es gilt:

$$\dim \mathcal{V} = \infty$$

Die *Ritz-Galerkin-Methode* besteht darin, einen Raum $\mathcal{V}_h$ mit den Eigenschaften

$$\mathcal{V}_h \subset \mathcal{V},\ \dim \mathcal{V}_h = N < \infty$$

zu wählen. $\mathcal{V}_h$ wird mit der Norm $\|\cdot\|_\mathcal{V}$ aus $\mathcal{V}$ ausgestattet und ist damit wieder ein Banach-Raum. Die Bilinearform $a(\cdot,\cdot)$ und die Linearform $f(\cdot)$ sind auch auf $\mathcal{V}_h$ definiert, da $\mathcal{V}_h \subset \mathcal{V}$. Damit läßt sich nun das diskretisierte Variationsproblem stellen:

$$\text{Finde } u_h \in \mathcal{V}_h, \text{ so daß } a(u_h,v) = f(v)\ \forall v \in \mathcal{V}_h. \tag{1.17}$$

Man kann zeigen, daß (1.17) eine eindeutige Lösung, die *Ritz-Galerkin-Lösung*, besitzt und daß diese Lösung die kontinuierliche Lösung aus (1.16) approximiert.

Da $\mathcal{V}_h$ endlichdimensional ist, gibt es eine endliche Basis $\mathbf{b} = \{b_1, \ldots, b_N\}$, die $\mathcal{V}_h$ aufspannt:

$$\mathcal{V}_h = \text{span}\{b_1, \ldots, b_N\}.$$

Jedes $v \in \mathcal{V}_h$ kann mit Hilfe eines Koeffizientenvektors $\mathbf{v} = \{v_1, \ldots, v_N\}$ und der Basis als Linearkombination eindeutig dargestellt werden:

$$v = \sum_{i=1}^{N} v_i \cdot b_i = \mathbf{v}^\top \cdot \mathbf{b}. \tag{1.18}$$

Mit dieser Darstellung kann das diskretisierte Variationsproblem (1.17) umformuliert werden:

$$\text{Finde } u_h \in \mathcal{V}_h, \text{ so daß } a(u_h,b_i) = f(b_i)\ \forall i = 1, \ldots, N. \tag{1.19}$$

Die Linearkombination (1.18) definiert eine isomorphe Abbildung des $\mathcal{R}^N$ (Koeffizientenvektoren) in den Raum $\mathcal{V}_h$:

$$\mathbf{P} : \mathcal{R}^N \to \mathcal{V}_h, \quad \mathbf{P}(\mathbf{v}) = \mathbf{v}^\top \cdot \mathbf{b} = v, \quad \mathbf{P}^{-1}(v) = \mathbf{v}$$

Für u_h aus (1.19) wird nun analog zu (1.18) der Ansatz

$$u_h = \sum_{i=1}^{N} u_{h,i} \cdot b_i = \mathbf{u}_h^\top \cdot \mathbf{b}$$

genommen und in $a(\cdot,\cdot)$ eingesetzt. Dann folgt:

$$a\left(\sum_{i=1}^{N} u_{h,i} \cdot b_i, b_j\right) = \sum_{i=1}^{N} u_{h,i} a(b_i, b_j) = f(b_j).$$

Mit $\mathbf{A} := (A_{ij})$, $A_{ij} := a(b_i, b_j)$, $\mathbf{f} := (f_j)$ und $f_j := f(b_j)$ wird ein lineares Gleichungssystem

$$\mathbf{A}\mathbf{u}_h = \mathbf{f} \tag{1.20}$$

definiert, das mit dem Variationsproblem (1.19) äquivalent ist. Das heißt, u_h ist eine Lösung von (1.19) genau dann, wenn der Koeffizientenvektor $\mathbf{u}_h = \mathbf{P}^{-1}(u_h)$ eine Lösung des Systems (1.20) ist.

Damit ist das Variationsproblem (1.16) diskretisiert. Die Aufgabe ist nun, ein lineares Gleichungssystem zu lösen, dessen Matrixelemente sowie rechte Seite aus Integralen über Produkten von Basisfunktionen (bzw. deren Ableitungen) besteht. Leider eignet sich diese Matrix im allgemeinen nicht besonders gut für die numerische Berechnung der Lösung, weil sie voll besetzt ist. Die im nächsten Abschnitt beschriebene Kunst besteht nun darin, besondere Basisfunktionen zu finden, so daß die Matrix schwach besetzt ist.

Als Einschub soll an dieser Stelle erklärt werden, in welchem Sinne $\mathcal{V}$ durch $\mathcal{V}_h$ angenähert wird. Zunächst gilt folgende Fehlerabschätzung (siehe z.B. [55]):

Satz 1.4 (Fehlerabschätzung) *Sei $u \in \mathcal{V}$ eine Lösung des (kontinuierlichen) Variationsproblems (1.16) und $u_h \in \mathcal{V}_h$ eine Ritz-Galerkin-Lösung des Problems (1.17). Dann gilt für den Diskretisierungsfehler die folgende Abschätzung:*

$$\|u - u_h\|_{\mathcal{V}} \leq C \inf_{w \in \mathcal{V}_h} \|u - w\|_{\mathcal{V}} \tag{1.21}$$

Bezeichne d den Abstand der Funktion u von $\mathcal{V}_h$:

$$d(u, \mathcal{V}_h) := \inf_{w \in \mathcal{V}_h} \|u - w\|_{\mathcal{V}}$$

Der Diskretisierungsfehler wird klein (d.h. $\mathcal{V}_h$ approximiert $\mathcal{V}$), wenn es eine Folge von Unterräumen $\mathcal{V}_l \subset \mathcal{V}$ gibt, die in folgendem Sinne gegen $\mathcal{V}$ strebt:

Satz 1.5 (Approximation durch diskrete Unterräume) *Sei $\mathcal{V}_l \subset \mathcal{V}$ eine Folge von Unterräumen mit $\dim \mathcal{V}_l = N_l < \infty$, $l \in \mathcal{N}$ und*

$$\lim_{l \to \infty} d(u, \mathcal{V}_l) = 0 \quad \forall u \in \mathcal{V}.$$

Falls die diskreten Probleme (1.17) mit $\mathcal{V}_l$ eine Lösung u_l besitzen und $a(\cdot,\cdot)$ stetig und beschränkt ist, dann besitzt auch das Problem (1.16) eine Lösung und die Ritz-Galerkin-Lösungen u_l konvergieren gegen u:

$$\|u - u_l\|_{\mathcal{V}} \to 0 \quad \text{für } l \to \infty.$$

Bemerkung 1.6 *Eine Folge von Unterräumen, die die Bedingungen des Satzes erfüllt, ist gegeben durch die Reihe:*

$$\mathcal{V}_1 \subset \mathcal{V}_2 \subset \cdots \subset \mathcal{V}, \quad \bigcup_{l=1}^{\infty} \mathcal{V}_l \text{ dicht in } \mathcal{V}.$$

1.3.2 Die Finite-Elemente-Methode

Die erste Frage, die in diesem Abschnitt beantwortet werden soll, lautet: Wie kann eine Basis gewählt werden, so daß die Matrix **A** aus (1.20) schwach besetzt ist. Dazu werden zunächst die Matrixelemente betrachtet:

$$A_{ij} = \mathrm{a}(b_i, b_j) = \int_{\mathcal{G}} \nabla b_i \cdot \nabla b_j \, d\mathcal{G}.$$

Die Matrix **A** ist schwach besetzt, wenn möglichst viele der Integrale verschwinden. Die Idee der FEM besteht darin, Basisfunktionen b_i mit möglichst kleinem Träger $Tr(b_i)$ zu wählen. Es gilt nämlich:

$$\int_{\mathcal{G}} \nabla b_i \cdot \nabla b_j \, d\mathcal{G} = \int_{Tr(b_i) \cap Tr(b_j)} \nabla b_i \cdot \nabla b_j \, d\mathcal{G}.$$

Falls $Tr(b_i) \cap Tr(b_j) = \emptyset$, dann ist $L_{ij} = 0$.

Um eine Finite-Elemente-Diskretisierung durchzuführen, sind mehrere Schritte in der folgenden Reihenfolge erforderlich:

1. Wähle eine Unterteilung des Gebiets $\mathcal{G}$ in Teilgebiete, deren Inneres disjunkt ist.
2. Definiere Basisfunktionen auf den Teilgebieten, deren Träger nur eine kleine Anzahl Teilgebiete umfaßt und die dem gegebenen Problem möglichst gut angepaßt sind.
3. Stelle das lineare Gleichungssystem (1.20) auf und löse es.

Die Teilgebiete im ersten Schritt heißen auch *finite Elemente* und geben damit der Methode den Namen. Die Unterteilung des Gebietes in Teilgebiete heißt auch *Triangulierung.*

Definition 1.7 (Triangulierung) *$T = \{\tau_1, \ldots, \tau_M\}$ heißt zulässige Triangulierung des (durch einen Polygonzug begrenzten) Gebietes $\mathcal{G}$ falls gilt:*

1. *Die finiten Elemente τ_i $(i = 1, \ldots, M)$ sind offene, disjunkte polygonale Teilmengen von $\mathcal{G}$, die $\mathcal{G}$ überdecken:*
$$\tau_i \cap \tau_j = \emptyset \text{ für } i \neq j, \quad \bigcup_{i=1}^{M} \overline{\tau_i} = \overline{\mathcal{G}}.$$
2. *Für $i \neq j$ ist $\overline{\tau_i} \cap \overline{\tau_j}$ entweder leer, oder eine gemeinsame Seite oder eine gemeinsame Ecke.*

Bemerkung 1.8 *Die Einschränkung auf Polygongebiete erfolgt hier nur zur Vereinfachung. Es können auch krummlinig berandete Gebiete $\mathcal{G}$ zugelassen werden. Dann müssen jedoch auch krummlinig berandete Elemente zugelassen werden (siehe [117]).*

Zur Einführung der FEM-Basisfunktionen werden zunächst die (einfachen) linearen Elemente betrachtet. Sei also eine Triangulierung T von $\mathcal{G}$ gegeben. Auf den Elementen τ_i werden lineare Funktionen definiert, die in einem der *Knoten* (d.h. der Ecken) des Elementes den Wert 1 annehmen und in den anderen Knoten Null sind.

Definition 1.9 (Lineare FEM-Basisfunktion) *Die Basisfunktion $b_i(x, y)$ zum Knoten n_i mit den Koordinaten $\mathbf{x}_i = (x_i, y_i)$ ist auf dem Element τ_j mit den Knotennummern n_{j_1}, n_{j_2}, n_i durch*

$$b_i(x, y) = \frac{(x - x_{j_1})(y_{j_2} - y_{j_1}) - (x_{j_2} - x_{j_1})(y - y_{j_1})}{(x_i - x_{j_1})(y_{j_2} - y_{j_1}) - (x_{j_2} - x_{j_1})(y_i - y_{j_1})} \tag{1.22}$$

gegeben.

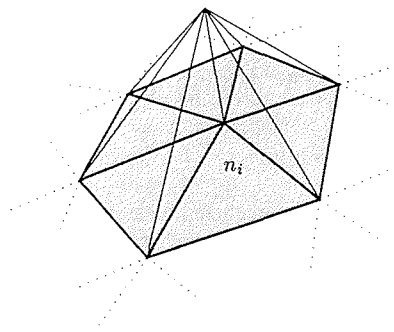

Abbildung 1.1: Lineare FEM-Basisfunktion b_i zum Knoten n_i, der Träger $Tr(b_i)$ ist schraffiert

Bemerkung 1.10 *Aus (1.22) folgt für die Knoten n_j mit Koordinaten $\mathbf{x}_j$:*

$$b_i(\mathbf{x}_j) = \begin{cases} 1, & \textit{falls} \quad i = j, \\ 0, & \forall j \neq i. \end{cases}$$

Definition 1.11 (Raum $\mathcal{V}_h^{lin}$) *Der Raum der stetigen, stückweise linearen Funktionen, $\mathcal{V}_h^{lin}$, wird definiert:*

$$\mathcal{V}_h^{lin} := \{v : v \textit{ ist linear auf } \tau_i \in T, \forall i = 1, \ldots, M, \; v \textit{ ist stetig auf } \mathcal{G}\}. \tag{1.23}$$

Für $\mathcal{G} \in \mathcal{R}^2$ haben die linearen FEM-Basisfunktionen die Form der sogenannten „Hütchenfunktionen" (siehe Abb. 1.1). Die den Knoten n_i enthaltenden Elemente bilden den Träger der Basisfunktion b_i.

Mit den in (1.22) definierten Basisfunktionen kann jedes $v \in \mathcal{V}_h^{lin}$ eindeutig dargestellt werden:

$$v(x, y) = \sum_{i=1}^{N} v_i \cdot b_i(x, y).$$

Dabei ist N die Anzahl der Knoten der Triangulierung T, b_i sind die Basisfunktionen und v_i sind die Funktionswerte $v(x_i, y_i)$ an den Knotenpunkten. v ist in $\mathcal{V}_h^{lin}$ eindeutig durch die Funktionswerte an den Knotenpunkten definiert, weil die Basisfunktionen gerade so gewählt sind, daß sie an den Knoten den Wert 1 annehmen.

Mit den so gewählten Elementen und Basisfunktionen läßt sich das lineare Gleichungssystem $\mathbf{Au} = \mathbf{f}$ aufstellen. Die Integrale, welche die Matrixeinträge A_{ij} bilden, verschwinden, falls i und j in verschiedenen Elementen liegen. Die Berechnung der Integrale ist wegen der linearen Basisfunktionen reduziert auf die Berechnung des Volumens des Elementes, da die Gradienten der Basisfunktionen konstant sind. Auch die Integrale auf der rechten Seite sind mit Hilfe einer Quadraturformel leicht zu berechnen. Der Lösungsvektor $\mathbf{u}$ besteht aus den Funktionswerten der gesuchten Funktion an den Knotenpunkten.

Es gibt eine Vielzahl weiterer Elemente und Basisfunktionen. Laufend werden neue finite Elemente vorgestellt, die speziellen Anforderungen genügen. An dieser Stelle sollen lediglich quadratische Elemente kurz dargestellt werden, weil sie bei der Implementierung des Fehlerschätzers im Abschnitt 10.5 Verwendung finden.

Für die Definition der Basisfunktionen auf einem Dreieckselement werden sechs Koeffizienten $c_0, \ldots, c_5$ für die eindeutige Definition der quadratischen Funktion

$$q(x,y) = c_0 + c_1 x + c_2 y + c_3 xy + c_4 x^2 + c_5 y^2 \tag{1.24}$$

benötigt. Eine Möglichkeit, diese Funktion zu bestimmen, ist die Wahl der Funktionswerte an den Knoten und den Seitenmittelpunkten.

Als lokale Basisfunktionen werden dann Kombinationen der linearen Basisfunktionen aus (1.22) definiert:

$$\begin{aligned} q_j(x,y) &:= [b_j(2b_j - 1)](x,y), \\ q_\nu(x,y) &:= 4[b_i b_j](x,y). \end{aligned} \tag{1.25}$$

Dabei ist $j = 1,2,3$ der lokale Index der Knoten, b_j die zum Knoten j gehörige lineare Basisfunktion, $\nu = (i,j) = (1,2),(2,3),(3,1)$ sind die Seitenmittelpunkte. Eine Funktion v läßt sich damit auf jedem Element darstellen als Linearkombination

$$v(x,y) = \sum_{i=1}^{3} v_i q_i(x,y) + \sum_{\nu=(1,2)}^{(3,1)} v_\nu q_\nu(x,y).$$

Diese Darstellung ist besonders attraktiv, weil die linearen Basisfunktionen b_i verwendet werden, welche einfach zu berechnen sind. Falls die quadratischen Basisfunktionen im Fehlerschätzer für eine stückweise lineare Finite-Elemente-Funktion benötigt werden, stehen die linearen Basisfunktionen in der Implementierung sowieso schon zur Verfügung.

Kapitel 2

Implementierung der Finite-Elemente-Methode

2.1 Übersicht

Eine wirklich umfassende Einführung in die Implementierungstechniken der FEM zu geben, ist an dieser Stelle nicht möglich. Allein zwei der Hauptkomponenten eines Finite-Elemente-Programmes, der Gittergenerator und der Gleichungslöser, sind weite eigenständige Forschungsfelder. In [4] wird die Implementierung eines FEM-Programmes genauer vorgestellt. Dort sind einige der gängigen Datenstrukturen für FEM-Implementierungen zu finden.

Unabhängig von der speziellen Implementierung besteht ein FEM-Programm aus Komponenten für die folgenden Aufgaben:

1. Gittergenerierung.
2. Aufstellung des linearen Gleichungssystems.
3. Lösung des linearen Gleichungssystems.
4. Postprocessing (Lösungsdarstellung, etc.).

Dabei ist vorausgesetzt, daß das Variationsproblem, insbesondere die Bilinearform $a(\cdot,\cdot)$ und die Linearform $f(\cdot)$, gegeben ist.

Die Gittergenerierung umfaßt sowohl die Erzeugung eines Anfangsgitters auf einem gegebenen Rechengebiet $\mathcal{G}$, als auch die lokalen Verfeinerungsstrategien bei adaptiven Verfahren. Dabei wird das Gitter dort lokal verfeinert, wo der geschätzte Diskretisierungsfehler groß ist. An den Gittergenerator werden verschiedene Anforderungen gestellt. Das Gitter soll keine *hängenden Knoten* enthalten, die Gitterelemente sollen nicht zu spitzwinklig werden und der Abstand der Gitterpunkte soll sich nicht zu schnell ändern. Den Datenstrukturen ist besondere Aufmerksamkeit zu widmen, weil lokale Verfeinerung sehr irreguläre Strukturen hervorbringt.

Bei der Aufstellung der Matrix des linearen Gleichungssystems geht es in erster Linie um die numerische Approximation der Integrale aus der Bilinearform. Aber auch die geschickte Bearbeitungsreihenfolge bei der Berechnung der Matrixkoeffizienten spielt eine Rolle, insbesondere dann, wenn das Programm auf Parallelrechnern implementiert wird. Wenn die Implementierung für kleine Hauptspeicherkapazitäten optimiert werden soll, kann auf die explizite Aufstellung der Matrix ganz verzichtet werden. In diesem Fall werden die Matrixkoeffizienten jeweils neu berechnet, wenn sie gebraucht werden. Falls die Matrix aufgestellt wird, ist auf die Datenstrukturen zu achten. Für ihre Speicherung können spezielle Datenformate für schwach besetzte Matrizen verwendet werden.

Große Gleichungssysteme, wie sie bei FEM-Verfahren auftreten, können in der Regel aufgrund der Speicherplatzbegrenzung und der operationellen Ordnung der Algorithmen nur mit iterativen Verfahren gelöst werden. Daher ist die Wahl eines optimalen Verfahrens von grundlegender Bedeutung für die numerische Effizienz des FEM-Programmes. Zwei Verfahren werden im folgenden vorgestellt: Das präkonditionierte CG-Verfahren für symmetrische und das Verfahren BiCGSTAB für unsymmetrische Matrizen.

Kommerzielle FEM-Programmpakete legen großen Wert auf das Postprocessing, vor allem auf die gute und ergonomische Darstellung der Lösungen. Dieser Teil soll in dieser Arbeit nicht weiter behandelt werden. Für die hier vorgestellte Implementierung steht eine rudimentäre Graphikausgabe der Ergebnisse zur Verfügung, die auf der IBM-Implementierung der GL-Graphikbibliothek basiert.

Grundlage für die Beschreibung der Implementierung ist zunächst immer die Poissongleichung mit homogenen Dirichlet-Randbedingungen (1.6), die hier zusammen mit dem zugehörigen Variationsproblem zusammengefaßt ist:

$$\begin{aligned} -\triangle u &= f \text{ in } \mathcal{G} \\ u &= 0 \text{ auf } \Gamma \end{aligned}$$

$$\begin{aligned} \text{Finde } u_h \in \mathcal{V}_h : \ a(u_h, b_j) &= f(b_j), \forall j = 1, \ldots, N \\ a(b_i, b_j) &= \int_{\mathcal{G}} \nabla b_i \cdot \nabla b_j \, d\mathcal{G} \\ f(b_j) &= \int_{\mathcal{G}} f b_j \, d\mathcal{G} \end{aligned} \tag{2.1}$$

Den hier vorgestellten Techniken liegt ein sehr einfaches Verfahren zugrunde. Die Probleme mit entsprechenden Randbedingungen aus dem Kapitel 1 werden mit linearen finiten Elementen auf einem Dreiecksgitter implementiert. Die Implementierung wird in den folgenden Teilen II und III weiterverwendet, um adaptive Semi-Lagrange-Verfahren für Strömungsprobleme zu entwickeln. Dabei wird die Parallelisierbarkeit jeweils besonders beachtet.

2.2 Gittergenerierung und Datenstrukturen

Die Erzeugung von Gittern auf (möglicherweise beliebig) gegebenen Rechengebieten ist eine nichttriviale Aufgabe, die immer noch intensiv erforscht wird. Der Anspruch dieses Abschnittes kann daher nicht sein, Gittergenerierung umfassend zu beschreiben. Vielmehr werden einige Ansätze zur Gittergenerierung aus der Literatur dargestellt (für numerische Gittergenerierung siehe z.B. [135]).

Die Triangulierung eines Rechengebietes wird häufig in zwei Schritten vollzogen:

1. Erzeugung eines Anfangsgitters;
2. Verfeinerung des Anfangsgitters.

Das im Unterabschnitt 2.2.3 beschriebene Verfahren zur Triangulierung des Rechengebietes arbeitet halbautomatisch. Die Anfangsverfeinerung muß von Hand vorgegeben werden. Der Gittergenerator verfeinert das vorgegebene Gitter dann bis zur gewünschten Auflösung.

Die Definition einer zulässigen Triangulierung (1.7) verbietet sogenannte *hängende Knoten*. Dabei handelt es sich um Knoten, die auf einer unverfeinerten Kante des Nachbarn liegen (siehe

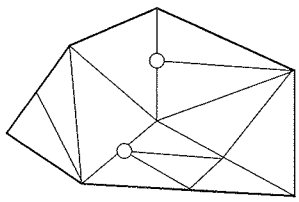

Abbildung 2.1: Hängende Knoten in einer Triangulierung

Abb. 2.1). Die Vermeidung hängender Knoten ist eine der größten Schwierigkeiten bei der Entwicklung von Algorithmen zur Gittergenerierung.

In diesem Abschnitt werden einige Ansätze zur automatischen Erzeugung von Dreiecksgittern kurz vorgestellt. Der darauf folgende Unterabschnitt beschreibt die Methoden zur lokalen Verfeinerung von gegebenen Anfangsgittern, wobei der Schwerpunkt auf der regulären Verfeinerung liegt, deren Implementierung angegeben wird. Die zugehörigen Datenstrukturen werden ausführlich behandelt, weil diese Darstellung in vielen Publikationen fehlt. Zuvor werden noch Notationen angegeben, die für den Umgang mit Triangulierungen notwendig sind.

2.2.1 Notationen

Die folgenden Notationen werden vereinbart, um die Beschreibung lokal verfeinerter Gitter zu ermöglichen:

- T bezeichne eine Triangulierung, $T = \{\tau_1, \ldots, \tau_M\}$, M ist die Anzahl der Elemente.
- Seien $T_1 = \{\tau_{1,1}, \ldots, \tau_{M_1,1}\}$ und $T_2 = \{\tau_{1,2}, \ldots, \tau_{M_2,2}\}$ Triangulierungen. Wenn T_2 durch Verfeinerung aus T_1 hervorgeht, so schreibe: $T_1 \prec T_2$
- Als Parameter h für die Gittergröße kann eine der folgenden Zahlen gewählt werden:

$$\begin{aligned} \lambda &= \min_{\tau \in T}\{\lambda_\tau \text{ Länge der längsten Seite in } \tau\} \\ h_B &= \min_{\tau \in T}\{h_{B,\tau} \text{ Außenkreisdurchmesser von } \tau\} \\ h_b &= \min_{\tau \in T}\{h_{b,\tau} \text{ Innenkreisdurchmesser von } \tau\} \end{aligned}$$

- Der kleinste innere Winkel der Triangulierung T ist gegeben durch:

$$\vartheta = \min_{\tau \in T}\{\vartheta_\tau \text{ kleinster Winkel in } \tau\}$$

- Die Triangulierung heißt *regulär*, falls $0 \ll \vartheta \ll \pi$. Diese Forderung ist äquivalent zu: $0 \ll \frac{h_b}{h_B} < 1$.
- Die Fläche eines Elements $\tau \in T$ wird durch $F_\tau^\triangle$ bezeichnet

Bemerkung 2.1 *Die Triangulierung T_2 gehe aus T_1 durch Verfeinerung hervor, d.h. $T_1 \prec T_2$. Dann gilt für die zugehörigen Elementzahlen M_1 und M_2: $M_1 \leq M_2$. Andererseits gilt für die Gitterparameter h_1 und h_2: $h_2 \leq h_1$.*

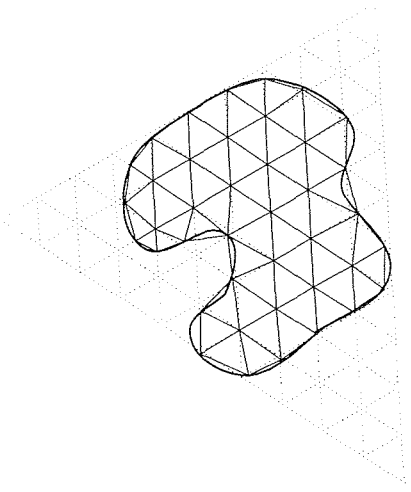

Abbildung 2.2: Triangulierung eines Gebietes mit Hilfe eines Mustergitters aus Dreiecken

2.2.2 Ansätze zur Gittergenerierung

Die Aufgabe eines Gittergenerators besteht darin, ein gegebenes Rechengebiet $\mathcal{G} \in \mathcal{R}^d$, $d = 1, 2, 3$, mit einem Gitternetz zu überdecken. Dabei werden in der Regel noch einige Anforderungen an die Qualität des Gitters gestellt. Beispielsweise wird die Form der Gitterelemente (rechteckig, dreieckig, etc.) verlangt, die Elemente sollen meist nicht zu schmal oder zu spitz sein, die Größe der Elemente soll entweder uniform oder dem zugrundeliegenden physikalischen Problem angepaßt sein, oder die Ränder des Rechengebietes sollen so genau wie möglich nachgebildet werden. Im folgenden werden nur noch Dreiecksgitter betrachtet.

Neben der manuellen Angabe der Gitterdaten – beispielsweise mit Hilfe von CAD basierten Werkzeugen – gibt es gängige Methoden zu automatischen Triangulierung von Polygongebieten. In [60] wird eine Klassifizierung der verschiedenen Methoden zur automatischen Gittergenerierung vorgenommen. Danach werden vier Hauptgruppen unterschieden:

- Methoden, die erst die Gittertopologie definieren.
- Methoden, die erst die Knoten definieren.
- Methoden, die Mustergitter verwenden.
- Methoden, die simultan Topologie und Knoten definieren.

Innerhalb der Hauptgruppen gibt es zum Teil mehrere Untergruppen. Eine häufig verwendete Methode zur Triangulierung eines Rechengebietes ist in der Hauptgruppe der Methoden, die zuerst Knoten definieren, enthalten: die *Delaunay-Triangulierung*. Dabei werden die Knoten so verbunden, daß die Summe der inneren Winkel der entstehenden Elemente maximiert wird [30, 77].

Methoden, die Mustergitter verwenden, lassen sich in Methoden untergliedern, die ein Referenzgitter auf die gegebene Geometrie abbilden, oder die Geometrie mit einem Gitter überdecken, das am Rand modifiziert wird (siehe Abbildung 2.2). Die letzte Hauptgruppe umfaßt Methoden,

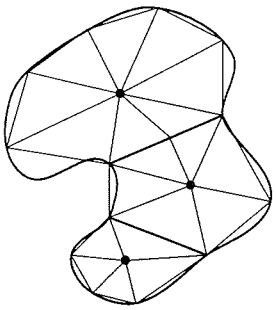

Abbildung 2.3: Triangulierung eines Gebietes mit Hilfe von Strahlen zum Rand

die das gegebene Gebiet schrittweise unterteilen. Zwei dieser Methoden für Gebiete $\mathcal{G} \in \mathcal{R}^2$ werden prinzipiell erklärt.

Die Triangulierung konvexer Gebiete läßt sich mit Hilfe von Strahlen aus dem Schwerpunkt triangulieren. Dabei wird der Schwerpunkt des Gebietes als erster Knoten der Triangulierung gewählt (im Prinzip läßt sich jeder innere Punkt wählen, der Schwerpunkt ist aber eine gute Näherung, wenn die Triangulierung auch noch Regularitätsbedingungen erfüllen soll). Vom Schwerpunkt ausgehend werden Strahlen in alle Richtungen gerichtet. Die Schnittpunkte dieser Strahlen mit dem Rand ergeben dann die weiteren Knoten der Triangulierung. Damit die Triangulierung regulär ist, dürfen die Winkel zwischen den Strahen am Schwerpunkt nicht zu klein oder zu groß sein (siehe [146]).

Ein konkaves Gebiet läßt sich mit dieser Methode triangulieren, indem es in mehrere Konvexgebiete zerlegt wird, in denen dann jeweils die oben beschriebene Methode angewendet wird (s. Abb. 2.3). Zusammengefaßt lautet die Methode:

Methode 2.1 (Triangulierung mit Strahlen) *Gegeben sei ein Gebiet $\mathcal{G} \in \mathcal{R}^2$, das durch eine Triangulierung mit polygonalem Rand approximiert wird. Dann führe folgende Schritte aus:*

1. *Falls das Gebiet konkav ist, unterteile es in konvexe Teilgebiete.*
2. *In jedem konvexen (Teil-) Gebiet wähle den Schwerpunkt als ersten Knotenpunkt.*
3. *Werfe Strahlen vom ersten Knoten kreisförmig und wähle die Schnittpunkte der Strahlen mit dem Rand von $\mathcal{G}$ als weitere Knoten.*
4. *Bei zusammengesetzten Gebieten müssen die Knoten zweier benachbarter Gebiete übereinstimmen.*

Eine zweite grundsätzliche Vorgehensweise trianguliert das gegebene Gebiet vom Rand ausgehend in Schichten [84, 60]. Dabei werden zunächst Randknoten in gleichmäßigen Abständen verteilt. Von diesen Randknoten ausgehend werden Dreiecke definiert, die jeweils einen zusätzlichen neuen Knoten im Innern des Gebietes benötigen. Wenn die inneren Knoten jeweils wieder so verbunden werden, daß zwei innere Knoten und ein Randknoten ein Element bilden, erhält man eine Schicht des Gebietes, die trianguliert ist. Es bleibt dann ein inneres noch nicht triangulierten Teilgebiet übrig, mit dem dann genauso verfahren werden kann wie mit der ersten Schicht (siehe Abb. 2.4).

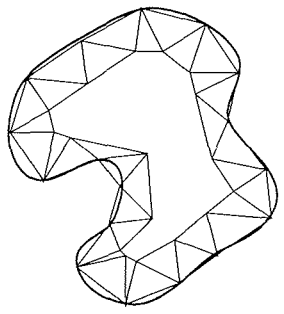

Abbildung 2.4: Erste Schicht der schichtweisen Triangulierung eines Gebietes

Methode 2.2 (Schichtweise Triangulierung) *Es sei wieder ein Gebiet $\mathcal{G} \in \mathcal{R}^2$ gegeben. Zunächst werden auf dem Rand Randknoten verteilt. Solange noch ein inneres Teilgebiet nicht trianguliert ist, durchlaufe folgende Schritte:*

1. *Definiere zu jeweils zwei Randknoten einen inneren Knoten, so daß ein Element entsteht, das den gegebenen Regularitätsbedingungen genügt.*
2. *Verbinde jeweils benachbarte innere Knoten, so daß ein neuer Rand zum noch nicht triangulierten Inneren des Gebietes entsteht.*
3. *Bilde die nächste Schicht mit den gleichen Schritten.*

Das Verfahren endet, wenn ein kleines konvexes inneres Gebiet übrig ist (möglicherweise auch mehrere konvexe Gebiete). In diesem Gebiet wird dann ein neuer Knoten definiert und mit allen Knoten der letzten Schicht verbunden.

2.2.3 Gitterverfeinerung

Ausgehend von einem gegebenen globalen Gitter aus Dreieckselementen soll nun lokal verfeinert werden. Dazu stehen verschiedene Strategien zur Verfügung, die im folgenden kurz vorgestellt werden.

Zwei Typen der Dreiecksverfeinerung lassen sich unterscheiden (siehe Abb. 2.5):

1. Reguläre Teilung in vier Tochterdreiecke (rote Verfeinerung).
2. Irreguläre Teilung in zwei Tochterdreiecke (grüne Verfeinerung).

An eine Verfeinerungsstrategie werden weitere Anforderungen gestellt. Sie muß (möglicherweise nach einigen Iterationen) zu einer zulässigen Triangulierung führen. Die Dreiecke sollen aus numerischen Gründen nicht zu spitz oder schmal werden, d.h. die inneren Winkel sollen nicht zu klein werden. Die Dreiecksgröße soll sich in lokal verfeinerten Gittern nicht zu schnell ändern. Diese Annahmen lassen sich zusammenfassen:

Forderung 2.2 (Eigenschaften des lokal verfeinerten Gitters)
Eine Verfeinerungsstrategie soll eine Triangulierung T mit den folgenden Eigenschaften erzeugen:

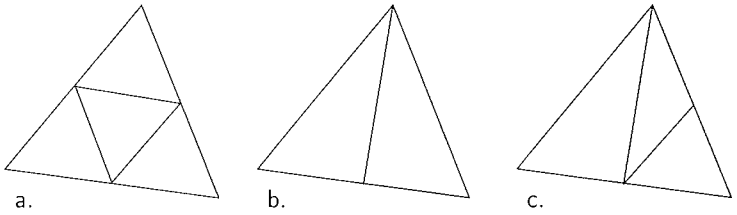

Abbildung 2.5: Verfeinerungsstrategien: a. rote Verfeinerung in vier Tochterdreiecke, b. grüne Verfeinerung, c. blaue Verfeinerung

1. **Zulässigkeit:** *Die Triangulierung T soll* zulässig *sein, d.h. keine hängenden Knoten enthalten.*
2. **Regularität:** *Die Triangulierung T soll regulär sein.*
3. **Quasi-Uniformität:** *Für zwei Elemente $\tau_1, \tau_2 \in T$ mit einem gemeinsamen Knoten oder einer gemeinsamen Kante soll gelten: $F^{\triangle}_{\tau_1} = \rho F^{\triangle}_{\tau_2}$, wobei $0 \ll \rho \leq 1$.*

Die erste Methode zur Verfeinerung einer gegebenen Triangulierung verwendet nur grüne Verfeinerungen. Sie führt zu einem irregulär verfeinerten Gitter.

Methode 2.3 (Bisektion der längsten Seite)
Die Verfeinerung durch Teilung der längsten Seite *wird durch die folgenden zwei Schritte beschrieben:*

1. *Dreiecke werden geteilt, indem der Mittelpunkt der längsten Seite mit dem gegenüberliegenden Knotenpunkt durch eine neue Kante verbunden wird.*
2. *Falls die Triangulierung hängende Knoten enthält, dann teile diejenigen Dreiecke mit hängenden Knoten erneut durch Teilung der längsten Seite. Dieser Schritt muß gegebenenfalls mehrmals durchgeführt werden.*

Es kann gezeigt werden, daß die mit Methode 2.3 erzeugte Triangulierung zulässig, quasi-uniform und regulär ist [86]. Der kleinste innere Winkel aller Verfeinerungen ist mindestens halb so groß wie der der Ausgangstriangulierung.

Der Schritt 2. aus Methode 2.3 kann durch folgende Alternativen ergänzt werden:

2a. Dreiecke mit zwei hängenden Knoten werden rot verfeinert.

2b. Dreiecke mit einem hängenden Knoten, der nicht auf der längsten Seite liegt, werden blau verfeinert (d.h. zuerst wird die längste Seite geteilt und dann die Seite mit dem hängenden Knoten).

Eine Variante der Methode 2.3, bei der die teilbare Seite der Dreiecke nicht natürlich durch die Länge ausgezeichnet ist, sondern künstlich markiert wird, lautet:

Methode 2.4 (Bisektion einer markierten Seite)
Verfeinerung durch Teilung einer markierten Seite *läßt sich durch folgende Schritte charakterisieren (siehe Abb. 2.6):*

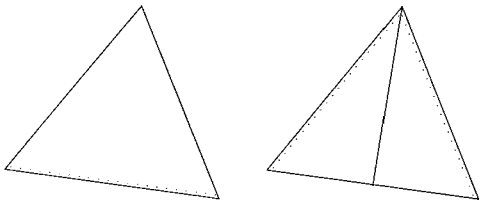

Abbildung 2.6: Bisektion der markierten Seite: Nach der Teilung werden die beiden unmarkierten Seiten markiert

1. *Jedes Dreieck besitzt genau eine markierte Seite.*
2. *Ein Dreieck wird an der markierten Seite grün geteilt. Nach der Teilung werden die zuvor unmarkierten Seiten der beiden Tochterdreiecke markiert.*
3. *Ein Dreieck darf nur geteilt werden, wenn seine markierte Seite auch markierte Seite des zugehörigen Nachbardreiecks ist.*

Bänsch zeigt, daß Methode 2.4 eine zulässige und reguläre Triangulierung erzeugt [12]. Wie in Methode 2.3 sind die inneren Winkel nach unten beschränkt.

Die Teilung an der markierten Seite läßt sich relativ leicht implementieren. Die erzeugten Triangulierungen sind quasi-uniform. Es handelt sich um eine schlanke und elegante Methode.

In der folgenden Methode ist die Standardteilung der Elemente regulär (rot). Zur Vermeidung von hängenden Knoten müssen aber auch hier grüne Verfeinerungen eingefügt werden.

Methode 2.5 (Reguläre Verfeinerung)
Folgende Schritte beschreiben die reguläre Verfeinerung *einer Triangulierung:*

1. *Dreiecke werden rot verfeinert.*
2. *Dreiecke mit zwei hängenden Knoten werden rot verfeinert.*
3. *Dreiecke mit einem hängenden Knoten werden grün verfeinert (d.h. der hängende Knoten wird mit dem gegenüberliegenden Knoten verbunden).*
4. *Dreiecke mit einem hängenden Knoten an einer Kante, die zuvor grün geteilt wurde, müssen rot verfeinert werden (siehe Abbildung 2.7).*

Zum Schritt 4. in Methode 2.5 gibt es zwei Alternativen:

4a. *Dreiecke mit einem hängenden Knoten an einer Kante, die zuvor grün geteilt wurde, werden blau verfeinert.*

4b. *Dreiecke mit einem hängenden Knoten an einer Kante, die zuvor grün geteilt wurde, werden zunächst wieder vergröbert, dann rot verfeinert. Schließlich wird das Dreieck mit einem hängenden Knoten grün verfeinert (siehe Abb. 2.8).*

Man kann zeigen, daß mit Methode 2.5 eine Triangulierung erzeugt wird, die zulässig, regulär und quasi-uniform ist [10].

Ein Vorteil der regulären Verfeinerung ist, daß die Töchter ihrem Mutterdreieck ähnlich sind (d.h. sie besitzen dieselben inneren Winkel). Ein weiterer Vorteil ist durch die bessere Vergröberungsmöglichkeit gegeben. Für mit Bisektion verfeinerte Gitter gibt es Situationen, in denen die Verfeinerung keine „inverse" Operation besitzt. Regulär verfeinerte Gitter können jedoch so vergröbert werden, daß der Anfangzustand wieder hergestellt wird.

Andererseits müssen durch die grünen Verfeinerungen für hängende Knoten viele Fälle berücksichtigt werden. Die Implementierung wird dadurch aufwendig und unübersichtlich. Für die Implementierung der FEM wird wegen der Parallelisierungsstrategie mit der im Kapitel 3 beschriebenen Farbindizierung trotzdem die reguläre Verfeinerung gewählt.

Das FEM-Programm implementiert die Gitterverfeinerung neu. Der Schritt 4. in Methode 2.5 wird durch die Alternative 4b. ersetzt. Das wesentliche neue Element der Implementierung ist die Erstellung des Gitters in drei Schritten. Im ersten Schritt wird ein „gesäubertes" Ausgangsgitter erzeugt. Grüne Verfeinerungen werden aufgehoben und rote Verfeinerungen dort eingefügt, wo die Situation aus 4b. auftritt. Im zweiten Schritt werden die neuen Verfeinerungen zunächst nur virtuell (über Flaggen, Datenstrukturen, die den Zustand des Dreiecks beschreiben) durchgeführt. Die virtuelle Verfeinerung wird auf ihre Zulässigkeit geprüft und gegebenenfalls (virtuell) modifiziert. Erst wenn eine zulässige virtuelle Verfeinerung fertiggestellt ist, werden die Verfeinerungen im dritten Schritt auch real ausgeführt.

Diese Vorgehensweise bietet zwei Vorteile. Zunächst dient sie der Übersichtlichkeit des Programmcodes. Die vielen Fallunterscheidungen, vor allem in bezug auf die Informationen über Nachbarelemente, können entfallen. Erst wenn am Ende klar ist, ob ein Element rot oder grün verfeinert wird, werden auch die Nachbarn über die Änderungen informiert.

Der zweite Vorteil, der hier indes noch nicht genutzt wird, liegt in der potentiell besseren Parallelisierbarkeit. Alle virtuellen Änderungen am Gitter können parallel durchgeführt werden. Die Überprüfung der Zulässigkeit muß natürlich synchronisiert werden, aber die Modifikationen sind immer lokal unabhängig (und damit parallelisierbar). Der Schritt der realen Gitterverfeinerung bleibt als serieller Teil erhalten, ist aber wegen der nicht mehr notwendigen Zulässigkeitsprüfung nicht aufwendig.

Die Implementierung für die Verfeinerung wird im folgenden Algorithmus zusammengefaßt:

Algorithmus 2.1 (lokale Verfeinerung)
Sei $T_0 = \{\tau_i\}_{i=1,\ldots,M_0}$ eine gegebene Triangulierung, in der einige τ_j zur Verfeinerung markiert sind. Dann erhalte eine verfeinerte Triangulierung T_1 durch:

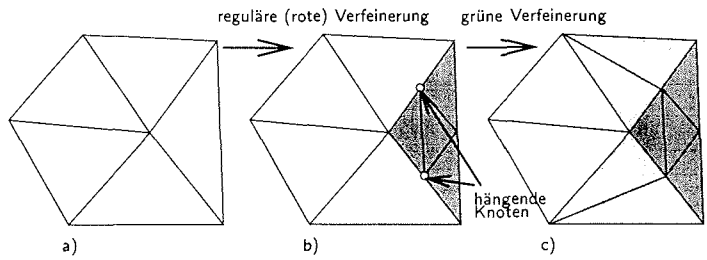

Abbildung 2.7: Reguäre Verfeinerungsstrategie: a. rote Verfeinerung eines Elements, b. hängende Knoten, c. grüne Verfeinerung

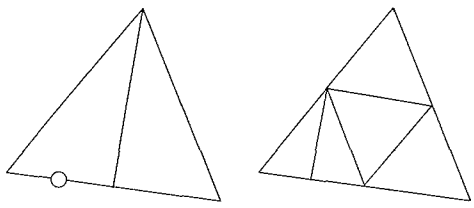

Abbildung 2.8: Ein hängender Knoten (○) wird verhindert durch rote Verfeinerung und anschließende grüne Teilung

Tabelle 2.1: Elementdaten für die Gittergenerierung

Name	Größe	Typ	Funktion
`nt_node`	3×`ntmax`	int	Globale Knotennummern
`nt_neig`	3×`ntmax`	int	Nachbar-Indizes
`nt_colr`	`ntmax`	int	Farbe (f. Parallelisierung)
`nt_type`	`ntmax`	int	Elementtyp (f. Parallelisierung)
`nt_chil`	`ntmax`	int	Erstes Tochterelement
`nt_flag`	`ntmax`	int	Verfeinerungszustand
`nt_levl`	`ntmax`	int	Verfeinerungsstufe
`nt_prnt`	`ntmax`	int	Mutterelement

1. *Bilde eine Liste L_1 aller $\tau_i \in T_0$ mit:* **a.** *τ_i ist grün geteilt,* **b.** *τ_i ist grün geteilt und zur Verfeinerung markiert,* **c.** *für τ_i tritt die Situation aus 4b. in Methode 2.5 ein.*
2. *Für alle $\tau_i \in L_1$: hebe die grüne Teilung auf und verfeinere ggfs. rot. Erhalte eine Ausgangstriangulierung ohne grüne Teilung (möglicherweise mit hängenden Knoten).*
3. *Bilde eine Liste L_2 aller $\tau_i \in T_0$ mit:* **a.** *τ_i ist zur Verfeinerung markiert,* **b.** *τ_i muß zur Vermeidung hängender Knoten grün verfeinert werden.*
4. *Für alle $\tau_i \in L_2$: Prüfe, ob τ_i verfeinert werden kann, modifiziere L_2 zu L_2'.*
5. *Für alle $\tau_i \in L_2'$: Verfeinere virtuell.*
6. *Prüfe das virtuelle Gitter auf Zulässigkeit. Falls das Gitter nicht zulässig ist, gehe zu 1.*
7. *Bilde eine Liste L_r aller $\tau_i \in T_0$ mit: τ_i ist virtuell rot verfeinert.*
8. *Bilde eine Liste L_g aller $\tau_i \in T_0$ mit: τ_i ist virtuell grün verfeinert.*
9. *Für alle $\tau_i \in L_r$: Verfeinere τ_i real rot.*
10. *Für alle $\tau_i \in L_g$: Verfeinere τ_i real grün.*
11. *Modifiziere globale Gitterinformationen und „säubere" die Datenstrukturen.*

Der Vergröberungsalgorithmus ist etwas einfacher. Insbesondere die Erstellung eines Ausgangsgitters ohne grüne Teilungen entfällt hier, weil grüne Teilungen einfach aufgehoben werden, falls sie entweder zur Vergröberung markiert sind oder falls sie überflüssig werden, weil sie keinen hängenden Knoten mehr eliminieren.

Algorithmus 2.2 (lokale Vergröberung)
Sei $T_1 = \{\tau_i\}_{i=1,\ldots,M_1}$ eine gegebene Triangulierung, in der einige τ_j zur Vergröberung markiert sind. Dann erhalte eine vergröberte Triangulierung T_0 durch:

Tabelle 2.2: Knotendaten für die Gittergenerierung

Name	Größe	Typ	Funktion
nn_boun	nnmax	int	Randbedingung
xy	2×nnmax	real	Knotenkoordinaten

1. *Für alle Verfeinerungsstufen* $l = L_{max}, \ldots, 1$*:*
2. *Bilde eine Liste* L_c *aller* $\tau_i \in T_l$ *mit:* τ_i *ist Mutterelement das vergröbert werden soll (d.h.* τ_i *enthält mindestens drei Töchter, die aufgelöst werden sollen).*
3. *Für alle* $\tau_i \in L_c$*: vergröbere virtuell.*
4. *Prüfe Zulässigkeit auf der Gitterstufe* l*. Bilde eine neue Liste* L_c *der nicht zulässigen* τ_i*. Falls* $L_c \neq \emptyset$*, gehe zu 3.*
5. *Ende der Schleife über* l*.*
6. *Bilde eine Liste* L_r *aller* $\tau_i \in T_l$ *mit:* τ_i *ist virtuell vergröbertes rotes Element.*
7. *Für alle* $\tau_i \in L_r$*: vergröbere real.*
8. *Bilde eine Liste* L_g *aller* $\tau_i \in T_l$ *mit:* τ_i *ist virtuell vergröbertes grünes Element.*
9. *Für alle* $\tau_i \in L_g$*: vergröbere real.*
10. *Modifiziere globale Gitterinformationen und „säubere" die Datenstrukturen.*

Die Schritte „virtuelle Verfeinerung/Vergröberung" und „Prüfung der Zulässigkeit" bestehen aus einer großen Zahl von Fallunterscheidungen. Die virtuelle Verfeinerung muß beispielsweise berücksichtigen, in welchem Zustand die Nachbarelemente sind: Gibt es hängende Knoten und an welchen Seiten? Ist die Seite schon grün geteilt, möglicherweise auch im Nachbarn? Die Zulässigkeit der Triangulierung muß ebenfalls eine Vielzahl von Kombinationen berücksichtigen. Diese Fälle im einzelnen aufzuführen, soll hier vermieden werden.

Festzuhalten bleibt lediglich, daß das implementierte Verfahren stabil ist und sowohl Verfeinerung als auch Vergröberung in adaptiven zeitabhängigen Berechnungen sicher beherrscht.

2.2.4 Datenstrukturen

Datenstrukturen für die Gittergenerierung und adaptive Mechanismen der Gitteranpassung sind in der Literatur nicht sehr oft zu finden. Die im letzten Abschnitt vorgestellte Implementierung eines Gitterverfeinerungsalgorithmus verwendet einige spezielle Datenstrukturen.

Grundsätzlich ist bei der Implementierung von Gitteralgorithmen zwischen *Elementdaten* und *Gitterpunktdaten* zu unterscheiden. Elementdaten werden für jedes Element gehalten. Dazu gehören Informationen über die Knoten des Elementes, über die Nachbarn des Elements und den Zustand (rot/grün verfeinert, unverfeinert, etc.). Gitterpunktdaten werden den Knotenpunkten einer Triangulierung zugeordnet. Dazu gehören beispielsweise die Koordinaten des Knotens.

Die Datenstrukturen der Implementierung sind in den Tabellen 2.1, 2.2 und 2.3 aufgelistet. Dabei bezeichnen `ntmax`, `nnmax` und `nrmax` die Anzahl der Elemente, die Anzahl der Knoten bzw. die Anzahl der Verfeinerungsstufen.

Die Felder `nt_node` und `nt_neig` speichern zu jedem Dreieck die globalen Knotennummern der drei lokalen Knoten und die Nachbarn an den drei Seiten. Dazu sind die lokalen Knoten entgegen

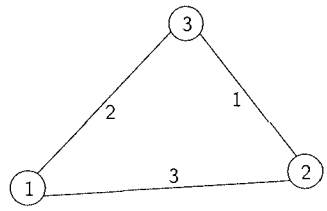

Abbildung 2.9: Nummerierung der lokalen Knoten und Seiten eines Dreiecks

dem Uhrzeigersinn numeriert, die Seiten tragen die Nummern der gegenüberliegenden Knoten (siehe Abb. 2.9).

`nt_node` kann damit die Elementnummern auf die globalen Knotenindizes „abbilden". Diese Datenstruktur ist bei der Berechnung der Steifigkeitsmatrix wichtig, wie im folgenden Abschnitt beschrieben wird. Die Nachbarelemente müssen vor allem zur Überprüfung der Zulässigkeit der Triangulierung bekannt sein.

Informationen über den Zustand und den geplanten Zustand eines Elementes werden im Feld `nt_flag` gespeichert. Durch geschickte Kodierung des Eintrages kann hier viel Information in nur einer natürlichen Zahl ausgedrückt werden. Einige Beispiele für diese Kodierung sind in der Tabelle 2.4 angegeben.

Die beiden Felder `nt_chil` und `nt_prnt` sind für die adaptive Gitterverfeinerung notwendig. Zur Vergröberung eines Dreiecks muß die Information über das Mutterdreieck vorhanden sein. Die Entscheidung über Vergröberung setzt Information über die Tochterelemente voraus. Das Vergröberungskriterium (siehe Algorithmus 2.2) verlangt, daß mindestens drei der vier Töchter eines Dreiecks aufgehoben werden müssen. Auch für die Überprüfung der Zulässigkeit ist die Information über Tochterelemente notwendig.

Bei den Knotendaten kommt dem Feld `nn_boun` besondere Bedeutung zu. Es enthält für jeden Knoten Information über die Randbedingungen (bzw. darüber ob der Knoten im Innern des Gebietes liegt). Für periodische Ränder ist in diesem Feld der jeweils identische Knotenindex abgelegt. Im einzelnen sind die Informationen in der in Tabelle 2.5 angegebenen Weise kodiert.

Die Felder mit globalen Informationen halten vor allem die Baumstruktur des verfeinerten Gitters. Dabei enthalten die Felder `nn_tree` und `nt_tree` zu jeder Gitterstufe jeweils den Index des ersten Knotens bzw. Elementes der Stufe.

Tabelle 2.3: Globale Daten für die Gittergenerierung

Name	Größe	Typ	Funktion
`nn_tree`	`nrmax`	int	Anzahl der Knoten für jede Stufe
`nt_tree`	`nrmax`	int	Anzahl der Elemente für jede Stufe
`i_fine`	`ntmax`	int	Indizes der Elemente der feinsten Stufe
`nn_cur`	1	int	Gesamtzahl der Knoten
`nt_cur`	1	int	Gesamtzahl der Elemente
`nr_cur`	1	int	Gesamtzahl der Stufen

Tabelle 2.4: Zustandsinformationen für Dreiecke

0	nicht verfeinert	1	bitte verfeinern	-1	bitte vergröbern
21	Seite 1 grün verfeinert	22	Seite 2 grün verfeinert	23	Seite 3 grün verfeinert
10	rot verfeinert	9	virtuell rot verfeinert	-9	virtuell rot vergröbert
3	bitte Seite 1 grün verf.	4	bitte Seite 2 grün verf.	5	bitte Seite 3 grün verf.
-3	bitte Seite 1 vergröbern	-4	bitte Seite 2 vergröbern	-5	bitte Seite 3 vergröbern

Tabelle 2.5: Kodierung der Randbedingungen für jeden Knoten

0	innerer Knoten
-1	Dirichlet-Randbedingung
-2	Neumann-Randbedingung
i	Index für identischen Punkt bei periodischem Rand

Wegen der adaptiven Gittergenerierung sind die Elementindizes der feinsten Triangulierung nicht notwendigerweise konsekutiv geordnet. Daher enthält das Feld `i_fine` die Nummern aller Elemente des feinsten Gitters. Diese Indexliste wird für Operationen benötigt, die nur auf den Elementen des feinsten Gitters ausgeführt werden (beispielsweise die Berechnung der Elementsteifigkeitsmatrizen).

Mit den vorgestellten Datenstrukturen läßt sich ein adaptiver Gittergenerator implementieren. In den weiteren Abschnitten werden noch einige Felder zur Parallelisierung oder für andere Aufgaben hinzukommen. Die entsprechenden Datenstrukturen werden an der jeweiligen Stelle eingeführt. Datenstrukturen für die sparsame Speicherung von spärlich besetzten Matrizen werden im nächsten Abschnitt behandelt.

2.3 Aufstellung der Steifigkeitsmatrix

Die Steifigkeitsmatrix wird bei adaptiven Verfahren immer wieder neu aufgestellt, weil sie sich mit der Modifikation des Gitters verändert. Daher muß die Implementierung möglichst effiziente Techniken verwenden.

Die Einträge der Steifigkeitsmatrix sind durch den folgenden Ausdruck gegeben:

$$A_{ij} = \int_{Tr(b_i) \cap Tr(b_j)} \nabla b_i \cdot \nabla b_j \, d\mathcal{G}. \tag{2.2}$$

Dieses Integral läßt sich auch schreiben als die Summe der Integrale über den Elementen $\tau \in T$ der Triangulierung, die den Träger $Tr(b_i) \cap Tr(b_j)$ bilden:

$$A_{ij} = \sum_{\tau \in Tr(b_i) \cap Tr(b_j)} \left[\int_\tau \nabla b_i \cdot \nabla b_j \, d\mathcal{G} \right]. \tag{2.3}$$

In der Regel werden die Integrale zunächst für alle $\tau \in T$ berechnet. Das ist vor allem in Hinblick auf die Parallelisierung vorteilhaft, weil diese Berechnungen unabhängig und ohne Kommunikation durchgeführt werden können. Die numerische Integration geschieht meist mit Hilfe von Quadraturformeln, dargestellt im ersten Unterabschnitt.

Dann müssen die Beiträge aus den einzelnen Elementen den jeweiligen Matrixeinträgen zugeordnet werden. Das läßt sich in unterschiedlicher Reihenfolge durchführen. Die entsprechenden Verfahren mit ihren Vor- und Nachteilen werden im zweiten Unterabschnitt behandelt.

Die spärlich besetzte Steifigkeitsmatrix soll dann möglichst platzsparend gespeichert werden. Die gängigen Verfahren und zugehörigen Datenstrukturen werden in einem weiteren Unterabschnitt vorgestellt.

2.3.1 Berechnung der Elementsteifigkeitsmatrizen

Als *Elementsteifigkeitsmatrizen* bezeichnet man die Matrizen $\mathbf{E}^\tau = \left[E^\tau_{i,j}\right]_{i,j=1,\ldots,\nu}, \tau \in T$ mit ν der Anzahl der Basisfunktionen in jedem Element, die definiert sind durch:

$$E^\tau_{i,j} = \int_\tau \nabla b_i \cdot \nabla b_j \, d\mathcal{G}. \tag{2.4}$$

Dabei seien die b_i, b_j jeweils die lokal definierten Basisfunktionen aus (1.22).

Das Integral in (2.4) läßt sich im allgemeinen nicht analytisch lösen, sondern muß numerisch berechnet werden. Das geschieht mit Hilfe einer *Quadraturformel*:

$$\int_\tau g(x)\, dx \approx \sum_{k=1}^{L} g(\xi_k) w_k, \tag{2.5}$$

wobei w_k mit $k = 1, \ldots, L$ *Gewichte* sind und ξ_k bestimmte Punkte (*Stützstellen*) im Element τ. Die Güte der Quadraturformel läßt sich mit Hilfe der Polynomordnung von g ausdrücken, für welche in (2.5) Gleichheit gilt. Die Güte hängt von der Anzahl der Stützstellen und von der Wahl der Gewichte ab.

Für quadratische Polynome ist die folgende Quadraturformel exakt:

$$\int_\tau g(x)\, dx \approx \sum_{k=1}^{3} \left[g(\xi_k) \frac{F^\triangle_\tau}{3} \right].$$

Dabei sind ξ_k die Seitenmittelpunkte. In der Implementierung wird diese Quadraturformel mit ξ_k den Knotenpunkten verwendet. Das entspricht der folgenden für lineare Polynome exakten Formel mit ξ_k dem Gewichtsmittelpunkt von τ:

$$\int_\tau g(x)\, dx \approx g(\xi_k) F^\triangle_\tau$$

Es werden alle Elementsteifigkeitsmatrizen des feinsten Gitters berechnet. Sie werden in einem reellen Array `e_mat` der Größe $3 \times 3 \times$`ntmax` gespeichert. Es läßt sich noch Speicherplatz einsparen, falls die Elementsteifigkeitsmatrizen symmetrisch sind; dann reicht es, die Größe auf $6 \times$`ntmax` festzusetzen. Diese Angaben gelten für lineare Elemente, die nur drei Knotenpunkte enthalten.

Eine Möglichkeit, die Berechnung der Elementsteifigkeitsmatrizen zu umgehen, ist die Abbildung aller Elemente auf ein Referenzelement (bei Dreieckselementen beispielsweise auf das Einheitsdreieck). Dann muß für jedes Dreieck die Determinante der zugehörigen affinen Abbildung berechnet und gespeichert werden (siehe [117]).

```
forall τ ∈ T do
(   call Berechne Elementmatrix e_mat )
    forall j=1,2,3 do
        node_j= nt_node(j,τ)
        forall i=1,2,3 do
            node_i= nt_node(i,τ)
            a_mat(node_i,node_j)= a_mat(node_i,node_j)+ e_mat(i,j,τ)
        enddo
    enddo
enddo
```

Abbildung 2.10: Pseudocode für die Matrixassemblierung in EBE-Ordnung

2.3.2 Assemblierung der Steifigkeitsmatrix

Die *Assemblierung* der Steifigkeitsmatrix besteht aus der Addition der Beiträge aus den Elementsteifigkeitsmatrizen auf die entsprechenden Einträge der (globalen) Steifigkeitsmatrix. Zusätzlich werden gegebenenfalls einige Matrixeinträge aufgrund der Randbedingungen modifiziert.

Bei der Aufstellung der Matrix sind zwei Reihenfolgen der Bearbeitung geläufig:

1. Elementweise (element-by-element, EBE)
2. Punktweise

Die Wahl der Bearbeitungsreihenfolge ist vor allem bei der Parallelisierung von Bedeutung und hängt von der Organisation des Lösungsverfahrens ab.

Die *elementweise* Bearbeitung orientiert sich an der Form in (2.3). Dabei werden in einer Schleife über die Elemente des feinsten Gitters die Matrixindizes gesucht und der Eintrag aus der entsprechenden Elementsteifigkeitsmatrix addiert. Ein großer Vorteil dieser Ordnung ist, daß die Elementsteifigkeitsmatrizen nicht explizit aufgestellt (und gespeichert!) werden müssen. Sie können innerhalb der Schleife berechnet und sofort verwendet werden. Einige parallele FEM-Implementierungen verwenden auch eine EBE-Ordnung im iterativen Lösungsverfahren. Das gesamte FEM-Programm kann dann in der gleichen Ordnung bearbeitet werden, die Werte der einzelnen den Elementen zugeordneten Größen bleiben im lokalen Speicher jedes Prozessors (siehe z.B. [85]). Ein Segment des zugehörigen Pseudocodes ist in Abb. 2.10 angegeben.

Ein weiterer Vorteil dieser Ordnung ist für Computerarchitekturen mit Speicherhierarchien gegeben: Die Elementsteifigkeitsmatrix wird jeweils nur einmal geladen, dann werden alle Operationen auf den Daten ausgeführt. Die Elementsteifigkeitsmatrix bleibt während dieser Operationen im schnellsten Speicher.

Der wesentliche Nachteil dieser Bearbeitungsreihenfolge ist der nichtkonsekutive Zugriff auf die Speicherplätze der globalen Steifigkeitsmatrix. Da die globalen Knotennummern `node_i` und `node_j` möglicherweise weit auseinander liegen, können die Schreibzugriffe auf `a_mat` nicht einfach determiniert werden. Auch für die Parallelisierung ergibt sich hier eine Schwierigkeit. Im Kapitel 3 wird für diesen Fall eine Parallelisierungsstrategie entwickelt.

Deterministisch und damit einfacher zu optimieren ist der Datenzugriff, wenn die Beaabeitungsreihenfolge in der Matrixassemblierung auf punktweise Ordnung umgestellt wird. Diese Ordnung ist aber nicht ganz einfach zu erreichen. Insbesondere ist dazu eine Umkehrung der Abbildung,

```
forall node_j=1,...,N do
    forall τ ∈ nn_ind(node_j) do
        j= lokale Nummer von node_j
        forall i=1,2,3 do
            node_i= nt_node(i,τ)
            a_mat(node_i,node_j)= a_mat(node_i,node_j)+ e_mat(i,j,τ)
        enddo
    enddo
enddo
```

Abbildung 2.11: Pseudocode für die Matrixassemblierung in punktweiser Ordnung

wie sie mit dem Feld `nt_node` erreichbar ist, notwendig. Zu gegebenem Knotenpunkt müssen also die angrenzenden Elemente ermittelt werden können. Mit den vorhandenen Datenstrukturen ist das nicht ohne weiteres möglich, daher wird ein zusätzliches Feld, `nn_ind` der Größe `nbmax`×`nnmax` (`nbmax` die maximale Anzahl Elemente, die einen Knoten enthalten), eingeführt, in dem zu jedem Knoten die angrenzenden Elementindizes gespeichert werden.

Nun läßt sich die Bearbeitungsreihenfolge ändern. Zusätzlich zeichnet sich diese Ordnung durch einfache Parallelisierbarkeit aus, weil die Schreibzugriffe jetzt konsekutiv und deterministisch vorgenommen werden. Ein Pseudocode-Segment ist in Abb. 2.11 zu finden

Zusammenfassend läßt sich festhalten: Die EBE-Ordnung bietet Vorteile bei der Speicheroptimierung des Algorithmus und benötigt weniger Speicher als die punktweise Ordnung. Andererseits ist der Schreibzugriff bei der EBE-Ordnung nicht determiniert und daher nicht einfach optimierbar.

2.3.3 Sparsame Speicherung spärlich besetzter Matrizen

Einer der Gründe für die Einführung der FEM ist der Wunsch nach sparsam besetzten Matrizen. Die FEM-Matrix für ein uniform regulär verfeinertes Gitter mit linearen Elementen hat beispielsweise eine *Bandbreite* (d.h. die Anzahl der Einträge ungleich Null pro Zeile) von 7. Bei großen Matrizen muß daher ein spezielles Verfahren zur sparsamen Speicherung dieser Matrizen angewendet werden.

Es stehen verschiedene Speicherformate zur Verfügung. Hier sollen nur die grundsätzlichen Typen für beliebige sparsam besetzte Matrizen vorgestellt werden. Es gibt eine Vielzahl von weiteren Speicherformaten für spezielle Matrixformen und eine Vielzahl vom Modifikationen zu den einzelnen Formaten (siehe z.B. [91]).

Zur Veranschaulichung der verschiedenen Formate sei die folgende FEM-Steifigkeitsmatrix $\mathbf{A}$

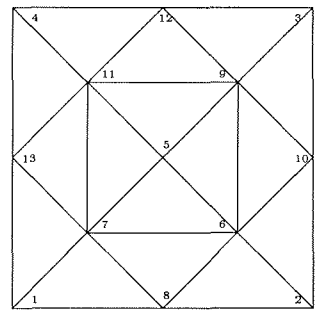

Abbildung 2.12: Triangulierung und Numerierung der Knoten

gegeben:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & & & & & & & & & & & & \\ & a_{2,2} & & & & & & & & & & & \\ & & a_{3,3} & & & & & & & & & & \\ & & & a_{4,4} & & & & & & & & & \\ & & & & a_{5,5} & a_{5,6} & a_{5,7} & & a_{5,9} & & a_{5,11} & & \\ & & & & a_{6,5} & a_{6,6} & a_{6,7} & & a_{6,9} & & & & \\ & & & & a_{7,5} & a_{7,6} & a_{7,7} & & & & a_{7,11} & & \\ & & & & & & & a_{8,8} & & & & & \\ & & & & a_{9,5} & a_{9,6} & & & a_{9,9} & & a_{9,11} & & \\ & & & & & & & & & a_{10,10} & & & \\ & & & & a_{11,5} & & a_{11,7} & & a_{11,9} & & a_{11,11} & & \\ & & & & & & & & & & & a_{12,12} & \\ & & & & & & & & & & & & a_{13,13} \end{bmatrix} \tag{2.6}$$

Sie entspricht der Triangulierung in Abb. 2.12, wobei für die Randknoten Dirichlet-Randbedingungen angenommen wurden. Die Matrix hat die Größe $N \times N$ ($N = 13$) und NZN Einträge ungleich Null ($NZN = 29$). Die Bandbreite der Matrix ist $\nu = 5$. Das entspricht bei vollständiger Speicherung 169 Einträgen, für die 676 Byte Speicherplatz (ein `real`-Wort benötigt 4 Byte) verbraucht werden.

Die einfachste Möglichkeit, die Matrix in sparsamer Weise zu speichern, besteht darin, die Einträge in einem Vektor der Länge NZN zu halten und dazu zwei Indexvektoren für die Zeilen- und Spaltenposition anzulegen.

Methode 2.6 (Koordinaten Speicherformat)
Die Einträge der Matrix werden in drei Arrays gespeichert:

1. *Werte-Array* `W` *für die Werte der Einträge.*
2. *Index-Array* `I` *für die Zeilenposition der Einträge.*
3. *Index-Array* `J` *für die Spaltenposition der Einträge.*

Speichere die Einträge in einer beliebigen Reihenfolge in `W`*, speichere an den gleichen Arraypositionen in* `I` *den Zeilenindex und in* `J` *den Spaltenindex des Eintrages (siehe Abb. 2.13).*

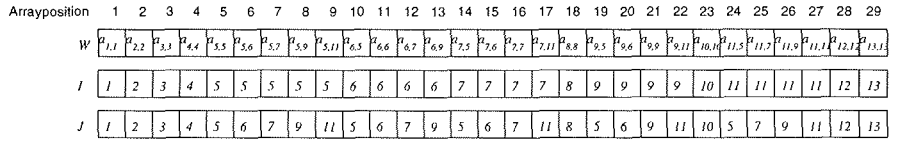

Abbildung 2.13: Koordinaten Speicherformat

Mit dieser Methode werden NZN reelle Werte und $2 \cdot NZN$ Indizes gehalten. In diesem Fall werden also 232 Byte Speicherplatz benötigt (`integer`-Werte benötigen 2 Byte).

Noch etwas sparsamer geht die folgende Methode mit dem vorhandenen Speicherplatz um. Der Indexvektor für die Zeilen wird hierbei eingespart. Hinzu kommt jedoch ein Indexvektor, in dem jeweils der Anfang der nächsten Zeile gespeichert wird.

Methode 2.7 (Kompaktes Zeilenweises Speicherformat)
Die Einträge ungleich Null der spärlich besetzten Matrix werden zeilenweise gespeichert. Dazu werden folgende Arrays benötigt:

1. *Werte-Array* `W` *für die Werte der Einträge.*
2. *Index-Array* `J` *für die Spaltenposition des Eintrages in der Matrix.*
3. *Pointerarray* `K` *für den Anfang (das Ende) der jeweiligen Spalte in der Liste.*

Initialisiere l mit Null. Gehe zeilenweise (Index i) durch die Matrix **A***: Erhöhe l für jeden Eintrag um 1, trage im i-ten Element von* `K` *die Anzahl der schon besetzten Einträge (l) ein, trage an der l-ten Position in* `J` *die Spalte (j) und an der l-ten Position in* `W` *den Wert von* $a_{i,j}$ *ein (siehe Abb. 2.14).*

Mit dieser Form der Speicherung werden NZN reelle Werte gespeichert, NZN natürliche Zahlen für die Spaltenindizes und $N + 1$ natürliche Zahlen für die Pointer. Insgesamt entspricht dies 202 Byte Speicherplatz.

Die Einfügung neuer Matrixelemente (bei adaptiver Gitterverfeinerung) ist in dieser Speichermethode nicht möglich. Diesen Nachteil kann man umgehen, indem man in *verketteten Listen* speichert (siehe [130]). Diese Methode ist jedoch recht aufwendig und soll hier nicht genauer beschrieben werden.

Für symmetrische Matrizen läßt sich noch mehr Platz sparen. Beispielsweise reicht es, nur die untere Diagonalmatrix mit der Diagonalen zu speichern. Beim Pointerarray läßt sich eine Stelle einsparen, indem nur die Endpositionen der Zeilen gehalten werden, weil klar ist, daß das Array an der Position 1 beginnt. Zusammengefaßt:

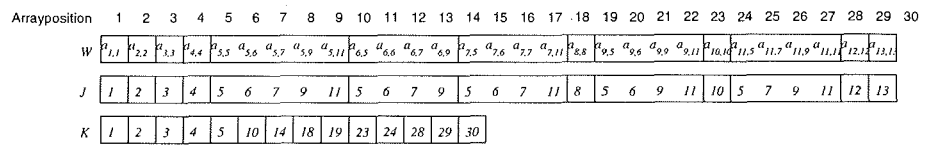

Abbildung 2.14: Kompaktes zeilenweises Speicherformat

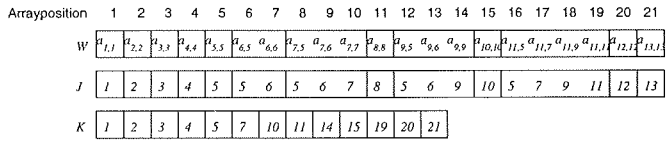

Abbildung 2.15: Symmetrisches kompaktes zeilenweises Speicherformat

Methode 2.8 (Symmetrisches Kompaktes Zeilenweises Speicherformat)
Die verschiedenen Einträge ungleich Null der symmetrischen, spärlich besetzten Matrix werden zeilenweise gespeichert. Dazu werden folgende Arrays benötigt:

1. *Werte-Array* `W` *für die Werte der Einträge.*
2. *Index-Array* `J` *für die Spaltenposition des Eintrages in der Matrix.*
3. *Pointerarray* `K` *für den Anfang (das Ende) der jeweiligen Spalte in der Liste.*

Initialisiere l mit Null. Gehe zeilenweise (Index i) durch die Matrix **A***: Erhöhe l für jeden Eintrag um 1, trage an der l-ten Position in* `J` *die Spalte (j) und an der l-ten Position in* `W` *den Wert von* $a_{i,j}$ *ein, wobei* $j \leq i$ *(Symmetrie), trage im i-ten Element von* `K` *die Anzahl der schon besetzten Einträge (l) ein, (siehe Abb. 2.15).*

Mit der Methode 2.8 erreicht man mit 21 Einträgen insgesamt einen Speicherbedarf von 154 Byte für die Matrix aus (2.6).

Eine einfache Methode zur Speicherung einer Matrix mit (vorhersagbarer) schmaler Bandbreite ν besteht darin, sie in eine Matrix der Größe $N \times \nu$ zu komprimieren und zusätzlich die Spaltenindizes in einer Indexmatrix derselben Größe abzulegen:

Methode 2.9 (Ellpack-Itpack Format)
Die Zeilen der spärlich besetzten $N \times N$*-Matrix* **A** *mit Bandbreite* ν *werden in folgenden Datenstrukturen gespeichert:*

1. *Reelle Matrix* `W` *der Größe* $N \times \nu$ *für die Einträge.*

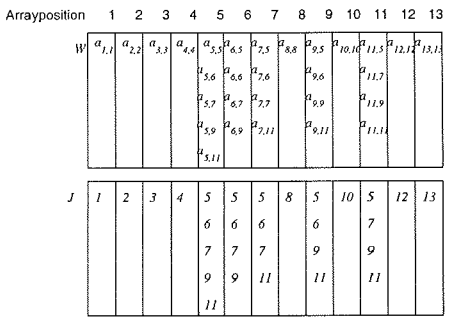

Abbildung 2.16: Ellpack-Itpack Format

2. Matrix J *der Größe* $N \times \nu$ *für die Spaltenindizes.*

Jede Zeile der Matrix W *enthält die Werte der Einträge der Zeile von* $\mathbf{A}$, *in* J *steht der zugehörige Spaltenindex des Wertes (siehe Abb. 2.16).*

Mit Hilfe der Methode 2.9 kann die Einsparung von Speicherplatz nicht ganz so hoch ausfallen wie mit den beiden zuvor genannten Methoden. Immerhin werden für die beiden Felder nun 546 Byte Speicherplatz benötigt. Vor allem bei größeren Matrizen mit vielen inneren Punkten (an denen die Zeilen dann die ganze Bandbreite der Matrix einnehmen) ist diese Methode annähernd so speichereffizient wie die Methoden 2.7 und 2.8 und wegen ihrer Einfachheit vorzuziehen.

Nachträgliches Einfügen von Zeilen ist bei dieser Speichermethode ebenso unmöglich wie bei den anderen vorgestellten Methoden. In [20] wird eine Erweiterung der Methode vorgeschlagen, um das Einfügen von Zeilen zu ermöglichen. In der Implementierung wird die Steifigkeitsmatrix nach jeder Gitteränderung neu aufgestellt, daher wird keine solche Erweiterung benötigt.

2.4 Lösungsverfahren für das lineare Gleichungssystem

In diesem Abschnitt werden im wesentlichen zwei iterative Lösungsverfahren für große lineare Gleichungssysteme vorgestellt. Diese Wahl hängt mit der Historie der Implementierung zusammen. Traditionell werden CG-Verfahren [57] zur Lösung von linearen Systemen aus FEM-Diskretisierungen verwendet, weil sie gut für die dünn besetzte Struktur der Matrizen geeignet sind. Die notwendige Struktur der Diskretisierungsmatrix (symmetrisch positiv definit) ist für einfache Differentialoperatoren und Dirichlet-Randbedingungen gegeben.

Ein wesentlicher Teil der Parallelisierung und Implementierung wurde mit einem CG-Programm durchgeführt in der Annahme, daß das im Flachwassermodell auftretende System symmetrisch sei. Als sich bei der Implementierung des Flachwassermodells herausstellte, daß wegen des Coriolisparameters und wegen der periodischen Randbedingungen keine Symmetrie erreicht werden kann, mußte auf ein weiteres Lösungsverfahren für unsymmetrische Matrizen eingegangen werden. Dabei stellte sich nach einigen Versuchen das Verfahren *BiCGSTAB* (siehe [137]) als geeignet heraus.

Die Beschreibung und Funktionsweise der Verfahren können zum Beispiel in [15] oder [104] gefunden werden. Es soll daher an dieser Stelle lediglich auf die Implementierungen und einige Besonderheiten eingegangen werden.

Das implementierte CG-Verfahren ist in Abb. 2.17 als Pseudocode angegeben. Die Präkonditionierung (siehe Abschnitt 2.5) ist in der Matrixmultiplikation „versteckt". Die auftretenden Vektoroperationen lassen sich leicht mit BLAS-Routinen [78] implementieren. Das ist sinnvoll, falls diese Bibliotheken optimiert sind, was für die Vektoroperationen (BLAS level 1) in der Regel jedoch nicht gilt. Daher ist in der Implementierung auf die Verwendung der Programmbibliothek verzichtet worden.

Die Implementierung von BiCGSTAB stammt aus der Programmsammlung zu [15]. Der Pseudocode ist in Abb. 2.18 angegeben.

2.5 Präkonditionierung für iterative Verfahren

Jede iterative Methode konvergiert in nur einem Schritt, falls die Matrix $\mathbf{A}$ des linearen Systems

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.7}$$

$h_0 = \mathbf{A}x_0$
$r_0 = b - h_0$
$p_0 = r_0$
$\beta_0 = r_0^\top \cdot r_0$
for $k = 1, 2, \ldots$ bis Konvergenz erreicht do
    $h_{k-1} = \mathbf{A}p_{k-1}$ (Matrixmultiplikation)
    $\gamma_{k-1} = p_{k-1}^\top \cdot h_{k-1}$ (1. Schleife)
    $\alpha_{k-1} = \frac{\beta_{k-1}}{\gamma_{k-1}}$
    $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$ (2. Schleife)
    $r_k = r_{k-1} - \alpha_{k-1}h_{k-1}$ (3. Schleife)
    $\beta_k = r_k^\top \cdot r_k$ (4. Schleife)
    $p_k = r_k + \frac{\beta_k}{\beta_{k-1}}p_{k-1}$ (5. Schleife)
enddo

Abbildung 2.17: Pseudocode für das implementierte CG-Verfahren

die Einheitsmatrix $\mathbf{I}$ ist. Das Ziel von Präkonditionierungsmethoden ist daher, eine leicht zu invertierende Matrix $\mathbf{Q}$ zu finden, so daß $\tilde{\mathbf{A}} := \mathbf{A}\mathbf{Q}^{-1} \approx \mathbf{I}$ gilt. Mit der präkonditionierten Matrix $\tilde{\mathbf{A}}$ wird dann das Iterationsverfahren auf ein modifiziertes System

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{2.8}$$

angewendet. Dabei ist in diesem Falle $\tilde{\mathbf{x}} = \mathbf{Q}\mathbf{x}$ und $\tilde{\mathbf{b}} = \mathbf{b}$. Da $\mathbf{A}$ von rechts mit $\mathbf{Q}^{-1}$ multipliziert wird, heißt diese Art der Präkonditionierung *rechte Präkonditionierung*. Analog läßt sich die *linke Präkonditionierung* durchführen, dann ist $\tilde{\mathbf{A}} = \mathbf{Q}^{-1}\mathbf{A}$, $\tilde{\mathbf{x}} = \mathbf{x}$ und $\tilde{\mathbf{b}} = \mathbf{Q}^{-1}\mathbf{b}$. Falls sich $\mathbf{Q}$ multiplikativ in $\mathbf{Q} = \mathbf{Q}_1\mathbf{Q}_2$ aufspalten läßt, kann *beidseitige Präkonditionierung* angewendet werden, wobei dann $\tilde{\mathbf{A}} = \mathbf{Q}_1^{-1}\mathbf{A}\mathbf{Q}_2^{-1}$, $\tilde{\mathbf{x}} = \mathbf{Q}_2\mathbf{x}$ und $\tilde{\mathbf{b}} = \mathbf{Q}_1^{-1}\mathbf{b}$.

Im folgenden werden lediglich zwei Formen der Präkonditionierung kurz vorgestellt. Bei der sogenannten *Jakobi-* (oder auch *diagonalen*) Präkonditionierung ist $\mathbf{Q}$ gerade die Diagonale der Matrix $\mathbf{A}$. Diese Präkonditionierung eignet sich bei diagonaldominanten Matrizen (d.h. $|A_{i,i}| > |\sum_{j \neq i} A_{i,j}|$). Sie hat den Vorteil vollständig parallelisierbar und numerisch sehr billig zu sein. Es lohnt sich manchmal, die relativ schlechte Konvergenzeigenschaft der Methode in Kauf zu nehmen, wenn dafür die Parallelisierbarkeit erhalten bleibt und die reale Ausführungszeit kürzer ist.

Die Präkonditionierung mit Hilfe der *hierarchischen Basen* wird etwas ausführlicher dargestellt. Weitere Methoden zur Präkonditionierung sind beispielsweise in [104] zu finden.

2.5.1 Präkonditionierung mit hierarchischen Basen

Die Beschreibung der Präkonditionierung mit hierarchischen Basen folgt der Darstellung in [147]. Zunächst wird das theoretische Ergebnis vorgestellt, das die hierarchischen Basen so attraktiv macht: Die spektrale Konditionszahl der Diskretisierungsmatrix bezüglich der hierarchischen Basen nimmt mit $\mathcal{O}(j^2)$ zu, während die Konditionszahl der nodalen Diskretisierungsmatrix mit $\mathcal{O}(2^j)$ wächst. Dabei bezeichnet j die Verfeinerungsstufe. Im Anschluß wird ein einfacher Algorithmus vorgestellt, der die nodale Diskretisierungsmatrix in eine hierarchische Matrix überführt und umgekehrt und sich daher wie ein Präkonditionierungsverfahren darstellt.

$r_0 = b - \mathbf{A}x_0$
Wähle ein $\widehat{r_0}$ z.B. $\widehat{r_0} = r_0$
for $k = 1, 2, \ldots$ bis Konvergenz erreicht do
    $\rho_{k-1} = \widehat{r_0}^{\top} \cdot r_{k-1}$ (1. Schleife)
    if $(\rho_{k-1} = 0)$: Verfahren bricht ab
    endif
    if $(k = 1)$:
        $p_k = r_{k-1}$
    else:
        $\beta_{k-1} = \frac{\rho_{k-2}}{\rho_{k-1}} \cdot \frac{\alpha_{k-1}}{\omega_{k-1}}$
        $p_k = r_{k-1} + \beta_{k-1}(p_{k-1} - \omega_{k-1} v_{k-1})$ (2., 3. Schleife)
    endif
    $v_k = \mathbf{A}p_k$ (Matrixmultiplikation)
    $\alpha_k = \frac{\rho_{k-1}}{\widehat{r_0}^{\top} \cdot v_{k-1}}$ (4. Schleife)
    $s_k = r_{k-1} - \alpha_k v_k$ (5. Schleife)
    if ($\|s_k\|$ „klein"): (6. Schleife)
        $x_k = x_{k-1} + \alpha_k p_k$
        stop
    endif
    $t_k = \mathbf{A}s_k$ (Matrixmultiplikation)
    $\omega_k = \frac{t_k^{\top} \cdot s_k}{t_k^{\top} \cdot t_k}$ (7., 8. Schleife)
    $x_k = x_{k-1} + \alpha_k p_k + \omega_k s_k$ (9., 10. Schleife)
    $r_k = r_{k-1} - \omega_k t_k$ (11. Schleife)
enddo

Abbildung 2.18: Pseudocode für das Verfahren BiCGSTAB, die doppelte Schleifenzählung ergibt sich aus der Nutzung von BLAS Routinen

Einführung der hierarchischen Basis und theoretisches Ergebnis

Sei N_k die Menge der Knoten der Triangulierung T_k, wobei hier k die Verfeinerungsstufe bezeichnet. Eine Triangulierung T_k geht aus einer Triangulierung T_{k-1} durch Verfeinerung nach der oben beschriebenen Methode 2.5 hervor (d.h. $T_{k-1} \prec T_k$). $\mathcal{S}_k$ bezeichne die Finite-Elemente-Funktionen (stetige, stückweise lineare Funktionen) auf den Knoten N_k.

Bemerkung 2.3 *Es gilt:* $N_k \subseteq N_{k+1}$ *und* $\mathcal{S}_k \subseteq \mathcal{S}_{k+1}$.

Die *Interpolation* $I_k : C^1(\mathcal{G}) \to \mathcal{S}_k$ definiert durch $(I_k u)(x) = u(x)\ \forall x \in N_k$ bildet stetige Funktionen in den Raum der FEM-Funktionen $\mathcal{S}_k$ ab.

Bemerkung 2.4 *Sei* $u \in \mathcal{S}_j$ *FEM-Funktion. Dann besitzt* u *die Darstellung*

$$u = I_0 u + \sum_{k=1}^{j} (I_k u - I_{k-1} u). \tag{2.9}$$

Die Funktion $J_k(u) := I_k u - I_{k-1} u$ ist offensichtlich in $\mathcal{S}_k$ enthalten, außerdem gilt:

$$(I_k u - I_{k-1} u)(x) = 0\ \forall x \in N_{k-1}.$$

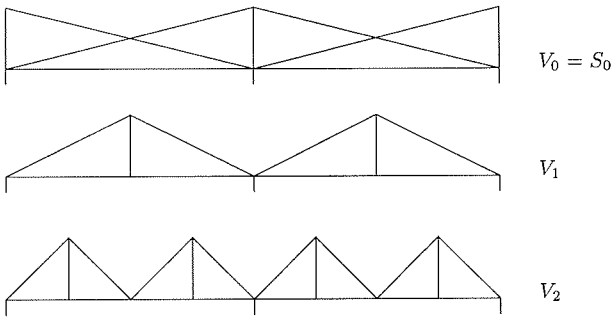

Abbildung 2.19: Hierarchische Basen, Basisfunktionen zu den Räumen $\mathcal{V}_0$, $\mathcal{V}_1$ und $\mathcal{V}_2$

Die J_k induzieren eine Aufspaltung des Raumes $\mathcal{S}_j$ in eine direkte Summe von Unterräumen $\mathcal{V}_k$. Dabei gilt $\mathcal{V}_k \subset \mathcal{S}_k$, $k = 1, \ldots, j$. Außerdem gilt für Funktionen $u \in \mathcal{V}_k$: $u(x) = 0 \; \forall x \in N_{k-1}$. $\mathcal{V}_k$ ist das Bild der Abbildung J_k. Das heißt wegen (2.9) und mit $\mathcal{V}_0 = \mathcal{S}_0$ ist

$$\mathcal{S}_j = \bigoplus_{k=0}^{j} \mathcal{V}_k.$$

Definition 2.5 (Hierarchische Basis) *Eine Triangulierung T_j mit dem dazugehörigen FEM-Raum $\mathcal{S}_j$ und der Aufspaltung in Unterräume $\mathcal{V}_k$, $k = 1, \ldots, j$ sei gegeben. Die Hierachische Basis wird rekursiv definiert:*

1. *Für $j = 0$ ist die hierarchische Basis durch die (nodale) Basis des FEM-Raumes $\mathcal{S}_j$ gegeben.*
2. *Für $j \geq 1$ ist die hierarchische Basis durch die Basis des Raumes $\mathcal{S}_{j-1}$ und die Basis des Raumes $\mathcal{V}_j$ gegeben*

Für eindimensionale Probleme ist eine hierarchische Basis in Abb. 2.19 angegeben. Die Basisfunktionen eines Raumes $\mathcal{V}_k$ spannen allein noch keinen FEM-Funktionenraum auf, erst die Summe der $\mathcal{V}_k$ ergibt eine solche vollständige Basis.

Mit Hilfe der Aufspaltung von $\mathcal{S}_j$ in Unterräume und der Darstellung (2.9) kann eine gitterabhängige Seminorm für $u \in \mathcal{S}_j$ definiert werden:

$$|u|^2 := \sum_{k=1}^{j} \sum_{x \in N_k \setminus N_{k-1}} |J_k(u)(x)|^2, \quad u \in \mathcal{S}_j. \tag{2.10}$$

Bemerkung 2.6 *Die Seminorm aus (2.10) hat eine einfache Interpretation. Sie beschreibt gerade die Euklidische Norm des Koeffizientenvektors der Funktion u mit Ausnahme der Koeffizienten die der Anfangstriangulierung (T_0) entsprechen.*

Definition 2.7 *Mit Hilfe von (2.10) wird die Norm*

$$\|u\|^2 := \|I_0 u\|^2_{H^1(\mathcal{G})} + |u|^2 \tag{2.11}$$

definiert.

Die in (2.11) definierte Norm ist nach Yserentant äquivalent zur Sobolevnorm von u [147]. Im Beweis werden lineare FEM-Basisfunktionen vorausgesetzt.

Satz 2.8 *Es gibt positive Konstanten c und C, so daß für alle Funktionen $u \in \mathcal{S}_j$ gilt:*

$$\frac{c}{(j+1)^2}\|u\|^2 \leq \|u\|^2_{H^1(\mathcal{G})} \leq C\|u\|^2.$$

Dabei hängen c und C nicht von der Form des Gebietes $\mathcal{G}$ und der Anzahl der Verfeinerungen j ab.

Bemerkung 2.9 *Da die Norm der Projektion von u auf den Ausgangsraum $\mathcal{S}_0$ fest ist, besagt der Satz 2.8, daß sich die Euklidische Norm des Koeffizientenvektors von u im wesentlichen so verhält, wie die Sobolevnorm.*

Sei nun das Randwertproblem (2.1) gegeben. Die Bilinearform a induziert die *Energienorm*

$$\|u\|^2_E := a(u,u). \tag{2.12}$$

Die Energienorm ist äquivalent zur Sobolevnorm, d.h. es gibt Konstanten ξ und Ξ, so daß $\xi\|u\|_{H^1(\mathcal{G})} \leq \|u\|_E \leq \Xi\|u\|_{H^1(\mathcal{G})}$. Mit Hife der diskreten Seminorm aus (2.10) und der Energienorm aus (2.12) wird die diskrete Norm

$$\|u\|^2_d := \|I_0 u\|^2_E + |u|^2 \tag{2.13}$$

definiert. Aus Satz 2.8 folgt unmittelbar:

Satz 2.10 *Es gibt positive Konstanten c und C, die unabhängig von der Anzahl der Verfeinerungen j sind, so daß gilt:*

$$\frac{c}{(j+1)^2}\|u\|^2_d \leq \|u\|^2_E \leq C\|u\|^2_d. \tag{2.14}$$

Die Norm in (2.13) ist induziert durch ein inneres Produkt. Sei $\mathbf{A}_0$ die Gramsche Matrix, die durch dieses innere Produkt mit der hierarchischen Basis gebildet wird. Im wesentlichen ist $\mathbf{A}_0$ die Diskretisierungsmatrix bezüglich der nodalen Basisfunktionen von $\mathcal{S}_0$. Dies ist leicht einzusehen, weil die hierarchischen Basisfunktionen orthogonal sind. Sei andererseits $\mathbf{A}_h$ die Diskretisierungsmatrix des Problems (2.1) bezüglich der hierarchischen Basen. Dann besagt der Satz 2.10, daß für alle Koeffizientenvektoren $\mathbf{x}$ gilt:

$$\frac{c}{(j+1)^2}(\mathbf{x}, \mathbf{A}_0\mathbf{x}) \leq (\mathbf{x}, \mathbf{A}_h\mathbf{x}) \leq C(\mathbf{x}, \mathbf{A}_0\mathbf{x}). \tag{2.15}$$

Dabei ist mit $(\cdot,\cdot)$ das Euklidische innere Produkt bezeichnet. Mit der Cholesky-Zerlegung der Matrix $\mathbf{A}_0 = \mathbf{L}\mathbf{L}^\top$ besagt (2.15), daß die Konditionszahl κ der Matrix $\mathbf{L}^{-1}\mathbf{A}_h\mathbf{L}^{-\top}$ begrenzt ist durch:

$$\kappa(\mathbf{L}^{-1}\mathbf{A}_h\mathbf{L}^{-\top}) \leq \frac{C}{c}(j+1)^2. \tag{2.16}$$

Da die Konditionszahl der Matrix $\mathbf{A}_0$ unabhängig von der Anzahl der Verfeinerungen j fest gegeben ist, bedeutet (2.16), daß die Konditionszahl der hierarchischen Diskretisierungsmatrix $\mathbf{A}_h$ selbst nur wie $\mathcal{O}(j^2)$ zunimmt. Die Konditionszahl der nodalen Diskretisierungsmatrix $\mathbf{A}_n$ steigt dagegen exponentiell mit der Anzahl der Verfeinerungsstufen (bei regulärer Verfeinerung).

```
for k = 1,2,...,j do
    for i ∈ M_k = {i : i Index eines Knotens in N_k \ N_{k-1}} do
        x(i)= x(i)+ [x(n_1(i))+ x(n_2(i))]* 0.5
    enddo
enddo

for k = j,j-1,...,1 do
    for i ∈ M_k do
        x(n_1(i))= x(n_1(i))+ x(i)* 0.5
        x(n_2(i))= x(n_2(i))+ x(i)* 0.5
    enddo
enddo
```

Abbildung 2.20: Pseudocode für die Operation der Transformationsmatrizen $\mathbf{S}^\top$ und $\mathbf{S}$ der hierarchischen Basen Präkonditionierung. $n_1(i)$ und $n_2(i)$ sind die Indizes der (groben) Nachbarknoten des Knotens i

Präkonditionierungsalgorithmus

Die Berechnung der hierarchischen Diskretisierungsmatrix $\mathbf{A}_h$ wird nicht explizit vorgenommen. Die numerisch vorteilhaften Eigenschaften der nodalen Matrix $\mathbf{A}_n$ (spärlich besetzt, möglicherweise symmetrisch und positiv definit) sollen nicht aufgegeben werden.

Sei $\mathbf{S}$ die Matrix, die einen Koeffizientenvektor $\mathbf{x}$ bezüglich der hierarchischen Basen in einen Koeffizientenvektor $\hat{\mathbf{x}}$ bezüglich der nodalen Basis überführt. Für alle (hierarchischen) Koeffizientenvektoren gilt dann:

$$(\mathbf{x}, \mathbf{A}_h\mathbf{y}) = (\mathbf{S}\mathbf{x}, \mathbf{A}_n\mathbf{S}\mathbf{y}) = (\mathbf{x}, \mathbf{S}^\top\mathbf{A}_n\mathbf{S}\mathbf{y}).$$

Die hierarchische Diskretisierungsmatrix läßt sich also darstellen:

$$\mathbf{A}_h = \mathbf{S}^\top\mathbf{A}_n\mathbf{S}. \tag{2.17}$$

Sind nun Algorithmen für die Operation von $\mathbf{S}$ und $\mathbf{S}^\top$ bekannt, ist die iterative Lösung des diskreten Systems bezüglich hierarchischer Basen äquivalent zu einem beidseitig präkonditionierten System bezüglich der nodalen Basis. Dabei gilt in Analogie zur Darstellung in (2.8): $\mathbf{Q}_1 = \mathbf{S}$, $\mathbf{Q}_2 = \mathbf{S}^{-1}$ und $\mathbf{S}^\top = \mathbf{S}^{-1}$.

Die Algorithmen sind in [147] angegeben. Sie wurden schon in einer Implementierung in [58] vorgestellt und sollen hier nur als Pseudocode angegeben werden (siehe Abb. 2.20).

2.6 Randbedingungen

Dieser Abschnitt stellt einige grundlegende Techniken zur Darstellung der verschiedenen Randbedingungen vor. Je nach Art der Randbedingung muß die FEM-Steifigkeitsmatrix oder die rechte Seite des Gleichungssystems modifiziert werden.

Die verschiedenen Randbedingungen aus dem Unterabschnitt 1.2.2 werden hier aufgegriffen und deren Implementierung kurz erklärt. **Homogene Dirichlet-Randbedingungen** (Bemerkung 1.1) sind am einfachsten zu implementieren. Die Steifigkeitsmatrix wird dabei so modifiziert,

daß die Spalten und Zeilen, die dem Index eines Randpunktes entsprechen, auf Null gesetzt werden bis auf das Diagonalelement, das auf Eins gesetzt wird. Gleichzeitig wird auf der rechten Seite das Vektorelement mit dem zugehörigen Index auf Null gesetzt. Wenn die Matrix zuvor symmetrisch war, so wird die Symmetrie nicht durch diese Modifikation zerstört.

Bei **nichthomogenen Dirichlet-Randbedingungen** (Bemerkung 1.2) muß der Eintrag auf der rechten Seite nicht auf Null, sondern auf den entsprechenden (vorgegebenen) Randwert gesetzt werden. Außerdem dürfen (im Unterschied zu homogenen Dirichlet-Randbedingungen) die Spalten nicht Null gesetzt werden. Dadurch erhält die Matrix eine (leicht) unsymmetrische Struktur.

Die Behandlung von **Neumann Randbedingungen** ist im Falle von **homogenen** Rändern trivial. Wie aus Bemerkung 1.3 ersichtlich ist, verändern homogene Neumann Randbedingungen weder die Matrix, noch die rechte Seite, wenn man die Linearform und die Bilinearform auf allen (inneren und Rand-) Punkten berechnet.

Etwas komplizierter wird die Einbeziehung **nichthomogener Neumann Randbedingungen**. Wie aus der Bemerkung 1.3 hervorgeht, muß bei der Berechnung der rechten Seite ein zusätzliches Randintegral berechnet werden. Die Berechnung der Einträge der rechten Seite erfolgt analog zur Aufstellung der Steifigkeitsmatrix in Abschnitt 2.3: Für jedes Element müssen die Integrale

$$\int_{\tau \in Tr(b_i)} f b_i \, d\mathcal{G}$$

berechnet werden und die entsprechenden Beiträge auf das zugehörige Vektorelement der rechten Seite addiert werden. Wie bei der Aufstellung der Steifigkeitsmatrix kann elementweise (EBE) oder punktweise Bearbeitungsreihenfolge gewählt werden.

Bei der Einbeziehung der nichthomogenen Neumann Randbedingung muß in den Elementen, deren Kante mit dem Rand übereinstimmt, zusätzlich das Linienintegral

$$\int_{\substack{\tau \in Tr(b_i) \\ \tau \cap \Gamma \neq \emptyset}} q b_i \, d\Gamma$$

berechnet werden. Diese Berechnung geschieht mit Hilfe einer Quadraturformel. Für lineare Basisfunktionen ist die folgende Quadraturformel exakt:

$$\frac{q(x_1) + q(x_2)}{2} \cdot |x_1 - x_2|. \tag{2.18}$$

Dabei seien x_1 und x_2 die beiden Knoten der Randkante.

Die Implementierung **periodischer Ränder** ist hier in einer sehr einfachen und numerisch nicht unbedigt optimalen Form erfolgt. Zunächst müssen jedoch noch einige Vorbemerkungen zu periodischen Rändern erfolgen: Die Implementierung sieht periodische Ränder sowohl in nur einer Koordinatenrichtung vor, als auch doppelt periodische Ränder. Jeder Punkt auf einem periodischen Rand tritt doppelt auf. Dabei sind die Koordinaten der Punkte verschieden, die Werte aber identisch. Daher wird ein „Master" und ein „Slave" definiert. Der Master wird in der Berechnung der Steifigkeitsmatrix und der rechten Seite wie ein innerer Punkt behandelt. Der Slave wird mit dem Master gleichgesetzt.

Zur Behandlung von Periodischen Randbedingungen wird die Matrix modifiziert, um Master und Slave zu identifizieren. Das erfolgt über die zugehörige Zeile in der Steifigkeitsmatrix: Auf der Diagonalen wird eine 1 eingetragen, an der Spaltenposition des Masters eine -1, und die rechte Seite wird auf Null gesetzt.

Diese Behandlung führt zu einer unsymmetrischen Matrixstruktur und zu künstlichen Eigenwerten des Systems. Praktisch wirkt sich diese Art der Randbehandlung aber nicht wesentlich auf die Leistung des Lösungsverfahrens aus.

Kapitel 3

Parallelisierung der Finite-Elemente-Methode

3.1 Übersicht

Die Lösung großer Probleme mit Hilfe von FEM läßt sich ohne die Nutzung von Parallelrechnern nicht mehr durchführen. Selbst auf der Ebene von Hochleistungsworkstations sind inzwischen Mehrprozessorsysteme im Angebot, die zur Beschleunigung der Antwortzeiten parallele Programme akzeptieren. Da FEM-Programme in der Regel numerisch aufwendiger sind als FDM-Verfahren, ist die Notwendigkeit für Parallelisierung besonders evident.

Die Parallelisierung von FEM läßt sich in drei verschiedene Aufgaben unterteilen:

1. Die Parallelisierung der Gittergenerierung
2. Die Parallelisierung der Matrixaufstellung
3. Die Parallelisierung des Lösungsverfahrens

Parallele Gittergenerierung wird in dieser Arbeit nicht behandelt werden. Einige Ansätze dazu sind in [122, 95, 62] zu finden.

Die Aufstellung der Diskretisierungsmatrix (und der rechten Seite) muß in adaptiven zeitabhängigen Verfahren in jedem Zeitschritt neu erfolgen. Die Parallelisierung ist daher notwendig. Außerdem wird die Rechenzeit für diese Aufgabe dominierend, sobald die Lösungsroutinen effizient parallelisieren.

Prinzipiell sind zwei Methoden zur Aufstellung der Matrix möglich. Die *Element-by-element*-Methode stellt die Matrixeinträge elementweise zusammen. Die dabei auftretende Datenabhängigkeit kann mit Hilfe einer Farbindizierung entkoppelt werden. In [21] wird ein Einfärbungsalgorithmus für globale Verfeinerungen vorgestellt, der auf Vererbungseigenschaften beruht. Die Ideen des Algorithmus werden in dieser Arbeit weiterentwickelt und modifiziert, so daß auch lokal verfeinerte Gitter eingefärbt werden können. Andere Möglichkeiten der Einfärbung werden in [96, 68] angegeben.

In der zweiten Methode wird die Diskretisierungsmatrix punktweise berechnet. Dabei entsteht keine Datenabhängigkeit, aber die Verteilung der Daten muß beachtet werden. Die Index-Array-Methode wird eingeführt, um effizienten Datenzugriff auch für Computer mit verteilter Speicherarchitektur zu gewährleisten. Diese Strategie, ursprünglich für die Parallelisierung der Matrixassemblierung entwickelt, läßt sich auch auf andere Routinen übertragen und führt dann zu einer Datenverteilung, die mit wenig Kommunikation auskommt.

$h_0 = \mathbf{A}x_0$
$r_0 = b - h_0$
$p_0 = r_0$
`for` $k = 1, 2, \ldots$ bis Konvergenz erreicht `do`
  $h_{k-1} = \mathbf{A}p_{k-1}$ (Matrixmultiplikation)
  $\gamma_{k-1} = p_{k-1}^{\top} \cdot h_{k-1}$ (1. Schleife)
  $\beta_{k-1} = r_{k-1}^{\top} \cdot r_{k-1}$ (1. Schleife)
  $\delta_{k-1} = r_{k-1}^{\top} \cdot h_{k-1}$ (1. Schleife)
  $\zeta_{k-1} = h_{k-1}^{\top} \cdot h_{k-1}$ (1. Schleife)
  $\alpha_{k-1} = \frac{\beta_{k-1}}{\gamma_{k-1}}$
  $\beta_k = \beta_{k-1} - 2\alpha_{k-1}\delta_{k-1} + \alpha_{k-1}^2\zeta_{k-1}$
  $x_k = x_{k-1} + \alpha_{k-1}p_{k-1}$ (2. Schleife)
  $r_k = r_{k-1} - \alpha_{k-1}h_{k-1}$ (2. Schleife)
  $p_k = r_k + \frac{\beta_k}{\beta_{k-1}}p_{k-1}$ (2. Schleife)
`enddo`

Abbildung 3.1: Pseudocode für das modifizierte CG-Verfahren

Die Parallelisierung der Löser wird im folgenden nur einführend untersucht. Falls parallelisierte BLAS (Basic Linear Algebra Subprograms, siehe [78]) Bibliotheksroutinen verfügbar sind, braucht keine Optimierung von Hand vorgenommen zu werden. Für BLAS level 1 (Vektor-Vektor-Operationen) stehen jedoch auf den wenigsten Systemen optimierte Routinen zur Verfügung. Daher wird eine Methode vorgestellt, die Anzahl der Schleifen im CG-Verfahren zu minimieren. Damit kann der Overhead für den Start paralleler Prozesse minimiert werden. Die Anwendung der Index-Array-Strategie auf CG-ähnliche Verfahren wird beispielhaft vorgeführt.

Weitere Ansätze zur Parallelisierung von CG-Verfahren sind in [82, 125, 40] zu finden. Die Parallelisierung der FEM mit anderen Methoden wird in [53, 14, 150, 121] behandelt.

3.2 Parallelisierung der iterativen Lösungsverfahren

Dieser Abschnitt wird zwei Aspekte der Parallelisierung von CG-ähnlichen Verfahren beleuchten: Die *Minimierung der Schleifenzahl* im Algorithmus zur Reduktion des Overheads für das Starten paralleler Prozesse und die *Anwendung der Index-Array-Strategie* zur effizienten Datenverteilung.

Die Zusammenfassung von Schleifen im Algorithmus aus Abb. 2.17 ist nur bei den Schleifen 2. und 3. möglich. Zur Berechnung der Schleife 5. muß das Skalarprodukt aus Schleife 4. vorliegen, zur Berechnung der Schleifen 2. und 3. muß das Skalarprodukt aus Schleife 1. berechnet sein.

Zur weiteren Minimierung der Schleifenzahl ist es notwendig, einen Trick anzuwenden[1]. Das Skalarprodukt aus Schleife 3. wird so dargestellt, daß nur „alte" Werte verwendet werden. Dann kann die Berechnung von β_k vorgezogen werden:

$$\begin{aligned} \beta_k &= r_k^{\top} \cdot r_k \\ &= (r_{k-1} - \alpha_{k-1}h_{k-1})^{\top} \cdot (r_{k-1} - \alpha_{k-1}h_{k-1}) \\ &= \beta_{k-1} - 2\alpha_{k-1}r_{k-1}^{\top} \cdot h_{k-1} + \alpha_{k-1}^2 h_{k-1}^{\top} \cdot h_{k-1} \end{aligned}$$

[1] Der Trick zur Minimierung der Schleifenzahl geht auf einen Hinweis von Stephen R. Breit von Kendall Square Research Corp. zurück

Damit können nun die Operationen im CG-Verfahren so zusammengefaßt werden, daß nur noch zwei Schleifen gestartet werden müssen (siehe Abb. 3.1).

Eine typische Zeile im Programmcode der zweiten Schleife hat die folgende Form (Pseudocode):

```
forall i ∈ {i Knoten der Triangulierung} do
    x_k(i)= x_{k-1}(i)+ α_{k-1} p_{k-1}(i)
enddo
```

Im Prinzip ist diese Schleife vollständig parallelisierbar. Im Zusammenhang mit allen anderen Berechnungen des Programms ist jedoch insbesondere bei Computerarchitekturen mit verteiltem Speicher darauf zu achten, daß die Daten, die jeder Prozessor bearbeitet, auch lokal verfügbar sind und nicht erst von entfernten Speicherpositionen abgerufen werden müssen. Um diese Datenlokalität zu gewährleisten, werden *Index-Arrays* eingeführt, die jedem Prozessor „seine" Daten zuordnen. Verschiedene Zugriffsreihenfolgen können in eigenen Index-Arrays abgebildet werden.

Die modifizierte Schleife hat die Form:

```
forall p = 1,...,#Prozessoren do parallel
    forall i ∈ Index-Array(p) do
        x_k(i)= x_{k-1}(i)+ α_{k-1} p_{k-1}(i)
    enddo
enddo parallel
```

In allen parallelen Schleifen des Programms wird nun Prozessor p ausschließlich auf die Knotenwerte schreiben, deren Indizes `i` in `Index-Array(`p`)` gespeichert sind.

In einigen Berechnungen werden Lesezugriffe von entfernten Prozessoren notwendig. Diese lassen sich aber asynchron ausführen, das heißt sie lassen sich mit Rechenoperationen überlappen, so daß sie die Gesamtausführungszeit nicht wesentlich beeinträchtigen. Schreibzugriffe auf entfernte Daten sind dagegen nur synchronisiert durchführbar und daher sehr zeitaufwendig.

Um die Kommunikationsmechanismen der KSR-1 zu nutzen, werden die Index-Arrays für diesen Rechner in einer bestimmten Form erstellt. Die Knotennummern stehen konsekutiv im Index-Array, wobei jedem Prozessor jeweils 16 aufeinander folgende Knotennummern zugeordnet werden. Diese Einteilung entspricht gerade der Größe einer *Subpage* auf der KSR-1 (siehe Anhang B).

Auch für Berechnungen, die Werte von Nachbarknoten einbeziehen, ist diese (heuristische) Datenaufteilung vorteilhaft. Durch den Verfeinerungsalgorithmus liegen die Nachbarn eines Knotens in den meisten Fällen auf nur zwei verschiedenen Subpages. Die kleinste Übertragungseinheit auf der KSR-1 ist eine Subpage. Wenn alle Daten zur Berechnung eines Knotenwertes auf zwei Subpages liegen, müssen lediglich zwei Kommunikationsoperationen durchgeführt werden. Die zusätzliche Kommunikation kann prinzipiell überlappt mit den Berechnungen der Daten der ersten Subpage stattfinden. Sie kostet daher kaum Rechenzeit.

3.3 Parallelisierung mit Farbindizierung

Um zu einem parallelen Verfahren zur Aufstellung der Steifigkeitsmatrix in der EBE Reihenfolge zu gelangen, muß zunächst die dabei auftretende Datenabhängigkeit untersucht werden. Bei der Aufstellung der Steifigkeitsmatrix (siehe Abb. 2.10) wird in einer Schleife über alle Elemente der jeweilige Beitrag auf den zugehörigen Knoteneintrag in der Matrix addiert. Es liegt nahe, über

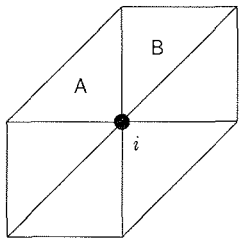

Abbildung 3.2: Typische Situation in einem regulär verfeinerten Gitter: die Beiträge aus den Elementen A und B sollen auf den Matrixeintrag zum Knoten i addiert werden

die Elementschleife zu parallelisieren. Wenn jedoch gleichzeitig von zwei Elementen auf einen Knotenwert i addiert werden soll, muß synchronisiert werden (siehe Abb. 3.2).

Die Idee der Farbindizierung basiert darauf, die Elemente so einzufärben, daß alle einen Knoten umgebenden Elemente verschiedene Farben haben. Dann kann innerhalb einer Farbe parallelisiert werden, ohne daß es zu gleichzeitigen Schreibzugriffen auf denselben Knotenwert kommen kann. Diese Idee wurde schon in [21] für regulär und uniform verfeinerte Gitter entwickelt. Etwa zur gleichen Zeit ist das Verfahren mit einem anderen Mechanismus zum Einfärben der Elemente veröffentlicht worden [96]. Hier ist das Verfahren aus [21] für lokal verfeinerte adaptive Verfahren erweitert und modifiziert worden.

Während in [21] die Färbung der Elemente mit Hilfe eines Vererbungsmechanismus vorgenommen wurde, wird hier eine etwas geänderte Variante verwendet. Die Elemente, die durch Verfeinerung aus einem gegebenen Element hervorgehen, werden nach einem tabellierten Schema eingefärbt, wobei die Wahl der Farben auch von der Art und Lage des Mutterelements abhängt.

Dazu müssen einige Annahmen an die Triangulierung gemacht werden.

Annahme 3.1 (Verfeinerungsstrategie)
Die Triangulierung geht durch (lokal verfeinerte) reguläre Verfeinerung aus einer gegebenen Anfangstriangulierung hervon wie in Methode 2.5 beschrieben. Für grün verfeinerte Elemente wird dabei Alternative 4b. verwendet.

Annahme 3.2 (Ausgangstriangulierung)
Für die Ausgangstriangulierung wird angenommen:

1. *Die Ausgangstriangulierung bestehe aus Dreieckselementen von nur vier Typen mit den Typenindikatoren $I^{Typ} = \{1, -1, 2, -2\}$. Ein Dreieck des Typs $i \in I^{Typ}$ hat folgende Dreieckstypen als Nachbarn (für die Seitennummern siehe Abb. 2.9):*

$$\begin{array}{ll} \textit{Seite 1:} & (-1)^i \cdot sign(i) \cdot j \\ \textit{Seite 2:} & (-1)^i \cdot sign(i) \cdot -j \\ \textit{Seite 3:} & -i \end{array}$$

   wobei $j = |i| \bmod 2 + 1$.
2. *Nicht mehr als zwei Dreiecke des selben Typs dürfen sich in einem Knoten treffen.*

Die vier Dreickstypen und die resultierenden Typen bei regulärer Verfeinerung sind in Abb. 3.3 dargestellt.

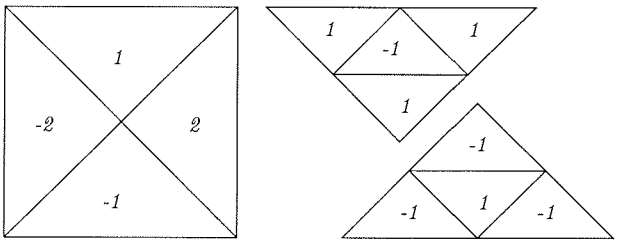

Abbildung 3.3: Dreieckstypen in der Ausgangstriangulierung und nach Verfeinerung

Bemerkung 3.3 *Ein Dreieck des Typs i wird in vier Dreiecke verfeinert. Das mittlere Tochterdreieck hat den Typ $-i$, die äußeren Töchter den Typ i ihrer Mutter.*

In den meisten zweidimensionalen Situationen kann durch Streckung oder Rotation eine Ausgangstriangulierung mit diesen vier Typen erreicht werden. In einigen Fällen muß eine etwas eigenwillige Anfangstriangulierung gewählt werden (siehe [22]). Unmöglich wird die Triangulierung mit dieser Methode, wenn die Orientierung der Dreiecke sich ändert (wie bei der Triangulierung der Sphärenoberfläche). Mit Hilfe der in den Tabellen 3.1 und 3.2 angegebenen Anweisungen zur Färbung der Elemente kann nun folgender Satz formuliert werden:

Satz 3.4 (Einfärbung von Elementen)
Mit den Annahmen 3.1 und 3.2 kann ein Farbmuster in einer Triangulierung erzeugt werden, so daß jeder Knoten der Triangulierung von (paarweise) verschiedenen Elementen umgeben ist, falls gilt:

1. *Dreiecke aus regulärer Verfeinerung werden eingefärbt wie in Tabelle 3.1 angegeben.*
2. *Dreiecke aus grüner Verfeinerung werden wie in Tabelle 3.2 eingefärbt.*
3. *Die Einfärbung wird mit jedem Verfeinerungsschritt permutiert, wie in den Tabellen angegeben.*

Bemerkung 3.5 *Für die Dreiecke einer Farbe in einer nach Satz 3.4 eingefärbten Triangulierung gilt: Jeder Knoten der Triangulierung liegt in maximal einem Element.*

Der Beweis des Satzes ist etwas umständlich und soll hier nicht ausführlich durchgeführt werden (er ist in [22] zu finden). Einige Lemmata helfen bei der Beweisführung. Zunächst wird gezeigt, daß der Satz für die Verfeinerung nur eines Elementes gilt:

Tabelle 3.1: Farbtabelle für reguläre (rote) Verfeinerung (gerader Level/ ungerader Level)

Typ	Farbe des Tochterelements			
	Mitte	Links	Rechts	Oben
+1	1/ 9	2/ 10	3/ 11	4/ 12
−1	5/ 13	6/ 14	7/ 15	8/ 16
+2	3/ 11	4/ 12	5/ 13	6/ 14
−2	7/ 15	8/ 16	1/ 9	2/ 10

Tabelle 3.2: Farbtabelle für grüne Verfeinerung (gerader Level/ ungerader Level)

	Farbe des Tochterelements					
Typ	Seite 1 geteilt		Seite 2 geteilt		Seite 3 geteilt	
	Links	Rechts	Links	Rechts	Links	Rechts
+1	17/33	18/ 34	19/35	20/ 36	21/37	22/ 38
−1	23/39	24/ 40	25/41	26/ 42	27/43	28/ 44
+2	17/33	24/ 40	19/35	30/ 46	27/43	31/ 47
−2	23/39	18/ 34	25/41	32/ 48	21/37	29/ 45

Lemma 3.6 *Gegeben sei ein Dreieckselement, das global (rot) in Übereinstimmung mit Annahme 3.1 verfeinert ist und dessen Kindelemente nach den Bedingungen in Satz 3.4 eingefärbt sind. Die Elemente der so entstehenden Triangulierung, die sich in einem Knoten treffen, haben dann (paarweise) unterschiedliche Farben.*

Lemma 3.7 *Gegeben sei eine Triangulierung, die aus der globalen und anschließenden lokalen Verfeinerung eines Dreieckselements hervorgeht (Annahme 3.1). Mit der Einfärbung aus Satz 3.4 erhält man eine Triangulierung, in der sich keine gleichfarbigen Elemente berühren.*

Im Anschluß kann gezeigt werden, daß der Satz für eine global verfeinerte Anfangstriangulierung mit mehreren Elementen gilt:

Lemma 3.8 *Gegeben sei eine Ausgangstriangulierung mit den Eigenschaften aus Annahme 3.2, die global in Übereinstimmung mit Annahme 3.1 verfeinert ist und deren Elemente eingefärbt sind wie oben beschrieben. Dann sind alle Elemente, die einen Knoten enthalten, paarweise verschieden gefärbt.*

Schließlich kann mit Hilfe der Ergebnisse der Lemmata die Aussage des Satzes auch für lokal verfeinerte Triangulierungen gezeigt werden.

Wenn eine Triangulierung erzeugt ist, deren Elemente so eingefärbt sind, daß keine gleichfarbigen Dreiecke sich berühren, dann kann innerhalb einer Farbe parallelisiert werden. Der Algorithmus aus Abb. 2.10 wird um eine äußere Schleife über alle Farben erweitert. In der inneren Schleife wird über alle Elemente der Farbe parallelisiert (siehe Abb. 3.4).

3.4 Parallelisierung mit Hilfe von Index-Arrays

Die Parallelisierung mit Hilfe von Index-Arrays wurde für eine einzelne Schleife schon im Abschnitt 3.2 vorgestellt. Der in Abb. 2.11 angegebene Algorithmus für die Matrixassemblierung in punktweiser Reihenfolge läßt sich analog zur einzelnen Schleife parallelisieren.

Da jede Zeile der Steifigkeitsmatrix exklusiv von nur einem Prozessor (nämlich dem Prozessor, dem die entsprechende Knotennummer zugeordnet ist) berechnet wird, braucht bei der Parallelisierung nicht gesondert synchronisiert zu werden. Kommunikation zu entfernten Speicherpositionen entsteht lediglich durch den Lesezugriff auf Elementsteifigkeitsmatrizen.

Für die KSR-1 läßt sich jede Elementsteifigkeitsmatrix auf einer eigenen Subpage speichern, so daß nur eine Kommunikation pro Elementsteifigkeitsmatrix auftritt. Da jede Elementsteifigkeitsmatrix mindestens achtmal benötigt wird, ist der Kommunikationsaufwand im Vergleich zum

```
forall φ ∈ {φ Farbe} do
    forall τ ∈ T_φ = {τ hat Farbe φ} do parallel
    (   call Berechne Elementmatrix e_mat )
        forall j=1,2,3 do
            node_j= nt_node(j,τ)
            forall i=1,2,3 do
                node_i= nt_node(i,τ)
                a_mat(node_i,node_j)= a_mat(node_i,node_j)+ e_mat(i,j,τ)
            enddo
        enddo
    enddo parallel
enddo
```

Abbildung 3.4: Pseudocode für die parallelisierte Matrixassemblierung in EBE-Ordnung mit Farbindizierung

arithmetischen Aufwand klein. Der Pseudocode der parallelisierten punktweisen Matrixaufstellung ist in Abb. 3.5 angegeben.

In diesem Fall kann von einer doppelten Index-Array-Methode gesprochen werden. Das erste Index-Array hält zu jedem Prozessor die Knotennummern, das zweite Index-Array speichert zu jedem Knoten die angrenzenden Elementindizes. Aufgrund der Kommunikationsmechanismen auf der KSR-1 und der entsprechenden Datenpartitionierung ergibt sich ein sehr effizientes paralleles Verfahren.

3.5 Probleme und Bemerkungen

Eine abschließende Bewertung der verschiedenen Parallelisierungstechniken kann nur schwer vorgenommen werden. Die EBE-Sortierung der Bearbeitung hat in einigen Fällen Vorteile. Wenn das gesamte Verfahren (also insbesondere die Matrixmultiplikation im iterativen Löser) auf die EBE-Ordnung umgestellt ist, kann auf die Aufstellung einer globalen Steifigkeitsmatrix verzichtet werden. In Verbindung mit Gebietszerlegungstechniken für die Parallelisierung und Präkondi-

```
forall p = 1,...,#Prozessoren do parallel
    forall node_j ∈ Index-Array(p) do
        forall τ ∈ nn_ind(node_j) do
            j= lokale Nummer von node_j
            forall i=1,2,3 do
                node_i= nt_node(i,τ)
                a_mat(node_i,node_j)= a_mat(node_i,node_j)+ e_mat(i,j,τ)
            enddo
        enddo
    enddo
enddo parallel
```

Abbildung 3.5: Pseudocode für die Matrixassemblierung in punktweiser Ordnung

tionierung lassen sich effiziente parallele Verfahren formulieren, die vor allem für Messagepassing Architekturen geeignet sind (siehe [85]).

Auf der anderen Seite ist diese Ordnung bei reinen punktweisen Berechnungen ungeeignet und es entstehen möglicherweise verschiedene Zugriffsmuster für die gleichen Daten. In diesem Fall ist die punktweise Parallelisierungstechnik vorzuziehen. Auch wenn mit einer explizit aufgestellten Diskretisierungsmatrix gerechnet wird, kann es bei der Assemblierung zu unstrukturiertem Zugriff auf die Daten kommen, der wiederum für Messagepassing Architekturen besonders schwierig zu handhaben ist. Im folgenden werden einige Vor- und Nachteile der jeweiligen Parallelisierungsart aufgezählt.

Eigenschaften der Farbindizierung

⊕ Es wird keine globale Steifigkeitsmatrix benötigt, wenn die Matrixmultiplikation ebenfalls in (paralleler) EBE-Reihenfolge durchgeführt wird.

⊕ Blockung ist leicht durchzuführen, weil die Dateneinheiten meist die Größe einer Elementsteifigkeitsmatrix haben.

⊕ Messagepassing ist einfacher, weil Daten eines Elements leicht zu strukturieren sind.

⊖ Bei ausschließlich punktweisen Berechnungen sind verschiedene Zugriffsmuster (und damit zusätzliche Kommunikation) auf denselben Daten möglich.

⊖ Bei expliziter Aufstellung der Steifigkeitsmatrix können unstrukturierte Schreibzugriffe nicht effizient parallelisiert werden.

Eigenschaften der Index-Arrays

⊕ Auch bei verschiedenen Bearbeitungsreihenfolgen lassen sich die Daten lokal fixieren.

⊕ Effizientes Loadbalancing ist möglich.

⊕ Auch strikt punktweise Berechnungen in anderen Programmteilen sind so parallelisierbar, daß keine zusätzliche Kommunikation auftritt.

⊖ Messagepassing schwieriger, weil keine „natürliche“ (elementweise) Strukturierung der Daten vorliegt.

⊖ Blockung schwieriger, weil nicht in kleinen Dateneinheiten (Elementen) gerechnet wird.

In den Verfahren der Teile II und III wird die Parallelisierungsmethode mit Index-Arrays verwendet, weil die zeitaufwendigsten Programmteile strikt punktweise organisiert sind. Bis auf die Berechnung der Elementsteifigkeitsmatrizen und die lokalen Fehlerschätzer sind damit alle Berechnungen in der gleichen Weise organisiert. Die Knotendaten bleiben innerhalb einer Iteration also lokal, das minimiert den Kommunikationsaufwand.

Kapitel 4

Ergebnisse für die Finite-Elemente-Methode

4.1 Übersicht

Die FEM ist als numerische Methode zur Lösung elliptischer Probleme gut eingeführt und abgesichert. Daher werden in diesem Abschnitt keine umfassenden numerischen Tests oder Fehleranalysen durchgeführt. Für die Rechengenauigkeit der Implementierung sind einige Konvergenztests angegeben. Das Gebiet $\mathcal{G}$ wird dabei mit verschiedenen Verfeinerungen trianguliert. Verschiedene Modellprobleme mit bekannten analytischen Lösungen werden dann numerisch gelöst. Der Fehler, der bei der numerischen Lösung auftritt, muß für jedes Modellproblem proportional zur Gitterweite abnehmen.

Der Schwerpunkt der Tests in diesem Kapitel liegt auf der Bewertung der parallelen Effizienz der Verfahren. Dazu werden *Speedup*, *Scaleup* und *Effizienz* der Parallelisierungsmethoden auf diversen Gittern mit unterschiedlichen Modellproblemen untersucht. Für die Assemblierung der Steifigkeitsmatrix werden zwei Parallelisierungsstrategien untersucht, die Farbindizierung und die Index-Array-Methode, das CG-Verfahren wird mit Hilfe von drei Methoden parallelisiert und getestet und auch die parallele Leistung von BiCGSTAB wird untersucht.

Die Parallelisierungstests werden auf einer KSR-1 mit 32 Prozessoren durchgeführt. Das größte Prozessorset besteht aus 26 Prozessoren, die parallel verwendet werden können (siehe Anhang B). Für die Parallelisierungsmethode mit Farbindizierung wurden einige Zeitmessungen auf der Alliant FX/2800 durchgeführt, die bis 1993 als Entwicklungsrechner für parallele Algorithmen am Alfred-Wegener-Institut verfügbar war.

Die Ergebnisse der Parallelisierungstests für die Matrixassemblierung zeigen für beide Parallelisierungsstrategien gute Effizienz. Die KSR-1 erreicht mit der Index-Array-Methode über 90% Effizienz. Die Messungen ergeben annähernd gleiche Ergebnisse für globale wie für lokale Verfeinerungen. Die Farbindizierung schneidet bei Computerarchitekturen mit verteiltem Hauptspeicher jedoch eindeutig schlechter ab. Verschiedene Gitterkonfigurationen sind so gewählt, daß global und lokal verfeinerte Gitter vorkommen und daß zur Bestimmung des Scaleups die Knotenzahl jeweils etwa verdoppelt wird. In den meisten Publikationen werden ausschließlich die Effizienz oder der Speedup des Verfahrens dargestellt. Neben der absoluten Leistung des Verfahrens ist aber im Zusammenhang mit „grand-challenge"-Problemen der Scaleup eine viel wichtigere Größe. Der Scaleup beschreibt das Verhalten der Methode wenn die Problemgröße mit der Rechenleistung skaliert wird. Wenn der Scaleup bei eins liegt, dann kann davon ausgegangen werden, daß das entwickelte Verfahren für große Probleme auf entsprechend vielen Prozessoren in der gleichen Zeit wie das zugrundeliegende Modellproblem gerechnet werden kann (für eine

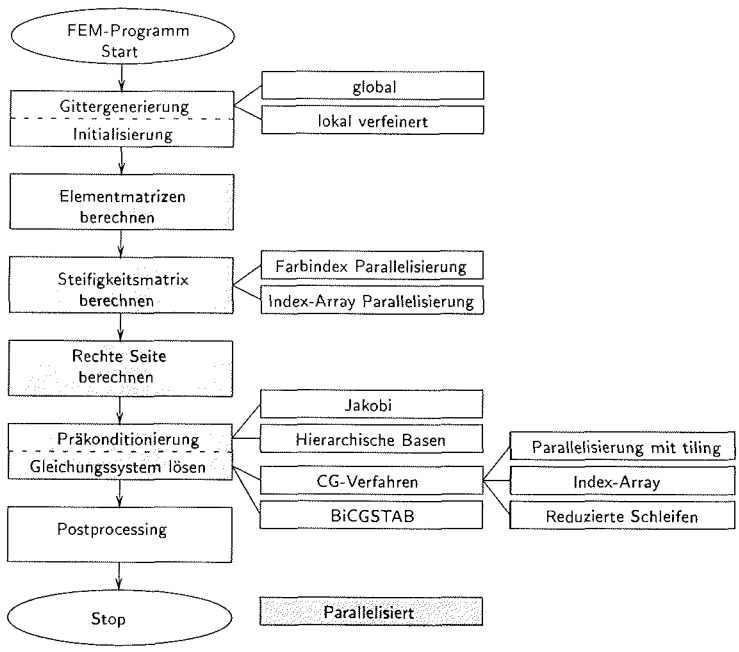

Abbildung 4.1: Flußdiagramm des implementierten FEM-Programms; die grau unterlegten Programmteile sind parallelisiert, alternative Methoden sind rechts neben jedem Programmteil angegeben

ausführliche Diskussion der Parametrisierung paralleler Programmleistung siehe [61]).

Abbildung 4.1 zeigt ein Flußdiagramm des implementierten Programms. In den folgenden Abschnitten werden verschiedene alternative Methoden für die einzelnen Programmsegmente besprochen. Die Alternativen sind jeweils rechts neben der Funktionseinheit angegeben.

4.2 Modellprobleme und Konvergenztests

Für alle weiteren Tests werden die folgenden drei Modellprobleme verwendet:

Problem 4.1 (Homogener Dirichlet-Rand) *Die Lösung der Poissongleichung (1.6) mit homogenen Dirichlet-Randbedingungen auf dem Gebiet* $\mathcal{G} = [0,1]^2$ *und folgender rechter Seite ist gesucht:*

$$f(x,y) = 2 \cdot (x + y - x^2 - y^2)$$

Die analytische Lösung ist gegeben durch die Funktion:

$$u(x,y) = x \cdot y \cdot (1-x) \cdot (1-y)$$

Bei den beiden anderen Modellproblemen handelt es sich um Eigenwertprobleme:

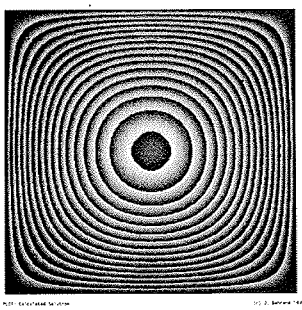

Abbildung 4.2: Modellproblem mit homogenen Dirichlet-Randbedingungen

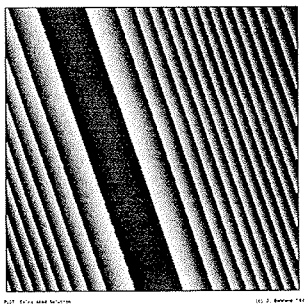

Abbildung 4.3: Modellproblem mit inhomogenen Dirichlet-Randbedingungen

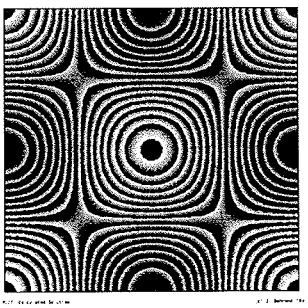

Abbildung 4.4: Modellproblem mit periodischen Randbedingungen

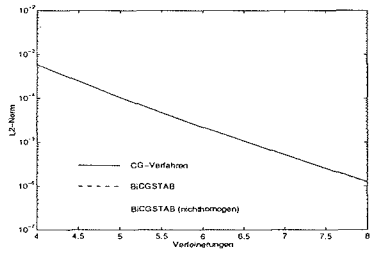

Abbildung 4.5: L^2-Norm des Fehlers für Modellprobleme mit homogenen und inhomogenen Dirichlet-Randbedingungen mit den Verfahren BiCGSTAG und CG berechnet

Problem 4.2 (Inhomogener Dirichlet-Rand) *Gesucht wird die Lösung der Poissongleichung (1.6) mit inhomgenen Dirichlet-Randbedingungen auf dem Gebiet* $\mathcal{G} = [0,1]^2$. *Die rechte Seite* f *und die Randbedingungen* g *sind gegeben durch die Funktionen:*

$$\begin{aligned} f(x,y) &= 10 \cdot \sin(3x+y) \\ g(x,y) &= \sin(3x+y) \end{aligned}$$

Die analytische Lösung ist:

$$u(x,y) = \sin(3x+y)$$

Problem 4.3 (Doppelt periodischer Rand) *Gesucht wird die Lösung der Poissongleichung (1.6) auf dem Gebiet* $\mathcal{G} = [0,1]^2$, *dessen Ränder in allen Richtungen periodisch fortgesetzt sind. Die rechte Seite ist durch die Funktion* f *gegeben:*

$$f(x,y) = 8\pi^2 \cos 2\pi x \cos 2\pi y$$

Die analytische Lösung lautet:

$$u(x,y) = \cos 2\pi x \cos 2\pi y$$

Die Lösungen der Modellprobleme sind in den Abbildungen 4.2, 4.3 und 4.4 dargestellt.

Für die Konvergenztests werden nun die drei Modellprobleme auf Gittern mit 4 bis 8 globalen Verfeinerungsstufen gelöst. Die L^2-Norm des Fehlers $e = u - u_h$, wobei u die exakte analytische Lösung und u_h die numerisch berechnete Lösung bezeichnet, wird in Abbildung 4.5 gegen die Verfeinerungsstufe aufgetragen. Die Kurve sollte bei logarithmischer Darstellung der L^2-Norm eine Gerade bilden. Beispielhaft sind die beiden ersten Modellprobleme mit dem Verfahren BiCGSTAB und das erste Modellproblem mit dem CG-Verfahren dargestellt.

4.3 Effizienz der parallelen Verfahren

In diesem Abschnitt werden verschiedene Aspekte der parallelen Verfahren untersucht. Zunächst ist die parallele Effizienz der Routinen zur Aufstellung der Steifigkeitsmatrix in den beiden Parallelversionen zu untersuchen. Im zweiten Unterabschnitt wird die Parallelisierung des Lösungsverfahrens untersucht. Verschiedene Parallelisierungsmethoden für das CG-Verfahren werden gegenübergestellt, BiCGSTAB wird mit CG verglichen und verschiedene Präkonditionierungen werden gezeigt. Die Varianten sind in Abbildung 4.1 angegeben. Für die Tests in diesem Abschnitt wird das Modellproblem mit homogenen Dirichlet-Randbedingungen verwendet.

Tabelle 4.1: Verschiedene Gitterkonfigurationen zur Messung der Skalierbarkeit, (g): global verfeinert, (l): zusätzlich lokal verfeinert

Test	Verfeinerungen	Knoten	Elemente (feinste Stufe)
A	7 (g)	8 321	16 384
B	7 (l)	16 261	32 152
C	8 (g)	33 025	65 536

4.3.1 Ergebnisse für die Matrixassemblierung

Die parallele Effizienz der Matrixaufstellung in zwei verschiedenen Varianten wird in diesem Unterabschnitt untersucht. Dazu werden Tests mit verschiedenen Gittern (zum Teil lokal verfeinert) durchgeführt. Um den Scaleup messen zu können, werden die Gitter für die Tests in diesem Unterabschnitt so gewählt, daß die Anzahl der Unbekannten (d.h. der Knoten) jeweils etwa verdoppelt wird. Die gewählten Gitterkonfigurationen sind in Tabelle 4.1 angegeben. Das verwendete lokal verfeinerte Gitter ist in Abb. 4.6 dargestellt. Zunächst werden in den Abbildungen 4.7 und 4.8 die Ergebnisse der Matrixassemblierung für die KSR-1 betrachtet. Die Gitterkonfigurationen B und C werden dargestellt. Die Farbindizierung zur Entkopplung der Datenabhängigkeit bei der Aufstellung der Diskretisierungsmatrix ist auf allen Gittern weniger leistungsfähig als die Index-Array-Methode. Es ist außerdem deutlich zu sehen, daß die Farbindizierungsmethode vom Gitter abhängig ist. Bei lokaler Verfeinerung ist die Effizienz schlechter als bei global verfeinertem Gitter. Dieses Verhalten erklärt sich mit der Zunahme der Synchronisationsschritte bei lokaler Verfeinerung. Mit lokal verfeinerten Gittern müssen, wie in Abschnitt 3.3 beschrieben, zusätzliche Farben eingeführt werden.

Die mit Hilfe von Index-Arrays parallelisierte Assemblierung der Matrix zeigt bessere Leistungen als die mit Farbindizes parallelisierte Variante. Abgesehen von einigen Schwankungen, die sich durch wechselnde Systemaktivität erklären lassen, erreicht die KSR-1 mit der Index-Array-Methode eine Effizienz von ca. 90% auf bis zu 26 Prozessoren. Das ist ein sehr guter Wert. Die Effizienzkurve verläuft darüberhinaus fast parallel zur x-Achse, so daß auch für größere Prozessorzahlen mit hoher Effizienz und Skalierbarkeit des Verfahrens gerechnet werden kann. Diese Aussage wird durch die Skalierungstests bestätigt. In Abbildung 4.8 ist der Scaleup der Matrixassemblierung für die beiden vorgestellten Parallelisierungsstrategien dargestellt. Beide Strategien skalieren gut. Die Index-Array-Methode ist wegen der Vorteile bei der absoluten Ausführungszeit und der parallelen Effizienz jedoch vorzuziehen.

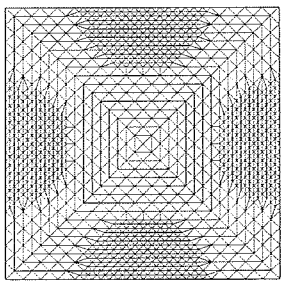

Abbildung 4.6: Lokal verfeinertes Gitter, wie es in den Tests zur Parallelisierung verwendet wird

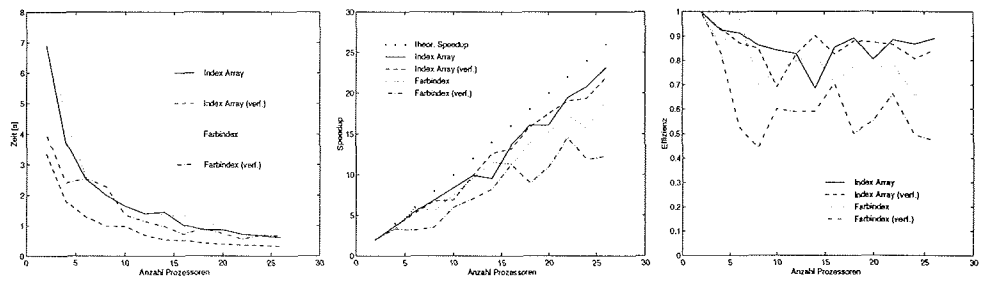

Abbildung 4.7: Ausführungszeit, Speedup und Effizienz für die Assemblierung der Steifigkeitsmatrix mit globaler und lokaler Gitterverfeinerung (Fälle B und C) und mit Farbindex- bzw. Index-Array-Parallelisierungsstrategie auf der KSR-1

Abbildungen 4.9 und 4.10 stellen die Ergebnisse der Paralelisierung der Matrixaufstellung auf einem Rechner mit gemeinsamem Hauptspeicher dar. Die Farbindex-Parallelisierung wurde ursprünglich für die Alliant FX/2800 entwickelt. Dieser Rechner besaß neben dem gemeinsamen Hauptspeicher einen schnellen Synchronisationsmechanismus. Damit konnten parallele Prozesse sehr schnell gestartet werden. Wegen der langsamen Datenkommunikation war die Effizienz der Parallelverarbeitung jedoch meist nicht so hoch wie auf der KSR-1.

Die Farbindizierung wurde auf bis zu vier Prozessoren getestet. Ausführungszeit, Speedup und Effizienz der Matrixassemblierung sind in Abbildung 4.9 dargestellt. Während die Einzelprozessorleistung der Alliant höher als die der KSR-1 ist, kann die KSR-1 mit deutlich besserer Effizienz insbesondere für die Index-Array-Methode aufwarten. Die parallele Leistung der Farbindizierung ist auf der Alliant nicht vom Gitter abhängig. Die Anzahl der Synchronisationen, die mit lokaler Verfeinerung zunimmt, beeinflußt die Effizienz wegen des gemeinsamen Hauptspeichers und der speziellen Hardwaresynchronisation nicht.

Die Skalierbarkeit der Matrixassemblierung ist auf der Alliant gut, wie Abbildung 4.10 veranschaulicht[1]. Im Vergleich mit der KSR-1 skaliert die Methode auf der Alliant etwas schlechter,

[1]Skalierbarkeit auf bis zu vier Prozessoren zu messen, ist sicher nur unter Vorbehalt zulässig. Die meisten Probleme mit Skalierbarkeit treten erst bei größeren Prozessorzahlen auf. Der Scaleup ist zur Vergleichbarkeit trotzdem angegeben.

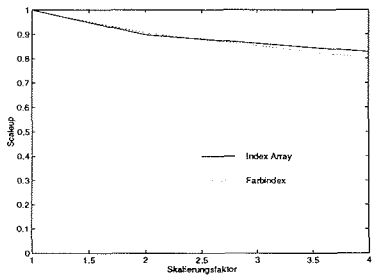

Abbildung 4.8: Scaleup der Matrixassemblierung für die Farbindizierung und die Index-Array-Methode auf der KSR-1

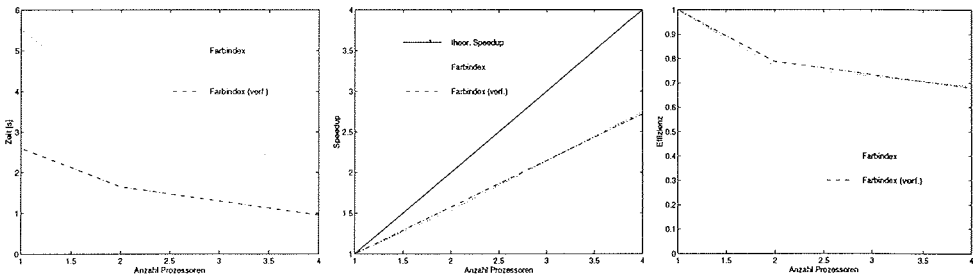

Abbildung 4.9: Wie Abb. 4.7, jedoch lediglich Farbindizierung auf der Alliant FX/2800

auf Computerarchitekturen mit gemeinsamem Speicher und schneller Synchronisation ist die Farbindizierung aber eine geeignete Parallelisierungsstrategie für die Matrixassemblierung.

4.3.2 Ergebnisse für das Lösungsverfahren

Parallelisierungsmethoden für das CG-Verfahren

Eine weitere Reihe von Tests betrachtet die parallelen Eigenschaften der Löser. Das CG-Verfahren ist im Abschnitt 3.2 in einer modifizierten Version mit nur zwei Schleifen vorgestellt worden. Die Auswirkung dieser Optimierung wird in der Abbildung 4.11 dargestellt. Auch die Auswirkung verschiedener KSR-Direktiven ist gezeigt: Wenn automatisch mittels der *tiling*-Direktive parallelisiert und damit die Datenverteilung vom Compiler vorgenommen wird, so ist die Ausführungszeit für kleine Prozessorzahlen kürzer als bei der manuellen Datenverteilung mit Hilfe von Index-Arrays. Bei zunehmender Prozessorzahl ist die Index-Array-Methode jedoch schneller als die automatische Parallelisierung. Vor allem die Effizienz und der Speedup sind für die Version mit tiling deutlich schlecher als für die handoptimierten Varianten. Die Version mit reduzierter Schleifenzahl verhält sich wie die Index-Array-Version. Sie ist aber in allen Fällen etwas schneller als die Version mit Index-Arrays. Es ist zu beachten, daß in die Zeit für das parallele Iterationsverfahren auch die parallelisierte Matrix-Vektor-Multiplikation eingeht.

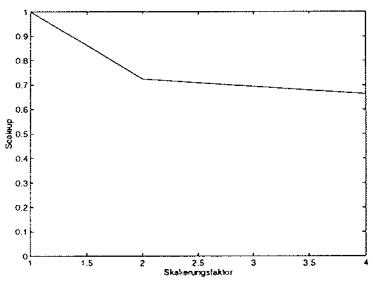

Abbildung 4.10: Scaleup der Matrixassemblierung für die Farbindizierung auf der Alliant FX/2800

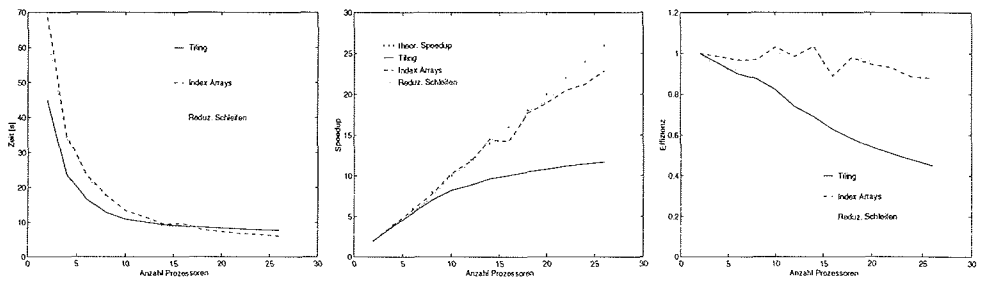

Abbildung 4.11: Ausführungszeit, Speedup und Effizienz für das parallelisierte CG-Verfahren, Parallelisierung mit tiling, Index-Arrays und reduzierten Schleifen

Präkonditionierung mit hierarchischen Basen und Jakobi-Präkonditionierung

Die Auswirkung der beiden im Abschnitt 2.5 vorgestellten Präkonditionierungen auf die parallele Leistung und die Ausführungszeit des CG-Verfahrens ist in Abbildung 4.12 dargestellt. Die Jakobi-Präkonditionierung ist für die getesteten Gittergrößen und Matrizen gegenüber der Methode mit hierarchischen Basen im Vorteil. Sie ist parallel effizienter und verbraucht wegen der geringen Operationszahl deutlich weniger Rechenzeit. Da die Matrix stark diagonaldominant und damit für Jakobi-Präkonditionierung geeignet ist, liegt sogar die Iterationszahl unter der mit hierarchischen Basen (Abb. 4.13). Es ist jedoch zu beobachten, daß die Iterationszahl für hierarchische Basen Präkonditionierung weniger schnell ansteigt. Theoretisch steigt die Konditionszahl der Diskretisierungsmatrix bezüglich der hierarchischen Basen ledig mit quadratischer Ordnung, während die Konditionszahl der nodalen Matrix exponentiell steigt. Damit ist die hierarchische Präkonditionierung für sehr große Probleme geeignet.

Iterative Lösungsverfahren CG und BiCGSTAB

Das Verfahren BiCGSTAB wurde gewählt, weil im adaptiven Semi-Lagrange-Verfahren für die Flachwassergleichungen wegen des Coriolisparameters ein unsymmetrisches System zu lösen ist. Für symmetrische Probleme ist das CG-Verfahren jedoch sehr viel effizienter. Dieses Verhalten

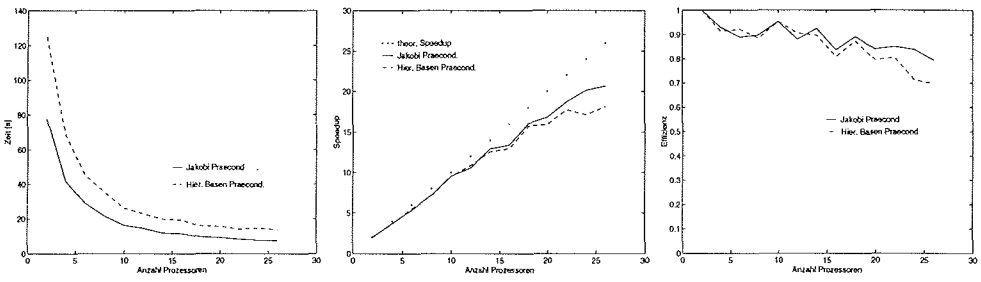

Abbildung 4.12: Ausführungszeit, Speedup und Effizienz für das parallelisierte CG-Verfahren mit Jakobi- bzw. hierarchischer Basen Präkonditionierung

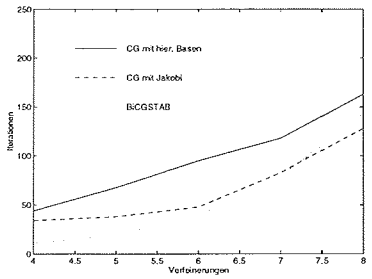

Abbildung 4.13: Anzahl der Iterationen für das CG-Verfahren mit Jakobi- und hierarchischer Basen Präkonditionierung und für BiCGSTAB mit Jakobi-Präkonditionierung

ist in Abbildung 4.14 erkennbar: Das CG-Verfahren benötigt weniger als halb soviel Zeit und es ist effizienter.

Der Rechenzeitgewinn ist wegen der geringeren Anzahl Operationen zu erwarten. BiCGSTAB benötigt zwei Matrixmultiplikationen und vier innere Produkte im Gegensatz zu einer Matrixmultiplikation und zwei inneren Produkten im CG-Verfahren. BiCGSTAB enthält daher mehr Schleifen, deren Synchronisationsaufwand zur schlechteren parallelen Effizienz führt. Immerhin beträgt der Speedup des Verfahrens für unsymmetrische Systeme ohne weitreichende Veränderungen des Codes etwa 20 auf 26 Prozessoren, was einer Effizienz von knapp 80% entspricht.

4.4 Zusammenfassung

In diesem Teil wurde die Finite-Elemente-Methode anhand der Poissongleichung mit verschiedenen Randbedingungen eingeführt. Einige wichtige Aspekte der Implementierung wurden vorgestellt. Neue Datenstrukturen für lokal verfeinerte Gitter wurden definiert, ein Gittergenerator entwickelt, der in mehreren – zum Teil virtuellen – Schritten ein gegebenes Gitter verfeinert, weitere Gitterroutinen und speziell angepaßte Lösungsverfahren wurden implementiert.

Wesentliches Ziel war die Parallelisierung des Verfahrens. Die Parallelisierung der Aufstellung

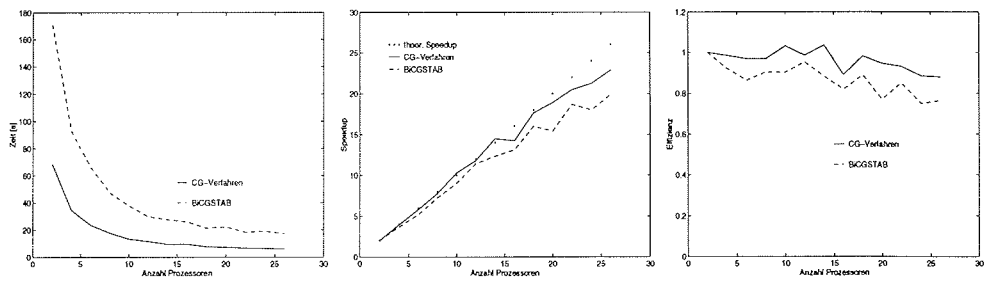

Abbildung 4.14: Ausführungszeit, Speedup und Effizienz für das CG-Verfahren und für das Verfahren BiCGSTAB

der Steifigkeitsmatrix stellt dabei eine besondere Schwierigkeit dar. Zwei Strategien für die Parallelisierung der Matrixassemblierung wurden vorgestellt. Die Strategie mit Hilfe von Farbklassen bedient sich eines Algorithmus zur Einfärbung, der auf tabellierten Farbwerten basiert und wenig Rechenaufwand erfordert. Wegen der relativ hohen Anzahl von Farben, die bei dieser Einfärbung entstehen, ist die parallelisierte Matrixaufstellung auf Systemen mit verteiltem Speicher nicht so effizient wie die zweite Strategie.

Diese zweite Strategie verwendet Index-Arrays, um die Daten in den lokalen Speichern zu fixieren und in einer determinierten Weise auf die Daten zuzugreifen. Damit werden Datenabhängigkeiten aufgelöst und Datenlokalität wird erhalten. Effizienz von bis zu 90% auf 26 Prozessoren der KSR-1 zeigt die Überlegenheit dieser Strategie.

Die Methodik der Index-Arrays wird auf die Parallelisierung des Lösungsverfahrens ebenfalls angewendet und führt zu geringem Kommunikationsaufwand innerhalb des gesamten Programms. Alle Operationen an Gitterpunkten werden auf den Prozessoren fixiert. Schreibzugriffe sind dann nicht mehr notwendig. Parallele Effizienz von 70% und mehr auf 26 Prozessoren sind so erreichbar, ohne daß weitreichende Eingriffe in die Algorithmen vorgenommen werden müssen.

Verschiedene Präkonditionierungen wurden miteinander verglichen. Die einfache Jakobi-Präkonditionierung erweist sich für die getesteten Problemgrößen bei Parallelverarbeitung als effizient und schnell.

Eine Variante des CG-Verfahrens mit nur zwei Schleifenblöcken wurde vorgestellt. Sie zeigt auf Computerarchitekturen mit verhältnismäßig langer Startupzeit für parallele Prozesse bessere Leistungen als der herkömmliche Algorithmus.

Das implementierte Programm, welches im Teil III zur Anwendung kommt, kann periodische, Dirichlet- und Neumann-Randbedingungen behandeln. Ausgehend von einem groben vordefinierten Gitter kann eine lokal verfeinerte hierarchische Triangulierung aus Dreieckselementen automatisch erzeugt werden. Ohne Graphikroutinen umfaßt die Implementierung etwa 8000 Zeilen Fortran Code.

Teil II

Die Semi-Lagrange-Methode (SLM) für die Advektionsgleichung

Kapitel 5

Einführung in die Semi-Lagrange-Methode

5.1 Übersicht

Die Diskretisierung von zeitabhängigen Strömungsgleichungen wird häufig in zwei Schritten vorgenommen. Zunächst wird die Zeitabhängigkeit diskretisiert. Dabei wird die kontinuierliche Zeitachse in Intervalle (*Zeitschritte*) eingeteilt und ein geeigneter Differenzenquotient zur Approximation des Differentials gebildet. Zu lösen bleibt dann in jedem Zeitschritt ein stationäres (d.h. zeitunabhängiges) Teilproblem, welches wiederum mit einem geeigneten Verfahren (z.B. FEM) diskretisiert wird.

Strömungsgleichungen lassen sich in der Eulerschen oder der Lagrangeschen Darstellung formulieren. In der numerischen Behandlung von Strömungsproblemen herrscht die Eulersche Betrachtungsweise vor. Die Strömungsgleichungen werden dabei aus der Sicht eines festen geographischen Punktes formuliert. Dabei wird das Verhalten von Ein- und Ausströmung infinitesimaler Volumina einer Flüssigkeit beschrieben. Gleichungen der Eulerschen Form werden diskretisiert, indem die Volumina als diskretes Gitter in die Flüssigkeit gelegt werden und wiederum die Ein- und Ausströmung jedes diskreten Volumens beschrieben wird [105].

Ein Vorteil dieser Formulierung ist, daß mit regulären Gittern leicht eine diskrete Form der Gleichungen gefunden werden kann, die sich gut für die Berechnung mit Computern eignet. Die Vektorisierung bzw. Parallelisierung von solchen Verfahren ist in der Regel einfach und effizient durchzuführen [65]. Ein Nachteil der Methode ist, daß die Stabilität der Diskretisierung vom Verhältnis von Strömungsgeschwindigkeit und Zeitschritt zur Gitterweite abhängt. Vereinfacht ausgedrückt kann das Verfahren nur dann funktionieren, wenn innerhalb eines Zeitschrittes die Flüssigkeit nicht mehr als eine Gitterweite zurücklegen kann. Diese Abhängigkeit wird durch das Courant-Friedrichs-Levy-Kriterium (CFL) beschrieben.

Als Konsequenz daraus muß die Zeitschrittlänge insbesondere bei hoch aufgelösten Modellen sehr viel kürzer gewählt werden als es der zeitliche Abschneidefehler eigentlich erlauben würde, um die Stabilität des Verfahrens zu erhalten. Das führt zu unnötigem Rechenaufwand.

Der Eulerschen Betrachtungsweise steht die Lagrangesche Formulierung gegenüber. Dabei wird die Strömung einer Flüssigkeit durch die Wege der Flüssigkeitspartikel (die *Trajektorien*) beschrieben [17]. Mit unendlich vielen Partikeln und entsprechend vielen Trajektorien läßt sich ein vollständiges Bild des Flusses ermitteln. Diskretisiert wird diese Formulierung, indem nur eine endliche Anzahl Trajektorien verfolgt wird. Wenn die zugehörigen Partikel gleichmäßig über das Gebiet verteilt sind, läßt sich der Fluß hinreichend vollständig beschreiben. Es ist bei dieser Formulierung aber nicht sichergestellt, daß die Partikel über längere Zeit hinweg gleichmäßig

verteilt bleiben. Daher kann die Vollständigkeit der Beschreibung verloren gehen.

Lagrangesche Verfahren sind unabhängig von der Zeitschrittlänge stabil. Natürlich verliert man bei sehr langen Zeitschritten Information, der zeitliche Diskretisierungsfehler wird groß. Aber das Verfahren bleibt stabil. Problematisch für die Implementierung solcher Verfahren ist die möglicherweise starke Irregularität der Daten [46].

Einen Mittelweg stellt die Semi-Lagrange-Methode (SLM) dar. Die Formulierung und Berechnung der Gleichungen wird zwar in der Lagrangeschen Form vorgenommen, aber es werden in jedem Zeitschritt neue gleichmäßig verteilte Partikel gewählt. Einerseits sind die Datenstrukturen dadurch regulär und geeignet für die effiziente Benutzung auf Computern, andererseits ist die Zeitschrittlänge unabhängig von der räumlichen Auflösung [127].

Dazu wird ein Gitter definiert, das eine geeignete Verteilung der Flüssigkeitspartikel gewährleistet. In jedem Zeitschritt werden Trajektorienstücke berechnet, die an den Gitterpunkten enden. Die Advektion wird dann mit Hilfe einer Interpolation der stromaufwärts gelegenen Werte des Flusses (die im allgemeinen zwischen Gitterpunkten ermittelt werden müssen) berechnet.

Es gibt neben der Semi-Lagrange-Methode noch andere numerische Verfahren, mit denen die Advektion einer Flüssigkeit berechnet werden kann und die zusätzlich unabhängig von der Zeitschrittlänge stabil sind. Dazu gehören das *Taylor-Galerkin* Verfahren [106] oder Verfahren mit *flußkorregiertem Transport* (FCT) [148]. Die Semi-Lagrange-Methode erscheint aber nicht nur wegen ihrer günstigen Stabilitätseigenschaften vorteilhaft, sondern auch wegen der guten Parallelisierbarkeit.

In den folgenden Abschnitten wird zunächst das einfache Problem der passiven Advektion beschrieben, das als Beispielproblem zur Einführung der Semi-Lagrange-Methode (SLM) dient. Die SLM wird dann in zwei Varianten, dem *Drei-Schritt-Verfahren* und dem *Zwei-Schritt-Verfahren* definiert. Zwei Komponenten der SLM, die Trajektorienberechnung und die Interpolation werden etwas ausführlicher diskutiert.

5.2 Das Advektionsproblem

Die SLM kann anhand der eindimensionalen Gleichung für passive Advektion veranschaulicht werden. Die Gleichung lautet:

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + \frac{dx}{dt}\frac{\partial u}{\partial x} = 0, \tag{5.1}$$

wobei $\frac{dx}{dt} = a(x,t)$ und eine Anfangsbedingung $u(x, t = 0) = u^0(x)$ gegeben sei. Die Gleichung (5.1) drückt aus, daß eine skalare Größe u entlang eines Stromes (bzw. einer Trajektorie) konstant ist. Die Form der Trajektorie wird dabei von der Funktion a (dem Wind) bestimmt.

Anschaulich wird die Aussage, wenn man sich eine fließende Flüssigkeit vorstellt, in die ein Tropfen Tinte gegeben wird. Die Eigenschaft der skalaren Größe u ist nun die Farbe. Die Gleichung besagt, daß (unter Vernachlässigung der Reibung) der Tintenklecks mit der Strömung so transportiert wird, daß jeder Partikel, welcher mit Tinte behaftet ist, diese auch nicht mehr verliert (siehe [113]).

Für Dimensionen größer eins schreibt sich (5.1):

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + \mathbf{a} \cdot \nabla u = 0. \tag{5.2}$$

Dabei sei wieder $\mathbf{a}(\mathbf{x}, t)$ gegeben, $\mathbf{x} = (x_1, x_2, \ldots)$ ist der Ortsvektor, t die Zeit.

Die Advektionsgleichung ist ein einfaches „Spielproblem", das jedoch schon einige wesentliche Elemente für kompliziertere Probleme enthält. Ein Semi-Lagrange-Verfahren, wie es in den

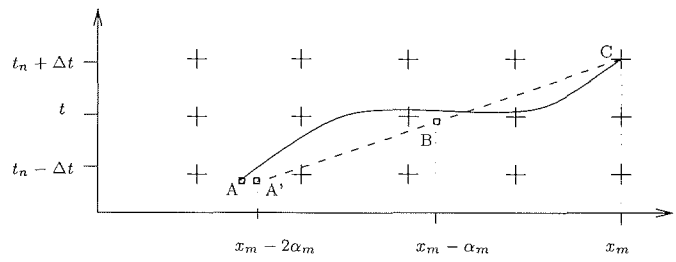

Abbildung 5.1: Drei-Schritt-SLM für die eindimensionale passive Advektion

nächsten Abschnitten vorgestellt wird, muß die wichtigsten Komponenten (Interpolation, Trajektorienschätzung) enthalten. Da die Advektionsgleichung eine konstante skalare Größe beschreibt, läßt sich die Rechengenauigkeit des Verfahrens gut überprüfen.

5.3 Semi-Lagrange-Form der Advektionsgleichung

5.3.1 Das Drei-Schritt-Verfahren

Zur Einführung der SLM wird nun die eindimensionale Advektionsgleichung (5.1) betrachtet. Es wird zunächst die sogenannte Drei-Schritt-Methode vorgestellt. Die Darstellung lehnt sich an die Notationen in [127] an.

Abbildung 5.1 veranschaulicht die Situation: An Gitterpunkten zu den Zeitpunkten t_n und $t_n - \Delta t$ seien die Werte von u bekannt. Ziel ist die Berechnung der Gitterwerte von u zur Zeit $t_n + \Delta t$. Die durchgezogene Linie markiere die exakte Trajektorie eines Partikels, der zum Zeitpunkt $t_n + \Delta t$ den Punkt x_m erreichen wird. Um nun den Wert von u am Punkt x_m zur Zeit $t_n + \Delta t$ zu berechnen, reicht es, den Wert von u am Punkt A zur Zeit $t_n - \Delta t$ zu kennen, denn entlang der Trajektorie bleibt der Wert konstant.

Im allgemeinen wird es nicht möglich sein, den Punkt A exakt zu bestimmen. Daher wird die exakte Trajektorie durch eine angenäherte lineare Trajektorie approximiert (gestrichelte Linie). Berechnet man nun den Wert von u am Anfangspunkt der genäherten Trajektorie, A'= $(x_m - 2\alpha_m, t_n - \Delta t)$, so kann dieser Wert als neuer Wert von u am Punkt x_m zur Zeit $t_n + \Delta t$ genommen werden. Die diskretisierte Form der Gleichung (5.1) lautet damit:

$$\frac{u(x_m, t_n + \Delta t) - u(x_m - 2\alpha_m, t_n - \Delta t)}{2\Delta t} = 0. \tag{5.3}$$

Um Punkt A' bestimmen zu können, wird der Wert von α_m benötigt. Dazu betrachte (5.1): Der Wind a ist gleich der inversen Steigung der Trajektorie $\frac{dx}{dt}$. Diskretisiere dx mit Δx und dt mit Δt. Wenn $\Delta x = x_m - (x_m - \alpha_m) = \alpha_m$ und $\Delta t = t_{n+1} - t_n$ gewählt wird, erhält man eine $\mathcal{O}(\Delta t)$-Näherung

$$\alpha_m = \Delta t \; a(x_m, t_n) \tag{5.4}$$

In [132] stellen Staniforth und Temperton fest, daß diese Näherung zwar theoretisch ausreichte, der zeitliche Abschneidefehler dann aber so groß wäre, daß die erhöhte Stabilität der SLM durch die große Ungenauigkeit kompensiert würde.

Daher wird eine $\mathcal{O}(\Delta t^2)$-Näherung für die Berechnung der α_m verwendet. Dabei muß $\Delta x = x_m - (x_m - 2\alpha_m) = 2\alpha_m$ und $2\Delta t = t_{n+1} - t_{n-1}$ gewählt werden. Damit gilt:

$$\alpha_m = \Delta t \; a(x_m - \alpha_m, t_n). \tag{5.5}$$

Diese Gleichung läßt sich iterativ lösen mit Hilfe von

$$\alpha_m^{(k+1)} = \Delta t \; a(x_m - \alpha_m^{(k)}, t_n), \tag{5.6}$$

wobei $\alpha_m^{(0)}$, eine geeignete Anfangsnäherung, vorgegeben wird; a muß dabei zwischen den Gitterpunkten berechnet (d.h. interpoliert) werden können.

Mit den in (5.3) und (5.6) gegebenen Formeln läßt sich das Semi-Lagrange-Verfahren für die Advektionsgleichung formulieren:

Algorithmus 5.1 (Drei-Schritt-Semi-Lagrange-Verfahren)

1. *Löse (5.6) für alle α_m zu Gitterpunkten x_m;*
2. *Berechne u an den stromaufwärts gelegenen Punkten $x_m - 2\alpha_m$ zur Zeit $t_n - \Delta t$ mit Hilfe einer geeigneten Interpolation;*
3. *Berechne u an den Gitterpunkten x_m zur Zeit $t_n + \Delta t$ mit Hilfe von (5.3).*

Beim Algorithmus 5.1 ist die wesentliche Schwierigkeit, eine genaue und billige Interpolation für die Werte zwischen Gitterpunkten zu finden.

5.3.2 Das Zwei-Schritt-Verfahren

Das Verfahren 5.1 aus dem letzten Abschnitt läßt sich weiter verbessern: Die Gleichungen (5.3) und (5.6) lassen sich vereinfacht schreiben:

$$\frac{u^+ - u^-}{2\Delta t} = 0 \tag{5.7}$$

$$\alpha = \Delta t \; a^0 \tag{5.8}$$

Dabei soll das hochgesetzte $+$ die Zeit $t_n + \Delta t$, das $-$ die Zeit $t_n - \Delta t$ und die hochgesetzte 0 die Zeit t_n symbolisieren.

Der wesentliche Nachteil dieser Methode besteht darin, daß der Zeitschritt relativ kurz gewählt werden muß, um den zeitlichen Abschneidefehler nicht zu groß werden zu lassen. Daher wird das Zwei-Schritt-Verfahren eingeführt, das bei gleicher Ordnung des zeitlichen Fehlers die doppelte Zeitschrittlänge erlaubt.

Da a unabhängig von u gegeben ist, werden für die Berechnung von u^+ nur Werte von u zur Zeit $t_n - \Delta t$ benötigt (u^-). Für die Berechnung von u zur Zeit $t_n + 3\Delta t$ werden die Werte von u zur Zeit $t_n + \Delta t$ benötigt und so fort. Es entstehen zwei ungekoppelte Integrationen für u, die entweder nur von den geraden oder nur von den ungeraden Zeitschritten abhängen. Eine der beiden Integrationsreihen ist daher überflüssig. Lediglich die Berechnung von α wird an den jeweils dazwischenliegenden Zeitpunkten vorgenommen.

Durch Umbenennung $t_n - \Delta t \to t_n$ und $t_n \to t_n + \frac{\Delta t}{2}$ für die Berechnung von α wird das Zwei-Schritt-Verfahren definiert. Die Zeitschrittlänge ist jetzt bei gleichem zeitlichen Abschneidefehler doppelt so lang wie beim Drei-Schritt-Verfahren. Der Algorithmus lautet:

Algorithmus 5.2 (Zwei-Schritt-Semi-Lagrange-Verfahren)

1. *Löse (5.6) für alle α_m zu Gitterpunkten x_m, jedoch zur Zeit $t_n + \frac{\Delta t}{2}$;*
2. *Berechne u an den stromaufwärts gelegenen Punkten $x_m - 2\alpha_m$ zur Zeit t_n mit Hilfe einer geeigneten Interpolation;*
3. *Berechne u an den Gitterpunkten x_m zur Zeit $t_n + \Delta t$ in Analogie zu Formel (5.3).*

Im folgenden wird die zweidimensionale Form der Advektionsgleichung verwendet. Das Semi-Lagrange-Verfahren wird dabei analog formuliert: Für die eindimensionalen Größen x_m, a und α_m werden nun zweidimensionale Felder $\mathbf{x}_m, \mathbf{a}$ und $\boldsymbol{\alpha}_m$ eingesetzt.

5.4 Berechnung der Trajektorien

Der erste Schritt in den Algorithmen 5.1 und 5.2 besteht jeweils aus der Berechnung der Trajektorien $\boldsymbol{\alpha}$. Für die Advektionsgleichung 5.1 ist der Wind $\mathbf{a}$ gegeben und läßt sich an jedem beliebigen Punkt zu jeder beliebigen Zeit unabhängig von u berechnen.

Die Welt wäre langweilig, wenn sie einfach wäre und die Situation ist meist nicht so einfach, wie im letzten Abschnitt. In der Regel ist der Wind $\mathbf{a}$ nicht unabhängig von der Größe u gegeben. Die Impulsgleichungen der Strömungsdynamik, und damit insbesondere die Gleichungen in Strömungsmodellen für Ozean oder Atmosphäre, enthalten einen Term $u \cdot \nabla u$ (statt $\mathbf{a} \cdot \nabla u$).

Die Berechnung der $\boldsymbol{\alpha}_m$ ist damit nicht so einfach wie im letzten Abschnitt angegeben. Für das Drei-Schritt-Verfahren ist die Situation noch klar, weil u zum Zeitpunkt t_n an der Stelle $\mathbf{x}_m - \boldsymbol{\alpha}_m$ mit Hilfe einer geeigneten Interpolation berechnet werden kann. Beim Zwei-Schritt-Verfahren hingegen wird die Kenntnis des Windes zur Zeit $t_n + \frac{\Delta t}{2}$ vorausgesetzt. Im Falle von Selbstadvektion ist u aber nur zum Zeitpunkt t_n bekannt.

In Anlehnung an (5.6) lautet die Gleichung für die iterative Berechnung der Trajektorien $\boldsymbol{\alpha}_m$:

$$\boldsymbol{\alpha}_m^{(k+1)} = \Delta t \; u(\mathbf{x}_m - \frac{\boldsymbol{\alpha}_m^{(k)}}{2}, t_n + \frac{\Delta t}{2}). \tag{5.9}$$

Da u zur Zeit $t_n + \frac{\Delta t}{2}$ nicht bekannt ist, muß ein Extrapolationsschema gefunden werden, das die Trajektoriengleichung unter Verwendung bekannter Gitterwerte zu früheren Zeiten mit der Ordnung $\mathcal{O}(\Delta t^2)$ auflöst.

Ein mögliches Zwei-Schritt-Extrapolationsschema, das Werte zu Zeiten t und $t - \Delta t$ einbezieht, ist gegeben durch:

$$u(\mathbf{x}, t + \frac{\Delta t}{2}) = \frac{1}{2}\,[\,3\,u(\mathbf{x}, t) - u(\mathbf{x}, t - \Delta t)]. \tag{5.10}$$

Für die Berechnung von $\boldsymbol{\alpha}$ nach der Formel (5.9) wird dann noch eine Interpolation zur Bewertung von u an Punkten $\mathbf{x} - \frac{\boldsymbol{\alpha}}{2}$ benötigt. Es kann gezeigt werden, daß die Interpolationsordnung für die Berechnung der Trajektorienstücke um eins kleiner sein kann, als die Ordnung für die Interpolation der stromaufwärts gelegenen Werte [90, 89]. In der Praxis treten keine wesentlichen Unterschiede bei der Verwendung linearer oder kubischer Interpolation auf [127]. Eine weitere Möglichkeit der Extrapolation ist das folgende Drei-Schritt-Schema:

$$u(\mathbf{x}, t + \frac{\Delta t}{2}) = \frac{1}{8}\,[\,15\,u(\mathbf{x}, t) - 10\,u(\mathbf{x}, t - \Delta t) + 3\,u(\mathbf{x}, t - 2\,\Delta t)]. \tag{5.11}$$

Für die Berechnung der (extrapolierten) Werte von u zur Zeit $t+\frac{\Delta t}{2}$ werden also die Werte zu Zeiten t und $t-\Delta t$ benötigt. Die Einführung des Zwei-Schritt-Verfahrens hatte dagegen gerade den Effekt, daß die überflüssig gewordene Zeit $t-\Delta t$ entfiel. Der wesentliche Vorteil des Zwei-Schritt-Verfahrens, die Verdoppelung des Zeitschrittes bei Erhaltung des zeitlichen Abschneidefehlers, ist aber nach wie vor gültig. Auch wenn in die Extrapolation wieder mehrere alte Zeitschrittwerte eingehen, bleibt die Diskretisierung der Advektionsgleichung ein Zwei-Schritt-Verfahren.

5.5 Interpolation der stromaufwärts gelegenen Werte

Im zweiten Schritt in den Algorithmen des Abschnittes 5.3 werden die Werte von u jeweils an den stromaufwärts gelegenen Punkten $\mathbf{x}-\boldsymbol{\alpha}$ bzw. $\mathbf{x}-2\boldsymbol{\alpha}$ interpoliert. Dieser Interpolation kommt eine besondere Bedeutung zu. Die Wahl der Interpolation bestimmt die Genauigkeit und Effizienz des Verfahrens.

5.5.1 Bedeutung der Interpolationsordnung

Staniforth und Côté empfehlen in [127] wenigstens Interpolation vierter Ordnung zu verwenden. Kubische Spline Interpolation erscheint den Autoren als guter Kompromiß zwischen Rechenaufwand und Genauigkeit.

Die Dämpfung dieser Interpolation ist gering, sie selektiert Skalen, das heißt sie dämpft vor allem die hochfrequenten Anteile (kleine Skalen). Demgegenüber ist die Dämpfung von linearer Interpolation zu hoch.

Für die Interpolation zur Trajektorienberechnung (Schritt 1. in den Algorithmen 5.1 bzw. 5.2) spielt die Ordnung der Interpolation eine untergeordnete Rolle. Die oben genannten Autoren berichten von guten Ergebnissen bei linearer Interpolation. In [89] wird gezeigt, daß die Interpolationsordnung im ersten Schritt um eins kleiner sein sollte als im Schritt 2. In der Praxis reicht lineare Interpolation. Lineare Interpolation für die ersten Schritten und quadratische Interpolation beim letzten Iterationsschritt zu verwenden, scheint eine sinnvolle Lösung zu sein.

Eine kubische Interpolation zwischen den Gitterpunkten ist auch im Kontext der FEM mit linearen Basisfunktionen sinnvoll. Die Idee, die Interpolation zu sparen und stattdessen die Finite-Elemente-Lösung (die auch zwischen Gitterpunkten vorliegt) zu verwenden, hat sich als nicht sinnvoll erwiesen. Wegen der Superkonvergenzeigenschaft der FEM (siehe [139]), ist die Verwendung einer Interpolation höherer Ordnung sinnvoll, weil die Interpolation die Knotenwerte verwendet. An den Knotenwerten liegt die FEM-Lösung aber in höherer Genauigkeit vor als die Gesamtordnung der FEM besagt. Andererseits ist es nicht sinnvoll eine FEM höherer Ordnung zu verwenden (so daß die Zwischengitterwerte mit höherer Ordnung vorliegen), weil der numerische Aufwand nicht zu rechtfertigen ist.

5.5.2 Quasi-monotone Interpolation

Kubische Interpolation, wie sie im vorigen Absatz vorgeschlagen wurde, neigt in der Nähe von Unstetigkeiten zum Über- und Unterschwingen. Das bedeutet insbesondere, daß Monotonieeigenschaften der zu interpolierenden Funktion nicht erhalten werden.

In Abbildung 5.2 ist ein Schnitt durch das im nächsten Kapitel genauer beschriebene Modellproblem des geschlitzten Zylinders dargestellt. Ein kreisförmiger Wind treibt das skalare Feld u an, das die Form eines geschlitzten Zylinders aufweist. Nach fünf Umdrehungen ist das Über- und Unterschwingen der kubischen Interpolation an den Kanten des Zylinders gut auszumachen.

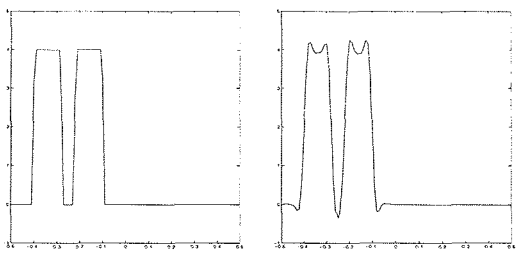

Abbildung 5.2: Modellproblem „geschlitzter Zylinder“ bei der Initialisierung und nach fünf Umdrehungen mit kubischer Spline Interpolation (Schnitt entlang der x-Achse)

Die exakte Lösung sähe so aus wie die Initialisierung, weil weder Reibung noch andere Quellen oder Senken im Modell vorhanden sind und damit sowohl Form als auch Volumen des Zylinders exakt erhalten werden müßten.

Dieses Über- oder Unterschwingen ist im allgemeinen nicht erwünscht. Zur Vermeidung dieses Verhaltens gibt es verschiedene Verfahren, bei denen die Interpolation so modifiziert wird, daß sie die Monotonie der Funktion erhält. Zwei Methoden sollen hier vorgestellt werden:

Das sogenannte *clipping* schneidet mögliches Überschwingen einfach ab. Sei (x, y) der zu Interpolierende Punkt in zwei Dimensionen. Ein (dreieckiges) Element mit den Eckpunkten (x_i, y_i), $i = 1, \ldots, 3$ enthalte den Punkt (x, y). Die Werte des skalaren Feldes u sind an diesen Eckpunkten gegeben, gesucht ist der interpolierte Wert $u(x, y)$. Die Berechnung erfolgt in zwei Schritten:

Algorithmus 5.3 (Clipping)

1. *Berechne die kubische Spline Interpolation $u_{kub}(x, y)$ von u an (x, y).*
2. *Setze*

$$u(x,y) = \begin{cases} u_{max} & , falls\ u_{max} < u_{kub}(x,y), \\ u_{min} & , falls\ u_{min} > u_{kub}(x,y), \\ u_{kub}(x,y) & , sonst. \end{cases}$$

Dabei ist $u_{max} = \max_{i=1,\ldots,3}(u(x_i, y_i))$ und u_{min} analog das Minimum.

Die *quasi-monotone Semi-Lagrange-Interpolation* (QMSL) ist ausführlich in [25] beschrieben. Die Idee ist einfach: Berechne eine Interpolation hoher Ordnung (die nicht monotonieerhaltend ist) und eine Interpolation niederer Ordnung (die monotonieerhaltend ist) und bilde ein gewichtetes Mittel der beiden Interpolationen, welches die Monotonie erhält und möglichst genau ist (d.h. der Anteil der Interpolation hoher Ordnung ist möglichst hoch). Der interpolierte Wert $u(x, y)$ wird dargestellt:

$$u(x,y) = c_{x,y} u_{high}(x,y) + (1 - c_{x,y}) u_{low}(x,y), \tag{5.12}$$

wobei u_{low} die Interpolation niederer Ordnung repräsentiert und u_{high} die Interpolation hoher Ordnung; $0 \leq c_{x,y} \leq 1$ ist ein Gewichtungsfaktor.

Der QMSL-Algorithmus läßt sich in vier Schritten angeben:

Algorithmus 5.4 (QMSL)

1. *Berechne eine Interpolation u_{low} niederer Ordnung (z.B. lineare Interpolation).*
2. *Berechne eine Interpolation u_{high} hoher Ordnung (z.B. kubische Spline Interpolation).*
3. *Berechne einen Wichtungsfaktor $0 \leq c_{x,y} \leq 1$ mit Hilfe der folgenden Schritte:*
   (a) *Berechne u_{min} und u_{max} wie in Algorithmus 5.3*
   (b) *Setze:*

$$\begin{aligned} q_{max} &= u_{max} - u_{low}, \\ q_{min} &= u_{min} - u_{low}, \\ p &= u_{high} - u_{low}. \end{aligned} \tag{5.13}$$

   (c) *Setze:*

$$c_{x,y} = \begin{cases} min\left(1, \frac{q_{max}}{p}\right) & , falls\ p > 0, \\ min\left(1, \frac{q_{min}}{p}\right) & , falls\ p < 0, \\ 0 & , falls\ p = 0. \end{cases} \tag{5.14}$$

4. *Berechne $u(x,y)$ wie in Gleichung (5.12).*

Über- oder Unterschwingen der Interpolation hoher Ordnung geschieht nur an Stellen hoher Rauhheit. Die Güte der Interpolation ist in jedem Fall durch den Grad der Glattheit der interpolierten Funktion begrenzt. An Stellen geringer Glattheit kann auch ein Verfahren hoher Ordnung keine besonders genauen Approximationen liefern. Es ist daher gerechtfertigt, gerade an diesen Unstetigkeiten eine (monotone) Interpolation niedriger Ordnung zu verwenden (siehe [25]).

5.5.3 Quasi-konservative Interpolation

In Klimamodellen oder Ozeanmodellen sind die Zeiten, über die ein Modellauf integriert werden muß, sehr lang. Daher ist folgende Eigenschaft der Interpolation sehr wünschenswert: Die Erhaltung bestimmter physikalischer Größen.

In vielen heute genutzten Modellen für Ozeanzirkulationen oder Klimamodellierung ist ein erheblicher mathematischer Aufwand in die Entwicklung masse- oder energieerhaltender Operatoren gesteckt worden. Bis vor kurzem war eine wesentliche Kritik an den Semi-Lagrange-Verfahren, daß sie keine nachgewiesene Erhaltungseigenschaften besaßen. In [107] schlägt Priestley eine masseerhaltende Interpolation für eine SLM für die Advektionsgleichung vor.

Ausgehend von den Gewichtungsfaktoren $c_{x,y}$ aus dem QMSL Verfahren 5.4 werden nun modifizierte Faktoren $d_{x,y}$ gewählt, so daß das Integral des skalaren Feldes u über dem Rechengebiet $\mathcal{G}$ konstant bleibt (das entspricht der Formulierung der Masse- oder Energieerhaltung):

$$\int_{\mathcal{G}} u(x,y)\, d\mathcal{G} = \int_{\mathcal{G}} u^0(x,y)\, d\mathcal{G} = C. \tag{5.15}$$

Um also eine Interpolation möglichst hoher Ordnung zu erhalten, die außerdem noch monotonieerhaltend und masseerhaltend ist, stellt sich das folgende Problem:

Problem 5.1 (Wichtungsfaktoren für masseerhaltende Interpolation)
Maximiere die $d_{x,y}$ so daß folgende Gleichung erfüllt ist:

$$\int_{\mathcal{G}} d_{x,y}(u_{high}(x,y) - u_{low}(x,y))d\mathcal{G} = C - \int_{\mathcal{G}} u_{low}(x,y)d\mathcal{G} = \hat{C}. \tag{5.16}$$

Priestley gibt einen Algorithmus für die diskreten Integrale an. Dabei wird das Integral angenähert durch die Summe

$$\int_{\mathcal{G}} u(x,y)d\mathcal{G} \approx \sum_{x_i,y_i} u(x_i,y_i)\, S_{x_i,y_i}, \tag{5.17}$$

wobei die (x_i, y_i) die Knoten des Gitters sind und S_{x_i,y_i} Flächenelemente zu jedem Knoten bezeichnen, die zusammen ganz $\mathcal{G}$ ausfüllen. Die diskrete Form der Gleichung (5.16) lautet dann:

$$\sum_i d_i(u_{high}(i) - u_{low}(i))S_i = C - \sum_i u_{low}(i)S_i = \hat{C}, \tag{5.18}$$

wobei der Index i abkürzend für den Knoten (x_i, y_i) verwendet wurde.

Um maximale d_i zu finden, so daß (5.18) erfüllt ist, kann der folgende Algorithmus verwendet werden. Zuvor wird zur Vereinfachung der Schreibweise $p_i = u_{high}(i) - u_{low}(i)$ gesetzt.

Algorithmus 5.5 (Masseerhaltung)

1. *Führe einen Schritt des QMSL-Algorithmus 5.4 durch und erhalte Gewichte c_i. Nimm an, $\sum_i c_i p_i > \hat{C}$ (falls dies nicht der Fall sein sollte, setze $p_i \to -p_i$ und $\hat{C} \to -\hat{C}$).*
2. *Für alle i setze*
$$d_i = \begin{cases} c_i & , falls\ p_i \leq 0\ (setze\ marke(i) = 1), \\ 0 & , sonst\ (setze\ marke(i) = 0). \end{cases}$$
3. *Definiere einen Überschuß:*
$$\mathcal{U} = \hat{C} - \sum_{marke(k)=1} d_k p_k.$$
4. *Definiere einen mittleren Faktor*
$$d = \frac{\mathcal{U}}{\sum_{marke(k)=0} p_k}.$$
5. *Falls $d < c_k$ für alle k mit $marke(k) = 0$, dann setze $d_k = d$ für diese k und berechne u_i nach (5.12); andernfalls setze $d_k = c_k$ und $marke(k) = 1$ für diejenigen k, für welche gilt: $marke(k) = 0$ und $d > c_k$*
6. *Gehe zurück zu 2, falls u noch nicht berechnet werden konnte.*

Kapitel 6

Implementierung der Semi-Lagrange-Methode

6.1 Übersicht

Zwei Ziele stehen bei der Implementierung der SLM im Vordergrund: Die Methode soll im Kontext der FEM mit den entsprechenden Datenstrukturen auf einem Dreiecksgitter implementiert werden und sie soll adaptiv sein.

Grundsätzlich besteht ein SLM-Programm aus den folgenden drei Komponenten, die die Schritte des SLM-Algorithmus 5.1 abbilden:

1. Berechnung der Trajektorienstücke zu jedem Gitterpunkt.
2. Interpolation der stromaufwärts gelegenen Werte.
3. Berechnung der neuen Gitterpunktwerte.

Adaptivität kann auf verschiedenen Ebenen ansetzen. Adaptive FEM können zur Auflösung lokaler Phänomene Gitterverfeinerungen vornehmen oder die Diskretisierungsordnung verbessern, indem die Polynomordnung der Basisfunktionen erhöht wird. Bei zeitabhängigen Gleichungen kann darüber hinaus die Zeitschrittlänge angepaßt werden. Im Kontext der SLM soll *adaptiv* im Sinne einer Anpassung des Rechengitters an die Situation verstanden werden.

Adaptive Verfahren zeichnen sich dadurch aus, daß sie automatisch auf lokale Phänomene reagieren. Bei der Modellierung von Ozeanzirkulationen sind dies beispielsweise einzelne Wirbel, Fronten oder Grenzschichten. Wenn ein lokales Phänomen auftritt, kann es von einem adaptiven Verfahren lokalisiert und hoch aufgelöst werden. Wenn es verschwindet, kann das Gitter wieder vergröbert werden, so daß Rechenzeit und Speicherplatz gespart werden. Das erste Ziel dieses Kapitels ist daher, einen adaptiven Algorithmus zu entwickeln und ein Verfahren zu finden, die lokalen Phänomene zu lokalisieren.

Wie im Kapitel 5 dargestellt, ist die Interpolation der aufwendigste Teil in der SLM. Wesentliches zweites Ziel dieses Kapitels ist die Implementierung der Interpolation auf lokal verfeinerten (d.h. irregulären) Dreiecksgittern. Dabei wird im wesentlichen auf bekannte Bibliotheksroutinen zurückgegriffen, die an die Datenstrukturen aus Kapitel 2 angepaßt sind.

Zusätzlich zu den im Teil I vorgestellten Methoden müssen einige weitere Aufgabenstellungen bewältigt werden. Dazu gehört die Suche von Elementen zu gegebenen Koordinaten, die Berechnung von Gradienten auf irregulären Dreiecksgittern sowie die Berechnung von diagnostischen Größen.

In den folgenden Abschnitten werden konkrete Implementierungen der Gitterroutinen, der Interpolation, der Diagnoseroutinen und weiterer Programmteile beschrieben. Die Wahl der zugehörigen Algorithmen und Verfahren wird begründet.

6.2 Algorithmus und adaptive Strategie

Strategien zur automatischen Anpassung des Gitters zur Minimierung des Diskretisierungsfehlers werden unter anderem von Bänsch und Verfürth beschrieben [13, 138]. Die grundlegende Idee sogenannter *a posteriori* Fehlerkontrolle besteht darin, die gegebenen Gleichungen in jedem Zeitschritt zunächst zu lösen, dann den Fehler abzuschätzen, das Gitter so zu modifizieren, daß der abgeschätzte Fehler minimiert wird, und dann erneut zu lösen. Dieser iterative Prozeß ist zunächst aufwendiger als das herkömmliche Verfahren, bei dem in jedem Zeitschritt lediglich einmal auf einem festen Gitter gelöst wird. Wird das Gitter so modifiziert, daß zwar der abgeschätzte Fehler minimiert wird, zugleich aber möglichst wenige Gitterknoten erzeugt werden, dann kann durch die wesentlich geringere Anzahl an benötigten Knoten eine Effizienzsteigerung erzielt werden.

Im folgenden Programm wird der Teil zur Lösung der Advektionsgleichung in drei Routinen implementiert, die die Algorithmen 5.1 bzw. 5.2 abbilden. Zunächst werden die Positionen der stromaufwärts gelegenen Punkte berechnet, dann werden die rechten Seiten berechnet (im wesentlichen ist dies die Interpolation der Werte zwischen Gitterknoten), und schließlich wird die Funktion zum neuen Zeitpunkt mit Hilfe der diskretisierten Semi-Lagrange-Form der Advektionsgleichung (5.3) ermittelt.

Die Gittermodifikation wird jeweils in zwei Schritten für Verfeinerung bzw. Vergröberung vorgenommen: Zunächst werden mit Hilfe eines Fehlerschätzers die Elemente des Gitters ermittelt, die verfeinert werden sollen. Im zweiten Schritt werden die Elemente auf Ihre Verfeinerbarkeit überprüft und entsprechend verfeinert (analog wird bei der Vergröberung vorgegangen).

Ein Fehlerschätzer wird in Teil III genauer beschrieben. Hier wird lediglich der Gradient der Funktion als Indikator für den Fehler verwendet: Das Gitter wird verfeinert, wo der Gradient der Funktion groß ist:

Methode 6.1 (Verfeinerungsindikator) *Gegeben sei eine Toleranz θ_{fein}. Dann markiere ein Dreieck τ der feinsten Stufe der Triangulierung T zur Verfeinerung, wenn gilt:* $|\nabla u|_\tau \geq \theta_{fein} \cdot \max_{\tau \in T}(|\nabla u|_\tau)$.

Analog wird die Vergröberung vorgenommen:

Methode 6.2 (Vergröberungsindikator) *Gegeben sei eine Toleranz θ_{grob}. Dann markiere ein Dreieck τ der Triangulierung T zur Vergröberung, wenn es regulär verfeinert ist und für wenigstens drei der vier Tochterelemente gilt:* $|\nabla u|_{\tau_{kind}} \leq \theta_{grob} \cdot \max_{\tau_{kind} \in T}(|\nabla u|_{\tau_{kind}})$.

Die Triangulierung wird dann entsprechend der Methode 2.5 zur regulären Gitterverfeinerung modifiziert.

Meist wird im ersten Schritt der inneren Iteration (Algorithmus 6.1) eine große Anzahl Elemente der Triangulierung vergröbert oder verfeinert. In den weiteren Iterationen werden nur noch wenige Gitterpunkte hinzugefügt bzw. gelöscht. Daher wird eine Schranke für die minimale Anzahl der zu modifizierenden Elemente eingeführt, ab der das Gitter tatsächlich verändert wird.

Methode 6.3 (Pegel für Gittermodifikation) *Gegeben sei ein* Verfeinerungspegel ρ_{fein}. *Ermittle die Zahl* m_T^{fein} *der Elemente, die vom Fehlerschätzer für Verfeinerung markiert sind. Falls die relative Anzahl der zu modifiziernenden Elemente größer als der Pegel ist, d.h.* $\frac{m_T^{fein}}{m_T^{gesamt}} \geq \rho_{fein}$, *dann verfeinere, sonst verwende weiter die alte Triangulierung. Analog wird für einen Pegel* ρ_{grob} *für die Vergröberung verfahren.*

Für die Berechnung der rechten Seiten, die sich aus den Werten der Funktion u zur Zeit t ergibt, ist die Zwischenspeicherung des alten Gitters notwendig, denn u zur Zeit t ist auf einem Gitter zur Zeit t gegeben. Bänsch schlägt für zeitlich variable Gitter eine Interpolation/Restriktion von Gitterwerten vor [13]. Diese Interpolation/Restriktion erscheint hier nicht sinnvoll, da nicht nur auf Gitterpunkten gerechnet wird, sondern zwischen Gitterpunkten interpoliert werden muß. Es müssen jedoch nicht alle Parameter des Gitters gespeichert werden, es reicht aus, die Geometrie (Koordinaten und Knotennumerierung), die Konnektivität (Nachbarbeziehung) und die hierarchische Struktur (für den Suchalgorithmus bei der Dreieckssuche) zu speichern. Die vorgestellten Verfahren lassen sich zum folgenden adaptiven Algorithmus zusammenfassen:

Algorithmus 6.1 (Adaptive Semi-Lagrange-Methode)

1. *Lese Daten ein, initialisiere die Felder, erzeuge Anfangstriangulierung, etc.*
2. *Durchlaufe für alle Zeitschritte die folgenden Schritte:*
   (a) *Benutze das Gitter des vorherigen (alten) Zeitschrittes t als Anfangsgitter für den neuen (jetzigen) Zeitschritt $t + \Delta t$.*
   (b) *Führe einen Semi-Lagrange-Schritt auf dem jetzigen Gitter durch (d.h. Berechne die Trajektorien $\boldsymbol{\alpha}$, berechne die rechte Seite durch Interpolation der $u(\mathbf{x} - \boldsymbol{\alpha}, t)$, berechne die neuen Gitterwerte $u(\mathbf{x}, t + \Delta t)$).*
   (c) *Verfeinere die Elemente τ, die nach Methode 6.1 markiert sind.*
   (d) *Vergröbere die Elemente τ, die nach Methode 6.2 markiert sind.*
   (e) *Falls sich das Gitter geändert hat (beachte den Pegel, Methode 6.3), gehe zum Schritt 2b (innere Iteration), sonst:*
   (f) *Führe Diagnoseroutinen aus, erhöhe den Zeitschritt und gehe zu 2a (äußere Iteration).*
3. *Postprocessing: Sichere Aufsetzdaten, Graphik, etc.*

Der erste Schritt des Programms, *die Initialisierung*, beinhaltet alle gängigen Aufgaben für Modelle dieser Art:

- Die Kommandozeile wird auf Eingaben überprüft (hier kann insbesondere die Ausgabe der Ergebnisse beeinflußt werden),
- die Eingabedaten werden gelesen (dazu gehören die Zeitschrittsteuerung, physikalische Parameter, Daten von vorherigen Modelläufen, etc.),
- die Ausgabe (z.B. graphische Darstellung der Funktion oder des Gitters) wird initialisiert,
- Anfangswerte und Konstanten (z.B. für die Masseerhaltung) werden berechnet,
- das Anfangsgitter wird erstellt.

Die *Hauptschleife* über alle Zeitschritte enthält eine *innere Schleife* zur adaptiven Verfeinerung des Gitters. Dazu muß jeweils das Gitter des vorherigen Zeitschrittes zwischengespeichert werden. Die innere Schleife enthält den SLM-Zyklus (Berechnung der Trajektorien, Berechnung der rechten Seiten, Berechnung der neuen Gitterwerte) und die Routinen zur Anpassung des Gitters. Das *Postprocessing*, die Nachbehandlung der Daten, beschränkt sich hier auf das Speichern der Daten und die Berechnung neuer Eingabedaten, falls die Integration fortgesetzt werden soll.

6.3 Gitterroutinen

Unter dem Oberbegriff *Gitterroutinen* werden alle Programmteile zusammengefaßt, die zur Erzeugung und Verwaltung des Gitters dienen. Insbesondere zählen dazu die:

- Gittergenerierung (Verfeinerung, Vergröberung),
- Elementsuche für gegebene Koordinaten.

Auf die Gittergenerierung ist im ersten Teil schon eingegangen worden (siehe Abschnitt 2.2). Sie soll daher hier nicht näher behandelt werden. In diesem Abschnitt wird nur die Elementsuche im Dreiecksgitter behandelt.

6.3.1 Elementsuche im Dreiecksgitter

Die Datenstruktur der Triangulierung ist elementorientiert. Das heißt, sämtliche Informationen sind auf die Elementnummer bezogen: Die Knoten eines Elements können diesem direkt zugeordnet werden, die Koordinaten zu den Knoten sind bekannt, die Nachbarelemente zu einem gegebenen Element sind bekannt, usw.

Zu einem gegebenen Punkt dasjenige Element der Triangulierung zu finden, welches den Punkt enthält, ist dagegen nicht einfach. Dieses Problem tritt jedoch relativ häufig auf. Insbesondere bei der Interpolation der stromaufwärts gelegenen Punkte muß das zugehörige Element bestimmt werden. Zwei Faktoren spielen bei der Suche eine zeitkritische Rolle:

1. Das Kriterium (innerhalb – außerhalb).
2. Der Suchalgorithmus.

Kriterium

Das Kriterium, ob ein gegebener Punkt innerhalb oder außerhalb eines Elementes liegt, soll verschiedenen Ansprüchen genügen:

- Es soll effizient sein, d.h. möglichst wenige Operationen benötigen;
- es soll genau sein, d.h. auch Punkte, die auf dem Rand des Elementes liegen, sollen mit hinreichender Genauigkeit dem Element zugeordnet werden können;
- es soll stabil sein, d.h. Punkte in der Nähe der Ränder sollen nicht zu großen numerischen Fehlern führen.

Das Problem läßt sich etwas allgemeiner formulieren:

Problem 6.4 *Gegeben sei ein Punkt in der Fläche* ($\mathbf{x} = (x, y) \in \mathbf{R}^2$) *und ein geschlossener Polygonzug* ($P = \{\mathbf{x}_i = (x_i, y_i) \in \mathbf{R}^2 : i = 1, \ldots, n; \mathbf{x}_1 = \mathbf{x}_n; \mathbf{x}_i$ *Ecken*$\}$). *Entscheide, ob* (x, y) *innerhalb oder außerhalb von P liegt.*

Übliche Methoden aus der graphischen Datenverarbeitung verwenden Strahlen, die ihren Ursprung im Punkt $\mathbf{x}$ haben und den Polygonzug schneiden. Ein Punkt $\mathbf{x}$ liegt beispielsweise innerhalb des Polygons, wenn eine beliebige Gerade durch $\mathbf{x}$ den Polygonzug einmal rechts von $\mathbf{x}$ und einmal links davon schneidet. Ein weiteres Kriterium besagt, daß die Summe der Winkel

aller Linien zwischen $\mathbf{x}$ und den Ecken $\mathbf{x}_i$ 360° beträgt, wenn $\mathbf{x}$ im Polygon liegt. Diese Kriterien sind sehr allgemein, aber auch aufwendig zu berechnen.

Eine einfachere Methode zur Lösung des Problems 6.4 stammt aus der Theorie der linearen Programmierung und läßt sich allgemein formulieren:

Methode 6.5 (Lineare Programmierung) *Ein Punkt* $\mathbf{x}$ *liegt innerhalb eines Polygons* P*, wenn es für das folgende lineare Programmierungsproblem eine sinnvolle Lösung gibt:*

$$\begin{aligned} \mathbf{x} &= a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_{n-1}\mathbf{x}_{n-1} \\ \sum_{i=1}^{n-1} a_i &= 1 \\ a_1, \ldots, a_{n-1} &\geq 0. \end{aligned} \tag{6.1}$$

Für das spezielle Problem, bei dem P ein Dreieck ist, lassen sich die erste und die zweite Zeile aus Gleichung (6.1) in einem Gleichungssystem zusammenfassen:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6.2}$$

$\mathbf{x} = (x, y)$ liegt dann im Dreieck $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\}$, wenn $a_1, a_2, a_3 \geq 0$.

Diese Methode ist einfach zu implementieren. Sie ist allerdings nicht die effizienteste Methode, weil die Lösung eines 3×3-Gleichungssystems gesucht ist. Es zeigt sich weiterhin, daß in der Nähe der Gebietsränder numerische Instabilitäten auftreten, die auch mit einfacher Pivotsuche im Gleichungslöser nicht zu beheben sind.

Eine andere geometrisch motivierte Methode zur Lösung des Problems 6.4 verwendet die Fläche des Polygons (in diesem Falle Dreiecks) und vergleicht sie mit den Flächen der Teildreiecke:

Methode 6.6 (Geometrie) *Ein Punkt* $\mathbf{x}$ *liegt innerhalb eines Polygons* P*, wenn die Summe der (disjunkten) Teildreiecke aus je zwei Ecken und dem Punkt* $\mathbf{x}$ *gleich der Fläche des Polygons ist.*

Für das oben genannte spezielle Problem mit Dreieckselementen müssen also im wesentlichen vier Flächen berechnet werden. Die *Heronische Flächenformel* ist eine elegante Methode, die Fläche $F^{\triangle}$ eines Dreiecks zu bestimmen, weil sie lediglich die Kenntnis der drei Seitenlängen voraussetzt. Sie lautet:

$$F^{\triangle} = \sqrt{h_{\frac{1}{2}}(h_{\frac{1}{2}} - h_1)(h_{\frac{1}{2}} - h_2)(h_{\frac{1}{2}} - h_3)}, \tag{6.3}$$

dabei sind h_1, h_2, h_3 die drei Seitenlängen, $h_{\frac{1}{2}} = \frac{h_1+h_2+h_3}{2}$ der halbe Umfang des Dreiecks. Die Seitenlängen werden mit der Euklidischen Norm berechnet. Dazu ist jedesmal eine Wurzel zu ziehen, pro Flächenberechnung ergeben sich vier Wurzeloperationen.

Eine etwas effizientere Methode ist die Flächenberechnung mit Hilfe des *Kreuzproduktes* zweier Vektoren (Dreiecksseiten). Das Kreuzprodukt

$$\mathbf{h}_1 \times \mathbf{h}_2 = det\,|\mathbf{h}_1\ \mathbf{h}_2| = h_{1_1}h_{2_2} - h_{1_2}h_{2_1} \tag{6.4}$$

beschreibt die Fläche des von den Vektoren $\mathbf{h}_1 = (h_{1_1}, h_{1_2})$ und $\mathbf{h}_2 = (h_{2_1}, h_{2_2})$ aufgespannten Parallelogramms. Die Dreiecksfläche ergibt sich also aus der Hälfte des Kreuzproduktes zweier Dreiecksseiten. Dazu wird keine Wurzeloperation benötigt, und die Anzahl der übrigen arithmetischen Operationen ist kleiner als bei der Heronischen Flächenberechnung (6.3). Die Rechenzeit wird dadurch um einen Faktor über 2 verbessert.

Suchalgorithmus

Um das Element der Triangulierung zu finden, welches gegebene Koordinaten enthält, könnte man alle Elemente der feinsten Verfeinerung mit dem Kriterium überprüfen und die Suche abbrechen, sobald man ein Dreieck gefunden hat.

Bei regulärer und globaler Verfeinerung bis zur j-ten Stufe (siehe Abschnitt 2.2) besteht das feinste Gitter aus $\mathcal{O}(4^j)$ Elementen. Pro Punkt müßten also im Durchschnitt $\mathcal{O}(\frac{4^j}{2})$ Elemente überprüft werden.

Eine wesentlich effizientere Suchmethode bietet sich bei hierarchisch strukturierten Gittern an. Da die jeweils höhere Stufe der Verfeinerung in der niedrigeren Stufe enthalten ist, kann die Suche bei der gröbsten Verfeinerung beginnen. Wenn auf dieser Stufe ein Element gefunden wurde, werden dessen Kind-Elemente überprüft. Pro Stufe müssen dann im Durchschnitt nur zwei Elemente überprüft werden. Pro Punkt werden dann $\mathcal{O}(2j)$ Dreiecke mit dem Kriterium geprüft.

6.4 Interpolation

Im Abschnitt 5.5 wird kubische Interpolation als guter Kompromiß zwischen Recheneffizienz und Genauigkeit angegeben. Hier werden nun zwei verschiedene Verfahren beschrieben, die kubische Interpolation auf Dreiecksgittern realisieren.

Grundsätzlich werden im folgenden Spline-Interpolationsverfahren verwendet, die die Knotenwerte $u(\mathbf{x}_i)$ und jeweils die beiden ersten partiellen Ableitungen $\frac{\partial u}{\partial x}(\mathbf{x}_i)$ und $\frac{\partial u}{\partial y}(\mathbf{x}_i)$ an den Knoten des Dreiecks benötigen. Neben dem eigentlichen Interpolationsverfahren wird also noch eine Berechnungsmethode für die ersten partiellen Ableitungen benötigt. Bei linearen FEM-Funktionen sind die ersten Ableitungen an den Knoten nicht unmittelbar bekannt (wie etwa bei kubischen Elementen).

6.4.1 Berechnung der ersten partiellen Ableitungen

Zwei alternative Verfahren zur Berechnung der partiellen Ableitungen werden in diesem Unterabschnitt beschrieben. Die Methoden stammen von Klucewicz bzw. Akima.

Die Methode von Klucewicz [73] basiert auf der Mittelung der Gradienten der Dreiecksflächen, die den Knoten umgeben. Im Kontext der lokal verfeinerten Triangulierung aus Teil I wird die folgende modifizierte Methode formuliert:

Methode 6.7 (Klucewicz modifiziert) *Sei $\mathbf{x}_i = (x_i, y_i)$ der Dreiecksknoten, an dem die partiellen Ableitungen gesucht sind, sei weiter $I_i = \{(j,k) : j \neq k \neq i, (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ bilden ein Dreieck der Triangulierung$\}$ eine Indexmenge. Eine Ebene durch die Knoten $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ ist gegeben durch*

$$\mathcal{E}_{ijk}(x,y) = u(\mathbf{x}_i) + c_{ijk}^{(x)}(x - x_i) + c_{ijk}^{(y)}(y - y_i), \tag{6.5}$$

wobei

$$c_{ijk}^{(x)} = \frac{\partial \mathcal{E}_{ijk}}{\partial x}, \quad c_{ijk}^{(y)} = \frac{\partial \mathcal{E}_{ijk}}{\partial y}$$

so gewählt werden müssen, daß gilt:

$$\mathcal{E}_{ijk}(x_j, y_j) = u(\mathbf{x}_j), \quad \mathcal{E}_{ijk}(x_k, y_k) = u(\mathbf{x}_k).$$

Bilde die konvex kombinierte Fläche

$$\overline{\mathcal{E}_i(x,y)} = \frac{1}{w_i} \sum_{j,k \in I_i} w_{ijk} \mathcal{E}_{ijk}(x,y) \tag{6.6}$$

mit $w_i = \sum_{j,k \in I_i} w_{ijk}$, w_{ijk} *Gewichte. Die gesuchten partiellen Ableitungen sind dann:*

$$\frac{\partial u}{\partial x}(\mathbf{x}_i) = \frac{1}{w_i} \sum_{j,k \in I_i} w_{ijk} c_{ijk}^{(x)}, \quad \frac{\partial u}{\partial y}(\mathbf{x}_i) = \frac{1}{w_i} \sum_{j,k \in I_i} w_{ijk} c_{ijk}^{(y)} \tag{6.7}$$

Die ursprüngliche Implementierung des Verfahrens stammt von Renka [110]. Die Modifizierungen betreffen vor allem die Datenstrukuren für die Gitterdarstellung.

Ein anderes Verfahren für die Schätzung der partiellen Ableitungen stammt von Akima [3]. In den Tests (siehe Kapitel 8) erzielt es etwas bessere Ergebnisse als das Verfahren aus Methode 6.7. Es basiert auf der Berechnung von Vektorprodukten der Dreieckskanten um $\mathbf{x}_i$.

Methode 6.8 (Akima modifiziert) *Bilde die Vektorprodukte (Kreuzprodukte) der Vektoren*

$$\overline{\mathbf{x}_i \mathbf{x}_j}, \quad \overline{\mathbf{x}_i \mathbf{x}_k},$$

normiere, so daß die dritte Komponente nichtnegativ ist, summiere sie und erhalte einen gemittelten Ergebnisvektor $\overline{\mathbf{x}} = (\overline{x}_1, \overline{x}_2, \overline{x}_3)$ *mit* $\overline{x}_3 \geq 0$. *Dieser Vektor steht senkrecht zur Fläche, die die mittlere Steigung in* $\mathbf{x}_i$ *beschreibt und durch die Gleichung*

$$\widehat{\mathcal{E}_i(x,y)} = \frac{\overline{x}_1}{\overline{x}_3} x - \frac{\overline{x}_2}{\overline{x}_3} y - C \tag{6.8}$$

gegeben ist. Die geschätzten Gradienten ergeben sich also:

$$\frac{\partial u}{\partial x}(\mathbf{x}_i) = -\frac{\overline{x}_1}{\overline{x}_3}, \quad \frac{\partial u}{\partial y}(\mathbf{x}_i) = -\frac{\overline{x}_2}{\overline{x}_3} \tag{6.9}$$

Die Originalroutine wurde von Akima implementiert [2]. Modifizierungen betreffen wieder die Datenstrukturen. Die Methode 6.8 ist numerisch etwas aufwendiger als die Methode 6.7, der Unterschied ist aber so gering, daß Akimas Methode wegen der besseren Ergebnisse vorgezogen wird.

6.4.2 Berechnung der kubischen Spline-Interpolation

Zwei verschiedene Routinen für kubische Spline Interpolation auf einem Dreieck werden angewendet. Das erste Verfahren basiert auf einem Algorithmus von Renka [110]. Die Interpolationsmethode geht dabei auf Lawson [77] zurück.

Das Dreieck, welches den zu interpolierenden Punkt enthält, wird in drei Teildreiecke unterteilt, die jeweils aus zwei Ecken und dem Schwerpunkt gebildet werden. Innerhalb jeden Teildreiecks ist die Interpolationsfunktion ein kubisches Polynom:

$$q(x,y) = a_1 x^3 + a_2 x^2 y + a_3 x y^2 + a_4 y^3 + a_5 x^2 + a_6 xy + a_7 y^2 + a_8 x + a_9 y + a_{10} \tag{6.10}$$

Die Interpolationsbedingungen sind durch die Werte und die beiden ersten partiellen Ableitungen in jedem Eckpunkt des Dreiecks gegeben. Es wird C^1-Stetigkeit über die äußeren und inneren Kanten angenommen.

Das zweite Verfahren ist eine Modifikation der Routine von Montefusco und Casciola [94]. Für die Interpolation innerhalb eines Dreiecks wird hier ein Verfahren von Nielson [98] angewendet. Dabei werden Interpolierende gesucht, die eine bestimmte Pseudonorm über dem Dreieck minimieren. Man kann zeigen, daß die so gefundenen interpolierenden Funktionen die beste Approximation an die ursprüngliche Funktion darstellen. Nielson gibt ein 9-parametriges C^1-Element an, das in diesem Falle Verwendung findet.

6.4.3 Quasi-monotone und quasi-konservative Interpolation

In den Abschnitten 5.5.2 und 5.5.3 sind Verfahren beschrieben worden, die weitere Eigenschaften wie Monotonie und Erhaltungseigenschaften zur kubischen Interpolation hinzufügen. Hier sollen nun einige implementierungsspezifische Details erörtert werden.

Interpolation niederer Ordnung

Für die beiden in 5.4 und 5.5 gegebenen Algorithmen wird eine Interpolation niederer Ordnung benötigt. In diesem Falle liegt diese Interpolation in Form der FEM-Darstellung der Funktion auf linearen Elementen bereits vor. Mit Hilfe der stetigen, stückweise linearen Basisfunktionen

$$b^i_{ijk}(x,y) = \frac{(x-x_j)(y_k-y_j)-(x_k-x_j)(y-y_j)}{(x_i-x_j)(y_k-y_j)-(x_k-x_j)(y_i-y_j)}. \tag{6.11}$$

zu einem Dreieck mit den Knoten $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ kann die lineare Interpolation leicht mit der folgenden Formel berechnet werden:

$$u_{low}(\mathbf{x}) = u(\mathbf{x}_i)b^i_{ijk}(\mathbf{x}) + u(\mathbf{x}_j)b^j_{ijk}(\mathbf{x}) + u(\mathbf{x}_k)b^k_{ijk}(\mathbf{x}), \tag{6.12}$$

wobei $\mathbf{x}$ im Dreieck $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ liegt.

Flächenberechnung im quasi-konservativen Schema

Im Abschnitt 5.5.3 wird eine Methode beschrieben, mit der das Integral der Funktion konstant gehalten wird. Zur diskreten Berechnung des Integrals werden Flächenelemente S_{x_i,y_i} benötigt. Die Interpolation der Funktion erfolgt in der Regel zwischen Gitterpunkten. Die zugehörigen Flächenelemente werden jedoch an Gitterpunkten berechnet. Das Problem besteht also darin, ein Integral zu berechnen, dessen Funktionswerte möglicherweise nicht innerhalb der zugehörigen Flächenelemente liegen.

Dabei wird die Eigenschaft des Verfahrens für die Advektionsgleichung genutzt, daß zu jedem Gitterpunkt ein stromaufwärts gelegener Punkt gehört, dessen interpolierter Wert später den neuen Gitterwert ergibt. Das Integral der interpolierten Werte mit Flächenelementen, die sich auf die zugehörigen Gitterpunkte beziehen, ist also gleich dem Integral der Funktion zum neuen Zeitpunkt. Daher wird durch die Wahl der Flächenelemente an Gitterpunkten zu Werten an stromaufwärts gelegenen Punkten keine künstliche Ungenauigkeit eingeführt.

Die Flächenelemente ergeben sich aus der folgenden Summe (siehe Abb. 6.1):

$$S_{x_i,y_i} = \sum_{\tau \in I_i} \frac{F^{\triangle}_{\tau}}{3}.$$

Dabei ist I_i die Indexmenge der Elemente τ, die den Knoten (x_i, y_i) umgeben.

6.5 Diagnose

Um die Güte des Verfahrens überprüfen zu können, bedarf es einiger Parameter, die leicht überschaubar und zusammengefaßt Rechenfehler und Effizienz darstellen. Ein Grund zur Wahl des einfachen Advektionsproblems ohne Reibung und Quellen ist die Möglichkeit einer guten Fehleranalyse. Die Eigenschaften der Anfangsbedingungen müssen erhalten bleiben, die Abweichungen sind daher einfach zu berechnen.

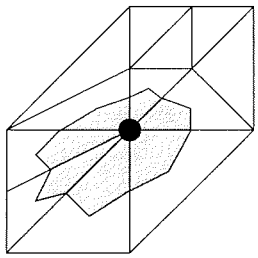

Abbildung 6.1: Flächenelement zu einem Gitterpunkt mit irregulärer Verfeinerung

In Anlehnung an die Arbeiten von Bermejo und Staniforth [25] und Priestley [107] werden die ersten und zweiten Momente (RFM bzw. RSM) der Funktion über dem Rechengebiet berechnet. Sie sind definiert:

$$RFM = \frac{\int_{\mathcal{G}} u_h(\mathbf{x},t)dx}{\int_{\mathcal{G}} u(\mathbf{x},0)dx}, \tag{6.13}$$

$$RSM = \frac{\int_{\mathcal{G}} u_h^2(\mathbf{x},t)dx}{\int_{\mathcal{G}} u^2(\mathbf{x},0)dx}, \tag{6.14}$$

wobei u die exakte Lösung repräsentiert und u_h die berechnete Näherung.

Da die L^2-Norm des Fehlers in den beiden Arbeiten angegeben ist, wird sie auch hier zu Vergleichszwecken berechnet:

$$\|e\|_{L^2}^2 = \|u - u_h\|_{L^2}^2 = \int_{\mathcal{G}} (u(\mathbf{x},t) - u_h(\mathbf{x},t))^2 dx. \tag{6.15}$$

Die Integralberechnungen in den Gleichungen (6.13) bis (6.15) werden mit der folgenden Quadraturformel durchgeführt:

$$\int_{\mathcal{G}} g(\mathbf{x})dx \approx \sum_{\tau \in T} F_\tau^\triangle \sum_{i=1}^{3} \frac{g(\mathbf{x}_i^{(\tau)})}{3}. \tag{6.16}$$

Dabei ist $F_\tau^\triangle$ die Fläche des Dreiecks τ, T ist die Menge aller Dreiecke des feinsten Gitters der Triangulierung, $\mathbf{x}_i^{(\tau)}$ sind die Knoten des Dreiecks τ. Es wird also der Mittelwert der Funktion g auf jedem Dreieck gebildet und mit der Fläche gewichtet.

Als weitere diagnostische oder einfach nur informative Größen werden die Anzahl der Knoten, der Elemente insgesamt und der Elemente auf dem feinsten Gitter für jeden Zeitschritt ausgelesen, außerdem das Maximum und das Minimum der Funktion, sowie die Maximumsnorm des Fehlers:

$$\|e\|_{max} = \|u - u_h\|_{max} = \max_i(|u(\mathbf{x}_i,t) - u_h(\mathbf{x}_i,t)|), \tag{6.17}$$

wobei der Index i über alle Knotennummern läuft.

Alle diese Berechnungen sind in einer Routine zusammengefaßt, die in jedem Zeitschritt aufgerufen wird (siehe Algorithmus 6.1). Der Aufruf der Diagnoseroutine kann aber auch ausgeschaltet werden.

Kapitel 7

Parallelisierung der Semi-Lagrange-Methode

7.1 Übersicht

Eine besonders attraktive Eigenschaft der SLM ist die potentiell gute Parallelisierbarkeit der Methode. Neben der Stabilität des Verfahrens und damit der Möglichkeit, die Zeitschrittlänge unabhängig von der lokalen Verfeinerung zu wählen, ist die Parallelisierbarkeit ein Kriterium für die Wahl des Verfahrens.

Alle in der SLM auftretenden Operationen an Gitterpunkten sind unabhängig voneinander und damit parallelisierbar. Auf diese Unabhängigkeit der Berechnung wird im Abschnitt 7.2 dieses Kapitels eingegangen.

Es zeigt sich, daß die SLM theoretisch sehr einfach zu parallelisieren ist. Durch die Adaptivität des Verfahrens ergibt sich eine praktische Schwierigkeit bei der Parallelisierung, weil irreguläre Datenstrukturen auftreten. Eine geeignete Datenpartitionierung ist daher wichtig. Die Irregularität ist mit Hilfe der im Kapitel 3 eingeführten Index-Array-Methode zur daten-parallelen Aufteilung beherrschbar. Im Abschnitt 7.3 wird die Anwendung der Methode auf die SLM dargestellt.

Die serielle Gittergenerierung beeinträchtigt die Effizienz der Parallelisierung. Immerhin verbrauchen die Gitterroutinen etwa 30% der Rechenzeit. Wenn die übrigen Programmteile parallelisiert sind und bei Erhöhung der Prozessoranzahl beschleunigen, dominieren die Gitterroutinen ab 4 Prozessoren die Gesamtrechenzeit. Das Verfahren skaliert daher nicht optimal. Auf die Parallelisierung der Gitterroutinen wird in diesem Teil jedoch verzichtet. In der eigentlich angestrebten Implementierung der SLM für die Flachwassergleichungen (Teil III) dominieren die Gitterroutinen nicht mehr stark, so daß dort für moderate Prozessorzahlen gute Skalierbarkeit erreicht wird.

Zielrechner für die Parallelisierung ist die Kendall Square Research KSR-1. Die Architektur dieses massiv parallelen Computers ist im Anhang B beschrieben.

7.2 Parallelisierung der Operationen an Gitterpunkten

Im Zusammenhang der SLM von „Operationen an Gitterpunkten“ zu sprechen, wird der Sache nicht ganz gerecht. Präziser ausgedrückt handelt es sich um Operationen an einer Trajektorie. Gemeint sind nämlich alle Operationen, die jeweils zur Berechnung aller Werte an einer Trajektorie benötigt werden:

1. die Berechnung des Anfangspunktes der Trajektorie,
2. die Berechnung des (interpolierten) Wertes am Anfangspunkt der Trajektorie $u(\mathbf{x} - \boldsymbol{\alpha}, t)$,
3. die Berechnung des neuen Wertes am Endpunkt (Gitterpunkt) der Trajektorie $u(\mathbf{x}, t + \Delta t)$.

Diese Berechnungen für eine Trajektorie lassen sich völlig unabhängig von den Berechnungen einer anderen Trajektorie durchführen. Es bietet sich daher an, über die Anzahl der Trajektorien (das entspricht gerade der Anzahl der neuen Gitterpunkte) zu parallelisieren.

Die drei Berechnungsschritte entlang einer Trajektorie lassen sich im einzelnen untersuchen. Für die passive Advektion besteht der dritte Schritt lediglich aus dem Kopieren der im Schritt 2. ermittelten stromaufwärts gelegenen Daten an die Gitterpunkte. Diese Operation ist trivialerweise vollkommen parallelisierbar. Im Schritt 1. werden mit Hilfe der Funktion für den Wind, $\mathbf{a}(x, y)$ die Trajektorien $\boldsymbol{\alpha}$ exact berechnet. Auch diese Operation läßt sich ohne Schwierigkeiten parallelisieren.

Datenzugriffe (zu möglicherweise entfernten Speicherpositionen) sind lediglich im 2. Schritt notwendig. Wenn die Trajektorien so lang sind, daß sie in ein entfernt liegendes Element reichen, muß aus den Daten dieses Elements interpoliert werden. Die Elementdaten können dann im Speicher eines anderen Prozessors liegen.

Eine weitere Möglichkeit des Zugriffs auf entfernte Daten ergibt sich bei der kubischen Spline-Interpolation: Dort werden als Eingabeparameter die Werte an den drei Ecken eines Dreiecks sowie die ersten partiellen Ableitungen benötigt. Beim Schätzen der Ableitungen werden die Normalenvektoren aus den Knotenwerten der umliegenden Dreiecke berechnet. Dabei kann Datenzugriff auf entfernte Speicherelemente auftreten. In beiden Fällen handelt es sich aber lediglich um Lesezugriffe. Das Lesen entfernt liegender Daten ist für die Parallelisierbarkeit ohne Bedeutung. Wenn die Kommunikationskosten minimiert werden sollen, muß aber bei der Datenpartitionierung darauf geachtet werden, daß Lesezugriffe auf möglichst wenige Speicherelemente notwendig wird.

7.3 Datenpartitionierung

Die KSR-1 ist ein Computer mit virtuellem gemeinsamem Hauptspeicher. Das entbindet den Programmierer von der Aufgabe, Daten explizit zu versenden und bestimmten Prozessoren bzw. Speichersegmenten zuzuordnen. Da die zugrundeliegende Hardware jedoch nur lokalen Speicher kennt und daher entsprechende Latenzzeiten aufweist, muß trotzdem auf eine effiziente Datenverteilung geachtet werden.

Dabei können einzelne Datenbewegungen vernachlässigt werden. Das macht die Parallelisierung der SLM sehr angenehm. Während Thomas und Côté mit Hilfe von explizitem Message Passing die Länge der Trajektorien (d.h. die Courantzahl der Strömung) beschränken müssen [134], ist es auf der KSR-1 kein Problem, in einzelnen Fällen über die Prozessorgrenze hinaus auf Daten zuzugreifen. Die Beschränkung der Courantzahl ist notwendig, um Speicherplatz zu sparen. Wäre der lokale Speicher jedes Prozessors groß genug, das gesamte Datenfeld des alten Zeitschrittes als Kopie zu fassen, dann könnte zu Beginn jedes Zeitschrittes einmal eine globale Datenverteilung durchgeführt und danach parallel ohne Datenkommunikation gerechnet werden.

Massiv parallele Computer sind nicht nur für die Beschleunigung der Berechnung attraktiv, sondern auch wegen ihrer großen Hauptspeicher. So besitzt die KSR-1 mit 32 Prozessoren und 32MB pro Prozessor 1GB an (preiswertem) Hauptspeicher. Die Daten des gesamten Problems in jeden lokalen Speicher zu kopieren, ist daher nicht sinnvoll. Vielmehr sollte jeder lokale Speicher

einen Teil der Daten enthalten, der vom zugehörigen Prozessor bearbeitet wird. Dazu ist eine Datenpartitionierung notwendig.

Wie bei der Parallelisierung der FEM im ersten Teil werden auch hier Index-Arrays eingesetzt, um die Daten über verschiedene Teile des Programms hinweg im Speicher eines Prozessors zu fixieren. Damit werden die teuren Schreibzugriffe vermieden, weil Daten zwar von entfernten Prozessoren gelesen, aber ausschließlich vom lokalen Prozessor geschrieben werden. Die Speicherposition bleibt für alle Operationen innerhalb eines Zeitschrittes gleich.

Die Partitionierung der Daten erfolgt in der Reihenfolge der Knotennummern. Dabei werden die Knoten in Blöcken der Größe einer KSR-Subpage auf die Prozessoren verteilt. Wegen der Verfeinerungsstrategie (bei regulärer Verfeinerung entstehen drei benachbarte Knoten mit aufeinander folgenden Knotennummern) liegen die Knoten einer Subpage meistens in räumlicher Nähe zueinander. Für den Lesezugriff aufeinanderfolgender Knotendaten braucht nur einmal kommuniziert zu werden.

Wenn Berechnungen mit räumlich benachbarten Knoten durchgeführt werden, sind die zugehörigen Daten aufgrund dieser Eigenschaft häufig in der selben Subpage enthalten und liegen damit schon im lokalen Speicher. Insbesondere auf feinen Gittern trifft diese Aussage zu, während die Knoten des gröbsten Gitters physikalisch weit auseinander liegen. Für die Parallelisierung stellen die weit entfernten Knoten aber eine kleine Zahl dar, so daß das parallelisierte Verfahren gut skaliert.

Nach jeder Gitteränderung wird eine Neuverteilung der Daten vorgenommen. Der parallele Algorithmus ist im folgenden angegeben (vergleiche mit Agorithmus 6.1).

Algorithmus 7.1 (Parallele adaptive Semi-Lagrange-Methode)

1. *Lese Daten ein, initialisiere die Felder, erzeuge Anfangstriangulierung, etc.*
2. *Durchlaufe für alle Zeitschritte die folgenden Schritte:*
   (a) *Benutze das Gitter des vorherigen (alten) Zeitschrittes t als Anfangsgitter für den neuen (jetzigen) Zeitschritt $t + \Delta t$.*
   (b) **Verteile die Daten auf die Prozessoren und speichere die Datenverteilung in Index-Arrays.**
   (c) *Führe einen Semi-Lagrange-Schritt auf dem jetzigen Gitter* **parallel** *durch.*
   (d) *Verfeinere oder vergröbere das Gitter wie im Algorithmus 6.1.*
   (e) *Falls sich das Gitter geändert hat, gehe zum Schritt 2b (innere Iteration), sonst:*
   (f) *Führe Diagnoseroutinen aus, erhöhe den Zeitschritt und gehe zu 2a (äußere Iteration).*
3. *Postprocessing: Sichere Aufsetzdaten, Graphik, etc.*

7.4 Probleme bei der Parallelisierung

Aus Abbildung 7.1 wird ersichtlich, wo das Problem bei der Parallelisierung der SLM für die passive Advektion liegt. Der Anteil des seriellen Programmteils zur Erzeugung des Gitters an der gesamten Rechenzeit beträgt etwa 30%. Dieser Anteil wird bei Parallelisierung der Routinen für die Berechnung der Trajektorienoperationen dominant.

Mit Hilfe von Amdahls Gesetz (A.1) läßt sich theoretisch zeigen, daß kein adäquater Speedup zu erreichen ist. Mit einem parallelen Anteil von 60% an der Gesamtrechenzeit kann der Speedup theoretisch nicht über 2,5 hinausgehen. Dieser Sachverhalt ist in Abbildung 7.2 verdeutlicht.

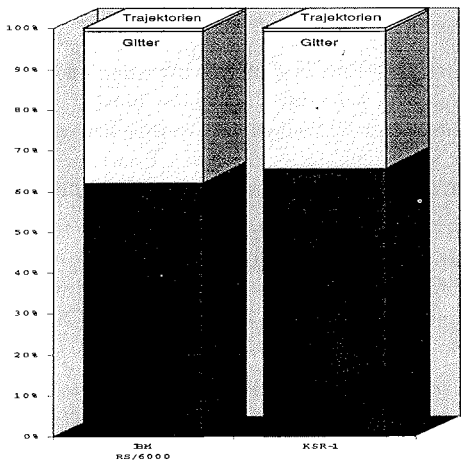

Abbildung 7.1: Anteile der einzelnen Routinen an der gesamten Rechenzeit des Zeitschrittes (a: gemessen auf einer IBM RS/6000, b: gemessen auf einem Prozessor der KSR-1)

Die Parallelisierung der Gitterroutinen wird in dieser Arbeit nicht weiter verfolgt, weil davon ausgegangen wird, daß der Rechenzeitanteil der Gitterverfeinerung im Semi-Lagrange-Verfahren für die Flachwassergleichungen (dem eigentlichen Ziel der Arbeit) nicht mehr so stark ins Gewicht fällt. Die übrigen Operationen sind dort wesentlich aufwendiger, so daß bis zu einer moderaten Anzahl Prozessoren mit guter Skalierbarkeit des Gesamtverfahrens gerechnet werden kann (vergleiche Kapitel 11).

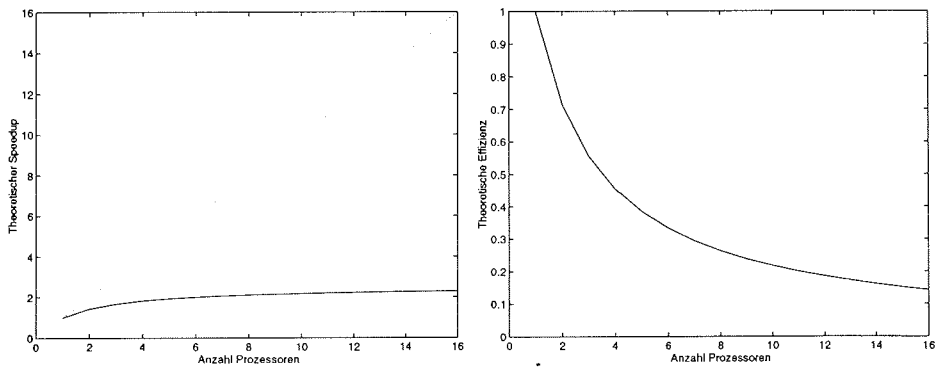

Abbildung 7.2: Amdahls Gesetz für 60% parallelen Programmteil, 16 Prozessoren und einer seriellen Ausführungszeit von 1 (Links: Speedup, rechts: Effizienz)

Kapitel 8

Ergebnisse für die Semi-Lagrange-Methode

8.1 Übersicht

In diesem Kapitel werden die Eigenschaften der SLM anhand von experimentellen Ergebnissen beschrieben. Zunächst ist es wichtig, die numerische Konsistenz des Verfahrens auch experimentell zu zeigen. Dazu wird ein Testbeispiel gerechnet, das auch in anderen Arbeiten (z.B. von Bermejo und Staniforth [25] und von Priestley [107]) verwendet wird. Ursprünglich stammt das Beispiel des *geschlitzten Zylinders* von Zalesak [148].

Das Testbeispiel stellt hohe Anforderungen bezüglich der numerischen Reibung und der Auflösung von scharfen Kanten an das Verfahren. Die Eigenschaften der adaptiven SLM im Hinblick auf die Genauigkeit werden im zweiten Abschnitt diskutiert. Einflüsse von Interpolationsroutine, Gradienten- und Trajektorienberechnung werden beschrieben.

Eine der Neuerungen des hier vorgestellten Verfahrens ist die Adaptivität des Gitters. Verschiedene Versuche mit und ohne lokale Gitterverfeinerung werden durchgeführt, um den Einfluß der Adaptivität zu untersuchen. Im Abschnitt 8.3 werden Ergebnisse aus adaptiven Modellrechnungen denen aus global verfeinerten Fällen gegenübergestellt.

Der Abschnitt 8.5 beschreibt die Ergebnisse der Parallelisierung. Die parallelen Teile des Verfahrens sind gut skalierbar. Das Modellproblem mit global verfeinertem Gitter ist ebenfalls parallel effizient. Lediglich bei adaptiver (lokaler) Gitterverfeinerung werden die seriellen Gitterroutinen zum Flaschenhals für die Parallelisierung. Im letzten Abschnitt dieses Kapitels wird eine Zusammenfassung gegeben.

Das Implementierte Verfahren ist als Flußdiagramm in Abbildung 8.1 dargestellt. Die Varianten der Trajektorienberechnung und die Interpolationsmethoden sind rechts neben den entsprechenden Programmteilen angegeben.

8.2 Numerische Eigenschaften der SLM

Ein Standard-Modellproblem für Verfahren zur Lösung von Advektionsproblemen ist von Zalesak [148] eingeführt worden. Es ist in Abbildung 8.2 dargestellt.

Beispiel 8.1 (Geschlitzter Zylinder) *Die passive Advektionsgleichung (5.1) wird mit den folgenden Bedingungen gelöst:*

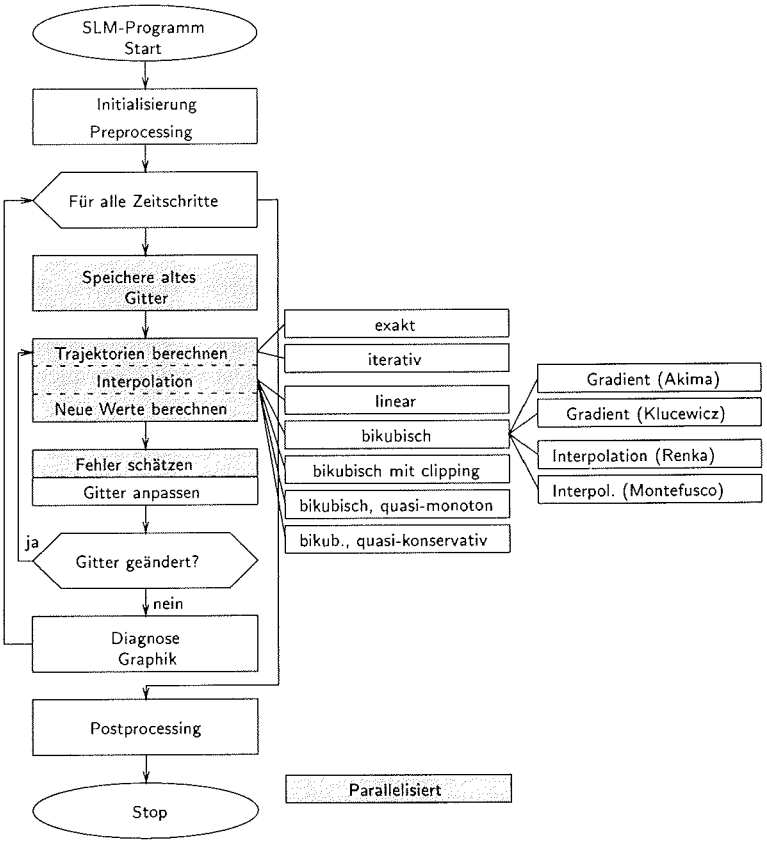

Abbildung 8.1: Flußdiagramm des SLM-Verfahrens, angegeben sind implementierte alternative Methoden für die Trajektorienberechnung und die SLM-Interpolation, die grau unterlegten Teile sind parallelisiert

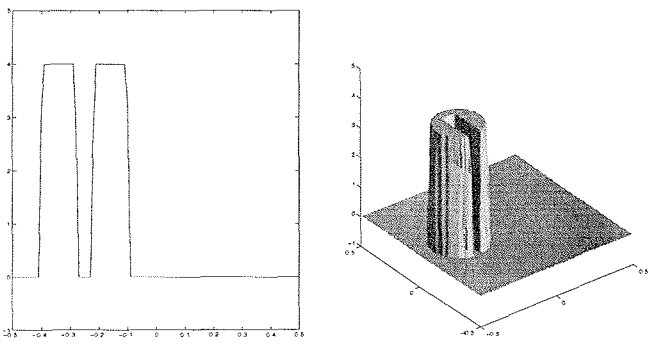

Abbildung 8.2: Anfangsbedingung für das Problem 8.1 des geschlitzten Zylinders (Schnitt entlang der x-Achse, Oberflächenansicht)

1. *Das Rechengebiet sei* $\mathcal{G} = [-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$.
2. *Die Anfangsbedingung sei durch eine Funktion* $u(\mathbf{x}, t = 0) = u^0(\mathbf{x})$ *gegeben, die innerhalb eines geschlitzten Zylinders den Wert vier und außerhalb den Wert null annimmt. Der Zylinder habe den Radius 0,15, das Zentrum im Punkt* $(-\frac{1}{4}, 0)$*, der Schlitz habe die Länge 0,22 und die Breite 0,06.*
3. *Der Wind* $\mathbf{a}$ *sei gegeben durch* $\mathbf{a} = -\omega(-y, x)$ *mit* $\omega = 0.3636 \times 10^{-4}$.
4. *Die Zeitschrittlänge für den Referenzlauf ist* $\Delta t = 1800\ s$.

Zunächst werden einige Versuche mit unterschiedlichen Interpolations- und Gradientenschätzungsalgorithmen angegeben. Daß lineare Interpolation für die Berechnung der stromaufwärts gelegenen Werte nicht ausreicht sieht man in der Abbildung 8.3. Bereits nach einer Umdrehung ist der geschlitzte Zylinder stark „erodiert".

Um die Genauigkeit des Verfahrens untersuchen und mit anderen Ergebnissen aus der Literatur vergleichen zu können, werden diagnostische Größen berechnet und angegeben. Analog zur Ar-

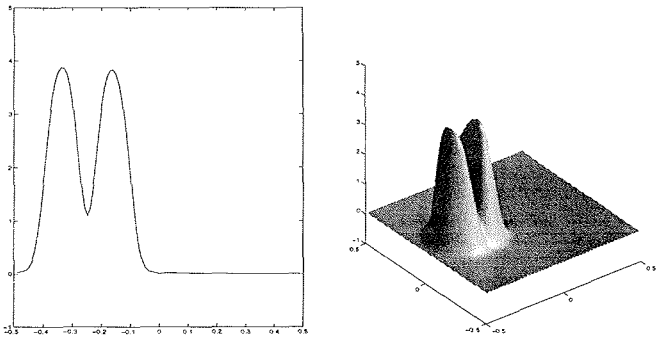

Abbildung 8.3: Geschlitzter Zylinder nach einer Umdrehung mit linearer Interpolation (Schnitt entlang der x-Achse, Oberflächenansicht)

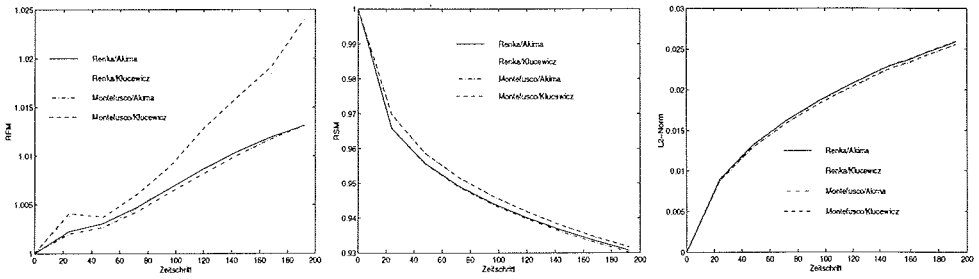

Abbildung 8.4: RFM, RSM und L^2-Norm für zwei Umdrehungen und die vier Kombinationen von Interpolation und Gradientenschätzer

beit von Bermejo und Staniforth [25] werden die ersten beiden Momente, RFM (erstes Moment) und RSM (zweites Moment), aus (6.13) und (6.14) angegeben. Die L^2-Norm ist die Summe des Dissipationsfehlers und des Dispersionsfehlers, die in der zitierten Arbeit angegeben werden. Ein Vergleich des gesamten numerischen Fehlers läßt sich also anstellen.

Die im Abschnitt 6.4 angegebenen verschiedenen Verfahren zur Berechnung von Interpolation und Gradient (siehe Abb. 8.1) wurden auf ihre Genauigkeit untersucht und miteinander verglichen. In Abbildung 8.4 sind die diagnostischen Größen RFM, RSM und L^2-Norm angegeben. Für die ersten Momente zeigen die Verfahren mit Gradientenschätzer von Akima deutliche Überlegenheit, vor allem in Kombination mit der Interpolation von Montefusco et al. Die Werte für die zweiten Momente und die L^2-Norm liegen bei allen Routinen eng zusammen. Die Vorteile, die hier durch den Gradientenschätzer von Klucewicz erzielt werden sind so gering, daß für die weiteren Tests die Kombination Interpolation von Montefusco et al. und Gradienteschätzer von Akima verwendet wird.

Die Genauigkeit des Verfahrens hängt im wesentlichen von der Interpolation und weniger von der Extrapolation der Trajektorienstücke $\boldsymbol{\alpha}_m$ ab. Die Abbildungen 8.5 bis 8.8 stellen jeweils die diagnostischen Werte mit exakten und mit iterativ berechneten Trajektorienstücken dar. Der Vergleich von exakten mit iterativ berechneten $\boldsymbol{\alpha}_m$ zeigt für die passive Advektion mit bekanntem Wind kaum signifikante Unterschiede. Lediglich bei der reinen bikubischen Interpolation ohne Monotonie oder Masseerhaltung sind Unterschiede beim RFM-Wert feststellbar.

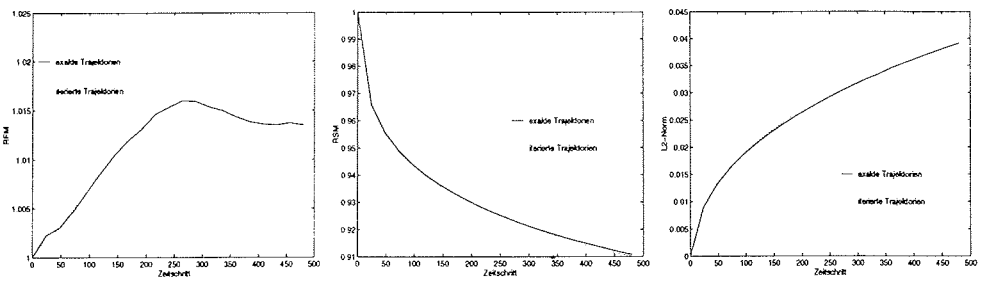

Abbildung 8.5: RFM, RSM und L^2-Norm für fünf Umdrehungen mit bikubischer Interpolation und exakt, bzw. iterativ berechneten Trajektorienstücken

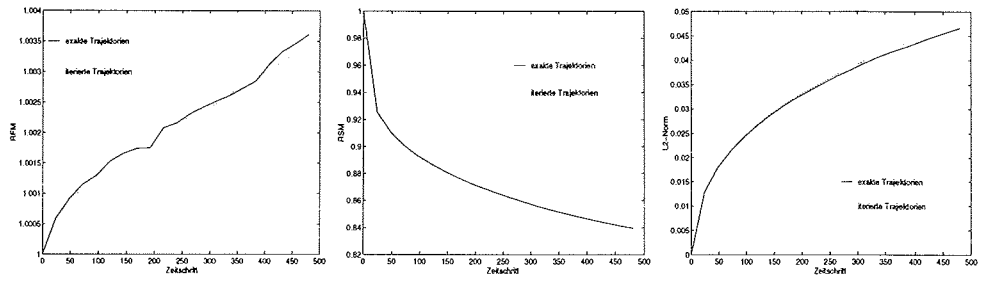

Abbildung 8.6: Wie Abbildung 8.5, jedoch mit bikubischer Interpolation und clipping

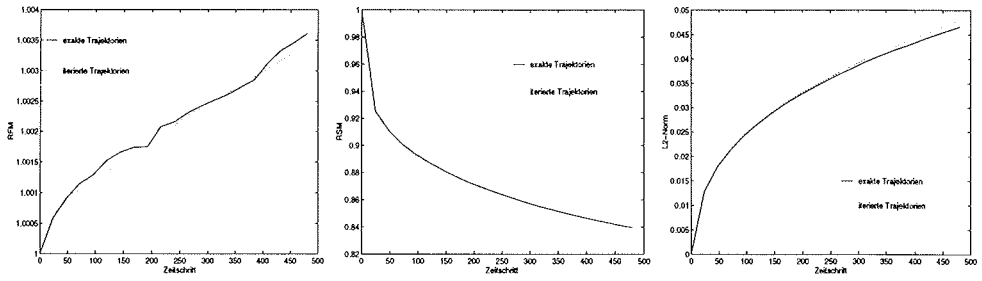

Abbildung 8.7: Wie Abbildung 8.5, jedoch mit bikubischer QMSL-Interpolation

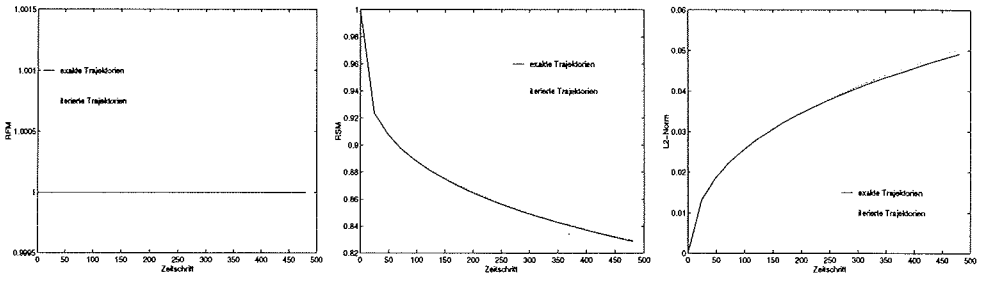

Abbildung 8.8: Wie Abbildung 8.5, jedoch mit bikubischer, quasi masseerhaltender Interpolation

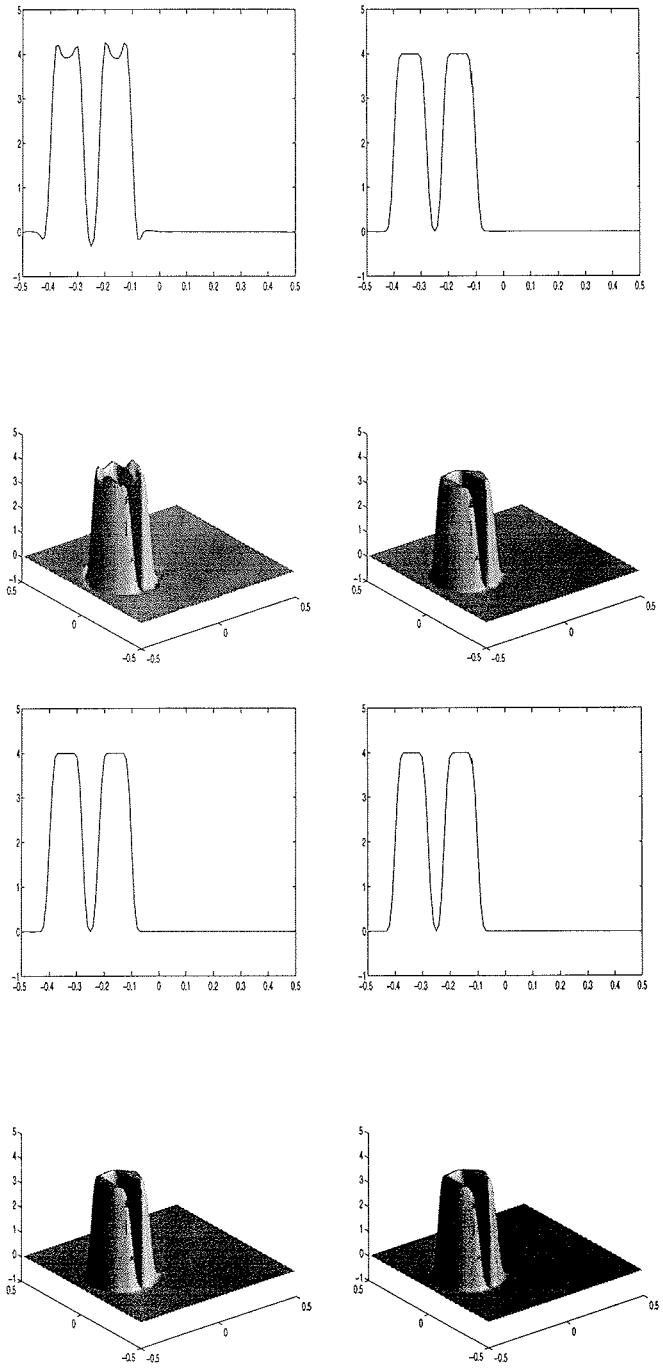

Abbildung 8.9: Wie Abbildung 8.2, jedoch nach fünf Umdrehungen für bikubische Interpolation, Interpolation mit Clipping, QMSL-Interpolation und masseerhaltende Interpolation (von links)

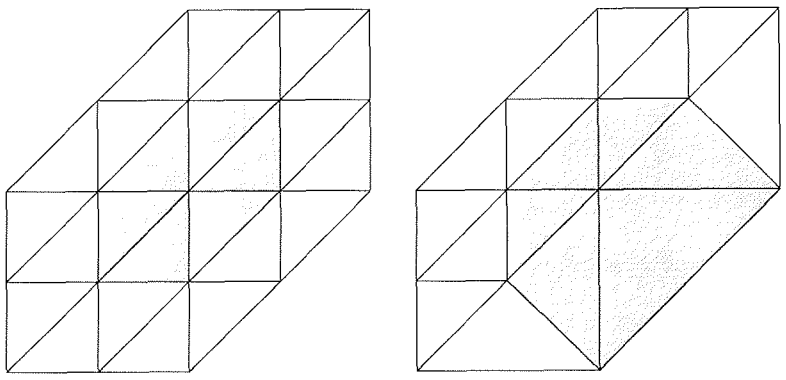

Abbildung 8.10: Elemente, die in die Berechnung des Gradienten am zentralen Knoten eingehen. A. bei globaler Verfeinerung, B. bei lokaler Verfeinerung in der Nähe einer Kante der Funktion

Die diagnostischen Größen der vier vorgestellten Varianten der bikubischen SLM-Interpolation (reine bikubische Spline-Interpolation, Interpolation mit clipping, quasi-monoton bzw. quasi-konservativ) werden gegenübergestellt. In Abbildung 8.9 sind außerdem die advektierten Funktionen nach fünf Umdrehungen angegeben. Wird lediglich bikubische Spline-Interpolation verwendet, so kommt es an den Kanten des Zylinders zu massivem Über- und Unterschwingen (overshooting/ undershooting). Die in den Abschnitten 5.5.2 und 5.5.3 beschriebenen Verfahren zur monotonen und masseerhaltenden Interpolation ergeben sehr viel ansprechendere Ergebnisse. Die numerische Dissipation ist zwar geringfügig höher, aber Oszillationen verschwinden ganz.

Dabei nimmt die Rechenzeit nicht wesentlich zu. Durch die Adaptivität des Verfahrens verkürzt sich die absolute Rechenzeit sogar, weil weniger Elemente verfeinert werden. Dieses Verhalten ergibt sich aus der Tatsache, daß die Oszillationen eine größere Fläche des Rechengebietes überdecken als der Zylinder selbst.

8.3 Eigenschaften des adaptiven Gitters

Im Abschnitt 8.2 wurde gezeigt, daß die Genauigkeit des Verfahrens fast ausschließlich von der Güte der Interpolation abhängt. Die Interpolation, insbesondere die Lokalisierung der zugrundeliegenden Werte, hängt aber wiederum wesentlich vom Gitter ab.

Bei globaler Gitterverfeinerung hat man an allen Stellen des Rechengebietes eine gleichbleibende Lokalität der Daten, die für die Interpolation herangezogen werden. Das ist bei lokal verfeinerten Gittern nicht unbedingt so. Das Kriterium für Gitterverfeinerung ist der Gradient der advektierten Funktion. Es wird also in der Nähe steiler Kanten verfeinert. Bei der Interpolation eines Wertes in der Nähe einer steilen Kante können daher Unterschiede zwischen lokaler und globaler Verfeinerung auftreten, weil beim lokal verfeinerten Gitter die Knotenwerte möglicherweise weiter vom interpolierten Punkt entfernt liegen. Abbildung 8.10 veranschaulicht dies.

Experimente mit lokaler und globaler Verfeinerung – jeweils mit quasi-konservativer Interpolation – wurden durchgeführt. Die Ergebnisse sind in Abbildung 8.11 gegenübergestellt. Es zeigt

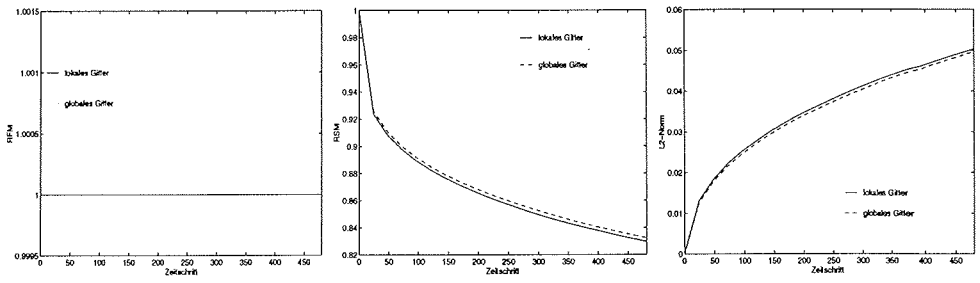

Abbildung 8.11: RFM, RSM und L^2-Norm für fünf Umdrehungen mit global und lokal verfeinertem Gitter und quasi-konservativer Interpolation

sich, daß die lokal verfeinerte Verfahrensvariante tatsächlich nur minimal von der global verfeinerten abweicht. Das Gitter wird in einem Umkreis um die steilen Kanten verfeinert, so daß die Abweichungen gering bleiben.

Drei Zustände des lokal verfeinerten Gitters sind in Abbildung 8.12 abgebildet. Das Gitter folgt den Kanten des geschlitzten Zylinders, wobei die Fläche mit Elementen der feinsten Verfeinerungsstufe mit fortschreitender Integration zunimmt, weil die Zylinderkanten weniger scharf werden.

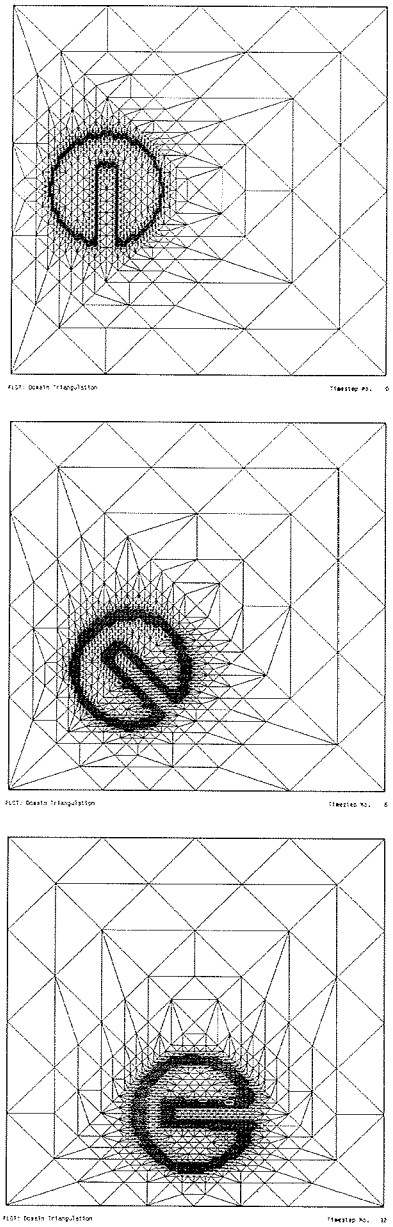

Abbildung 8.12: Das lokal verfeinerte Gitter am Anfang, nach 1/8 und 1/4 Umdrehung

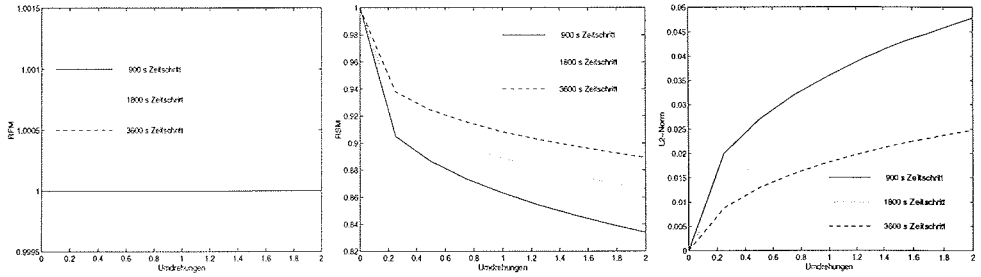

Abbildung 8.13: RFM, RSM und L^2-Norm für zwei Umdrehungen mit quasi-konservativer Interpolation mit exakten Trajektorien, Zeitschrittlänge 900/1800/3600 s

8.4 Absolute Stabilität der SLM

Die Semi-Lagrange-Methode ist absolut, d.h. unabhängig von der Zeitschrittlänge, stabil. Da die passive Advektion mit einem bekannten Wind ein lineares Problem ist, kann mit exakt berechneten Trajektorien die Zeitschrittlänge beliebig gewählt werden. Für die Experimente dieses Abschnitts wird der Zeitschritt 900, 1800 bzw. 3600 Sekunden lang gewählt. Für eine Umdrehung werden also 192, 96 bzw. 48 Zeitschritte benötigt. Da die Anzahl der Interpolationen im letzten Fall am geringsten ist, sind die Rechenergebnisse bei exakter Berechnung der Trajektorien am besten.

Die Abbildungen 8.13 und 8.14 zeigen die Ergebnisse für erste und zweite Momente sowie für die L^2-Norm mit den drei verschiedenen Zeitschrittlängen. Dabei wird im einen Fall mit exakten und im anderen Falle mit den iterierten Trajektorien gerechnet. Wenn die Trajektorien mit Hilfe der Iteration (5.6) berechnet werden, dann darf die Zeitschrittlänge nicht zu groß gewählt werden. Die Berechnung der Anfangspunkte der Trajektorienstücke wird sonst zu ungenau. Für die kurzen Zeitschritte unterscheiden sich die Ergebnisse nicht, bei 3600 Sekunden Zeitschrittlänge ist jedoch die L^2-Norm des Tests mit iterierten Trajektorien deutlich schlechter als mit exakten Trajektorien.

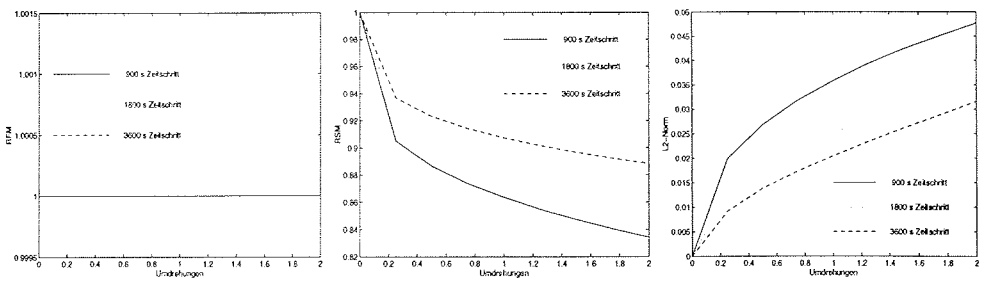

Abbildung 8.14: Wie Abbildung 8.13, jedoch mit iterierten Trajektorien

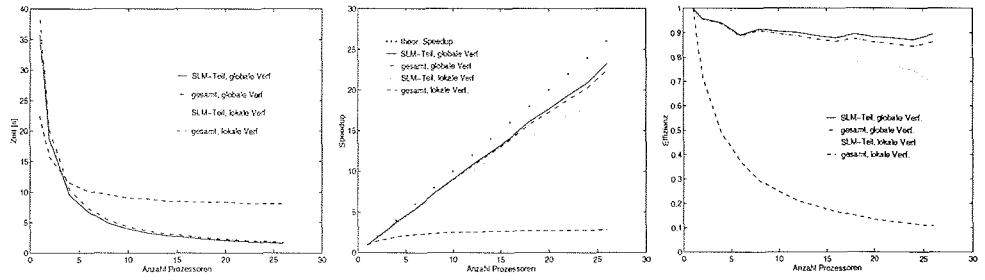

Abbildung 8.15: Zeit für einen Zeitschritt, paralleler Speedup und parallele Effizienz für das Advektionsverfahren, globale und lokale Verfeinerung, SLM-Routinen und Gesamtprogramm

8.5 Parallele Leistung der SLM

Im Kapitel 7 wurde die Möglichkeit der Parallelisierung von SLM beschrieben. Auch die Beschränkung der Effizienz durch die serielle Gittergenerierung wurde erörtert. Die theoretischen Überlegungen des Kapitels 7 werden hier mit experimentellen Daten untermauert.

In Abbildung 8.15 sind die Zeit für einen Zeitschritt des Verfahrens (in Sekunden), der Speedup und die Effizienz für das Advektionsproblem dargestellt. Die Anzahl der Knoten beträgt für den global verfeinerten Fall 33025, die Anzahl der Elemente des feinsten Gitters beträgt 65536. Für den lokal verfeinerten Fall sind 1858 Knotenpunkte und 3695 Elemente vorhanden. In der inneren (adaptiven) Schleife des lokal verfeinerten Verfahrens werden sieben Iterationen benötigt, um zu einem befriedigenden Gitter zu kommen.

Wegen der geringen Knotenzahl ist die Zeit für den SLM-Teil des lokal verfeinerten Verfahrens auch bei sieben inneren Iterationen kleiner als im global verfeinerten Testlauf. Die SLM-Routinen sind im lokal verfeinerten Fall sehr effizient. Unterschiede zum global verfeinerten Lauf ergeben sich vor allem durch die Notwendigkeit der erneuten Datenverteilung, wenn sich das Gitter geändert hat.

Die parallele Leistung der SLM-Routinen ist sehr gut. Sowohl mit lokal verfeinertem Gitter als auch mit globaler Verfeinerung erreichen die parallelisierten SLM-Routinen hohe parallele Effizienz (80-90%). Die Leistung des gesamten Programms ist bei globaler Verfeinerung sehr gut. Eine parallele Effizienz von etwa 85% bei einer Prozessorzahl zwischen 6 und 26 wird erreicht. Das ist auch auf der relativ effizienten KSR-1 ein sehr guter Wert. Die Effizienz ist dabei in diesem Bereich fast konstant. Es ist davon auszugehen, daß das Verfahren global verfeinert gut skaliert. Die angegebenen Zeiten beziehen sich auf den fünften Zeitschritt. Die Datenverteilung ist also schon vorgenommen worden, durch die Index-Arrays müssen Daten nicht mehr bewegt werden. Der erste Schritt der Zeititeration benötigt aufgrund der notwendigen Datenverteilung etwa 10% mehr Zeit als die folgenden Schritte.

In Abbildung 8.16 ist die Lastverteilung auf zehn KSR-1 Prozessoren dargestellt. Jeder Balken in der Graphik veranschaulicht die Arbeitslast eines Prozessors, verschiedene Muster bezeichnen verschiedene Programmsegmente. Die meiste Zeit wird in der Interpolation verbraucht (lange, nach rechts unten schraffierte Balken). Vor der Interpolation ist (als kleines Trapez) die parallelisierte Trajektorienberechnung erkennbar. Nach der Interpolation werden die neuen Knotenwerte ebenfalls parallel berechnet. Gitterroutinen entfallen bei globaler Verfeinerung. Ohne die serielle Gittergenerierung in jedem Zeitschritt kann fast optimale Lastverteilung und Parallelität

erreicht werden.

Die parallele Leistung des Gesamtverfahrens für den lokal verfeinerten Fall ist schlecht. Der Grund liegt in der Dominanz der seriellen Gitterverfeinerungsroutinen. Selbst theoretisch kann die Effizienz des Gesamtverfahrens bei einem seriellen Anteil von ca. 30% nicht über 0,15 liegen (siehe Abschnitt 7.4). In Abbildung 8.17 ist die Lastverteilung und Parallelität des Programmes für lokale Verfeinerung dargestellt. Es ist erkennbar, daß die SLM-Routinen (vor allem die Interpolation) gut parallelisiert sind und die Last gleich verteilt ist. Der serielle Anteil der Gitterroutinen auf Prozessor 0 ist jedoch stark dominierend.

8.6 Zusammenfassung

Eine Implementierung der Semi-Lagrange-Methode für die Advektionsgleichung liegt vor, mit deren Hilfe die Eigenschaften von Interpolation, Gradientenberechnung, Gitteranpassung, und Parallelisierungsstrategie untersucht werden können. Das Modellproblem, der geschlitzte Zylinder als Anfangsbedingung der passiven Advektionsgleichung in zwei Dimensionen, eignet sich für die Ermittlung des numerischen Dispersions- und Dissipationsfehlers, weil sie reibungsfrei und ohne Quellen die Form der Anfangsbedingung erhält.

Das Programm umfaßt etwa 10500 Zeilen Fortran Code. Neben den wesentlichen Routinen des SLM-Algorithmus sind einige Diagnoseroutinen enthalten. Das Programm enthält außerdem die wichtigsten Mechanismen zur Durchführung komplexer Experimente. Modellzustände sind speicherbar, damit das Experiment an beliebigen Punkten neu aufgesetzt werden kann. Einige elementare Graphikroutinen stehen zur Verfügung, mit denen der Lauf des Programms überwacht werden kann, außerdem lassen sich Graphikdaten für die weitere Verarbeitung mit *Matlab* oder *gnuplot* auslesen.

Der neue adaptive Semi-Lagrange-Algorithmus erweist sich als geeignet lokal hohe Auflösung – und damit Genauigkeit – des Verfahrens zu gewährleisten, ohne die Grenzen beschränkter Hardwareressourcen zu überschreiten.

Es kann der Nachweis erbracht werden, daß auch mit irregulären Gittern genaue und effiziente Interpolation und Trajektorienberechnung möglich ist. Gute numerische Ergebnisse können mit bikubischer Spline-Interpolation für die Berechnung stromaufwärts gelegener Werte erzielt werden. Dabei werden Algorithmen verwendet, die in Programmbibliotheken (`netlib`) zur Verfügung stehen.

Die SLM eignet sich besonders für die Parallelisierung, weil alle Berechnungen entlang der Trajektorien unabhängig voneinander sind. Die hervorragende Parallelisierbarkeit kann auch experimentell gezeigt werden. Dabei ist die Strategie, Knoten mit Hilfe von Index-Arrays fest den Prozessoren zuzuweisen, sehr erfolgreich, weil der Kommunikationsaufwand minimiert wird. Im implementierten Verfahren führen lediglich die seriellen Gitterroutinen bei lokaler Gitterverfeinerung auf vielen Prozessoren zu Ineffizienz. Die Parallelisierung der Gitterroutinen ist jedoch nicht Thema dieser Arbeit. Darüberhinaus ist die serielle Dominanz dieses Programmteils im Semi-Lagrange-Verfahren für die Flachwassergleichungen (Teil III) bei moderaten Prozessorzahlen nicht mehr so ausgeprägt.

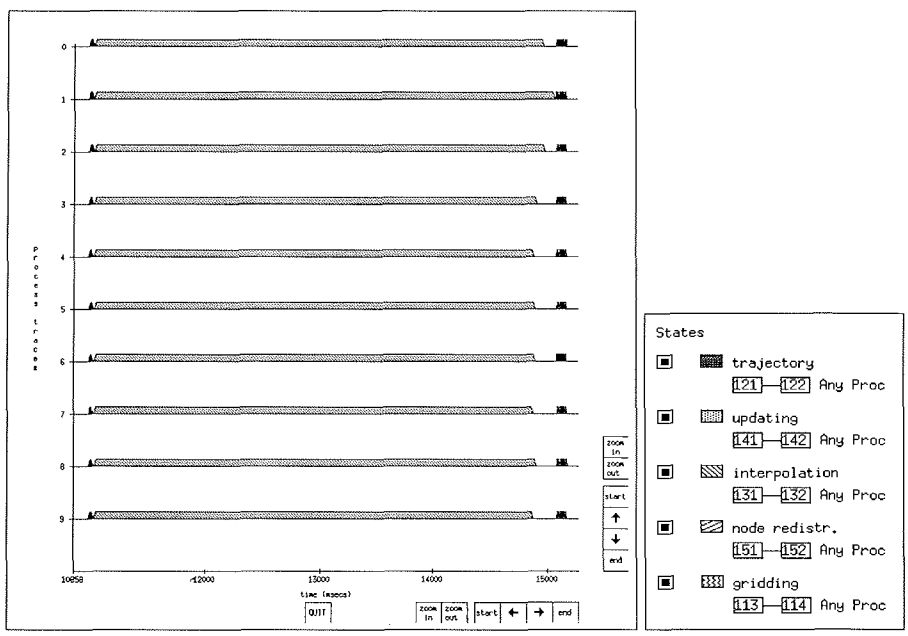

Abbildung 8.16: Darstellung der Lastverteilung und der Parallelität für einen Zeitschritt des Advektionsverfahrens bei globaler Verfeinerung mit einer inneren Iterationen auf 10 Prozessoren der KSR-1

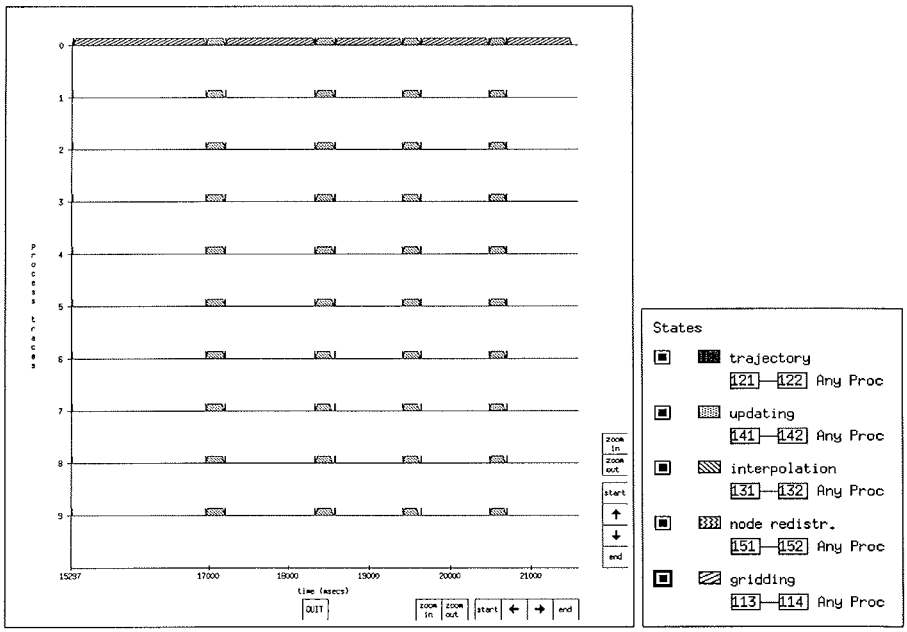

Abbildung 8.17: Wie Abbildung 8.16, jedoch mit lokaler Verfeinerung und vier inneren Iterationen

Teil III

Die Adaptive Semi-Lagrange-Finite-Elemente-Methode (ASLM) für die Flachwassergleichungen

Kapitel 9

Einführung in die adaptive Semi-Lagrange-Finte-Elemente-Methode

9.1 Übersicht

Die üblichen Gleichungen zur Beschreibung großskaliger atmosphärischer oder ozeanischer Zirkulation, die sogenannten *Primitiven (Bewegungs-) Gleichungen*, beschreiben sowohl die horizontalen, wie die vertikalen Abhängigkeiten der Zustandsgrößen. Für diese Gleichungen gibt es verschiedene numerische Lösungsverfahren, die in der Hauptsache auf finiten Differenzen und spektralen Methoden basieren.

Die wesentlichen numerischen Schwierigkeiten bei der Lösung der Primitiven Gleichungen treten aber schon in einem einfacheren zweidimensionalen Zusammenhang, den sogenannten *Flachwassergleichungen* auf. Sie werden daher häufig als Testfall für neue numerische Verfahren verwendet [142].

Die Flachwassergleichungen (oder *Primitiven Gleichungen in einer Schicht*) beschreiben das Verhalten einer rotierenden, homogenen, inkompressiblen und hydrostatischen Flüssigkeit mit einer freien, endlichen Oberflächenhöhe [48]. Sie entsprechen in ihrer mathematischen Form den barotropen Gleichungen, die bei den meisten Lösungsverfahren für die Primitiven Gleichungen auftreten.

Sadourny beschreibt ein FDM-Modell für die Flachwassergleichungen [114]. Das Modell benutzt eine explizite *leap-frog* Zeitdiskretisierung und ein äquidistantes uniformes Rechteckgitter für die räumliche Diskretisierung. Es stellt die einfachste mögliche Diskretisierung der Flachwassergleichungen dar und ist daher unflexibel und nur mit sehr kurzen Zeitschritten zu betreiben. Der Anspruch dieses Modells ist jedoch nicht, als Modellierungssoftware zu dienen. Es ist vielmehr zu akademischen Zwecken erstellt worden.

Ein weiteres bekanntes Modell für die Flachwassergleichungen, das in der Implementierung sehr viel kompletter und als Testmodell für neue numerische Methoden gedacht ist, wird von Hack und Jakob beschrieben [54]. Dieses Modell verwendet die spektrale Transformationsmethode zur Lösung der Gleichungen. Es ist numerisch effizient und präsentiert den aktuellen Stand der Technik in der Modellierungssoftware, die auch für Produktionsmodelle zur Verfügung steht.

Es gibt viele weitere Ansätze zur numerischen Modellierung [145, 69, 76, 44]. Alle Modelle weisen jedoch eine oder mehrere Beschränkungen auf. Dazu gehört, daß sie die räumliche Auflösung nicht den physikalischen Phänomenen anpassen können, daß sie bei der Berechnung der Ränder und Randbedingungen unflexibel sind und daß ihre Zeitschrittlänge durch das CFL-Kriterium beschränkt ist, so daß bei höherer räumlicher Auflösung die Zeitschritte verkürzt werden müssen.

Es ist daher heute nur mit den größten verfügbaren Computern überhaupt möglich, bis in Mesoskalenbereiche hinein aufzulösen.

Die *adaptive Semi-Lagrange-Finite-Elemente-Methode (ASLM)*, die in diesem Teil neu eingeführt wird, setzt an diesem Punkt an. Durch Adaptivität soll die räumliche Auflösung den tatsächlichen physikalischen Prozessen besser angepaßt werden. Durch die Verwendung von FEM für die räumliche Diskretisierung soll die Behandlung von Randbedingungen und die Darstellung von irregulären Rändern vereinfacht werden. Schließlich soll durch die Verwendung der (bedingungslos stabilen) Semi-Lagrange-Methode zur Zeitdiskretisierung die Beschränkung der Zeitschrittlänge gelockert werden.

In den folgenden Abschnitten werden zunächst die Flachwassergleichungen eingeführt. Die Herkunft der Gleichungsterme soll dabei kurz erläutert werden. Die Diskretisierung mit Hilfe von FEM und Semi-Lagrange-Methode wird vorgestellt. Dabei wird von einem vereinfachten Problem in ebenen Koordinaten ausgegangen. Schließlich wird das Modellproblem mit Anfangsbedingungen und ein Referenzmodell mit FDM-Diskretisierung vorgestellt.

9.2 Die Flachwassergleichungen

In diesem Teil sollen die Flachwassergleichungen eingeführt werden. Ausgangspunkt für die Herleitung sind die *Bewegungsgleichungen* (*Impulsgleichungen*) und die *Kontinuitätsgleichung* für eine Flüssigkeit auf der rotierenden Kugel. Dabei werden die Dichte $\rho(\mathbf{x}, t)$ und die Geschwindigkeit $\mathbf{V}(\mathbf{x}, t) = (U(\mathbf{x}, t), V(\mathbf{x}, t), W(\mathbf{x}, t))$ als beschreibende Größen verwendet ($\mathbf{x} = (x, y, z)$ Ortsvektor). Die Kontinuitätsgleichung lautet:

$$\frac{d\rho}{dt} + \rho \nabla \cdot \mathbf{V} = 0 \tag{9.1}$$

Gleichung (9.1) beschreibt die Erhaltung der Masse (der Flüssigkeit). Der erste Term stellt die zeitliche Dichteänderung dar, der zweite Term die Volumenänderung.

Das zeitliche (totale) Differential läßt sich auch in der folgenden Form schreiben (*Eulersche Form*):

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \mathbf{V} \cdot \nabla. \tag{9.2}$$

Den *Bewegungsgleichungen* liegt das Zweite Newtonsche Bewegungsgesetz ($\mathbf{F} = m\mathbf{a}$) zugrunde. Angewendet auf die relevanten Kräfte in Ozean oder Atmosphäre lauten sie (in Vektorform):

$$\frac{d\mathbf{V}}{dt} = -\frac{1}{\rho}\nabla p - 2\Omega \times \mathbf{V} + g + F. \tag{9.3}$$

Die Aussage der Gleichungen ist: Die zeitliche Änderung der Geschwindigkeit wird bestimmt durch die Druckänderung, die Corioliskraft, die Gravitationskraft und andere zusammengefaßte Kräfte (z.B. Reibungskraft). Dabei ist Ω die Winkelgeschwindigkeit der Erde, p der Druck und F die Summe weiterer Beschleunigungen (siehe [48, 93, 102, 105]).

Für die Herleitung der Flachwassergleichungen wird eine Flüssigkeit von konstanter und uniformer Dichte angenommen. Die Höhe der Flüssigkeitsschicht über einer angenommenen Nullinie $z = 0$ ist $h(x, y, t)$. Die Bodentopographie wird durch $h_B(x, y)$ beschrieben. Es gebe eine charakteristische Größe H für die Höhenausdehnung $(h - h_B)$ der Flüssigkeit (z.B. die gemittelte Höhe). H charakterisiert auch die Größenordnung der vertikalen Geschwindigkeitskomponente. Außerdem sei L eine charakteristische Größe für die horizontalen Geschwindigkeitskomponenten (siehe Abb. 9.1).

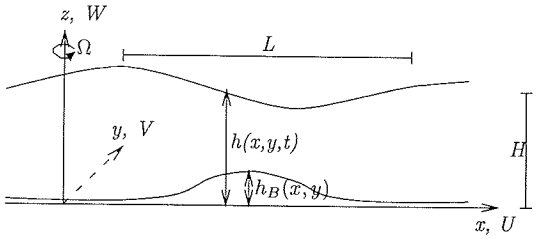

Abbildung 9.1: Annahmen des Flachwassermodells

Die wesentliche Annahme (neben den Annahmen über die Eigenschaften der Flüssigkeit), die den Flachwassergleichungen zugrunde liegt, lautet:

$$\delta = \frac{H}{L} \ll 1.$$

Das heißt, die Vertikalgeschwindigkeit ist gegüber der Horizontalgeschwindigkeit vernachlässigbar.

Eine Abschätzung der Größenordnungen der Terme ergibt dann die folgenden Gleichungen (Komponentenform):

$$\begin{aligned} \frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + V\frac{\partial U}{\partial y} - fV + g\frac{\partial h}{\partial x} &= 0, \\ \frac{\partial V}{\partial t} + U\frac{\partial V}{\partial x} + V\frac{\partial V}{\partial y} + fU + g\frac{\partial h}{\partial y} &= 0, \\ \frac{\partial h}{\partial t} + \frac{\partial (h-h_B)U}{\partial x} + \frac{\partial (h-h_B)V}{\partial y} &= 0. \end{aligned} \tag{9.4}$$

Dabei ist der *Coriolisparameter* f gegeben durch $f = 2\Omega \sin y$ (in ebenen Koordinaten ist f =const.). Statt der Höhe h wird die *geopotentielle Höhe* Φ

$$\Phi = g(h - h_B),$$

verwendet. Weiterhin wird ein flacher Boden angenommen. Die Gleichungen (9.4) haben dann die folgende Form und heißen *Flachwassergleichungen*:

$$\begin{aligned} \frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + V\frac{\partial U}{\partial y} - fV + \frac{\partial \Phi}{\partial x} &= 0, \\ \frac{\partial V}{\partial t} + U\frac{\partial V}{\partial x} + V\frac{\partial V}{\partial y} + fU + \frac{\partial \Phi}{\partial y} &= 0, \\ \frac{\partial \Phi}{\partial t} + U\frac{\partial \Phi}{\partial x} + V\frac{\partial \Phi}{\partial y} + \Phi\left[\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y}\right] &= 0. \end{aligned} \tag{9.5}$$

9.3 Diskretisierung der Flachwassergleichungen mit Hilfe der Semi-Lagrange-Methode und der finiten Elemente

In diesem Abschnitt werden die Flachwassergleichungen diskretisiert. Die Gleichungen werden dabei in einer modifizierten Form verwendet, die weitere Quellterme enthalten kann. Diese Terme sind sowohl implizit als auch explizit in die Gleichungen integrierbar.

Mehrere Schritte sind für die Diskretisierung notwendig. Zunächst wird die semi-implizite Semi-Lagrange-Formulierung der Gleichungen eingeführt. Dabei werden die Ergebnisse des Abschnittes 5.3 verwendet, um die Gleichungen aufzustellen.

Die diskreten Gleichungen werden so modifiziert, daß man eine elliptische Differentialgleichung für die geopotentielle Höhe erhält. Die Lösung des elliptischen Problems dient dann zur Berechnung der Geschwindigkeitskomponenten. Das elliptische Problem wird mittels der FEM gelöst. Dazu muß die schwache Formulierung gefunden werden.

Die Flachwassergleichungen seien auf dem Rechteck $G = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \in \mathcal{R}^2$ gegeben. Die Gleichungen lauten unter Verwendung des totalen Differentials (9.2):

$$\frac{dU}{dt} + \Phi_x - fV = 0, \tag{9.6}$$

$$\frac{dV}{dt} + \Phi_y + fU = 0, \tag{9.7}$$

$$\frac{d\Phi}{dt} + \Phi(U_x + V_y) = 0. \tag{9.8}$$

Dabei ist $\mathbf{V} = (U, V)$ der Geschwindigkeitsvektor, $f = 2\Omega \sin y$ der Coriolisparameter, (x, y) Länge, bzw. Breite, Ω die Erdrotation, und Φ die geopotentielle Höhe. X_v bezeichne die partielle Ableitung $\frac{\partial X}{\partial v}$ mit $v = x, y$, $X = U, V, \Phi$.

Wenn zusätzliche Terme in die Impulsgleichungen (9.6), (9.7) eingehen und die rechten Seiten inhomogen angenommen werden, könnte man die Gleichungen modifizieren zu:

$$\frac{dU}{dt} + \Phi_x - fV + A = R_1, \tag{9.9}$$

$$\frac{dV}{dt} + \Phi_y + fU + B = R_2, \tag{9.10}$$

$$\frac{d\Phi}{dt} + \Phi(U_x + V_y) = R_3. \tag{9.11}$$

$A = A(x, y, t)$ und $B = B(x, y, t)$ seien unabhängig von U, V, Φ, $R_k = R_k(x, y, t)$, $k = 1, 2, 3$.

Die Kontinuitätsgleichung (9.8) ist äquivalent zu der folgenden Formulierung:

$$\frac{d \ln \Phi}{dt} + (U_x + V_y) = 0. \tag{9.12}$$

Die Semi-Lagrange-Form der Flachwassergleichungen wird für die Gleichungen (9.9) bis (9.11) angegeben. Aus den Ergebnissen des Abschnitts 5.3 erhält man die diskrete Form der Gleichungen:

$$\frac{U^+ - U^\circ}{\Delta t} + \frac{\Phi_x^+ + \Phi_x^\circ}{2} - \frac{(fV)^+ + (fV)^\circ}{2} + \frac{A^+ + A^\circ}{2} = R_1^{(1/2)} \tag{9.13}$$

$$\frac{V^+ - V^\circ}{\Delta t} + \frac{\Phi_y^+ + \Phi_y^\circ}{2} + \frac{(fU)^+ + (fU)^\circ}{2} + \frac{B^+ + B^\circ}{2} = R_2^{(1/2)} \tag{9.14}$$

$$\frac{\Phi^+ - \Phi^\circ}{\Delta t} + \frac{\Phi^+(U_x^+ + V_y^+) + \Phi^\circ(U_x^\circ + V_y^\circ)}{2} = R_3^{(1/2)}. \tag{9.15}$$

Dabei steht X^+ für den Wert der Größe am Gitterpunkt zur neuen Zeit $X(x, t + \Delta t)$ und X° für den Wert am stromaufwärts gelegenen Punkt zur alten Zeit $X(x - \alpha, t)$. Die Terme auf der linken Seite werden implizit behandelt.

Die numerische Behandlung von (9.15) bereitet Schwierigkeiten, weil durch die implizite Behandlung der geopotentiellen Höhe Φ in der unten hergeleiteten elliptischen Gleichung ein schwach

nichtlinearer Term auftritt. Daher wird häufig eine semi-implizite Formulierung der Kontinuitätsgleichung gewählt. Zwei verschiedene semi-implizite Formen von (9.15) können angegeben werden. Die erste Form berücksichtigt zwar die Divergenz in impliziter Form, die geopotentielle Höhe wird aber explizit verwendet:

$$\frac{\Phi^+ - \Phi^\circ}{\Delta t} + \frac{\Phi^\circ(U_x^+ + V_y^+) + \Phi^\circ(U_x^\circ + V_y^\circ)}{2} = R_3^{(1/2)}. \tag{9.16}$$

In der zweiten Version wird Φ in zwei Komponenten, einen konstanten mittleren Wert $\bar{\Phi}$ und eine zeitabhängige Abweichung $\Phi' = \Phi'(x, y, t)$ aufgespalten:

$$\Phi(x, y, t) = \bar{\Phi} + \Phi'(x, y, t). \tag{9.17}$$

Dabei wird angenommen, daß $\Phi' \ll \bar{\Phi}$. Diese Darstellung hat auch Auswirkungen auf die Form der Flachwassergleichungen, weil die Ableitungen lediglich von Φ' abhängen:

$$\frac{dU}{dt} + \Phi'_x - fV + A = R_1, \tag{9.18}$$

$$\frac{dV}{dt} + \Phi'_y + fU + B = R_2, \tag{9.19}$$

$$\frac{d\Phi'}{dt} + (\bar{\Phi} + \Phi')(U_x + V_y) = R_3. \tag{9.20}$$

Die semi-implizite Formulierung der Kontinuitätsgleichung wird gewonnen, indem Φ' explizit verwendet wird:

$$\frac{\Phi'^+ - \Phi'^\circ}{\Delta t} + \Phi'^\circ(U_x^\circ + V_y^\circ) + \frac{\bar{\Phi}(U_x^+ + V_y^+) + \bar{\Phi}(U_x^\circ + V_y^\circ)}{2} = R_3^{(1/2)}. \tag{9.21}$$

Die zugehörigen diskretisierten Impulsgleichungen lauten:

$$\frac{U^+ - U^\circ}{\Delta t} + \frac{\Phi_x'^+ + \Phi_x'^\circ}{2} - \frac{(fV)^+ + (fV)^\circ}{2} + \frac{A^+ + A^\circ}{2} = R_1^{(1/2)} \tag{9.22}$$

$$\frac{V^+ - V^\circ}{\Delta t} + \frac{\Phi_y'^+ + \Phi_y'^\circ}{2} + \frac{(fU)^+ + (fU)^\circ}{2} + \frac{B^+ + B^\circ}{2} = R_2^{(1/2)} \tag{9.23}$$

Um nun ein elliptisches Problem zu erhalten, ist folgende Vorgehensweise möglich:

1. Setze die Impulsgleichungen (9.13) und (9.14) wechselseitig ein.
2. Bilde die Divergenz der beiden Gleichungen und erhalte einen Ausdruck für $(U_x^+ + V_y^+)$.
3. Setze die so erhaltene rechte Seite in (9.16) oder (9.21) ein und erhalte ein elliptisches Problem für Φ^+.

Die Umformung von (9.13) und (9.14) ergibt:

$$U^+ + \phi\frac{\Delta t}{2}\Phi_x^+ + \varphi\frac{\Delta t}{2}\Phi_y^+ + \varphi\frac{\Delta t}{2}B^+ + \phi\frac{\Delta t}{2}A^+ = \phi r_1 + \varphi r_2 \tag{9.24}$$

$$V^+ + \phi\frac{\Delta t}{2}\Phi_y^+ - \varphi\frac{\Delta t}{2}\Phi_x^+ - \varphi\frac{\Delta t}{2}A^+ + \phi\frac{\Delta t}{2}B^+ = \phi r_2 - \varphi r_1 \tag{9.25}$$

Dabei ist:

$$\begin{aligned}
\phi &:= \frac{1}{[1 + (\frac{\Delta t}{2})^2 f^2]} \\
\varphi &:= \frac{\Delta t}{2} f \cdot \phi \\
r_1 &:= U^\circ + \frac{\Delta t}{2} f V^\circ - \frac{\Delta t}{2}\Phi_x^\circ - \frac{\Delta t}{2}A^\circ + \Delta t\, R_1^{(1/2)} \\
r_2 &:= V^\circ - \frac{\Delta t}{2} f U^\circ - \frac{\Delta t}{2}\Phi_y^\circ - \frac{\Delta t}{2}B^\circ + \Delta t\, R_2^{(1/2)}
\end{aligned}$$

Die Divergenz ergibt:

$$\begin{aligned} U_x^+ + V_y^+ &= -\frac{\Delta t}{2}\left[(\phi\Phi_x^+)_x - \left(\phi\Phi_y^+\right)_y - \left(\varphi\Phi_y^+\right)_x + (\varphi\Phi_x^+)_y\right] \\ &\quad -\frac{\Delta t}{2}\left[(\varphi B^+)_x - (\phi B^+)_y + (\varphi A^+)_y - (\phi A^+)_x\right] \\ &\quad + (\phi r_1)_x + (\varphi r_2)_x + (\phi r_2)_y - (\varphi r_1)_y\,. \end{aligned} \tag{9.26}$$

Einsetzen in (9.16) ergibt:

$$\begin{aligned} (\phi\Phi_x^+)_x + \left(\phi\Phi_y^+\right)_y + \left(\varphi\Phi_y^+\right)_x - (\varphi\Phi_x^+)_y \;&-\; \frac{4\,\Phi^+}{\bar{\Phi}^\circ(\Delta t)^2} = \\ &-\; (\phi A^+)_x + (\varphi A^+)_y - (\varphi B^+)_x - (\phi B^+)_y \\ &+\; \frac{2}{\Delta t}\left[(\phi r_1)_x + (\varphi r_2)_x + (\phi r_2)_y - (\varphi r_1)_y\right] \\ &+\; \frac{4}{(\Delta t)^2}\left[\Delta t(U_x^\circ + V_y^\circ) - 1 - \frac{\Delta t\; R_3^{(1/2)}}{\Phi^\circ}\right]. \end{aligned} \tag{9.27}$$

Einsetzen in die zweite Gleichung (9.21) ergibt:

$$\begin{aligned} (\phi\Phi_x'^+)_x + \left(\phi\Phi_y'^+\right)_y + \left(\varphi\Phi_y'^+\right)_x - (\varphi\Phi_x'^+)_y \;&-\; \frac{4}{\bar{\Phi}(\Delta t)^2}\Phi'^+ = \\ &-\; (\phi A^+)_x - (\phi B^+)_y - (\varphi B^+)_x + (\varphi A^+)_y \\ &+\; \frac{2}{\Delta t}\left[(\phi r_1)_x + (\phi r_2)_y + (\varphi r_2)_x - (\varphi r_1)_y\right] \\ &+\; \frac{2\Phi'^\circ}{\bar{\Phi}\Delta t}(U_x^\circ + V_y^\circ) - \frac{4\Phi'^\circ}{\bar{\Phi}(\Delta t)^2} - \frac{4}{\bar{\Phi}\Delta t}R_3^{(1/2)}. \end{aligned} \tag{9.28}$$

Die Diskretisierung von (9.27) bzw. (9.28) mit Hilfe der FEM erfolgt, indem mit den FEM-Basisfunktionen skalar multipliziert wird (siehe Abschnitt 1.2).

Das Problem (9.27) bzw. (9.28) sei in vereinfachter Form gegeben:

$$(\phi\Phi_x)_x + (\phi\Phi_y)_y - (\varphi\Phi_x)_y + (\varphi\Phi_y)_x - \chi\Phi = R \tag{9.29}$$

Multiplikation mit der Testfunktion b und Integration über dem Rechengebiet $\mathcal{G}$ ergibt:

$$\int_{\mathcal{G}}\left[(\phi\Phi_x)_x + (\phi\Phi_y)_y - (\varphi\Phi_x)_y + (\varphi\Phi_y)_x - \chi\Phi\right]\cdot b\; d\mathcal{G} = \int_{\mathcal{G}} R\cdot b\; d\mathcal{G}. \tag{9.30}$$

Mit Hilfe der FEM-Darstellung der Funktion Φ,

$$\Phi(x,y) = \sum_i \Phi_i b_i(x,y), \tag{9.31}$$

wobei Φ_i Koeffizienten, b_i FEM-Basisfunktion zum Knoten i, und $i = 1,\ldots,N$ für die Knoten, sowie partieller Integration erhält man ein lineares Gleichungssystem,

$$\mathbf{A}\boldsymbol{\Phi} = \mathbf{f}, \tag{9.32}$$

wobei

$$\begin{aligned} \mathbf{A} &:= (\mathrm{a}(b_i,b_j))_{i,j=1,\ldots,N}\,, \\ \mathbf{f} &:= (f(b_j))_{j=1,\ldots,N}^\top\,, \\ \boldsymbol{\Phi} &:= (\Phi_j)_{j=1,\ldots,N}^\top\,, \end{aligned}$$

und

$$\begin{aligned} a(b_i, b_j) &:= \int_{\mathcal{G}} -\phi(\nabla b_i \cdot \nabla b_j) + b(\nabla b_i \cdot \tilde{\nabla} b_j) - \chi(b_i b_j)\, d\mathcal{G}, \\ f(b_j) &:= \int_{\mathcal{G}} R_j b_j\, d\mathcal{G}. \end{aligned}$$

Dabei seien die Operatoren ∇ und $\tilde{\nabla}$ definiert:

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) \qquad \tilde{\nabla} = \left(\frac{\partial}{\partial y}, -\frac{\partial}{\partial x}\right)$$

Der Ausdruck $a(b_i, b_j)$ kann weiter vereinfacht werden. Das Integral über das Gebiet $\mathcal{G}$ läßt sich in die Summe der Integrale über die Dreiecke τ der Triangulierung T zerlegen:

$$a(b_i, b_j) = \sum_{\tau \in T} \int_{\tau} -\phi(\nabla b_i \cdot \nabla b_j) + \varphi(\nabla b_i \cdot \tilde{\nabla} b_j) - \chi(b_i b_j)\, dT$$

Da die Gradienten der Basisfunktionen b_i lokal konstant sind, läßt sich die Summe auch schreiben:

$$a(b_i, b_j) = \sum_{\tau \in T} \left[-(\nabla b_i \cdot \nabla b_j) \int_{\tau} \phi\, dT + (\nabla b_i \cdot \tilde{\nabla} b_j) \int_{\tau} \varphi\, dT - \int_{\tau} \chi(b_i b_j)\, dT \right]$$

Das Gleichungssystem (9.32) kann mit einem geeigneten Löser (z.B. BiCGSTAB-Verfahren mit Präkonditionierung) gelöst werden.

9.4 Dimensionslose Gleichungen

Die Gleichungen des letzten Abschnitts können zu numerischen Instabilitäten führen, weil die Terme in den Gleichungen sehr verschiedene Größenordnungen aufweisen. Eine Möglichkeit diese Unausgewogenheit auszugleichen, besteht darin, charakteristische Größen einzuführen, und die Gleichungen damit so zu skalieren, daß die Unbekannten in der Größenordnung $\mathcal{O}(1)$ liegen. Werden die charakteristischen Größen mit Dimensionen behaftet und mit den Termen skaliert, so lassen sich dimensionslose Gleichungen formulieren.

Zunächst werden die Dimensionen der einzelnen Terme der Geichungen (9.9)–(9.11) untersucht. Für (9.9) gilt:

$$\left.\begin{aligned} U = V &= [ms^{-1}] \\ \Phi &= [m^2 s^{-2}] \\ f &= [s^{-1}] \end{aligned}\right\} \Longrightarrow \frac{dU}{dt} = \Phi_x = fV = \left[ms^{-2}\right] \Rightarrow A = R_1 = \left[ms^{-2}\right].$$

Analoges gilt für (9.10); für (9.11) ergibt die Analyse der Dimensionen:

$$\frac{d\Phi}{dt} = \Phi \cdot U_x = \left[m^2 s^{-3}\right] \Rightarrow R_3 = \left[m^2 s^{-3}\right].$$

Mit diesen Informationen können nun charakteristische Größen für die verschiedenen Variablen angesetzt werden. Damit kann dann jede Variable mit Hilfe einer dimensionslosen Variable und der zugehörigen charakteristischen Größe dargestellt werden:

$$\begin{aligned} (x, y) &= L(\hat{x}, \hat{y}) & t &= T\hat{t} \\ U &= \tfrac{L}{T}\hat{U} & V &= \tfrac{L}{T}\hat{V} \\ \Phi &= GH\hat{\Phi} & f &= F\hat{f} \end{aligned}$$

Für die zusätzlichen Antriebsterme müssen entsprechende charakteristische Größen gefunden werden, hier werden zunächst folgende Parametrisierungen angenommen, die sich aus der Analyse der Dimensionen ergeben:

$$\begin{aligned} A &= \tfrac{L}{T^2}\hat{A} \qquad B = \tfrac{L}{T^2}\hat{B} \\ R_1 &= \tfrac{L}{T^2}\hat{R}_1 \qquad R_2 = \tfrac{L}{T^2}\hat{R}_2 \\ R_3 &= \tfrac{L^2}{T^3}\hat{R}_3 \end{aligned}$$

Damit können nun die Gleichungen (9.9)–(9.11) in der folgenden Form geschrieben werden:

$$\left[\frac{L}{T^2}\right]\frac{d\hat{U}}{d\hat{t}} + \left[\frac{GH}{L}\right]\hat{\Phi}_{\hat{x}} - \left[\frac{FL}{T}\right]\hat{f}\hat{V} + \left[\frac{L}{T^2}\right]\hat{A} = \left[\frac{L}{T^2}\right]\hat{R}_1, \tag{9.33}$$

$$\left[\frac{L}{T^2}\right]\frac{d\hat{V}}{d\hat{t}} + \left[\frac{GH}{L}\right]\hat{\Phi}_{\hat{y}} + \left[\frac{FL}{T}\right]\hat{f}\hat{U} + \left[\frac{L}{T^2}\right]\hat{B} = \left[\frac{L}{T^2}\right]\hat{R}_2, \tag{9.34}$$

$$\left[\frac{GH}{T}\right]\frac{d\hat{\Phi}}{d\hat{t}} + [GH]\hat{\Phi}\left(\left[\frac{L}{TL}\right]\hat{U}_{\hat{x}} + \left[\frac{L}{TL}\right]\hat{V}_{\hat{y}}\right) = \left[\frac{L^2}{T^3}\right]\hat{R}_3. \tag{9.35}$$

Nach Umformung der Faktoren zu ebenfalls dimensionslosen Zahlen ergibt sich das folgende dimensionslose Gleichungssystem:

$$\frac{d\hat{U}}{d\hat{t}} + \left[\frac{T^2GH}{L^2}\right]\hat{\Phi}_{\hat{x}} - [FT]\hat{f}\hat{V} + \hat{A} = \hat{R}_1, \tag{9.36}$$

$$\frac{d\hat{V}}{d\hat{t}} + \left[\frac{T^2GH}{L^2}\right]\hat{\Phi}_{\hat{y}} + [FT]\hat{f}\hat{U} + \hat{B} = \hat{R}_2, \tag{9.37}$$

$$\frac{d\hat{\Phi}}{d\hat{t}} + \hat{\Phi}(\hat{U}_{\hat{x}} + \hat{V}_{\hat{y}}) = \left[\frac{L^2}{T^2GH}\right]\hat{R}_3. \tag{9.38}$$

Die diskretisierte Form der Gleichungen (9.36)–(9.38) lautet analog zur Darstellung in (9.13)–(9.15):

$$\frac{\hat{U}^+ - \hat{U}^\circ}{\Delta\hat{t}} + \lambda\frac{\hat{\Phi}^+_{\hat{x}} + \hat{\Phi}^\circ_{\hat{x}}}{2} - \mu\frac{(\hat{f}\hat{V})^+ + (\hat{f}\hat{V})^\circ}{2} + \frac{\hat{A}^+ + \hat{A}^\circ}{2} = \hat{R}_1^{(1/2)} \tag{9.39}$$

$$\frac{\hat{V}^+ - \hat{V}^\circ}{\Delta\hat{t}} + \lambda\frac{\hat{\Phi}^+_{\hat{y}} + \hat{\Phi}^\circ_{\hat{y}}}{2} + \mu\frac{(\hat{f}\hat{U})^+ + (\hat{f}\hat{U})^\circ}{2} + \frac{\hat{B}^+ + \hat{B}^\circ}{2} = \hat{R}_2^{(1/2)} \tag{9.40}$$

$$\frac{\hat{\Phi}^+ - \hat{\Phi}^\circ}{\Delta\hat{t}} + \frac{\hat{\Phi}^+(\hat{U}^+_{\hat{x}} + \hat{V}^+_{\hat{y}}) + \hat{\Phi}^\circ(\hat{U}^\circ_{\hat{x}} + \hat{V}^\circ_{\hat{y}})}{2} = \lambda^{-1}\hat{R}_3^{(1/2)}. \tag{9.41}$$

Dabei seien

$$\begin{aligned} \lambda &= \left[\frac{T^2GH}{L^2}\right], \\ \mu &= [FT]. \end{aligned}$$

Entsprechend sind die semi-impliziten Formen zu modifizieren. Auch die Formulierung mit aufgespaltetem Φ lassen sich in analoger Weise schreiben.

Das elliptische Problem gewinnt man wiederum analog zur Vorgehensweise im letzten Abschnitt. Die modifizierten Gleichungen, die aus der Umformung von (9.39) und (9.40) hervorgehen, lauten:

$$\hat{U}^+ = -\hat{\phi}\frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}^+_{\hat{x}} - \hat{\varphi}\frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}^+_{\hat{y}} - \hat{\phi}\frac{\Delta\hat{t}}{2}\hat{A}^+ - \hat{\varphi}\frac{\Delta\hat{t}}{2}\hat{B}^+ + \hat{\phi}\hat{r}_1 + \hat{\varphi}\hat{r}_2, \tag{9.42}$$

$$\hat{V}^+ = -\hat{\phi}\frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}^+_{\hat{y}} + \hat{\varphi}\frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}^+_{\hat{x}} - \hat{\phi}\frac{\Delta\hat{t}}{2}\hat{B}^+ + \hat{\varphi}\frac{\Delta\hat{t}}{2}\hat{A}^+ + \hat{\phi}\hat{r}_2 - \hat{\varphi}\hat{r}_1, \tag{9.43}$$

wobei jetzt

$$\begin{aligned}
\hat{\phi} &:= \frac{1}{[1+(\frac{\mu\Delta\hat{t}}{2})^2\hat{f}^2]}, \\
\hat{\varphi} &:= \frac{\mu\Delta\hat{t}}{2}\hat{f}\cdot\hat{\phi}, \\
\hat{r}_1 &:= \hat{U}^\circ + \frac{\mu\Delta\hat{t}}{2}\hat{f}\hat{V}^\circ - \frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}^\circ_{\hat{x}} - \frac{\Delta\hat{t}}{2}\hat{A}^\circ + \Delta\hat{t}\,\hat{R}_1^{(1/2)}, \\
\hat{r}_2 &:= \hat{V}^\circ - \frac{\mu\Delta\hat{t}}{2}\hat{f}\hat{U}^\circ - \frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}^\circ_{\hat{y}} - \frac{\Delta\hat{t}}{2}\hat{B}^\circ + \Delta\hat{t}\,\hat{R}_2^{(1/2)}.
\end{aligned}$$

Das elliptische Problem (9.27) in der dimensionslosen Form lautet:

$$\begin{aligned}
\left(\hat{\phi}\hat{\Phi}^+_{\hat{x}}\right)_{\hat{x}} + \left(\hat{\phi}\hat{\Phi}^+_{\hat{y}}\right)_{\hat{y}} + \left(\hat{\varphi}\hat{\Phi}^+_{\hat{y}}\right)_{\hat{x}} &- \left(\hat{\varphi}\hat{\Phi}^+_{\hat{x}}\right)_{\hat{y}} - \frac{4}{\hat{\Phi}^\circ\lambda(\Delta\hat{t})^2}\hat{\Phi}^+ = \\
&- \frac{1}{\lambda}\left[\left(\hat{\phi}\hat{A}^+\right)_{\hat{x}} + \left(\hat{\phi}\hat{B}^+\right)_{\hat{y}} + \left(\hat{\varphi}\hat{B}^+\right)_{\hat{x}} - \left(\hat{\varphi}\hat{A}^+\right)_{\hat{y}}\right] \\
&+ \frac{2}{\Delta\hat{t}\lambda}\left[\left(\hat{\phi}\hat{r}_1\right)_{\hat{x}} + \left(\hat{\phi}\hat{r}_2\right)_{\hat{y}} + (\hat{\varphi}\hat{r}_2)_{\hat{x}} - (\hat{\varphi}\hat{r}_1)_{\hat{y}}\right] \\
&+ \frac{2}{\Delta\hat{t}\lambda}(\hat{U}^\circ_{\hat{x}} + \hat{V}^\circ_{\hat{y}}) - \frac{4}{(\Delta\hat{t})^2\lambda} - \frac{4}{\hat{\Phi}^\circ\lambda^2\Delta\hat{t}}\hat{R}_3^{(1/2)}.
\end{aligned} \tag{9.44}$$

Wenn zusätzlich Φ aufgespalten wird, können die dimensionslosen Gleichungen formuliert werden:

$$\begin{aligned}
\left(\hat{\phi}\hat{\Phi}'^+_{\hat{x}}\right)_{\hat{x}} + \left(\hat{\phi}\hat{\Phi}'^+_{\hat{y}}\right)_{\hat{y}} + \left(\hat{\varphi}\hat{\Phi}'^+_{\hat{y}}\right)_{\hat{x}} &- \left(\hat{\varphi}\hat{\Phi}'^+_{\hat{x}}\right)_{\hat{y}} - \frac{4}{\hat{\bar{\Phi}}\lambda(\Delta\hat{t})^2}\hat{\Phi}'^+ = \\
&- \frac{1}{\lambda}\left[\left(\hat{\phi}\hat{A}^+\right)_{\hat{x}} + \left(\hat{\phi}\hat{B}^+\right)_{\hat{y}} + \left(\hat{\varphi}\hat{B}^+\right)_{\hat{x}} - \left(\hat{\varphi}\hat{A}^+\right)_{\hat{y}}\right] \\
&+ \frac{2}{\Delta\hat{t}\lambda}\left[\left(\hat{\phi}\hat{r}_1\right)_{\hat{x}} + \left(\hat{\phi}\hat{r}_2\right)_{\hat{y}} + (\hat{\varphi}\hat{r}_2)_{\hat{x}} - (\hat{\varphi}\hat{r}_1)_{\hat{y}}\right] \\
&+ \frac{4\hat{\Phi}'^\circ + 2\hat{\bar{\Phi}}}{\hat{\bar{\Phi}}\lambda\Delta\hat{t}}(\hat{U}^\circ_{\hat{x}} + \hat{V}^\circ_{\hat{y}}) - \frac{4\hat{\Phi}'^\circ}{\hat{\bar{\Phi}}\lambda(\Delta\hat{t})^2} - \frac{4}{\hat{\bar{\Phi}}\lambda^2\Delta\hat{t}}\hat{R}_3^{(1/2)},
\end{aligned} \tag{9.45}$$

wobei in diesem Fall für $\hat{r}_1$ und $\hat{r}_2$ gilt:

$$\hat{r}_1 := \hat{U}^\circ + \frac{\mu\Delta\hat{t}}{2}\hat{f}\hat{V}^\circ - \frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}'^\circ_{\hat{x}} - \frac{\Delta\hat{t}}{2}\hat{A}^\circ + \Delta\hat{t}\,\hat{R}_1^{(1/2)}, \tag{9.46}$$

$$\hat{r}_2 := \hat{V}^\circ - \frac{\mu\Delta\hat{t}}{2}\hat{f}\hat{U}^\circ - \frac{\lambda\Delta\hat{t}}{2}\hat{\Phi}'^\circ_{\hat{y}} - \frac{\Delta\hat{t}}{2}\hat{B}^\circ + \Delta\hat{t}\,\hat{R}_2^{(1/2)}. \tag{9.47}$$

9.5 Modellproblem und numerisches Referenzmodell

In den Vorschlägen von Williams et al. [54] ist ein Standardtest unter dem Namen *Rossby-Haurwitz Wellen* angegeben. Diese stehenden Wellen stellen eine analytische Lösung der nichtlinearen barotropen Vorticitygleichung auf der Sphäre dar. Die Anfangsbedingung ist durch die folgende Stromfunktion und die zugehörige geopotentielle Höhe gegeben (in sphärischen Koordinaten (θ, λ, r)):

$$\Psi_0 = -a^2\omega\sin\theta + a^2K\cos^R\theta\sin\theta\sin R\lambda. \tag{9.48}$$

$$\Phi_0 = a^2A(\theta) + a^2B(\theta)\cos R\lambda + a^2C(\theta)\cos 2R\lambda + \bar{\Phi} \tag{9.49}$$

Dabei ist a der Erdradius, ω, R und K sind Konstanten und A, B und C sind gegeben durch:

$$\begin{aligned} A(\theta) &= \frac{\omega}{2}(2\Omega+\omega)\cos^2\theta + \frac{1}{4}K^2\cos^{2R}\theta\left[(R+1)\cos^2\theta + (2R^2-R-2) - 2R^2\cos^{-2}\theta\right], \\ B(\theta) &= \frac{2(\Omega+\omega)K}{(R+1)(R+2)}\cos^R\theta\left[(R^2+2R+2) - (R+1)^2\cos^2\theta\right], \\ C(\theta) &= \frac{1}{4}K^2\cos^{2R}\theta\left[(R+1)\cos^2\theta - (R+2)\right]. \end{aligned}$$

Die Anfangswerte der Geschwindigkeitskomponenten können mit Hilfe des Gradienten der Stromfunktion berechnet werden:

$$\begin{aligned} U &= -\frac{\partial\Psi}{\partial y} \\ V &= \frac{\partial\Psi}{\partial x} \end{aligned}$$

Referenzergebnisse des Modellproblems sind in [66] angegeben.

Sadourny beschreibt ein FDM-Modell mit ähnlichen Anfangsbedingungen, die jedoch in ebenen karthesischen Koordinaten gegeben sind [114]. Diese Anfangsbedingungen werden als Modellproblem für die numerischen Tests der adaptiven Semi-Lagrange-Finite-Elemente-Methode verwendet. Sie sind gegeben durch (K_1, K_2 Konstanten):

$$\Psi_0 = K_1 \sin x \sin y \tag{9.50}$$

$$\Phi_0 = \pi^2 K_2(\cos 2x + \cos 2y) + \bar{\Phi} \tag{9.51}$$

Das FDM-Modell dient darüberhinaus als Referenzmodell für Vergleiche. Es handelt sich um eine explizite Zeitdiskretisierung, die mit Hilfe eines Filters stabil gemacht wird. Die Differenzenoperatoren sind mit geeigneten Mittelungsoperatoren so kombiniert, daß das resultierende numerische Verfahren energieerhaltend (bzw. enstrophieerhaltend) ist.

Kapitel 10

Implementierung der adaptiven Semi-Lagrange-Finite-Elemente-Methode

10.1 Übersicht

Die Idee der Semi-Lagrange-Methode und die Einführung von adaptiven Strukturen in dieses Verfahren wurden schon in Teil II vorgestellt. Das neue Programm implementiert die dimensionslosen Gleichungen (9.36)-(9.38) in ebenen Koordinaten, wobei die geopotentielle Höhe in einen konstanten Term und die zeitabhängige Abweichung aufgespalten ist.

Der Programmaufbau unterscheidet sich nicht grundsätzlich von der Struktur, die in Teil II vorgestellt wurde. Bei der Implementierung des Verfahrens für die Flachwassergleichungen ergeben sich jedoch einige bedeutende Unterschiede und Ergänzungen zum linearen Advektionsproblem, die in diesem Kapitel beschrieben werden sollen.

Die Trajektorienberechnung muß hier mit Hilfe eines Extrapolationsschemas erfolgen. In (5.10) ist das verwendete Schema angegeben. Im Gegensatz zur Implementierung der linearen Advektionsgleichung ist dazu ein Interpolationsverfahren notwendig. Die Güte der Interpolation ist hier allerdings nicht kritisch, wie etwa bei der Interpolation der stromaufwärts gelegenen Werte.

In der Berechnung (und Interpolation) der rechten Seite müssen recht aufwendige Terme ausgewertet werden. Die auftretenden Ableitungen werden numerisch berechnet. Dazu bedarf es stabiler und genauer Verfahren. Numerische Differentiation ist ein interessantes Forschungsgebiet, das hier nicht ausführlich behandet werden kann [74]. In diesem Kapitel werden naive Ansätze zur Stabilisierung und zur Verbesserung der Güte der Ergebnisse angegeben. Die Reihenfolge der Berechnungen ist für die Stabilität und Genauigkeit der rechten Seite nach Staniforth und Côté ebenfalls von Bedeutung [127].

Das entstehende elliptische Problem ist durch den Einfluß des Coriolisparameters nicht symmetrisch. Es muß mit einem Lösungsverfahren für unsymmetrische Probleme – in diesem Fall das Verfahren BiCGSTAB – gelöst werden. Als gut parallelisierender und hinreichend beschleunigender Vorkonditionierer wird die Jacobi-Präkonditionierung gewählt.

Die Anpassung des Gitters durch lokale Verfeinerung kann hier nicht einfach durch ein Kriterium erfolgen, das lediglich auf den Gradienten der Funktion aufbaut. Es ist zwar prinzipiell denkbar (und in der Praxis wird dieser Weg auch beschritten), bestimmte physikalische Größen (beispielsweise die Stromfunktion) zu betrachten und dort zu verfeinern, wo diese Funktionen rauh werden. Mathematisch befriedigender ist aber die Verwendung sogenannter *a posteriori* Fehlerschätzer. Im folgenden wird ein solcher Fehlerschätzer für das auftretende elliptische Problem vorgestellt.

Um ein Flachwassermodell für Experimente nutzen zu können, bedarf es einer kompletteren

```
forall i = 1,...,N do
    α_i = 0
    for j = 1,...,itmax do
        x_neu = x − ½α_i
        if(x_neu ∉ G) then:
            Projeziere x_neu auf Γ
        endif
        α_i = ΔtV(x_neu, t + Δt/2)
    enddo
    α(i) = α_i
enddo
```

Abbildung 10.1: Pseudocode für die Trajektorienberechnung

Implementierung. Ozeanographen oder Meteorologen sind daran interessiert, verschiedene abgeleitete Größen aus dem Modell zu gewinnen. Dazu gehören die Vorticity oder die Stromfunktion. Außerdem sollen sogenannte *passive Tracer*, Stoffe, die mit der Flüssigkeit transportiert werden, verfolgt werden. Dies dient beispielsweise der Vorhersage von Schadstoffausbreitung bei Unfällen in Industrieanlagen oder von Staubverteilung nach Vulkanausbrüchen.

10.2 Trajektorienberechnung

Die Trajektorienberechnung im Kontext der Flachwassergleichungen besteht aus einer (inneren) Iteration, in der die Trajektorienstücke α mit Hilfe von (5.9) berechnet werden. Als Extrapolationsformel wird das Zwei-Schritt-Schema aus (5.10) verwendet. Der Pseudocode für die Trajektorienberechnung ist in Abbildung 10.1 angegeben.

Eine wesentliche Schwierigkeit ergibt sich aus der Verwendung adaptiver Methoden: Für jeden Zeitschritt, dessen Werte in die Extrapolation eingehen, muß neben den Werten auch das Gitter gespeichert werden. Das macht das Verfahren in Bezug auf den Speicherbedarf aufwendig. Für diese Operationen werden jedoch nicht alle Felder benötigt, die das Gitter definieren. Lediglich die Arrays, die zur Auffindung eines Elementes zu gegebenen Koordinaten verwendet werden (siehe Abschnitt 6.3), müssen zusätzlich zu den Werten im Speicher gehalten werden.

Die Interpolation der Werte $V(\mathbf{x}_{neu}, \cdot)$ muß für jeden Zeitpunkt getrennt vorgenommen werden, weil sich das Gitter möglicherweise geändert hat. Als Interpolationsroutine wird in der Implementierung die bilineare Interpolation mit Hilfe der FEM-Basisfunktionen verwendet. Diese Wahl ist nach [128] begründet, weil die Interpolationsordnung in der Trajektorienberechnung keine wesentlichen Unterschiede ergibt. Numerisch ist die bilineare Interpolation weniger aufwendig. Die Berechnung des Wertes $V(\mathbf{x}_{neu}, t + \frac{\Delta t}{2})$ ist im folgenden Algorithmus für das Zwei-Schritt-Extrapolationsschema beschrieben:

Algorithmus 10.1 (Extrapolation auf adaptivem Gitter)
Gegeben seien die Koordinaten $\mathbf{x}_{neu} = \mathbf{x} - \frac{1}{2}\alpha$, *sowie die Gitter* T^t *und* $T^{t-\Delta t}$ *zu den Zeiten* t *und* $t - \Delta t$. *Dann durchlaufe folgende Schritte:*

1. *Finde Element* $\tau_t \in T^t$, *so daß* $\mathbf{x}_{neu} \in \tau_t$.
2. *Finde Element* $\tau_{t-\Delta t} \in T^{t-\Delta t}$, *so daß* $\mathbf{x}_{neu} \in \tau_{t-\Delta t}$.

3. *Interpoliere* $V(\mathbf{x}_{neu}, t)$ *an der Stelle* $\mathbf{x}_{neu}$ *in* τ_t, *erhalte* V_t.
4. *Interpoliere* $V(\mathbf{x}_{neu}, t-\Delta t)$ *an der Stelle* $\mathbf{x}_{neu}$ *in* $\tau_{t-\Delta t}$, *erhalte* $V_{t-\Delta t}$.
5. $V(\mathbf{x}_{neu}, t+\frac{\Delta t}{2}) = \frac{3}{2}V_t - \frac{1}{2}V_{t-\Delta t}$.

10.3 Berechnung der rechten Seite des elliptischen Problems

Die rechte Seite besteht aus einer Vielzahl von Termen, die zum Teil numerisch abgeleitet oder interpoliert werden müssen. Die Abfolge der Operationen ist daher wichtig. Aus Gründen der numerischen Stabilität und des Rechenaufwandes werden die Terme weitestgehend zusammengefaßt, bevor interpoliert wird. Als Interpolationsroutinen kommen die Verfahren aus Abschnitt 6.4 zum Tragen. Die rechte Seite R lautet:

$$\begin{aligned} R &= -\frac{1}{\lambda}\left[(\hat{\phi}\hat{A}^+)_{\hat{x}} + (\hat{\phi}\hat{B}^+)_{\hat{y}} + (\hat{\varphi}\hat{B}^+)_{\hat{x}} - (\hat{\varphi}\hat{A}^+)_{\hat{y}}\right] \\ &+ \frac{2}{\Delta\hat{t}\lambda}[(\hat{\phi}\hat{r}_1)_{\hat{x}} + (\hat{\phi}\hat{r}_2)_{\hat{y}} + (\hat{\varphi}\hat{r}_2)_{\hat{x}} - (\hat{\varphi}\hat{r}_1)_{\hat{y}}] \\ &+ \frac{4\hat{\Phi}'^{\circ} + 2\hat{\tilde{\Phi}}}{\hat{\tilde{\Phi}}\lambda\Delta\hat{t}}(\hat{U}^{\circ}_{\hat{x}} + \hat{V}^{\circ}_{\hat{y}}) - \frac{4\hat{\Phi}'^{\circ}}{\hat{\tilde{\Phi}}\lambda(\Delta\hat{t})^2} - \frac{4}{\hat{\tilde{\Phi}}\lambda^2\Delta\hat{t}}\hat{R}_3^{(1/2)}, \end{aligned} \tag{10.1}$$

wobei $\hat{\phi}, \hat{\varphi}, \hat{r}_1, \hat{r}_2$ wie in (9.46) und (9.47) gegeben sind. Die Berechnung erfolgt dann in folgender Weise (im folgenden sind zur Vereinfachung die „$\hat{}$“ gelöscht worden):

Algorithmus 10.2 (Rechte Seite)

1. *Berechne numerisch* $\left.\frac{\partial\Phi'}{\partial x}\right|_{x,y,t}$ *und* $\left.\frac{\partial\Phi'}{\partial y}\right|_{x,y,t}$.
2. *Berechne*

$$\begin{aligned} r_1|_{x,y,t} &= U|_{x,y,t} + \frac{\mu\Delta t}{2}(fV)|_{x,y,t} - \frac{\lambda\Delta t}{2}\left.\frac{\partial\Phi'}{\partial x}\right|_{x,y,t} - \frac{\Delta t}{2}A|_{x,y,t} + \Delta t\, R_1|_{x,y,t+\frac{\Delta t}{2}}, \\ r_2|_{x,y,t} &= V|_{x,y,t} - \frac{\mu\Delta t}{2}(fU)|_{x,y,t} - \frac{\lambda\Delta t}{2}\left.\frac{\partial\Phi'}{\partial y}\right|_{x,y,t} - \frac{\Delta t}{2}B|_{x,y,t} + \Delta t\, R_2|_{x,y,t+\frac{\Delta t}{2}}. \end{aligned}$$

3. *Interpoliere* $r_1|_{x-\alpha_x,y-\alpha_y,t}$ *und* $r_2|_{x-\alpha_x,y-\alpha_y,t}$.
4. *Berechne*

$$\begin{aligned} w_1|_{x-\alpha_x,y-\alpha_y,t} &= (\phi r_1 + \varphi r_2)|_{x-\alpha_x,y-\alpha_y,t} \\ w_2|_{x-\alpha_x,y-\alpha_y,t} &= (\phi r_2 - \varphi r_1)|_{x-\alpha_x,y-\alpha_y,t} \end{aligned}$$

   mit $\phi = \left((\frac{\Delta t}{2})^2 f^2 + 1\right)^{-1}$ *und* $\varphi = \frac{\Delta t}{2}f\phi$.
5. *Berechne numerisch* $\frac{\partial w_1}{\partial x}$ *und* $\frac{\partial w_2}{\partial y}$.
6. *Summiere* $R = R + \frac{2}{\Delta t\lambda}\left(\frac{\partial w_1}{\partial x} + \frac{\partial w_2}{\partial y}\right)$.
7. *Berechne*

$$\begin{aligned} w_3|_{x-\alpha_x,y-\alpha_y,t} &= (\phi A + \varphi B)|_{x-\alpha_x,y-\alpha_y,t}, \\ w_4|_{x-\alpha_x,y-\alpha_y,t} &= (\phi B - \varphi A)|_{x-\alpha_x,y-\alpha_y,t}. \end{aligned}$$

8. *Berechne numerisch* $\frac{\partial w_3}{\partial x}$ *und* $\frac{\partial w_4}{\partial y}$.
9. *Summiere* $R = R - \frac{1}{\lambda}\left(\frac{\partial w_3}{\partial x} + \frac{\partial w_4}{\partial y}\right)$.
10. *Berechne numerisch* $\left.\frac{\partial U}{\partial x}\right|_{x,y,t}$ *und* $\left.\frac{\partial V}{\partial y}\right|_{x,y,t}$.
11. *Berechne*
$$w_5|_{x,y,t} = \frac{4\Phi' + 2\bar{\Phi}}{\bar{\Phi}\lambda\Delta t}\left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y}\right)\Big|_{x,y,t} - \left.\frac{4\Phi'}{\bar{\Phi}\lambda(\Delta t)^2}\right|_{x,y,t}.$$
12. *Interpoliere* $w_5|_{x-\alpha_x, y-\alpha_y, t}$.
13. *Summiere* $R = R + w_5 - \frac{4}{\bar{\Phi}\lambda^2\Delta t} R_3|_{x-\alpha_x, y-\alpha_y, t+\frac{\Delta t}{2}}$.

Damit sind alle Terme auf der rechten Seite berechnet.

Die Stabilität der numerischen Differentiation ist kritisch. Vor allem bei wechselnder Gitterausrichtung werden hier Schwächen bei den Verfahren aus Abschnitt 6.4 deutlich. Eine Verbesserung kann aber dadurch erzielt werden, daß die Vektorprodukte zur Berechnung der Dreiecksfläche stabilisiert werden.

Im Originalverfahren von Akima wird dieses Vektorprodukt für zwei beliebige Kanten gebildet. Je nach Ausrichtung des Elements kann die Wahl dieser Kanten zu unterschiedlich großen Rundungsfehlern führen. Daher wird in der verwendeten modifizierten Version für jedes Element das Mittel der Vektorprodukte gebildet, die durch alle drei verschiedenen Kombinationen von Kantenvektoren entstehen. Dieses Verfahren zeigt eine bessere Stabilität.

10.4 Das elliptische Problem

Das elliptische Problem (9.29) und die rechte Seite aus dem letzten Abschnitt seien gegeben. Zur Lösung des Problems müssen mehrere Schritte durchgeführt werden. Falls sich das Gitter nach Adaption geändert hat, muß die Steifigkeitsmatrix neu aufgestellt werden. Die rechte Seite muß diskretisiert werden und möglicherweise muß die Präkonditionierung initialisiert werden. Damit ergibt sich ein Algorithmus für die Aufstellung und Lösung des elliptischen Problems:

Algorithmus 10.3 (Elliptisches Problem)

1. *Diskretisiere die rechte Seite:* $f = \int_{\mathcal{G}} R\, b\, d\mathcal{G}$.
2. *Berechne die Elementsteifigkeitsmatrizen* $\mathbf{E}^\tau$.
3. *Stelle die Steifigkeitsmatrix* $\mathbf{A}$ *auf.*
4. *Initialisiere die Präkonditionierung (speichere die inverse Diagonale von* $\mathbf{A}$*).*
5. *Initialisiere den Lösungsvektor* $\mathbf{\Phi}$ *mit einer Anfangsnäherung.*
6. *Löse das System* $\mathbf{A\Phi} = \mathbf{f}$ *mit Hilfe des Verfahrens BiCGSTAB.*

Es treten bei der Lösung des elliptischen Problems keine wesentlichen Änderungen im Vergleich mit den im Teil I vorgestellten Methoden auf.

Die Berechnung der Elementsteifigkeitsmatrizen $\mathbf{E}^\tau = (E^\tau_{i,j})$ erfolgt mit Hilfe der Formel:

$$E^\tau_{i,j} = -(\nabla b_i \cdot \nabla b_j)\int_\tau a\, dT + (\nabla b_i \cdot \tilde{\nabla} b_j)\int_\tau b\, dT - \int_\tau c(b_i b_j)\, dT \qquad (10.2)$$

Die Gleichung (10.2) wird diskretisiert, indem die Ableitungen der Basisfunktionen (siehe (1.22)) berechnet werden:

$$\begin{aligned}\frac{\partial b_i}{\partial x} &= \frac{-(y_{j_2} - y_{j_1})}{(x_i - x_{j_1})(y_{j_2} - y_{j_1}) - (x_{j_2} - x_{j_1})(y_i - y_{j_1})}, \\ \frac{\partial b_i}{\partial y} &= \frac{(x_{j_2} - x_{j_1})}{(x_i - x_{j_1})(y_{j_2} - y_{j_1}) - (x_{j_2} - x_{j_1})(y_i - y_{j_1})}.\end{aligned}$$

Die Integrale werden mit Hilfe einer Quadraturformel berechnet (siehe Unterabschnitt 2.3.1).

In der Implementierung wird lediglich eine Jacobi-Präkonditionierung verwendet. Diese Form der Präkonditionierung erweist sich als (in diesem Fall) effizient und einfach zu parallelisieren. Außerdem ist der numerische Aufwand gering. Da Divisionen auf RISC-Prozessoren in der Regel aufwendig sind, wird vor dem Aufruf des iterativen Lösungsverfahrens die Matrixdiagonale invertiert und in einem Hilfsvektor abgespeichert.

Das Verfahren BiCGSTAB wird zur iterativen Lösung des linearen Gleichungssystems verwendet (siehe Abschnitt 2.4). Als Anfangsnäherung des Lösungsvektors bietet sich die alte Lösung aus dem vorherigen Zeitschritt an. Beim BiCGSTAB-Verfahren ist diese Anfangsnäherung jedoch nicht optimal. Eine bessere Anfangsnäherung ist der Vektor mit Elementen konstant Null.

10.5 Fehlerschätzer und Adaptivität

Um ein Kriterium für die lokale Verfeinerung oder Vergröberung des Gitters zu bekommen, soll ein a posteriori Fehlerschätzer eingeführt werden. Verschiedene Arten von Verfeinerungskriterien sind gebräuchlich. Die einfachsten Kriterien verwenden den Gradienten einer physikalisch relevanten Größe als Verfeinerungskriterium. Dieser Weg wird im Teil II beschritten, wo als Verfeinerungskriterium der Gradient der advektierten Funktion verwendet wird. Dort erscheint diese Wahl sinnvoll, weil kein elliptisches Problem zu lösen ist und weil die Interpolation die einzige Operation ist, die von lokaler Verfeinerung profitiert. Die Interpolation ist nämlich an den Stellen fehlerbehaftet, an denen die Funktion rauh ist, d.h. an denen der Gradient groß ist.

Bei der Lösung der Flachwassergleichungen tritt ein elliptisches Problem auf, das in einem Fehlerkriterium berücksichtigt werden kann. Es gibt darüberhinaus mehrere Größen, die als physikalisch relevant angesehen werden und daher ebenfalls im Fehlerkriterium Eingang finden können. Daher ist der Gradient einer Funktion nicht unbedingt eine geeignete Wahl für ein Fehlerkriterium.

In diesem Abschnitt wird ein anderer Weg für die Wahl eines Verfeinerungskriteriums beschritten. Diese Klasse von Fehlerschätzern wurde von Babuška und Rheinboldt [9] eingeführt. Sie basiert auf der Lösung lokaler Probleme der gleichen Art wie das globale Problem. Damit kann der Diskretisierungsfehler bei der Lösung des elliptischen Problems mathematisch mit Hilfe des Residuums abgeschätzt werden.

Das Konzept dieser Fehlerschätzer soll zunächst am abstrakten Beispiel erklärt werden. Sei ein elliptisches Randwertproblem

$$D(u) = R \text{ auf } \mathcal{G} \tag{10.3}$$

gegeben, wobei $u|_\Gamma = g$ geeignet vorgegeben sei. Die zugehörige Bilinearform $a(u, v)$ und die Linearform $f(v) = (R, v)$ seien ebenfalls gegeben. Dann wird der Fehler $e = |u - u_h|$ gesucht, wobei u die (unbekannte) exakte Lösung und u_h die berechnete Näherungslösung sei. In der Praxis wird ein Fehlerkriterium η gesucht, das als Euklidische Norm von lokalen Fehlerkriterien

η_τ definiert ist:

$$\eta = \left[\sum_{\tau \in T} \eta_\tau^2\right]^{\frac{1}{2}}.$$

Dabei werden die η_τ als die lokalen Fehler $\epsilon = e|_\tau$ in der Energienorm berechnet:

$$\eta_\tau = \|\epsilon\|_E^{\frac{1}{2}} = a(\epsilon, \epsilon)^{\frac{1}{2}}.$$

Zwei Ziele werden mit der nun folgenden Vorgehensweise verfolgt. Es soll erstens eine lokale Abschätzung des Fehlers vorgenommen werden und diese Abschätzung soll zweitens numerisch möglichst wenig Aufwand erfordern. Zu diesem Zweck werden nun lokale Hilfsprobleme definiert, die die gleiche Form wie das Problem (10.3) haben, aber nur auf einem Element definiert sind:

$$D(w) = R \quad \text{auf } \tau. \tag{10.4}$$

Dabei müssen die Randwerte so vorgegeben werden, daß tatsächlich das ursprüngliche Problem approximiert wird. Wähle also $w|_{\partial\tau} = u|_{\partial\tau}$. Könnte man w auf τ exakt bestimmen, so gelte:

$$a(w - u_h, v) = a(\epsilon, v) \quad \text{auf } \tau.$$

Damit könnte η_τ bestimmt werden und η wäre der exakte Fehler.

Im allgemeinen wird die exakte Bestimmung von w auf τ genauso wenig möglich sein wie die Lösung des Problems (10.3). Daher wird eine Näherungslösung von (10.4) zur Abschätzung von ϵ berechnet, die mit einer höheren Diskretisierungsordnung bestimmt wird. Dazu kann entweder ein feineres Gitter verwendet werden oder es werden Elemente mit Basisfunktionen einer höheren Polynomordnung gewählt.

In der vorliegenden Implementierung wird ein Fehlerschätzer von Bank und Weiser verwendet [11]. Dabei werden die Randbedingungen und die Diskretisierung des Hilfsproblems so gewählt, daß der Rechenaufwand minimiert wird. Letztlich muß bei diesem Fehlerschätzer auf jedem Element nur noch ein (3×3)-Problem gelöst werden. Die Randbedingungen des lokalen Problems werden durch die Sprünge der Normalenableitungen an den Kanten des Elements festgelegt. Zu lösen ist dann in jedem (inneren) Element das Problem

$$a(\epsilon, v) = (R - D(u_h), v) + \langle \frac{\partial u_h}{\partial n}, v \rangle, \tag{10.5}$$

wobei $\langle f, v \rangle = \int_{\partial\tau} f v \, d\tau$ das Skalarprodukt auf dem Rand des Elementes τ bezeichnet und a die Bilinearform aus (9.32) ist. Dabei werden aber nur die Kantenmittelpunkte jedes (quadratischen) Elements für die Aufstellung eines lokalen linearen Gleichungssystems

$$\mathbf{A}_\tau \epsilon_\tau = \mathbf{r}_\tau$$

verwendet (siehe für die Details [11]). Der Algorithmus des implementierten Fehlerschätzers ist im folgenden angegeben.

Algorithmus 10.4 (Fehlerschätzer) *Für alle Elemente der Triangulierung T durchlaufe die folgenden Schritte:*

1. *Berechne die quadratische Elementsteifigkeitsmatrix mit den Einträgen $a(q_i, q_j)$ für die q_i, q_j die auf den Seitenmittelpunkten definiert sind.*
2. *Berechne das Residuum $(R - D(u_h), q_j)$ für die Seitenmittelpunkte.*

3. *Berechne den Sprung der Normalenableitung der Funktion* u_h, $\langle \frac{\partial u_h}{\partial n}, q_j \rangle$ *über die Kanten des Elementes hinweg.*
4. *Füge die rechte Seite des Systems zusammen:* $r_j = (R - D(u_h), q_j) + \langle \frac{\partial u_h}{\partial n}, q_j \rangle$
5. *Löse das* (3×3)*-Problem* $[\mathrm{a}(q_i, q_j)][\epsilon_j] = [r_j]$.
6. *Berechne den lokalen Fehlerschätzer* $\eta_\tau = \mathrm{a}(\epsilon, \epsilon)^{\frac{1}{2}}$ *und Bestimme das Maximum* $\eta_{max} = \max_\tau \eta_\tau$.

Mit dem so abgeschätzten Fehler kann ein Verfeinerungskriterium definiert werden, wie es schon im Abschnitt 6.2 eingeführt wurde. Heuristische Untersuchungen legen nahe, daß ein annähernd optimales Gitter dann erreicht wird, wenn der Fehler überall gleich groß ist.

Das implementierte adaptive Schema verfeinert das Gitter für diejenigen Elemente τ, für die gilt:

$$\eta_\tau \geq \theta_{fein} \eta_{max}.$$

Dabei ist $0 \leq \theta_{fein} \leq 1$ ein Faktor, der vorgegeben werden kann. Analog wird die Verfeinerung von τ aufgehoben, falls für wenigstens drei der vier Tochterelemente gilt:

$$\eta_\tau \leq \theta_{grob} \eta_{max},$$

wobei $0 \leq \theta_{grob} < \theta_{fein} \leq 1$ ein weiterer vorzugebender Faktor ist. Je kleiner θ_{fein} (und damit auch θ_{grob}) gewählt wird, desto eher wird das Gitter weiter verfeinert.

10.6 Abgeleitete Größen

Das Modell stellt einige abgeleitete Größen und Routinen zur Advektion von *Tracervariablen* zur Verfügung. Die Einbindung passiver Tracer in das Semi-Lagrange-Modell ist einfach. Nachdem die Flachwassergleichungen gelöst sind, sind U, V, Φ als Größen für die Strömung und die Trajektorien $\boldsymbol{\alpha}$ bekannt. Damit kann der Transport des Tracers als ein lineares Advektionsproblem verstanden werden (siehe Teil II). Die Gleichung für die Advektion eines passiven Tracers ϑ lautet:

$$\frac{d\vartheta}{dt} = \frac{\partial \vartheta}{\partial t} + \mathbf{V} \cdot \nabla \vartheta = 0. \tag{10.6}$$

Dabei wird der Wind $\mathbf{V}$ aus dem Flachwassermodell vorgegeben. Die Gleichung ist aus Teil II bekannt. Da die Trajektorienstücke $\boldsymbol{\alpha}$ bekannt sind, muß nur noch die Interpolation an den stromaufwärts gelegenen Punkten erfolgen.

Eine Größe, die häufig von Interesse für Wissenschaftler ist, ist die *Vorticity* ζ. Sie läßt sich leicht mit Hilfe von numerischer Differentiation berechnen:

$$\zeta = \frac{\partial V}{\partial x} - \frac{\partial U}{\partial y} + f.$$

In vielen Fällen wird auch die *Stromfunktion* Ψ betrachtet. Um die Stromfunktion zu berechnen muß allerdings ein Poissonproblem gelöst werden. Ψ ist gegeben durch die Gleichung:

$$\triangle \Psi = \zeta - f$$

Analog zu den diagnostischen Größen im Abschnitt 6.5 lassen sich auch hier die ersten und zweiten Momente der geopotentiellen Höhe Φ oder der Geschwindigkeit berechnen und analysieren.

Kapitel 11

Parallelisierung der adaptiven Semi-Lagrange-Finite-Elemente-Methode

11.1 Übersicht

Die Simulation ozeanischer oder atmosphärischer Zirkulation wird zu den „grand challenges" gerechnet, das sind Anwendungen, die große Herausforderungen bezüglich der Numerik und des Rechenbedarfs darstellen. Parallelisierung solcher Anwendungen ist daher unumgänglich, insbesondere dann, wenn hohe Auflösungen für lokale Phänomene erreicht werden sollen.

Das vorgestellte adaptive Semi-Lagrange-Verfahren für die Flachwassergleichungen ist numerisch recht aufwendig. In jedem Zeitschritt muß ein elliptisches Problem gelöst werden, die Berechnung der rechten Seite ist mit mehreren Interpolationsschritten und Gradientenschätzungen recht teuer und weitere Berechnungen benötigen ebenfalls einige Rechenzeit. Um diese aufwendigen Berechnungen auch in hoher räumlicher Auflösung rechnen zu können ist neben der Adaptivität die Parallelisierung der einzelnen Rechenschritte geboten.

Das Semi-Lagrange-Verfahren bietet dafür gute Voraussetzungen, weil alle Rechenschritte zu einem Gitterpunkt lokal und damit unabhängig von den Nachbarpunkten durchgeführt werden.

Die in den Teilen I und II entwickelten Methoden zur Parallelisierung werden hier ebenfalls angewendet. Es treten dabei gegenüber den ersten beiden Teilen keine neuen Schwierigkeiten auf. Die Datenpartitionierung muß nun für mehrere Zeitschritte erfolgen (und gespeichert werden), weil bei der Aufstellung der rechten Seite Berechnungen sowohl auf dem alten Gitter als auch auf dem aktuellen Gitter durchgeführt werden.

Das Verfahren zur Lösung des elliptischen Problems wird aus dem Teil I direkt übernommen. Neu ist in diesem Teil lediglich der Fehlerschätzer. Da die Fehler aber lokal (elementweise) geschätzt werden, können diese Berechnungen vollständig parallelisiert werden.

Im Gegensatz zur SLM für die Advektionsgleichung aus Teil II ist die Gittergenerierung aufgrund des höheren numerischen Aufwands der Methode für die Flachwassergleichungen nicht mehr dominant. Bis zu moderaten Prozessorzahlen kann daher effizient parallelisiert werden, ohne auch die Gitterverfeinerung zu parallelisieren.

11.2 Parallelisierung des Semi-Lagrange-Teils

Die Parallelisierung der Programmteile, die die Semi-Lagrange-Methode repräsentieren ist analog zu den Methoden im Abschnitt 7.2 möglich. Dabei werden Schleifen, deren Zähler bisher über

alle Knoten zählten, jetzt in eine äußere Schleife über alle Prozessoren und eine innere Schleife über die Knotennummern, auf die der jeweilige Prozessor exclusiv schreibt, geschachtelt. Eine allgemeine Form der ursprünglichen Schleife lautet (Pseudocode):

```
forall i ∈ {i Knoten der Triangulierung} do
    [Führe punktweise Berechnung am Knoten i durch]
enddo
```

Diese allgemeine Schleife wird modifiziert zu:

```
forall p = 1,...,#Prozessoren do parallel
    forall i ∈ Index-Array(p) do
        [Führe punktweise Berechnung am Knoten i durch]
    enddo
enddo parallel
```

Da diese abstrakte Schleife in verschiedenen Prozeduren des Programms auftaucht, ist es aus programmiertechnischen Erwägungen heraus günstig, die Schleifenstruktur in einer `include`-Datei abzulegen und jeweils identisch in den Prozeduren aufzurufen.

Die Berechnungen werden zum Teil auf verschiedenen Gittern durchgeführt (altes Gitter, neues adaptiertes Gitter). Für jedes Gitter werden daher die Knotennummern zu jedem Prozessor in einem eigenen Index-Array gespeichert.

11.3 Parallelisierung des Fehlerschätzers

Die Berechnungen des Fehlerschätzers sind strikt lokal, d.h. für jedes Element wird ein lokaler Fehler geschätzt. Dazu muß für jedes Element ein (3×3)-System aufgestellt und gelöst werden (siehe Abschnitt 10.5). Diese Berechnung erfolgt vollständig unabhängig von den Berechnungen für andere Elemente. Daher kann ohne Einschränkungen parallelisiert werden.

Zu beachten ist hier, daß die Schleife im Fehlerschätzer über alle Elemente der feinsten Triangulierungsstufe läuft. Die Zugriffsstrukur ist hier also nicht punktweise organisiert. Für die Datenpartitionierung ergibt sich hier also ein Problem. Auf der KSR-1 läßt sich diese Schleife wegen der virtuellen Shared-Memory-Architektur aber immer noch effizient (automatisch) parallelisieren:

```
forall τ ∈ {τ Element der feinsten Triangulierung} do parallel
    [Berechne lokalen Fehlerschätzer η_τ]
enddo parallel
```

Die Datenverteilung wird dabei der *Allcache Engine* überlassen. Die Elemente werden gleichmäßig in Blöcken auf die Prozessoren verteilt.

11.4 Parallelisierung des Lösers

Die Parallelisierung des Lösers folgt der Verfahrensweise des Kapitels 3. Die Index-Array-Methode wird verwendet, um die Datenkommunikation zu minimieren. Die Organisation des Speicherzugriffs ist damit punkt- (bzw. knoten-) orientiert. Die Funktionswerte werden in den Semi-Lagrange-Routinen in derselben Reihenfolge zugegriffen wie im elliptischen Löser.

In der inneren Iteration werden Knotendaten daher nur einmal bewegt. Wenn sie verteilt sind, erfolgt allenfalls noch Lesezugriff auf Daten eines anderen Prozessors. Diese Lesezugriffe werden aber (automatisch) durch Rechenoperationen überlappt, weil dabei jeweils eine KSR-Subpage übertragen wird, deren Daten für folgende Operationen lokal zur Verfügung stehen. Sie beeinträchtigen die parallele Leistung des Verfahrens daher nur wenig. Schreibzugriffe erfolgen ausschließlich lokal.

Insgesamt kommen im Programm nur zwei Typen von Schleifen vor:

1. Schleife über die Knoten der Triangulierung (Datenverteilung mit Hilfe von Index-Arrays).
2. Schleife über die Elemente der feinsten Triangulierungsstufe (Datenverteilung automatisch).

Der erste Schleifentyp dominiert nach Anzahl der Schleifen und der Anzahl der arithmetischen Operationen. In jedem Zyklus der inneren Iteration kommen nur zwei Schleifen des zweiten Typs vor. Die Berechnung der Elementsteifigkeitsmatrizen und der lokalen Fehlerschätzer erfolgt jeweils in einer solchen Schleife.

Kapitel 12

Ergebnisse für die adaptive Semi-Lagrange-Finite-Elemente-Methode

12.1 Übersicht

Bei der Präsentation der Ergebnisse der adaptiven Semi-Lagrange-Finite-Elemente-Methode für die Flachwassergleichungen sind verschiedene Aspekte zu berücksichtigen. Zunächst geht es natürlich um den Nachweis, daß die Methode numerisch stabil und genau ist. Da ein wesentlicher Aspekt der Arbeit die Implementierung der Methode ist, kann dieser Nachweis nur anhand einer Referenzlösung erfolgen. Ein analytischer Konvergenz- und Konsistenzbeweis muß daher an dieser Stelle entfallen.

Die numerischen Methoden, die in der Implementierung verwendet werden, sind bereits eingeführt und erprobt. Die Kombination von Semi-Lagrange-Verfahren und adaptiver FEM ist neu, nicht die einzelnen Komponenten. Da nach Möglichkeit auf die Erfahrungen anderer Arbeiten bezüglich Interpolationsordnung, Diskretisierungsordnung, etc. geachtet wurde, erscheint es gerechtfertigt, den formalen Beweis schuldig zu bleiben.

Die im Abschnitt 9.5 angegebenen Anfangsbedingungen für das Flachwassermodell können als „annähernd" analytische Lösung angesehen werden. Die Form und Höhe der durch die Gleichungen (9.50) und (9.51) gegebenen Anfangswerte für Φ und Ψ werden erhalten, es kann also der numerische Fehler beobachtet werden. Die Anfangsbedingungen für die geopotentielle Höhe Φ, die Geschwindigkeitskomponenten U und V, die Geschwindigkeitsvektoren $\mathbf{V} = (U, V)$ und die Vorticity ζ sind in den Abbildungen 12.1 bis 12.3 dargestellt. Auch das Dreiecksgitter ist dargestellt. Die Testrechnungen wurden jedoch auf einer höheren Verfeinerungsstufe durchgeführt, die aus Gründen der Darstellbarkeit nicht abgebildet ist.

Referenzläufe mit einem Finite-Differenzen-Modell, das in [114] beschrieben ist, werden mit den Ergebnissen der ASLM verglichen. Wichtige Vergleichszahlen sind dabei die ersten und zweiten Momente der relevanten physikalischen Größen (Φ, und (U, V)).

Die parallele Leistung des Verfahrens ist von Interesse. Dazu werden die bekannten Leistungsparameter dargestellt. Das Verfahren ist parallel effizient. Durch Parallelisierung kann insbesondere erreicht werden, daß die Ordnung der Rechenzeit der ASLM gleich der Rechenzeitordnung des Referenzprogramms mit FDM-Diskretisierung ist.

Ein Flußdiagramm des implementierten Verfahrens ist in Abbildung 12.4 dargestellt. Die unterlegten Programmteile sind parallelisiert. Die Wahl der speziellen Algorithmen für einzelne Programmsegmente ist jeweils angegeben.

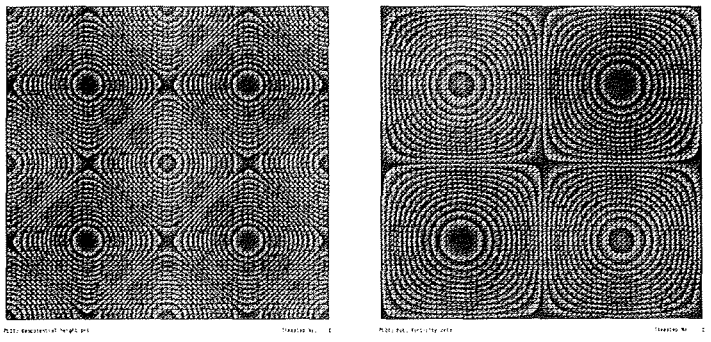

Abbildung 12.1: Modellproblem: Geopotentielle Höhe Φ und Vorticity ζ

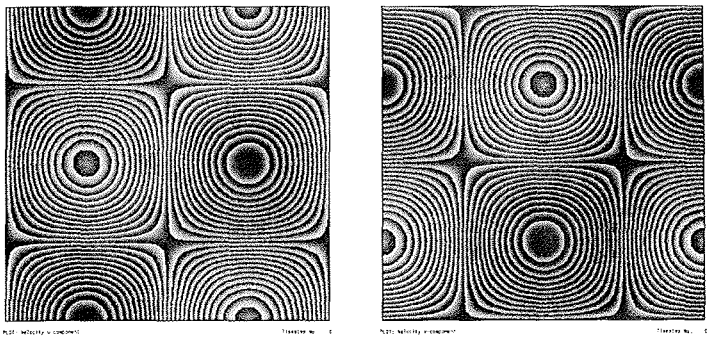

Abbildung 12.2: Modellproblem: Geschwindigkeitskomponenten U und V

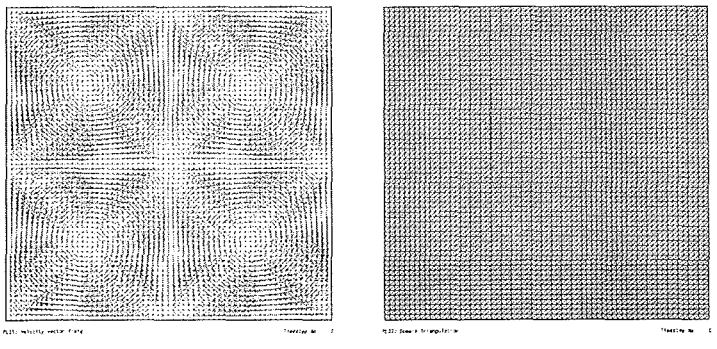

Abbildung 12.3: Modellproblem: Geschwindigkeitsvektoren $\mathbf{V} = (U, V)$, Triangulierung des Gebietes

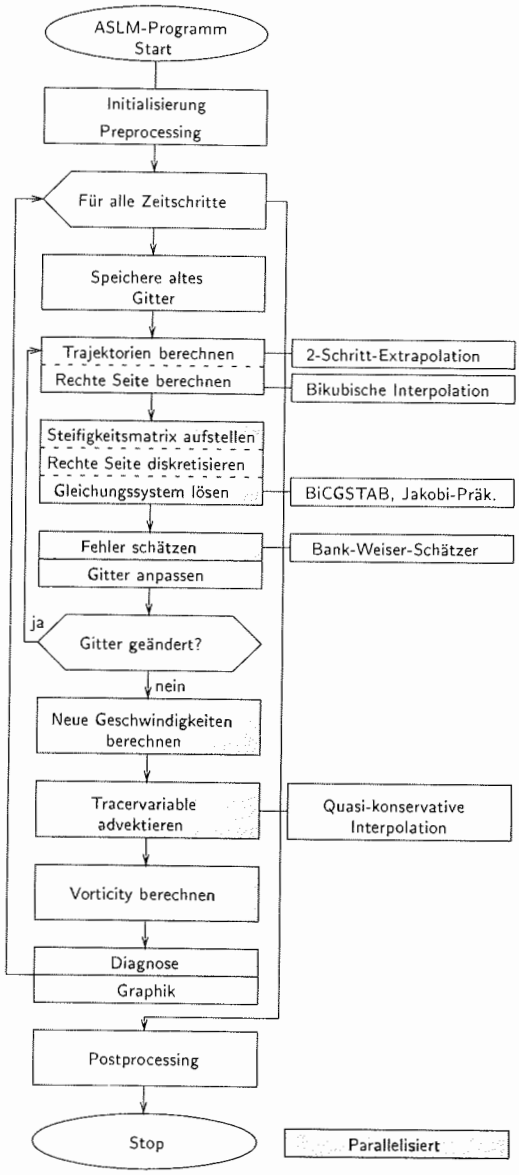

Abbildung 12.4: Flußdiagramm der ASLM, angegeben ist die Wahl der jeweils implementierten Methode, die grau unterlegten Teile sind parallelisiert

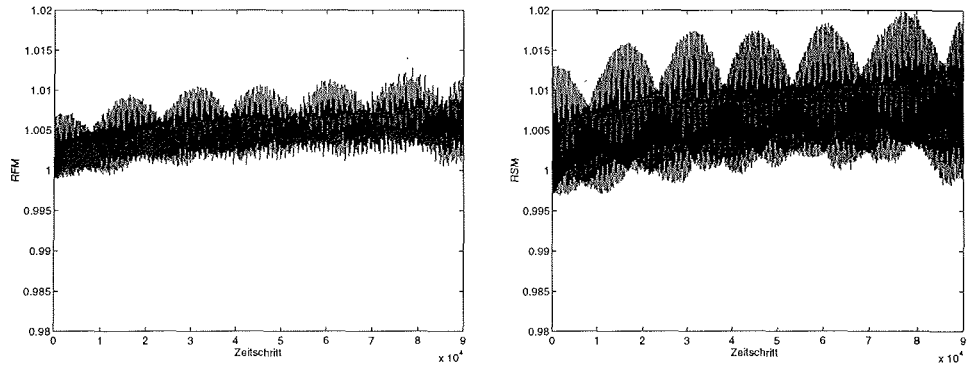

Abbildung 12.5: Erstes und zweites Moment (RFM bzw. RSM) der geopotentiellen Höhe für das Referenzmodell

12.2 Numerische Ergebnisse und Referenzläufe

In diesem Abschnitt werden die numerischen Eigenschaften der ASLM untersucht und mit Referenzergebnissen verglichen. Dazu werden zunächst die Ergebnisse eines Referenzmodells beschrieben. Das Referenzmodell ist in [114] beschrieben und wurde von Paul N. Swarztrauber 1984 am National Center for Atmospheric Research (NCAR) in Boulder, Colorado, implementiert. Die Anfangsbedingungen aus Abschnitt 9.5 werden verwendet. Für die Tests wird das Modell 500 Modellstunden vorwärts integriert. Mit diesen Referenzergebnissen werden dann die Ergebnisse der ASLM verglichen.

12.2.1 Ergebnisse des Referenzmodells

Die Zeitschrittlänge beträgt im Referenzmodell 20 Sekunden und kann wegen des expliziten Zeitschrittes nicht länger gewählt werden. Insgesamt müssen für 500 Modellstunden also 90000 Zeitschritte gerechnet werden. Dabei werden die ersten und zweiten Momente der Geschwindig-

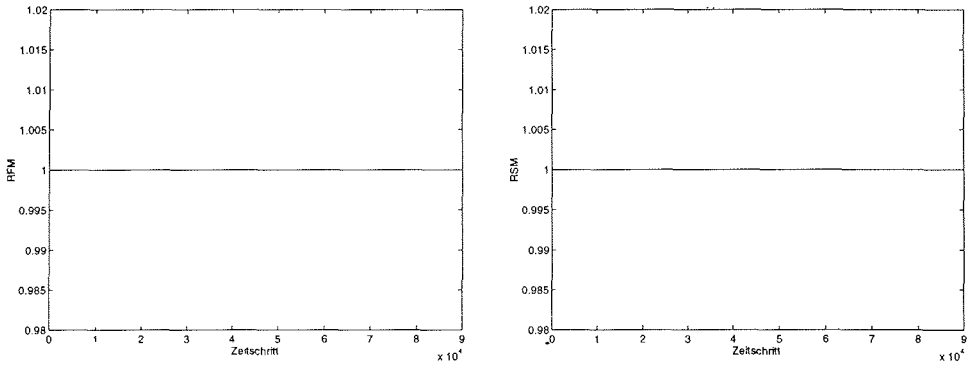

Abbildung 12.6: Wie Abbildung 12.5, jedoch Betrag der Geschwindigkeit

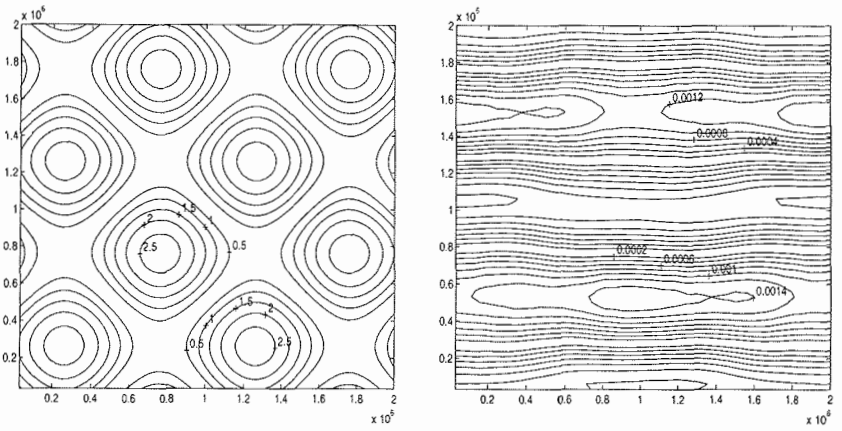

Abbildung 12.7: Konturlinien der Differenz zwischen wahrer und berechneter Lösung nach 500 Modellstunden für die geopotentielle Höhe und die Geschwindigkeitskomponente U

keit $\mathbf{V} = (U, V)$ und der geopotentiellen Höhe Φ aufgezeichnet. Am Ende der Integration wird die Differenz zwischen analytischer und berechneter Lösung für die Geschwindigkeitskomponenten und die geopotentielle Höhe berechnet.

In Abbildung 12.5 sind das erste und zweite Moment der geopotentiellen Höhe dargestellt. Es sind deutliche Oszillationen erkennbar. Insgesamt gewinnt das Verfahren etwas Masse. Dieser Zugewinn liegt bei durchschnittlich etwa einem Prozent, die maximale Schwankung bei zwei Prozent.

Der Zugewinn an Masse wird durch die guten Eigenschaften bezüglich der Geschwindigkeit (Energieerhaltung) kompensiert. Abbildung 12.6 zeigt erstes und zweites Moment des Betrages der Geschwindigkeit. Wegen des energieerhaltenden Verfahrens bleiben diese Werte (bis auf vernachlässigbare Oszillationen) konstant.

Das Modellproblem kann gut zur Analyse des Rechenfehlers des numerischen Verfahrens verwendet werden, weil die Funktionen für Geschwindigkeit, Höhe und Stromfunktion in der Zeit konstant sind. Am Ende des 500-Stundenlaufes wird daher die Differenz zwischen der analytischen und der numerischen Lösung gebildet. Für die geopotentielle Höhe und eine Geschwindigkeitskomponente sind diese Differenzen in den Konturdiagrammen in Abbildung 12.7 dargestellt.

12.2.2 Ergebnisse der ASLM

Analog zu den Ergebnissen des Referenzmodells werden hier die ersten und zweiten Momente der geopotentiellen Höhe und der Geschwindigkeit angegeben. Es werden genau wie im Referenzbeispiel 500 Modellstunden gerechnet. Da die ASLM mit wesentlich längeren Zeitschritten gerechnet werden kann (in diesem Fall 1800 Sekunden), reichen 1000 Zeitschritte für den Testlauf aus.

Die ASLM ist nicht masse-, energie- oder enstrophieerhaltend. Die von Gravel und Staniforth angegebene quasi-masseerhaltende Methode zur Modifikation der ASLM für die Flachwassergleichungen [51] ist nicht direkt auf die Formulierung der Gleichungen in dieser Arbeit übert-

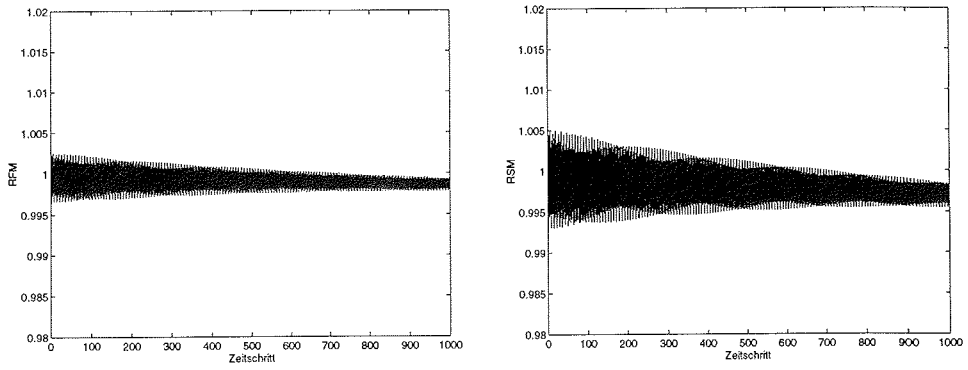

Abbildung 12.8: Wie Abbildung 12.5 für die ASLM

ragbar. Die Entwicklung und Implementierung eines masseerhaltenden Verfahrens konnte in der verfügbaren Zeit nicht durchgeführt werden. Als SLM-Interpolation wird bikubische Spline-Interpolation verwendet. Daher kann kein konstanter Verlauf der RFM- bzw. RSM-Kurven erwartet werden. Abbildungen 12.8 und 12.9 zeigen erste und zweite Momente für die geopotentielle Höhe und den Betrag der Geschwindigkeit.

Für alle Werte, die sich aus der geopotentiellen Höhe ableiten, lassen sich deutliche Oszillationen ausmachen. Diese sind auch in den Darstellungen zum Referenzmodell zu finden. In der ASLM werden die Schwingungen im Verlauf der Zeitintegration gedämpft. Dieses Verhalten läßt sich möglicherweise mit inkonsistenten Anfangsbedingungen für die Geschwindigkeiten und die geopotentielle Höhe erklären. Während Φ als analytische Funktion exakt berechnet wird, werden die Geschwindigkeitskomponenten numerisch aus einer Stromfunktion berechnet. Der numerische Fehler bei der Berechnung von (U, V) führt zu inkonsistenten Anfangsbedingungen. Eine andere Ursache könnte die Verwendung nicht-versetzter Gitter (non-staggered grid) sein.

Die ASLM verliert sowohl Masse als auch Energie. Die Kurven für RFM und RSM fallen (unter Vernachlässigung der Schwingungen) monoton. Dieser Verlust beträgt für die Höhe aber etwa

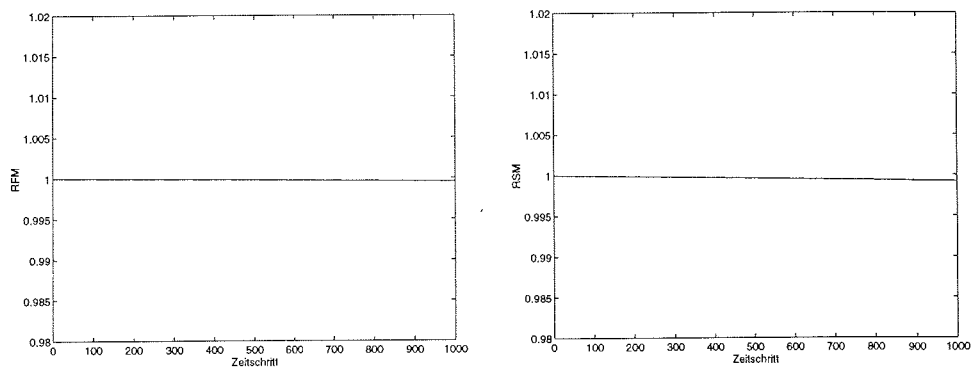

Abbildung 12.9: Wie Abbildung 12.8, jedoch Betrag der Geschwindigkeit

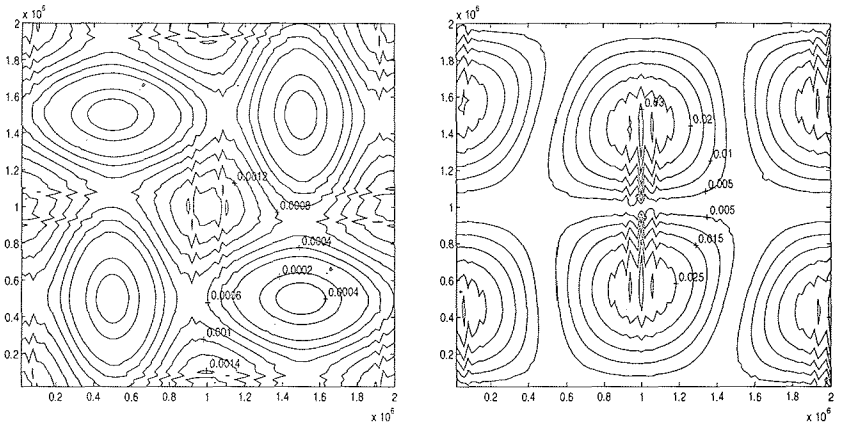

Abbildung 12.10: Wie Abbildung 12.7 für die ASLM

0,1% bei 1000 Zeitschritten, bei der Geschwindigkeit weit unter 0,1%.

In Abbildung 12.10 sind die Differenzen zwischen wahrer Lösung und berechneter Lösung nach 500 Modellstunden dargestellt. Die Werte für die Abweichungen der geopotentiellen Höhe sind mit denen in Abbildung 12.7 nicht direkt vergleichbar. Wegen der verwendeten dimensionslosen Gleichungen, werden die Abweichungen für Φ' angegeben und nicht für $\Phi = \bar{\Phi} + \Phi'$. Außerdem müssen sowohl die Werte für die Abweichungen der Geschwindigkeitskomponente U als auch die Werte für Φ' mit den entsprechenden Skalierungsfaktoren multipliziert werden, um die wahre Abweichung zu erhalten. Die Faktoren betragen für U 1,0 und für Φ' 9,869.

Die berechnete Geschwindigkeit weicht also stärker von der wahren Lösung ab, als im Referenzmodell. Dies ist in den Diagrammen für die ersten und zweiten Momente schon dokumentiert. Die Abweichungen bei der geopotentiellen Höhe fallen dagegen deutlich niedriger aus, als im Referenzmodell.

Werden die Konturlinien der geopotentiellen Höhe von Referenzmodell und ASLM verglichen (Abb. 12.7 und 12.10), dann fällt auf, daß die Abweichungen für die ASLM dort am größten sind, wo die Funktionswerte am größten sind, während die Fehler des Referenzmodells an den Stellen groß sind, an denen der Betrag des Gradienten groß ist. Dieses Verhalten ist typisch für die jeweilige Methode.

Da das Modellproblem sehr glatt ist, konnte der Fehlerschätzer keine sinnvolle Stelle für lokale Verfeinerung ausmachen. Es wird im folgenden Abschnitt zwar auf den Einsatz und den Einfluß der seriellen Gitterroutinen eingegangen, das erzeugte Gitter bleibt aber in allen Fällen gleich.

Als zusätzliche diagnostische Größe wurde für die ASLM die L^2-Norm des Fehlers bei der Berechnung der geopotentiellen Höhe ausgegeben. Diese Größe ist in Abbildung 12.11 dargestellt.

12.3 Parallele Effizienz

Die parallele Effizienz der implementierten ASLM wird wie in den Teilen I und II dargestellt (Abb. 12.12 und 12.13). Die Programmteile, die die SLM repräsentieren sind sehr gut paralleli-

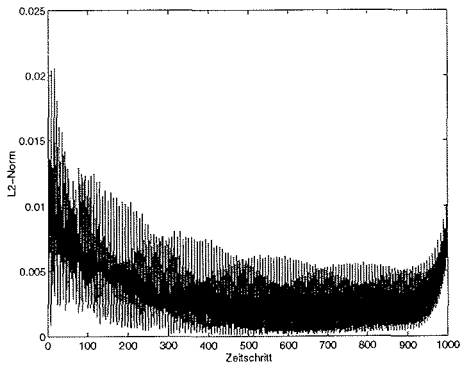

Abbildung 12.11: L^2-Norm des Fehlers bei der Berechnung der geopotentiellen Höhe

sierbar. Sogar die Berechnung der rechten Seite des ellipischen Problems, die aufwendig und aus vielen Schleifen aufgebaut ist, skaliert fast perfekt.

Wie im ersten Teil der Arbeit schon bemerkt, skaliert das Lösungsverfahren BiCGSTAB nicht ganz so gut, wie das CG-Verfahren oder die Semi-Lagrange-Routinen. Dieses Verhalten beeinflußt in der parallelen ASLM für die Flachwassergleichungen die Gesamtleistung.

Die parallele Leistung des Programms wird mit und ohne die adaptiven (seriellen) Programmteile gemessen. Die Adaptivität wirkt zwar für bis zu 26 Prozessoren nicht mehr so dominierend wie im Teil II, aber auch hier ist dieser serielle Programmteil für Leistungseinbußen verantwortlich. Mit diesem erheblichen seriellen Programmanteil kann das Verfahren für große Prozessorzahlen nicht skalieren.

Abbildung 12.12 zeigt die Ergebnisse der Parallelisierungstests für das Programm mit eingeschalteter Adaptivität. Wenn das Programm adaptiv rechnet, so werden die Gitterroutinen mindestens einmal durchlaufen, auch wenn – wie in diesem Fall – keine tatschliche lokale Verfeinerung bzw. Vergröberung vorgenommen wird. Das adaptive Gitter unterscheidet sich hier also nicht vom globalen Gitter. Die Adaptivität wird trotzdem eingeschaltet, damit der Einfluß der seriellen Gitterroutinen auf die Gesamtleistung des Verfahrens gemessen werden kann.

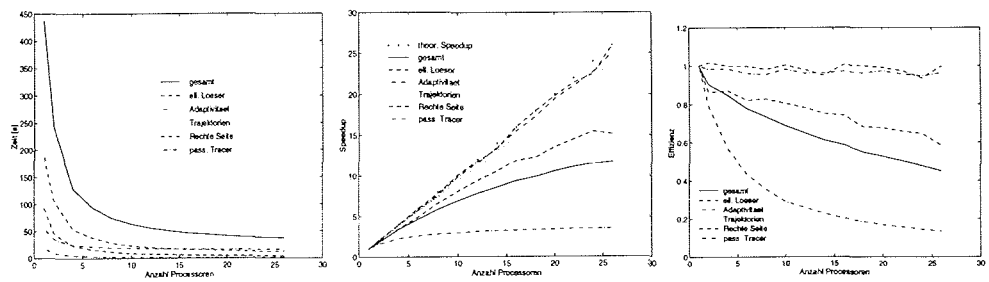

Abbildung 12.12: Zeit für einen Zeitschritt, paralleler Speedup und parallele Effizienz für die ASLM mit adaptiven Routinen

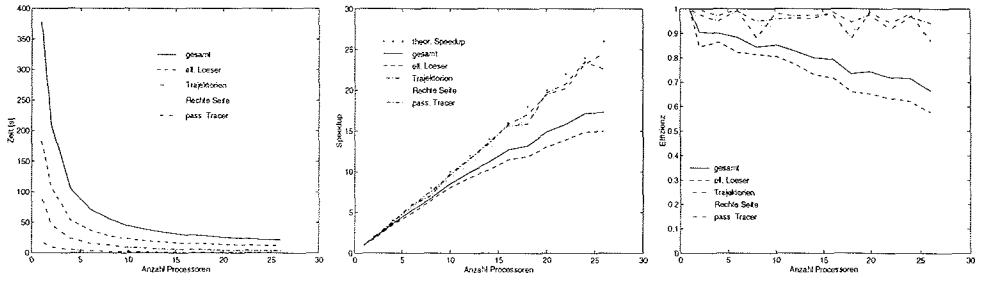

Abbildung 12.13: Wie Abbildung 12.12, jedoch ohne adaptive Programmteile

Es ist erkennbar, daß die seriellen Routinen, die mit der Adaptivität des Verfahrens verbunden sind, nicht skalieren. Trotzdem ist ihr Einfluß nicht so groß, denn das Gesamtprogramm erreicht eine Effizienz von etwa 50%. Die parallelisierten SLM-Routinen (Trajektorien, rechte Seite und Tracervariable berechnen) erreichen sehr gute parallele Effizienz (über 90%).

Das global verfeinerte Verfahren ohne adaptive Routinen (Abb. 12.13) erreicht auch insgesamt hohe parallele Effizienz. In diesem Fall wird der elliptische Löser zum begrenzenden Faktor für die Effizienz des Gesamtverfahrens. Auch in diesem Fall sind die SLM-Routinen mit teilweise über 90% höchst effizient. Der elliptische Löser erreicht 60% Effizienz auf 26 Prozessoren. Beim globalen Verfahren erlaubt das eine Gesamteffizienz von etwa 70%.

Der Scaleup des Verfahrens ist in Abbildung 12.14 dargestellt. Die SLM-Routinen sind gut skalierbar mit Scaleupfaktoren um 80%, das Gesamtverfahren zeigt einen etwas schlechteren Wert (ca. 50%). Für die komplette Implementierung ist dieses Ergebnis jedoch akzeptabel zumal in die Scaleupmessung eingeht, daß das iterative Lösungsverfahren auf den feineren Gittern mehr Iterationen benötigt. Daher ist die Problemgröße eigentlich mehr als mit dem Scaleupfaktor skaliert.

Die Lastverteilung auf 26 Prozessoren ist in Abbildung 12.15 dargestellt. Die Daten sind gleichmäßig auf die Prozessoren verteilt, so daß insbesondere in den SLM-Routinen optimale Parallelisierung erreicht wird. Die Balken im Diagramm stellen die Arbeitslast des jeweiligen

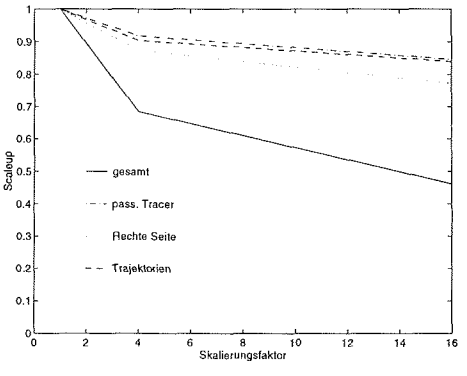

Abbildung 12.14: Scaleup für die parallelisierte ASLM auf der KSR-1

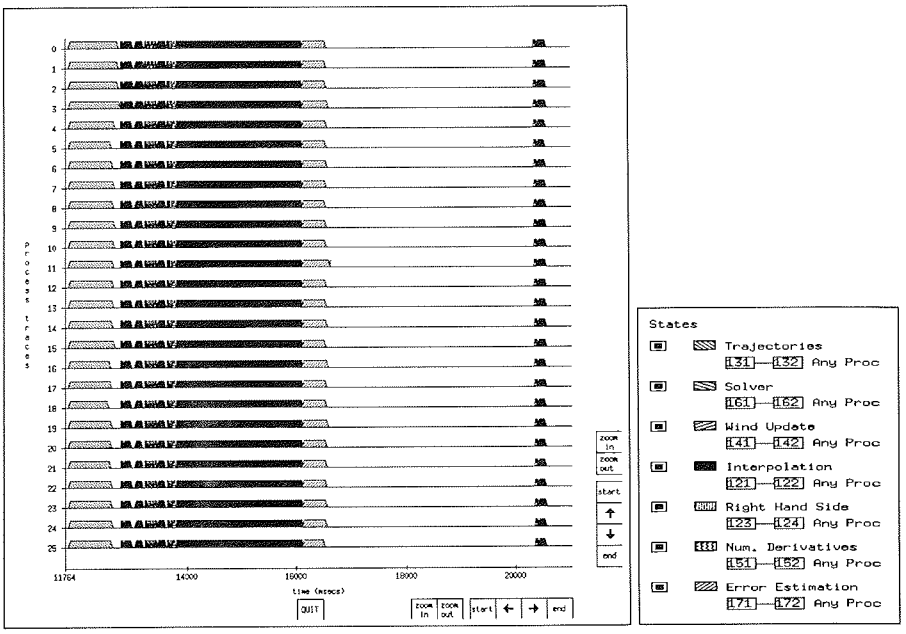

Abbildung 12.15: Lastverteilung und Parallelität für die ASLM, dargestellt ist ein Zeitschritt auf 26 Prozessoren der KSR-1

Prozessors dar. Die Trajektorienberechnung ist viel aufwendiger als bei der ASLM im Teil II (vergleiche Abb. 8.16). Hier muß eine Extrapolation vorgenommen werden, während der Wind in Teil II gegeben war. Die Interpolation tritt mehrmals auf, weil bei der Berechnung der rechten Seite mehrere Interpolationsschritte notwendig sind. Die Aufstellung des Gleichungssystems ist in der Darstellung bei genauer Betrachtung als schmale Spitze auf jedem Prozessor zu sehen. Der schwarze Balken ist zusammengesetzt aus vielen schmalen Spitzen für jede parallele Schleife innerhalb des iterativen Verfahrens. Deutlich ist der parallelisierte Fehlerschätzer erkennbar.

Nicht dargestellt sind die seriellen Programmteile. Sie füllen die Lücke zwischen der parallelen Fehlerschätzung und der ebenfalls parallelen Berechnung der neuen Geschwindigkeitskomponenten sowie der Interpolation für die Tracerberechnung (erkennbar als Spitzen am rechten Rand der Graphik). Die parallelen Programmteile werden aufgrund der Index-Arrays sehr gleichmäßig auf die Prozessoren verteilt und optimal parallelisiert.

12.4 Zusammenfassung

Ein Verfahren für die Lösung der Flachwassergleichungen in karthesischen Koordinaten mit Hilfe einer adaptiven Semi-Lagrange-Methode ist implementiert. Die Implementierung (ca. 15500 Zeilen Fortran Code und ca. 4000 Zeilen C Graphik Code) umfaßt alle wesentlichen Teile, die für die Durchführung von Experimenten notwendig sind. Dazu gehören:

- Einfache Graphikausgabe.
- Parametersteuerung mit Hilfe interaktiver oder dateigesteuerter Dateneingabe.

- Zwischenspeicherung von Systemzuständen zum Aufsetzen neuer Experimente.
- Diagnoseroutinen.
- Semi-Lagrange-Verfahren zur Berechnung passiver Tracer.
- Berechnung der Vorticity.
- Dokumentation.

Das Semi-Lagrange-Verfahren ist semi-implizit und unbedingt stabil. Damit lassen sich relativ lange Zeitschritte wählen, auch wenn die räumliche Auflösung des Problems (lokal) sehr hoch ist. Das auftretende elliptische Randwertproblem wird mit Hilfe einer Finite-Elemente-Methode gelöst.

Eine Anfangstriangulierung des Rechengebietes muß von Hand programmiert werden, globale und lokale Gitterverfeinerung wird dann über Parameter gesteuert automatisch vorgenommen. Das Rechengebiet ist dabei nicht auf rechteckige Geometrien beschränkt, auch wenn für das Modellproblem eine solche Geometrie gewählt wurde. Es steht eine adaptive Gittersteuerung anhand eines mathematischen Fehlerkriteriums zur Verfügung. Damit kann das Gitter automatisch anhand des geschätzten Diskretisierungsfehlers lokal verfeinert oder vergröbert werden.

Die Implementierung ist auf verschiedenen Unix Workstations (LinuX auf Intel Pentium, AIX auf IBM RS/6000, SunOS auf Sun Spark-Workstation) und einer Parallelarchitektur (KSR-1) getestet. Die KSR-1 Implementierung verwendet optimierte Strategien zur Parallelverarbeitung. Die Index-Array-Methode wird dazu verwendet, alle Operationen an Knotenpunkten (bzw. entlang Trajektorien zu einem Knoten) auf den Prozessoren zu fixieren. Damit wird der Aufwand für die Kommunikation minimiert. Das parallelisierte Verfahren, insbesondere die SLM-Routinen, sind effizient und skalierbar.

Die numerischen Vergleiche mit einem auf der Finite-Differenzen-Methode basierenden Referenzmodell zeigen geringe Abweichungen von den Referenzlösungen. Einige charakteristische Unterschiede ergeben sich aus der Verwendung der FEM als Lösungsverfahren für das elliptische Problem: Während das Referenzmodell die größten Fehler an den Stellen aufweist, an denen der Gradient am größten ist, weicht die Lösung der ASLM dort am meisten von der tatsächlichen Lösung ab, wo die Funktion die größte Auslenkung hat.

Die implementierte getestete ASLM ist nicht energie- oder masseerhaltend. Diese Zusatzeigenschaft konnte in der verfügbaren Zeit nicht entwickelt werden, denn die Methode aus [51] ließ sich hier nicht einfach übertragen. Der Energie- bzw. Masseverlust ist mit unter einem Promille bei 1000 Zeitschritten gering. Das Verfahren ist lediglich für flache Koordinaten implementiert. Es eignet sich daher nicht für die Berechnung globaler Zirkulationen.

Während das Referenzmodell auf einem Prozessor der KSR-1 für 500 Modellstunden Integrationszeit (entspricht 90000 Zeitschritten) etwa zwei CPU-Stunden benötigt, werden die 500 Modellstunden der ASLM (1000 Zeitschritte) bei gleicher (globaler) räumlicher Auflösung auf 26 Prozessoren in ca. vier Stunden (elapsed) berechnet. Dabei ist die Adaptivität des Verfahrens noch gar nicht zum Tragen gekommen, weil das Modellproblem adaptive Gitterverfeinerung nicht sinnvoll zuläßt. Die Angleichung der Zeitordnungen beider Methoden ist allein durch die Parallelisierung erreicht worden. Mit Hilfe der parallelisierten ASLM ist es also erstmals möglich, die Vorteile der FEM bezüglich der Randbehandlung, der flexiblen Geometrie und der adaptiven Gitterverfeinerung zu nutzen, ohne die Zeitkomplexität der Experimente zu erhöhen.

Wenn physikalische Probleme mit lokalen Phänomenen (einzelne Wirbel, Fronten, etc.) gerechnet werden, dann kann erwartet werden, daß die Adaptivität des Verfahrens einen zusätzlichen Rechenzeitgewinn erbringt. Ähnlich wie bei der SLM für die passive Advektion aus Teil II kann dann lokal sehr hoch aufgelöst werden, während global grob aufgelöst gerechnet wird.

Mit einer kompletteren Implementierung, die sowohl sphärische Koordinaten, masseerhaltende Interpolation als auch ergonomischere Dateneingabe für die Anfangstriangulierung umfassen sollte, könnte also ein Werkzeug zur Modellierung barotroper Probleme mit lokalen Phänomenen in komplexen Geometrien geschaffen werden.

Zum ersten Mal konnte anhand eines Modellproblems der Nachweis erbracht werden, daß ein adaptives FEM Verfahren auf Dreiecksgittern in Kombination mit der Semi-Lagrange-Methode auf parallelen Rechnerarchitekturen effizient genaue Ergebnisse liefert.

Anhang A

Grundbegriffe der Parallelisierung

In diesem Kapitel werden Grundbegriffe der Parallelverarbeitung zusammengefaßt. Eine sehr gute Darstellung vieler Begrifflichkeiten der parallelen Datenverarbeitung ist bei Lemke zu finden [81]. Einige der folgenden Beschreibungen sind daraus entnommen.

Für die Bewertung der Güte von parallelen Programmen ist das Verständnis von Parallelsystemen notwendig. Es gibt verschiedene Ebenen der Parallelisierung und die unterschiedlichsten Parametrisierungen der Leistung von parallelisierten Algorithmen. Die folgenden Abschnitte geben einen Überblick über die gängigen Methoden und Parametrisierungen und führen die verwendeten Notationen ein, die in den Kapiteln über Parallelisierung verwendet werden.

A.1 Begriffe der Parallelverarbeitung

In diesem Abschnitt sollen häufig verwendete Begriffe eingeführt werden. Die Liste der hier beschriebenen Begriffe ist bei weitem nicht vollständig, die Darstellung konzentriert sich auf die Definitionen, die im Hauptteil der Arbeit Verwendung finden.

Ebenen der Parallelisierung
Parallelverarbeitung kann auf verschiedenen Ebenen ansetzen. Viele der parallel ausgeführten Schritte in einem Programm sind für die Anwender transparent und werden automatisch vom Compiler oder der Hardware durchgeführt. Es werden folgende Ebenen unterschieden:

Bit-Ebene: Parallelität bei der Verarbeitung der Zahlendarstellung.

Instruktions-Ebene: Parallelität bei der Verarbeitung von Instruktionen (z.B. Multiply und Add oder Daten laden und Arithmetik).

Programm-Ebene: Parallelität von Programmteilen (z.B. parallele Schleifen oder parallele Unterroutinen).

Job-Ebene: Parallelität verschiedener Jobs/Programme (Multitasking).

Die Betrachtungen in dieser Arbeit betreffen vor allem die Parallelität auf Programm-Ebene. Die feineren Ebenen werden von optimierenden Compilern recht gut abgedeckt, die Job-Ebene ist von intelligenten Betriebssystemen implementiert.

Granularität
Unter der Granularität paralleler Prozesse ist die Anzahl der Instruktionen in Relation zur Kommunikationszeit und zu Startup-Zeiten zu verstehen. Grobe Granularität liegt vor, wenn

das parallele Programm aus einer relativ geringen Anzahl paralleler Prozesse besteht, die lange unabhängig voneinander parallel arbeiten können, weil jeder Prozeß aus einer großen Anzahl Instruktionen und nur wenigen Kommunikationsanweisungen besteht.

Effizienz
Effizienz ist ein sehr unterschiedlich verwendeter Begriff in der Parallelverarbeitung. Unter der Effizienz eines parallelen Programms kann allgemein das Maß der Nutzung der Rechnerressourcen verstanden werden. Ein Programm ist effizient, wenn die Kommunikation im Verhältnis zur Operationszahl niedrig ist, alle Prozesse gleich viel Arbeit haben und der serielle Anteil klein ist. Bei der Definition der Leistungsparameter wird die Effizienz präzise definiert, allgemein sei ein Programm effizient, wenn es auf p Prozessoren etwa p-mal so schnell ausgeführt wird wie auf einem Prozessor.

Lastverteilung
Die Last der Operationen muß bei parallelen Programmen gleichmäßig auf alle Prozesse verteilt werden, weil die gesamte Ausführungszeit so lang wie die Ausführungszeit des längsten Teilprozesses ist. Lastverteilung ist einfach bei regulären Problemen, weil die Operationen dann häufig statisch gleichmäßig verteilt werden können. Bei irregulären (z.B. adaptiven) Problemen muß die Lastverteilung dynamisch (während der Laufzeit) erfolgen.

Datenverteilung
Häufig ist die Datenverteilung mit der Lastverteilung gekoppelt (siehe datenparalleles Programmiermodell). Die Aufgabe der Datenverteilung besteht allgemein darin, Daten rechzeitig am richtigen Ort zu plazieren und das mit möglichst wenig Kommunikation zu erreichen. Häufig steht der Kommunikationsaufwand bei der Datenverteilung im umgekehrten Verhältnis zum redundant benötigten Speicherplatz.

Skalierbarkeit
Ein paralleler Computer heißt skalierbar, wenn mit dem Ausbau der Prozessorzahl die Kommunikationsbandbreite, der Speicherplatz und die mögliche Leistung im gleichen Verhältnis zunimmt. Ein Algorithmus ist skalierbar, wenn die Ausführungszeit im gleichen Verhältnis sinkt, wie die Prozessorzahl steigt. Skalierbarkeit ist vor allem bei der Entwicklung paralleler Algorithmen ein wichtiger Begriff. Meist stehen den Entwicklern keine voll ausgebauten Parallelcomputer zur Verfügung. Es muß daher sichergestellt werden, daß das entwickelte Verfahren auch in größeren Zusammenhängen effizient ist.

Programmiermodelle
Es gibt eine Fülle von Programmiermodellen für die Parallelisierung. Ein Programmiermodell beschreibt die grundsätzliche Vorgehensweise und das Verständnis der Parallelität in einem Programm. Einige der für die Programmierung auf der KSR-1 relevanten Programmiermodelle sollen hier kurz dargestellt werden.

SPMD Programmiermodell: Das SPMD Programmiermodell (Single Program Multiple Data) geht von einem einzigen Programm in allen Prozessen aus, das in jedem Prozeß eigene Daten bearbeitet. Dieses Programmiermodell findet seine physikalische Entsprechung in der SIMD Computerarchitektur.

MPMD Programmiermodell: Das MPMD Programmiermodell (Multiple Program Multiple Data) ist die Entsprechung zur MIMD Architektur. Jeder Prozeß führt ein eigenes (Teil-) Programm mit eigenen Daten aus.

Datenparalleles Programmiermodell: Das datenparallele Programmiermodell setzt zunächst ein SPMD Modell voraus. Parallelität wird durch Sprachkonstrukte erreicht, die die Daten gleichmäßig auf die Prozesse verteilen und parallel darauf arbeiten.

VSM Programmiermodell: Das VSM Programmiermodell (Virtual Shared Memory) stellt einen gemeinsamen Adreßraum für physikalisch verteilte Daten bereit.

Message-Passing Programmiermodell: Beim Message-Passing Programmiermodell arbeiten mehrere (möglicherweise verschiedene) Prozesse parallel und kommunizieren über Nachrichten (Messages) miteinander. Diese Nachrichten (z.B. Anfragen nach Daten in entfernten Speicherpositionen, Synchronisationen, etc.) müssen explizit programmiert werden.

Kommunikation
Kommunikation ist ein entscheidender Einflußfaktor für die Leistung eines parallelen Programms. Immer dann, wenn zwei Prozesse Daten austauschen, wird kommuniziert. Es kann sich dabei sowohl um Kommunikation für die Synchronisation des Programmablaufs handeln, als auch um Kommunikation zur Beschaffung von Rechendaten.

Für ein gutes Parallelsystem ist es wichtig, daß die Kommunikationsgeschwindigkeit der Rechengeschwindigkeit angepaßt ist. Außerdem sollte die Kommunikationsgeschwindigkeit mit der Anzahl der Prozessoren skalieren.

Synchronisation
Parallele Programmteile müssen immer dann synchronisiert werden, wenn ein Programmteil parallel ausgeführt wird, für den folgenden Programmteil aber die Ergebnisse aller Prozesse vorliegen müssen. Es gibt verschiedene Formen der Synchronisation. *Globale Synchronisation* wird mit Hilfe von Barrieren realisiert. Die Berechnung kann nur dann fortgesetzt werden, wenn alle Prozesse an der Barriere angekommen sind. Die *lokale Synchronisation* findet zwischen einzelnen Prozessen statt. Beispielsweise muß der Schreibzugriff auf ein Datum von verschiedenen Prozessen synchronisiert werden, so daß nicht mehrere Prozesse gleichzeitig schreiben.

Synchronisation bewirkt meistens eine Sequentialisierung des Programms. Daher sollte ein Programm so wenige Synchronisationen wie möglich enthalten.

Overhead
Als Overhead wird der Anteil an Rechenzeit bezeichnet, der zur Synchronisation, zur Kommunikation und zum Starten paralleler Prozesse zusätzlich aufgewendet werden muß, wenn das Programm parallelisiert wird. Häufig wird der Overhead in der Anzahl verlorener Operationen gemessen.

A.2 Leistungsbestimmung bei parallelen Algorithmen

In diesem Abschnitt werden einige der Größen vorgestellt, die für die Leistungsmessung von parallelen Programmen notwendig sind. Viele der hier verwendeten Notationen wurden ursprünglich von Hockney und Jesshope entwickelt [61]. Sie sind hier zum Teil in vereinfachter Form wiedergegeben. Parallelisierung dient in erster Linie der Beschleunigung der Ausführungszeit eines Programms, daraus ergibt sich die folgende Meßgröße.

Definition A.1 (Ausführungszeit) *Die Ausführungszeit $t = t(p, N)$ eines Programms hängt von der Anzahl der beteiligten Prozessoren p und der Problemgröße N ab. t wird als die Zeit gemessen, die ein Programm (oder Programmteil) vom Start bis zum Ende benötigt. Üblicherweise wird t in Sekunden* [s] *gemessen.*

Bemerkung A.2 *Bei parallelen Programmteilen ist die Ausführungszeit t die Zeit, die der längste Prozeß benötigt.*

Bemerkung A.3 *Die Ausführungszeit t ist stark vom verwendeten Computersystem abhängig. Wenn Ausfürungszeiten verschiedener Systeme (Sys1 und Sys2) verwendet werden, dann wird zusätzlich die Indizierung $t^{(Sys1)}$ bzw. $t^{(Sys2)}$ eingeführt.*

Eine weitere wichtige Größe ist die Leistung eines Programmes als Anzahl der Operationen pro Zeiteinheit. Diese Größe, insbesondere aber die asymptotische Maximalleistung eines Systems wird von Herstellern gerne als Werbemittel eingesetzt. Die Leistung ist stark von der Architektur des Computers, vom Compiler und von der Problemstellung abhängig.

Definition A.4 (Leistung, asymptotische Maximalleistung) *Die Leistung r ist definiert als die Anzahl Operationen pro Zeiteinheit. Bei numerischen Anwendungen ist die Maßeinheit* **[Mflop/s]** *(Millionen Fließkommaoperationen pro Sekunde).*

Die asymptotische Maximalleistung r_∞ eines Systems ist die Leistung in **[Mflop/s]**, *die ein Computer theoretisch aufgrund der Taktzeit, der Operationen pro Takt und eventuell der Vektorstartupzeit erreichen kann.*

Für die Messung der Güte der Parallelisierung sind drei Größen – Speedup, parallele Effizienz und Scaleup – sehr gebräuchlich. Der Speedup beschreibt die Beschleunigung des Programms auf p Prozessoren gegenüber einem Prozessor bei konstanter Problemgröße:

Definition A.5 (Speedup) *Der Speedup $s_p = s_p(N)$ eines auf p Prozessoren parallelen Programms mit der Problemgröße N wird definiert durch*

$$s_p := \frac{t(1,N)}{t(p,N)}.$$

Bemerkung A.6 *Falls Parallelisierung nicht die Anzahl der Operationen ändert, gilt für den Speedup: $s_p \leq p$.*

Die parallele Effizienz eines Programms ist der Speedup skaliert mit der Anzahl der Prozessoren. Anders ausgedrückt mißt diese Größe das Verhältnis der Ausführungszeit auf einem Prozessor zur theoretischen parallelen Ausführungszeit auf einem Prozessor.

Definition A.7 (Parallele Effizienz) *Die parallele Effizienz $E_p = E_p(N)$ wird definiert durch*

$$E_p := \frac{t(1,N)}{p \cdot t(p,N)}.$$

Bemerkung A.8 *E_p mißt den Anteil an Overhead, der durch Parallelisierung entsteht. Theoretisch gilt: $E_p \leq 1$.*

Die dritte wichtige Größe bei der Bestimmung der parallelen Leistung eines Programms ist der Scaleup. Diese Größe ist ein sehr viel realistischeres Maß für gute Parallelisierung. Sie beschreibt die Möglichkeit eines Verfahrens mit steigender Prozessoranzahl und steigender Problemgöße zu skalieren. Ein Programm sollte auf einem doppelt so großen System bei doppelter Problemgröße die gleiche Ausführungszeit haben.

Definition A.9 (Scaleup) *Der Scaleup $S = S(l,p,N)$ eines Programms ist definiert durch*

$$S := \frac{t(p,N)}{t(l \cdot p, l \cdot N)}.$$

Ein optimaler Scaleup ($S = 1$) wird erreicht, wenn Kommunikationskosten, Synchronisationskosten und Overhead des parallelen Programms proportional zur Problemgröße wachsen. Dann werden die einzelnen Prozessoren bei Verdoppelung der Problemgröße und gleichzeitiger Verdoppelung der Prozessorzahl dieselbe Arbeitslast haben. Vorraussetzung für optimalen Scaleup ist ein skalierbarer Algorithmus *und* eine skalierbare Rechnerarchitektur.

Zwei weitere Größen sind für die Bestimmung des Overheads von Bedeutung, die Startupzeit und die Latenzzeit. Unter der Startupzeit wird die Zeit verstanden, die ein System zum Starten benötigt. Beispiele sind die Zeit, die eine Vektorpipeline benötigt, bis sie gefüllt ist und das erste Ergebnis produziert wird oder die Zeit, die benötigt wird, um p parallele Prozesse zu aktivieren.

Die Latenzzeit ist die Zeit, die benötigt wird, Daten zwischen den Teilen des Computersystems zu kommunizieren. Je kleiner die Startupzeit und je kürzer die Latenzzeit, desto kleiner können die Programmsegmente sein, die es sich lohnt, zu parallelisieren, d.h. desto feiner kann die Granularität der parallelen Programmteile sein.

Definition A.10 (Startupzeit, Latenzzeit) *Die Startupzeit ist die Zeit zwischen dem ersten Aktivieren eines parallelen Prozesses und dem ersten Ergebnis, das parallel produziert wird.*

Die Latenzzeit ist die Zeit zwischen der ersten Anfrage nach Daten und der tatsächlichen Verfügbarkeit der Daten.

Die meisten Programme sind nicht vollständig parallelisierbar. Daher kann im allgemeinen keine optimale Effizienz erreicht werden. Wenn der Anteil serieller Operationen in einem parallelisierten Programm bekannt ist, kann mit Amdahls Gesetz der theoretisch erreichbare Speedup des Verfahrens ermittelt werden.

Satz A.11 (Amdahls Gesetz) *Sei ν der serielle Anteil der Operationen eines Programms, t_s und t_p seien die Ausführungszeiten für die seriellen bzw. parallelen Operationen. Dann ist die Gesamtzeit gegeben durch*

$$t_{ges}(\nu) = \nu t_s + (1 - \nu) t_p. \tag{A.1}$$

Bemerkung A.12 *Der effektive Speedup s_ν, der bei einem seriellen Programmanteil ν erreicht wird, ist*

$$\begin{aligned} s_\nu &= \frac{t_{ges}(0)}{t_{ges}(\nu)} \\ \Rightarrow s_\nu &= [1 + \nu(R-1)]^{-1}, \end{aligned}$$

wobei $R = \frac{t_s}{t_p}$ das Verhältnis von serieller Zeit zu paralleler Zeit ist.

Anhang B

Merkmale der Architektur von Parallelrechnern

In diesem Kapitel sollen einige Merkmale der Architektur von Parallelrechnern, insbesondere der KSR-1, vorgestellt werden. Dazu werden zunächst einige einführende Bemerkungen zu Computerarchitekturen gemacht. Die Besonderheiten der KSR-1 im Gegensatz zu anderen Architekturen und die Konsequenzen für Programmiermodelle auf diesem System sollen herausgestellt werden. Einige Details, die für die Implementierung von Bedeutung sind, werden ebenfalls erläutert. Ein kurzer Abschnitt beschreibt die Architekturdetails der Alliant FX/2800, auf der wenige Parallelisierungstests im Teil I durchgeführt werden.

B.1 Architekturen von Parallelrechnern

Die Entwicklung paralleler Architekturen reicht bis zu den Anfängen der Computertechnik zurück [61]. Schon der erste elektronische Digitalrechner, der „Electronic Numerical Integrator and Computer" (ENIAC, gebaut zwischen 1943 und 1946), wies viele parallele Merkmale auf. Die Vervielfachung von Recheneinheiten und die parallele Bearbeitung von Rechenoperationen führte zu einer Beschleunigung der Ausführungszeit.

Die ersten Computer, die nach dem heutigen Verständnis Parallelrechner genannt werden können, wurden in den 70er Jahren entwickelt. Zunächst handelte es sich um Prozessorarrays, die aus einfachen vernetzten Prozessoren bestanden und mit einem Instruktionssatz betrieben wurden (ILLIAC IV).

Eine Taxonomie der Rechnerarchitekturen vorzunehmen ist schwierig, weil sehr viele Architekturmerkmale eine Rolle spielen. Gängig ist aber folgende Grobeinteilung nach der Anzahl der Daten- und Instruktionsflüsse (nach Flynn [43]):

SISD: (Single instruction stream, single data stream) Ein Instruktionssatz steuert die Bearbeitung eines Datensatzes.

SIMD: (Single instruction stream, multiple data stream) Ein Instruktionssatz steuert die Bearbeitung vieler Datensätze.

MISD: (Multiple instruction stream, single data stream) Mehrere Instruktionssätze steuern die Bearbeitung eines Datensatzes.

MIMD: (Multiple instruction stream, multiple data stream) Mehrere Instruktionssätze steuern die Bearbeitung vieler Datensätze.

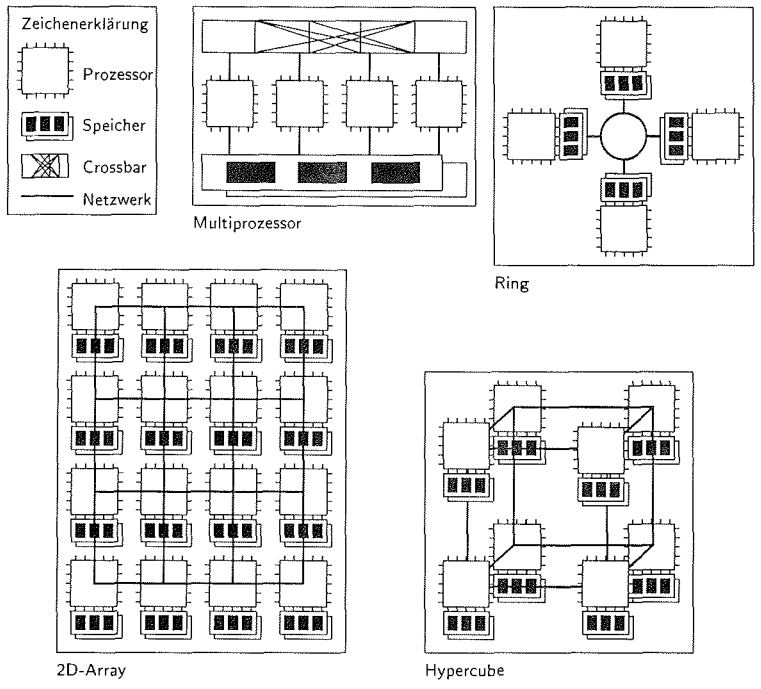

Abbildung B.1: Verschiedene ausgewählte Computerarchitekturen schematisch

SISD-Architekturen sind die herkömmlichen Von-Neumann-Rechner mit einem Prozessor und einer (sequentiellen) Datenleitung. Bei der oben erwähnten ILLIAC IV handelte es sich um eine SIMD-Architektur. Die bekanntesten SIMD-Architekturen stammen von der Firma „Thinking Machines" (CM-2), die erst kürzlich vom Markt verschwunden ist. Auch klassische Vektorrechner (die bekanntesten Systeme stammen von Cray) sind in diese Kategorie einzuordnen. MIMD-Architekturen sind sehr verbreitet. Diese Klasse umfaßt sowohl Mehrprozessor-Server auf PC-Basis bis hin zu Cray C-90 Supercomputern. Echte MISD-Rechner sind dem Autor nicht bekannt, aber diese Technologie wird inzwischen in modernen RISC-Prozessoren eingesetzt, um „branching" Operationen zu optimieren. Die KSR-1 gehört zur MIMD Klasse, die hier weiter spezifiziert werden soll [50].

Ein weiteres wichtiges Architekturmerkmal ist der Aufbau des Hauptspeichers. Grundsätzlich lassen sich Parallelcomputer mit verteiltem und gemeinsamem Hauptspeicher unterscheiden. Die KSR-1 hat zwar einen verteilten Speicher, aber diese Verteilung wird mittels einer Cache-Logik für die Anwender transparent (siehe nächsten Abschnitt).

Die Verbindungstechnik ist ebenfalls ein wesentliches Merkmal der Computerarchitektur. Verschiedene Netzwerktopologien stehen dabei der Verbindung mit Kreuzverteilern (crossbar switches) gegenüber. Die Verbindung der KSR-1-Prozessoren wird durch eine unidirektionale hierarchische Ringtopologie verwirklicht.

Einige wichtige Architekturen sollen beispielhaft vorgestellt werden (siehe Abb. B.1). Die meisten Mehrprozessor-Vektorrechner sind mit gemeinsamem Hauptspeicher ausgestattet, wobei die Prozessoren beliebig miteinander verbunden sind (Cray Vektor-Systeme). Die Kommunikations-

Tabelle B.1: Ordnungen für globale Kommunikation in verschiedenen Computerarchitekturen

Architektur	Kommunikationszeit ($\mathcal{O}(\cdot)$)
gem. Hauptspeicher	1
2D-Array	$2(p-1)$
d-dim. Hypercube	d
Ring	p

zeit (Latenzzeit) bei solchen Systemen ist sehr kurz, es gibt einen gemeinsamen Adreßraum, und jede Adresse ist für jeden Prozessor in derselben Zeit erreichbar. Die Architektur ist jedoch nicht skalierbar, weil schneller Hauptspeicher nicht beliebig ausgebaut werden kann, Kreuzverbindungen nur begrenzte Anzahlen Anschlüsse zulassen, etc. Diese Architekturen werden häufig mit dem Begriff „Multiprozessoren“ benannt.

Zweidimensionale Netze von Prozessoren (2D-Arrays) sind leicht zu implementieren. Diese Netztopologie ist vor allem bei Systemen verbreitet, die auf Transputer Chips basieren. Die Kommunikationszeit nimmt hier mit der Entfernung der Prozessoren zu. Die Architektur ist skalierbar, weil jeder Prozessor mit seinen Verbindungen lokal vervielfältigt werden kann. Für eine globale Kommunikation ist diese Architektur dann allerdings nicht skalierbar, weil die Entfernung zwischen Prozessoren zunimmt. Globale Kommunikationszeiten für die verschiedenen Architekturen sind in Tabelle B.1 angegeben.

Auch im Hypercube nimmt die Kommunikationszeit mit der Entfernung der Prozessoren zu. Die Latenzzeit für globale Kommunikation im d-dimensionalen Hypercube steigt lediglich mit der Ordnung der Dimensionen. Dafür ist eine unbegrenzte Erhöhung der Dimensionen nicht möglich, weil damit auch die Anzahl der Verbindungen zunimmt. Unterhalb bestimmter Dimensionszahlen ist diese Architektur aber skalierbar, weil jeder Prozessor mit eigenem Hauptspeicher ausgestattet werden kann und die Verbindungen lokal sind.

Im Ring ist die Kommunikation von Prozessor zu Prozessor unabhängig von der Entfernung (jedenfalls wenn dem Senden von Daten eine Anfrage vorhergeht). Die globale Kommunikation ist proportional zur Anzahl der Prozessoren im Ring. Ein unendlicher Ausbau des Ringes ist zwar möglich aber wegen der Zunahme der Kommunikationszeit nicht sinnvoll. KSR hat zum massiven Ausbau der Ringtopologie eine Hierarchie von Ringen vorgesehen. Die Kommunikationszeit in entfernte Ringe steigt dann nicht mehr linear mit der Anzahl der Prozessoren, sondern logarithmisch.

Die KSR-1 läßt sich nun einordnen als MIMD-Rechner mit verteiltem Speicher und einer hierarchischen Ringtopologie. Damit ist die Architektur skalierbar (in der Realität ist eine Ausbaustufe bis zu 1088 Prozessoren vorgesehen: ein Ring von 34 Ringen zu je 32 Prozessoren).

B.2 Virtueller gemeinsamer Hauptspeicher

Für die Programmierung und Entwicklung von Algorithmen ist es wünschenswert, so wenig wie möglich mit der Computerhardware in Berührung zu kommen. Für die Datenverwaltung bedeutet das, einen globalen Adreßraum zur Verfügung zu haben (gemeinsamer Hauptspeicher). Leider ist eine Computerarchitektur mit globalem (gemeinsamem) Speicher durch technologische Begrenzungen nicht beliebig skalierbar. Daher haben massiv parallele Rechner lokalen (verteilten) Speicher.

Verteilter Speicher muß in der Regel explizit adressiert werden, das heißt der Programmierer muß

Tabelle B.2: Latenzzeiten für die Cache-Ebenen der KSR-1

Cache Ebene	Clockcycles	Microsekunden
„Subcache“	2	0,1
„Local Cache“	20	1,0
Cache im eigenen Ring	130	6,5
Cache im entfernten Ring	570	28,5

wissen, wo sich seine Daten physikalisch befinden. Diese (häufig nicht portierbare) Komplikation zu umgehen, ist das Ziel virtuellen globalen Speichermanagements (siehe z.B. [6]).

Ein Lösungsansatz für das Problem, auf verteiltem Speicher einen globalen Adreßraum zur Verfügung zu stellen, ist von Kendall Square Research (KSR) unter dem Namen *Allcache* entwickelt worden [70]. Dabei wird der gesamte Speicher als eine Hierarchie von Cachespeichern verstanden. Daten für die Verarbeitung fordert jeder Prozessor aus dem nächstgelegenen Cache an. Falls sie dort nicht zu finden sind, geht die Anfrage in die nächst höhergelegene Ebene der Hierarchie.

Die Anfragelogik und die Cache-Kohärenz wird dabei von Hardwarekomponenten gesteuert. Dadurch wird eine relativ kurze Latenzzeit auch bei Zugriffen auf entfernte Cache-Komponenten erreicht. Ein Datenzugriff besteht dabei immer aus einer Anfrage und einer Antwort. Da die Ringe unidirektional ausgelegt sind, ist die Zeit für einen Datenzugriff determiniert und lediglich von der Anzahl der zu überbrückenden Cache-Hierarchiestufen bestimmt.

Insgesamt gibt es vier Cache-Ebenen mit länger werdenden Latenzzeiten (siehe Tablelle B.2). Die Vorgehensweise ist damit analog zur Programmierung auf Workstations mit Cache-Hierarchien. Es muß darauf geachtet werden, daß Daten möglichst lange im lokalen Cache gehalten werden, oder anders ausgedrückt, es müssen möglichst viele Operationen auf den Daten im lokalen Cache durchgeführt werden (siehe auch [27]).

B.3 Details der KSR-1 Architektur

In diesem Abschnitt werden die allgemein gehaltenen Beschreibungen der vorherigen beiden Abschnitte für die KSR-1 spezifiziert. Werte für Latenzzeit, Prozessorleistung und Hauptspeichergröße werden angegeben. Die Werte sind der einschlägigen KSR-Dokumentation entnommen [70].

Prozessoren
Die Prozessoren der KSR-1 sind 64 Bit RISC-Prozessoren, die von KSR selbst entwickelt wurden. Sie können maximal zwei Instruktionen pro Taktzyklus durchführen. Multiplikation und Addition können parallel ausgeführt werden, d.h. zwei Fließkommaoperationen pro Takt sind erreichbar. Die Taktgeschwindigkeit beträgt 20 Mhz (50 ns). Damit beträgt die theoretische Maximalleistung r_∞ 40 Mflop/s pro Prozessor. Bei Matrixmultiplikationen werden real immerhin 34 Mflop/s erreicht. Bei maximal 1088 Prozessoren erreicht dieses System eine theoretische Maximalleistung von 43520 Mflop/s.

Zu jeder Prozessoreinheit gehören weitere logische Bausteine: Die *Cell Interconnect Unit* verbindet das Prozessorboard mit dem Ring, die *Cache Controll Units* (4 Stück) stellen die Cache-Kohärenz sicher, der *Cache Event Monitor* erlaubt die Diagnose des Datentransfers, es gibt eine *Integer Processing Unit* für ganzzahlige Operationen und einige Boards haben extra *I/O Processors*.

Speicher
Die Prozessoren sind mit relativ großzügigen 256 KB sogenanntem *Subcache* für Daten und ebenfalls 256 KB Instruktionscache ausgestattet. Der Zugriff auf diesen Speicher erfolgt schnellstens alle zwei Taktzyklen. Der Subcache entspricht dem „on-chip cache“ anderer Prozessoren. Zu jedem Prozessormodul gehört ein lokaler Speicher (*local cache*) von 32 MB Größe. Die Zugriffszeit zum lokalen Speicher beträgt 20 Taktzyklen (siehe Tab. B.2). Bei maximal 1088 Prozessoren läßt sich der Hauptspeicher auf 34 GB ausbauen.

Netzwerk
Bis zu 32 Prozessoren können in einem Ring (*Allcache Engine*) zusammengefaßt werden. Die Übertragunsbandbreite jedes Ringes beträgt 450 MB/s. Bei 34 Ringen ergibt das eine Bandbreite von 15300 MB/s. Der Speicher der KSR-1 ist in Einheiten (*Subpage*) der Größe 128 Byte (16 64-Bit-Worte) organisiert. Die kleinste übertragene Datenmenge ist eine Subpage.

Das hat Auswirkungen auf die Programmierung. Wenn auf eine Subpage häufig von mehreren Prozessoren exklusiv zugegriffen wird, kommt es zu erheblichen Leistungseinbrüchen. Diese Situation kann auftreten, wenn verschiedene Daten auf einer Subpage liegen, die jeweils auf verschiedenen Prozessoren benötigt werden.

Eingabe/Ausgabe
In jedem Ring gibt es mindestens einen I/O-Prozessor. Für Ein-/Ausgabe-intensive Anwendungen können mehrere I/O-Prozessoren im Ring installiert werden, so daß die Leistung für Ein-/Ausgabe den Anforderungen angepaßt werden kann. Jeder I/O-Prozessor hat eine Übertragungsbandbreite von 30 MB/s.

Konfiguration des Entwicklungsrechners

Der am Alfred-Wegener-Institut verfügbare Entwicklungsrechner KSR-1 befindet sich in einer relativ kleinen Ausbaustufe. Es existiert ein Ring mit 32 Prozessoren, davon einem I/O-Prozessor. Die Prozessoren sind in sogenannten *Processorsets* so konfiguriert, daß entweder sechs oder 26 Prozessoren parallel für Anwendungen verwendbar sind.

Das System verfügt über 10 GB Plattenplatz auf einem eingebauten *RAID*-Array. Mit 32 MB Cache pro Prozessor sind insgesamt 1024 MB Hauptspeicher vorhanden.

Das System erlaubt als eigenständiger Unix-Rechner interaktiven Zugang vom Workstation-Netz des Instituts. Da die KSR-1 hauptsächlich als Entwicklungsrechner genutzt wird, werden die parallelen Leistungstests meist auf einer leeren Maschine durchgeführt. Es handelt sich aber nicht um ein dediziertes System, in dem viele der Systemdienste zur Leistungssteigerung abgeschaltet sind, sondern um ein System im laufenden Betrieb.

B.4 Details der Alliant FX/2800 Architektur

In den Tests zur Parallelisierung der FEM (Kapitel 4) werden Zahlen von der Alliant FX/2800 genannt. Hier sollen einige Details der Architektur dieses Rechners angegeben werden [5]. Die Alliant FX/2800 war bis 1993 als Entwicklungsrechner am Alfred-Wegener-Institut installiert.

Prozessoren
Alliant verwendet für die *Prozessor-Module* Intel i860 64 Bit RISC-Prozessoren, die mit 40 Mhz (25 ns) getaktet sind. Die Module bestehen aus je vier Prozessoren. Der i860 Prozessor kann gleichzeitig Addition und Multiplikation ausführen, so daß eine theoretische Maximalleistung von 80 Mflop/s erreicht wird. Dieser Wert wird aber aufgrund des kleinen Cache und schlecht

optimierter Compiler in der Realität nicht erreicht. Mit maximal 28 Prozessoren beträgt die theoretische Maximalleistung des Systems 2.240 Mflop/s. Da maximal die Hälfte der Prozessoren in einem sogenannten *Cluster* zur Parallelverarbeitung zusammengefaßt werden können, sollte die Maximalleistung jedoch mit 1.120 Mflop/s angegeben werden.

Die Prozessor-Module verfügen über zwei *concurrency control units* (CCU), die zur parallelen Synchronisation verwendet werden. Jedes Modul ist über zwei *Ports* mit dem Cache verbunden.

Zu jedem System gehört mindestens ein *I/O-Modul.* Es besteht aus einem i860 Prozessor und zwei Kanälen zu Ein-/Ausgabegeräten. Maximal können sieben Prozessor-Module und ein I/O-Modul in einem System konfiguriert werden.

Speicher
Die Alliant FX/2800 ist ein System mit gemeinsamem Hauptspeicher, der modular aufgebaut ist. Ein voll ausgebautes System besteht aus maximal 16 MB Cache in acht Modulen zu je 2 MB, und 4 GB Hauptspeicher in 16 Modulen mit je 256 MB Speicherkapazität. Jeder Prozessor verfügt über 8 KB „on-chip cache" für Daten und 4 KB für Instruktionen. Datenübertragung zwischen Hauptspeicher und Cache benötigt 6 Clockcycles, es werden jeweils vier Worte übertragen.

Jeder Prozessor kann alle vier Clockcycles auf den Cache zugreifen. Jedes Cache-Modul kann zwar alle zwei Clockcycles Daten liefern, aber es gibt doppelt soviele Datenkanäle zu den Prozessoren wie zu den Cache-Modulen.

Netzwerk
Die Prozessor-Module sind mit den Cache-Modulen über einen *Crossbar-Interconnect* verbunden. Dieser Datenbus verbindet jeden Prozessor Port mit jedem Cache-Modul. Je ein 64 Bit Wort kann gleichzeitig an jedem Cache-Modul mit 50 ns Taktzeit und an jedem Prozessorport mit 100 ns Taktzeit übertragen werden. Damit wird eine maximale Bandbreite von 1280 MB/s erreicht.

Neben dem Crossbar-Interconnect zur Datenübertragung gibt es den *Concurrency Control Bus* zur Synchronisation paralleler Prozesse. Dieser Bus verbindet die CCUs und sorgt für eine sehr schnelle Synchronisation.

Konfiguration des Entwicklungsrechners

Die Alliant FX/2800, die bis 1993 am AWI zur Verfügung stand, war mit acht Intel i860 Prozessoren ausgerüstet, die auf den gemeinsamen Hauptspeicher von 256 MB und einen Cache von 2 MB Zugriff hatten. Vier Prozessoren konnten in einem Cluster für die parallele Verarbeitung eines Programms verwendet werden. Typischerweise wurden für Anwendungsprogramme höchstens 5 Mflop/s pro Prozessor erreicht. Die Effizienz der Alliant bei parallelisierten Algorithmen lag im allgemeinen im Bereich von 70-80%.

Bezeichnungen

Einige Notationen sind so gewählt, daß sie im gesamten Text unverändert verwendet werden. Weitere Notationen werden jeweils im Kontext erklärt.

$\mathbf{A}, A_{i,j}, \tilde{\mathbf{A}}$	Diskretisierungsmatrix und Matrixkoeffizienten, präkonditionierte Matrix
a, f	Bilinearform und Linearform im Variationsproblem
$\mathbf{a}$	Windfeld für die Advektion
$\alpha, \boldsymbol{\alpha}$	Trajektorienstück ein- bzw. zweidimensional
b, b_i, q_i	FEM-Basisfunktionen, lineare Basisfunktion und quadratische Basisfunktion zum Knoten i
$C^0(\mathcal{G}), C^k(\mathcal{G}), C_0^\infty(\mathcal{G})$	Raum der stetigen, k-mal stetig differenzierbaren und unendlich oft stetig differenzierbaren Funktionen mit kompaktem Träger
c, C, k, K	Konstanten
D	Differentialoperator
$\Delta x, \Delta t$	Diskrete Gitterweite, Zeitschritt
$\mathbf{E}^\tau, E_{i,j}^\tau$	Elementsteifigkeitsmatrix, Matrixkoeffizienten zum Element τ
$e = u - u_h$	Diskretisierungsfehler
η, η_τ	Globaler und lokaler Fehlerschätzer
$F^\triangle, F_\tau^\triangle$	Dreiecksfläche, Fläche des Elements τ
F	Kraft (unbestimmt)
$\mathbf{f}, \mathrm{f}_i$	Diskrete rechte Seite des elliptischen Problems und Koeffizienten
f	Rechte Seite, im Teil III Coriolisparameter
$\mathcal{G}, \Gamma, \gamma$	Rechengebiet und Rand des Rechengebietes, Randstück
g, q	Funktionen (Randwerte), Polynome; g ist im Teil III auch die Gravitationskonstante
$H^1(\mathcal{G}), H_0^1(\mathcal{G}), H_\gamma^1(\mathcal{G})$	Sobolevraum, mit Rand Null, mit gemischten Randwerten
j	Anzahl der Verfeinerungsstufen
κ	Konditionszahl der Diskretisierungsmatrix
$L^k(\mathcal{G})$	Raum der k-integrierbaren Funktionen
$\triangle, \nabla, \tilde{\nabla}$	Laplace-Operator, Gradienten-Operatoren $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y})$ und $(\frac{\partial}{\partial y}, -\frac{\partial}{\partial x})$
N, N_k, M	Anzahl der Knoten, Knoten der k-ten Stufe, Elemente
Ω	Winkelgeschwindigkeit der Erde
$\Phi, \tilde{\Phi}, \Phi'$	Geopotentielle Höhe, mittlerer Wert, Abweichung
RFM, RSM	Erstes bzw. zweites Moment
R	Rechte Seite im Teil III

$\mathcal{R}, \mathcal{N}$	Reelle Zahlen, natürliche Zahlen
$\rho(\mathbf{x}, t), p(\mathbf{x}, t)$	Dichtefeld, Druckfeld
$\mathcal{S}_k$	diskreter Funktionenraum bezüglich der hierarchischen Basis
s_p, E_p, S	Paralleler Speedup, parallele Effizienz und Scaleup
T, T_k, τ_i	Triangulierung, Triangulierung der Stufe k, Element
$Tr(g)$	Träger der Funktion g
t, t_n	Zeitkoordinate, diskreter Zeitpunkt
ϑ, ζ, Ψ	Tracervariable, Vorticity, Stromfunktion
$\hat{U}$	Dimensionslose Größe
$u^+, u^-, u^\circ, u^{(1/2)}$	u zur Zeit $t + \Delta t$, $t - \Delta t$, t und $t + \frac{\Delta t}{2}$
$u_{high}, u_{low}, u_{kub}$	Interpolation hoher bzw. niedriger Ordnung und mit kubischen Splines
u, v, w, u_h, v_h	Funktionen (unbekannte), zugehörige diskrete Funktionen
$\mathbf{V}, U, V, W$	Geschwindigkeitsvektor, Geschwindigkeitskomponenten
$\mathcal{V}, \mathcal{V}_h, \mathcal{V}_h^{lin}$	Funktionenraum, diskreter (FEM-) Funktionenraum, stückweise lineare FEM-Funktionen
$\mathbf{v} = (v_1, \ldots, v_N)$	Koeffizientenvektor der diskreten FEM-Funktion v_h
$\mathbf{x}, x, y, z, x_i$	Koordinatenvektor, Koordinaten
$\mathbf{x}_m, \boldsymbol{\alpha}_m,$	Koordinatenvektor, Trajektorienstück zum Gitterpunkt m
$(\cdot, \cdot)_\mathcal{V}$	Skalarprodukt auf Raum $\mathcal{V}$
$\|\cdot\|_\mathcal{V}, \|\cdot\|_E$	Norm auf Raum $\mathcal{V}$, Energienorm
$\lvert\cdot\rvert$	Seminorm oder Betrag

Abbildungsverzeichnis

1.1 Lineare FEM-Basisfunktion b_i zum Knoten n_i, der Träger $Tr(b_i)$ ist schraffiert . 22

2.1 Hängende Knoten in einer Triangulierung 26

2.2 Triangulierung eines Gebietes mit Hilfe eines Mustergitters aus Dreiecken 27

2.3 Triangulierung eines Gebietes mit Hilfe von Strahlen zum Rand 28

2.4 Erste Schicht der schichtweisen Triangulierung eines Gebietes 29

2.5 Verfeinerungsstrategien: a. rote Verfeinerung in vier Tochterdreiecke, b. grüne Verfeinerung, c. blaue Verfeinerung 30

2.6 Bisektion der markierten Seite: Nach der Teilung werden die beiden unmarkierten Seiten markiert 31

2.7 Reguäre Verfeinerungsstrategie: a. rote Verfeinerung eines Elements, b. hängende Knoten, c. grüne Verfeinerung 32

2.8 Ein hängender Knoten (○) wird verhindert durch rote Verfeinerung und anschließende grüne Teilung 33

2.9 Nummerierung der lokalen Knoten und Seiten eines Dreiecks 35

2.10 Pseudocode für die Matrixassemblierung in EBE-Ordnung 38

2.11 Pseudocode für die Matrixassemblierung in punktweiser Ordnung 39

2.12 Triangulierung und Numerierung der Knoten 40

2.13 Koordinaten Speicherformat 41

2.14 Kompaktes zeilenweises Speicherformat 41

2.15 Symmetrisches kompaktes zeilenweises Speicherformat 42

2.16 Ellpack-Itpack Format 42

2.17 Pseudocode für das implementierte CG-Verfahren 44

2.18 Pseudocode für das Verfahren BiCGSTAB, die doppelte Schleifenzählung ergibt sich aus der Nutzung von BLAS Routinen 45

2.19 Hierarchische Basen, Basisfunktionen zu den Räumen $\mathcal{V}_0$, $\mathcal{V}_1$ und $\mathcal{V}_2$ 46

2.20 Pseudocode für die Operation der Transformationsmatrizen $\mathbf{S}^\top$ und $\mathbf{S}$ der hierarchischen Basen Präkonditionierung. $n_1(i)$ und $n_2(i)$ sind die Indizes der (groben) Nachbarknoten des Knotens i 48

3.1 Pseudocode für das modifizierte CG-Verfahren 51

3.2 Typische Situation in einem regulär verfeinerten Gitter: die Beiträge aus den Elementen A und B sollen auf den Matrixeintrag zum Knoten i addiert werden . 53

3.3 Dreieckstypen in der Ausgangstriangulierung und nach Verfeinerung 54

3.4 Pseudocode für die parallelisierte Matrixassemblierung in EBE-Ordnung mit Farbindizierung 56

3.5 Pseudocode für die Matrixassemblierung in punktweiser Ordnung 56

4.1 Flußdiagramm des implementierten FEM-Programms; die grau unterlegten Programmteile sind parallelisiert, alternative Methoden sind rechts neben jedem Programmteil angegeben 59

4.2 Modellproblem mit homogenen Dirichlet-Randbedingungen 60

4.3 Modellproblem mit inhomogenen Dirichlet-Randbedingungen 60

4.4 Modellproblem mit periodischen Randbedingungen 60

4.5 L^2-Norm des Fehlers für Modellprobleme mit homogenen und inhomogenen Dirichlet-Randbedingungen mit den Verfahren BiCGSTAG und CG berechnet . . . 61

4.6 Lokal verfeinertes Gitter, wie es in den Tests zur Parallelisierung verwendet wird 62

4.7 Ausführungszeit, Speedup und Effizienz für die Assemblierung der Steifigkeitsmatrix mit globaler und lokaler Gitterverfeinerung (Fälle B und C) und mit Farbindex- bzw. Index-Array-Parallelisierungsstrategie auf der KSR-1 63

4.8 Scaleup der Matrixassemblierung für die Farbindizierung und die Index-Array-Methode auf der KSR-1 63

4.9 Wie Abb. 4.7, jedoch lediglich Farbindizierung auf der Alliant FX/2800 64

4.10 Scaleup der Matrixassemblierung für die Farbindizierung auf der Alliant FX/2800 64

4.11 Ausführungszeit, Speedup und Effizienz für das parallelisierte CG-Verfahren, Parallelisierung mit tiling, Index-Arrays und reduzierten Schleifen 65

4.12 Ausführungszeit, Speedup und Effizienz für das parallelisierte CG-Verfahren mit Jakobi- bzw. hierarchischer Basen Präkonditionierung 65

4.13 Anzahl der Iterationen für das CG-Verfahren mit Jakobi- und hierarchischer Basen Präkonditionierung und für BiCGSTAB mit Jakobi-Präkonditionierung . . . 66

4.14 Ausführungszeit, Speedup und Effizienz für das CG-Verfahren und für das Verfahren BiCGSTAB 66

5.1 Drei-Schritt-SLM für die eindimensionale passive Advektion 73

5.2 Modellproblem „geschlitzter Zylinder“ bei der Initialisierung und nach fünf Umdrehungen mit kubischer Spline Interpolation (Schnitt entlang der x-Achse) . . . 77

6.1 Flächenelement zu einem Gitterpunkt mit irregulärer Verfeinerung 88

7.1 Anteile der einzelnen Routinen an der gesamten Rechenzeit des Zeitschrittes (a: gemessen auf einer IBM RS/6000, b: gemessen auf einem Prozessor der KSR-1) . 92

7.2 Amdahls Gesetz für 60% parallelen Programmteil, 16 Prozessoren und einer seriellen Ausführungszeit von 1 (Links: Speedup, rechts: Effizienz) 92

8.1 Flußdiagramm des SLM-Verfahrens, angegeben sind implementierte alternative Methoden für die Trajektorienberechnung und die SLM-Interpolation, die grau unterlegten Teile sind parallelisiert 94

8.2 Anfangsbedingung für das Problem 8.1 des geschlitzten Zylinders (Schnitt entlang der x-Achse, Oberflächenansicht) 95

8.3 Geschlitzter Zylinder nach einer Umdrehung mit linearer Interpolation (Schnitt entlang der x-Achse, Oberflächenansicht) 95

8.4 *RFM*, *RSM* und L^2-Norm für zwei Umdrehungen und die vier Kombinationen von Interpolation und Gradientenschätzer 96

8.5 *RFM*, *RSM* und L^2-Norm für fünf Umdrehungen mit bikubischer Interpolation und exakt, bzw. iterativ berechneten Trajektorienstücken 96

8.6 Wie Abbildung 8.5, jedoch mit bikubischer Interpolation und clipping 97

8.7 Wie Abbildung 8.5, jedoch mit bikubischer QMSL-Interpolation 97

8.8 Wie Abbildung 8.5, jedoch mit bikubischer, quasi masseerhaltender Interpolation 97

8.9 Wie Abbildung 8.2, jedoch nach fünf Umdrehungen für bikubische Interpolation, Interpolation mit Clipping, QMSL-Interpolation und masseerhaltende Interpolation (von links) 98

8.10 Elemente, die in die Berechnung des Gradienten am zentralen Knoten eingehen. A. bei globaler Verfeinerung, B. bei lokaler Verfeinerung in der Nähe einer Kante der Funktion 99

8.11 *RFM*, *RSM* und L^2-Norm für fünf Umdrehungen mit global und lokal verfeinertem Gitter und quasi-konservativer Interpolation 100

8.12 Das lokal verfeinerte Gitter am Anfang, nach 1/8 und 1/4 Umdrehung 101

8.13 *RFM*, *RSM* und L^2-Norm für zwei Umdrehungen mit quasi-konservativer Interpolation mit exakten Trajektorien, Zeitschrittlänge 900/1800/3600 s 102

8.14 Wie Abbildung 8.13, jedoch mit iterierten Trajektorien 102

8.15 Zeit für einen Zeitschritt, paralleler Speedup und parallele Effizienz für das Advektionsverfahren, globale und lokale Verfeinerung, SLM-Routinen und Gesamtprogramm 103

8.16 Darstellung der Lastverteilung und der Parallelität für einen Zeitschritt des Advektionsverfahrens bei globaler Verfeinerung mit einer inneren Iterationen auf 10 Prozessoren der KSR-1 105

8.17 Wie Abbildung 8.16, jedoch mit lokaler Verfeinerung und vier inneren Iterationen 106

9.1 Annahmen des Flachwassermodells 111

10.1 Pseudocode für die Trajektorienberechnung 120

12.1 Modellproblem: Geopotentielle Höhe Φ und Vorticity ζ 130

12.2 Modellproblem: Geschwindigkeitskomponenten U und V 130

12.3 Modellproblem: Geschwindigkeitsvektoren $\mathbf{V} = (U, V)$, Triangulierung des Gebietes 130

12.4 Flußdiagramm der ASLM, angegeben ist die Wahl der jeweils implementierten Methode, die grau unterlegten Teile sind parallelisiert 131

12.5 Erstes und zweites Moment (RFM bzw. RSM) der geopotentiellen Höhe für das Referenzmodell 132

12.6 Wie Abbildung 12.5, jedoch Betrag der Geschwindigkeit 132

12.7 Konturlinien der Differenz zwischen wahrer und berechneter Lösung nach 500 Modellstunden für die geopotentielle Höhe und die Geschwindigkeitskomponente U 133

12.8 Wie Abbildung 12.5 für die ASLM 134

12.9 Wie Abbildung 12.8, jedoch Betrag der Geschwindigkeit 134

12.10 Wie Abbildung 12.7 für die ASLM 135

12.11 L^2-Norm des Fehlers bei der Berechnung der geopotentiellen Höhe 136

12.12 Zeit für einen Zeitschritt, paralleler Speedup und parallele Effizienz für die ASLM mit adaptiven Routinen 136

12.13 Wie Abbildung 12.12, jedoch ohne adaptive Programmteile 137

12.14 Scaleup für die parallelisierte ASLM auf der KSR-1 137

12.15 Lastverteilung und Parallelität für die ASLM, dargestellt ist ein Zeitschritt auf 26 Prozessoren der KSR-1 138

B.1 Verschiedene ausgewählte Computerarchitekturen schematisch 147

Tabellenverzeichnis

2.1 Elementdaten für die Gittergenerierung 33

2.2 Knotendaten für die Gittergenerierung 34

2.3 Globale Daten für die Gittergenerierung 35

2.4 Zustandsinformationen für Dreiecke 36

2.5 Kodierung der Randbedingungen für jeden Knoten 36

3.1 Farbtabelle für reguläre (rote) Verfeinerung (gerader Level/ ungerader Level) . . 54

3.2 Farbtabelle für grüne Verfeinerung (gerader Level/ ungerader Level) 55

4.1 Verschiedene Gitterkonfigurationen zur Messung der Skalierbarkeit, (g): global verfeinert, (l): zusätzlich lokal verfeinert 62

B.1 Ordnungen für globale Kommunikation in verschiedenen Computerarchitekturen 148

B.2 Latenzzeiten für die Cache-Ebenen der KSR-1 149

Literaturverzeichnis

[1] J. Adams, B. Garcia, B. Gross, J. Hach, D. Haidvogel, and V. Pizzo. Applications of multigrid software in the atmospheric sciences. *Mon. Wea. Rev.*, 120:1447–1458, 1992.

[2] H. Akima. Algorithm 526: Bivariate interpolation and smooth surface fitting for irregularly distributed data points. *ACM Trans. on Math. Softw.*, 4(2):160–164, 1978.

[3] H. Akima. On estimating partial derivatives for bivariate interpolation of scattered data. *Rocky Mountain J. of Math.*, 14(1):41–52, 1984.

[4] J. E. Akin. *Application and Implementation of Finite Element Methods.* Computational Mathematics and Applications. Academic Press, London, 1982.

[5] Alliant Computer Systems Corp., Littleton (Massachusetts). *FX/2800 System Description*, 1991. Part no. 300-00500.

[6] D. P. Anderson, D. Ferrari, P. V. Rangan, and S.-Y. Tzou. The DASH project: Issues in the design of very large distributed systems. Technical Report csd-87-338, Computer Science Department, University of California, Berkeley, 1987.

[7] P. Andrich, P. Delecluse, C. Levy, and G. Madec. A multitasked general circulation model of the ocean. In *Science and engineering on Cray supercomputers, Proc. of the fourth int. Symposium*, Minneapolis, Minnesota, 1988.

[8] D. C. Arney and J. E. Flaherty. An adaptive mesh-moving and local refinement method for time-dependent partial differential equations. *ACM Trans. on Math. Softw.*, 16:48–71, 1990.

[9] I. Babuška and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM J. Numer. Anal.*, 15(4):736–754, 1978.

[10] R. E. Bank. The efficient implementation of local refinement algorithms. In I. et. al. Babuska, editor, *Adaptive computational methods*, pages 74–81, Philadelphia, 1983. SIAM.

[11] R. E. Bank and A. Weiser. Some a posteriori error estimators for elliptic partial differential equations. *Math. Comp.*, 44, No. 170:283–301, 1985.

[12] E. Bänsch. Local mesh refinement in 2 and 3 dimensions. SFB 256 report no. 6, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, 1989.

[13] E. Bänsch. An adaptive finite-element strategy for the three-dimensional time-dependent Navier-Stokes equations. *J. Comput. App. Math.*, 36:3–28, 1991.

[14] E. Barragy, G. F. Carey, and R. Geijn, van de. Performance and scalability of finite element analysis for distributed parallel computation. *Journal of Parallel and Distributed Computing*, 21:202–212, 1994.

[15] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems.* SIAM, Philadelphia, 1993.

[16] S. R. M. Barros, D. P. Dee, and F. Dickstein. A multigrid solver for semi-implicit global shallow-water models. *Atmos. Ocean.*, 28:24–47, 1990.

[17] G. K. Batchelor. *An Introduction to Fluid Dynamics.* Cambridge University Press, Cambridge, 1967.

[18] J. R. Bates. An efficient semi-Lagrangian and alternating-direction implicit method for integrating the shallow-water equations. *Mon. Wea. Rev.*, 112:2033–2047, 1984.

[19] J. R. Bates, F. H. M. Semazzi, R. W. Higgins, and S. R. M. Barros. Integration of the shallow-water equations on the sphere using a vector semi-Lagrangian scheme with a multigrid solver. *Mon. Wea. Rev.*, 118:1615–1627, 1990.

[20] I. Beg, L. Wu, A. Müller, P. Przybyszewski, R. Rühl, and W. Sawyer. PLUMP: Parallel library for unstructured mesh problems. to appear in Proceedings of IRREGULAR 1994, Geneva, 1994.

[21] J. Behrens. Optimierung eines Mehrgitterverfahrens und eines Hierarchische Basen Verfahrens auf einer Alliant FX/80. Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität, Bonn, 1990. Erstellt am Alfred-Wegener-Institut Bremerhaven.

[22] J. Behrens. Parallelization strategies for matrix assembly in finite element methods. In A. Ferreira and J. D. P. Rolim, editors, *Parallel Algorithms for Irregular Problems: State of the Art*, pages 3–24, Dordrecht, 1995. Kluwer Academic Publishers.

[23] J. Behrens, B. Fritzsch, W. Hiller, and H. P. Kersken. Vergleich eines 2-D Mehrgitter Helmholtzlösers mit einem FFT-basierten direkten Löser in QG-Modellen. Report 37, Alfred-Wegener-Institut, Bremerhaven, 1993. AWI Berichte aus dem Fachbereich Physik.

[24] R. Bermejo. On the equivalence of semi-Lagrangian schemes and particle-in-cell finite-element methods. *Mon. Wea. Rev.*, 118:979–987, 1990.

[25] R. Bermejo and A. Staniforth. The conversion of semi-Lagrangian advection schemes to quasi-monotone schemes. *Mon. Wea. Rev.*, 120:2622–2632, 1992.

[26] V. Bjerknes. Das Problem von der Wettervorhersage, betrachtet vom Standpunkt der Mechanik und der Physik. *Meteor. Z.*, 21:1–7, 1904.

[27] E. Boman. Experiences on the KSR1*computer. Report RNR-93-008, NASA Ames Research Centre, NAS, Moffet Field CA, 1993.

[28] K. Bryan. A numerical investigation of a non-linear model of a wind-driven ocean. *J. Atmos. Sci.*, 20:594–606, 1963.

[29] K. Bryan and M. D. Cox. A numerical investigation of the oceanic general circulation. *Tellus*, 19:54–80, 1967.

[30] J. C. Cavendish, D. A. Field, and W. H. Frey. An approach to automatic three-dimensional finite element mesh generation. *Int. J. Numer. Meth. Eng.*, 21:329–347, 1985.

[31] J. G. Charney, R. Fjortøft, and J. von Neumann. Numerical integration of the barotropic vorticity equation. *Tellus*, 2:237–254, 1950.

[32] P. G. Ciarlet. *The finite element method for elliptic problems.* North–Holland Publishing Co., Amsterdam, 1978.

[33] G. Comini, M. Manzan, and C. Nonino. Analysis of finite element schemes for convection-type problems. *Int. Journ. Num. Meth. Fluids*, 20:443–458, 1995.

[34] J. W. Cooley and J. W. Tuckey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[35] J. Côté and A. Staniforth. An accurate and efficient finite-element global model of the shallow-water equations. *Mon. Wea. Rev.*, 118:2707–2717, 1990.

[36] R. Courant, K. O. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Annalen*, 100:67–108, 1928.

[37] M. D. Cox. A primitive equation, three-dimensional model of the ocean. Tech. Report 1, GFDL Ocean Group, Princeton, 1984.

[38] A. W. Craig and O. C. Zienkiewicz. A multigrid algorithm using a hierarchical finite element basis. In D. J. Paddon and H. Holstein, editors, *Multigrid methods for integral and differetial equations*, pages 301–312, Oxford, 1985. Clarendon Press.

[39] W. Dörfler. Hierarchical bases for elliptic problems. SFB 265 Preprint 123, Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, 1990.

[40] D. E. Dougherty. Hydrologic applications of the Connection Machine CM-2. *Water Resour. Research*, 27, No. 12:3137–3147, 1991.

[41] J. K. Dukowics, R. D. Smith, and R. C. Malone. A reformulation and implementation of the Bryan-Cox-Semtner ocean model on the Connection Machine. Preprint LA-UR-92-201, LANL, Los Alamos, New Mexico, 1992. (subm. J. Phys. Ocean.).

[42] G. J. Fix. Finite element model for ocean circulation problems. *SIAM J. Appl. Math.*, 29, No. 3:371–387, 1975.

[43] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. on Comp.*, C-21(9):948–960, 1972.

[44] I. Foster, W. Gropp, and R. Stevens. The parallel scalability of the spectral transform method. *Mon. Wea. Rev.*, 120:835–850, 1992.

[45] K. O. Friedrichs. Symmetric positive linear differential equations. *Comm. Pure and Appl. Math.*, 11:333–418, 1958.

[46] M. J. Fritts, W. P. Crowley, and H. Trease, editors. *The free-Lagrange Method*, volume 238 of *Lecture Notes in Physics.* Springer-Verlag, Berlin, 1985.

[47] P. Garcia-Navarro and A. Priestley. A conservative and shape-preserving semi-lagrangian method for the solution of the shallow water equations. *Int. Journal for Num. Methods in Fluids*, 18:273–294, 1994.

[48] A. E. Gill. *Atmosphere-Ocean Dynamics.* Academic Press, London, 1982.

[49] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and Lanczos algorithms: 1948–1976. *SIAM Review*, 31(1):50–102, 1989.

[50] A. Gottlieb. Architectures for parallel supercomputing. electronic newsgroup com.parallel, 6th Oct 1992.

[51] S. Gravel and A. Staniforth. A mass-conserving semi-Lagrangian scheme for the shallow-water equations. *Mon. Wea. Rev.*, 122:243–248, 1994.

[52] S. Gravel, A. Staniforth, and J. Côté. A stability analysis of a family of baroclinic semi-Lagrangian forecast models. *Mon. Wea. Rev.*, 121:815–824, 1993.

[53] G. Haase, U. Langer, and A. Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. Manuskript, 1992.

[54] J. J. Hack and R. Jakob. Description of a global shallow water model based on the spectral transform method. Technical report, NCAR, Boulder, Colorado, 1992.

[55] W. Hackbusch. *Theorie und Numerik partieller Differentialgleichungen.* Teubner Studienbücher Mathematik. B. G. Teubner, Stuttgart, 1986.

[56] D. B. Haidvogel, J. L. Wilkin, and R. Young. A semi-spectral primitive equation ocean circulation model using vertical sigma and orthogonal curvilinear horizontal coordinates. *Journal of Computational Physics*, 94:151–185, 1991.

[57] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[58] W. Hiller and J. Behrens. Parallelisierung von Mehrgitteralgorithmen auf der Alliant FX/80. In H. W. Meuer, editor, *Parallelisierung komplexer Probleme*, pages 37–82, Berlin, 1991. Springer-Verlag.

[59] J. E. Hirsch. The finite-element method applied to ocean circulation problems. In *Numerical Models of Ocean Circulation*, pages 340–346, Washington, 1975. Nat. Acad. Sciences.

[60] K. Ho-Le. Finite element mesh generation methods: A review and classification. *Computer-Aided Design*, 20(1):27–38, 1988.

[61] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2, Architecture, Programming and Algorithms.* Adam Hilger, Bristol Philadelphia, 2nd edition, 1988.

[62] D. C. Hodgson and P. K. Jimack. A domain decomposition preconditioner for a parallel finite element solver on distributed unstructured grids. Report 95.1, University of Leeds, Divivsion of Computer Science, 1995.

[63] W. R. Holland. Quasi-geostrophic modelling of eddy-resolved ocean circulation. In J. J. O'Brien, editor, *Advanced Physical Oceanographical Numerical Modelling*, pages 203–231, 1986.

[64] W. R. Holland, T. Keffer, and P. B. Rhines. Dynamics of the oceanic general circulation: The potential vorticity field. *Nature*, 308:698–705, 1984.

[65] M. Holt. *Numerical Methods in Fluid Dynamics.* Springer Series in Computational Physics. Springer-Verlag, Berlin, 2 edition, 1984.

[66] R. Jakob-Chien, J. J. Hack, and D. L. Williamson. Spectral transform solutions to the shallow water test set. *Jour. Comp. Phys.*, 119:164–187, 1995.

[67] C. Johnson. *Numerical solution of partial differential equations by the finite element method.* Cambridge University Press, Cambridge, 1990.

[68] M. T. Jones and P. E Plassmann. A parallel graph coloring heuristic. *SIAM J. Sci. Comput.*, 14(3):654–669, 1993.

[69] M. Juckes. A shallow water model of the winter stratosphere. *Journal of the Atmos. Sciences*, 46(19):2934–2955, 1989.

[70] Kendall Square Research Corp., Waltham, MA. *KSR/Series Principles of Operation*, 1994. Revision 7.0.

[71] H.-P. Kersken, B. Fritzsch, O. Schenk, W. Hiller, J. Behrens, and E. Krauße. Parallelization of large scale ocean models by data decomposition. In W. Gentzsch and U. Harms, editors, *High-Performance Computing and Networking*, number 796 in Lecture Notes in Computer Science, pages 323–328, Berlin, 1994. Springer-Verlag.

[72] P. D. Killworth, D. Stainforth, D. J. Webb, and S. M. Paterson. The development of a free-surface bryan-cox-semtner ocean model. *Jou. Phys. Ocean.*, 21:1333–1348, 1991.

[73] I. M. Klucewicz. A piecewise C^1 interpolant to arbitrarily spaced data. *Comp. Graph. and Image Proc.*, 8:92–112, 1978.

[74] I. Knowles and R. Wallace. A variational method for numerical differentiation. *Numer. Math.*, 70:91–110, 1995.

[75] F. Kruse, A. Hense, D. Olbers, and J. Schröter. A quasigeostrophic eddy resolving model of the Antarctic Circumpolar Current: a description of the model and the first experiment. Technical Report 6, Alfred-Wegener-Institut, Bremerhaven, 1990.

[76] G. Lang. Kurzbeschreibung des Programmes PFRG02. Technical report, Bundesanstalt für Wasserbau, Hamburg, 1992.

[77] C. L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Mathematical Software (3.)*, pages 161–194. Academic Press, 1977.

[78] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. on Math. Softw.*, 5(3):308–323, 1979.

[79] C. le Provost. On the use of finite element methods for ocean modelling. In J. J. O'Brien, editor, *Advanced Physical Oceanographical Numerical Modelling*, pages 557–580, 1986.

[80] D. Y. Le Roux, C. A. Lin, and A. Staniforth. An accurate interpolation sceme for semi-Lagrangian advection on an unstructured mesh for ocean modelling. to be published, 1995.

[81] M. Lemke. *Multilevelverfahren mit selbstadaptiven Gitterverfeinerungen für Parallelrechner mit verteiltem Speicher.* Number 227 in GMD-Bericht. R. Oldenbourg Verlag, München/Wien, 1994.

[82] J. G. Lewis and R. A. van de Geijn. Distributed memory matrix-vector multiplication and conjugate gradient algorithms. ftp'ed from the Internet, 1994.

[83] Z. Li and M. B. Reed. Convergence analysis for an element-by-element finite element method. *Comput. Methods Appl. Mech. Engrg.*, 123:33–42, 1995.

[84] D. A. Lindholm. Automatic triangular mesh generation on surfaces of polyhedra. *IEEE Trans. Mag.*, 19(6):1539–1542, 1984.

[85] J. Liou and T. E. Tezduyar. Clustered element-by-element computations for fluid flow. In H. D. Simon, editor, *Parallel Computational Fluid Dynamics: Implementations and Results*, Scientific and Engineering Computation, pages 167–187, Cambridge Massachusetts, 1992. The MIT Press.

[86] Rivara. M-C. A grid generator based on 4-triangles conforming mesh-refinement algorithms. *Int. J. for Num. Meth. in Eng.*, 24:1343–1354, 1987.

[87] D. Marsal. *Finite Differenzen und Elemente - Numerische Lösung von Variationsproblemen und partiellen Differentialgleichungen.* Springer-Verlag, Berlin, 1989.

[88] J. D. McCalpin. A quantitative analysis of the dissipation inherent in semi-Lagrangian advection. *Mon. Wea. Rev.*, 116:2330–2336, 1988.

[89] A. McDonald. Accuracy of multiply-upstream, semi-Lagrangian advective schemes II. *Mon. Wea. Rev.*, 115:1446–1450, 1987.

[90] A. McDonald and J. R. Bates. Improving the estimate of the departure point position in a two-time level semi-Lagrangian and semi-implicit scheme. *Mon. Wea. Rev.*, 115:737–739, 1987.

[91] U. Meier, G. Skinner, and J. Gunnels. A collection of codes for sparse matrix computations. CSRD Report 1134, University of Illinois at Urbana-Champaign, Center for Supercomputing Research and Development, 1991.

[92] W. F. Mitchell. A comparison of adaptive refinement techniques for elliptic problems. *ACM Trans. in Math. Softw.*, 15(4):326–347, 1989.

[93] A. S. Monin. *Theoretical Geophysical Fluid Dynamics*, volume 6 of *Environmental Fluid Mechanics.* Kluwer Academic Publishers, Dordrecht, Boston, London, 1990.

[94] L. B. Montefusco and G. Casciola. Algorithm 677: C^1 surface interpolation. *ACM Trans. on Math. Softw.*, 15(4):365–374, 1989.

[95] D. Moore, J. Warren, and J. E. Akin. Parallel local adaptive mesh generation. In *Proceedings of the MAFELAP Conference 1990*, pages 93–100. Academic Press, 1991.

[96] R. Natarajan. Finite element applications on a shared-memory multiprocessor: Algorithms and experimental results. *J. Comp. Phys.*, 94:325–381, 1991.

[97] S. Ničković. On the use of hexagonal grids for simulation of atmospheric processes. *Beitr. Phys. Atmosph.*, 67(2):103–107, 1994.

[98] G. Nielson. Minimum norm interpolation in triangles. *SIAM J. Numer. Anal.*, 17(1):44–62, 1980.

[99] A. Oliveira and A. M. Babtista. A comparison of integration and interpolation Eulerian-Lagrangian methods. *Int. Journ. Num. Meth. Fluids*, 21:183–204, 1995.

[100] S. A. Orszag. Transform method for calculation of vector coupled sums: Application to the spectral form of the vorticity equation. *J. Atmos. Sci.*, 27:890–895, 1970.

[101] R. Pacanowski, K. Dixon, and T. Rosati. The G.F.D.L. modular ocean model users guide. Tech. Report 2, G.F.D.L. Ocean Group, Princeton, 1991.

[102] J. Pedlosky. *Geophysical Fluid Dynamics.* Springer-Verlag, New York, 2nd edition, 1987.

[103] N. A. Phillips. The general circulation of the atmosphere: A numerical experiment. *Quart. J. Roy. Meteor. Soc.*, 82:123–164, 1956.

[104] C. Pommerell. *Solution of Large Unsymmetric Systems of Linear Equations*, volume 17 of *Series in Microelectronics*. Hartung-Gorre Verlag, Konstanz, 1992.

[105] S. Pond and G. L. Pickard. *Introductory Dynamical Oceanography*. Pergamon Press, Oxford, 2nd edition, 1983.

[106] A. Priestley. The Taylor-Galerkin method for the shallow-water equations on the sphere. *Mon. Wea. Rev.*, 120:3003–3015, 1992.

[107] A. Priestley. A quasi-conservative version of the semi-Lagrangian advection scheme. *Mon. Wea. Rev.*, 121:621–629, 1993.

[108] M. Rančić and G. Sindjić. Noninterpolating semi-Lagrangian advection scheme with minimized dissipation and dispersion errors. *Mon. Wea. Rev.*, 117:1906–1911, 1989.

[109] P. J. Rasch and D. L. Williamson. On shape-preserving interpolation and semi-Lagrangian transport. *SIAM J. Sci. Stat. Comput.*, 11(4):656–687, 1990.

[110] R. J. Renka. Algorithm 624: Triangulation and interpolation at arbitrarily distributed points in the plane. *ACM Trans. on Math. Softw.*, 10(4):440–442, 1984.

[111] L. F. Richardson. *Weather Prediction by Numerical Process*. Cambridge University Press, 1922. reprinted Dover 1965.

[112] H. Ritchie. Application of the semi-Lagrangian method to a spectral model of the shallow-water equations. *Mon. Wea. Rev.*, 116:1587–1598, 1988.

[113] P. Roache. *Computational Fluid Dynamics*. Hermosa Publishers, Albuquerque NM, 1972.

[114] Robert Sadourny. The dynamics of finite-difference models of the shallow-water equations. *J. Atmos. Sci.*, 32:680–689, 1975.

[115] A. S. Sarkisyan. Osnovy teorii i raschet okeanicheskyky techeny (Fundamentals of the theory and calculation of ocean currents). *Gidrometeoizdat*, 1966.

[116] O. Schenk, H. P. Kersken, B. Fritzsch, and W. Hiller. Parallelisierung des quasigeostrophischen Zirkulationsmodells. Report 47, Alfred-Wegener-Institut, Bremerhaven, 1994. AWI Berichte aus dem Fachbereich Physik.

[117] H. R. Schwarz. *Methode der finiten Elemente*. Teubner Studienbücher Mathematik. B. G. Teubner, Stuttgart, 1991.

[118] F. H. M. Semazzi and P. Dekker. Optimal accuracy in semi-Lagrangian models. *Mon. Wea. Rev.*, 122:2139–2159, 1994.

[119] A. J. Semtner, Jr. Finite-difference formulation of a world ocean model. In J. J. O'Brien, editor, *Advanced Physical Oceanographical Numerical Modelling*, pages 187–202, 1986.

[120] A. J. Semtner Jr. An oceanic general circulation model with bottom topography. Numerical simulation of weather and climate. Tech. Report 9, University of California, Los Angeles, 1974.

[121] J. R. Shewchuk and O. Ghattas. A compiler for parallel finite element methods with domain-decomposed unstructured meshes. Preprint, Carnegie Mellon University, Pittsburgh, 1994. by anon ftp.

[122] A. Shostko and R. Löhner. Three-dimensional parallel unstructured grid generation. *International Journal for Numerical Methods in Engineering*, 38:905–925, 1995.

[123] H. D. Simon, editor. *Parallel computational fluid dynamics – implementations and results.* Scientific and Engineering Computation. The MIT Press, Cambridge Massachusetts, 1992.

[124] J. Smagorinsky, S. Manabe, and J. L. Holloway. Numerical results from a nine-level general circulation model for the atmosphere. *Mon. Wea. Rev.*, 93:727–768, 1965.

[125] R. D. Smith, J. K. Dukowicz, and R. C. Malone. Parallel ocean general circulation modeling. Preprint LA-UR-92-200, LANL, Los Alamos, New Mexico, 1992.

[126] P. K. Smolarkiewicz and P. J. Rasch. Monotone advection on the sphere: An Eulerian versus semi-Lagrangian approach. *J. Atmosph. Sci.*, 48(6):793–810, 1991.

[127] A. Staniforth and J. Côté. Semi-Lagrangian integration schemes for atmospheric models - a review. *Mon. Wea. Rev.*, 119:2206–2223, 1991.

[128] A. Staniforth and C. Temperton. Semi-implicit semi-Lagrangian integration schemes for a barotropic finite-element regional model. *Mon. Wea. Rev.*, 114:2078–2090, 1986.

[129] P. K. Stansby and P. M. Lloyd. A semi-implicit Lagrangian scheme for 3D shallow water flow with a two-layer turbulence model. *Int. Journ. Num. Meth. Fluids*, 20:115–133, 1995.

[130] J. Stoer. *Numerische Mathematik 1.* Springer-Lehrbuch. Springer-Verlag, Berlin u.a., 5th edition, 1989.

[131] A. Sunmonu. Implementation of a novel element-by-element finite element method on the hypercube. *Comput. Methods Appl. Mech. Engrg.*, 123:43–51, 1995.

[132] C. Temperton and A. Staniforth. An efficient two-time-level semi-Lagrangian semi-implicit integration scheme. *Quart. J. Roy. Meteor. Soc.*, 113:1025–1039, 1987.

[133] The FRAM Group. An eddy-resolving model of the Southern Ocean. *EOS Trans. of the American Geophys. Union*, 72(15):169–174, 1991.

[134] S. J. Thomas and J. Côté. Parallel semi-Lagrangian advection using PVM. personal communication (1994), 1994.

[135] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation – Foundations and Applications.* North-Holland, New York Amsterdam, 1985.

[136] K. E. Trenberth, editor. *Climate System Modeling.* Cambridge University Press, Cambridge, 1992.

[137] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.

[138] R. Verführth. A posteriori error estimators and adaptive mesh-refinement techniques for the Navier-Stokes equations. In M. D. Gunzenburger and R. A. Nicolaides, editors, *Incompressible Computational Fluid Dynamics Trends and Advances*, pages 447–475, Cambridge, 1993. Cambridge University Press.

[139] R. Wait and A. R. Mitchell. *Finite Element Analysis and Applications.* John Wiley & Sons, Chichester, 1985.

[140] W. M. Washington and C. L. Parkinson. *An Introduction to Three-Dimensional Climate Modeling.* University Science Books, Mill Valley, California, 1986.

[141] J. L. Wilkin, J. V. Mansbridge, and K. S. Hedström. An application of the capacitance matrix method to accomodate masked land areas and island circulations in a primitive equation ocean model. *Int. Journ. Num. Meth. Fluids*, 20:649–662, 1995.

[142] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *J. Comp. Phys.*, 102:211–224, 1992.

[143] D. L. Williamson and J. G. Olson. Climate simulations with a semi-Lagrangian version of the NCAR community climate model. *Mon. Wea. Rev.*, 122:1594–1610, 1994.

[144] J. O. Wolff and D. J. Olbers. The dynamical balance of the antarctic circumpolar current studied with an eddy resolving quasigeostrophic model. Preprint, Alfred-Wegener-Institut für Polar- und Meeresforschung, Bremerhaven, 1989.

[145] I. Yavneh and J. C. McWilliams. Robust multigrid solution of the shallow-water balance equations. *Jour. Comp. Phys.*, 119:1–25, 1995.

[146] C. Yeker and I. Zeid. Automatic three-dimensional finite element mesh generation via modified ray casting. *Int. J. Numer. Meth. Eng.*, 38:2573–2601, 1995.

[147] H. Yserentant. On the multi-level splitting of finite element spaces. *Numer. Math.*, 49:379–412, 1986.

[148] S. T. Zalesak. Fully multidimensional flux-corrected transport algorithms for fluids. *Jou. Comput. Phys.*, 31:335–362, 1979.

[149] Y. Zang and R. L. Street. A composite multigrid method for calculating unsteady incompressible flows in geometrically complex domains. *Int. Journ. Num. Meth. Fluids*, 20:341–361, 1995.

[150] P. Zave and W. C. Rheinboldt. Design of an adaptive, parallel finite-element system. *ACM Trans. on Math. Softw.*, 5(1):1–17, 1979.

[151] O. C. Zienkiewicz. Where now finite elements. In *Proceedings of the MAFELAP Conference 1990*, pages 1–11. Academic Press, 1991.

[152] O. C. Zienkiewicz and P. Ortiz. A split-characteristic based finite element model for the shallow water equations. *Int. Journ. Num. Meth. Fluids*, 20:1061–1080, 1995.