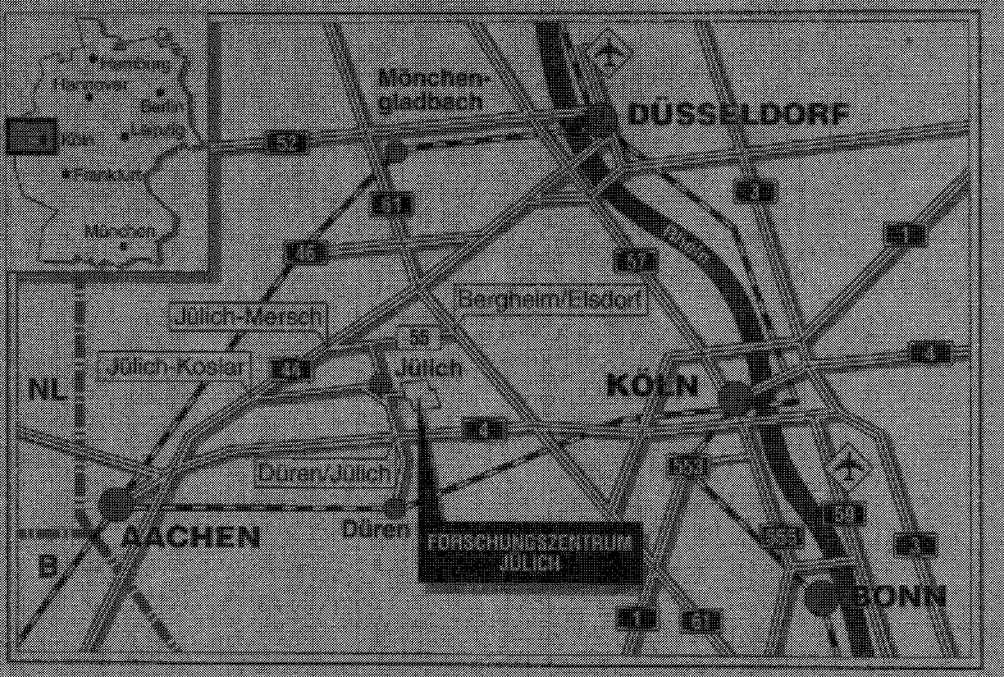
FORSCHUNGSZENTRUM JÜLICH GmbH
KFA

Zentralinstitut für Angewandte Mathematik

Parallelisierung der QMR-Methode zur Lösung linearer Gleichungssysteme

Martin Bücker



Berichte des Forschungszentrums Jülich ; 2955
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-2955

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Parallelisierung der QMR-Methode zur Lösung linearer Gleichungssysteme

Martin Bücker

Abstract

The solution of discretized partial differential equations leads to large sparse systems of linear equations. If properties of such systems are known, methods that specifically benefit from these properties could be used. E.g., the conjugate gradient method would be appropriate for symmetric positive definite systems. Unfortunately, characteristics are often unknown so that more general techniques have to be applied. This report introduces two iterative methods that are applicable to arbitrary regular non–symmetric coefficient matrices. Although different, both methods share the same basic idea called quasi–minimal residual approach. In this approach, an iterative method is defined by minimizing a factor of the residual norm rather than the residual norm as a whole. The method QMR (*Quasi–Minimal Residual*) combines the classical unsymmetric Lanczos algorithm with the quasi–minimal residual approach. Each iteration of the resulting process contains a matrix–vector product with the coefficient matrix as well as with its transpose. TFQMR (*Transpose–Free Quasi–Minimal Residual*) is derived by applying the quasi–minimal residual approach to the iterative method CGS (*Conjugate Gradient Squared*). TFQMR and CGS both involve two matrix–vector products per iteration. In contrast to QMR, matrix–vector products with the transpose of the coefficient matrix are not contained. While the two matrix–vector products of QMR are independent, this is not the case with TFQMR and CGS. It is shown how the parallelization of QMR profits by the independence of the matrix–vector products. The parallelization of the three algorithms is compared using a specific massively parallel computer, the Paragon XP/S 10.

Kurzfassung

Bei der Lösung diskretisierter partieller Differentialgleichungen entstehen überwiegend große dünnbesetzte Gleichungssysteme. Sind Eigenschaften dieser Systeme bekannt, können Lösungsmethoden verwendet werden, die diese Eigenschaften gezielt ausnutzen, wie etwa die Methode der konjugierten Gradienten für symmetrische positiv definite Systeme. Sind jedoch nur wenige oder keine Merkmale gegeben, so müssen allgemeinere Verfahren eingesetzt werden. Die vorliegende Arbeit stellt zwei Iterationsverfahren vor, die für beliebige reguläre nicht–symmetrische Koeffizientenmatrizen anwendbar sind und auf dem Ansatz der quasi–minimalen Residuen beruhen. Bei diesem Ansatz wird die Definition eines Iterationsverfahrens durch die Minimierung eines Faktors der Residuumsnorm vorgenommen. Das Verfahren QMR (*Quasi–Minimal Residual*) kombiniert den klassischen unsymmetrischen Lanczos–Algorithmus mit dem Ansatz der quasi–minimalen Residuen und enthält in jeder Iteration sowohl ein Matrix–Vektor–Produkt mit der Koeffizientenmatrix des zu lösenden Gleichungssystems als auch ein Matrix–Vektor–Produkt mit deren Transponierter. Das Verfahren TFQMR (*Transpose–Free Quasi–Minimal Residual*) fügt dem Iterationsverfahren CGS (*Conjugate Gradient Squared*) den Ansatz der quasi–minimalen Residuen hinzu. TFQMR und CGS berechnen in jeder Iteration zwei Matrix–Vektor–Produkte mit der Koeffizientenmatrix. Im Unterschied zu QMR sind in diesen beiden Algorithmen keine Matrix–Vektor–Produkte mit der Transponierten enthalten. Während die beiden Matrix–Vektor–Produkte in QMR unabhängig voneinander berechnet werden können, sind die Matrix–Vektor–Produkte der Methoden TFQMR und CGS voneinander abhängig. Die vorliegende Arbeit zeigt, wie die Unabhängigkeit der beiden Matrix–Vektor–Produkte bei der Parallelisierung von QMR ausgenutzt werden kann. Sie vergleicht die Ergebnisse einer Parallelisierung der drei iterativen Verfahren, die auf dem massiv–parallelen Rechner Paragon XP/S 10 implementiert wurden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BCG	Biconjugate Gradient
BiCGSTAB	Biconjugate Gradient Stabilized
BLAS	Basic Linear Algebra Subroutines
CG	Conjugate Gradient
CGS	Conjugate Gradient Squared
CRS	Compressed Row Storage
GFLOPS	Giga Floating Point Operations per Second
GMRES	Generalized Minimal Residual
MFLOPS	Million Floating Point Operations per Second
QMR	Quasi–Minimal Residual
RISC	Reduced Instruction Set Computer
TFQMR	Transpose–Free Quasi–Minimal Residual

ICG2D	
ICG3D	
IMPLFE	
ISR	
JPWH_991	
LNSP3937	Bezeichnungen für Matrizen, vgl. Abschnitt 3.8
MCFE	
NNC1374	
ORSREG_1	
PORES 2	
WATT1	
WEST2021	

Symbolverzeichnis

$\boldsymbol{A}$	Koeffizientenmatrix des zu lösenden Gleichungssystems
$\boldsymbol{A}^T$	Transponierte der Matrix $\boldsymbol{A}$
b	Rechte Seite des zu lösenden Gleichungssystems
N	Ordnung des zu lösenden Gleichungssystems
t	Anzahl der Nichtnull–Elemente von $\boldsymbol{A}$
x^*	Exakte Lösung von $\boldsymbol{A}x = b$, nämlich $x^* := \boldsymbol{A}^{-1}b$
$x^{(n)}$	n–ter Iterationsvektor
$r^{(n)}$	n–tes Residuum, $r^{(n)} = b - \boldsymbol{A}x^{(n)}$
$e^{(m)}$	Erster Einheitsvektor aus $\mathbb{R}^m$, $e^{(m)} := [1, 0, \ldots, 0]^T \in \mathbb{R}^m$
$\mathrm{diag}\,(\alpha_1, \alpha_2, \ldots, \alpha_r)$	Diagonalmatrix mit Diagonalelementen $\alpha_1, \alpha_2, \ldots, \alpha_r$
$[x_1 x_2 \cdots x_r]$	Matrix mit den Spaltenvektoren $x_1, x_2, \ldots, x_r$
$\mathrm{span}\,\{x_1, x_2, \ldots, x_r\}$	Menge aller Linearkombinationen der Vektoren $x_1, x_2, \ldots, x_r$
$\mathcal{K}_n\,(z, \boldsymbol{B})$	n–ter von z und $\boldsymbol{B}$ erzeugter Krylov–Teilraum
$\mathcal{P}_n$	Menge aller Polynome vom Grad kleiner oder gleich n
$\langle x, y \rangle$	Skalarprodukt der Vektoren x und y
$\|x\|$	Euklidische Vektornorm des Vektors x
$\lfloor \alpha \rfloor$	Untere Gaußklammer von $\alpha \in \mathbb{R}$
$\mathcal{O}\,(N)$	Komplexitätsklasse, vgl. [33]
p	Knotenanzahl eines Systems mit verteiltem Speicher
s	Anzahl von Matrix–Vektor–Produkten pro Iteration
$i \oplus j$	i Produkte der Form $\boldsymbol{A}x$ und j der Form $\boldsymbol{A}^T x$, in Tabelle 2.1
(n)	Kennzeichnung einer speziellen Zeile im Pseudocode

Kapitel 1

Einleitung

Viele wissenschaftliche Aufgabenstellungen erfordern die Lösung von linearen Gleichungssystemen. Diese Problemstellung tritt als Teilaufgabe beispielsweise bei der Lösung von diskretisierten partiellen Differentialgleichungen aus den verschiedensten Bereichen auf. Die Gleichungssysteme sind meist sehr groß, durchaus Millionen von Unbekannten, jedoch sind nur verhältnismäßig wenige Koeffizienten der Matrix von Null verschieden. Solche Systeme bezeichnet man als dünnbesetzt. Eine Technik, mit der dünnbesetzte Gleichungssysteme näherungsweise gelöst werden können, stellen iterative Verfahren dar. Dabei wird in aufeinanderfolgenden Schritten versucht, jeweils eine bessere Approximation der gesuchten Lösung zu bestimmen. Ein bekanntes und leistungsstarkes Iterationsverfahren für dünnbesetzte Gleichungssysteme mit symmetrisch positiv definiter Koeffizientenmatrix ist die Methode der konjugierten Gradienten [35]. Oft sind die aus Problemstellungen resultierenden Matrizen jedoch unsymmetrisch, so daß allgemeinere Methoden angewendet werden müssen. In den letzten Jahren sind durch eine Reihe von Fortschritten iterative Verfahren in den Vordergrund getreten, die auch für reguläre unsymmetrische Koeffizientenmatrizen anwendbar sind. Freund [26] stellte 1991 ein derartiges Verfahren vor, das auf dem neuen Ansatz der *quasi–minimalen Residuen*, Quasi–Minimal Residual (QMR), beruht. Dieser Ansatz ist seitdem zur Definition weiterer Iterationsverfahren verwendet worden und stellt derzeit ein aktives Forschungsgebiet dar [10, 19, 20, 31, 48].

Die Größe der Gleichungssysteme erfordert Rechner, an die hinsichtlich sowohl der Speicherkapazität als auch der Rechenleistung höchste Anforderungen gestellt werden. Um diese Probleme mit einem vernünftigen Zeitaufwand zu bearbeiten, erscheint der Einsatz von massiv–parallelen Rechnern mit bis zu Tausenden von Prozessoren aussichtsreich oder sogar notwendig. Die vorliegende Arbeit untersucht, wie die Parallelisierung der Methode der quasi–minimalen Residuen auf einem speziellen Vertreter dieser Rechnerklasse, dem Intel Paragon, durchgeführt werden kann. Die zur Parallelisierung eingesetzten Konzepte sind zwar auf den speziellen Rechner zugeschnitten, bleiben jedoch so allgemein, daß sie auf andere Parallelrechner übertragen werden können.

Diese Arbeit beschreibt die Parallelisierung von zwei Iterationsverfahren, die auf der Methode der quasi–minimalen Residuen basieren. Während das erste dieser Verfahren bereits in gerade verfügbar gewordener paralleler Software enthalten ist [11, 34] und in [53] unter dem Aspekt der parallelen Vorkonditionierung betrachtet wird, gibt es zur Parallelisierung des zweiten Verfahrens noch keine Veröffentlichungen. Mit dieser Arbeit soll die bisher erfolgreich durchgeführte Parallelisierung der Methode der konjugierten Gradienten [3, 5, 52], die im Bereich der Umwelttechnik am Forschungszentrum Jülich eingesetzt wird, auf Gleichungssysteme mit unsymmetrischer Koeffizientenmatrix erweitert werden.

In Kapitel 2 werden die mathematischen Grundlagen zur Verfügung gestellt. Dort werden zwei iterative Verfahren zur Lösung von Systemen linearer Gleichungen mit regulärer unsymmetrischer Koeffizientenmatrix vorgestellt. Ein drittes Verfahren zur Lösung der gleichen Problemstellung, das in der gesamten Arbeit als Vergleichsverfahren herangezogen wird, ist ebenfalls enthalten. Das Kapitel 3 bereitet die Parallelisierung der betrachteten Algorithmen vor. Hier werden beispielsweise der verwendete Parallelrechner, die vorgenommene Datenverteilung und das benutzte Kommunikationsschema skizziert. Darüberhinaus werden Kenngrößen von Matrizen zusammengestellt, die in den numerischen Experimenten verwendet werden. In Kapitel 4 werden experimentelle Ergebnisse beschrieben, die mit den Iterationsverfahren an den untersuchten Gleichungssystemen beobachtet worden sind. Diese Teiluntersuchung, bei der das Konvergenzverhalten der Methoden im Mittelpunkt steht, ist mit einem sequentiellen Rechner durchgeführt worden, um Eigenschaften der Iterationsverfahren von Parallelisierungsaspekten trennen zu können. Das Kapitel 5 fügt den bis dahin durchgeführten Betrachtungen solche hinzu, die die Parallelisierung betreffen. Es stellt verschiedene parallele Implementierungsvarianten vor und analysiert deren Unterschiede hinsichtlich ihres Zeitverhaltens. Anschließend werden numerische Ergebnisse und benötigte Programmausführungszeiten anhand ausgewählter Testbeispiele präsentiert, wobei die Abhängigkeit von der verwendeten Knotenanzahl des Parallelrechners im Vordergrund steht. Die Arbeit endet mit Kapitel 6, das die gewonnenen Resultate zusammenfaßt und einen Ausblick auf anzuschließende Untersuchungen gibt.

Kapitel 2

Iterative Methoden zur Lösung von Gleichungssystemen

In vielen Berechnungen von Naturwissenschaft und Technik tritt als Teilproblem das Lösen eines linearen Gleichungssystems

$$\boldsymbol{A}x = b \tag{2.1}$$

auf. Beispielsweise führt die Berechnung der Ströme und Spannungen in den Zweigen eines elektrischen Netzwerkes, das nur aus linearen Bauelementen wie Impedanzen, Induktivitäten und Kapazitäten besteht, auf ein lineares Gleichungssystem. Bei einigen Aufgabenstellungen müssen nacheinander viele Systeme mit unterschiedlicher Koeffizientenmatrix berechnet werden, wie bei der Lösung von diskretisierten, nicht–linearen partiellen Differentialgleichungen.

Lineare Gleichungssysteme werden durch die Eigenschaften ihrer Koeffizientenmatrix $\boldsymbol{A}$ charakterisiert. Enthält $\boldsymbol{A}$ überwiegend von Null verschiedene Elemente, so werden die Matrix und das zugehörige System als *dichtbesetzt* (*dense*) bezeichnet. Besteht $\boldsymbol{A}$ dagegen hauptsächlich aus Nullen, heißen Matrix und zugehöriges System *dünnbesetzt* (*sparse*). Bei einer dichtbesetzten (bzw. dünnbesetzten) Matrix spricht man von einem hohen (bzw. niedrigen) *Besetzungsgrad*. In der Praxis treten häufig dünnbesetzte Systeme auf, so z.B. bei der Diskretisierung von partiellen Differentialgleichungen nach der Methode der finiten Elemente.

Nach einer Erklärung der in dieser Arbeit verwendeten Notationen folgt in diesem Kapitel eine kurze Beschreibung der Grundlagen iterativer Methoden zur Lösung von linearen Gleichungssystemen. Darauf folgend werden zwei spezielle, unterschiedliche Iterationsverfahren ausführlicher vorgestellt und deren Gemeinsamkeit herausgearbeitet. Das Kapitel schließt mit einigen Hinweisen auf andere iterative Verfahren, die in dieser Arbeit nicht beschrieben werden.

2.1 Notationen

In der gesamten vorliegenden Arbeit werden die folgenden Notationen eingesetzt. Eine Matrix $\boldsymbol{A}$ wird durch einen fettgedruckten Großbuchstaben hervorgehoben. Die Elemente a_{ij} einer Matrix werden durch indizierte, korrespondierende Kleinbuchstaben gekennzeichnet. Ein Vektor x wird durch einen lateinischen Kleinbuchstaben dargestellt. Skalare werden mit griechischen Kleinbuchstaben identifiziert. Ein lineares Gleichungssystem wird als *symmetrisch* bezeichnet, falls die Koeffizientenmatrix $\boldsymbol{A}$ eine symmetrische Matrix ist, d.h. $\boldsymbol{A} = \boldsymbol{A}^T$. Für eine $m \times m$ Diagonalmatrix, deren Hauptdiagonale aus den Elementen $\alpha_1, \alpha_2, \ldots, \alpha_m$ besteht, wird der Ausdruck

$$\operatorname{diag}(\alpha_1, \alpha_2, \ldots, \alpha_m) := \begin{bmatrix} \alpha_1 & & & 0 \\ & \alpha_2 & & \\ & & \ddots & \\ 0 & & & \alpha_m \end{bmatrix}$$

benutzt. Die Menge aller Linearkombinationen der Vektoren $x_1, x_2, \ldots, x_r \in \mathbb{R}^m$ wird durch die Bezeichnung

$$\operatorname{span}\{x_1, x_2, \ldots, x_r\} \subseteq \mathbb{R}^m$$

ausgedrückt. Eine Matrix, die die Vektoren $x_1, x_2, \ldots, x_r \in \mathbb{R}^m$ als Spaltenvektoren enthält, wird mit

$$[x_1 x_2 \cdots x_r] \subseteq \mathbb{R}^{m \times r}$$

notiert. Für zwei Vektoren $x, y \in \mathbb{R}^m$ bezeichnet

$$\langle x, y \rangle := x^T y = x_1 y_1 + x_2 y_2 + \cdots + x_m y_m$$

das Skalarprodukt der beiden Vektoren. Als Vektornorm wird, soweit nicht anders erwähnt, die euklidische Norm

$$\|x\| := \sqrt{\langle x, x \rangle}$$

verwendet. Die mit der euklidischen Vektornorm verträgliche Matrixnorm ist

$$\|\boldsymbol{A}\| := \max_{\|x\|=1} \|\boldsymbol{A}x\|.$$

Die Menge aller Polynome vom Grad kleiner oder gleich n wird mit

$$\mathcal{P}_n := \left\{ \alpha_0 + \alpha_1 \nu + \cdots + \alpha_n \nu^n \;\middle|\; \alpha_0, \alpha_1, \ldots, \alpha_n \in \mathbb{R} \right\}$$

bezeichnet, wobei ν ein Platzhalter für einen Skalar oder auch eine quadratische Matrix ist. Für $\alpha \in \mathbb{R}$ wird die Bezeichnung $\lfloor \alpha \rfloor$ für die größte ganze Zahl, die kleiner oder gleich α ist, gewählt.

In der vorliegenden Arbeit wird ausschließlich ein System von N linearen Gleichungen in N Unbekannten behandelt. Die Koeffizientenmatrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ des linearen Systems $\boldsymbol{A}x = b$ wird durchweg als regulär angenommen. Das Gleichungssystem mit rechter Seite $b \in \mathbb{R}^N$ ist daher eindeutig lösbar. Die exakte Lösung dieses Gleichungssystems wird mit $x^* := \boldsymbol{A}^{-1}b$ gekennzeichnet. In dieser Arbeit wird durchgängig der Körper der reellen Zahlen zugrunde gelegt. Alle Verfahren, die in dieser Arbeit beschrieben werden, lassen sich jedoch auf den Körper der komplexen Zahlen erweitern, wobei dann die Begriffe *symmetrisch* durch *hermitesch*, *orthogonal* durch *unitär* und *transponiert* durch *konjugiert transponiert* ersetzt werden müssen.

In Iterationsverfahren wird der aktuelle Iterationsschritt bei Vektoren durch eingeklammerte, hochgestellte Indizierung gekennzeichnet, während bei Skalaren ein tiefgestellter Index notiert wird, etwa α_n. Insbesondere werden die Notationen

$$\begin{array}{lcll} x^{(0)} & & \text{für den Startvektor,} \\ x^{(n)} & & \text{für den } n\text{–ten Iterationsvektor und} \\ r^{(n)} & := & b - \boldsymbol{A}x^{(n)} & \text{für das } n\text{–te Residuum} \end{array}$$

verwendet.

2.2 Grundlagen iterativer Methoden

Bei den Techniken zur Lösung von linearen Gleichungssystemen werden direkte Methoden von iterativen Methoden unterschieden. Direkte Methoden formen Gleichungen solange um, bis die exakte Lösung x^* des Systems (2.1) aus der so erhaltenen Darstellung einfach bestimmt werden kann. Sofern dabei keine Vertauschungen von Zeilen oder Spalten vorgenommen werden, lassen sich die Umformungen bei direkten Methoden durch eine Faktorisierung der Koeffizientenmatrix beschreiben, wie z.B. bei der Gauß–Elimination. Ist das System dünnbesetzt, führen die Umformungen bei direkten Methoden in der Regel zu einem Auffüllen (*fill–in*) der mit Null besetzten Stellen der Matrix mit Nichtnull–Elementen. Da fill–in im ungünstigsten Fall auf allen Positionen der Matrix auftreten kann, muß bei Implementierungen, die mit statischer Speicherverwaltung arbeiten, Speicherplatz für eine vollbesetzte Matrix bereitgestellt werden. Hinweise auf spezielle Techniken, die einen niedrigen Besetzungsgrad bei direkten Methoden ausnutzen, findet man in [12].

Eine Alternative zu direkten Methoden bieten iterative Methoden, die die Koeffizientenmatrix des zu lösenden Systems nicht verändern und nur deren von Null verschiedene Elemente verwenden. Da fill–in nicht auftreten kann, sind iterative Methoden für große dünnbesetzte Systeme hinsichtlich des Speicherbedarfs günstiger als direkte Methoden. Beginnend mit einem Startvektor versuchen iterative Methoden, die Lösung eines Systems in aufeinanderfolgenden Schritten jeweils genauer zu bestimmen. Nach einer bestimmten Anzahl von Iterationen approximiert

ein Iterationsvektor die exakte Lösung im Sinne eines Abbruchkriteriums hinreichend genau. Das Verfahren endet dann mit diesem Vektor als Näherung für die exakte Lösung.

Die Idee der hier betrachteten Klasse von Iterationsverfahren besteht darin, die Lösung eines linearen Systems in eine äquivalente Optimierungsaufgabe zu transformieren. Das zu bestimmende Minimum wird iterativ in der Form

$$x^{(n)} = x^{(n-1)} + \alpha_{n-1}\, p^{(n-1)}$$

gefunden, wobei der Skalar α_{n-1} als Schrittweite und der Vektor $p^{(n-1)}$ als Suchrichtung bezeichnet wird. Bei der Methode der konjugierten Gradienten, CG (*Conjugate Gradient*), die für symmetrische positiv definite Koeffizientenmatrizen anwendbar ist, wird eine Funktion betrachtet, deren einziges Minimum an der Stelle $x^* = \boldsymbol{A}^{-1}b$ angenommen wird und deren negativer Gradient das Residuum ist. Da der negative Gradient die Richtung des steilsten Abstieges einer Funktion beschreibt, wird zur Bestimmung der aktuellen Suchrichtung das Residuum verwendet. Dabei wird das Residuum zu allen vorangehenden Suchrichtungen $\boldsymbol{A}$*–konjugiert*, d.h. $\langle p^{(n-1)}, \boldsymbol{A}p^{(j)}\rangle = 0$ für $j \in \{0, 1, \ldots, n-2\}$. Da so gewählte Suchrichtungen linear unabhängig sind, liefert CG in exakter Arithmetik die Lösung x^* nach höchstens N Schritten. In endlicher Arithmetik ist es aber auch möglich, daß durch auftretende Rundungsfehler mehr Schritte benötigt werden. CG kann daher als Iterationsverfahren angesehen werden. In der Praxis werden gewöhnlich weit weniger als N Schritte benötigt, um bereits eine gute Approximation von x^* zu erhalten. Ausführliche Beschreibungen von CG sind in der Arbeit von Hestenes und Stiefel [35] sowie in zahlreichen Lehrbüchern enthalten.

Viele iterative Methoden lassen sich durch das Aufspannen derjenigen Teilräume beschreiben, in denen die Iterationsvektoren enthalten sind. Insbesondere die Methoden, die in dieser Arbeit beschrieben werden, fallen in diese Klasse. Um sich bei den noch vorzustellenden Methoden auf deren Merkmale konzentrieren zu können, ist eine vorgeschobene Betrachtung der grundlegenden Vorgehensweise sinnvoll. Dazu wird der folgende Vektorraum eingeführt. Für einen Vektor $z \in \mathbb{R}^m$ und eine Matrix $\boldsymbol{B} \in \mathbb{R}^{m\times m}$ bezeichnet

$$\mathcal{K}_n(z, \boldsymbol{B}) = \operatorname{span}\left\{z, \boldsymbol{B}z, \boldsymbol{B}^2z, \ldots, \boldsymbol{B}^{n-1}z\right\}$$

den n–ten, von z und $\boldsymbol{B}$ erzeugten Krylov–Teilraum von $\mathbb{R}^m$. Beginnt man eine iterative Methode mit einem Startvektor $x^{(0)} \in \mathbb{R}^N$, so lassen sich die Iterationsvektoren $x^{(n)}$ zur Approximation von $x^* = \boldsymbol{A}^{-1}b$ durch

$$x^{(n)} \in x^{(0)} + \mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right)$$

charakterisieren. Dabei ist $r^{(0)} = b - \boldsymbol{A}x^{(0)}$ das zum Startvektor gehörige Residuum. Ein beliebiger Iterationsvektor kann also durch den zu seiner Iterationsanzahl korrespondierenden Krylov–Teilraum beschrieben werden, der durch das Startresiduum und die Koeffizientenmatrix erzeugt wird. Der erste wichtige Schritt bei der Entwicklung einer iterativen Methode ist daher die Konstruktion einer Basis für die

Krylov–Teilräume. Iterative Methoden, die sich durch ein Aufspannen von Krylov–Teilräumen darstellen lassen, werden wegen

$$\mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right) = \left\{\phi\left(\boldsymbol{A}\right) r^{(0)} \;\middle|\; \phi \in \mathcal{P}_{n-1}\right\}$$

auch als iterative Methoden auf Polynombasis bezeichnet. Der zweite entscheidende Aspekt bei der Entwicklung eines Iterationsverfahrens ist die Festlegung der aktuellen Iterierten, die häufig über das Residuum erfolgt. Das n–te Residuum kann ebenfalls in der polynomialen Form

$$r^{(n)} = b - \boldsymbol{A}x^{(n)} = \psi_n\left(\boldsymbol{A}\right) r^{(0)} \tag{2.2}$$

angegeben werden. Dabei gilt für das Polynom ψ_n

$$\psi_n \in \mathcal{P}_n, \quad \text{mit} \quad \psi_n(0) = 1.$$

Aufgrund der Regularität von $\boldsymbol{A}$, die in dieser Arbeit grundsätzlich vorausgesetzt wird, besteht nach (2.2) ein eindeutiger Zusammenhang zwischen dem Residuum $r^{(n)}$ und der Iterierten $x^{(n)}$. Einen Iterationsvektor kann man daher durch das zugehörige Residuum festlegen. Da das Residuum der exakten Lösung der Nullvektor ist, besteht die Idee einer iterativen Methode darin, das Residuum in jedem Schritt des Iterationsprozesses in einem zu spezifizierenden Sinne kleiner zu machen. Das Ziel ist dabei die Minimierung einer Norm des Residuums. Wie Gleichung (2.2) zeigt, muß dann in jedem Iterationsschritt ein Polynom ψ_n gewählt werden, so daß $\|r^{(n)}\|$ möglichst klein wird. Genauer formuliert, wird im n–ten Iterationsschritt das Minimierungsproblem

$$\left\|r^{(n)}\right\| = \min_{z \in x^{(0)} + \mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right)} \|b - \boldsymbol{A}z\| \tag{2.3}$$

gelöst. Diese Aufgabenstellung kann in polynomialer Schreibweise äquivalent durch

$$\left\|r^{(n)}\right\| = \min_{\psi \in \mathcal{P}_n\,,\ \psi(0)=1} \left\|\psi\left(\boldsymbol{A}\right) r^{(0)}\right\|$$

beschrieben werden. In diesem Zusammenhang ist $\|\cdot\|$ eine beliebige Norm. Ein — nicht notwendig eindeutiger — Vektor aus $x^{(0)} + \mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right)$, der das Minimierungsproblem (2.3) löst, wird dann als aktueller Iterationsvektor genommen.

Eine andere Idee zur Definition eines Iterationsverfahrens basiert nicht auf der Minimierung einer Norm des Residuums sondern auf der Wahl des n–ten Residuums so, daß es orthogonal zu allen Vektoren eines Vektorraums der Dimension n ist. Diese *Galerkin–ähnliche* Bedingung besitzt den Nachteil, daß in gewissen Iterationsschritten möglicherweise keine neue Iterierte gefunden werden kann. In diesem Falle muß das Verfahren abbrechen, ohne eine Approximation der Lösung gefunden zu haben. Die Methode der Bikonjugierten Gradienten, BCG (*Biconjugate Gradient*) [43], beispielsweise, benutzt eine solche Bedingung zur Definition ihrer Iterierten.

2.3 Transpositionsfreie QMR–Methode

Sind Eigenschaften eines Gleichungssystems $\boldsymbol{A}x = b$ bekannt, so können Lösungsmethoden verwendet werden, die die vorhandenen Eigenschaften gezielt ausnutzen (z.B. CG für symmetrische positiv definite Systeme). Sind jedoch nur wenige oder keine Merkmale gegeben, so müssen allgemeinere Verfahren eingesetzt werden. Die in diesem Abschnitt vorzustellende Methode stellt ein Iterationsverfahren dar, das für beliebige reguläre Koeffizientenmatrizen anwendbar ist. Das Verfahren enthält die Koeffizientenmatrix ausschließlich in der Form von Matrix–Vektor–Produkten mit $\boldsymbol{A}$. Insbesondere enthält sie kein Matrix–Vektor–Produkt mit der Transponierten der Koeffizientenmatrix und wird deshalb als *transpositionsfrei* bezeichnet. Die transpositionsfreie Methode der Quasi–Minimalen Residuen, TFQMR (*Transpose–Free Quasi–Minimal Residual*), besitzt gegenüber Verfahren, die ein Produkt der Form $\boldsymbol{A}^T x$ enthalten, dann einen Vorteil, wenn das Matrix–Vektor–Produkt mit $\boldsymbol{A}^T$ nur sehr ineffizient implementiert werden kann. In Abschnitt 2.4 wird das Iterationsverfahren QMR zur gleichen Problemstellung vorgestellt. QMR und TFQMR verfolgen zwar bei der Definition ihrer Iterierten die gleiche Grundidee, sie stellen jedoch zwei unterschiedliche, völlig eigenständige iterative Methoden dar.

Die Methode TFQMR ist von Freund [16, 19] entwickelt worden. TFQMR wird aus dem Algorithmus CGS (*Conjugate Gradient Squared*) [54] hergeleitet. CGS löst ein lineares System mit regulärer Koeffizientenmatrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ und einer rechten Seite $b \in \mathbb{R}^N$ durch den Algorithmus in Abbildung 2.1. Dabei werden die Iterationsvektoren, die die Lösung des Systems approximieren sollen, in der Form

$$x^{(n)} = x^{(n-1)} + \alpha_{n-1} \left(u^{(n-1)} + q^{(n)} \right)$$

mit den Suchrichtungen $u^{(n-1)}$ und $q^{(n)}$ gebildet. Die zwei grundlegenden Teile einer iterativen Methode auf Polynombasis sind die Konstruktion der Krylov–Teilräume und die Definition der aktuellen Iterierten. In CGS werden die Suchrichtungen zum Aufspannen der Krylov–Teilräume verwendet. Die Idee von TFQMR basiert darauf, die gleiche Konstruktion der Krylov–Teilräume wie CGS zu benutzen, die beiden Suchrichtungen aber auf eine andere Art auszunutzen. CGS erzeugt in jedem Iterationsschritt zwei Suchrichtungen, die jedoch lediglich als Linearkombination zur Berechnung der aktuellen Iterierten verwendet werden. Durch die Bildung der Linearkombination kann möglicherweise Information, die in den Suchrichtungen einzeln enthalten ist, verlorengehen. TFQMR bildet daher zu jeder Suchrichtung eine eigene Iterierte. Dadurch werden pro Iteration in TFQMR zwei Iterierte, in CGS jedoch nur eine Iterierte bestimmt.

Normalerweise endet der Algorithmus CGS nach einer endlichen Anzahl von Iterationen mit der Approximation für die exakte Lösung. Diese Anzahl werde mit $n_\star$ bezeichnet. Generell kann jedoch nicht ausgeschlossen werden, daß in CGS an den in Abbildung 2.1 mit (1) und (3) gekennzeichneten Stellen eine Division durch Null

Algorithmus *CGS*

/* Der Algorithmus löst ein lineares Gleichungssystem $\boldsymbol{A}x = b$, */
/* mit $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ und $b \in \mathbb{R}^N$. Die exakte Lösung x^* wird */
/* durch die Iterationsvektoren $x^{(n)}$ approximiert. */

Wähle $x^{(0)} \in \mathbb{R}^N$
$r^{(0)} = b - \boldsymbol{A}x^{(0)}$
$u^{(0)} = r^{(0)}$
$p^{(0)} = r^{(0)}$
$v^{(0)} = \boldsymbol{A}p^{(0)}$
Wähle $\tilde{r}^{(0)}$ *so, daß* $\rho_0 = \langle \tilde{r}^{(0)}, r^{(0)} \rangle \neq 0$
for $n = 1, 2, 3, \ldots$ **do**
$\sigma_{n-1} = \langle \tilde{r}^{(0)}, v^{(n-1)} \rangle$
(1) $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$
(2) $q^{(n)} = u^{(n-1)} - \alpha_{n-1}\, v^{(n-1)}$
$x^{(n)} = x^{(n-1)} + \alpha_{n-1}\left(u^{(n-1)} + q^{(n)}\right)$
$r^{(n)} = r^{(n-1)} - \alpha_{n-1}\boldsymbol{A}\left(u^{(n-1)} + q^{(n)}\right)$
if $x^{(n)}$ *konvergiert* **then** *beende Iterationsverfahren*
$\rho_n = \langle \tilde{r}^{(0)}, r^{(n)} \rangle$
(3) $\beta_n = \rho_n/\rho_{n-1}$
(4) $u^{(n)} = r^{(n)} + \beta_n q^{(n)}$
(5) $p^{(n)} = u^{(n)} + \beta_n\left(q^{(n)} + \beta_n p^{(n-1)}\right)$
$v^{(n)} = \boldsymbol{A}p^{(n)}$
end for
end /* *CGS* */

Abbildung 2.1: CGS Algorithmus

erfolgt. Ein Abbruch, der als *breakdown* bezeichnet wird, ist in diesem Falle notwendig. Ein breakdown kommt in der Praxis nur selten vor und kann durch *look–ahead* Strategien [9, 48] verhindert werden. Neben der Bedingung $n \in \{1, 2, \ldots, n_*\}$ wird bei der Herleitung des Verfahrens im folgendem vorausgesetzt, daß breakdowns nicht auftreten. Für die Skalare α_{n-1}, die in der Form $\alpha_{n-1} = \rho_{n-1}/\sigma_{n-1}$ gebildet werden, bedeutet dies

$$\alpha_{n-1} \neq 0 \qquad \forall\, n \in \{1, 2, \ldots, n_*\}. \tag{2.4}$$

Für die Suchrichtungen $u^{(n-1)}$ und $q^{(n)}$ aus CGS gelten die Beziehungen

$$u^{(n-1)} = \varphi_{n-1}(\boldsymbol{A})\,\psi_{n-1}(\boldsymbol{A})\,r^{(0)} \quad \text{und} \tag{2.5}$$
$$q^{(n)} = \varphi_{n}(\boldsymbol{A})\,\psi_{n-1}(\boldsymbol{A})\,r^{(0)}, \tag{2.6}$$

wobei die Polynome $\varphi_n, \psi_n \in \mathcal{P}_n$ in [54] hergeleitet sind und zusammen mit den Skalaren α_{n-1} und β_n aus Abbildung 2.1 den Rekursionen

$$\varphi_n(\nu) = \varphi_{n-1}(\nu) - \alpha_{n-1}\nu\,\psi_{n-1}(\nu) \quad \text{und} \tag{2.7}$$
$$\psi_n(\nu) = \varphi_n(\nu) + \beta_n\psi_{n-1}(\nu) \tag{2.8}$$

mit $\varphi_0 \equiv \psi_0 \equiv 1$ genügen. Die Suchrichtungen $u^{(n-1)}$ und $q^{(n)}$ aus CGS werden nun benutzt, um das neue Verfahren TFQMR herzuleiten. Dazu werden zunächst für $n \in \{1, 2, \ldots, n_\star\}$ zwei Vektoren definiert, nämlich der Vektor

$$y^{(m)} := \begin{cases} u^{(n-1)} & \text{falls} \quad m = 2n-1 \\ q^{(n)} & \text{falls} \quad m = 2n \end{cases} \tag{2.9}$$

sowie unter Zuhilfenahme der in (2.7) und (2.8) definierten Polynome der Vektor

$$w^{(m)} := \begin{cases} r^{(0)} & \text{falls} \quad m = 1 \\ (\varphi_n(\boldsymbol{A}))^2\,r^{(0)} & \text{falls} \quad m = 2n+1 \\ \varphi_n(\boldsymbol{A})\,\varphi_{n-1}(\boldsymbol{A})\,r^{(0)} & \text{falls} \quad m = 2n. \end{cases} \tag{2.10}$$

Es wird nun ein Zusammenhang zwischen den Vektoren $y^{(m)}$ und $w^{(m)}$ hergestellt. Dazu werden die Gleichungen (2.5) – (2.7), (2.9) und (2.10) verwendet.

Falls $m = 2n$, gilt

$$\begin{aligned} w^{(m)} - w^{(m+1)} &= \varphi_n(\boldsymbol{A})\,\varphi_{n-1}(\boldsymbol{A})\,r^{(0)} - (\varphi_n(\boldsymbol{A}))^2\,r^{(0)} \\ &= \varphi_n(\boldsymbol{A})\,[\varphi_n(\boldsymbol{A}) + \alpha_{n-1}\boldsymbol{A}\psi_{n-1}(\boldsymbol{A})]\,r^{(0)} - (\varphi_n(\boldsymbol{A}))^2\,r^{(0)} \\ &= \alpha_{n-1}\boldsymbol{A}\varphi_n(\boldsymbol{A})\,\psi_{n-1}(\boldsymbol{A})\,r^{(0)} \\ &= \alpha_{n-1}\boldsymbol{A}q^{(n)} \\ &= \alpha_{m/2-1}\boldsymbol{A}y^{(m)}. \end{aligned} \tag{2.11}$$

Falls $m = 2n+1$, so gilt für die Differenz

$$\begin{aligned} w^{(m)} - w^{(m+1)} &= (\varphi_n(\boldsymbol{A}))^2\,r^{(0)} - \varphi_{n+1}(\boldsymbol{A})\,\varphi_n(\boldsymbol{A})\,r^{(0)} \\ &= (\varphi_n(\boldsymbol{A}))^2\,r^{(0)} - [\varphi_n(\boldsymbol{A}) - \alpha_n\boldsymbol{A}\psi_n(\boldsymbol{A})]\,\varphi_n(\boldsymbol{A})\,r^{(0)} \\ &= \alpha_n\boldsymbol{A}\varphi_n(\boldsymbol{A})\,\psi_n(\boldsymbol{A})\,r^{(0)} \\ &= \alpha_n\boldsymbol{A}u^{(n)} \\ &= \alpha_{(m-1)/2}\boldsymbol{A}y^{(m)}. \end{aligned} \tag{2.12}$$

Für $m = 1$ ist wegen $\varphi_0 \equiv 1$ Gleichung (2.12) ebenfalls gültig. Und daher lassen

sich die Gleichungen (2.11) und (2.12) für $m \in \{1, 2, 3, \ldots, 2n_\star\}$ zu

$$w^{(m)} - w^{(m+1)} = \alpha_{\lfloor (m-1)/2 \rfloor} \boldsymbol{A} y^{(m)} \tag{2.13}$$

zusammenfassen. Nach Gleichung (2.4) sind die Skalare α_i von Null verschieden. Daher kann in Gleichung (2.13) durch sie dividiert werden. Mit der $N \times m$ Matrix

$$\boldsymbol{Y}^{(m)} := \left[y^{(1)} y^{(2)} \cdots y^{(m)} \right]$$

und der $N \times (m+1)$ Matrix

$$\boldsymbol{W}^{(m+1)} := \left[w^{(1)} w^{(2)} \cdots w^{(m+1)} \right]$$

läßt sich Gleichung (2.13) in Matrix–Notation durch

$$\boldsymbol{W}^{(m+1)} \boldsymbol{B}^{(m)} = \boldsymbol{A} \boldsymbol{Y}^{(m)} \tag{2.14}$$

darstellen, wobei die $(m+1) \times m$ Matrix

$$\boldsymbol{B}^{(m)} := \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \\ & \boldsymbol{0} & & -1 \end{bmatrix} \left(\operatorname{diag} \left(\alpha_0, \alpha_0, \alpha_1, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor} \right) \right)^{-1} \tag{2.15}$$

eingeführt wurde.

Die Gleichungen (2.5), (2.6) und (2.9) zeigen, daß der Vektor $y^{(m)}$ ein Polynom in der Matrix $\boldsymbol{A}$ ist. Nach (2.4), (2.7) und (2.8) besitzt dieses Polynom den Grad $m-1$. Daraus folgt der Zusammenhang

$$\mathcal{K}_m \left(r^{(0)}, \boldsymbol{A} \right) = \operatorname{span} \left\{ y^{(1)}, y^{(2)}, \ldots, y^{(m)} \right\} = \left\{ \boldsymbol{Y}^{(m)} z \;\middle|\; z \in \mathbb{R}^m \right\}.$$

Jeder Iterationsvektor $x^{(m)} \in x^{(0)} + \mathcal{K}_m \left(r^{(0)}, \boldsymbol{A} \right)$ kann daher in der Form

$$x^{(m)} = x^{(0)} + \boldsymbol{Y}^{(m)} z \quad \text{für ein} \quad z \in \mathbb{R}^m$$

dargestellt werden. Für das zur m–ten Iterierten gehörige Residuum gilt

$$\begin{aligned} r^{(m)} &= b - \boldsymbol{A} x^{(m)} \\ &= b - \boldsymbol{A} \left(x^{(0)} + \boldsymbol{Y}^{(m)} z \right) \\ &= r^{(0)} - \boldsymbol{A} \boldsymbol{Y}^{(m)} z \quad \text{für ein} \quad z \in \mathbb{R}^m. \end{aligned}$$

Mit dem Einheitsvektor $e^{(m+1)} := [1, 0, \ldots, 0]^T \in \mathbb{R}^{m+1}$ ist das Residuum aufgrund

der Gleichungen (2.10) und (2.14) auch in der Form

$$\begin{aligned} r^{(m)} &= w^{(1)} - \boldsymbol{W}^{(m+1)}\boldsymbol{B}^{(m)}z \\ &= \boldsymbol{W}^{(m+1)}\left(e^{(m+1)} - \boldsymbol{B}^{(m)}z\right) \quad \text{für ein} \quad z \in \mathbb{R}^m \end{aligned}$$

darstellbar. Wie in dem Abschnitt über Krylov–Teilräume erwähnt wurde, besteht das Ziel darin, das Residuum $r^{(m)}$ klein zu machen. Dies kann durch eine geeignete Wahl des freien Parametervektors z geschehen. Die Bestimmung eines Vektors $z \in \mathbb{R}^m$ so, daß die Norm des Residuums $\|r^{(m)}\|$ minimal wird, stellt ein lineares Ausgleichsproblem dar. Diese Problemstellung kann durch Standardverfahren, wie etwa die Methode der Normalgleichungen oder über eine QR–Zerlegung der $N \times m$ Matrix $\boldsymbol{W}^{(m+1)}\boldsymbol{B}^{(m)}$ gelöst werden. Dazu sind allerdings $\mathcal{O}(Nm^2)$ Operationen notwendig [32]. In einem Iterationsverfahren, das die Norm $\|r^{(m)}\|$ minimiert, müßte ein lineares Ausgleichsproblem in jedem Iterationsschritt gelöst werden. Da dies zu aufwendig wäre, wird stattdessen das Residuum in Produktform dargestellt und eine Minimierung, die mit weniger Aufwand zu realisieren ist, nur auf einen Faktor des Residuums angewendet.

Bei der Faktorisierung des Residuums wird für $i \in \{1, 2, \ldots, m+1\}$ ein Satz von Parametern $\lambda_i > 0$ eingeführt, der in der Diagonalmatrix

$$\boldsymbol{\Lambda}^{(m+1)} := \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{m+1}) \tag{2.16}$$

enthalten ist. Zusätzlich werden zwei weitere Bezeichnungen eingeführt. Der Vektor $f^{(m+1)} \in \mathbb{R}^{m+1}$, der durch

$$f^{(m+1)} := \lambda_1 e^{(m+1)} \tag{2.17}$$

definiert ist, enthält in seiner ersten Komponente den Parameter λ_1. Außerdem geht der Parametersatz $\lambda_1, \lambda_2, \ldots, \lambda_{m+1}$ in die $(m+1) \times m$ Matrix $\boldsymbol{H}^{(m)}$ ein, die durch

$$\boldsymbol{H}^{(m)} := \boldsymbol{\Lambda}^{(m+1)}\boldsymbol{B}^{(m)} \tag{2.18}$$

definiert ist. Mit den beiden neuen Größen $f^{(m+1)}$ und $\boldsymbol{H}^{(m)}$ kann das Residuum in der Form

$$r^{(m)} = \boldsymbol{W}^{(m+1)}{\boldsymbol{\Lambda}^{(m+1)}}^{-1}\left(f^{(m+1)} - \boldsymbol{H}^{(m)}z\right) \quad \text{für ein} \quad z \in \mathbb{R}^m \tag{2.19}$$

angegeben werden. Der m–te Iterationsvektor $x^{(m)}$ von TFQMR wird nun durch

$$x^{(m)} = x^{(0)} + \boldsymbol{Y}^{(m)}z^{(m)} \tag{2.20}$$

definiert, wobei der Vektor $z^{(m)}$ die Lösung des linearen Ausgleichsproblems

$$\tau_m := \left\| f^{(m+1)} - \boldsymbol{H}^{(m)}z^{(m)} \right\| = \min_{z \in \mathbb{R}^m} \left\| f^{(m+1)} - \boldsymbol{H}^{(m)}z \right\| \tag{2.21}$$

ist. Durch die Gleichungen (2.15), (2.16) und (2.18) ist die $(m+1) \times m$ Matrix $\boldsymbol{H}^{(m)}$ definiert. Offensichtlich besitzt sie den vollen Rang m. Daher ist das Ausgleichsproblem (2.21) eindeutig lösbar. Der Lösungsvektor $z^{(m)}$ hängt von der Wahl der

Parameter λ_i ab. In dieser Arbeit wird ausschließlich der Fall

$$\lambda_i = \left\| w^{(i)} \right\|, \quad \text{für} \quad i \in \{1, 2, \ldots, m+1\} \tag{2.22}$$

betrachtet. Dies bedeutet eine Normierung der Spalten von $\boldsymbol{W}^{(m+1)} \boldsymbol{\Lambda}^{(m+1)^{-1}}$, also desjenigen Faktors des Residuums in (2.19), auf den die Minimierung des Residuums nicht bezogen wird.

Die Berechnung der TFQMR–Iterierten $x^{(m)}$, die durch die Gleichungen (2.20) und (2.21) definiert sind, erfolgt über Rekursionen. Zur Herleitung dieser Rekursionen sei $\tilde{\boldsymbol{H}}^{(m)}$ diejenige $m \times m$ Matrix, die aus $\boldsymbol{H}^{(m)}$ durch Streichen der letzten Zeile entsteht, nämlich

$$\tilde{\boldsymbol{H}}^{(m)} := \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)\, \boldsymbol{J} \left(\operatorname{diag}\left(\alpha_0, \alpha_0, \alpha_1, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor} \right) \right)^{-1}, \tag{2.23}$$

wobei die $m \times m$ Matrix

$$\boldsymbol{J} := \begin{bmatrix} 1 & & \boldsymbol{0} \\ -1 & \ddots & \\ & \ddots & \\ \boldsymbol{0} & & -1 \quad 1 \end{bmatrix}$$

verwendet wurde. Über die Matrix $\tilde{\boldsymbol{H}}^{(m)}$ und den in Gleichung (2.17) definierten Vektor $f^{(m)}$ werde nun der Hilfsvektor $\tilde{z}^{(m)} \in \mathbb{R}^m$ durch

$$\tilde{z}^{(m)} := \tilde{\boldsymbol{H}}^{(m)^{-1}} f^{(m)} \tag{2.24}$$

definiert. Nach dieser Definition ist $\tilde{z}^{(m)}$ das λ_1–fache der ersten Spalte der Matrix $\tilde{\boldsymbol{H}}^{(m)^{-1}}$. Um den Vektor $\tilde{z}^{(m)}$ berechnen zu können, wird daher nach Gleichung (2.23) die Inverse von $\boldsymbol{J}$ benötigt, die sich zu

$$\boldsymbol{J}^{-1} = \begin{bmatrix} 1 & & \boldsymbol{0} \\ \vdots & \ddots & \\ 1 & \cdots & 1 \end{bmatrix}$$

bestimmen läßt. Betrachtet man die erste Spalte der Matrix $\boldsymbol{J}^{-1}$ und verwendet die Beziehung

$$\tilde{\boldsymbol{H}}^{(m)^{-1}} = \operatorname{diag}\left(\alpha_0, \alpha_0, \alpha_1, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor} \right) \boldsymbol{J}^{-1} \left(\operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m) \right)^{-1},$$

so folgt für den durch Gleichung (2.24) definierten Vektor die Darstellung

$$\tilde{z}^{(m)} = \left[\alpha_0, \alpha_0, \alpha_1, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor} \right]^T. \tag{2.25}$$

Um den Ausdruck $\|f^{(m+1)} - \boldsymbol{H}^{(m)}\tilde{z}^{(m)}\|$ berechnen zu können, multipliziert man die Gleichung (2.25) von links mit $\left(\operatorname{diag}\left(\alpha_0, \alpha_0, \alpha_1, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor}\right)\right)^{-1}$ und erhält so denjenigen Vektor aus $\mathbb{R}^m$, der in allen Komponenten eine Eins enthält. Nach der Definition von $\boldsymbol{B}^{(m)}$ aus Gleichung (2.15) ist dieser Vektor in

$$\boldsymbol{B}^{(m)}\tilde{z}^{(m)} = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \\ & & & -1 \end{bmatrix} \left(\operatorname{diag}\left(\alpha_0, \alpha_0, \alpha_1, \alpha_1, \ldots, \alpha_{\lfloor (m-1)/2 \rfloor}\right)\right)^{-1} \tilde{z}^{(m)}$$

enthalten. Mit den Gleichungen (2.16) und (2.18) folgt daher für das Produkt

$$\boldsymbol{H}^{(m)}\tilde{z}^{(m)} = \operatorname{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_{m+1}\right) \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \\ & & & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_1 \\ 0 \\ \vdots \\ 0 \\ -\lambda_{m+1} \end{bmatrix}.$$

Unter Verwendung der Definition von $f^{(m+1)}$ aus Gleichung (2.17) folgt dann

$$\left\| f^{(m+1)} - \boldsymbol{H}^{(m)}\tilde{z}^{(m)} \right\| = \lambda_{m+1}. \tag{2.26}$$

Freund [19] zeigt, daß mit dem durch die Gleichung (2.24) definierten Hilfsvektor $\tilde{z}^{(m)}$ die Lösung des linearen Ausgleichsproblems (2.21) formuliert werden kann. Mit den Definitionen

$$\vartheta_m := \frac{1}{\tau_{m-1}} \left\| f^{(m+1)} - \boldsymbol{H}^{(m)}\tilde{z}^{(m)} \right\| \quad \text{und} \tag{2.27}$$

$$\gamma_m := \frac{1}{\sqrt{1 + {\vartheta_m}^2}} \tag{2.28}$$

besitzt der Lösungsvektor $z^{(m)}$ des Problems (2.21) die Darstellung

$$z^{(m)} = \left(1 - {\gamma_m}^2\right) \begin{bmatrix} z^{(m-1)} \\ 0 \end{bmatrix} + {\gamma_m}^2 \tilde{z}^{(m)}, \tag{2.29}$$

wobei der mit eckigen Klammern versehene Ausdruck einen Vektor aus $\mathbb{R}^m$ darstellt, dessen erste $m-1$ Komponenten die Komponenten des Vektors $z^{(m-1)}$ enthalten

und dessen letzte Komponente mit Null besetzt ist. An der Stelle des Lösungsvektors $z^{(m)}$ wird in Gleichung (2.21) das Minimum

$$\tau_m = \tau_{m-1}\vartheta_m\gamma_m \tag{2.30}$$

angenommen. Tatsächlich beweist Freund, daß diese Lösung sogar für ein allgemeineres Ausgleichsproblem gilt. Um ein Iterationsverfahren für die Iterierten von TFQMR zu erhalten, werden mit dem Hilfsvektor $\tilde{z}^{(m)}$ aus Gleichung (2.24) die Hilfsiterierten

$$\tilde{x}^{(m)} = x^{(0)} + \boldsymbol{Y}^{(m)}\tilde{z}^{(m)} \tag{2.31}$$

definiert. Multipliziert man Gleichung (2.29) von links mit $\boldsymbol{Y}^{(m)}$ und addiert $x^{(0)}$ hinzu, so folgt

$$x^{(0)} + \boldsymbol{Y}^{(m)}z^{(m)} = \left(1-{\gamma_m}^2\right)\left(x^{(0)} + \boldsymbol{Y}^{(m)}\begin{bmatrix} z^{(m-1)} \\ 0 \end{bmatrix}\right) + {\gamma_m}^2\left(x^{(0)} + \boldsymbol{Y}^{(m)}\tilde{z}^{(m)}\right).$$

Nach Gleichung (2.20) sind daher die Iterierten von TFQMR und die Hilfsiterierten aus Gleichung (2.31) in der Form

$$x^{(m)} = \left(1-{\gamma_m}^2\right)x^{(m-1)} + {\gamma_m}^2\tilde{x}^{(m)} \tag{2.32}$$

miteinander verbunden. Mit den beiden folgenden Definitionen

$$\eta_m := {\gamma_m}^2\alpha_{\lfloor(m-1)/2\rfloor} \tag{2.33}$$

$$d^{(m)} := \frac{1}{\alpha_{\lfloor(m-1)/2\rfloor}}\left(\tilde{x}^{(m)} - x^{(m-1)}\right) \tag{2.34}$$

können nun die Iterierten nach Gleichung (2.32) in die äquivalente Darstellung

$$x^{(m)} = x^{(m-1)} + \eta_m d^{(m)} \tag{2.35}$$

überführt werden. Dies ist eine Rekursion für die TFQMR–Iterierten $x^{(m)}$. Sie eignet sich für ein Iterationsverfahren, sobald eine rekursive Vorschrift zur Erzeugung der Vektoren $d^{(m)}$ vorhanden ist. Mit den Gleichungen (2.25) und (2.31) gilt für die Hilfsiterierte $\tilde{x}^{(m)}$ die Beziehung

$$\begin{aligned}
\tilde{x}^{(m)} &= x^{(0)} + \left[y^{(1)}y^{(2)}\cdots y^{(m)}\right]\left[\alpha_0,\alpha_0,\alpha_1,\ldots,\alpha_{\lfloor(m-1)/2\rfloor}\right]^T \\
&= x^{(0)} + \boldsymbol{Y}^{(m-1)}z^{(m-1)} + \alpha_{\lfloor(m-1)/2\rfloor}y^{(m)} \\
&= \tilde{x}^{(m-1)} + \alpha_{\lfloor(m-1)/2\rfloor}y^{(m)}.
\end{aligned} \tag{2.36}$$

Benutzt man nun die Gleichungen (2.28) und (2.33) – (2.36), dann folgt für den Vektor $d^{(m)}$ die Rekursion

$$\begin{aligned}
d^{(m)} &= \frac{1}{\alpha_{\lfloor(m-1)/2\rfloor}}\left(\tilde{x}^{(m-1)} + \alpha_{\lfloor(m-1)/2\rfloor}y^{(m)} - x^{(m-2)} - \eta_{m-1}d^{(m-1)}\right) \\
&= y^{(m)} + \frac{1}{\alpha_{\lfloor(m-1)/2\rfloor}}\left(\alpha_{\lfloor(m-2)/2\rfloor}d^{(m-1)} - \eta_{m-1}d^{(m-1)}\right) \\
&= y^{(m)} + \frac{\left(1-{\gamma_{m-1}}^2\right)\eta_{m-1}}{{\gamma_{m-1}}^2\alpha_{\lfloor(m-1)/2\rfloor}}d^{(m-1)} \\
&= y^{(m)} + \frac{{\vartheta_m}^2\,\eta_{m-1}}{\alpha_{\lfloor(m-1)/2\rfloor}}d^{(m-1)}.
\end{aligned} \tag{2.37}$$

Damit sind nun alle Vorbereitungen getroffen, um aus dem in Abbildung 2.1 skizzierten Algorithmus CGS die neue Methode TFQMR zu konstruieren. Die Rekursion für das Residuum von CGS, das in der Formulierung von CGS mit $r^{(n)}$ nun aber mit ${r_{CGS}}^{(n)}$ gekennzeichnet ist, wird jetzt nicht mehr benötigt. Der Vektor ${r_{CGS}}^{(n)}$ ist jedoch noch in der in Abbildung 2.1 mit (4) gekennzeichneten Anweisung enthalten. In [54] wird für das Residuum von CGS die Beziehung ${r_{CGS}}^{(n)} = \left(\varphi_n\left(\boldsymbol{A}\right)\right)^2 {r_{CGS}}^{(0)}$, hergeleitet, so daß nach der Definition der $w^{(n)}$ aus Gleichung (2.10) das Residuum durch

$$ {r_{CGS}}^{(n)} = w^{(2n+1)} \tag{2.38} $$

ersetzt wird. Den CGS Algorithmus kann man nun in der folgenden Weise modifizieren. Die Rekursionen für die Suchrichtungen $q^{(n)}$ und $u^{(n)}$, die in Abbildung 2.1 mit (2) und (4) gekennzeichnet sind, lassen sich mit den Gleichungen (2.9) und (2.38) als

$$ y^{(2n)} = y^{(2n-1)} - \alpha_{n-1} v^{(n-1)} \tag{2.39} $$

$$ y^{(2n+1)} = w^{(2n+1)} + \beta_n y^{(2n)} \tag{2.40} $$

ausdrücken. Multipliziert man die in Abbildung 2.1 mit (5) gekennzeichnete Anweisung für $p^{(n)}$ von links mit der Matrix $\boldsymbol{A}$, so erhält man für den Vektor $v^{(n)}$ die rekursive Formulierung

$$ v^{(n)} = \boldsymbol{A} y^{(2n+1)} + \beta_n \left(\boldsymbol{A} y^{(2n)} + \beta_n v^{(n-1)}\right). \tag{2.41} $$

Nach Gleichung (2.13) werden die Vektoren $w^{(m)}$ rekursiv durch

$$ w^{(m+1)} = w^{(m)} - \alpha_{\lfloor (m-1)/2 \rfloor} \boldsymbol{A} y^{(m)} \tag{2.42} $$

berechnet. Damit sind auch die Skalare ϑ_m bekannt, denn mit den Gleichungen (2.26), (2.27) und der Wahl der Parameter (2.22) gilt

$$ \vartheta_m = \frac{\left\| w^{(m+1)} \right\|}{\tau_{m-1}}. \tag{2.43} $$

Fügt man die Gleichungen (2.28), (2.30), (2.33), (2.35), (2.37), (2.39) – (2.43) in den CGS Algorithmus ein, dann folgt das Verfahren TFQMR. Der resultierende Algorithmus ist in zwei Abbildungen dargestellt. In der Abbildung 2.2 ist die Initialisierungsphase von TFQMR notiert. Die Abbildung 2.3 beschreibt die Iterationsphase, bei der auf einen Unterschied zu CGS hingewiesen sei. Wie die Abbildung 2.1 veranschaulicht, wird in CGS in einer äußeren Schleife eine Iterierte berechnet. Im Unterschied dazu werden in TFQMR in einer äußeren Schleife, die in Abbildung 2.3 über die Variable n läuft, zwei Iterierte erzeugt. Von nun an wird für den Rest dieser Arbeit der Ausdruck *Iteration* jeweils für eine äußere Schleife benutzt. Daher wird im folgenden davon gesprochen, daß CGS eine Iterierte pro Iteration bildet, während bei TFQMR zwei Iterierte pro Iteration berechnet werden. Da sowohl Rechenaufwand als auch Speicherbedarf von CGS und TFQMR in einer äußeren

Schleife vergleichbar sind, ist diese Festlegung des Ausdrucks *Iteration* gerechtfertigt.

Abschließend sei darauf hingewiesen, daß Abschätzungen für die Norm des n–ten TFQMR–Residuums existieren. In [18] ist erstmals ein solches Ergebnis über das Konvergenzverhalten einer transpositionsfreien, Lanczos–ähnlichen Methode zur Lösung von unsymmetrischen linearen Gleichungssystemen enthalten.

Subroutine *Initialisiere (TFQMR)*
$W\ddot{a}hle \quad x^{(0)} \in \mathbb{R}^N$
$r^{(0)} = b - \boldsymbol{A}x^{(0)}$
$y^{(1)} = r^{(0)}$
$w^{(1)} = r^{(0)}$
$v^{(0)} = \boldsymbol{A}y^{(1)}$
$d^{(0)} = 0$
$\tau_0 = \left\| r^{(0)} \right\|$
$\vartheta_0 = 0$
$\eta_0 = 0$
$W\ddot{a}hle \quad \tilde{r}^{(0)} \quad so,\ da\beta \quad \rho_0 = \langle \tilde{r}^{(0)}, r^{(0)} \rangle \neq 0$
end /* *Initialisiere* */

Abbildung 2.2: TFQMR Initialisierungen

Algorithmus *TFQMR*

/* Der Algorithmus löst ein lineares Gleichungssystem $\boldsymbol{A}x = b$, */

/* mit $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ und $b \in \mathbb{R}^N$. Die exakte Lösung x^* wird */

/* durch die Iterationsvektoren $x^{(n)}$ approximiert. */

Initialisiere(TFQMR)

for $n = 1, 2, 3, \ldots$ **do**

$\sigma_{n-1} = \langle \tilde{r}^{(0)}, v^{(n-1)} \rangle$

$\alpha_{n-1} = \rho_{n-1} / \sigma_{n-1}$

$y^{(2n)} = y^{(2n-1)} - \alpha_{n-1} v^{(n-1)}$

for $m = 2n-1, 2n$ **do**

$w^{(m+1)} = w^{(m)} - \alpha_{n-1} \boldsymbol{A} y^{(m)}$

$\vartheta_m = \left\| w^{(m+1)} \right\| \Big/ \tau_{m-1}$

$\gamma_m = 1 \Big/ \sqrt{1 + {\vartheta_m}^2}$

$\tau_m = \tau_{m-1} \vartheta_m \gamma_m$

$\eta_m = {\gamma_m}^2 \, \alpha_{n-1}$

$d^{(m)} = y^{(m)} + \frac{{\vartheta_{m-1}}^2 \, \eta_{m-1}}{\alpha_{n-1}} \, d^{(m-1)}$

$x^{(m)} = x^{(m-1)} + \eta_m d^{(m)}$

if $x^{(m)}$ *konvergiert* **then** *beende Iterationsverfahren*

end for

$\rho_n = \langle \tilde{r}^{(0)}, w^{(2n+1)} \rangle$

$\beta_n = \rho_n / \rho_{n-1}$

$y^{(2n+1)} = w^{(2n+1)} + \beta_n y^{(2n)}$

$v^{(n)} = \boldsymbol{A} y^{(2n+1)} + \beta_n \left(\boldsymbol{A} y^{(2n)} + \beta_n v^{(n-1)} \right)$

end for

end /* *TFQMR* */

Abbildung 2.3: TFQMR Algorithmus

2.4 Methode der Quasi–Minimalen Residuen

Zur Lösung eines Systems $\boldsymbol{A}x = b$ mit regulärer Koeffizientenmatrix $\boldsymbol{A}$ ist im vorigen Abschnitt die transpositionsfreie Methode der Quasi–Minimalen Residuen (TFQMR) beschrieben worden. TFQMR benötigt in jedem Iterationsschritt zwei Matrix–Vektor–Produkte mit der Koeffizientenmatrix $\boldsymbol{A}$. In diesem Abschnitt wird ein Verfahren zur Lösung der gleichen Problemstellung vorgestellt, das zwei unterschiedliche Matrix–Vektor–Produkte enthält. Die Methode der Quasi–Minimalen Residuen, QMR (*Quasi–Minimal Residual*), die nun beschrieben wird, enthält sowohl ein Matrix–Vektor–Produkt mit der Koeffizientenmatrix $\boldsymbol{A}$ als auch mit deren Transponierter $\boldsymbol{A}^T$. QMR und TFQMR sind mathematisch nicht äquivalent; insbesondere unterscheiden sich die von den beiden Verfahren erzeugten Iterationsvektoren.

Die Methode QMR [25, 26] ist ein iteratives Verfahren auf Polynombasis. Wie in Abschnitt 2.2 erwähnt lassen sich solche Methoden durch das Aufspannen von Krylov–Teilräumen beschreiben. Jeder Iterationsvektor $x^{(n)}$ eines solchen Verfahrens läßt sich in der Form

$$x^{(n)} \in x^{(0)} + \mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right) \tag{2.44}$$

darstellen, wobei $x^{(0)}$ einen Startvektor und $r^{(0)}$ das zugehörige Startresiduum bezeichnet. Die zwei Hauptbestandteile eines Verfahrens auf Polynombasis sind die Konstruktion einer Basis für die Krylov–Teilräume $\mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right)$ und die Definition der aktuellen Iterierten. Bei dem im vorigen Abschnitt vorgestellten Verfahren TFQMR wird die Basis für die Krylov–Teilräume durch das Verfahren CGS geliefert. Dagegen wird bei QMR der unsymmetrische Lanczos–Algorithmus mit look–ahead [24] zum Aufspannen der Teilräume verwendet. Die Definition der aktuellen Iterierten von QMR erfolgt in Analogie zu TFQMR, so daß in diesem Abschnitt auf eine Herleitung von QMR im Detail verzichtet werden kann. Stattdessen werden an dieser Stelle die wesentlichen Punkte des QMR–Ansatzes skizziert.

Der klassische unsymmetrische Lanczos–Algorithmus [32] erzeugt zu zwei Startvektoren je eine Folge von Vektoren. Die so erzeugten zwei Folgen von Vektoren bilden jeweils eine Basis von unterschiedlichen Krylov–Teilräumen. Von diesen beiden erzeugten Folgen von Vektoren wird im folgenden nur eine näher betrachtet. Der Lanczos–Algorithmus benötigt einen so geringen Operationsaufwand und Speicherbedarf, daß er als Grundbaustein einer iterativen Methode zur Lösung von Gleichungssystemen verwendet werden kann. Zur Konstruktion von QMR verwendet man das Startresiduum $r^{(0)}$ als einen Startvektor $v^{(1)}$ des Lanczos–Algorithmus. Mit der Bedingung

$$v^{(1)} = r^{(0)} \neq 0 \tag{2.45}$$

generiert der Lanczos–Algorithmus Vektoren $v^{(1)}, v^{(2)}, \ldots, v^{(n)}$. Diese Vektoren werden Lanczos–Vektoren genannt und werden als Spaltenvektoren in der $N \times n$ Matrix

$$\boldsymbol{V}^{(n)} := \left[v^{(1)} v^{(2)} \ldots v^{(n)}\right]$$

zusammengefaßt. Die Lanczos–Vektoren spannen den Krylov–Teilraum

$$\mathcal{K}_n\left(r^{(0)}, \boldsymbol{A}\right) = \text{span}\left\{v^{(1)}, v^{(2)}, \ldots, v^{(n)}\right\} = \left\{\boldsymbol{V}^{(n)} z \;\middle|\; z \in \mathbb{R}^n\right\} \tag{2.46}$$

auf. Neben der Matrix $\boldsymbol{V}^{(n)}$ erzeugt der Lanczos–Algorithmus eine obere $(n+1) \times n$ Hessenberg–Matrix $\boldsymbol{H}^{(n)}$. In [26] ist für die Koeffizientenmatrix, die Lanczos–Vektoren und die Hessenberg–Matrix die Beziehung

$$\boldsymbol{A}\boldsymbol{V}^{(n)} = \boldsymbol{V}^{(n+1)}\boldsymbol{H}^{(n)} \tag{2.47}$$

hergeleitet. Wegen (2.44) und (2.46) lassen sich die QMR–Iterierten in der Form

$$x^{(n)} = x^{(0)} + \boldsymbol{V}^{(n)} z \quad \text{für ein} \quad z \in \mathbb{R}^n$$

darstellen. Daher gilt unter Verwendung der Gleichungen (2.45) und (2.47) für das n–te Residuum

$$\begin{aligned} r^{(n)} &= b - \boldsymbol{A}x^{(n)} \\ &= b - \boldsymbol{A}\left(x^{(0)} + \boldsymbol{V}^{(n)} z\right) \\ &= r^{(0)} - \boldsymbol{A}\boldsymbol{V}^{(n)} z \\ &= v^{(1)} - \boldsymbol{V}^{(n+1)}\boldsymbol{H}^{(n)} z \quad \text{für ein} \quad z \in \mathbb{R}^n. \end{aligned}$$

Mit dem Einheitsvektor $e^{(n+1)} := [1, 0, \ldots, 0]^T \in \mathbb{R}^{n+1}$ ist das Residuum auch in der Form

$$r^{(n)} = \boldsymbol{V}^{(n+1)}\left(e^{(n+1)} - \boldsymbol{H}^{(n)} z\right) \quad \text{für ein} \quad z \in \mathbb{R}^n \tag{2.48}$$

darstellbar. Ziel bei der Definition eines Iterationsverfahrens ist die Minimierung der Norm des Residuums in jedem Iterationsschritt. Die Bestimmung des Parametervektors z in Gleichung (2.48) so, daß $\|r^{(n)}\|$ minimal wird, stellt ein lineares Ausgleichsproblem dar, das durch eine QR–Zerlegung der $N \times n$ Matrix $\boldsymbol{V}^{(n+1)}\boldsymbol{H}^{(n)}$ gelöst werden kann. Der dazu in jedem Iterationsschritt notwendige Berechnungsaufwand von $\mathcal{O}(Nn^2)$ wäre allerdings zu hoch. Stattdessen wird die Minimierung nur auf den in Gleichung (2.48) geklammerten Ausdruck bezogen. Das entsprechende lineare Ausgleichsproblem

$$\left\|e^{(n+1)} - \boldsymbol{H}^{(n)} z^{(n)}\right\| = \min_{z \in \mathbb{R}^n} \left\|e^{(n+1)} - \boldsymbol{H}^{(n)} z\right\| \tag{2.49}$$

kann nun durch eine QR–Zerlegung der $(n+1) \times n$ Matrix $\boldsymbol{H}^{(n)}$ mit dem geringeren Aufwand von $\mathcal{O}(n^3)$ berechnet werden. Die Lösung des Minimierungsproblems (2.49) liefert den Vektor $z^{(n)} \in \mathbb{R}^n$, mit dem die QMR–Iterierten

$$x^{(n)} = x^{(0)} + \boldsymbol{V}^{(n)} z^{(n)} \tag{2.50}$$

definiert werden. Der entscheidende Punkt beim QMR–Ansatz ist, daß der Lanczos–Algorithmus die Matrix $\boldsymbol{V}^{(n+1)}$ mit nur wenig Aufwand liefert und der dadurch gewonnene Vorteil nicht durch das berechnungsintensive Minimieren von $\|r^{(n)}\|$ verlorengeht. (Führt man die aufwendige Minimierung von $\|r^{(n)}\|$ durch, erhält man ein zu GMRES [51] mathematisch äquivalentes Verfahren.)

In jedem Iterationsschritt von QMR wird das Minimierungsproblem (2.49) gelöst. Dazu wird eine QR–Zerlegung der Hessenberg–Matrix $\boldsymbol{H}^{(n)}$ vorgenommen, die auf Givens–Rotationen [32] basiert. Die QR–Zerlegung liefert eine Faktorisierung der Matrix $\boldsymbol{H}^{(n)}$ in eine orthogonale $(n+1) \times (n+1)$ Matrix $\boldsymbol{Q}^{(n)}$ und eine obere $n \times n$ Dreiecksmatrix $\boldsymbol{R}^{(n)}$, für die die Beziehung

$$\boldsymbol{H}^{(n)} = \boldsymbol{Q}^{(n)^T} \begin{bmatrix} \boldsymbol{R}^{(n)} \\ 0 \cdots 0 \end{bmatrix}$$

gilt. Um den Lösungsvektor $z^{(n)}$ des Minimierungsproblems (2.49) zu bestimmen, wird diese Darstellung der Matrix $\boldsymbol{H}^{(n)}$ zusammen mit der Orthogonalität der Matrix $\boldsymbol{Q}^{(n)}$, nämlich

$$\boldsymbol{Q}^{(n)^T} \boldsymbol{Q}^{(n)} = \boldsymbol{E}^{(n+1)},$$

wobei $\boldsymbol{E}^{(n+1)} \in \mathbb{R}^{(n+1)\times(n+1)}$ die Einheitsmatrix bezeichnet, in die Gleichung (2.49) eingesetzt. Es folgt die äquivalente Formulierung des Minimierungsproblems

$$\begin{aligned}
\left\| e^{(n+1)} - \boldsymbol{H}^{(n)} z^{(n)} \right\| &= \min_{z \in \mathbb{R}^n} \left\| e^{(n+1)} - \boldsymbol{Q}^{(n)^T} \begin{bmatrix} \boldsymbol{R}^{(n)} \\ 0 \cdots 0 \end{bmatrix} z \right\| \\
&= \min_{z \in \mathbb{R}^n} \left\| \boldsymbol{Q}^{(n)^T} \left(\boldsymbol{Q}^{(n)} e^{(n+1)} - \begin{bmatrix} \boldsymbol{R}^{(n)} \\ 0 \cdots 0 \end{bmatrix} z \right) \right\| \\
&= \min_{z \in \mathbb{R}^n} \left\| \boldsymbol{Q}^{(n)} e^{(n+1)} - \begin{bmatrix} \boldsymbol{R}^{(n)} \\ 0 \cdots 0 \end{bmatrix} z \right\| \\
&= \min_{z \in \mathbb{R}^n} \left\| \begin{bmatrix} t^{(n)} - \boldsymbol{R}^{(n)} z \\ \tau_{n+1} \end{bmatrix} \right\|,
\end{aligned}$$

wobei der Vektor $\boldsymbol{Q}^{(n)} e^{(n+1)} \in \mathbb{R}^{n+1}$ in seine ersten n Komponenten $t^{(n)}$ und seine letzte Komponente τ_{n+1} aufgespalten worden ist. Da die Matrix $\boldsymbol{R}^{(n)}$ invertierbar ist [47], ist die Lösung des linearen Ausgleichsproblems (2.49) eindeutig durch

$$z^{(n)} = \boldsymbol{R}^{(n)^{-1}} t^{(n)}$$

gegeben. Die eindeutigen QMR–Iterierten sind daher nach (2.50) in der Form

$$x^{(n)} = x^{(0)} + \boldsymbol{V}^{(n)} \boldsymbol{R}^{(n)^{-1}} t^{(n)} \tag{2.51}$$

darstellbar. Demnach kann man die grundlegenden Bestandteile von QMR wie folgt zusammenfassen. In jedem Iterationsschritt von QMR werden zunächst die Matrizen $\boldsymbol{V}^{(n+1)}$ und $\boldsymbol{H}^{(n)}$ durch den Lanczos–Algorithmus erzeugt. Dann wird eine QR–Zerlegung der Matrix $\boldsymbol{H}^{(n)}$ durchgeführt, und schließlich werden die aktuellen

Subroutine *Initialisiere (QMR)*
$W\ddot{a}hle \quad x^{(0)} \in \mathbb{R}^N$
$r^{(0)} = b - \boldsymbol{A}x^{(0)}$
$\tilde{v}^{(1)} = r^{(0)}$
$L\ddot{o}se\ \boldsymbol{M}_1\, y = \tilde{v}^{(1)}$
$\rho_1 = \|y\|$
$W\ddot{a}hle \quad \tilde{w}^{(1)},\ z.B. \quad \tilde{w}^{(1)} = r^{(0)}$
$L\ddot{o}se\ \boldsymbol{M}_2^T\, z = \tilde{w}^{(1)}$
$\xi_1 = \|z\|$
$\gamma_0 = 1$
$\eta_0 = -1$
end /* *Initialisiere* */

Abbildung 2.4: QMR Initialisierungen

Iterierten nach Gleichung (2.51) berechnet. In [25, 26, 47] sind Details des Verfahrens QMR beschrieben. Tatsächlich ist der QMR–Ansatz dort etwas allgemeiner formuliert und zwar mit einem zusätzlichen Parametersatz in Analogie zu den Gleichungen (2.16) – (2.19), die einen Parametersatz für das in dieser Arbeit vorgestellte Verfahren TFQMR enthalten.

In den Abbildungen 2.4, 2.5 und 2.6 ist eine QMR–Version aus [2] mit Vorkonditionierung dargestellt. Diese Version entspricht in leicht veränderter Form dem Algorithmus 7.1 aus der Arbeit von Freund und Nachtigal [30]. Diese Darstellung wurde gewählt, da QMR in numerischen Beispielen üblicherweise mit Vorkonditionierung benutzt wird [26, 28, 29, 47]. Unter Vorkonditionierung (*preconditioning*) versteht man die Transformation eines Gleichungssystems in ein äquivalentes System in dem Sinne, daß das transformierte System dieselbe Lösung wie das ursprüngliche System besitzt, jedoch ein günstigeres Konvergenzverhalten aufweist. Falls das Originalsystem $\boldsymbol{A}x = b$ lautet und die Vorkonditionierung mit Hilfe einer Matrix $\boldsymbol{M} = \boldsymbol{M}_1\boldsymbol{M}_2$ durchgeführt wird, so besitzt das vorkonditionierte System

$$\boldsymbol{M}_1^{-1}\boldsymbol{A}\boldsymbol{M}_2^{-1}\,(\boldsymbol{M}_2\,x) = \boldsymbol{M}_1^{-1}b$$

dieselbe Lösung wie das Originalsystem. Der Algorithmus QMR, wie er in den genannten Abbildungen formuliert ist, enthält die Vorkonditionierung ausschließlich in Operationen der Form

$$L\ddot{o}se\ \boldsymbol{M}_1^T\,\tilde{z} = z \qquad \text{und}$$
$$L\ddot{o}se\ \boldsymbol{M}_2\,\tilde{y} = y.$$

Bei der Wahl der Matrix $\boldsymbol{M}$ ist daher darauf zu achten, daß diese Gleichungssysteme einfach, d.h. mit nur geringem Operationsaufwand, zu lösen sind. Eine ausführli-

che Beschreibung der Techniken zur Vorkonditionierung von Gleichungssystemen bieten [32, 49].

In Abbildung 2.4 sind die Initialisierungen von QMR notiert. QMR berechnet in jeder Iteration zwei Matrix–Vektor–Produkte. Die Abbildung 2.5 stellt denjenigen Teil von QMR dar, in dem diese beiden Matrix–Vektor–Produkte gebildet werden. Der restliche Algorithmus, der in Abbildung 2.6 formuliert ist, enthält keine Matrix–Vektor–Produkte. Daher genügt bei Betrachtungen, die sich ausschließlich auf die beiden Matrix–Vektor–Produkte beziehen, die Kenntnis der Abbildung 2.5. Die Abbildung zeigt die beiden Matrix–Vektor–Produkte $\boldsymbol{A}p^{(n)}$ und $\boldsymbol{A}^T q^{(n)}$. Da die Vektoren $p^{(n)}$ und $q^{(n)}$ lediglich als Eingabe für die Prozedur *Matrix_Vektor_Produkte* dienen und im Verlaufe derselben nicht mehr verändert werden, ist offensichtlich, daß beide Matrix–Vektor–Produkte unabhängig voneinander ausgeführt werden können.

Abschließend sei darauf hingewiesen, daß Abschätzungen für die Norm des n–ten QMR–Residuums existieren. In [26] ist erstmals ein solches Ergebnis über das Konvergenzverhalten einer Lanczos–ähnlichen Methode zur Lösung von unsymmetrischen linearen Gleichungssystemen enthalten. Dieses Resultat ist später allgemeiner formuliert worden [18].

Subroutine *Matrix_Vektor_Produkte (QMR)*

$\tilde{p} = \boldsymbol{A}p^{(n)}$

$\varepsilon_n = \langle q^{(n)}, \tilde{p} \rangle$

$\beta_n = \varepsilon_n / \delta_n$

$\tilde{v}^{(n+1)} = \tilde{p} - \beta_n v^{(n)}$

Löse $\boldsymbol{M}_1 \, y = \tilde{v}^{(n+1)}$

$\rho_{n+1} = \|y\|$

$\tilde{w}^{(n+1)} = \boldsymbol{A}^T q^{(n)} - \beta_n w^{(n)}$

Löse $\boldsymbol{M}_2^T \, z = \tilde{w}^{(n+1)}$

$\xi_{n+1} = \|z\|$

$\theta_n = \rho_{n+1} / \left(\gamma_{n-1} |\beta_n|\right)$

$\gamma_n = 1 \Big/ \sqrt{1 + {\theta_n}^2}$

$\eta_n = -\eta_{n-1} \rho_n {\gamma_n}^2 / \left(\beta_n {\gamma_{n-1}}^2\right)$

end /* *Matrix_Vektor_Produkte* */

Abbildung 2.5: Berechnungen QMR

Algorithmus *QMR*

/* Der Algorithmus löst ein lineares Gleichungssystem $\boldsymbol{A}x = b$, */
/* mit $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ und $b \in \mathbb{R}^N$. Die exakte Lösung x^* wird */
/* durch die Iterationsvektoren $x^{(n)}$ approximiert. */

Initialisiere(QMR)
for $n = 1, 2, 3, \ldots$ **do**
$v^{(n)} = \tilde{v}^{(n)} / \rho_n$
$y = y / \rho_n$
$w^{(n)} = \tilde{w}^{(n)} / \xi_n$
$z = z / \xi_n$
$\delta_n = \langle z, y \rangle$
Löse $\boldsymbol{M}_2 \, \tilde{y} = y$
Löse $\boldsymbol{M}_1^T \, \tilde{z} = z$
if $n = 1$ **then**
$p^{(1)} = \tilde{y}$
$q^{(1)} = \tilde{z}$
else
$p^{(n)} = \tilde{y} - (\xi_n \delta_n / \varepsilon_{n-1}) \, p^{(n-1)}$
$q^{(n)} = \tilde{z} - (\rho_n \delta_n / \varepsilon_{n-1}) \, q^{(n-1)}$
end if
Matrix_Vektor_Produkte(QMR)
if $n = 1$ **then**
$d^{(1)} = \eta_1 \, p^{(1)}$
$s^{(1)} = \eta_1 \, \tilde{p}$
else
$d^{(n)} = \eta_n \, p^{(n)} + (\theta_{n-1} \, \gamma_n)^2 \, d^{(n-1)}$
$s^{(n)} = \eta_n \, \tilde{p} + (\theta_{n-1} \, \gamma_n)^2 \, s^{(n-1)}$
end if
$x^{(n)} = x^{(n-1)} + d^{(n)}$
$r^{(n)} = r^{(n-1)} - s^{(n)}$
if $x^{(n)}$ *konvergiert* **then** *beende Iterationsverfahren*
end for
end /* *QMR* */

Abbildung 2.6: QMR Algorithmus

2.5 Andere Iterationsverfahren

In diesem Abschnitt werden zuerst Iterationsverfahren aufgezählt, die einen ähnlichen Ansatz wie die Verfahren TFQMR und QMR besitzen. Anschließend wird ein kurzer Überblick über weitere Verfahren zur iterativen Lösung von unsymmetrischen Gleichungssystemen gegeben.

2.5.1 Iterative Methoden mit QMR–Ansatz

In diesem Abschnitt wird der Bezug zu anderen Verfahren zur Lösung von linearen Gleichungssystemen hergestellt, die den QMR–Ansatz verfolgen. Unter dem Begriff *QMR–Ansatz* wird in diesem Zusammenhang die Definition einer Iterierten aufgrund der Minimierung nur eines Faktors des Residuums — im Gegensatz zu einer Minimierung der Norm des vollständigen Residuums — verstanden. Dabei werden sowohl Verfahren, die ein Matrix–Vektor–Produkt mit der Transponierten der Koeffizientenmatrix enthalten, als auch transpositionsfreie Methoden berücksichtigt.

Der QMR–Ansatz wurde zunächst für lineare Systeme mit symmetrischer Koeffizientenmatrix entwickelt [17]. Solche Systeme können etwa bei Problemen des Elektromagnetismus auftreten, in denen Symmetrie zwischen den zu modellierenden Beziehungen zweier Punkte herrscht. Später ist der QMR–Ansatz auf den unsymmetrischen Fall [26], so wie er in dieser Arbeit enthalten ist, erweitert worden. Motiviert durch die Tatsache, daß ein Matrix–Vektor–Produkt mit der Transponierten der Koeffizientenmatrix unter Umständen nur sehr aufwendig berechnet werden kann, sind die Ideen der Minimierung eines Faktors des Residuums auf transpositionsfreie Methoden übertragen worden, deren bekanntester Vertreter, TFQMR [19], in dieser Arbeit vorgestellt ist. Eine andere transpositionsfreie QMR–Variante [10] benutzt den klassischen unsymmetrischen Lanczos–Algorithmus ohne look–ahead [32] zum Aufspannen der Krylov–Teilräume. In [31] wird eine transpositionsfreie QMR–Variante durch Quadrieren der Residuum–Polynome aus QMR hergeleitet. Im Gegensatz zu den Methoden, die in dieser Arbeit vorgestellt sind und die in jeder Iteration jeweils zwei Matrix–Vektor–Produkte enthalten, benötigen die Methoden aus [10] und [31] in jeder Iteration jeweils drei Matrix–Vektor–Produkte. Daher ist bei diesen Verfahren der benötigte Rechenaufwand höher. Erst kürzlich sind zwei weitere Algorithmen vorgestellt worden, die auf dem QMR–Ansatz beruhen. Das Lösen von linearen Gleichungssystemen mit mehreren rechten Seiten [20] gehört ebenso dazu wie eine Version des TFQMR mit look–ahead [48].

2.5.2 Weitere iterative Methoden

Das Entwickeln von iterativen Methoden zur Lösung von unsymmetrischen linearen Gleichungssystemen, die auf der Theorie der Krylov–Teilräume basieren, ist derzeit ein aktives Forschungsgebiet. Einen Überblick über die Methoden bis zum

Jahre 1991 ist in [21, 23] enthalten. Neuere Arbeiten findet man vorwiegend in Konferenzbänden, die sich mit diesem Thema befassen [6, 46]. In diesem Abschnitt werden häufig verwendete Verfahren skizziert, die in [2] näher beschrieben sind. Dabei werden insbesondere die Eigenschaften der genannten Verfahren in bezug auf die pro Iterationsschritt benötigten Matrix–Vektor–Produkte hervorgehoben. Neben der Anzahl der Matrix–Vektor–Produkte pro Iteration wird beschrieben, ob die Methoden ein Matrix–Vektor–Produkt mit der Transponierten der Koeffizientenmatrix enthalten oder ob sie transpositionsfrei sind. Treten pro Iteration mehrere Matrix–Vektor–Produkte auf, so ist von Interesse, welche Matrix–Vektor–Produkte unabhängig voneinander berechnet werden können. Geht nämlich das Ergebnis eines Matrix–Vektor–Produktes nicht in die Berechnung eines zweiten ein, so können diese beiden Matrix–Vektor–Produkte simultan berechnet werden. Dieser Punkt erhält bei der Implementierung auf parallelen Rechnern Bedeutung.

Die Methode GMRES (*Generalized Minimal Residual*) [51] basiert auf dem Arnoldi–Prozeß [32]. Der Arnoldi–Prozeß erzeugt mit dem Gram–Schmidt–Verfahren eine orthonormale Basis der Krylov–Teilräume, in denen die GMRES–Iterierten über die Minimierung der Norm des Residuums definiert werden. Ein Problem von GMRES liegt darin, daß die Anzahl der pro Iteration benötigten Operationen nicht konstant ist, sondern linear mit der Iterationsanzahl zunimmt. Pro Iteration wird zwar nur ein Matrix–Vektor–Produkt benötigt, jedoch werden in der n–ten Iteration $n + 1$ Skalarprodukte berechnet.

Eine Erweiterung des bekannten Verfahrens CG (*Conjugate Gradient*) [35], das nur für symmetrische positiv definite Systeme anwendbar ist, führt auf die Methode BCG (*Biconjugate Gradient*) [43], mit der auch unsymmetrische Systeme gelöst werden können. BCG erzeugt zwei Folgen von Iterierten. Eine Folge basiert auf Matrix–Vektor–Produkten mit der Koeffizientenmatrix, die andere auf Matrix–Vektor–Produkten mit der Transponierten der Koeffizientenmatrix. Pro Iterationsschritt sind zwei Matrix–Vektor–Produkte enthalten, die unabhängig voneinander berechnet werden können.

Durch Quadrieren der Residuum–Polynome von BCG kann das Verfahren CGS (*Conjugate Gradient Squared*) [54] hergeleitet werden, das wie BCG zwei Matrix–Vektor–Produkte pro Iteration benötigt. CGS enthält jedoch kein Matrix–Vektor–Produkt mit der Transponierten der Koeffizientenmatrix. Pro Iteration werden zwei Matrix–Vektor–Produkte ausgeführt, wobei das Ergebnis des ersten Matrix–Vektor–Produktes für die Berechnung des zweiten benötigt wird.

Das Verfahren BiCGSTAB (*Biconjugate Gradient Stabilized*) [55] basiert auf dem unsymmetrischen Lanczos–Algorithmus. BiCGSTAB kann als eine Variante von CGS bzw. BCG aufgefaßt werden, die ein günstiges Fehlerverhalten besitzt. Wie CGS ist BiCGSTAB ein transpositionsfreies Verfahren, dessen zwei Matrix–Vektor–Produkte pro Iteration nicht unabhängig voneinander berechnet werden können.

Jede der genannten Methoden besitzt Vor– und Nachteile. Eine iterative Methode, die in bezug auf die Gesamtheit aller Gleichungssysteme gleichermaßen von Vorteil ist, existiert nicht. Die Auswahl einer geeigneten Methode kann immer nur für

Methode	Operation		
	Mat_Vek_Pro	Abh./Unabh.	Ska_Pro
GMRES	1	—	$n+1$
BCG	$1 \oplus 1$	unabhängig	2
CGS	2	abhängig	2
BiCGSTAB	2	abhängig	4
TFQMR	2	abhängig	4
QMR	$1 \oplus 1$	unabhängig	2

Tabelle 2.1: Anzahl der benötigten Operationen im n–ten Iterationsschritt. Die Notation „$i \oplus j$“ bedeutet i Matrix–Vektor–Produkte mit der Koeffizientenmatrix und j Matrix–Vektor–Produkte mit der Transponierten der Koeffizientenmatrix

eine bestimmte Problemklasse erfolgen. Aus Sicht der Implementierung eines Iterationsverfahrens hängt die Wahl einer geeigneten Methode von vielen Faktoren ab, etwa dem zur Verfügung stehenden Speicherplatz, der Verfügbarkeit der Transponierten der Koeffizientenmatrix oder wie aufwendig ein Matrix–Vektor–Produkt im Vergleich zu einem Skalarprodukt ist. In der Tabelle 2.1 sind die Eigenschaften der skizzierten Verfahren zusammengefaßt. Zum Vergleich sind in die Tabelle die entsprechenden Eigenschaften der Methoden TFQMR und QMR aufgenommen. In der Spalte, die mit *Mat_Vek_Pro* gekennzeichnet ist, ist die Anzahl der Matrix–Vektor–Produkte pro Iterationsschritt enthalten. In der rechts folgenden Spalte ist vermerkt, ob diese Matrix–Vektor–Produkte voneinander abhängig sind oder nicht. Die letzte Spalte führt die Anzahl der pro Iterationsschritt benötigten Skalarprodukte auf.

Kapitel 3

Vorbereitungen zur Parallelisierung

Dieses Kapitel enthält eine Reihe von Vorbemerkungen, die für das Verständnis der Arbeit notwendig erscheinen. Da die Arbeit die Parallelisierung der QMR–Methode untersucht, beginnt das Kapitel mit einem kurzen Abschnitt zur Parallelverarbeitung und einer Beschreibung des bei der Implementierung verwendeten Rechners. Daran anschließend werden einige technische Details erläutert, die beispielsweise das für die Zeitmessungen benutzte Meßverfahren betreffen. Weiterhin werden in diesem Kapitel die implementierten Datenstrukturen beschrieben. Es wird gezeigt, nach welchen Kriterien die Verteilung der Daten auf dem verwendeten Parallelrechner vorgenommen wird. Zur parallelen Berechnung von Matrix–Vektor–Produkten ist ein Kommunikationsschema notwendig, das festlegt, wie die einzelnen Prozessoren des Parallelrechners miteinander Daten austauschen. Im Anschluß an einen Hinweis bezüglich des Kommunikationsschemas werden die verwendeten Abbruchkriterien der iterativen Methoden aufgeführt. Am Ende des Kapitels werden die Koeffizientenmatrizen derjenigen linearen Gleichungssysteme beschrieben, mit denen numerische Untersuchungen durchgeführt werden.

3.1 Parallelverarbeitung

In knapper Form werden in diesem Abschnitt nur diejenigen Aspekte angesprochen, die für das Verständnis dieser Arbeit aus Sicht der Parallelverarbeitung notwendig erscheinen. Eine umfassende Einführung in Konzepte der Parallelverarbeitung kann an dieser Stelle nicht gegeben werden. Stattdessen werden nur einige spezielle Begriffe der Parallelverarbeitung beschrieben sowie Parallelrechner in Systeme mit gemeinsamem Speicher und solche mit verteiltem Speicher klassifiziert.

3.1.1 Spezielle Begriffe der Parallelverarbeitung

Parallelrechner können vorteilhaft eingesetzt werden, wenn die zu verrichtende Arbeit möglichst gleichmäßig auf alle vorhandenen Prozessoren verteilt wird. Die Parallelverarbeitung muß deshalb auf die Reduzierung derjenigen Zeiten ausgerichtet sein, in denen einzelne Prozessoren keine Berechnung durchführen. Die Aufgabe, Arbeit gleichmäßig zu verteilen, wird auch als Lastausgleich (*Load Balancing*) [39] bezeichnet. Der Lastausgleich ist besonders schwierig, wenn beispielsweise beim Verteilen eines Arbeitsschrittes nicht bekannt ist, wieviel Zeit ein Prozessor für die ihm zugewiesene Arbeit benötigt.

Durch *Synchronisation* von Prozessoren wird die korrekte Reihenfolge von Anweisungen sichergestellt. Viele Synchronisationsprobleme lassen sich abstrakt durch das Problem des gegenseitigen Ausschlusses (*Mutual Exclusion*) darstellen. Eine Aktivität A_1 von Prozessor P_1 und eine Aktivität A_2 von P_2 schließen sich gegenseitig aus, wenn die Ausführung von A_1 nicht die Ausführung von A_2 überschneiden darf. Falls P_1 und P_2 gleichzeitig ihre Aktivität ausführen wollen, muß sichergestellt werden, daß dies nur einem der beiden P_i gelingt. Das Problem des gegenseitigen Ausschlusses kann z.B. durch die von Dijkstra eingeführten Semaphore oder die von Hoare formalisierten Monitore gelöst werden [7].

Arithmetische Operationen werden auf allen Prozessoren ausgeführt. Wird von einem Prozessor das Ergebnis der Berechnung eines anderen Prozessors benötigt, werden die Daten durch *Kommunikation* [36] zwischen Prozessoren bereitgestellt. Durch das explizite Aufbauen von Verbindungen beim Nachrichtenaustausch entstehen Verzögerungszeiten, die sich besonders nachteilig bei kurzen Nachrichten auswirken, da dabei der Aufwand zum Verbindungsaufbau in bezug auf die ausgetauschte Informationsmenge höher ist als bei langen Nachrichten.

Als Maß für die Güte eines parallelen Programms dient der Begriff *Speedup* [15]. Darunter wird der Quotient aus der Zeit, die das schnellste sequentielle Programm zur Lösung eines Problems benötigt, zu der Zeit, die ein paralleles Programm zur Lösung desselben Problems benötigt, verstanden. Die bei der Ausführung eines Programms benutzte Prozessoranzahl bildet eine obere Schranke für den Speedup.

3.1.2 Klassifizierung von Parallelrechnern

Parallelrechner können nach mehreren Kriterien klassifiziert werden. Systeme mit gemeinsamem Speicher können von denen mit verteiltem Speicher unterschieden werden. Weiterreichende Klassifizierungen sind in [36] enthalten.

Systeme mit gemeinsamem Speicher

Systeme mit gemeinsamem Speicher (*Shared Memory*) besitzen einen einzigen physikalischen Speicher, auf den alle Prozessoren zugreifen können. Der physikalische Adreßraum ist global, d.h. jeder Prozessor kann jedes Element des Adreßraumes bearbeiten, indem das Datum an der physikalischen Adresse referenziert wird. Kommunikation zwischen den Prozessoren findet über den gemeinsamen Speicher statt.

Ein Beispiel für einen Rechner mit gemeinsamem Speicher ist der Rechner CRAY X–MP/416. Dieser Mehrprozessorvektorrechner verfügt über vier Prozessoren und einen Hauptspeicher, der aus 16 Megaworten zu je 64 Bit besteht. Jeder der vier Prozessoren besitzt vier Verbindungspfade (*Ports*) zum gemeinsamen Speicher. Drei Ports dienen dem Datentransfer zwischen dem Hauptspeicher und den Registern der Prozessoren, während das vierte Port für die Verbindung zu einem Ein–/Ausgabe–Subsystem und einem Erweiterungsspeicher reserviert ist. Der Erweiterungsspeicher ist ein schneller Zwischenspeicher, der wie ein Plattenspeicher benutzt werden kann oder durch das Betriebssystem zur Speicherung von temporären Daten genutzt wird. Die Architektur der CRAY X–MP wird ausführlicher in [38, 36] beschrieben.

Systeme mit verteiltem Speicher

Im Gegensatz zu Systemen mit gemeinsamem Speicher besitzen Systeme mit verteiltem Speicher (*Distributed Memory*) keinen globalen physikalischen Adreßraum. Bei diesen Systemen sind den Prozessoren Speichereinheiten unmittelbar zugeordnet. Jeder Prozessor kann deshalb nur die Daten bearbeiten, die in seinem eigenen Speicher enthalten sind. Ein Zugriff auf Daten eines anderen Prozessors muß durch explizite Kommunikation eingeleitet werden. Die Kommunikation in Systemen mit verteiltem Speicher unter dem Programmiermodell des *Message Passing*, die in dieser Arbeit betrachtet werden, erfolgt durch den Austausch von Nachrichten. Daten anderer Prozessoren können daher erst dann bearbeitet werden, wenn sie vom entsprechenden Prozessor versendet und beim Adressaten empfangen worden sind. Ähnlich zu den Konzepten der Ein–/Ausgabe findet Message Passing auf einer niedrigen Abstraktionsebene statt und erschwert dadurch die Programmierung. Systeme mit verteiltem Speicher sind daher i.a. aufwendiger zu programmieren als Systeme mit gemeinsamem Speicher.

Die Prozessoren eines Systems mit verteiltem Speicher werden im folgenden als *Knoten* bezeichnet. Knotenleistung und –anzahl kann von System zu System deutlich unterschiedlich sein. Es existieren Syteme mit bis zu einigen tausend Knoten. Solche Systeme werden durch den Begriff *massiv–parallel* charakterisiert. Die Struktur, in der die einzelnen Knoten untereinander verbunden sind, bestimmt die *Topologie* eines Systems mit verteiltem Speicher. Mögliche Topologien sind beispielsweise der Hypercube [44] oder eine Gitter–Struktur, wie sie der im nun folgenden Abschnitt skizzierte Rechner besitzt.

3.2 Der Rechner Intel Paragon XP/S 10

In diesem Abschnitt wird der im Forschungszentrum Jülich installierte Rechner Intel Paragon skizziert, auf dem die Implementierungen der vorliegenden Arbeit durchgeführt worden sind. Einzelheiten des Rechnersystems sind im Handbuch [40] des Herstellers enthalten. In [37] ist neben der Systemarchitektur auch der verwendete Prozessortyp beschrieben.

Der Intel Paragon ist ein massiv–paralleles System mit verteiltem Speicher, der auf der Grundlage seines Vorgängers iPSC/860 entwickelt worden ist, einem Rechner mit Hypercube–Topologie. Ein Problem des iPSC/860 ist seine begrenzte Bandbreite bei der Ein– und Ausgabe. Der Entwurf des Intel Paragon zielt auf eine Behebung dieses Mangels, um so Problemstellungen mit sehr großen Datensätzen aus unterschiedlichsten Anwendungsbereichen lösen zu können.

Der Intel Paragon verfügt in seiner größten Ausbaustufe über nahezu 2000 Rechenknoten, die über ein Verbindungsnetzwerk in Form eines zweidimensionalen Gitters miteinander verbunden sind. Dadurch ist der Benutzer des Intel Paragon nicht mehr auf eine Knotenanzahl festgelegt, die eine Zweierpotenz darstellt, wie es bei dem Hypercube iPSC/860 der Fall ist. Jeder Rechenknoten besitzt zwei Prozessoren vom Typ i860 XP, die im folgenden als *Applikations–* und *Kommunikationsprozessor* bezeichnet werden. Darüberhinaus sind auf jedem Knoten 32 Megabyte lokaler Hauptspeicher verfügbar. Der Applikationsprozessor führt die arithmetischen und logischen Operationen aus, während der Kommunikationsprozessor alleine für den Austausch von Nachrichten verantwortlich ist. Durch die Verlagerung der Nachrichtenübertragung auf einen separaten Prozessor, den Kommunikationsprozessor, wird der Applikationsprozessor von den aufwendigen Protokoll– und Verwaltungsaufgaben der Kommunikation befreit.

Der auf den Knoten verwendete Prozessor vom Typ i860 XP ist ein RISC–Prozessor (*Reduced Instruction Set Computer*) mit mehr als einer Millionen Transistoren auf einem Chip. Der Prozessor erreicht eine theoretisch maximal erreichbare Rechenleistung von 75 MFLOPS in 64–bit–Arithmetik und arbeitet mit einer Taktfrequenz von 50 MHz. Daraus resultiert eine Taktzeit von 20 ns. Die Zugriffszeit auf den lokalen Speicher beträgt 60 ns.

Auf allen Knoten ist das Betriebssystem *Paragon OSF/1* der Open Software Foundation installiert. Dieses Betriebssystem verfügt über den vollen Standard von OSF/1. Es bietet darüberhinaus Möglichkeiten zur Unterstützung der Aufgaben der Parallelverarbeitung wie z.B. den Austausch von Nachrichten zwischen allen Knoten oder den Zugriff auf ein paralleles Dateisystem.

Am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich ist ein Paragon XP/S 10 mit 140 Knoten installiert, der über eine theoretisch maximal erreichbare Rechenleistung von 10,5 GFLOPS in 64–bit–Arithmetik verfügt. In der vorliegenden Arbeit ist das Betriebssystem Paragon OSF/1, Release 1.1, verwendet worden. In dieser Version des Betriebssystems ist der Kommunikationsprozessor

nicht aktiviert, so daß der Applikationsprozessor zusätzlich die gesamte Kommunikation übernimmt. Dieser Sachverhalt wird in der Beurteilung der gemessenen Zeiten in späteren Ausführungen von Bedeutung sein. In [8] sind u.a. Leistungsmessungen enthalten, die ein realistisches Bild der verwendeten Installation geben.

3.3 Technische Vorbemerkungen

Dieser Abschnitt beginnt mit der Beschreibung der Notation, in der die Algorithmen in dieser Arbeit präsentiert werden. Anschließend werden das benutzte Meßverfahren und der verwendete Compiler beschrieben, sowie diejenigen Software–Bibliotheken genannt, die in den Implementierungen dieser Arbeit benutzt werden. Abschließend werden die Kommunikationsarten des Paragon XP/S 10 erläutert, soweit sie in den Implementierungen Anwendung finden.

3.3.1 Pseudocode und schrittweise Verfeinerung

In der vorliegenden Arbeit werden die zu untersuchenden Algorithmen durch Pseudocode beschrieben. Pseudocode besteht aus einer Mischung von Schlüsselwörtern und weniger formal beschriebenen Anweisungen. Um zu einem ausführbaren Programm zu gelangen, müssen die Schlüsselwörter des Pseudocode durch die entsprechenden Schlüsselwörter der gewählten Programmiersprache ersetzt werden. Außerdem müssen die Anweisungen in aufeinanderfolgenden Schritten so lange durch präzisere Anweisungen umgewandelt werden, bis man auf einer Ebene angelangt ist, die dem Sprachumfang der gewählten Programmiersprache entspricht. Diese Vorgehensweise ist als *schrittweise Verfeinerung* [1] bekannt.

Schlüsselwörter sind im Pseudocode durch Fettdruck hervorgehoben, Kommentarzeilen werden durch das Symbol /* eingeleitet und durch das Symbol */ beendet. Einzelne Anweisungen können durch eingeklammerte Ziffern am Zeilenanfang markiert werden, so daß im Text auf sie Bezug genommen werden kann. Deklarationen von Datenstrukturen sind im Pseudocode grundsätzlich nicht enthalten.

3.3.2 Beschreibung des Meßverfahrens

Alle Messungen der parallelen Prozeduren zur Lösung von linearen Gleichungssystemen werden mit demselben Rahmenprogramm durchgeführt. Den Pseudocode dieses Programms zeigt Abbildung 3.1. Zunächst übernimmt ein einzelner Knoten die Eingabe. Dieser Knoten liest aus einer Datei einen Satz von Eingabeparametern, die im folgenden kurz erläutert werden, und sendet die gelesenen Daten unmittelbar an alle anderen Knoten weiter. Die Eingabeparameter betreffen die Wahl des Abbruchkriteriums der iterativen Methode, die vorzunehmende Datenverteilung und die Wahl des Gleichungssystems.

```
program Meßverfahren

/*  Mißt die Zeit einer Prozedur, die ein lineares Gleichungs-  */
/*  systems Ax = b mit einer iterativen Methode löst.           */

begin
    Lies Eingabeparameter ein ;
    Lies Koeffizientenmatrix A und rechte Seite b ein ;
    Erstelle Kommunikationsschema für paralleles Matrix–Vektor–Produkt ;
    Starte Zeitmessung ;
    Rufe zu messende Prozedur auf ;
    Beende Zeitmessung ;
    Bilde Maximum der Zeitmessungen aller Knoten ;
    Postprocessing ;
end  /* Meßverfahren */
```

Abbildung 3.1: Pseudocode des Rahmenprogramms zur Zeitmessung

Ein Abbruchkriterium kann aus mehreren vorgegebenen ausgewählt werden. In der Eingabedatei wird festgelegt, welches der möglichen Kriterien verwendet wird. Zusätzlich wird die zu erreichende Genauigkeit vorgegeben. Ein weiterer Parameter bezieht sich auf die Datenverteilung, die in Abschnitt 3.5 beschrieben wird. Der genannte Parameter regelt, wie die Daten auf die Knoten verteilt werden. Schließlich wird durch einen separaten Parameter die Wahl des zu lösenden Gleichungssystems gesteuert.

Nachdem sämtliche Eingabeparameter bei allen Knoten bekannt sind, beginnt das Einlesen des ausgewählten Gleichungssystems. Es werden die Koeffizientenmatrix $\boldsymbol{A}$ des linearen Systems, deren Ordnung, die rechte Seite b sowie die maximale Anzahl der Nichtnull–Elemente innerhalb einer Zeile von $\boldsymbol{A}$ eingelesen. Damit sind alle notwendigen Voraussetzungen geschaffen, um bei gegebener Datenverteilung aus diesen Daten ein Schema zu entwickeln, mit dem parallele Matrix–Vektor–Produkte ausgeführt werden können. Die Ermittelung eines Kommunikationsschemas ist notwendig, da in der Implementierung sowohl die Zeilen der Koeffizientenmatrix $\boldsymbol{A}$ als auch die Komponenten aller Vektoren auf unterschiedliche Knoten verteilt sind. Dieses Kommunikationsschema wird in [3] entwickelt.

Die Zeitmessung umfaßt ausschließlich die Prozedur, die das Gleichungssystem mit einer iterativen Methode löst. Der Zeitaufwand für das Einlesen der Eingabeparameter und der Daten, die das Gleichungssystem repräsentieren, wird genauso wenig gemessen wie die Erstellung des Kommunikationsschemas. Diese Vorgehensweise ist durch die Tatsache motiviert, daß die Ergebnisse aus dem Kommunikationsschema zwar bei jedem Matrix–Vektor–Produkt verwendet werden, das Schema selbst aber nur einmal erstellt zu werden braucht und der Erstellungsaufwand nur einen ver-

nachlässigbaren Anteil an der Gesamtausführungszeit des Programms besitzt. Als Zeit, die die parallele Ausführung benötigt, wird das Maximum der Zeiten, die auf den einzelnen Knoten gemessen worden sind, angegeben.

Die Nachbearbeitung besteht überwiegend in der Aufbereitung der numerischen Ergebnisse und gemessenen Zeiten.

Alle Messungen sind auf dem Rechner Paragon XP/S 10 des Forschungszentrums Jülich durchgeführt worden. Um den Produktionsbetrieb des Rechnersystems nicht zu unterbrechen, sind alle Zeiten auf einer nicht dedizierten Maschine im Mehrbenutzerbetrieb gemessen worden. Daher kann die Verteilung der Betriebsmittel, die durch das Betriebssystem vorgenommen wird, Einfluß auf die gemessenen Zeiten haben.

3.3.3 Der FORTRAN–Compiler

Für die Untersuchungen der vorliegenden Arbeit stand der FORTRAN–Compiler der Firma The Portland–Group, Inc., Release 4.5, zur Verfügung. Dieser Compiler arbeitet unter dem Betriebssystem Paragon OSF/1. Die Codeerzeugung wird durch Compiler–Optionen gesteuert. Eine detaillierte Beschreibung befindet sich im Handbuch [41] des Compilers. Hier werden nur die Compiler–Optionen *–O<num>*, *–Mvect* und *–Knoieee* beschrieben, dabei ist *<num>* ein Wert zwischen 1 und 4. Durch die Wahl der Compiler–Optionen bzw. Optimierungsstufen werden unterschiedliche Modelle der Codegenerierung festgelegt, deren Hauptmerkmale im folgenden skizziert werden. Jede höhere Optimierungsstufe beinhaltet grundsätzlich die gesamte Leistungsfähigkeit aller niedrigeren Optimierungsstufen.

Bei den Optimierungsstufen *O1* und *O2* wird eine *skalare Optimierung* durchgeführt.

- In der Optimierungsstufe *O1* werden nur solche Veränderungen des Code vorgenommen, die sich auf eine einzelne FORTRAN–Anweisung beziehen. Dazu zählt beispielsweise die Elimination von redundanten Speicherzugriffen.
- Optimierungen, die sich auf mehr als eine FORTRAN–Anweisung beziehen können, werden bei der Optimierungsstufe *O2* vorgenommen. Zum Beispiel werden konstante Teile eines Code aus einer Schleife herausgeholt, sofern die Semantik der Schleife dabei erhalten bleibt.

Bei den bisher beschriebenen Optimierungsstufen werden die auf dem Prozessorchip vorhandenen Pipelines für Fließkomma–Additionen und –Multiplikationen nicht eingesetzt. Sie finden erst bei den Optimierungsstufen *O3* und *O4* Anwendung, in denen *software–pipelining* verwendet wird. In dieser Technik wird sowohl *Pipelining* angewendet als auch der *dual–instruction mode* des i860–Prozessors verwendet. Unter Pipelining versteht man das Akzeptieren von neuer Eingabe in eine Verarbeitungseinheit, während vorherige Eingaben noch bearbeitet werden. Der dual–instruction mode erlaubt eine gleichzeitige Aktivierung der Fließkomma–Einheiten und der Integer–Arithmetik des i860–Prozessors.

- In der Optimierungsstufe *O3* wird zur Umstellung des Code nach dem Prinzip des software–pipelining ein einziger Algorithmus verwendet.
- Dagegen werden in der Optimierungsstufe *O4* mehrere Algorithmen angewendet und deren Resultate miteinander verglichen. Der Code, der zur Übersetzungszeit am günstigsten erscheint, wird ausgewählt.

Bei der Hinzuschaltung der Option *Mvect* zu den Optimierungsstufen *O3* und *O4* wird eine *Vektorisierung* durchgeführt. Welche expliziten Codeoptimierungen bei der Vektorisierung angewendet werden, ist in [41] beschrieben. Hier soll nur auf die Unterschiede zu den beiden Optimierungsstufen *O3*/*O4* hingewiesen werden. Bei den Optimierungsstufen *O3*/*O4* wird Pipelining auf den Code angewendet, ohne diesen dabei umzustellen.

- Im Unterschied dazu versucht *Mvect* den Code so umzustellen, daß ein effizienteres Pipelining möglich ist. Wird also *Mvect* zusammen mit den beiden Optimierungsstufen *O3*/*O4* verwendet, werden mehr Möglichkeiten hinsichtlich Pipelining untersucht als ohne die Option *Mvect*.

Die Option *Knoieee* kann sowohl beim Übersetzen als auch beim Binden verwendet werden. In der vorliegenden Arbeit ist die Option ausschließlich in Verbindung mit beiden genannten Vorgängen untersucht worden.

- Die Option *Knoieee* bewirkt ein Abweichen vom IEEE 754 Standard für Fließkommaoperationen. Zum Beispiel wird für Divisionen ein schnellerer, aber ungenauerer Algorithmus verwendet als derjenige, der ohne die Verwendung der Option angewendet wird.

3.3.4 Verwendete Software–Bibliotheken

In den Implementierungen werden einige Prozeduren aus Software–Bibliotheken verwendet. Aus der *NAG Library Mark 15* werden die Prozeduren `m01dbf` , `m01eaf` und `m01ebf` aufgerufen, aus der *Basic Math Library*, Release 1.3 [42], die BLAS–Routine `idamax`. In den Algorithmen CGS und TFQMR muß jeweils ein Vektor $\tilde{r}^{(0)}$ so gewählt werden, daß er nicht orthogonal zu dem Startresiduum ist, vergleiche Abbildungen 2.1 und 2.2. Diese Wahl wird mit Hilfe der Prozeduren `raseed` und `rangen` aus der *KFALIB*, einer Bibliothek des Forschungszentrums Jülich, durchgeführt. Da so die Komponenten des Vektors $\tilde{r}^{(0)}$ durch einen Zufallszahlengenerator bestimmt werden, ist die Orthogonalität der Vektoren unwahrscheinlich.

3.3.5 Kommunikationsarten des Paragon XP/S 10

Die Knoten des Paragon XP/S 10 kommunizieren untereinander durch Message Passing. Der Benutzer kann zwischen unterschiedlichen Kommunikationsarten wählen, die im Handbuch [40] des Systems aufgeführt sind. Hier werden zwei Arten

vorgestellt, die in den Implementierungen der parallelen Algorithmen benutzt worden sind. Es sind dies die *blockierende Kommunikation* und die *nicht–blockierende Kommunikation*. Beide Kommunikationsarten können nur solche Daten senden bzw. empfangen, die sich in konsekutiven Speicherplätzen befinden. Diese Daten werden in der Parameterliste der Kommunikationsroutinen durch Übergabe einer Startadresse und der Länge der zu übertragenen Nachricht spezifiziert. Eine Nummer, die die Nachricht kennzeichnet, sowie die Angabe des Adressaten vervollständigen die Parameterliste.

Die blockierende Kommunikation wird durch Aufruf des Paares `csend` und `crecv` beim sendenden und empfangenden Knoten durchgeführt. Die Reihenfolge, in der die miteinander kommunizierenden Knoten auf diese Anweisung treffen, ist beliebig. Der Sender setzt die Ausführung seines Programms solange aus, bis die Nachricht an das Netzwerk abgegeben worden ist. Dies garantiert nicht, daß die Nachricht korrekt empfangen worden ist, sondern nur, daß die Nachricht den Sender verlassen hat. Auf der Empfangsseite wird die Programmausführung so lange unterbrochen, bis die Nachricht in dem spezifizierten Puffer eingetroffen ist.

Das Paar `isend` und `irecv` bildet die nicht–blockierende Kommunikation. Trifft ein sendender oder empfangender Knoten auf eine Anweisung einer nicht–blockierenden Kommunikation, wird die Programmausführung nicht ausgesetzt sondern unmittelbar mit der nächsten Anweisung fortgesetzt. Bevor der Inhalt eines Sende– oder Empfangspuffers verändert wird, muß der Programmierer selbst dafür sorgen, daß eine durch `isend` und `irecv` initiierte Nachrichtenübertragung korrekt beendet worden ist. Dazu stehen ihm eine Reihe von Systemaufrufen zur Verfügung, die hier nicht näher betrachtet werden. Der Nachteil einer komplizierteren Kommunikationsstruktur der nicht–blockierenden Kommunikation wird durch eine erhöhte Leistungsfähigkeit ausgeglichen.

In den Implementierungen wird die blockierende Kommunikation für die Verteilung der Eingabeparameter und zur Erstellung des Kommunikationsschemas benutzt. Die nicht–blockierende Kommunikation wird bei der Berechnung der parallelen Matrix–Vektor–Produkte verwendet. Da auf dem verwendeten Rechner derzeit einige Systemaufrufe nicht in ihrer vollen Funktionsfähigkeit zur Verfügung stehen, sind einige Synchronisationspunkte eingefügt worden, die von der Programmlogik her nicht notwendig sind, die benötigte Rechenzeit jedoch verlängern.

3.4 Datenstruktur

Die Wahl einer Datenstruktur hängt unmittelbar von den im Algorithmus auszuführenden Operationen ab. Denn nur mit geeigneten Speichertechniken lassen sich Anweisungen effizient implementieren. Da die Lösung von Gleichungssystemen häufig als Bestandteil einer übergeordneten Problemstellung, wie beispielsweise der numerischen Lösung von Differentialgleichungen, verwendet wird, ist die Datenstruktur oft durch die Anwendung bereits vorgegeben. Sofern die Datenstruktur

MATRIX = **record**
`value` : **array** $[1..t]$ **of** REAL
`col_ind` : **array** $[1..t]$ **of** INTEGER
`row_ptr` : **array** $[1..N+1]$ **of** INTEGER
end record

Abbildung 3.2: Datenstruktur einer $N \times N$ Matrix mit t Nichtnull–Elementen

ausgewählt werden kann, bieten sich für dünnbesetzte Gleichungssysteme Speichertechniken an, in denen nicht die gesamte Koeffizientenmatrix sondern nur deren Nichtnull–Elemente abgespeichert werden. In [58] ist eine Übersicht über derartige Speichertechniken enthalten. Die vorliegende Arbeit soll nicht unterschiedliche Datenstrukturen untersuchen, sondern die Ergebnisse von [4] nutzen, in der die Speichertechnik *Compressed Row Storage* empfohlen wird, die im folgenden mit CRS bezeichnet wird.

In CRS werden nacheinander die Zeilen der Koeffizientenmatrix in aufeinanderfolgenden Speicherplätzen abgelegt. Die Matrix wird dabei in drei eindimensionalen Feldern mit unterschiedlichem Typ und Länge dargestellt. Die Datenstruktur einer $N \times N$ Matrix $\boldsymbol{A}$ mit insgesamt t reellen Nichtnull–Elementen ist in Abbildung 3.2 dargestellt. Ein Feld `value` vom Typ REAL enthält alle t Nichtnull–Elemente der Koeffizientenmatrix. In zwei weiteren Feldern vom Typ INTEGER, `col_ind` und `row_ptr`, ist die Information darüber abgelegt, in welcher Spalte und Zeile sich ein Matrixelement befindet. Ist ein Matrixelement a_{ij}, mit $i, j \in \{1, 2, \ldots, N\}$, an der Position $s \in \{1, 2, \ldots, t\}$ des Feldes `value` gespeichert, also

$$a_{ij} = \texttt{value}(s),$$

dann ist der zugehörige Spaltenindex dieses Matrixelementes in dem Feld `col_ind` an der gleichen Position s abgelegt, und es gilt

$$j = \texttt{col\_ind}(s).$$

Das Feld `row_ptr` der Länge $N+1$ wird benötigt, um den Zeilenindex des Matrixelementes zu bestimmen. Durch das Feld `row_ptr` wird die Segmentierung der eindimensionalen Darstellung der Matrixelemente in einzelne Zeilen beschrieben. Die Position, in der im Feld `value` die i–te Zeile der Koeffizientenmatrix beginnt, ist in $\texttt{row\_ptr}(i)$, mit $i \in \{1, 2, \ldots, N+1\}$, enthalten. Dabei wird per Definition der Zeilenanfang einer virtuellen $(N+1)$–ten Zeile durch $\texttt{row\_ptr}(N+1) = t+1$ bezeichnet. Folglich besteht die i–te Zeile aus $\texttt{row\_ptr}(i+1) - \texttt{row\_ptr}(i)$ Matrixelementen, die in dem Feld `value` an den Positionen $\texttt{row\_ptr}(i), \texttt{row\_ptr}(i)+1, \ldots, \texttt{row\_ptr}(i+1)-1$ gespeichert sind. Für den Zeilenindex des Elementes a_{ij} gilt in dieser Darstellung der Koeffizientenmatrix

$$i = \max_{1 \le r \le N} \{\, r \mid \texttt{row\_ptr}(r) \le s \,\}. \tag{3.1}$$

Als ein Beispiel für diese Datenstruktur sei die Matrix $\boldsymbol{A}$ der Ordnung $N = 5$ mit $t = 13$ Nichtnull–Elementen durch

$$\boldsymbol{A} = \begin{pmatrix} 5 & 2 & 0 & 0 & 0 \\ 1 & 3 & 0 & 6 & 0 \\ 7 & 0 & 4 & 9 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 1 & 8 & 4 & 2 \end{pmatrix}$$

gegeben. Die Abbildung 3.3 zeigt diese Matrix in der beschriebenen Speichertechnik. Dabei stellen die großen Zahlen im Feld `value` die Matrixelemente dar, während die kleinen Zahlen die Feldindizes angeben. Die Zeilen der Koeffizientenmatrix sind in den Feldern `value` und `col_ind` durch fettgedruckte Linien voneinander getrennt. Die Zeiger des Feldes `row_ptr` weisen auf den Zeilenanfang. Die Abbildung verdeutlicht, daß die Matrixelemente innerhalb einer Zeile in einer beliebigen Reihenfolge angeordnet sein können.

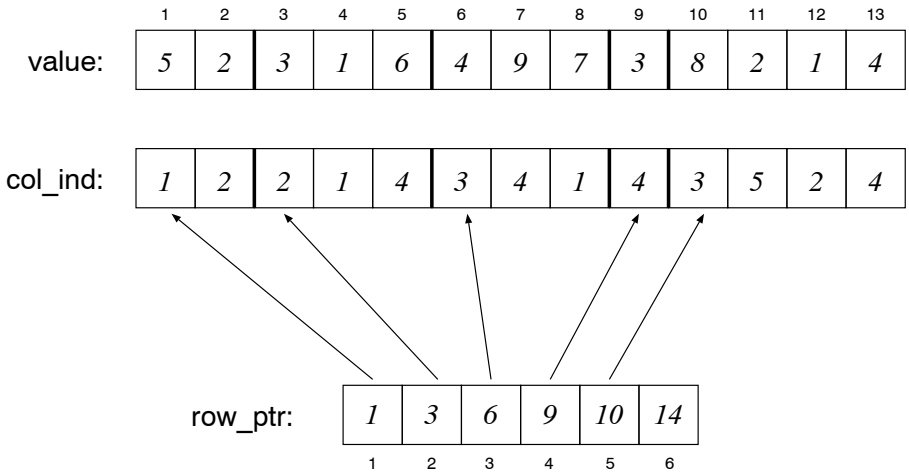


Abbildung 3.3: Beispiel einer Matrix der Ordnung $N = 5$ mit $t = 13$ Nichtnull–Elementen in der Speichertechnik CRS

Der Zugriff — sowohl lesend als auch schreibend — auf ein spezielles Element a_{ij} der Matrix ist in dieser Datenstruktur eine aufwendige Operation, da dazu nach Gleichung 3.1 ein Maximum berechnet werden muß. In den Algorithmen der iterativen Methoden, die in Kapitel 2 beschrieben sind, treten jedoch keine Operationen dieser Form auf. Diese Algorithmen verwenden die Koeffizientenmatrix in der Form von Matrix–Vektor–Produkten mit $\boldsymbol{A}$ und $\boldsymbol{A}^T$. Legt man die übliche Speichertechnik von Vektoren als eindimensionales Feld zugrunde, dann können diese Matrix–Vektor–Produkte in CRS effizient berechnet werden. Da die i–te Komponente des Matrix–Vektor–Produktes $y = \boldsymbol{A}x$ durch

$$y_i = \sum_{k=1}^{N} a_{ik} x_k$$

```
for i = 1, ..., N do
    y(i) = 0
    for k = row_ptr(i), ..., row_ptr(i + 1) − 1 do
        y(i) = y(i) + value(k) · x(col_ind(k))
    end for
end for
```

Abbildung 3.4: Sequentielles Matrix–Vektor–Produkt $y = \boldsymbol{A}x$

gegeben ist, wird bei einem Matrix–Vektor–Produkt mit $\boldsymbol{A}$ nacheinander auf die Elemente einer Zeile der Koeffizientenmatrix zugegriffen. Der Pseudocode einer solchen Operation ist in Abbildung 3.4 dargestellt. Da in dieser Datenstruktur nur mit den Nichtnull–Elementen operiert wird, beträgt die Komplexität des Matrix–Vektor–Produktes nach dieser Abbildung $\mathcal{O}(t)$ gegenüber $\mathcal{O}(N^2)$ bei vollständiger Abspeicherung der Koeffizientenmatrix.

Bei dem Matrix–Vektor–Produkt mit der Transponierten der Koeffizientenmatrix wird die i–te Komponente des Produktes $y = \boldsymbol{A}^T x$ durch

$$y_i = \sum_{k=1}^{N} a_{ki} x_k$$

beschrieben. Dies entspricht einem Zugriff auf Spalten der Koeffizientenmatrix, der in der verwendeten Datenstruktur sehr aufwendig ist. Durch indirekte Adressierung des Ergebnisvektors y kann man diesen Nachteil umgehen, wie die Abbildung 3.5 zeigt.

```
for i = 1, ..., N do
    y(i) = 0
end for
for k = 1, ..., N do
    for i = row_ptr(k), ..., row_ptr(k + 1) − 1 do
        y(col_ind(i)) = y(col_ind(i)) + value(i) · x(k)
    end for
end for
```

Abbildung 3.5: Sequentielles Matrix–Vektor–Produkt $y = \boldsymbol{A}^T x$

3.5 Datenverteilung

Auf einem System mit verteiltem Speicher wird in der Regel die Gesamtheit aller Daten auf alle vorhandenen Knoten verteilt. Bei der Datenverteilung ist zu beachten, daß die mit den Daten durchzuführenden Operationen effizient implementiert werden können. Die in den betrachteten iterativen Methoden enthaltenen Daten sind

- Skalare,
- Vektoren und
- Matrizen,

deren verwendete Datenstruktur in Abschnitt 3.4 beschrieben ist. Die Operationen, die mit diesen Daten ausgeführt werden, sind

- die Durchführung von Matrix–Vektor–Produkten,
- die Berechnung von Skalarprodukten von Vektoren
- die Bildung von Linearkombinationen von Vektoren,
- Vorkonditionierungsschritte und
- skalare Rechenoperationen.

Im folgenden wird die Verteilung der einzelnen Daten auf die Knoten beschrieben und auf die daraus resultierenden Auswirkungen hinsichtlich der Parallelisierung der mit den Daten auszuführenden Operationen hingewiesen. Dabei wird die Datenverteilung von Matrizen im Mittelpunkt stehen. Die parallele Berechnung von Matrix–Vektor–Produkten beruht im wesentlichen auf der Erstellung eines Kommunikationsschemas, das im nächsten Abschnitt beschrieben wird.

Die Verteilung der Matrix orientiert sich an der verwendeten Speichertechnik CRS, in der die Matrixzeilen hintereinander abgespeichert werden. In der parallelen Implementierung werden vollständige Zeilen der Koeffizientenmatrix auf die Knoten verteilt. Die Verteilung kann nach unterschiedlichen Kriterien erfolgen. Jeder Knoten kann gleich viele Zeilen erhalten — vorausgesetzt, daß die Anzahl der Zeilen ganzzahlig durch die Anzahl der verwendeten Knoten teilbar ist. Eine weitere Möglichkeit besteht darin, den Knoten in etwa gleich viele Nichtnull–Elemente zuzuweisen. Bei einer dritten Variante wird versucht, die Anzahl der durchzuführenden Operationen gleichmäßig auf die Knoten zu verteilen. Ziel dieses Kriteriums ist es, den Lastausgleich der parallelen Implementierung zu optimieren. Im Gegensatz zu den ersten beiden Kriterien, die sich ausschließlich auf die Struktur der Koeffizientenmatrix beziehen, fließen beim dritten Kriterium Eigenschaften des verwendeten Algorithmus und Rechnertyps in die Datenverteilung ein. Die Ergebnisse aus [4] belegen,

daß das dritte Kriterium besonders bei unregelmäßiger Besetzungsstruktur der Koeffizientenmatrix günstig ist. Daher wird dieses Kriterium zur Datenverteilung der vorliegenden Arbeit benutzt und im folgenden beschrieben, wobei die übliche Konvention $\sum_{i=j}^{k} \alpha_i = 0$, falls $j > k$, für die leere Summe verwendet wird.

Eine Matrix der Ordnung N mit insgesamt t Nichtnull–Elementen soll auf p Knoten verteilt werden. Weiterhin wird die Anzahl der Nichtnull–Elemente in Zeile k der Matrix mit z_k bezeichnet, so daß $t = \sum_{k=1}^{N} z_k$ gilt. Die vorzunehmende Datenverteilung weist den Knoten aufeinanderfolgende Matrixzeilen zu. Beginnend bei Knoten 0 erhalten die Knoten $i \in \{0, 1, \ldots, p-1\}$ in aufsteigender Knotennumerierung N_i aufeinanderfolgende Zeilen der Matrix. Einem Knoten i werden dabei insgesamt t_i Nichtnull–Elemente zugewiesen, so daß

$$N = \sum_{i=0}^{p-1} N_i \qquad \text{und} \qquad t = \sum_{i=0}^{p-1} t_i$$

gilt. Die Anzahl der Nichtnull–Elemente t_i eines Knotens i kann wie folgt berechnet werden. Die Nummer der letzten Zeile der Koeffizientenmatrix, die dem Knoten i zugewiesen wird, werde mit u_i bezeichnet. Es gilt

$$u_i = \sum_{k=0}^{i} N_k.$$

Bei der Datenverteilung erhält der Knoten i die Zeilen $u_{i-1} + 1, \ldots, u_i$ der Koeffizientenmatrix. Die gesuchte Funktion t_i ist durch die Aufsummierung der Nichtnull–Elemente der Zeilen des Knotens i, also durch

$$t_i(u_{i-1}, N_i) = \sum_{k=u_{i-1}+1}^{u_{i-1}+N_i} z_k$$

gegeben.

Die Aufgabe der vorzunehmenden Datenverteilung besteht darin, die Werte N_i zu bestimmen. Die Anzahl der arithmetischen Operationen eines Matrix–Vektor–Produktes ist proportional zu der Anzahl t der Nichtnull–Elemente der Matrix, während die auftretenden Vektoroperationen proportional zur Zeilenanzahl N sind. Werden pro Iterationsschritt s Matrix–Vektor–Produkte berechnet, so beträgt der Gesamtaufwand einer Iteration $c_1 s\, t + c_2 N + c_3$, mit $c_1, c_2, c_3 \in \mathbb{R}$. Die Konstante c_3 beschreibt dabei den Aufwand von skalaren Rechenoperationen und wird in der folgenden Betrachtung, die sich auf große Systeme bezieht, vernachlässigt. Der Anteil der arithmetischen Operationen des Knotens i am Gesamtaufwand ist daher durch

$$\frac{s\, t_i + \xi N_i}{s\, t + \xi N} \qquad \text{mit} \qquad \xi \in \mathbb{R}$$

gegeben. Der Parameter ξ ist ein Maß für das zeitliche Verhältnis von Matrix–Vektor–Produkten und übrigen Operationen. Aus diesem Grund ist ξ sowohl vom

verwendeten Rechner als auch vom ausgewählten Algorithmus abhängig. Der Lastausgleich von parallel ausführbaren Operationen ist optimal, wenn jeder Knoten den p–ten Anteil aller Operationen erhält. Die Datenverteilung versucht, eine möglichst gleichmäßige Verteilung der arithmetischen Operationen auf die Knoten vorzunehmen und wird daher durch

$$N_i = \begin{cases} \min\limits_{1 \le r \le N-u_{i-1}} \left\{ r \;\middle|\; \dfrac{s\,t_i(u_{i-1},r) + \xi r}{s\,t + \xi N} \ge \dfrac{1}{p} \right\} & , \quad i = 0,1,2,\ldots,b \\ N - \sum\limits_{k=0}^{b} N_k & , \quad i = b+1 \\ 0 & , \quad i = b+2,\ldots,p-1 \end{cases}$$

festgelegt. Dabei erfolgt die Verteilung der Zeilen von Knoten 0 in aufsteigender Knotennumerierung bis zu einem Knoten b, solange das erste Kriterium erfüllt werden kann. Der folgende Knoten erhält dann den Rest der Zeilen. Sind weitere Knoten vorhanden, erhalten sie keine Daten. Dieser Fall tritt nur dann auf, wenn ein kleines Gleichungssystem mit sehr vielen Knoten berechnet wird. Normalerweise wird die Knotenanzahl dem Datenvolumen des Gleichungssystems angepaßt, so daß der letzte Fall nicht auftritt.

Zu einem speziellen Rechner und einer gegebenen iterativen Methode, die in jedem Iterationsschritt s Matrix–Vektor–Produkte ausführt, approximiert man den Parameter ξ durch eine Zeitmessung. Dazu mißt man auf einem einzelnen Knoten den Anteil a_{MVP} der Matrix–Vektor–Produkte des Verfahrens am Gesamtaufwand. Dieser Anteil wird zu

$$a_{MVP} = \frac{s\,t}{s\,t + \xi N} = \frac{1}{1 + \xi/(s\,t_{mean})}$$

berechnet, wobei $t_{mean} = t/N$ die mittlere Anzahl der Nichtnull–Elemente einer Zeile der Matrix bezeichnet. Die Auflösung dieser Gleichung nach ξ ergibt eine Näherung für diesen Parameter

$$\xi = \left(\frac{1}{a_{MVP}} - 1 \right) s\,t_{mean}. \tag{3.2}$$

Die beiden oben genannten anderen Kriterien, nach denen man die Verteilung der Zeilen der Matrix durchführen kann, können im dritten Kriterium durch zwei spezielle Werte des Parameters ξ simuliert werden. Bei $\xi = 0$ erhält jeder Knoten in etwa gleich viele Nichtnull–Elemente. Der Grenzfall $\xi \to \infty$ entspricht einer Datenverteilung, in der jedem Knoten ungefähr gleich viele Zeilen zugewiesen werden.

Die Verteilung der Vektoren korrespondiert zu der Datenverteilung der Matrix. Ein Knoten erhält genau dann die i–te Komponente eines Vektors, wenn der Knoten die i–te Zeile der Matrix besitzt. Die Berechnung eines Skalarproduktes von zwei Vektoren $x, y \in \mathbb{R}^N$, also

$$\langle x, y \rangle = \sum_{k=1}^{N} x_k y_k,$$

erfolgt in zwei Phasen. Zuerst berechnet jeder Knoten i den lokalen Anteil l_i des Skalarproduktes

$$l_i = \sum_{k=u_{i-1}+1}^{u_i} x_k y_k.$$

Anschließend werden die lokalen Teile aller Knoten zum vollständigen Skalarprodukt

$$\langle x, y \rangle = \sum_{i=0}^{p-1} l_i$$

aufsummiert; dazu ist eine globale Synchronisation erforderlich. Speziell bei einer hohen Knotenanzahl ist die Synchronisation aller Knoten eine zeitaufwendige Operation. Daher ist es günstig, die Anzahl der globalen Synchronisationspunkte zu minimieren. Dies kann beispielsweise durch Zusammenfassen der Berechnungen von zwei oder mehr Skalarprodukten geschehen, sofern der Algorithmus dies zuläßt.

Mit der beschriebenen Verteilung der Vektorkomponenten auf die Knoten können Linearkombinationen von Vektoren bestimmt werden, ohne daß Knoten miteinander kommunizieren müssen. Die Vorkonditionierungsschritte, die in dem Verfahren QMR benötigt werden, sind von der Form *Löse* $\boldsymbol{M}z = y$. In dieser Arbeit wird $\boldsymbol{M}$ ausschließlich als Diagonalmatrix gewählt. Der Vorkonditionierungsschritt kann daher wie die Berechnung einer Linearkombination lokal auf allen Knoten durchgeführt werden.

Skalare sind lokal auf allen Knoten vorhanden, so daß die Berechnung von skalaren Rechenoperationen in einer parallelen Implementierung ebenfalls lokal erfolgen kann.

3.6 Kommunikationsschema

Im vorigen Abschnitt ist die Datenverteilung von Matrizen und Vektoren beschrieben. Danach besitzt jeder Knoten einen Teil der Vektorkomponenten. Im allgemeinen werden bei der Berechnung einer Komponente eines Vektors y, der als Ergebnis eines Matrix–Vektor–Produktes $y = \boldsymbol{A}x$ berechnet wird, beliebige Vektorkomponenten von x benötigt. Deshalb müssen Komponenten, die auf jeweils anderen Knoten gespeichert sind, angefordert werden. Dazu ist Kommunikation der Knoten untereinander notwendig. Die Information darüber, welcher Knoten welche Daten senden bzw. empfangen muß, wird als Kommunikationsschema bezeichnet. In dieser Arbeit wird das Kommunikationsschema aus [3, 5] verwendet.

Bei der Berechnung eines Skalarproduktes sendet jeder Knoten Daten zu allen anderen Knoten und empfängt seinerseits Daten von allen anderen Knoten. Im Gegensatz dazu benötigt die Durchführung eines Matrix–Vektor–Produktes bei dünnbesetzten Matrizen in der Regel keine globale Kommunikation. Hier muß ein Knoten nur mit einem Teil der anderen Knoten kommunizieren. Die Erstellung des Kommunikationsschemas erfolgt allein aufgrund der Besetzungsstruktur der Matrix. Sie kann

also durch eine Analyse derjenigen Felder der Datenstruktur erfolgen, die die Information über die Zeilen– und Spaltenindizes enthalten. Dazu werden diese Felder sortiert und so umgespeichert, daß Elemente, die einen lokalen Zugriff verursachen, von denen getrennt sind, die auf einen nicht–lokalen Zugriff führen. Das Kommunikationsschema wird zu Beginn einmalig erstellt. Die Ergebnisse werden dann bei jedem durchzuführenden Matrix–Vektor–Produkt wiederverwendet. Durch Überlappung der Kommunikation mit arithmetischen Operationen kann die benötigte Ausführungszeit des Matrix–Vektor–Produktes minimiert werden. Eine ausführlichere Beschreibung geben [3, 4, 5].

3.7 Abbruchkriterien

Bei der Wahl eines Abbruchkriteriums sollte i.a. darauf geachtet werden, daß das Verfahren endet, wenn der Fehler klein genug ist, nicht mehr schnell genug abnimmt oder sich nicht mehr wesentlich ändert. In der vorliegenden Arbeit werden im n–ten Iterationsschritt die folgenden Abbruchkriterien benutzt.

- Die euklidische Norm des Residuums ist kleiner als ein vorgegebener Wert ϵ,

$$\left\| r^{(n)} \right\| < \epsilon. \tag{3.3}$$

- Die euklidische Norm des Residuums ist kleiner als das Produkt eines vorgegebenen Wertes ϵ mit der Norm des Startresiduums,

$$\left\| r^{(n)} \right\| < \epsilon \left\| r^{(0)} \right\|. \tag{3.4}$$

- Die Komponentendifferenz der beiden letzten Iterationsvektoren geht in das Abbruchkriterium

$$\max_{1 \le i \le N} 2 \frac{\mid x_i{}^{(n)} - x_i{}^{(n-1)} \mid}{\mid x_i{}^{(n)} \mid + \mid x_i{}^{(n-1)} \mid} < \epsilon \tag{3.5}$$

  ein, wobei $x_i{}^{(j)}$ die i–te Komponente des Vektors $x^{(j)}$ bezeichnet.

Ein Problem bei den Abbruchkriterien (3.3) und (3.4) ist, daß die Norm des Residuums unter Umständen recht groß ist, obwohl die Lösung bereits gut angenähert ist, oder die Norm sehr klein ist, obwohl der Iterationsvektor noch weit von der Lösung entfernt ist. Kriterium (3.4) hängt zusätzlich sehr stark von der Wahl des Startvektors ab. Insbesondere bei Verfahren, die nach dem QMR–Ansatz arbeiten, werden diese Abbruchkriterien aber häufig benutzt [10, 19, 26, 29, 30]. Während in den Algorithmen CGS und QMR eine Rekursion für das Residuum enthalten ist, ist bei dem Verfahren TFQMR eine explizite Berechnung des Residuums nach der Definition

$r^{(n)} = b - \boldsymbol{A}x^{(n)}$ notwendig. Da hierzu ein weiteres Matrix–Vektor–Produkt berechnet werden muß, kann der Benutzer wählen, wie oft ein Konvergenztest durchgeführt werden soll.

Das Abbruchkriterium (3.5) hängt von der Differenz von zwei aufeinanderfolgenden Iterationsvektoren ab. Es erscheint daher ungünstig, wenn das Konvergenzverhalten plateauförmig verläuft, wie es typischerweise bei Methoden mit QMR–Ansatz der Fall ist. Das nächste Kapitel wird dies verdeutlichen.

Bei allen Implementierungen wird zusätzlich zu den beschriebenen Kriterien der Rechenaufwand nach oben durch die Festlegung einer maximal auszuführenden Iterationsanzahl begrenzt.

3.8 Testmatrizen

An dieser Stelle werden diejenigen linearen Gleichungssysteme beschrieben, die in den numerischen Beispielen der vorliegenden Arbeit verwendet werden. Sie werden durch ihre Koeffizientenmatrix und rechte Seite repräsentiert. Sofern nicht anders erwähnt, werden die rechten Seiten aus den Problemstellungen übernommen. Die wichtigen Kenngrößen aller Matrizen, die im folgenden näher beschrieben werden, sind in der Tabelle 3.1 enthalten. Die Angaben in dieser Tabelle bezüglich des Speicherbedarfes beziehen sich auf die in dieser Arbeit verwendete Speichertechnik. Sie beinhalten den benötigten Speicherplatz für die Koeffizientenmatrix, die rechte Seite und den Startvektor.

3.8.1 Matrizen aus der Umwelttechnik

Bei der Simulation der Ausbreitung von Schadstoffen in geologischen Systemen werden vom Institut für Chemie und Dynamik der Geosphäre (ICG 4) des Forschungszentrums Jülich die folgenden zwei Koeffizientenmatrizen verwendet [56, 57]. Die Matrizen **ICG2D** und **ICG3D** stammen aus einer zwei– bzw. dreidimensionalen Diskretisierung der Problemstellung mit Hilfe der Methode der finiten Elemente und weisen eine regelmäßige Besetzungsstruktur auf. Beide Matrizen sind symmetrisch und positiv definit.

3.8.2 Matrix aus der Strukturmechanik

Im Institut für Sicherheitsforschung und Reaktortechnik des Forschungszentrums Jülich wird in der Untersuchung von thermischen Spannungen in Materialien die symmetrische Matrix **ISR** verwendet. Diese Matrix besitzt im Vergleich zu den Matrizen aus der Umwelttechnik eine unregelmäßige Besetzungsstruktur. In einer Zeile der Matrix sind vergleichsweise viele Nichtnull–Elemente enthalten. Die Matrix ist positiv definit.

3.8.3 Matrix aus einem impliziten FE–Modell

Die Matrix **IMPLFE** enstammt einer Problemstellung aus der Automobilindustrie, in der die Verformung beim Pressvorgang von Motorhauben mit Hilfe der Methode der finiten Elemente beschrieben wird. Bei der Lösung von Gleichungssystemen mit dieser symmetrischen, positiv definiten Koeffizientenmatrix treten Schwierigkeiten auf, wenn man übliche Lösungsverfahren anwendet.

3.8.4 Harwell–Boeing Sparse Matrix Collection

In der Literatur findet man bei Verfahren zur iterativen Lösung von linearen Gleichungssystemen häufig numerische Beispiele, die aus einer Standardsammlung von dünnbesetzten Matrizen entnommen sind. Diese Sammlung (*Harwell–Boeing Sparse Matrix Collection*) [13, 14] enthält Matrizen aus vielen Anwendungsgebieten in unterschiedlichen Speichertechniken. Daher war eine Datenkonversion in die in dieser Arbeit verwendete Speichertechnik notwendig. Normalerweise enthalten die Beispiele aus der Sammlung sowohl eine Koeffizientenmatrix als auch eine rechte Seite des Gleichungssystems. Falls zu einer Matrix keine rechte Seite gegeben ist, werden die rechten Seiten so erzeugt, daß die exakte Lösung durch $x^* = (1, 1, \ldots, 1)^T$ gegeben ist. Alle Matrizen, die in dieser Arbeit aus der Harwell–Boeing Sparse Matrix Collection untersucht werden, sind unsymmetrisch.

- Bei Problemstellungen aus dem Bereich der nichtlinearen Astrophysik tritt die Matrix **MCFE** auf.
- Die Matrix **WEST2021** enthält Nullen auf der Hauptdiagonalen. Sie resultiert aus Aufgabenstellungen des Chemie–Ingenieurwesens.
- In Berechnungen, die die physikalischen Vorgänge in elektrischen Schaltkreisen modellieren, wird u.a. die Matrix **JPWH_991** verwendet.
- Die Matrix **LNSP3937** entsteht bei der Lösung von linearisierten Navier–Stokes–Gleichungen. Sie weist symmetrische Muster auf.
- Der Kerntechnik entstammt die Matrix **NNC1374**, die bei der Modellierung eines gasgekühlten Reaktorkerns benutzt wird.
- Aus dem Bereich der Erdölförderung ist die Matrix **ORSREG_1** entnommen, die aus einer Diskretisierung einer dreidimensionalen Problemstellung resultiert.
- Die Matrix **PORES 2** besitzt symmetrische Muster und kommt aus der Modellierung der Wasserversorgung.

- Die Elemente der Matrix **WATT1** stammen aus dem Bereich der Petro–Chemie. Die Einträge der Koeffizientenmatrix unterscheiden sich um mehrere Größenordnungen voneinander (bis zu einem Faktor von 10^{14}). Die Matrix besitzt in etwa Bandstruktur.

Bezeichnung	N	t	t_{max}	t_{mean}	b [%]	MB
ICG2D	17 368	304 000	18	17,5	0,1	4,0
ICG3D	49 392	1 242 814	27	25,2	0,1	15,9
ISR	25 222	3 856 386	485	152,9	0,6	46,8
IMPLFE	13 860	661 010	49	47,7	0,3	8,2
MCFE	765	24 382	109	31,9	4,2	0,3
JPWH_991	991	6 027	16	6,1	0,6	0,1
NNC1374	1 374	8 606	16	6,3	0,5	0,1
ORSREG_1	2 205	14 133	7	6,4	0,3	0,2
PORES 2	1 224	9 613	30	7,9	0,6	0,1
WEST2021	2 021	7 353	26	3,6	0,2	0,1
LNSP3937	3 937	25 407	13	6,5	0,2	0,4
WATT1	1 856	11 360	7	6,1	0,3	0,2

Tabelle 3.1: Daten der Testmatrizen, die in dieser Arbeit verwendet werden. Eine Matrix besitzt die Ordnung N und enthält insgesamt t Nichtnull–Elemente. Eine Zeile der Matrix besteht aus maximal t_{max} und im Mittel t_{mean} Nichtnull–Elementen. Der Besetzungsgrad beträgt $b = t_{mean}/N$ und ist in Prozent angegeben. Die letzte Spalte enthält den benötigten Speicherplatz in Megabyte des linearen Systems (einschließlich rechter Seite und Startvektor).

3.9 Startvektor

Wie schnell eine Folge von Iterationsvektoren gegen die exakte Lösung eines vorgegebenen Gleichungssystems konvergiert, hängt unter anderem von der Wahl des Startvektors ab. Auf eine Untersuchung der verwendeten Iterationsverfahren hinsichtlich unterschiedlicher Startvektoren wird im Rahmen dieser Arbeit verzichtet. Stattdessen wird in Anlehnung an [19, 26] durchgängig der Nullvektor als Startvektor gewählt.

Kapitel 4

Sequentielle Ergebnisse

In diesem Kapitel werden Resultate vorgestellt, die charakteristische Eigenschaften der drei Verfahren CGS, QMR und TFQMR vorstellen. Dazu werden Gleichungssysteme ausgesucht, deren Koeffizientenmatrix auf einem Knoten des Parallelrechners gespeichert werden kann. Alle in diesem Kapitel vorgestellten Ergebnisse sind aus sequentiellen Messungen gewonnnen worden. Sie bilden die Grundlage, auf der die im folgenden Kapitel enthaltenen parallelen Aspekte des QMR–Ansatzes beurteilt werden können. An dieser Stelle werden zunächst die betrachteten drei Verfahren durch ihr Konvergenzverhalten charakterisiert. Darunter wird hier der Verlauf der Residuumsnorm mit zunehmender Iterationsanzahl verstanden. Anschließend wird die Wahl einer Optimierungsstufe durchgeführt, mit der dann alle Programme der vorliegenden Arbeit einheitlich übersetzt werden. Am Ende des Kapitels wird beschrieben, wieviele Iterationen ein Verfahren bei einem bestimmten Gleichungssystem benötigt und wie zeitaufwendig die zugehörigen Berechnungen sind.

4.1 Konvergenzverhalten von CGS, QMR und TFQMR

Bei der Beurteilung einer iterativen Methode muß vor allem berücksichtigt werden, wie schnell die Folge der Iterationsvektoren gegen die exakte Lösung des linearen Gleichungssystems konvergiert. Das Konvergenzverhalten eines Verfahrens wird dabei extrem von dem vorgegebenen Gleichungssytem beeinflußt. In diesem Abschnitt wird für eine Auswahl von Gleichungssystemen beschrieben, wie die Norm des Residuums von der Iterationsanzahl abhängt. Bei einem Vergleich der Algorithmen CGS, QMR und TFQMR ist eine Besonderheit von TFQMR zu beachten, auf die vorab hingewiesen wird. Anschließend wird das Konvergenzverhalten der drei Algorithmen für ausgewählte Systeme betrachtet.

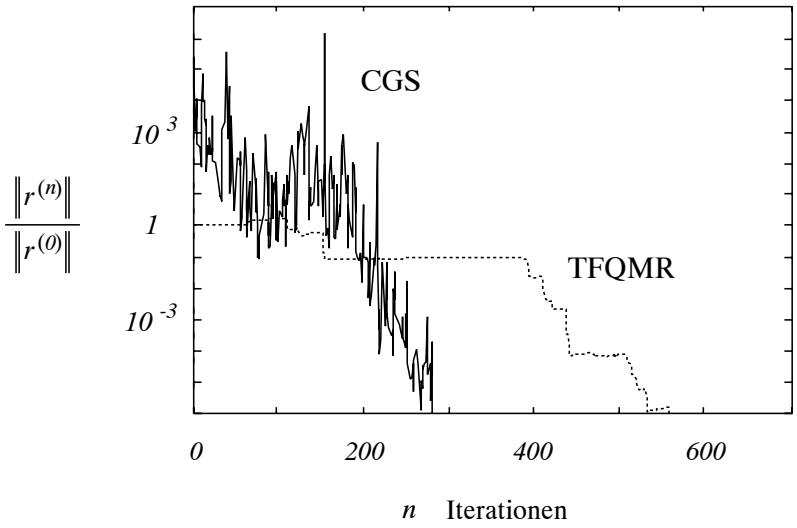


Abbildung 4.1: Skalierte euklidische Norm des Residuums in Abhängigkeit der Iterationen beim Beispiel **ICG2D**.

4.1.1 Besonderheiten von TFQMR

Der Algorithmus TFQMR unterscheidet sich in einem Punkt formal von den beiden anderen in dieser Arbeit vorgestellten Verfahren. Dieser Unterschied soll anhand des Beispiels **ICG2D** aufgezeigt werden. In der Abbildung 4.1 ist für die zwei Verfahren CGS und TFQMR die skalierte Norm des Vektors $r^{(n)}$ über n aufgetragen. Die Abbildung zeigt für CGS ein stark oszillierendes Verhalten, während die Norm in TFQMR einen glatten Verlauf vorweist. Allerdings konvergiert CGS wesentlich schneller als TFQMR und erscheint daher vorteilhafter.

Wie dem Kapitel 2, das die Iterationsverfahren vorstellt, entnommen werden kann, erzeugt TFQMR in jeder Iterationsschleife zwei Iterierte, wozu die Berechnung von zwei Matrix–Vektor–Produkten notwendig ist. Dagegen wird in CGS bei gleicher Anzahl von Matrix–Vektor–Produkten eine Iterierte erzeugt. Als Kostenmaß für die Berechnung der Iterierten kann die Anzahl der auszuführenden Matrix–Vektor–Produkte einer Iterationsschleife dienen. Eine Darstellung der Residuumsnorm, die sich an den Kosten orientiert, korrigiert das ungünstig erscheinende Konvergenzverhalten von TFQMR aus der oben genannten Abbildung. Die Abbildung 4.2 zeigt das Konvergenzverhalten der beiden Algorithmen nun in Abhänigkeit von der Anzahl der Matrix–Vektor–Produkte. Daraus resultiert dieselbe Kurve für TFQMR wie in der vorherigen Abbildung, während die CGS–Kurve um den Faktor zwei gedehnt wird. Die Abbildung verdeutlicht, daß TFQMR fast während des gesamten Verlaufs unterhalb der über mehrere Zehnerpotenzen oszillierenden Kurve von CGS liegt und daher günstiger als CGS ist.

Die kostenorientierte Darstellung von TFQMR erlaubt einen realistischen Vergleich zwischen den verwendeten Verfahren. Daher wird die Darstellung von TFQMR

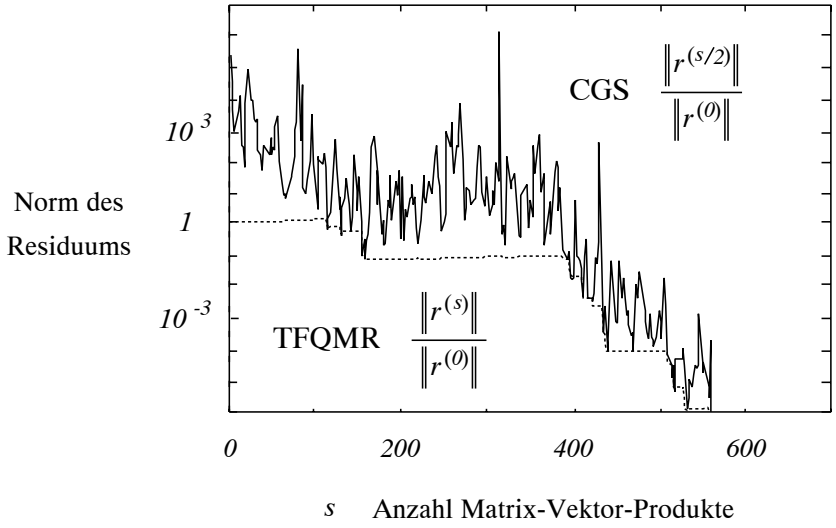


Abbildung 4.2: Skalierte euklidische Norm des Residuums in Abhängigkeit der Anzahl der Matrix–Vektor–Produkte beim Beispiel **ICG2D**. (Die Residuen von CGS sind nur für eine gerade Anzahl von Matrix–Vektor–Produkten s definiert.)

in allen Diagrammen, die das Konvergenzverhalten der Verfahren beschreiben, der Kostenorientierung angepaßt. Diese Betrachtungsweise wird wie folgt auf die Angabe von Iterationsanzahlen übertragen. Erfüllt eine TFQMR–Iterierte $x^{(2n-1)}$ oder $x^{(2n)}$ ein vorgegebenes Abbruchkriterium, dann wird die Iterationsanzahl, die das Verfahren benötigt, mit n statt mit dem tatsächlichen Index der Iterierten, die das Abbruchkriterium erfüllt, angegeben.

4.1.2 Konvergenzverhalten

Für die betrachteten iterativen Verfahren wird nun anhand von vier Gleichungssystemen untersucht, wie sich die Residuumsnorm mit der Iterationsanzahl verändert. Dazu werden die vier Abbildungen 4.3–4.6 betrachtet, in denen bei gleichem Ordinatenbereich für die absolute Residuumsnorm auf der Abszisse jeweils unterschiedliche Bereiche für die Iterationsanzahlen aufgetragen sind. Im ersten Überblick zeigen die Abbildungen ein stark oszillierendes Verhalten von CGS, während QMR und TFQMR ein glattes Verhalten aufweisen, das für den QMR–Ansatz charakteristisch ist [19, 24, 26, 47]. Im folgenden werden die Diagramme näher beschrieben.

In der Abbildung 4.3 sind die Residuumsnormen für das Beispiel **JPWH_991** dargestellt. In diesem Diagramm liegen alle drei Kurven nahe zusammen. Bei niedrigen Iterationsanzahlen ist die Residuumsnorm von QMR kleiner als die der beiden anderen Verfahren. Mit fortlaufender Iterationsanzahl nimmt dieser Unterschied jedoch immer weiter ab, bis schließlich für Iterationsanzahlen größer 35 die Norm von QMR größer als die von CGS und TFQMR ist. Die Norm von TFQMR liegt zu Beginn des betrachteten Bereichs zwischen den Normen von QMR und CGS. Im mittleren Bereich besitzt TFQMR die kleinste der drei Normen. Gegen Ende bleibt die Norm

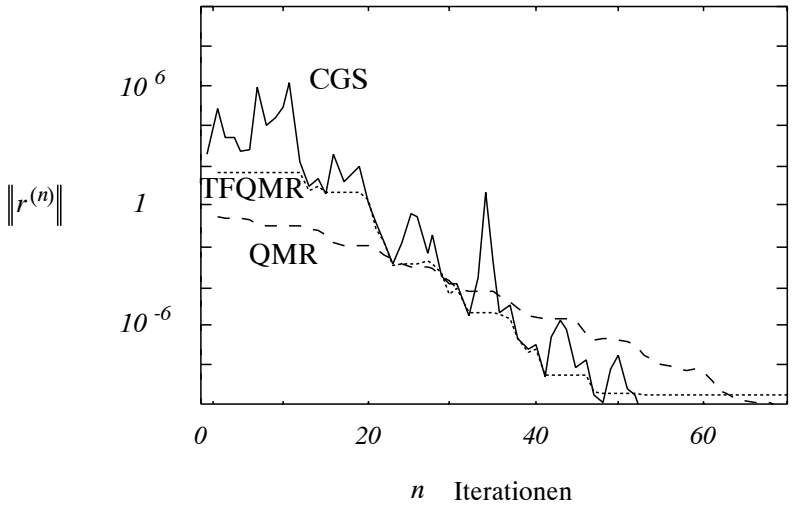


Abbildung 4.3: Euklidische Norm des Residuums in Abhängigkeit der Iterationen beim Beispiel **JPWH_991**.

von TFQMR konstant, während die Normen der anderen beiden Verfahren noch weiter abnehmen. Die Norm von CGS schwankt stark und wird von der TFQMR–Norm solange näherungsweise nach unten beschränkt, bis das konstante Verhalten der Norm von TFQMR einsetzt. Die Residuumsnorm von CGS erreicht als erstes Verfahren die Grenze des betrachteten Bereichs von 10^{-10}.

Die Unterschiede der drei Verfahren treten in dem Beispiel **ORSREG_1** deutlicher hervor. Wie die Abbildung 4.4 zeigt, ist die Norm von QMR bei niedrigen Iterationsanzahlen wiederum kleiner als bei den beiden anderen Verfahren, diesmal aber

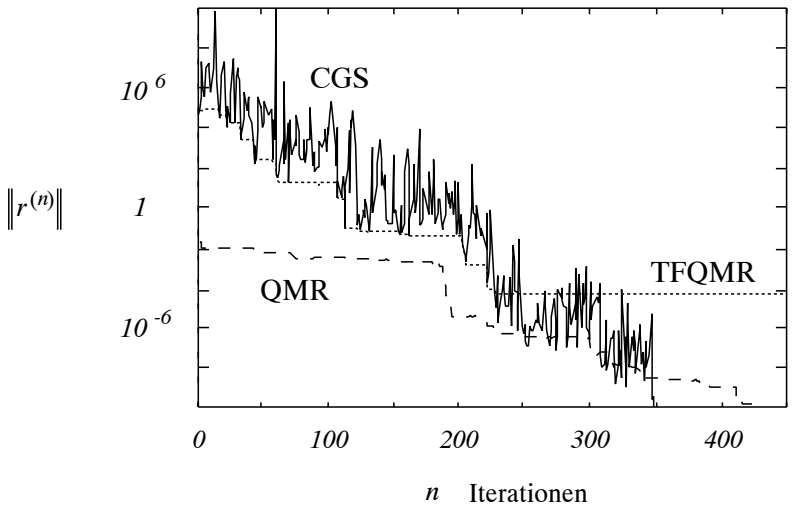


Abbildung 4.4: Euklidische Norm des Residuums in Abhängigkeit der Iterationen beim Beispiel **ORSREG_1**.

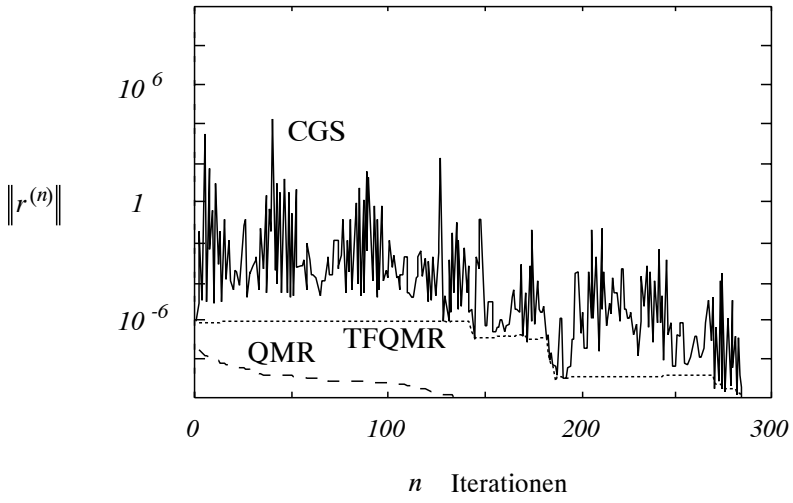


Abbildung 4.5: Euklidische Norm des Residuums in Abhängigkeit der Iterationen beim Beispiel **WATT1**.

um mehrere Größenordnungen. Im Unterschied zur vorigen Abbildung bleibt QMR über den gesamten betrachteten Bereich unterhalb von TFQMR. Ausgeprägt ist bei TFQMR das plateauförmige Verhalten, das für Iterationsanzahlen größer 250 keine weitere Abnahme der Residuumsnorm zeigt. Bei CGS kann man das oszillierende Verhalten, die Beschränkung durch TFQMR nach unten sowie das Erreichen des Grenzwertes von 10^{-10} vor den anderen Verfahren genauer beobachten.

Bei dem Beispiel **WATT1** aus Abbildung 4.5 wird CGS zwar noch von TFQMR nach unten beschränkt, jedoch ist die Schranke nicht mehr so scharf wie bei den anderen

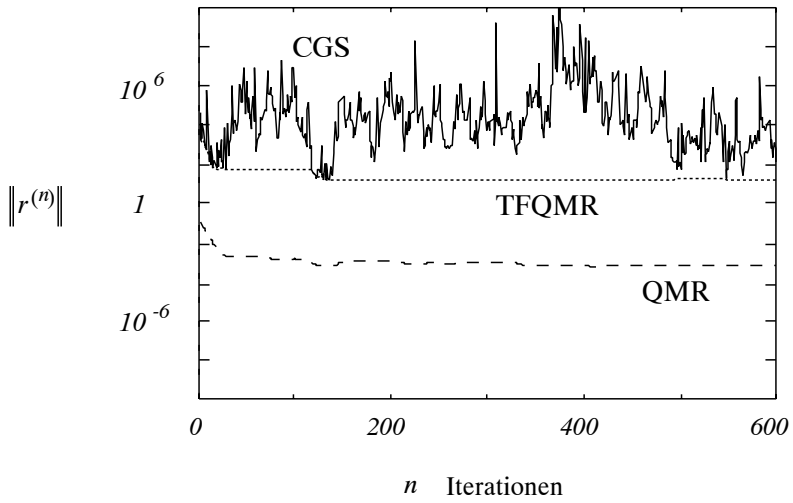


Abbildung 4.6: Euklidische Norm des Residuums in Abhängigkeit der Iterationen beim Beispiel **NNC1374**.

Beispielen. Über den gesamten betrachteten Bereich gilt für die Residuumsnormen der drei Verfahren, daß CGS bei einer festen Iterationsanzahl die größte Norm besitzt. Die zweitgrößte Norm ist die von TFQMR, während QMR die geringste Norm vorweist. Bei diesem Beispiel ist daher QMR hinsichtlich des Konvergenzverhaltens günstiger als TFQMR und CGS.

Bei einigen Systemen wird das Abbruchkriterium von allen drei Verfahren nach einer vorgegebenen, maximalen Anzahl von Iterationsschritten nicht erfüllt. Zu diesen Systemen gehören **WEST2021**, **MCFE**, **PORES 2**, **LNSP3937** und **NNC1374**. Die Norm des letzten Systems ist in Abbildung 4.6 dargestellt. Die Tendenz, daß die Residuumsnormen in der Reihenfolge CGS, TFQMR und QMR abnehmen, bleibt gültig.

4.2 Festlegung der Optimierungsstufe

Um ein Programm effizient ausführen zu können, benötigt man sowohl Kenntnisse über die verwendete Rechnerarchitektur als auch Erfahrungen im Umgang mit dem benutzten Compiler. In diesem Abschnitt werden die Ergebnisse zusammengefaßt, die aus Untersuchungen mit dem FORTRAN–Compiler der Firma The Portland–Group, Inc., Release 4.5, stammen. Die Wahl derjenigen Optimierungsstufe, die eine minimale Ausführungszeit liefert, ist einerseits vom Algorithmus und andererseits von den Eingabedaten abhängig. Anstatt für jeden der drei Algorithmen und jedes verwendete lineare Gleichungssystem eine eigene Optimierungsstufe mit minimaler Ausführungszeit zu bestimmen, wird eine einheitliche Optimierungsstufe festgelegt, um den Aufwand dieser Teiluntersuchung zu begrenzen. In dieser Arbeit werden dann alle Programme mit dieser Optimierungsstufe übersetzt.

Die Optimierungsstufen des verwendeten Compiler sind in Abschnitt 3.3.3 beschrieben. Alle Messungen dieses Abschnitts sind mit dem Abbruchkriterium $\|r^{(n)}\| < 10^{-5}$ durchgeführt worden. Aus der Vielzahl von Messungen werden an dieser Stelle wesentliche Aspekte anhand der zwei Beispiele **ICG2D** und **WATT1** erläutert. Dabei stehen die folgenden Fragestellungen im Vordergrund:

- Wie ändert sich die Iterationsanzahl, die zur Erfüllung des vorgegebenen Abbruchkriteriums benötigt wird, in Abhängigkeit von der gewählten Optimierungsstufe?
- Welchen Einfluß auf das zeitliche Verhalten der ausgeführten Programme besitzen die Optimierungsstufen?

Die Tabellen 4.1 und 4.2 geben die Ergebnisse der Verfahren CGS und QMR für das Beispiel **ICG2D** wieder. Das Verfahren TFQMR verhält sich bei diesem Beispiel in bezug auf die Abhängigkeit von den Optimierungsstufen ähnlich wie QMR. Daher werden die Ergebnisse von TFQMR an dieser Stelle nicht präsentiert.

ICG2D	CGS		
Optimierungsstufe	n	Zeit [s]	Zeit/n [ms]
O0	363	299,7	825,5
O1	363	265,1	730,3
O2	363	143,3	394,7
O3	363	104,0	286,6
O4	363	103,7	285,6
O4 Knoieee	413	118,0	285,6
O4 Knoieee Mvect	364	119,1	327,1

Tabelle 4.1: Einfluß der Optimierungsstufen beim Verfahren CGS. Dargestellt sind die Anzahl n der Iterationen, die dazu benötigte Zeit in Sekunden und der Quotient der beiden Größen in Millisekunden (bei verwendeter Matrix **ICG2D**).

Tabelle 4.1 zeigt, daß bei CGS die zur Erfüllung des Abbruchkriteriums benötigte Iterationsanzahl bei den Optimierungsstufen *O0* bis *O4* identisch ist. Bei Hinzuschaltung der Option *Knoieee* zur Optimierungsstufe *O4* steigt die Iterationsanzahl von 363 auf 413. Da bei der Option *Knoieee* die Programmsemantik durch die Verwendung einer unterschiedlichen Arithmetik geändert wird, können Rundungsfehler eine veränderte Iterationsanzahl bewirken. Bei zusätzlicher Benutzung der Option *Mvect* sinkt die Iterationsanzahl bis fast auf den Wert der Optimierungsstufen *O0* bis *O4*. Hinsichtlich des Zeitverhaltens von CGS gilt, daß bei den Optimierungsstufen *O0* bis *O4* eine Erhöhung der Optimierungsstufe jeweils einen zeitlichen Gewinn darstellt. Die Optimierungsstufe *O3* unterscheidet sich dabei jedoch nur geringfügig von *O4*. Die auf die Iterationsanzahl bezogene Zeit bleibt bei *O4 Knoieee* im Vergleich zu *O4* unverändert. Da die Iterationsanzahl steigt, ist jedoch der absolute Zeitbedarf größer. Auf die Iterationsanzahl bezogen verlangsamt die zusätzliche Verwendung von *Mvect* die Ausführungszeit.

In der Tabelle 4.2 sind für das gleiche Beispiel **ICG2D** die Ergebnisse des Verfahrens QMR dargestellt. Wie bei CGS sind die Iterationsanzahlen bei den Optimierungsstufen *O0* bis *O4* konstant. Im Unterschied zu CGS bleibt die Iterationsanzahl bei der Optimierungsstufe *O4 Knoieee* aber gleich. Wird zusätzlich die Option *Mvect* benutzt, so ist die Iterationsanzahl wie bei CGS gegenüber *O4* um eins erhöht. Bei QMR gilt ebenso die Aussage bezüglich der gemessenen Zeiten, daß bei Zunahme der Optimierungsstufe bis einschließlich *O4* die Programmausführung jeweils beschleunigt wird. Im Gegensatz zu CGS bewirkt die Hinzunahme von *Knoieee* bei QMR einen zeitlichen Vorteil. Im Vergleich zu der Optimierungsstufe *O4* ist die Ausführungszeit bei *O4 Knoieee* um den Faktor 1,7 deutlich kleiner. Wie bei CGS erhöht die Hinzunahme von *Mvect* die benötigte Zeit.

ICG2D	QMR		
Optimierungsstufe	n	Zeit [s]	Zeit/n [ms]
O0	268	352,4	1314,9
O1	268	329,5	1230,0
O2	268	192,4	718,1
O3	268	170,8	637,5
O4	268	170,7	637,1
O4 Knoieee	268	101,3	377,9
O4 Knoieee Mvect	269	116,0	431,4

Tabelle 4.2: Einfluß der Optimierungsstufen beim Verfahren QMR. Dargestellt sind die Anzahl n der Iterationen, die dazu benötigte Zeit in Sekunden und der Quotient der beiden Größen in Millisekunden (bei verwendeter Matrix **ICG2D**).

Weitere Beobachtungen werden anhand des Beispiels **WATT1** durchgeführt. Bei diesem Beispiel wird das Abbruchkriterium von allen drei Verfahren bereits nach einer geringen Iterationsanzahl erreicht. Aus den folgenden Gründen wird das Abbruchkriterium an dieser Stelle modifiziert. In jedem Iterationsschritt von TFQMR werden zwei Iterierte erzeugt, die zugehörigen Residuen sind aber nicht in expliziter Form im Algorithmus enthalten. Um jedes Residuum zu berechnen, müssen deshalb in jeder Iteration zwei zusätzliche Matrix–Vektor–Produkte durchgeführt werden. Daraus resultiert in TFQMR die Berechnung von vier Matrix–Vektor–Produkten, während in CGS und QMR nur zwei berechnet werden. Um bei einem Vergleich der Zeiten den zusätzlichen Aufwand der beiden Matrix–Vektor–Produkte in TFQMR zu eliminieren, wird als Abbruchkriterium eine zuvor bestimmte, feste Iterationsanzahl verwendet, bei der die Bedingung $\|r^{(n)}\| < 10^{-5}$ erfüllt ist. Die auf diese Iterationsanzahl bezogenen gemessenen Zeiten sind in der Tabelle 4.3 enthalten.

Das Verfahren CGS verhält sich beim Beispiel **WATT1** bis auf eine Ausnahme wie bei dem vorherigen Beispiel **ICG2D**. Diese Ausnahme besteht darin, daß die Zeit beim Übergang von *O4 Knoieee* nach *O4 Knoieee Mvect* konstant bleibt. Bei der Methode QMR steigt der Faktor, um den die Optimierungsstufe *O4 Knoieee* schneller als *O4* ist, von 1,7 auf den Wert 2. Bei einem Vergleich der drei Verfahren untereinander kann ein Trend festgestellt werden, der unabhängig von der Optimierungsstufe ist. CGS besitzt pro Iterationsschritt den geringsten Zeitbedarf der drei Methoden, während QMR die zeitaufwendigste Methode darstellt. TFQMR befindet sich zwischen diesen beiden Verfahren. Diese Tatsache kann der Tabelle durch die Zunahme der gemessenen Zeiten von links nach rechts innerhalb einer Zeile entnommen werden.

WATT1	Zeit/n [ms]		
Optimierungsstufe	CGS	TFQMR	QMR
O0	48,2	56,1	100,8
O1	42,6	51,6	95,4
O2	23,0	26,8	58,0
O3	20,0	23,7	55,8
O4	20,0	23,2	55,7
O4 Knoieee	20,0	22,9	28,4
O4 Knoieee Mvect	20,0	30,2	37,5

Tabelle 4.3: Einfluß der Optimierungsstufen beim Verfahren CGS, QMR und TFQMR. Dargestellt ist die auf die Iterationsanzahl bezogene Zeit in Millisekunden (bei verwendeter Matrix **WATT1**).

Zusammenfassend kann festgestellt werden, daß der Einfluß der Optimierungsstufen stark von dem verwendeten Beispiel und dem benutzten Algorithmus abhängt. Innerhalb der Optimierungsstufen *O1* bis *O4* tritt keine Änderung der Iterationsanzahlen auf, außerdem besitzen *O3* und *O4* den geringsten, in etwa gleichen Zeitbedarf. Die Verwendung von *Mvect* bringt keinen Vorteil. Die Hinzuschaltung von *Knoieee* kann bei bestimmten Verfahren einen erheblichen Zeitgewinn darstellen. Da diese Vorteile nur vereinzelt auftreten und eine mögliche Erhöhung der Iterationsanzahlen sowie numerische Ungenauigkeiten der Verwendung von *Knoieee* entgegenstehen, wird diese Optimierungsstufe nicht benutzt. Die Optimierungsstufe *O4* erzeugt in einigen Anwendungen fehlerhafte numerische Ergebnisse, so daß für die gesamte Arbeit die Optimierungsstufe *O3* festgelegt wird.

4.3 Numerisches Verhalten und Performance

Der vorherige Abschnitt beschreibt das Konvergenzverhalten der drei iterativen Verfahren CGS, TFQMR und QMR. Dabei steht die Änderung der Residuumsnorm mit fortschreitender Iterationsanzahl im Mittelpunkt der Darstellung. Diese Betrachtungsweise vermittelt einen allgemeinen Eindruck über die Eigenschaften der Iterationsverfahren. Speziellere Informationen bezüglich der Algorithmen sind in diesem Abschnitt enthalten. Zu einem vorgegebenen Abbruchkriterium wird untersucht, wieviele Iterationen ein Verfahren benötigt und wieviel Zeit insgesamt für die Berechnung der Lösung aufgewendet werden muß. Letzteres ist ein wichtiger Punkt in allen praktischen Anwendungen, denn die Angabe von Iterationsanzahlen allein gibt keine Auskunft darüber, wie aufwendig eine einzelne Iteration ist. Im folgenden werden die Ergebnisse diskutiert, die aus Untersuchungen mit drei unterschiedlichen

Gleichungssystemen resultieren. Als Abbruchkriterium wird

$$\left\| r^{(n)} \right\| < \epsilon := 10^{-x}$$

vorgegeben. Die Angaben von Iterationsanzahlen und Zeiten, die zur Erfüllung dieses Kriteriums benötigt werden, sind dann über $x = -\log_{10} \epsilon$ aufgetragen. Der Wert x ist ein Maß für die geforderte Genauigkeit der Iterierten.

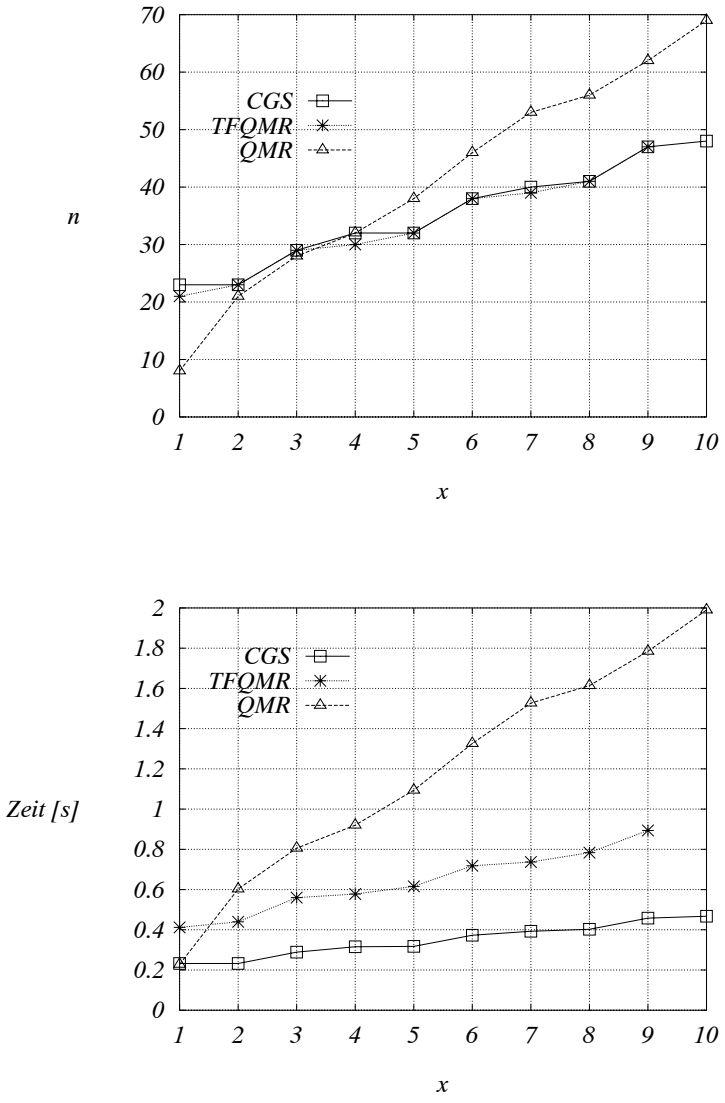


Abbildung 4.7: Für das Beispiel **JPWH_991** ist im oberen Bild die Iterationsanzahl n und im unteren Bild die Gesamtzeit der drei Verfahren in Abhängigkeit von der Genauigkeit $x = -\log_{10} \epsilon$ dargestellt.

Bei den Angaben zu dem Verfahren TFQMR ist eine Vorbemerkung angebracht. Da in diesem Abschnitt die expliziten Iterationsanzahlen zur Erfüllung des Abbruchkriteriums betrachtet werden, ist bei TFQMR die Berechnung der Residuen zu jeder Iterierten erforderlich. Dadurch werden in jeder Iteration von TFQMR vier Matrix–Vektor–Produkte ausgeführt. Die in diesem Abschnitt angegebenen Zeiten von TFQMR stellen daher eine obere Schranke für den Zeitverbrauch von TFQMR dar. Im Produktionsbetrieb wird man TFQMR so einsetzen, daß das Residuum nicht zu jeder Iterierten bestimmt wird und so der Konvergenztest nicht jedesmal durchgeführt wird.

In der Abbildung 4.7 sind die Ergebnisse des Beispiels **JPWH_991** dargestellt. QMR benötigt für kleine Genauigkeiten die geringste Anzahl von Iterationen der drei Verfahren. Mit steigender Genauigkeit werden im Vergleich zu den anderen Verfahren immer mehr Iterationen benötigt, so daß für $x \geq 4$ das Verfahren QMR die meisten Iterationen zur Erfüllung des Abbruchkriteriums verbraucht. Am Ende des betrachteten Bereichs ist die Iterationsanzahl von QMR um ungefähr 45 Prozent über der von CGS. Die Verfahren TFQMR und CGS benötigen bei diesem Beispiel fast im gesamten Genauigkeitsbereich eine vergleichbare Iterationsanzahl. Ein deutlicher Unterschied tritt nur bei $x = 10$ auf, wo die Iterationsanzahl von TFQMR nicht angegeben ist. Die Residuumsnorm zeigt dort ein plateauförmiges Verhalten, bevor diese Genauigkeit erreicht wird, vergleiche Abbildung 4.3. Das untere Diagramm in Abbildung 4.7 zeigt, daß die drei Verfahren beim Beispiel **JPWH_991** hinsichtlich des Zeitbedarfs zur Erfüllung des Abbruchkriteriums eindeutig unterschieden werden können. Von CGS über TFQMR bis hin zu QMR steigt der benötigte Zeitbedarf an.

Die Abbildung 4.8 zeigt das Verhalten der Algorithmen am Beispiel **ORSREG_1**. Auffallend ist hier, daß TFQMR bereits bei Genauigkeiten $x \geq 5$ aus oben genannten Gründen nicht mehr dargestellt ist. Wiederum ist QMR für kleine Genauigkeiten hinsichtlich der Iterationsanzahlen günstig. Der Bereich, ab dem die Iterationsanzahl von QMR diejenige von CGS übersteigt, beginnt bei $x \geq 7$. Sie ist also gegenüber dem vorherigen Beispiel **JPWH_991** zu höheren Genauigkeiten hin verschoben. Bei der Genauigkeit $x = 10$ benötigt QMR im Vergleich zum vorherigen Beispiel statt 45 Prozent nur noch ungefähr 20 Prozent mehr Iterationen. Das Diagramm, das in Abbildung 4.8 die Zeiten darstellt, verdeutlicht, daß sich der Vorteil der geringeren Iterationsanzahlen von QMR in der Zeitdarstellung nicht wiederfindet. In dem Bereich $3 \leq x \leq 6$ ist die Iterationsanzahl von QMR kleiner als die von CGS, der Zeitbedarf jedoch größer.

Das dritte Beispiel **WATT1** ist in Abbildung 4.9 dargestellt. Bei diesem Beispiel betragen die zur Erfüllung des Abbruchkriteriums notwendigen Iterationsanzahlen für Genauigkeiten $x \leq 4$ bei allen Verfahren zwei bzw. drei. Um sich auf die Unterschiede der Verfahren konzentrieren zu können, wird auf diesen Genauigkeitsbereich in den Diagrammen verzichtet. Die Iterationsanzahlen der Algorithmen CGS und TFQMR sind bis auf $x = 6$ und $x = 7$ annähernd gleich. Sie liegen im gesamten betrachteten Bereich über denen von QMR. Das untere Diagramm in Abbildung 4.9 zeigt einen neuen Aspekt beim Vergleich der Zeiten. Der Vorteil, den QMR bezüglich

der Iterationsanzahlen besitzt, ist im Beispiel **WATT1** so groß, daß QMR bis auf den Wert bei $x = 10$ den kleinsten Zeitbedarf vorweist. Das Verfahren TFQMR ist für Genauigkeiten $x \geq 7$ das zeitaufwendigste.

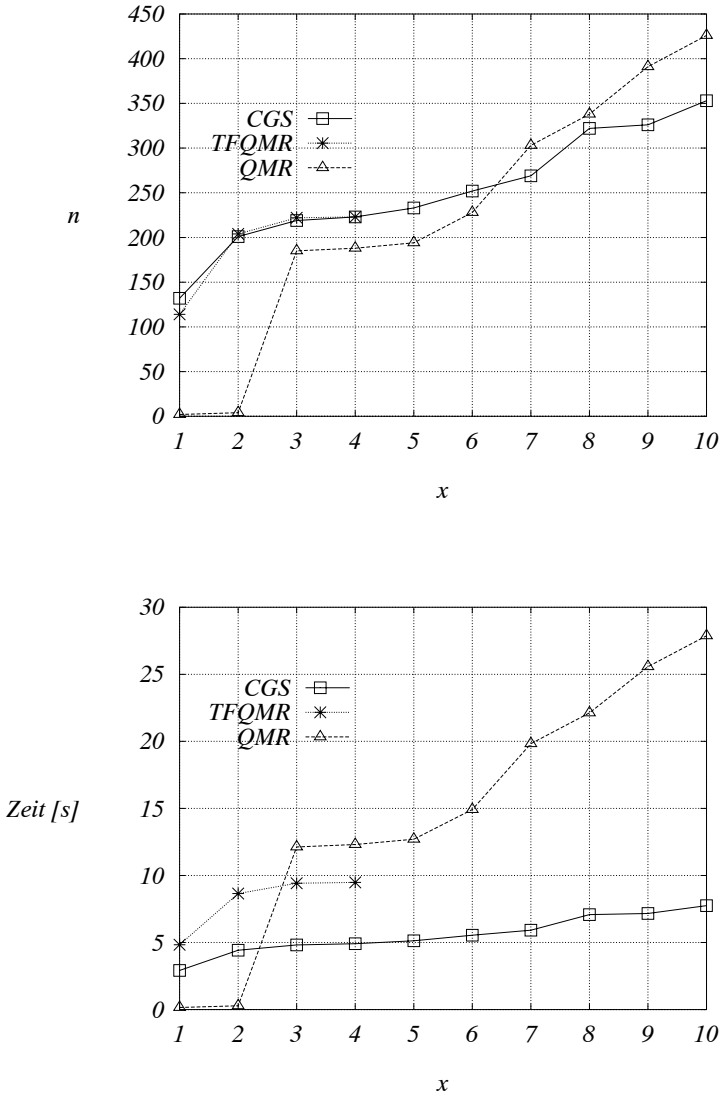


Abbildung 4.8: Für das Beispiel **ORSREG_1** ist im oberen Bild die Iterationsanzahl n und im unteren Bild die Gesamtzeit der drei Verfahren in Abhängigkeit von der Genauigkeit $x = -\log_{10} \epsilon$ dargestellt.

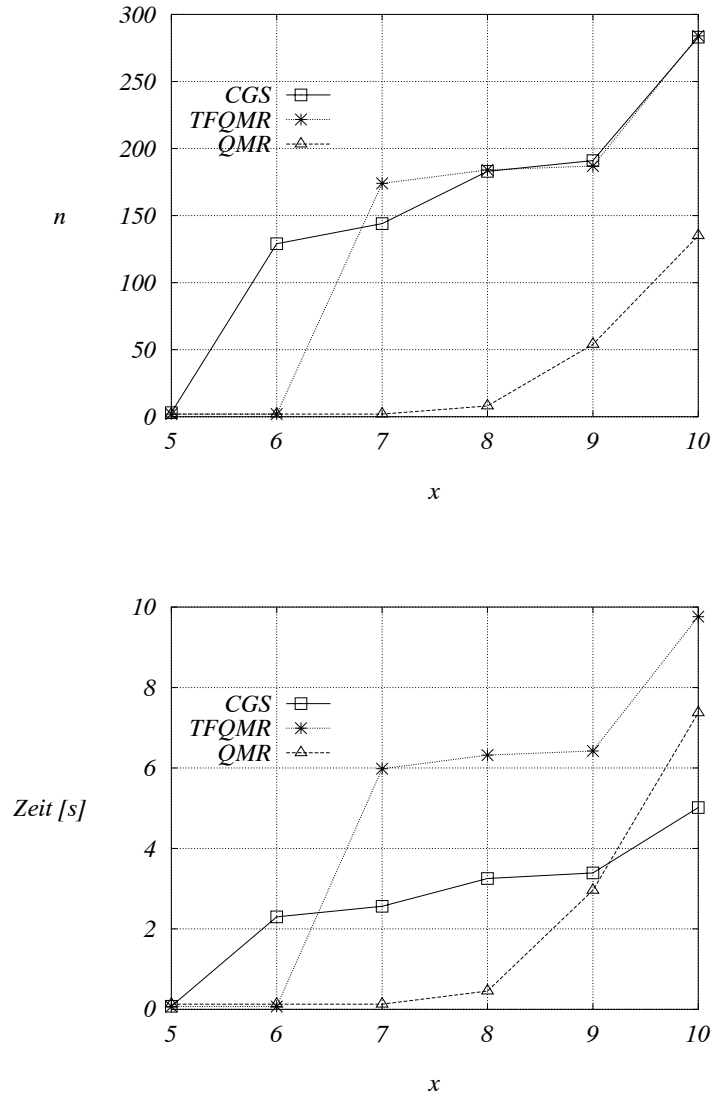


Abbildung 4.9: Für das Beispiel **WATT1** ist im oberen Bild die Iterationsanzahl n und im unteren Bild die Gesamtzeit der drei Verfahren in Abhängigkeit von der Genauigkeit $x = -\log_{10} \epsilon$ dargestellt.

Die erläuterten Beispiele zeigen den Trend, daß QMR bei geringen Genauigkeiten weniger Iterationen als die anderen beiden Verfahren benötigt. Eine Iteration von QMR ist jedoch, wie die Tabelle 4.4 zeigt, im Vergleich zu einer Iteration der anderen Verfahren sehr aufwendig. CGS benötigt pro Iterationsschritt die geringste Zeit. TFQMR ist ungefähr doppelt so zeitaufwendig. Diese Ergebnisse entsprechen der Tatsache, daß TFQMR in jeder Iteration vier statt zwei Matrix–Vektor–Produkte bei CGS berechnet. Die auf die Iterationsanzahl bezogenen Zeiten von QMR liegen etwa um den Faktor 3 höher als die von CGS. Obwohl der Aufwand einer QMR–Iteration gegenüber dem Aufwand der anderen Verfahren groß ist, zeigen die Beispiele, daß QMR nicht grundsätzlich als zu aufwendig verworfen werden

	Zeit/n [ms]		
Beispiel	CGS	TFQMR	QMR
JPWH_991	9,7	19,2	28,8
PORES 2	13,2	25,5	37,8
WEST2021	17,1	32,8	57,2
WATT1	17,8	35,3	54,7
ORSREG_1	22,0	42,4	65,4

Tabelle 4.4: Zeit pro Iterationsschritt der drei Verfahren in Millisekunden.

darf. Die geringere Anzahl von Iterationen kann den Nachteil einer aufwendigeren Berechnung einer Iterierten mehr als aufwiegen. Bei dieser Betrachtung ist zu berücksichtigen, daß hier ausschließlich kleine Gleichungssysteme mit einer geringen Anzahl von Nichtnull–Elementen untersucht worden sind. Bei größeren Problemen wird der Berechnungsanteil der Matrix–Vektor–Produkte den Anteil der übrigen Operationen dominieren. Der Unterschied in den Aufwänden pro Iterationsschritt der drei Verfahren wird dann immer mehr abnehmen.

Kapitel 5

Parallele Ergebnisse

Im vorigen Kapitel sind die drei Iterationsverfahren CGS, QMR und TFQMR auf ihr Konvergenzverhalten hin untersucht worden. Dazu sind Gleichungssysteme mit einer geringen Anzahl von Nichtnull–Elementen verwendet worden, so daß alle Messungen sequentiell durchgeführt werden konnten. In diesem Kapitel liegen die zu untersuchenden Datensätze im Speicherbedarf um bis zu zwei Größenordnungen über denen aus dem vorigen Kapitel. Der Speicherplatz eines einzelnen Knotens reicht nicht mehr zur Aufnahme des gesamten Datensatzes aus. Die Daten werden daher auf unterschiedliche Knoten eines Parallelrechners verteilt. Im Mittelpunkt der Betrachtung steht das Verhalten der drei Verfahren in Abhängigkeit von der benutzten Knotenanzahl, die im folgendem mit p bezeichnet wird.

Mit einem Übergang zu Gleichungssystemen, die einen hohen Besetzungsgrad aufweisen, wächst der Anteil, den die Matrix–Vektor–Produkte am Gesamtaufwand einer Iteration besitzen. Der Aufwand zur Berechnung eines Matrix–Vektor–Produktes mit einer dünnbesetzten Matrix ist proportional zu der Anzahl der Nichtnull–Elemente der Matrix. Der Aufwand der in den Algorithmen enthaltenen Vektoroperationen ist linear in der Ordnung der Matrix. Für kleine Systeme besitzen die Vektoroperationen einen nicht zu vernachlässigenden Anteil am Berechnungsaufwand. Bei Systemen mit hohem Besetzungsgrad wird der Aufwand für die Matrix–Vektor–Produkte dominieren. Da QMR bei gleicher Anzahl von Matrix–Vektor–Produkten im Vergleich zu den anderen beiden Verfahren einen hohen Anteil an Vektoroperationen besitzt, ist bei kleinen Systemen der zeitliche Unterschied von QMR zu CGS bzw. TFQMR groß. Für große Systeme mit hohem Besetzungsgrad ist zu erwarten, daß dieser Unterschied abnimmt und QMR im Zeitverhalten näher an die anderen Verfahren heranrückt. Um einen realitätsnahen Vergleich der drei Verfahren vornehmen zu können, wird in diesem Kapitel das Residuum von TFQMR nicht mehr zu jeder Iterierten bestimmt. Stattdessen wird die aufwendige Residuumsberechnung — und darauf aufbauend die Überprüfung des Abbruchkriteriums — nur in jedem fünfzigsten Iterationsschritt durchgeführt. Nach einigen Bemerkungen zur Datenverteilung werden in diesem Kapitel unterschiedliche Implementierungen von QMR beschrieben. Anschließend werden numerischen Ergebnisse und Zeiten dargestellt.

5.1 Parameter der Datenverteilung

In Abschnitt 3.5 ist das Modell beschrieben, nach dem die Datenverteilung in der vorliegenden Arbeit durchgeführt wird. In diesem Modell ist ein Parameter enthalten, der die Verteilung der Zeilen der Koeffizientenmatrix auf die Knoten des Parallelrechners steuert. Dieser Parameter ξ ist von dem verwendeten Algorithmus und von dem benutzten Rechner abhängig. Da in diesem Kapitel das Residuum von TFQMR nicht in jeder Iteration berechnet werden soll, ist bei den Messungen zur Bestimmung des Parameters ξ die Anzahl der Matrix–Vektor–Produkte in allen drei Algorithmen zu zwei gesetzt worden. (Besteht man auf der Berechnung des TFQMR–Residuums zu jeder Iterierten, ist in Gleichung (3.2) der Wert $s = 4$ einzusetzen.) Im Rahmen der Arbeit sind Messungen zur Bestimmung des Parameters der Datenverteilung für alle drei Algorithmen mit verschiedenen Gleichungssystemen durchgeführt worden. Die so bestimmten Werte von ξ liegen für CGS im Bereich 5–11, für TFQMR im Bereich 7–14 und für QMR im Bereich 30–70. Die unterschiedlichen Werte von ξ für einen Algorithmus sind auf Speicherzugriffsverhalten zurückzuführen. Da der Parameter ξ umgekehrt proportional zu dem Anteil der Matrix–Vektor–Produkte ist und bei QMR dieser Anteil im Vergleich zu den anderen Algorithmen klein ist, sind die Werte von ξ bei QMR höher als die bei CGS bzw. TFQMR. Der Einfluß, den der Wert des Parameters auf die Ausführungszeiten der Programme hat, steigt mit zunehmender Knotenanzahl. Im Bereich $0 \leq \xi \leq 200$ ist jedoch bei allen untersuchten Knotenanzahlen nur ein vernachlässigbarer Einfluß gemessen worden. Der Parameter ist daher für alle Algorithmen einheitlich auf den Wert $\xi = 8$ gesetzt worden.

5.2 Implementierte Varianten von QMR

In diesem Abschnitt werden Varianten von QMR beschrieben. Darunter sollen hier nicht algorithmische Varianten, die etwa basierend auf dem QMR–Ansatz neue Iterationsverfahren definieren, verstanden werden. Der Begriff Variante steht an dieser Stelle für unterschiedliche Implementierungen des Algorithmus QMR, der in den Abbildungen 2.4 – 2.6 vorgestellt wurde.

5.2.1 Speicherung der transponierten Koeffizientenmatrix

Das Verfahren QMR berechnet in jeder Iteration zwei Matrix–Vektor–Produkte, je eins der Form $\boldsymbol{A}x$ und $\boldsymbol{A}^T y$. Die im Kapitel 3 enthaltenen Abbildungen 3.4 und 3.5 zeigen, wie diese Operationen in der verwendeten Speichertechnik sequentiell implementiert werden. Bei dem Produkt der Form $\boldsymbol{A}x$ wird auf die Zeilen der Matrix $\boldsymbol{A}$ zugegriffen, während der Zugriff beim Produkt der Form $\boldsymbol{A}^T y$ spaltenweise erfolgt. Mit dem entwickelten Kommunikationsschema können diese beiden sequentiellen Operationen parallelisiert werden. Eine Ausgangsvariante, die das Verfahren

genauso implementiert, wie es in Kapitel 2 beschrieben ist, speichert ausschließlich die Koeffizientenmatrix des Systems. Eine zweite Variante unterscheidet sich von der Ausgangsvariante durch das zusätzliche Abspeichern der transponierten Koeffizientenmatrix in identischer Datenstruktur, um einen spaltenweisen Zugriff auf die Matrix $\boldsymbol{A}$ zu vermeiden. Die Tabelle 5.1 zeigt die Ergebnisse dieser beiden Varianten im Vergleich. Bei diesen Messungen ist das Abbruchkriterium $\|r^{(n)}\| < 10^{-1}$ verwendet worden, um die Anzahl der benötigten Iterationen und damit die Rechenzeit zu begrenzen. Die Vermeidung des spaltenweisen Zugriffes erzielt unabhängig von der Knotenanzahl einen zeitlichen Gewinn von ungefähr vier Prozent. Dieses Ergebnis korrespondiert zu den Resultaten aus [5], die den Grad der Überlappung von Kommunikation und Berechnung beim spaltenweisen Zugriff gegenüber dem zeilenweisen Zugriff als geringer nachweisen. Da dem ohnehin kleinen Zeitvorteil ein doppelt so hoher Speicherbedarf nachteilig gegenübersteht, wird die Variante, in der die Transponierte der Koeffizientenmatrix explizit abgespeichert wird, im folgenden nicht mehr betrachtet.

ICG2D		Zeit/n [ms]	
		$\boldsymbol{A}, \boldsymbol{A}^T$	$\boldsymbol{A}$
	8	116,9	121,4
	16	73,2	75,8
p	24	59,3	61,8
	32	52,6	55,0
	40	48,8	50,9

Tabelle 5.1: Vergleich der QMR–Varianten, in denen neben $\boldsymbol{A}$ zusätzlich $\boldsymbol{A}^T$ abgespeichert bzw. nicht abgespeichert ist. Dargestellt sind die auf die Iterationsanzahl bezogenen Zeiten in Millisekunden (bei verwendeter Matrix **ICG2D**).

5.2.2 Koppelung der Kommunikation

Die in jeder Iterationsschleife enthaltenen zwei Matrix–Vektor–Produkte der Algorithmen CGS und TFQMR sind jeweils voneinander abhängig. Dies bedeutet, daß die Ergebnisse des ersten Matrix–Vektor–Produktes in die Berechnung desjenigen Vektors einfließen, mit dem das zweite Matrix–Vektor–Produkt durchgeführt wird. Die beiden Matrix–Vektor–Produkte müssen deshalb nacheinander ausgeführt werden. Bei dem Verfahren QMR ist dies nicht der Fall. Die zwei Matrix–Vektor–Produkte von QMR sind voneinander unabhängig und können daher simultan berechnet werden. Bei einem parallelen Matrix–Vektor–Produkt dient das Kommunikationsschema zum Austauschen von Nachrichten zwischen den Knoten des Parallelrechners. Wird bei einem Matrix–Vektor–Produkt der Form $\boldsymbol{A}x$ eine Nachricht

zwischen den Knoten i und j ausgetauscht, müssen die gleichen Knoten nicht unbedingt bei einem Matrix–Vektor–Produkt der Form $\boldsymbol{A}^T x$ miteinander kommunizieren. Ist dies jedoch der Fall, so werden in einer QMR–Variante die zu versendenden Daten zwischengespeichert und in einer einzigen Nachricht verschickt. Die Kommunikation der beiden Matrix–Vektor–Produkte wird auf diese Weise gekoppelt durchgeführt. Die im vorherigen Abschnitt beschriebene Ausgangsvariante, in der die Kommunikation der Matrix–Vektor–Produkte ungekoppelt implementiert ist, wird in Tabelle 5.2 mit der gekoppelten Variante verglichen. Als Abbruchkriterium ist $\|r^{(n)}\| < 10^{-5}$ verwendet worden.

ISR		ungekoppeltes QMR			gekoppeltes QMR		
		n	Zeit	Zeit/n	n	Zeit	t/n
p	8	1242	511,9	412	1219	499,2	410
	16	1228	283,5	231	1229	282,3	230
	24	1219	202,0	166	1230	201,0	163
	32	1242	164,8	133	1221	159,2	130
	40	1234	140,2	114	1242	138,0	111
	48	1226	122,2	100	1260	121,5	96
	56	1228	110,4	90	1234	107,0	87
	64	1223	102,8	83	1224	97,8	80

Tabelle 5.2: Vergleich der QMR–Varianten, in denen die Kommunikation der zwei Matrix–Vektor–Produkte ungekoppelt bzw. gekoppelt durchgeführt wird. Dargestellt sind die Anzahl n der Iterationen, die dazu benötigte Zeit in Sekunden und der Quotient dieser beiden Größen in Millisekunden (bei verwendeter Matrix **ISR**).

Die Ergebnisse belegen, daß die zur Erfüllung des Abbruchkriteriums benötigte Anzahl von Iterationen eine Funktion der verwendeten Knotenanzahl ist. Da in der Implementierung nicht–blockierende Kommunikation stattfindet, können Nachrichtenübertragung und Berechnung in einem hohen Maße überlappt werden. Da der Nachrichtenaustausch jedoch nicht synchronisiert wird, ist die Reihenfolge der durchgeführten Operationen nicht festgelegt. Daher können Rundungsfehler eine unterschiedliche Iterationsanzahl verursachen. Dieser Effekt ist nicht spezifisch für den Vergleich der beiden QMR–Varianten. Er tritt an dieser Stelle erstmals auf, gilt aber für die gesamte restliche Arbeit. Der Tabelle kann nicht entnommen werden, daß eine QMR–Variante in bezug auf die Iterationsanzahlen stabiler ist als die andere. Es gilt jedoch die Aussage, daß die gekoppelte Variante sowohl in der absolut benötigten Zeit als auch in der auf die Iterationen bezogenen Zeit für alle Knotenanzahlen günstiger ist. Die Tabelle zeigt den Trend, daß die gekoppelte Variante der ungekoppelten mit steigender Knotenanzahl zunehmend überlegen ist. Bei $p = 64$ Knoten beträgt der Unterschied der beiden Varianten etwa vier Prozent.

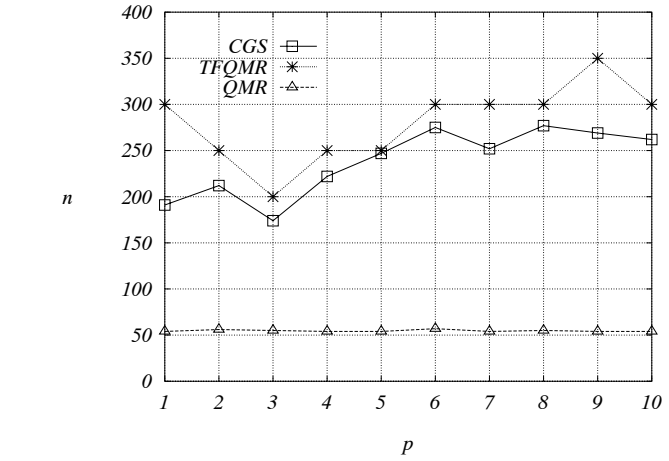
Größere zeitliche Gewinne werden erwartet, wenn die Knotenanzahl erhöht wird und der Kommunikationsprozessor in der nächsten Version des Betriebssystems aktiviert wird. Da die gekoppelte QMR–Variante günstiger ist, wird im folgenden ausschließlich diese Variante benutzt, ohne dabei den Hinweis auf die gekoppelte Kommunikation zu geben.

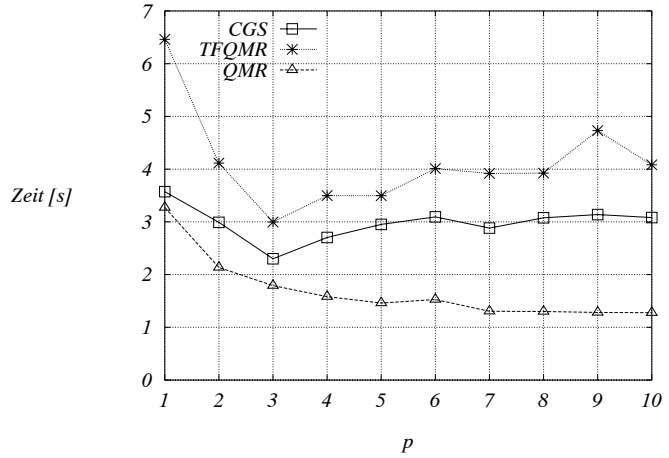
5.3 Performance

Sowohl Zeiten als auch Iterationsanzahlen, die die drei Verfahren zur Erfüllung eines vorgegebenen Abbruchkriteriums benötigen, werden in diesem Abschnitt anhand einiger Beispiele verglichen. Beginnend bei einem Gleichungssystem mit 11 360 Nichtnull–Elementen werden nacheinander Systeme betrachtet, deren Datenvolumen jeweils zunimmt. Das größte an dieser Stelle vorgestellte Beispiel besitzt 3 856 386 Nichtnull–Elemente. Die großen Systeme stammen aus Anwendungen des Forschungszentrums Jülich und sind symmetrisch positiv definit. Zur Beurteilung der parallelen Ergebnisse wird der Begriff Speedup abweichend von der konventionellen Bedeutung hier in der folgenden Art und Weise verwendet. Zur Berechnung des Speedup werden ausschließlich die auf die Iterationsanzahlen bezogenen Zeiten verwendet, wobei der Speedup der kleinsten betrachteten Knotenanzahl als optimal angenommen wird. Alle anderen Werte des Speedup werden aus dieser Festlegung berechnet. Insbesondere bedeutet dies, daß beim Begriff Speedup hier nicht die Zeit einer schnellsten sequentiellen Programmausführung berücksichtigt wird. Diese Vorgehensweise ist dadurch motiviert, daß die großen Systeme nicht auf einem einzelnen Knoten gespeichert werden können.

5.3.1 Harwell–Boeing Sparse Matrix Collection

Aus einer Standardsammlung dünnbesetzter Matrizen wird die Matrix **WATT1** entnommen. Die mit diesem Beispiel gewonnenen Ergebnisse werden in Abbildung 5.1 betrachtet, wobei das Abbruchkriterium $\|r^{(n)}\| < 10^{-9}$ verwendet wird. Das obere Diagramm zeigt für QMR eine von der Knotenanzahl unabhängige Iterationsanzahl, während die Iterationsanzahlen der anderen Verfahren mit der Knotenanzahl variieren. Im Vergleich zu den anderen Verfahren benötigt QMR bei allen Knotenanzahlen weniger Iterationen. Die Iterationsanzahlen von CGS und TFQMR sind in etwa vergleichbar. Bei der Kurve von TFQMR ist die für dieses Kapitel getroffene Vereinbarung zu beachten, daß das Residuum nur alle fünfzig Schritte berechnet wird. Im mittleren Diagramm wird deutlich, daß bei allen Verfahren die Gesamtzeit im Bereich $1 \leq p \leq 3$ mit zunehmender Knotenanzahl abnimmt. Aufgrund der Zunahme der Iterationen steigt die Zeit bei CGS und TFQMR für $p \geq 4$. Dagegen erzielt eine weitere Erhöhung der Knotenanzahl bei QMR im gesamten betrachteten Bereich einen Zeitgewinn. Für alle Knotenanzahlen gilt, daß TFQMR den größten, CGS einen mittleren und QMR den geringsten Zeitbedarf besitzt. Im unteren Bild





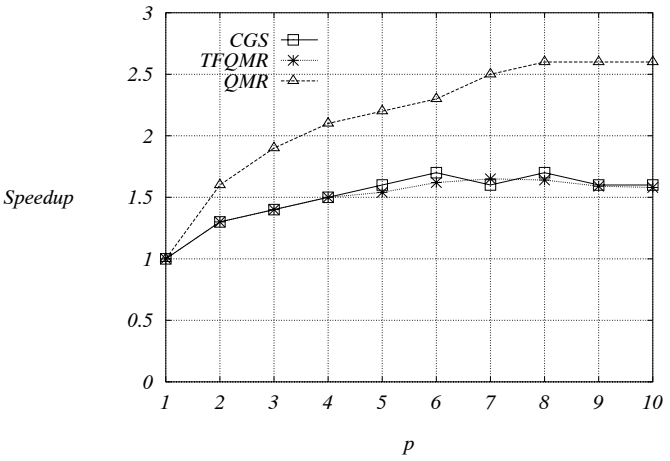


Abbildung 5.1: Ergebnisse der drei Verfahren beim Beispiel **WATT1**. Dargestellt sind die Iterationsanzahl n, die Gesamtzeit in Sekunden und der Speedup.

ist der Speedup der drei Verfahren dargestellt. CGS und TFQMR verhalten sich hier ähnlich. Der Speedup von QMR liegt für alle Knotenanzahlen über dem der beiden anderen Verfahren. Für einen Einsatz von einer hohen Knotenanzahl ist dieses Beispiel wegen des geringen Datenvolumens jedoch nicht geeignet.

5.3.2 Matrix aus einem impliziten FE–Modell

Die Lösung des Gleichungssystems, das durch die Matrix **IMPLFE** mit 661 010 Nichtnull–Elementen charakterisiert wird, stellt ein schwieriges numerisches Problem dar. Wie einige direkte Verfahren scheitern die iterativen Methoden CGS und TFQMR bei dieser Problemstellung. Dagegen liefert QMR bei der Verwendung des Abbruchkriteriums $\|r^{(n)}\| < 10^{-7}$ in ungefähr zehntausend Iterationen eine Approximation der Lösung. In der Abbildung 5.2 sind die Ergebnisse von QMR mit diesem Beispiel enthalten. Dabei wurde das Abbruchkriterium $\|r^{(n)}\| < 10^{-4}$ benutzt, um die Rechenzeit zu begrenzen. Die Eigenschaften der Parallelisierung werden durch diese Vorgehensweise nicht beeinflußt. Das obere Diagramm zeigt, daß die benötigte Iterationsanzahl von der Knotenanzahl abhängt. Der Unterschied in den Iterationsanzahlen ist aber moderat. Das mittlere Diagramm belegt, daß die Zeit zur Erfüllung des Abbruchkriteriums im gesamten betrachteten Bereich mit steigender Knotenanzahl abnimmt. Im unteren Diagramm ist der Speedup aufgetragen, der sich auf eine einzelne Iteration beziehen. Der Verlauf des Speedup zeigt eine gute Parallelisierung bei niedrigen Knotenanzahlen. Mit zunehmender Knotenanzahl steigt der Speedup bis zu $p = 56$ Knoten. Bei den letzten beiden betrachteten Knotenanzahlen bleibt der Speedup gleich.

5.3.3 Matrix aus der Umwelttechnik

Im folgenden werden die Ergebnisse der Matrix **ICG3D** betrachtet, die aus 1 242 814 Nichtnull–Elementen besteht. Diese Resultate sind in der Abbildung 5.3 dargestellt. Als Abbruchkriterium wurde $\|r^{(n)}\| < 10^{-4}$ verwendet. Bei dem Verfahren TFQMR ist dieses Kriterium innerhalb einer vorgegebenen maximalen Iterationsanzahl von 5 000 mit $p = 32$ Knoten nicht erfüllt worden. Für diesen speziellen Fall sind daher die Ergebnisse in den Diagrammen für TFQMR nicht angegeben. Bei den Iterationsanzahlen läßt sich bei diesem Beispiel wiederum beobachten, daß die mit der Knotenanzahl auftretenden Abweichungen bei QMR geringer sind als bei CGS. Weiterhin zeigt das Beispiel einen Unterschied der drei Verfahren in bezug auf die gesamte zur Erfüllung des Abbruchkriteriums benötigte Zeit. Mit zunehmender Knotenanzahl sinkt diese Zeit bei CGS bis zu $p = 56$ Knoten. Bei $p = 64$ Knoten verändert sich der Zeitbedarf von CGS nicht mehr. Bei dem Verfahren TFQMR steigt für $p = 48$ und $p = 64$ jeweils die Gesamtzeit. Bei QMR nimmt diese Zeit im gesamten betrachteten Bereich mit zunehmender

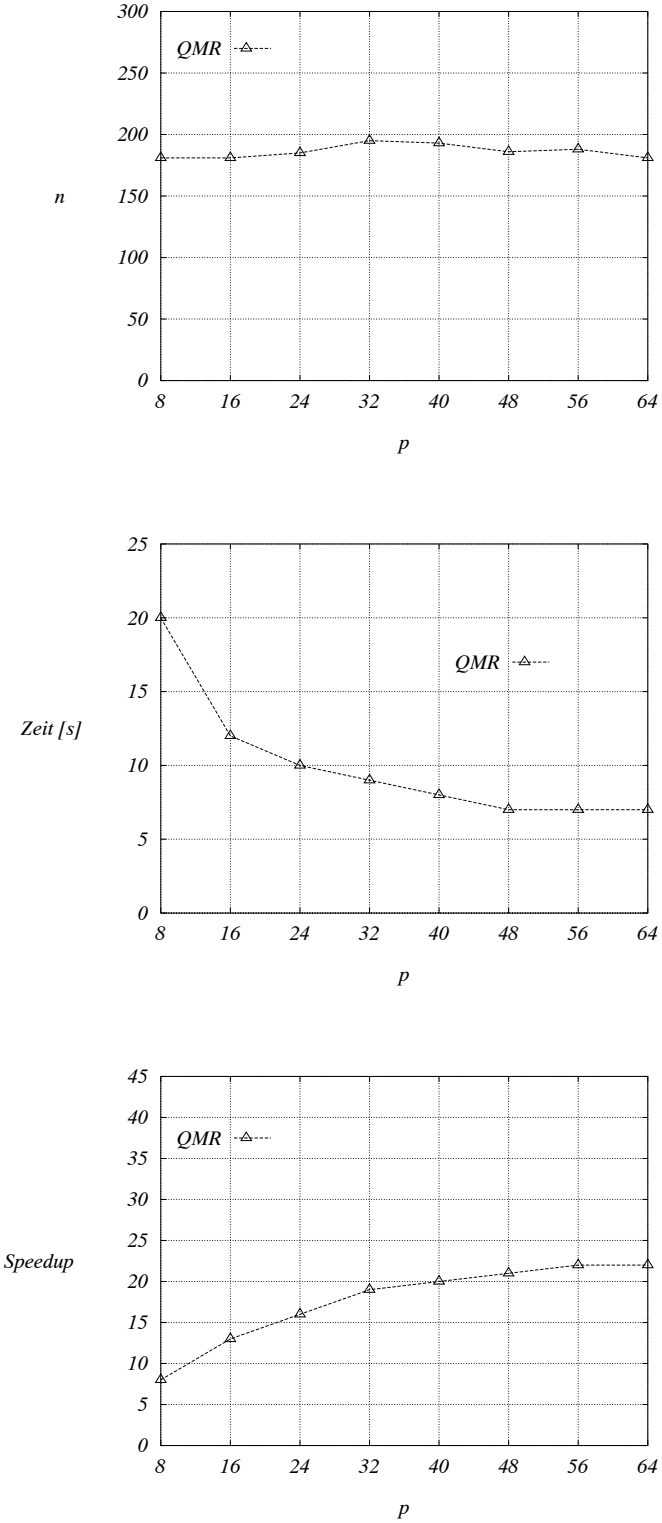
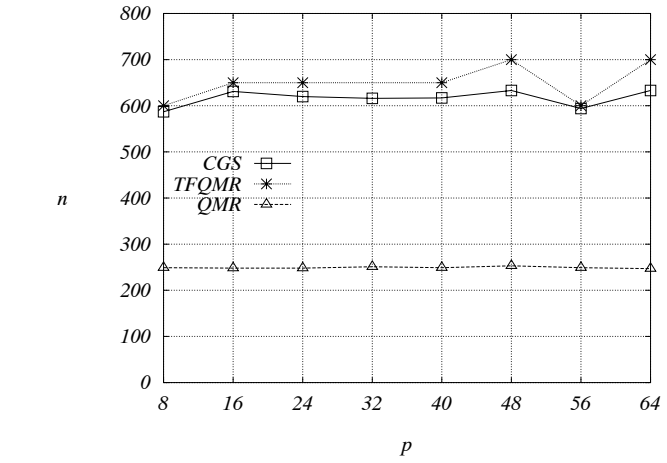
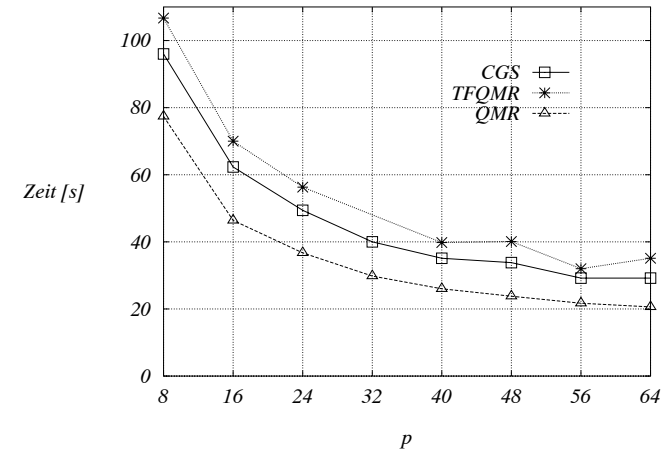


Abbildung 5.2: Ergebnisse des Verfahrens QMR beim Beispiel **IMPLFE**. Dargestellt sind die Iterationsanzahl n, die Gesamtzeit in Sekunden und der Speedup.





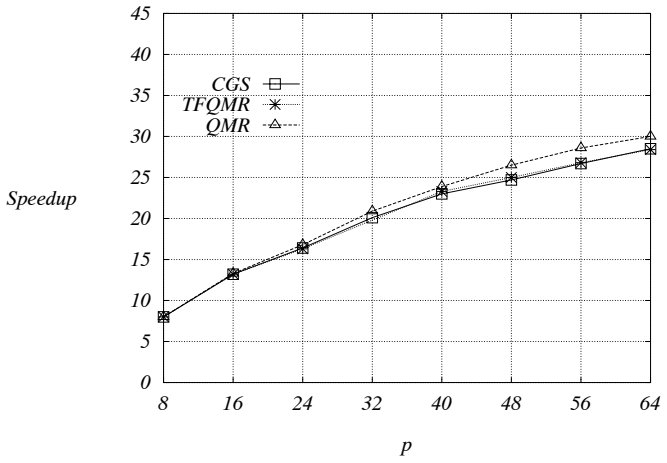


Abbildung 5.3: Ergebnisse der drei Verfahren beim Beispiel **ICG3D**. Dargestellt sind die Iterationsanzahl n, die Gesamtzeit in Sekunden und der Speedup.

Knotenanzahl ab. Aufgrund der niedrigen Iterationsanzahlen benötigt QMR bei diesem Beispiel für alle Knotenanzahlen die geringste Zeit aller drei Verfahren. Allen Verfahren ist gemeinsam, daß die auf die Iterationen bezogenen Zeiten mit zunehmender Knotenanzahl abnehmen. Dabei benötigen CGS und TFQMR in etwa die gleiche Zeit pro Iteration. QMR liegt um den Faktor 1,8 über diesen Zeiten. Der Zeitgewinn, der durch eine Erhöhung der Knotenanzahlen erzielt wird, ist bei allen Verfahren so groß, daß der Speedup bis zu der größten betrachteten Knotenanzahl steigt. Das untere Diagramm zeigt ebenfalls, daß die Verfahren CGS und TFQMR einen vergleichbaren Speedup erreichen. Die Werte des Speedup von QMR sind bei den höchsten betrachteten Knotenanzahlen etwas größer als die der beiden anderen Verfahren.

5.3.4 Matrix aus der Strukturmechanik

Die Matrix **ISR** besitzt mit 3 856 386 die meisten Nichtnull–Elemente aller in dieser Arbeit untersuchten Matrizen. Die Ergebnisse dieser Matrix sind in der Abbildung 5.4 enthalten, wobei als Abbruchkriterium $\|r^{(n)}\| < 10^{-5}$ verwendet worden ist. Die Ergebnisse des Verfahrens TFQMR sind nicht abgebildet, da das Abbruchkriterium innerhalb einer vorgegebenen maximalen Iterationsanzahl von 5 000 nicht erreicht worden ist. Das Beispiel **ISR** zeigt ein ähnliches Verhalten wie das Beispiel **ICG3D**. Die Iterationsanzahlen variieren bei CGS stärker als bei QMR. Weiterhin nimmt die gesamte benötigte Zeit — bis auf eine Ausnahme bei CGS für $p = 48$ Knoten — bei den Verfahren CGS und QMR mit zunehmender Knotenanzahl ab. Diese Zeiten liegen bei QMR unter denen von CGS. Außerdem nehmen die auf die Iterationsanzahlen bezogenen Zeiten bei den beiden Verfahren bei einer Erhöhung der verwendeten Knotenanzahl ab. Die Zeit, die QMR pro Iteration benötigt, ist größer als diejenige von CGS. Der Faktor, der zwischen den Zeiten von CGS und QMR liegt, beträgt in dem hier betrachteten Beispiel **ISR** ungefähr 1,3 (gegenüber 1,8 im vorherigen Beispiel **ICG3D**). Dieses Ergebnis entspricht dem zu Beginn des Kapitels ausgeführten Gedankengang, daß der Unterschied zwischen dem Aufwand einer QMR–Iteration und dem Iterationsaufwand der anderen Verfahren für Systeme mit hohem Besetzungsgrad mehr und mehr abnimmt. Bei dem Beispiel **ISR** sind die Werte des Speedup in den Verfahren CGS und QMR für alle Knotenanzahlen hoch und zeigen eine gute Parallelisierung der untersuchten Algorithmen in der verwendeten Implementierung.

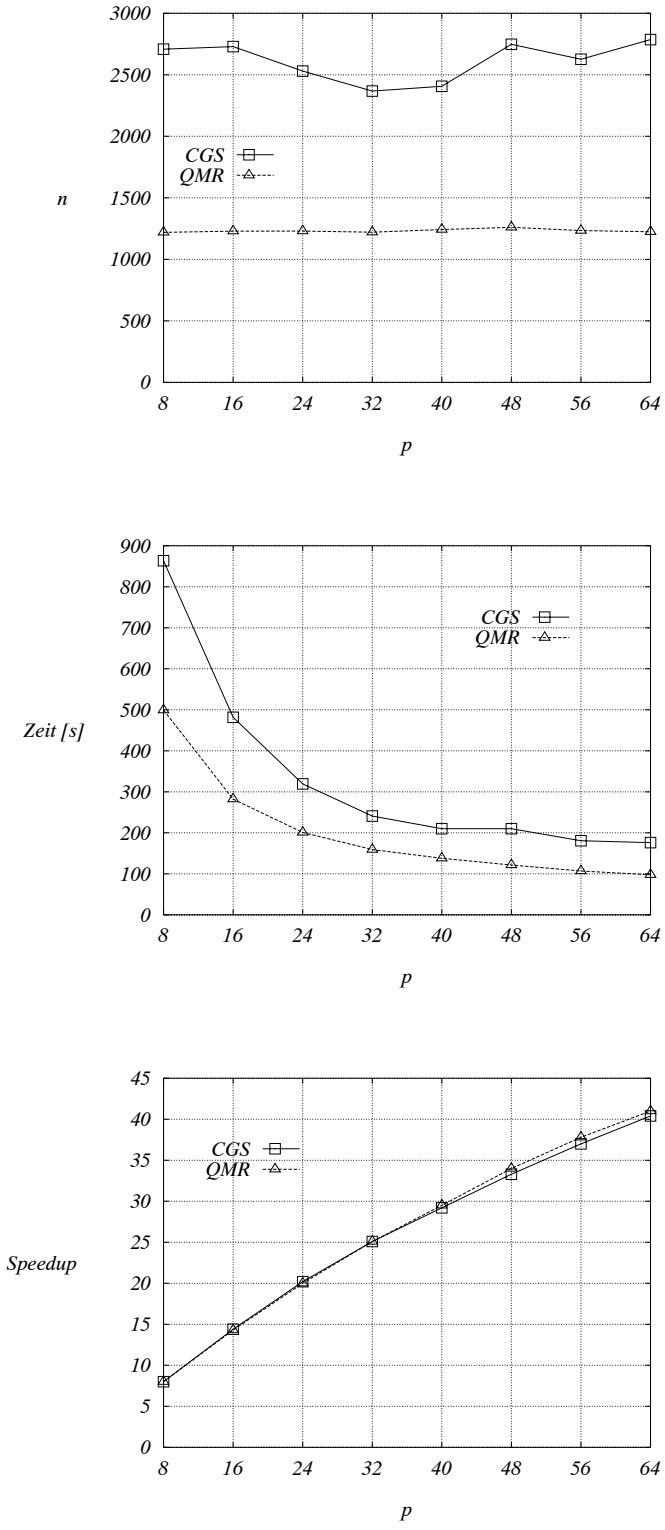


Abbildung 5.4: Ergebnisse der Verfahren CGS und QMR beim Beispiel **ISR**. Dargestellt sind die Iterationsanzahl n, die Gesamtzeit in Sekunden und der Speed-up.

Kapitel 6

Zusammenfassung und Ausblick

Diese Arbeit stellt zwei iterative Methoden zur Lösung von linearen Gleichungssystemen mit regulärer unsymmetrischer Koeffizientenmatrix vor. Beide Verfahren basieren auf dem Ansatz der quasi–minimalen Residuen. In diesem Ansatz wird eine bekannte Vorschrift zur Definition eines Iterationsverfahrens modifiziert: Anstelle der optimalen Strategie, in jedem Iterationsschritt die Residuumsnorm zu minimieren, wird die Minimierung lediglich auf einen Faktor des Residuums bezogen. Der entscheidende Punkt dieses Ansatzes ist die Tatsache, daß der Aufwand des so entstandenen Minimierungsproblems gering ist und die Verfahren bezüglich ihres Konvergenzverhaltens zu anderen Methoden konkurrenzfähig bleiben.

Das Verfahren QMR (*Quasi–Minimal Residual*) benutzt den klassischen unsymmetrischen Lanczos–Algorithmus, der als effizientes Hilfsmittel zur Erzeugung von Basisvektoren zweier unterschiedlicher Krylov–Teilräume angesehen werden kann. Der Lanczos–Algorithmus wird in QMR mit dem Ansatz der quasi–minimalen Residuen kombiniert. Der resultierende Algorithmus enthält in jeder Iteration sowohl ein Matrix–Vektor–Produkt mit der Koeffizientenmatrix des zu lösenden Gleichungssystems als auch ein Matrix–Vektor–Produkt mit deren Transponierter. Die Methode TFQMR (*Transpose–Free Quasi–Minimal Residual*) verwendet zum Aufspannen der Krylov–Teilräume das Verfahren CGS (*Conjugate Gradient Squared*) und fügt diesem den Ansatz der quasi–minimalen Residuen hinzu. CGS wird in dieser Arbeit als vergleichendes Verfahren mit in die Untersuchungen einbezogen. TFQMR und CGS berechnen in jeder Iterationsschleife zwei Matrix–Vektor–Produkte mit der Koeffizientenmatrix, jedoch keine Matrix–Vektor–Produkte mit der Transponierten. Die Verfahren TFQMR und CGS werden aus diesem Grunde als transpositionsfrei bezeichnet. Während die beiden Matrix–Vektor–Produkte in QMR unabhängig voneinander berechnet werden können, sind die Matrix–Vektor–Produkte der transpositionsfreien Methoden voneinander abhängig.

Bei der Beurteilung eines Iterationsverfahrens spielt das Konvergenzverhalten eine wichtige Rolle. Darunter wird hier die Abhängigkeit der Residuumsnorm von der Iterationsanzahl verstanden. Das Konvergenzverhalten wird u.a. stark durch das vorgegebene Gleichungssystem beeinflußt. Bei den Untersuchungen mit Systemen

aus Anwendungen des Forschungszentrums Jülich und einer Standardsammlung für dünnbesetzte Matrizen konnten die folgenden unterschiedlichen Trends für die drei Iterationsverfahren festgestellt werden. Bei CGS oszilliert die Norm des Residuums mit der Iterationsanzahl sehr stark. Diese Eigenschaft kann Probleme bei einem Abbruchkriterium bereiten, welches ein Verfahren beendet, sobald die Residuumsnorm erstmalig unter eine vorgegebene Schranke fällt. Denn die Abbruchbedingung wird unter Umständen durch ein stark nach unten abweichendes Residuum erfüllt. Ein Abbruchkriterium, das eine starke Oszillation der Residuen berücksichtigt, wird ein Verfahren erst dann stoppen, wenn beispielsweise mehrere Residuen eine vorgegebene Bedingung erfüllen. In der vorliegenden Arbeit wird jedoch für das Verfahren CGS das erstmalige Erfüllen einer Abbruchbedingung verwendet. Die Iterationsverfahren QMR und TFQMR zeigen einen glatten, plateauförmigen Verlauf der Residuumsnorm, der sich von dem oszillierenden Konvergenzverhalten von CGS deutlich unterscheidet. Darüberhinaus scheint QMR für geringe Genauigkeiten erheblich weniger Iterationsanzahlen zur Erfüllung eines Abbruchkriteriums zu benötigten, als dies bei den beiden anderen Verfahren der Fall ist. Für die betrachteten Verfahren ist jedoch nicht bekannt, welchen Einfluß Systemeigenschaften wie beispielsweise die Kondition der Koeffizientenmatrix auf das Konvergenzverhalten besitzen. Daher konnten die experimentellen Ergebnisse nicht mit Eigenschaften der Matrix in Zusammenhang gebracht werden. Die in dieser Arbeit zusammengetragenen Resultate bezüglich des Konvergenzverhaltens besitzen deshalb keine generelle Gültigkeit, sondern stellen nur für die speziell betrachteten Gleichungssysteme gesicherte Erkenntnisse dar.

Die Aspekte, die bei der Parallelisierung der Algorithmen auftreten, werden durch die des Konvergenzverhaltens überlagert. In der parallelen Implementierung variieren die Iterationsanzahlen mit der verwendeten Knotenanzahl des Parallelrechners. Da die Kommunikation der Knoten untereinander nicht synchronisiert wird, kann sich die Reihenfolge von arithmetischen Operationen ändern und so Rundungsfehler verursachen. Aufgrund des oszillierenden Verhaltens der Residuumsnorm schwanken die Iterationsanzahlen bei CGS stark. Bei QMR und TFQMR ist der Einfluß der Knotenanzahl auf die Iterationsanzahl geringer, da die Residuumsnorm in diesen Verfahren glatt verläuft. Dies ist ein wesentlicher Vorteil der Verfahren QMR und TFQMR. Denn ein Gewinn bei der gesamten zur Approximation der Lösung benötigten Zeit, der bei einer Erhöhung der Knotenanzahl aufgrund der Parallelisierung erzielt wird, kann nicht durch eine unter Umständen zunehmende Iterationsanzahl verlorengehen. Betrachtet man eine einzelne Iteration so wird die Parallelisierung aller drei Verfahren als gleichermaßen gut nachgewiesen, wobei das Verfahren QMR einen leichten Vorteil besitzt. Diese Aussage setzt die Tatsache voraus, daß ein sehr kleines Gleichungssystem nicht sinnvoll mit einer sehr großen Anzahl von Knoten gelöst werden kann.

Für das Verfahren QMR wird gezeigt, daß die simultane Berechnung der zwei in einer Iteration enthaltenen Matrix–Vektor–Produkte auf dem verwendeten Parallelrechner Intel Paragon XP/S 10 von Vorteil ist. Der durch die Koppelung der Kommunikation bedingte Zeitgewinn ist dabei von der Knotenanzahl abhängig und

ist um so größer, je mehr Knoten verwendet werden. QMR erscheint daher besonders für hohe Knotenanzahlen interessant. Da Fehler in der benutzten Version des Betriebssystems eine Verwendung einer hohen Knotenanzahl nicht zuließen, konnte dieser Effekt nur im Ansatz nachgewiesen werden. Zukünftige Untersuchungen werden zeigen müssen, ob QMR durch die Berechnung der zwei unabhängigen Matrix–Vektor–Produkte mit gekoppelter Kommunikation bei hohen Knotenanzahlen einen wesentlichen Vorteil gegenüber solchen Verfahren erzielen kann, die zwei voneinander abhängige Matrix–Vektor–Produkte enthalten. In der Praxis werden iterative Verfahren häufig in Verbindung mit einer Vorkonditionierung durchgeführt. Es wäre daher weiterhin zu untersuchen, welchen Einfluß die Vorkonditionierung auf das Verfahren QMR besitzt und wie die Vorkonditionierungsschritte parallelisiert werden können. Möglicherweise könnte das Verfahren so umstrukturiert werden, daß globale Synchronisationspunkte zusammengefaßt, vielleicht sogar eingespart werden können.

Literaturverzeichnis

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison–Wesley, Reading, 1987.

[2] R. Barrett, M. Berry, T. 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: Building Blocks for Iterative Methods.* SIAM, Philadelphia, 1993.

[3] A. Basermann. Conjugate Gradients Parallelized on the Hypercube. *International Journal of Modern Physics C*, 4(6):1295–1306, 1993.

[4] A. Basermann. Datenverteilungs– und Kommunikationsmodelle für parallele CG–Verfahren zur Lösung von Gleichungssystemen mit dünnbesetzter Koeffizientenmatrix. Aachener Informatik–Berichte 93–7, Graduiertenkolleg Informatik und Technik, Fachgruppe Informatik der RWTH Aachen, Aachen, 1993.

[5] A. Basermann. Parallelizing Iterative Solvers for Sparse Systems of Equations and Eigenproblems on Distributed Memory Machines. Internal Report KFA–ZAM–IB–9411, Research Centre Jülich, Jülich, June 1994.

[6] R. Beauwens and P. de Groen, editors. *Iterative Methods in Linear Algebra.* North–Holland, Amsterdam, 1992. Proceedings of the IMACS International Symposium on Iterative Methods, Brussels, April 1991.

[7] M. Ben–Ari. *Grundlagen der Parallel–Programmierung.* Hanser, München, 1984.

[8] R. Berrendorf, H. C. Burg, U. Detert, R. Esser, M. Gerndt, and R. Knecht. Intel Paragon XP/S — Architecture, Software Environment, and Performance. Internal Report KFA–ZAM–IB–9409, Research Centre Jülich, Jülich, May 1994.

[9] C. Brezinski and H. Sadok. Avoiding Breakdown in the CGS Algorithm. *Numer. Algorithms*, 1:199–206, 1991.

[10] T. F. Chan, L. de Pillis, and H. A. Van der Vorst. A Transpose–Free Squared Lanczos Algorithm and Application to Solving Nonsymmetric Linear Systems. Technical Report CAM 91–17, Department of Mathematics, University of California, Los Angeles, 1991.

[11] R. D. da Cunha and T. Hopkins. PIM 1.0: The Parallel Iterative Methods Package for Systems of Linear Equations User's Guide. Technical Report, University of Kent, Canterbury, January 1994.

[12] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[13] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse Matrix Test Problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989.

[14] I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' Guide for the Harwell–Boeing Sparse Matrix Collection, Release I. Technical Report TR/PA/92/86, CERFACS, Toulouse Cedex, October 1992.

[15] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors*, volume 1. Prentice–Hall, Englewood Cliffs, 1988.

[16] R. W. Freund. A Transpose–Free Quasi–Minimal Residual Algorithm for Non–Hermitian Linear Systems. RIACS Technical Report 91.18, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, September 1991.

[17] R. W. Freund. Conjugate Gradient–Type Methods for Linear Systems with Complex Symmetric Coefficient Matrices. *SIAM Journal on Scientific and Statistical Computing*, 13(1):425–448, 1992.

[18] R. W. Freund. Quasi–Kernel Polynomials and Convergence Results for Quasi–Minimal Residual Iterations. In D. Braess and L. L. Shumaker, editors, *Numerical Methods of Approximation Theory*, volume 9, pages 77–95. Birkhäuser, Basel, 1992.

[19] R. W. Freund. A Transpose–Free Quasi–Minimal Residual Algorithm for Non–Hermitian Linear Systems. *SIAM Journal on Scientific Computing*, 14(2):470–482, 1993.

[20] R. W. Freund. Block Quasi–Minimal Residual Iterations for Non–Hermitian Linear Systems. In T. Manteuffel and S. McCormick, editors, *Proceedings of the Colorado Conference on Iterative Methods*, volume 2, Breckenridge, April 1994. University of Colorado.

[21] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative Solution of Linear Systems. Numerical Analysis Project, Manuscript NA–91–05, Computer Science Department, Stanford University, Stanford, November 1991.

[22] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative Solution of Linear Systems. RIACS Technical Report 91.21, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, October 1991.

[23] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Recent Advances in Lanczos–Based Iterative Methods for Nonsymmetric Linear Systems. RIACS Technical Report 92.02, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, January 1992.

[24] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An Implementation of the Look–Ahead Lanczos Algorithm for Non–Hermitian Matrices. *SIAM Journal on Scientific Computing*, 14(1):137–158, 1993.

[25] R. W. Freund and N. M. Nachtigal. QMR: A Quasi–Minimal Residual Method for Non–Hermitian Linear Systems. RIACS Technical Report 90.51, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, December 1990.

[26] R. W. Freund and N. M. Nachtigal. QMR: A Quasi–Minimal Residual Method for Non–Hermitian Linear Systems. *Numerische Mathematik*, 60(3):315–339, 1991.

[27] R. W. Freund and N. M. Nachtigal. An Implementation of the QMR Method Based on Coupled Two–Term Recurrences. RIACS Technical Report 92.15, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, June 1992.

[28] R. W. Freund and N. M. Nachtigal. Implementation Details of the Coupled QMR Algorithm. RIACS Technical Report 92.19, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, October 1992.

[29] R. W. Freund and N. M. Nachtigal. QMR: A Quasi–Minimal Residual Method for Non–Hermitian Linear Systems. In Beauwens and de Groen [6], pages 151–154.

[30] R. W. Freund and N. M. Nachtigal. An Implementation of the QMR Method Based on Coupled Two–Term Recurrences. *SIAM Journal on Scientific Computing*, 15(2):313–337, 1994.

[31] R. W. Freund and T. Szeto. A Quasi–Minimal Residual Squared Algorithm for Non–Hermitian Linear Systems. RIACS Technical Report 91.26, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, 1991.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, second edition, 1989.

[33] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison–Wesley, Reading, 1989.

[34] W. Gropp and B. Smith. Simplified Linear Equation Solvers Users Manual. Technical Report ANL–93/8–REV 1, Argonne National Laboratory, Argonne, June 1993.

[35] M. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[36] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2.* Adam Hilger, Bristol, second edition, 1988.

[37] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw–Hill, New York, 1993.

[38] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw–Hill Book Company International editions, New York, 1984.

[39] R. N. Ibbett and N. P. Topham. *Architecture of High Performance Computers*, volume 2. Macmillan, Houndmills, 1989.

[40] Intel Supercomputing Systems Division, Beaverton, Oregon. *Paragon User's Guide*, October 1993. Order Number: 312489–002.

[41] Intel Supercomputing Systems Division, Beaverton, Oregon. *Paragon Fortran Compiler User's Guide*, March 1994. Order Number: 312491–002.

[42] Kuck & Associates, Inc., Champaign. *CLASSPACK: Basic Math Library User's Guide*, December 1992. Release 1.3.

[43] C. Lanczos. Solutions of Systems of Linear Equations by Minimized Iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.

[44] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes.* Morgan Kaufmann, San Mateo, 1992.

[45] J. G. Lewis, D. G. Payne, and R. A. van de Geijn. Matrix–Vector Multiplication and Conjugate Gradient Algorithms on Distributed Memory Computers. In *Proceedings of the Intel Supercomputer User's Group, 1993 Annual North America User's Conference*, pages 193–201, St. Louis, October 1993.

[46] T. Manteuffel and S. McCormick, editors. *Proceedings of the Colorado Conference on Iterative Methods*, volume 1 & 2, Breckenridge, April 1994. University of Colorado.

[47] N. M. Nachtigal. A Look–Ahead Variant of the Lanczos Algorithm and its Application to the Quasi–Minimal Residual Method for Non–Hermitian Linear Systems. RIACS Technical Report 91.19, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, August 1991.

[48] N. M. Nachtigal. A Look–Ahead Variant of TFQMR. In T. Manteuffel and S. McCormick, editors, *Proceedings of the Colorado Conference on Iterative Methods*, volume 2, Breckenridge, April 1994. University of Colorado.

[49] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, London, 1988.

[50] B. N. Parlett, D. R. Taylor, and Z. A. Liu. A Look–Ahead Lanczos Algorithm for Unsymmetric Matrices. *Mathematics of Computation*, 44(169):105–124, 1985.

[51] Y. Saad and M. H. Schulz. GMRES: A Generalized Minimum Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[52] C. Schelthoff. Vergleich von parallelen Verfahren zur Vorkonditionierung für die Methode der konjugierten Gradienten. Bericht Jül–2913, Forschungszentrum Jülich, Jülich, März 1994.

[53] J. N. Shadid and R. S. Tuminaro. A Comparison of Preconditioned Nonsymmetric Krylov Methods on a Large–Scale MIMD Machine. *SIAM Journal on Scientific Computing*, 15(2):440–459, 1994.

[54] P. Sonneveld. CGS, a Fast Lanczos–Type Solver for Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52, 1989.

[55] H. A. van der Vorst. Bi–CGSTAB: A Fast and Smoothly Converging Variant of Bi–CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[56] H. Vereecken, G. Lindenmayr, A. Kuhr, D. H. Welte, and A. Basermann. Numerical Modelling of Field Scale Transport in Heterogeneous Variably Saturated Porous Media. Internal Report KFA/ICG–4 No. 500393, Research Centre Jülich, Jülich, 1993.

[57] G. T. Yeh. 3DFEMWATER: A Three–Dimensional Finite Element Model of Water Flow through Saturated–Unsaturated Media. Technical Report ORNL–6386, Oak Ridge National Laboratory, Oak Ridge, 1987.

[58] Z. Zlatev. *Computational Methods for General Sparse Matrices.* Kluwer Academic Publishers, Dordrecht, 1991.

Danksagung

Die vorliegende Arbeit wurde als Diplomarbeit in Informatik am Lehrstuhl für Technische Informatik und Computerwissenschaften der Fakultät für Elektrotechnik der Rheinisch–Westfälischen Technischen Hochschule Aachen angefertigt.

Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich sowohl für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich (KFA) erstellen zu können, als auch für seine Unterstützung, die mir die Teilnahme an einer von der University of Colorado, Denver, organisierten Konferenz ermöglichte. Dieser Aufenthalt hat mir einen tieferen Einblick in die Theorie und Techniken von iterativen Methoden erlaubt. Herrn Prof. Dr. W. Oberschelp vom Lehrstuhl für Angewandte Mathematik insbesondere Informatik der RWTH Aachen bin ich für die Übernahme des Korreferates dankbar. An dieser Stelle dürfen A. Basermann, Dr. H. Burg und schließlich Dr. P. Weidner nicht unerwähnt bleiben, deren konstruktive Hinweise und Kommentare zum Gelingen der vorliegenden Arbeit beigetragen haben. Die zahlreichen Diskussionen mit Herrn C. Schelthoff haben mich ebenfalls positiv beeinflußt. Bei Frau J. Docter möchte ich mich für das Entgegenkommen bei der Zuteilung der Rechenzeiten auf dem PARAGON XP/S 10 der KFA bedanken. Herr H. Bast von der Firma Intel ermöglichte es mir, Programme unter einer neuen Version des Betriebssystems auf einem PARAGON–System in Beaverton, Oregon, zu testen.

Meinen Eltern danke ich an dieser Stelle für ihre Unterstützung während meiner Studienzeit.