Natascha Grammou

Diplomarbeit

Implementierung paralleler präkonditionierter iterativer Gleichungslöser unter Nutzung neuer Kommunikationsmethoden

Technische Universität Darmstadt

Institut für Numerische Methoden und Informatik im Bauwesen

Betreuer

Dipl.-Ing. J. Ruben

Univ.-Prof. Dr.-Ing. habil. U. F. Meißner

August 2003

Institut für Numerische Methoden und Informatik im Bauwesen
Univ.-Prof. Dr.-Ing. habil. Udo F. Meißner
Univ.-Prof. Dr.-Ing. Uwe Rüppel
Prof. Dr.-Ing. Ioan David

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich 13
Bauingenieurwesen und Geodäsie

Petersenstraße 13
D-64287 Darmstadt
Telefon: (06151) 16-34 44
Telefax: (06151) 16-55 52
e-mail: sekretariat@iib.tu-darmstadt.de

Mittwoch, 16. Juli 2003

Aufgabenstellung zur Diplomarbeit

Implementierung paralleler präkonditionierter iterativer Gleichungslöser unter Nutzung neuer Kommunikationsmethoden

für Frau cand. Ing. Natascha Grammou

Die numerische Lösung komplexer Ingenieurprobleme ist auf einem Prozessor aus Laufzeitgründen und aus Gründen der vorhandenen Systemressourcen oft nicht durchführbar. Ziel der aktuellen Forschung ist es, Methoden für die numerische Simulation im Internet bereitzustellen, um parallele Berechnungen über lokale Netzwerkgrenzen hinaus, durchzuführen.

Dünnbesetzte Gleichungssysteme haben eine zentrale Bedeutung bei der Lösung von Aufgaben der Strukturmechanik mit Hilfe der finiten Elemente. Gebietszerlegungsverfahren erlauben eine effiziente Parallelisierung iterativer Lösungsmethoden. Die Konvergenzeigenschaften des Konjugierten Gradienten Verfahrens (CG) und verwandter Methoden können mit vorkonditionierten Varianten deutlich verbessert werden.

Ziel der Arbeit ist es daher, die parallele Präkonditionierung in einer heterogen verteilten Umgebung auf Basis der Agententechnologie umzusetzen.

Im Einzelnen sind folgende Aufgaben zu bearbeiten:

1. **Analyse verschiedener Präkonditionierer**

   Frau Grammou soll verschiedene Präkonditionierer im Bezug auf die Verbesserung der Konditionierung der Steifigkeitsmatrix, des Implementierungsaufwandes und der Möglichkeit ihrer Parallelisierung untersuchen. Dabei sollen sowohl Verfahren zur Präkonditionierung der Gesamtsteifigkeitsmatrix als auch der einzelnen Elementsteifigkeitsmatrizen betrachtet werden.

2. **Implementierung eines parallelen präkonditionierten iterativen Gleichungslösers**

   Basierend auf den im ersten Aufgabenteil gewonnenen Erkenntnissen soll das Präkonditionierte Konjugierte Gradienten Verfahren (PCG) und verwandte Methoden zur Lösung nichsymmetrischer Gleichungssysteme in einer heterogen verteilten Umgebung auf Basis der Agententechnologie umgesetzt werden.

3. **Funktionsfähigkeit der Implementierung, Dokumentation und Vortrag**

   Die Funktionsfähigkeit des Systems ist an typischen Beispielen zu überprüfen. Alle Teilaufgaben sind umfangreich zu dokumentieren und im Rahmen eines Vortrags am Institut für Numerische Methoden und Informatik im Bauwesen zu präsentieren.

Bearbeitungszeit:	6 Wochen
Ausgabe der Diplomarbeit:	21.07.2003
Abgabe der Diplomarbeit:	01.09.2003

Ständige Rücksprache mit dem zuständigen Betreuer ist erforderlich.

Betreuer:	Dipl.-Ing. Jochen Ruben
Prüfer:	Univ.-Prof. Dr.-Ing. habil. Udo F. Meißner

……...

(Univ.-Prof. Dr.-Ing. habil. Udo F. Meißner)

Diplomarbeit von Frau Natascha Grammou

Erklärung zur Diplomarbeit gemäß § 19 Abs. 6 DPO/AT

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 01.09.2003

(Natascha Grammou)

Einleitung

Die Lösung von komplexen Problemen in der Ingenieurpraxis erfordert häufig Vereinfachungen des betrachteten Systems. Diese Vereinfachungen können sich auf die Abstraktion von Geometrie und Randbedingungen beziehen, die Idealisierung von Werkstoffeigenschaften und dem physikalischen Verhalten oder auf die Definition von Ersatzlasten. Eine analytisch geschlossene Lösung existiert aber oft nur für Sonderfälle und zum Teil nur unter groben Vereinfachungen und unrealistischen Randbedingungen, so dass numerische Verfahren für eine Berechnung zu Hilfe genommen werden. Dabei wird ein reales System durch ein approximiertes System ersetzt, für das eine Näherungslösung berechnet werden kann. Das Modell wird in finite Elemente diskretisiert, für die ein linearer oder nicht linearer Zusammenhang zwischen Elementkräften und -verschiebungen aufgestellt wird. Aus den einzelnen elementbezogenen Gleichungssystemen wird unter Berücksichtigung der vorgegebenen Randbedingungen ein lineares Gleichungssystem für das idealisierte Gesamtsystem zusammengefügt. Da die Diskretisierung mit finiten Elementen durchgeführt wird, bezeichnet man diese numerische Simulation als Finite-Element-Methode (FEM). Die Anwendung der FEM, die in der Regel ein symmetrisches Gleichungssystem zur Folge hat; wurde in zunehmendem Maße zur Berechnung komplexer Systeme anerkannt und gehört im Bauwesen mittlerweile zum Stand der Technik.

Lineare Gleichungssysteme werden in der Regel mit direkten Verfahren (10), wie beispielsweise die Verfahren von Gauß und Cholesky, gelöst. Dabei ist der Aufwand bei fehlender Bandstruktur von der Ordnung $O(n^2)$. Ab einer gewissen Größe des Gleichungssystems führt dies zu einem Rechenaufwand und auch zu einem Speicherplatzbedarf, der selbst bei modernen Hochleistungsrechnern zu Kapazitätsproblemen führen kann. Darin ist die Entwicklung von effizienteren Lösungsverfahren begründet.

Im Rahmen dieser Diplomarbeit werden zur effizienten Lösung numerischer Probleme iterative Verfahren (10) der Gruppe der konjugierten Gradienten vorgestellt. Die Effizienz wird jedoch erst durch eine Transformation des linearen Gleichungssystems erreicht. Diese Transformation wird als Präkonditionierung bezeichnet. Der Schwerpunkt dieser Arbeit liegt auf der Beschreibung der Präkonditionierer und ihrer Auswertung bezüglich der Rechenbeschleunigung und Stabilisierung der iterativen Verfahren.

Eine schnellere Problemlösung ist auch immer mit einer Kostenersparnis verbunden und somit von großem wirtschaftlichem Interesse. Um eine zusätzliche Leistungssteigerung zu erzielen, werden die Verfahren parallel auf Multiprozessorsystemen angewendet.

Gegenstand dieser Diplomarbeit ist zunächst die Beschreibung der iterativen Verfahren und der allgemeinen Vorgehensweise ihrer Transformation zur Beschleunigung und Stabilisierung (Präkonditionierung) in den ersten beiden Kapiteln. Kapitel 3 stellt Parallelisierungsmöglichkeiten iterativer Verfahren zur weiteren Leistungssteigerung vor. Kapitel 4 beschreibt detailliert die Funktionsweise und Implementierung der Präkonditionierungen. Zudem werden der Implementierungsaufwand, die Anwendung auf paralleler Ebene vergleichend ausgewertet. Kapitel 5 beinhaltet Anwendungsbeispiele, die in Kapitel 6 verglichen werden. Dabei wird insbesondere auf die beschleunigende und stabilisierende Wirkung der Präkonditionierer auf die iterativen Verfahren eingegangen. In Kapitel 7 wird die Implementierung der parallelen Operationen erläutert. Diese Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 8.

1 Funktion iterativer Gleichungslöser

Durch den starken Anstieg der Leistungsfähigkeit von Personal Computern und die zunehmende Anwendung der Finiten-Element-Methode steigt auch der Bedarf an numerischen Verfahren zur effizienten Lösung von großen linearen Gleichungssystemen. Zur Lösung eines linearen Gleichungssystems ermitteln iterative Verfahren durch wiederholtes Ausführen einer festgelegten Rechenvorschrift sukzessive Näherungslösungen, bis die gewünschte Lösungs-Genauigkeit erreicht wird.

Die direkten Verfahren erzielen nur für eine kleine Anzahl von Unbekannten zufriedenstellende Ergebnisse. In der praktischen Anwendung existieren jedoch überwiegend große Gleichungssysteme mit schwachbesetzten Matrizen (Nulleinträge stellen den größten Anteil der Matrixkoeffizienten dar). Die Vorteile der iterativen Verfahren im Vergleich zu den direkten Verfahren liegen im wesentlich geringeren Speicherplatzbedarf und kürzeren Rechenzeiten. Die für große Matrizen typische Schwachbesetzung wird ausgenutzt, das heißt die Nullelemente der Matrix werden nicht gespeichert, wodurch neben der Speicherplatzersparnis der Rechenablauf zusätzlich verkürzt wird. Selbst bei der Benutzung von Hochleistungsrechnern führen direkte Verfahren zu unakzeptablen Rechenzeiten. Ein weiterer Nachteil, den die direkten Verfahren mit sich bringen, ist die schlechte Parallelisierbarkeit aufgrund mangelnder Skalierbarkeit (10). In Kapitel 3 wird näher auf das Thema Parallelisierung eingegangen. Im Gegensatz zu den direkten Verfahren ist jedoch bei den iterativen Verfahren auch bei paralleler Anwendung der Rechenaufwand linear von der Größe des Gleichungssystems und der Anzahl der Iterationen abhängig. Folglich sind iterative Verfahren sehr gut für Parallelisierungen großer Systeme geeignet. Die Verteilung der Rechenoperationen auf mehrere Prozessoren führt bei sehr großen Systemen zu einer immensen Zeitersparnis und somit hohen Effizienz.

In den nachfolgenden Kapiteln bedeuten fettgedruckte Groß-Buchstaben Matrizen und fettgedruckte Klein-Buchstaben Vektoren. Andere Operanden in Verbindung mit Matrizen und Vektoren sind Skalare. In Kapitel 9 sind die verwendeten Symbole aufgelistet.

2 Iterative Gleichungslöser

2.1 Das konjugierte Gradientenverfahren (CG)

Das konjugierte Gradientenverfahren (engl. Conjugate Gradient) gehört zu den am meisten angewendeten iterativen Verfahren zur Lösung von großen linearen Gleichungssystemen der Form $\boldsymbol{Ax} = \boldsymbol{b}$.

$\boldsymbol{A}$ stellt die Steifigkeitsmatrix des Systems dar, $\boldsymbol{b}$ ist der Lastvektor und $\boldsymbol{x}$ der Unbekanntenvektor beziehungsweise der Verschiebungsvektor.

Voraussetzungen für das konjugierte Gradientenverfahren sind eine symmetrische und positiv definite Steifigkeitsmatrix $\boldsymbol{A}$. Zudem ist das CG-Verfahren unter Berücksichtigung der Rechenzeit sogar nur für schwachbesetzten Steifigkeitsmatrizen sinnvoll anwendbar. Andernfalls würde aus diesem Verfahren kein Geschwindigkeitsvorteil gegenüber einem direkten Verfahren resultiern.

Das Gleichungsproblem $\boldsymbol{Ax} = \boldsymbol{b}$ lässt sich in eine quadratische Form zu einem Minimierungsproblem umformen:

$$f(x) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\,\boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x} + \frac{1}{2}\boldsymbol{b}^T \boldsymbol{A}^{-1}\boldsymbol{b}\,. \tag{2.1}$$

Wenn $\boldsymbol{A}$ positiv definit ist, stellt dieses Funktional (10) ein Paraboloid dar (Abbildung 1 und Abbildung 2). In Abbildung 3 sind die Gradienten, das heißt die erste Ableitung des Funktionals, des Paraboloids dargestellt. Den Gradient Null im untersten Punkt des Paraboloids und somit die gesuchte Lösung erhält man durch Nullsetzen der ersten Ableitung des Funktionals (2.1):

$$f'(x) = \boldsymbol{Ax} - \boldsymbol{b} = 0\,. \tag{2.2}$$

Das heißt, dass die nichttriviale Lösung des Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$ dem globalen Minimum des Funktionals entspricht.

Wie in Kapitel 1 erwähnt, vollzieht sich dieser Vorgang iterativ, indem in jedem Schritt bezüglich einer gewählten Suchrichtung $\boldsymbol{p}_m$ das Minimum berechnet wird. Der Index m bezeichnet den jeweiligen Iterationsschritt. Bei der Wahl der neuen Suchrichtung muss die Bedingung

$$\boldsymbol{p}_{m+1}^T \boldsymbol{A}\,\boldsymbol{p}_m = 0 \tag{2.3}$$

erfüllt sein. Das heißt, dass das CG-Verfahren für eine positiv definite Matrix nach maximal n Schritten - n ist die Anzahl der Freiheitsgrade - die exakte Lösung $\boldsymbol{x}$ liefert, da kein Suchvektor erneut in eine der vorherigen Richtungen zeigt.

Im ersten Iterationsschritt wird der Suchvektor in entgegengesetzter Richtung des Gradienten, das heißt in Richtung des steilsten Abstiegs des Paraboloids gewählt:

$$\boldsymbol{p}_0 = -f'(\boldsymbol{x}_0) = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0 . \tag{2.4}$$

Wobei der Startvektor $\boldsymbol{x}_0$ üblicherweise aufgrund fehlender Vorkenntnisse auf Null gesetzt wird. Die Gleichung (2.4) entspricht auch dem Residuenvektor des ersten Iterationsschritts $\boldsymbol{r}_0$. Das Residuum $\|\boldsymbol{r}\|_2$ dient dazu, die Abweichung des jeweils in den einzelnen Schritten berechneten Verschiebungsvektors vom Lösungsvektor bestimmen zu können, denn der Fehlervektor $\boldsymbol{e}_m = \boldsymbol{x}_m - \boldsymbol{x}$ ist von $\boldsymbol{x}$ selbst abhängig und somit unbrauchbar (Das Residuum verschwindet analog zum Fehler, wenn die iterierte Lösung $\boldsymbol{x}_m$ mit der exakten Lösung übereinstimmt). Das Verhältnis des jeweils im Iterationsschritt m berechneten Residuums zum Referenzresiduum (Residuum des ersten Durchlaufs $\|\boldsymbol{r}_0\|_2$) wird für die iterativen Verfahren als Abbruchkriterium benutzt. Vorteil dieses Abbruchkriteriums gegenüber anderen (z.B. nur $\|\boldsymbol{r}_m\|_2$) ist die Lastunabhängigkeit. Das heißt, dass bei sehr großen Lasten nicht unnötig viele Iterationsschritte durchgeführt werden und bei sehr kleinen Lasten der Vorgang nicht abgebrochen wird, ohne eine hinreichende Genauigkeit des Ergebnisses erlangt zu haben. Die Vorgehensweise zur Ermittlung des Residuenvektors und dessen Länge für den Iterationsschritt m wird im Algorithmus 2-1 gezeigt.

Nachdem die Suchrichtung des neuen Vektors festgelegt ist, wird die Schrittweite α des Vektors bestimmt. Wie in Abbildung 4 zu sehen ist, entspricht die Bestimmung der Vektorlänge α der Ermittlung des tiefsten Punktes entlang der Schnittmenge (Parabel, siehe Abbildung 5) zwischen $\boldsymbol{p}_m$ und dem Funktional. Man erhält die Schrittweite, indem die Ableitung des Funktionals nach α gleich Null gesetzt wird (siehe [10]). Nach Umformen ergibt sich α zu:

$$\alpha = \frac{\boldsymbol{r}_m^T\,\boldsymbol{p}_m}{\boldsymbol{p}_m^T\,\boldsymbol{A}\,\boldsymbol{p}_m} . \tag{2.5}$$

Der CG-Algorithmus sieht folgendermaßen aus:

Algorithmus 2-1: CG-Verfahren

$\boldsymbol{x}_0 = 0\,; \quad \varepsilon = 0{,}001$		
$\boldsymbol{p}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$		
Iteration für $m = 1, 2, \ldots n$		
	J $\quad \dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon \quad$ N	
	$\boldsymbol{v}_m = \boldsymbol{A}\,\boldsymbol{p}_{m-1}$	STOP
	$\sigma_m = \boldsymbol{v}_m^T\,\boldsymbol{p}_{m-1}$	
	$\alpha_m = \dfrac{\boldsymbol{r}_{m-1}^T\,\boldsymbol{p}_{m-1}}{\sigma_m}$	
	$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \alpha_m\,\boldsymbol{p}_{m-1}$	
	$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} - \alpha_m\,\boldsymbol{v}_m$	
	$\beta_m = \dfrac{\boldsymbol{r}_m^T\,\boldsymbol{v}}{\sigma_m}$	
	$\boldsymbol{p}_m = \boldsymbol{r}_m - \beta_m\,\boldsymbol{p}_{m-1}$	

Die Konvergenzgeschwindigkeit dieses Algorithmus wird vom größten Eigenwert der Steifigkeitsmatrix bestimmt.

Für den Fall, dass alle Eigenwerte einer Matrix gleich sind, konvergiert das Verfahren unabhängig vom Startwert nach der ersten Iteration, da die Isolinien (Ellipsoide) des Paraboloids eine sphärische Form bekommen (Abbildung 6).

Das CG-Verfahren führt pro Iterationsschritt vier einfache Vektorberechnungen durch, darunter drei Skalarmultiplikationen. Die Steifigkeitsmatrix selbst wird nur für eine Matrix-Vektor-Multiplikation benötigt. Durch die einfache Struktur und mathematischen Operationen in oben stehenden Algorithmus werden die Anforderungen an kurze Rechenzeit [7] und geringen Speicherbedarf für große lineare Gleichungssysteme erfüllt.

2.2 Das stabilisierte BiCGSTAB-Verfahren

Wie schon in Kapitel 2.1 erwähnt, kann das CG-Verfahren nur für symmetrische Matrizen angewendet werden. Da jedoch in der Praxis auch nicht symmetrische Gleichungssysteme vorkommen, wie zum Beispiel bei der Berechnung der Interaktion zwischen Baugrund und Tragwerk, wird in diesem Kapitel das stabilisierte BiCGSTAB-Verfahren vorgestellt. Dieses Verfahren gehört zur Gruppe der „Verfahren der konjugierten Gradienten“. Die Abkürzung BiCGSTAB steht für Bi-Conjugate Gradient Stabilized.

Die einzige Voraussetzung für diesen Algorithmus ist eine reguläre Steifigkeitsmatrix $\boldsymbol{A}$.

Wie aus der Bezeichnung „stabilisiertes BiCGSTAB-Verfahren“ zu erkennen ist, wurde dieses Verfahren entwickelt, um das BiCGSTAB-Verfahren zu verbessern, welches wiederum zur Korrektur des BiCG-Verfahrens entwickelt wurde.

Die Nachteile des BiCG-Verfahrens sind:

1. Erforderliche Multiplikationen mit der zu $\boldsymbol{A}$ transponierten Matrix $\boldsymbol{A}^T$,
2. Oszillationen im Konvergenzverhalten,
3. Vorzeitiges Abbrechen des Verfahrens bei einer regulären Matrix $\boldsymbol{A}$.

Beim BiCGSTAB-Verfahren wurde der Punkt 1 beseitigt, der Punkt 2 nur teilweise behoben und ein schnelleres Konvergenzverhalten erzielt. Die beiden Verfahren werden aufgrund ihrer Nachteile nicht näher erläutert.

Um die Verfahrensabbrüche zu vermeiden, wurde das stabilisierte BiCGSTAB-Verfahren entwickelt, das auf der nachfolgenden Seite in einem Struktogramm dargestellt wird.

Algorithmus 2-2: Stabilisiertes BiCGSTAB-Verfahren

$\boldsymbol{x}_0 = 0; \quad \varepsilon = 0{,}001; \quad \tilde{\varepsilon} = 10^{-9}$		
$\boldsymbol{p}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$		
Iteration für m = 1, 2, ... Solange $\dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$		
$\boldsymbol{v}_m = \boldsymbol{A}\,\boldsymbol{p}_{m-1}$		
$\sigma_m = \boldsymbol{v}_m^T\,\boldsymbol{p}_{m-1}$		
J $\quad \sigma_m > \tilde{\varepsilon}\,\lVert \boldsymbol{v}_m \rVert_2 \lVert \boldsymbol{r}_0 \rVert_2 \quad$ N		
$\alpha_m = \dfrac{\boldsymbol{r}_{m-1}^T\,\boldsymbol{p}_0}{\sigma_m}$		Neustart mit $\boldsymbol{x}_0 = \boldsymbol{x}_m$
$\boldsymbol{s}_m = \boldsymbol{r}_{m-1} - \alpha_m\,\boldsymbol{v}_m$		
J $\quad \lVert \boldsymbol{s}_m \rVert_2 > \varepsilon \quad$ N		
$\boldsymbol{t}_m = \boldsymbol{A}\,\boldsymbol{s}_m$	$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \alpha_m\,\boldsymbol{p}_{m-1}$	
$\omega_m = \dfrac{\boldsymbol{t}_m\,\boldsymbol{s}_m}{\boldsymbol{t}_m\,\boldsymbol{t}_m}$	$\boldsymbol{r}_m = \boldsymbol{s}_{m-1}$	
$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \alpha_m\,\boldsymbol{p}_{m-1} + \omega_m\,\boldsymbol{s}_m$	$m = m + 1$	
$\boldsymbol{r}_m = \boldsymbol{s}_m - \omega_m\,\boldsymbol{t}_m$		
$\beta_m = \dfrac{\alpha_m}{\omega_m}\,\dfrac{\boldsymbol{r}_m^T\,\boldsymbol{r}_0}{\boldsymbol{r}_{m-1}\,\boldsymbol{r}_0}$		
$\boldsymbol{p}_m = \boldsymbol{r}_m - \beta_m\left(\boldsymbol{p}_{m-1} - \omega_m\,\boldsymbol{v}_m\right)$		
$m = m+1$		

Vor der Berechnung des Skalars α_m wird die Größe des Nenners σ_m überprüft, um bei großen, aber annähernd orthogonalen Vektoren $\boldsymbol{v}_m$ und $\boldsymbol{r}_0$, eine Division durch sehr kleine Werte (im schlimmsten Fall Null bei aufeinander senkrecht stehenden $\boldsymbol{v}_m$ und $\boldsymbol{r}_0$) zu vermeiden. Liegt der Betrag von σ_m unterhalb einer vorgegebenen Schranke $\tilde{\varepsilon}$, wird ein Neustart durchgeführt. Um bei der Berechnung von ω_m ebenfalls eine Division mit Null zu vermeiden, wird vorher der Betrag von $\boldsymbol{s}_m$ überprüft. Durch diese beiden Abfragen wird der vorzeitige Abbruch des Verfahrens vermieden.

Im Vergleich zum CG-Verfahren benötigt das stabilisierte BiCGSTAB-Verfahren pro Iteration drei zusätzliche Rechenschritte. Diese sind die Ermittlung der Vektoren $\boldsymbol{s}$ und $\boldsymbol{t}$ und des skalaren

Werts ω. Dem Mehraufwand der zusätzlichen Berechnungen steht mit diesem Verfahren die Möglichkeit der Lösung linearer Gleichungssystemen mit nicht symmetrischen Steifigkeitsmatrizen gegenüber.

2.3 Das präkonditionierte konjugierte Gradientenverfahren (PCG)

Das CG-Verfahren dient als Grundlage vieler, fortschrittlichster Iterationsverfahren zur Lösung von linearen Gleichungssystemen. Dabei ist die Konvergenzgeschwindigkeit jedoch stark von der Differenz der Eigenvektoren abhängig. Je stärker sich die Eigenwerte voneinander unterscheiden, desto mehr Iterationsschritte sind nötig. Die Kondition der Steifigkeitsmatrix $\boldsymbol{A}$ lässt sich über die Konditionszahl $\kappa(\boldsymbol{A})$, die das Verhältnis vom größten zum kleinsten Eigenwert angibt,

$$\kappa(\boldsymbol{A}) = \frac{\lambda_{max}}{\lambda_{min}} \text{ (mit } \kappa(\boldsymbol{A}) \geq 1) \tag{2.6}$$

beurteilen.

Für die Konditionszahl 1 liefert das CG-Verfahren im ersten Iterationsschritt hinreichend genaue Ergebnisse, das heißt je näher sich die Konditionszahl dem Wert 1 nähert, desto höher ist auch die Konvergenzgeschwindigkeit und damit die Anzahl der Iterationsschritte geringer. Ziel ist es also die Konditionszahl bei der Anwendung dieses Verfahrens möglichst klein zu halten.

Eine sehr effiziente Möglichkeit die Konditionszahl zu verbessern, ist die Anwendung von Präkonditionierern. In Kapitel 4 werden die Präkonditionierer ausführlich beschrieben. Nachfolgend sollen lediglich die Algorithmen des präkonditionierten CG-Verfahrens und des präkonditionierten stabilisierten BiCGSTAB-Verfahrens (Kapitel 2.4) vorgestellt werden.

Beim PCG-Verfahren (engl. Preconditioned Conjugate Gradient) wird das Gleichungssystem $\boldsymbol{Ax} = \boldsymbol{b}$ durch eine reguläre Matrix $\boldsymbol{P}$, die als Präkonditionierer bezeichnet wird, in die äquivalente Form

$$\boldsymbol{PAx} = \boldsymbol{Pb} \tag{2.7}$$

transformiert, mit dem Ziel, die Konditionszahl $\kappa(\boldsymbol{PA})$ gegenüber $\kappa(\boldsymbol{A})$ zu minimieren und das CG-Verfahren zu stabilisieren.

Die Struktur des PCG-Verfahrens verdeutlicht Algorithmus 2-3:

Algorithmus 2-3: PCG-Verfahren

$\boldsymbol{x}_0 = 0\,; \quad \varepsilon = 0{,}001$		
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$		
$\boldsymbol{p}_0 = \hat{\boldsymbol{r}}_0 = \boldsymbol{P}\,\boldsymbol{r}_0$		
Iteration für $m = 1, 2, ..., n$		
	J $\quad \dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon \quad$ N	
	$\boldsymbol{v}_m = \boldsymbol{A}\,\boldsymbol{p}_{m-1}$	STOP
	$\sigma_m = \boldsymbol{v}_m^T\,\boldsymbol{p}_{m-1}$	
	$\alpha_m = -\dfrac{\hat{\boldsymbol{r}}_{m-1}^T\,\boldsymbol{r}_{m-1}}{\sigma_m}$	
	$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} - \alpha_m\,\boldsymbol{p}_{m-1}$	
	$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} + \alpha_m\,\boldsymbol{v}_m$	
	$\hat{\boldsymbol{r}}_m = \boldsymbol{P}\,\boldsymbol{r}_m$	
	$\beta_m = \dfrac{\hat{\boldsymbol{r}}_m^T\,\boldsymbol{r}_m}{\sigma_m}$	
	$\boldsymbol{p}_m = \hat{\boldsymbol{r}}_m + \beta_m\,\boldsymbol{p}_{m-1}$	

Dieses Verfahren erfordert gegenüber dem CG-Verfahren zusätzlichen Speicherplatz für den Präkonditionierer und den Vektor $\hat{r}_m$. Die Anzahl der Rechenschritte wird lediglich um die Präkonditionierung des Vektors $\boldsymbol{r}_m$ erweitert.

2.4 Das präkonditionierte stabilisierte BiCGSTAB-Verfahren

Die Motivation, die Kondition des linearen Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$ durch geeignete Präkonditionierer $\boldsymbol{P}$ zu verbessern, um schneller an die exakte Lösung zu gelangen, kann entsprechend Kapitel 2.3 auch auf das stabilisierte BiCGSTAB-Verfahren übertragen werden. Der präkonditionierte stabilisierte BiCGSTAB-Algorithmus wird nachfolgend vorgestellt.

Algorithmus 2-4: Präkonditioniertes stabilisiertes BiCGSTAB-Verfahren

$\boldsymbol{x}_0 = 0; \quad \varepsilon = 0{,}001; \quad \tilde{\varepsilon} = 10^{-9}$

$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$

$\boldsymbol{p}_0 = \boldsymbol{P}\,\boldsymbol{r}_0$

Iteration für $m = 1, 2, \ldots$ Solange $\dfrac{\|\boldsymbol{r}_{m-1}\|_2}{\|\boldsymbol{r}_o\|_2} > \varepsilon$

$\boldsymbol{v}_m = \boldsymbol{A}\,\boldsymbol{p}_{m-1}$

$\hat{\boldsymbol{v}}_m = \boldsymbol{P}\,\boldsymbol{v}_m$

$\sigma_m = \hat{\boldsymbol{v}}_m^T\,\boldsymbol{p}_0$

$m == 1$

J:

$\alpha_m = \dfrac{\boldsymbol{p}_{m-1}^T\,\boldsymbol{p}_0}{\sigma_m}$

N:

$\alpha_m = \dfrac{\hat{\boldsymbol{r}}_{m-1}^T\,\boldsymbol{p}_0}{\sigma_m}$

$\boldsymbol{s}_m = \boldsymbol{r}_{m-1} - \alpha_m\,\boldsymbol{v}_m$

$\hat{\boldsymbol{s}}_m = \boldsymbol{P}\,\boldsymbol{s}_m$

$\|\hat{\boldsymbol{s}}_m\|_2 > \tilde{\varepsilon}$

J:

$\boldsymbol{t}_m = \boldsymbol{A}\,\hat{\boldsymbol{s}}_m$

$\hat{\boldsymbol{t}}_m = \boldsymbol{P}\,\boldsymbol{t}_m$

$\omega_m = \dfrac{\hat{\boldsymbol{t}}_m\,\hat{\boldsymbol{s}}_m}{\hat{\boldsymbol{t}}_m\,\hat{\boldsymbol{t}}_m}$

$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \alpha_m\,\boldsymbol{p}_{m-1} + \omega_m\,\hat{\boldsymbol{s}}_m$

$\boldsymbol{r}_m = \boldsymbol{s}_m - \omega_m\,\boldsymbol{t}_m$

$\hat{\boldsymbol{r}}_m = \hat{\boldsymbol{s}}_m - \omega_m\,\hat{\boldsymbol{t}}_m$

$m == 1$

J:

$\gamma = \boldsymbol{p}_{m-1}^T\,\boldsymbol{p}_0$

N:

$\gamma = \hat{\boldsymbol{r}}_{m-1}^T\,\boldsymbol{p}_0$

$\beta_m = \dfrac{\alpha_m}{\omega_m}\,\dfrac{\hat{\boldsymbol{r}}_m^T\,\boldsymbol{p}_0}{\gamma}$

N:

$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + \alpha_m\,\boldsymbol{p}_{m-1}$

$\boldsymbol{r}_m = \boldsymbol{s}_{m-1}$

$m = m + 1$

	$\boldsymbol{p}_m = \hat{\boldsymbol{r}}_m + \beta_m \left(\boldsymbol{p}_{m-1} - \omega_m \hat{\boldsymbol{v}}_m \right)$	
	$m = m+1$	

Auch in diesem Fall akzeptiert man einen erhöhten Speicherplatzbedarf und Rechenaufwand gegenüber dem stabilisierten BiCGSTAB-Verfahren. Im Vergleich zum PCG-Verfahren müssen im Algorithmus 2-4 jedoch die Vektoren $\boldsymbol{v}$, $\boldsymbol{s}$ und $\boldsymbol{t}$ in jedem Iterationsschritt präkonditioniert werden. Der Residuenvektor muss nicht explizit präkonditioniert werden, sondern lässt sich dann aus den zuvor präkonditionierten Vektoren ermitteln.

3 Parallele Lösung großer, linearer Gleichungssysteme

In den Kapiteln 2.3 und 2.4 wurden die iterativen Verfahren präkonditioniert, um eine schnellere Konvergenzgeschwindigkeit zu erzielen.

Eine weitere sehr effektive Möglichkeit, vor allem für Gleichungssysteme mit mehr als 5000 Unbekannten, die Rechenzeit zu verkürzen, ist die Parallelisierung der iterativen Verfahren. Dabei wird die Steifigkeitsmatrix in Teilmatrizen gesplittet und mindestens zwei Prozessoren zugeteilt. Die Prozessoren berechnen ihr Teilsystem, senden ihre Ergebnisse an den Host-Agenten, der wiederum die Teilergebnisse zu einer Gesamtlösung des Problems zusammenfügt.

Auf die genaue Implementierung der Parallelisierung wird erst in Kapitel 7 eingegangen. In diesem Kapitel werden lediglich das theoretische Vorgehen und die Algorithmen beschrieben.

3.1 Allgemeine Vorgehensweise bei der Parallelisierung iterativer Verfahren

Eine Parallelisierung erfordert zunächst eine Partitionierung des Finite-Element-Netzes in p Gebiete. Diese Arbeit beschränkt sich auf die nichtüberlappende Gebietszerlegung. Vorteil dieser Gebietszerlegung ist ein elementweiser Aufbau der Steifigkeitsmatrizen für jedes Teilgebiet s. Da die einzelnen finite Elemente eindeutig einem Teilgebiet beziehungsweise einem Prozessor zugewiesen werden, kann der Aufbau der Steifigkeitsmatrizen ohne jegliche Kommunikation zwischen den Rechnern erfolgen. Die zwischen den Teilgebieten liegenden Knoten werden jeweils den angrenzenden Gebieten zugeordnet. Diese Knotenmenge nennt man Koppelknotenmenge.

Als Folge erhält man eine Aufsplittung des linearen Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$ in ein Gleichungssystem aus Blockmatrizen:

$$\underbrace{\begin{pmatrix} \boldsymbol{A}^{11} & & & \boldsymbol{A}^{1c} \\ & \ddots & & \vdots \\ & & \boldsymbol{A}^{pp} & \boldsymbol{A}^{pc} \\ \boldsymbol{A}^{c1} & \cdots & \boldsymbol{A}^{cp} & \boldsymbol{A}^{cc} \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} \boldsymbol{x}^{1} \\ \vdots \\ \boldsymbol{x}^{p} \\ \boldsymbol{x}^{c} \end{pmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} \boldsymbol{b}^{1} \\ \vdots \\ \boldsymbol{b}^{p} \\ \boldsymbol{b}^{c} \end{pmatrix}}_{\boldsymbol{b}} \tag{3.1}$$

$\boldsymbol{A}^{pp}$ stellt die Blockmatrizen dar, die nur aus nicht gekoppelten Werten zusammengesetzt sind (Innenknoten). $\boldsymbol{A}^{cc}$ sind Blockmatrizen, die sich nur aus Koppelwerten zusammensetzen, und $\boldsymbol{A}^{pc}$ beziehungsweise $\boldsymbol{A}^{cp}$ beinhalten Koppel- und Nichtkoppelwerte.

Da die Werte der Gesamtsteifigkeitsmatrix knotenbezogen sind und sich additiv aus den Werten der Elementsteifigkeitsmatrizen ergeben, führt die nichtüberlappende Gebietszerlegung zu additiv verteilten Teilgebietssteifigkeitsmatrizen $\boldsymbol{A}^{s}$. Für einen Knotenfreiheitsgrad in der

Gesamtsteifigkeitsmatrix müssen, mit Hilfe eines Zuordnungsoperators $\boldsymbol{B}^s$, die Anteile aus den Teilgebietssteifigkeitsmatrizen aufaddiert werden:

$$\boldsymbol{A} = \sum_s \left(\boldsymbol{B}^{s^T} \boldsymbol{A}^s \boldsymbol{B}^s \right). \tag{3.2}$$

Die rechte Seite des Gesamtsystems setzt sich entsprechend aus den rechten Seiten der Teilsysteme additiv zusammen:

$$\boldsymbol{b} = \sum_s \left(\boldsymbol{B}^{s^T} \boldsymbol{b}^s \right). \tag{3.3}$$

Bei einer parallelen Berechnung werden jedoch auch in den einzelnen Prozessoren die akkumulierten Daten benötigt, das heißt, dass für einen Knotenfreiheitsgrad identische Werte in allen Prozessoren vorhanden sein sollen, denen dieser Knoten als Koppelknoten zugewiesen ist. Folglich müssen die additiven Daten über Kommunikation zwischen den Prozessoren in akkumulierte Daten überführt werden. Wie schon am Anfang erwähnt, wird die Implementierung erst in Kapitel 7 ausführlicher beschrieben.

Nachfolgend werden Symbole und Rechenoperationen, die in den Struktogrammen (Algorithmus 3-1 bis Algorithmus 3-4) verwendet werden, erläutert:

- Die Algorithmen beziehen sich beispielhaft auf die Arbeitsschritte eines Clients.
- Die Pfeile in den Struktogrammen symbolisieren die Kommunikation zwischen den Prozessoren und dem Host, bei der ein additiver Vektor oder Skalar in einen akkumulierten Vektor oder Skalar umgewandelt wird.
- Die akkumulierten Daten tragen in den Struktogrammen den Index *akk.*. Bei den additiven Daten wurde auf eine zusätzliche Bezeichnung verzichtet.
- Das Skalarprodukt aus einem additiv verteilten Vektor und einem akkumulierten Vektor ergibt einen additiv verteilten Vektor.

  $\boldsymbol{b} = \sum_s \left(\boldsymbol{B}^{s^T} \boldsymbol{b}^s \right)$ sei ein additiv verteilter Vektor und $\boldsymbol{x}$ ein akkumulierter Vektor:

  $$\boldsymbol{b}\,\boldsymbol{x} = \left(\sum_s \left(\boldsymbol{B}^{s^T} \boldsymbol{b}^s \right) \right)^T \boldsymbol{x} = \sum_s \left(\boldsymbol{b}^{s^T} \boldsymbol{B}^s \boldsymbol{x} \right) = \sum_s \left(\boldsymbol{b}^{s^T} \boldsymbol{x}^s \right)$$

- Das Produkt aus einer additiv verteilten Matrix und einem akkumulierten Vektor ergibt ebenfalls einen additiv verteilten Vektor:

  $$\boldsymbol{A}\,\boldsymbol{x} = \sum_s \left(\boldsymbol{B}^{s^T} \boldsymbol{A}^s \boldsymbol{B}^s \right) \boldsymbol{x} = \sum_s \boldsymbol{B}^{s^T} \left(\boldsymbol{A}^s \boldsymbol{B}^s \boldsymbol{x} \right) = \sum_s \boldsymbol{B}^{s^T} \left(\boldsymbol{A}^s \boldsymbol{x}^s \right)$$

- Das Produkt aus zwei additiven (akkumulierten) Vektoren ergibt wiederum einen additiven (akkumulierten) Vektor. Entsprechendes gilt auch für Matrizen und Skalare.

3.1.1 Parallelisierung des konjungierten Gradientenverfahrens

Nachfolgender Algorithmus erfordert gegenüber dem sequentiellen Verfahren (Algorithmus 2-1) an zusätzlichem Aufwand vier Kommunikationen pro Iteration.

Der Kommunikationsaufwand beschränkt sich je Iterationsschritt auf den Austausch von zwei Skalaren, eine Vektortypumwandlung und die Berechnung des Residuums. Die Berechnung des Referenzresiduums $\|\boldsymbol{r}_o\|_2$ erfolgt einmalig beim ersten Durchlauf des Algorithmus.

Algorithmus 3-1: Paralleles CG-Verfahren

$\boldsymbol{x}_0^{akk.} = 0\,; \quad \varepsilon = 0{,}001$		
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$	$\rightarrow$	$\boldsymbol{w}_0^{akk.}$
$\boldsymbol{s}_0^{akk.} = \boldsymbol{w}_0^{akk.}$		
$\gamma_0 = \boldsymbol{r}_0^T\,\boldsymbol{w}_0^{akk.}$	$\rightarrow$	$\gamma_0^{akk.}$
Iteration für $m = 1, 2, \ldots$		
J	$\rightarrow$ $\rightarrow$ $\dfrac{\|\boldsymbol{r}_{m-1}\|_2}{\|\boldsymbol{r}_o\|_2} > \varepsilon$	N
$\boldsymbol{v}_m = \boldsymbol{A}\,\boldsymbol{s}_{m-1}^{akk.}$		STOP
$\sigma_m = \boldsymbol{v}_m^T\,\boldsymbol{w}_{m-1}^{akk.}$	$\rightarrow$	$\sigma_m^{akk.}$
$\alpha_m^{akk.} = \dfrac{\gamma_{m-1}^{akk.}}{\sigma_m^{akk.}}$		
$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\,\boldsymbol{s}_{m-1}^{akk.}$		
$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} - \alpha_m^{akk.}\,\boldsymbol{v}_m$	$\rightarrow$	$\boldsymbol{w}_m^{akk.}$
$\gamma_m = \boldsymbol{r}_m^T\,\boldsymbol{w}_m^{akk.}$	$\rightarrow$	$\gamma_m^{akk.}$
$\beta^{akk.} = \dfrac{\gamma_m^{akk.}}{\gamma_{m-1}^{akk.}}$		
$\boldsymbol{s}_m^{akk.} = \boldsymbol{w}_m^{akk.} - \beta^{akk.}\,\boldsymbol{s}_{m-1}^{akk.}$		

3.1.2 Parallelisierung des stabilisierten BiCGSTAB-Verfahrens

Das Verhältnis der Rechenzeiten zwischen den sequentiellen Verfahren (stabilisiertes BiCGSTAB- zu CG-Verfahren) wird auf paralleler Ebene erhöht, da der Algorithmus 3-2 gegenüber dem Algorithmus 3-1 vier Kommunikationen zusätzlich pro Iterationsschritt, also insgesamt acht Typumwandlungen benötigt.

Algorithmus 3-2: Paralleles stabilisiertes BiCGSTAB-Verfahren

$\boldsymbol{x}_0^{akk.} = 0\,; \quad \varepsilon = 0{,}001; \quad \tilde{\varepsilon} = 10^{-9}$			
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0 \quad \rightarrow \quad \boldsymbol{p}_0^{akk.}$			
Iteration für m = 1, 2, ... Solange $\rightarrow \rightarrow \dfrac{\lVert \boldsymbol{r}_{m-1}^{akk.} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$			
$\boldsymbol{v}_m = \boldsymbol{A}\,\boldsymbol{p}_{m-1}^{akk.} \quad \rightarrow \quad \boldsymbol{v}_m^{akk.}$			
$\sigma_m = \boldsymbol{v}_m^T\,\boldsymbol{p}_0^{akk.} \quad \rightarrow \quad \sigma_m^{akk.}$			
J	m = = 1		N
$\alpha_m = \dfrac{\boldsymbol{p}_{m-1}^{akk.T}\,\boldsymbol{r}_0}{\sigma_m^{akk.}} \quad \rightarrow \quad \alpha_m^{akk.}$		$\alpha_m = \dfrac{\boldsymbol{r}_{m-1}^{akk.T}\,\boldsymbol{r}_0}{\sigma_m^{akk.}} \quad \rightarrow \quad \alpha_m^{akk.}$	
	m = = 1		
$\boldsymbol{s}_m^{akk.} = \boldsymbol{p}_{m-1}^{akk.} - \alpha_m^{akk.}\,\boldsymbol{v}_m^{akk.}$		$\boldsymbol{s}_m^{akk.} = \boldsymbol{r}_{m-1}^{akk.} - \alpha_m^{akk.}\,\boldsymbol{v}_m^{akk.}$	
J	$\rightarrow \quad \lVert \boldsymbol{s}_m^{akk.} \rVert_2 > \varepsilon$		N
$\boldsymbol{t}_m = \boldsymbol{A}\,\boldsymbol{s}_m^{akk.} \quad \rightarrow \quad \boldsymbol{t}_m^{akk.}$			$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\,\boldsymbol{p}_{m-1}^{akk.}$
$Z_m = \boldsymbol{t}_m^T\,\boldsymbol{s}_m^{akk.}$			$\boldsymbol{r}_m^{akk.} = \boldsymbol{s}_{m-1}^{akk.}$
$N_m = \boldsymbol{t}_m^T\,\boldsymbol{t}_m^{akk.}$			$m = m + 1$
$\rightarrow \quad \omega_m^{akk.} = \dfrac{Z_m^{akk.}}{N_m^{akk.}}$			
$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\,\boldsymbol{p}_{m-1}^{akk.} + \omega_m^{akk.}\,\boldsymbol{s}_m^{akk.}$			
$\boldsymbol{r}_m^{akk.} = \boldsymbol{s}_m^{akk.} - \omega_m^{akk.}\,\boldsymbol{t}_m^{akk.}$			
$Z2_m = \boldsymbol{r}_m^{akk.T}\,\boldsymbol{r}_0$			
J	m = = 1		N
$N2_m = \boldsymbol{p}_{m-1}^{akk.T}\,\boldsymbol{r}_0$		$N2_m = \boldsymbol{r}_m^{akk.T}\,\boldsymbol{r}_0$	

	$\rightarrow \quad \delta_m^{akk.} = \dfrac{Z2_m^{akk.}}{N2_m^{akk.}}$	
	$\beta_m^{akk.} = \dfrac{\alpha_m^{akk.}}{\omega_m^{akk.}}\,\delta_m^{akk.}$	
	$\boldsymbol{p}_m^{akk.} = \boldsymbol{r}_m^{akk.} - \beta_m^{akk.}\left(\boldsymbol{p}_{m-1}^{akk.} - \omega_m^{akk.}\,\boldsymbol{v}_m^{akk.}\right)$	
	$m = m+1$	

3.1.3 Parallelisierung des präkonditionierten CG-Verfahrens

Die Kommunikationsanforderungen des parallelen PCG-Verfahrens (Algorithmus 3-3) ändern sich gegenüber dem CG-Verfahren nicht.

Algorithmus 3-3: Parallelisiertes PCG-Verfahren

$\boldsymbol{x}_0^{akk.} = 0\,; \quad \varepsilon = 0{,}001$			
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$			
$\hat{\boldsymbol{r}}_0 = \boldsymbol{P}\,\boldsymbol{r}_0$	$\rightarrow$	$\hat{\boldsymbol{w}}_0^{akk.}$	
$\hat{\boldsymbol{s}}_0^{akk.} = \hat{\boldsymbol{w}}_0^{akk.}$			
$\gamma_0 = \boldsymbol{r}_0^T\,\hat{\boldsymbol{w}}_0^{akk.}$	$\rightarrow$	$\gamma_0^{akk.}$	
Iteration für $m = 1, 2, ..., n$			
J	$\rightarrow$ $\rightarrow$	$\dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$	N
$\boldsymbol{v}_m = \boldsymbol{A}\,\hat{\boldsymbol{s}}_{m-1}^{akk.}$			STOP
$\sigma_m = \boldsymbol{v}_m^T\,\hat{\boldsymbol{s}}_{m-1}^{akk.}$	$\rightarrow$	$\sigma_m^{akk.}$	
$\alpha_m^{akk.} = \dfrac{\gamma_{m-1}^{akk.}}{\sigma_m^{akk.}}$			
$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} - \alpha_m^{akk.}\,\hat{\boldsymbol{s}}_{m-1}^{akk.}$			
$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} + \alpha^{akk.}\,\boldsymbol{v}_m$			
$\hat{\boldsymbol{r}}_m = \boldsymbol{P}\,\boldsymbol{r}_m$	$\rightarrow$	$\hat{\boldsymbol{w}}_m^{akk.}$	
$\gamma_m = \boldsymbol{r}_m^T\,\hat{\boldsymbol{w}}_m^{akk.}$	$\rightarrow$	$\gamma_m^{akk.}$	

	$\beta_m^{akk.} = \dfrac{\gamma_m^{akk.}}{\gamma_{m-1}^{akk.}}$	
	$\hat{\boldsymbol{s}}_m^{akk.} = \hat{\boldsymbol{w}}_m^{akk.} + \beta_m^{akk.}\, \hat{\boldsymbol{s}}_{m-1}^{akk.}$	

3.1.4 Parallelisierung des präkonditionierten stabilisierten BiCGSTAB-Verfahrens

Beim parallelen stabilisierten BiCGSTAB-Verfahren fordert die Präkonditionierung eine zusätzliche Vektortypumwandlung im Vergleich zur nicht präkonditionierten Version (Algorithmus 3-2). Die insgesamt neun benötigten Kommunikationen werden im nachfolgenden Algorithmus verdeutlicht.

Algorithmus 3-4: Paralleles präkonditioniertes stabilisiertes BiCGSTAB-Verfahren

$\boldsymbol{x}_0 = 0;\quad \varepsilon = 0{,}001;\quad \tilde{\varepsilon} = 10^{-9}$			
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}_0$			
$\hat{\boldsymbol{p}}_0 = \boldsymbol{P}\,\boldsymbol{r}_0$	$\rightarrow$	$\hat{\boldsymbol{w}}_0^{akk.}$	
$\hat{\boldsymbol{p}}_0^{akk.} = \hat{\boldsymbol{w}}_0^{akk.}$			
Iteration für m = 1, 2, ... Solange	$\rightarrow$ $\rightarrow$	$\dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$	
$\boldsymbol{v}_m = \boldsymbol{A}\,\hat{\boldsymbol{p}}_{m-1}^{akk.}$			
$\hat{\boldsymbol{v}}_m = \boldsymbol{P}\,\boldsymbol{v}_m$	$\rightarrow$	$\hat{\boldsymbol{v}}_m^{akk.}$	
$\sigma_m = \hat{\boldsymbol{v}}_m^T\,\hat{\boldsymbol{w}}_0^{akk.}$	$\rightarrow$	$\sigma_m^{akk.}$	
J	$m = = 1$		N
$\alpha_m = \dfrac{\hat{\boldsymbol{w}}_{m-1}^{akk.T}\,\hat{\boldsymbol{p}}_0}{\sigma_m^{akk.}}$	$\rightarrow$	$\alpha_m^{akk.}$	$\alpha_m = \dfrac{\hat{\boldsymbol{r}}_{m-1}^T\,\hat{\boldsymbol{p}}_0}{\sigma_m^{akk.}}$ $\rightarrow$ $\alpha_m^{akk.}$
$\boldsymbol{s}_m = \boldsymbol{r}_{m-1} - \alpha_m^{akk.}\,\boldsymbol{v}_m$			
$\hat{\boldsymbol{s}}_m = \boldsymbol{P}\,\boldsymbol{s}_m$	$\rightarrow$	$\hat{\boldsymbol{s}}_m^{akk.}$	
J	$\rightarrow$	$\lVert \hat{\boldsymbol{s}}_m \rVert_2 > \tilde{\varepsilon}$	N
$\boldsymbol{t}_m = \boldsymbol{A}\,\hat{\boldsymbol{s}}_m^{akk.}$			$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\,\boldsymbol{p}_{m-1}^{akk.}$

	$\hat{\boldsymbol{t}}_m = \boldsymbol{P}\,\boldsymbol{t}_m \quad \rightarrow \quad \hat{\boldsymbol{t}}_m^{akk.}$	$\boldsymbol{r}_m = \boldsymbol{s}_m$
	$Z_m = \hat{\boldsymbol{t}}_m^T\,\hat{\boldsymbol{s}}_m^{akk.}$	$m = m + 1$
	$N_m = \hat{\boldsymbol{t}}_m^T\,\hat{\boldsymbol{t}}_m^{akk.}$	
	$\rightarrow \quad \omega_m^{akk.} = \dfrac{Z_m^{akk.}}{N_m^{akk.}}$	
	$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\,\boldsymbol{p}_{m-1}^{akk.} + \omega_m^{akk.}\,\hat{\boldsymbol{s}}_m^{akk.}$	
	$\boldsymbol{r}_m = \boldsymbol{s}_m - \omega_m^{akk.}\,\boldsymbol{t}_m$	
	$\hat{\boldsymbol{r}}_m^{akk.} = \hat{\boldsymbol{s}}_m^{akk.} - \omega_m^{akk.}\,\hat{\boldsymbol{t}}_m^{akk.}$	
	$Z2_m = \hat{\boldsymbol{r}}_m^{akk.T}\,\boldsymbol{p}_0$	
	J: $m = = 1$	N
	$\gamma = \hat{\boldsymbol{w}}_{m-1}^{akk.T}\,\boldsymbol{p}_0$	$\gamma = \hat{\boldsymbol{r}}_{m-1}^T\,\boldsymbol{p}_0$
	$\rightarrow \quad \delta_m^{akk.} = \dfrac{Z2_m^{akk.}}{\gamma^{akk.}}$	
	$\beta_m^{akk.} = \dfrac{\alpha_m^{akk.}}{\omega_m^{akk.}}\,\delta_m^{akk.}$	
	$\boldsymbol{p}_m^{akk.} = \hat{\boldsymbol{r}}_m^{akk.} + \beta_m^{akk.}\left(\boldsymbol{p}_{m-1}^{akk.} - \omega_m^{akk.}\,\hat{\boldsymbol{v}}_m^{akk.}\right)$	
	$m = m+1$	

3.2 Die Schurkomplement-Methode

In den aus den Kapiteln 3.1.1 bis 3.1.4 beschriebenen Algorithmen wird bei der Vektortypumwandlung der gesamte Vektor, inklusive der lokalen Anteile, an den Host-Agenten übergeben. Die Schurkomplement-Methode, die sich die Blockstruktur der Steifigkeitsmatrix $\boldsymbol{A}$ zu Nutze macht, ermöglicht die Beschränkung der Kommunikation auf die Koppelfreiheitsgrade. Das heißt, dass die Kommunikation sich nur noch auf die tatsächlich aufzusummierenden Werte reduziert. Dadurch wird der Parallelisierungsgrad (10) erhöht und die Iterationszahl verkleinert.

Die Schurkomplement-Methode benutzt die iterativen Verfahren, um lediglich die Lösung der Koppelknoten zu liefern. Die Lösung der rein lokalen Knotenfreiheitsgrade (Freiheitsgrade der

Knoten innerhalb eines Teilgebiets) wird vollkommen parallel (ohne Kommunikation) und direkt mit Hilfe der zuvor berechneten Verschiebungsvektoren der Koppelknoten bestimmt.

Wie in Kapitel 3.1 schon erwähnt, erhält das Gleichungssystem $\boldsymbol{Ax} = \boldsymbol{b}$ durch die Gebietszerlegung eine Blockstruktur (3.1). Für die Teilgebiete sehen die Gleichungssysteme entsprechend aus:

$$\underbrace{\begin{pmatrix} \boldsymbol{A}^{ss} & \boldsymbol{A}^{sc} \\ \boldsymbol{A}^{cs} & \boldsymbol{A}^{cc} \end{pmatrix}}_{\boldsymbol{A}^s} \underbrace{\begin{pmatrix} \boldsymbol{x}^s \\ \boldsymbol{x}^c \end{pmatrix}}_{\boldsymbol{x}^s} = \underbrace{\begin{pmatrix} \boldsymbol{b}^s \\ \boldsymbol{b}^c \end{pmatrix}}_{\boldsymbol{b}^s} \forall s,\ s = 1, \ldots, p \tag{3.4}$$

Eliminiert man durch Umformung den Unbekanntenvektor $\boldsymbol{x}^s$ aus dem Teilgleichungssystem (3.4), erhält man:

$$\underbrace{\left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\, \boldsymbol{A}^{ss^{-1}}\, \boldsymbol{A}^{sc}\right)}_{\boldsymbol{S}} \boldsymbol{x}^c = \underbrace{\boldsymbol{b}^c - \boldsymbol{A}^{cs}\, \boldsymbol{A}^{ss^{-1}}\, \boldsymbol{b}^s}_{\bar{\boldsymbol{b}}^c} \quad \forall s,\ s = 1, \ldots, p \tag{3.5}$$

$\boldsymbol{S}$ wird als Schurkomplement-Matrix bezeichnet und $\bar{\boldsymbol{b}}^c$ stellt die modifizierte rechte Seite des Gleichungssystems (3.5) dar.

Nachdem das „Schurkomplement-System“ $\boldsymbol{S}\, \boldsymbol{x}^c = \bar{\boldsymbol{b}}^c$ gelöst ist, können die Verschiebungsvektoren $\boldsymbol{x}^s$ im Teilgebietsinneren durch einsetzen von $\boldsymbol{x}^c$ in die obere Gleichung des Teilgleichungssystems (3.4) ohne Kommunikation gelöst werden:

$$\boldsymbol{x}^s = \boldsymbol{A}^{ss^{-1}} \left(\boldsymbol{b}^s - \boldsymbol{A}^{sc}\, \boldsymbol{x}^c\right). \tag{3.6}$$

Um die eben beschriebene Vorgehensweise in eine Programmiersprache umzusetzen, müssen zuvor die einzelnen Blockmatrizen $\boldsymbol{A}^{ss}$, $\boldsymbol{A}^{cc}$, $\boldsymbol{A}^{cs}$, $\boldsymbol{A}^{sc}$ und Vektoren $\boldsymbol{b}^s$, $\boldsymbol{b}^c$, $\boldsymbol{x}_o^s$, $\boldsymbol{x}_o^c$ zusätzlich zur Steifigkeitsmatrix $\boldsymbol{A}$ einzeln aufgestellt werden. Dadurch wird der Speicheraufwand erhöht.

Die Aufstellung der einzelnen Blockmatrizen und Vektoren erfolgt mit Hilfe der Klasse *Zuordnungszeile*, die in Kapitel 7 beschrieben wird.

In den nachfolgenden Struktogrammen wird auf die explizite Indizierung der Ergebnisse als Koppeldaten verzichtet, da sich die Algorithmen dieses Kapitels nur auf die Berechnung der Koppelknoten-Verschiebungen beschränken, und die Verschiebungen der lokalen Knoten durch Gleichung (3.6) ermittelt werden.

3.2.1 Anwendung des Schurkomplements auf die Verfahren der konjugierten Gradienten

Die Berechnung des Schurkomplementprodukts ist am rechenintensivsten pro Iteration, läuft jedoch parallel. Dabei ist $\boldsymbol{v}$, als Produkt aus Schurkomplement und $\boldsymbol{s}$, ein additiv verteilter Vektor, da auf dem Prozessor nur die Anteile dieses Vektors vorliegen, die sich aus den Steifigkeitsbeziehungen des entsprechenden Teilgebiets ergeben.

Für das CG- und PCG-Verfahren ist keine Vektortypumwandlung von $\boldsymbol{v}$ notwendig, da dieser Vektor nur zur Berechnung von σ und zur Aktualisierung von $\boldsymbol{r}_m$ (ein additiv verteilter Vektor) benötigt wird. Das Produkt σ nimmt als Skalar für eine Akkumulation eine geringere Kommunikationszeit in Anspruch als eine vorherige Vektortypumwandlung von $\boldsymbol{v}$.

Die Verfahren Schur-StabBiCGSTAB und Schur-PStabBiCGSTAB benötigen jedoch eine Vektortypumwandlung von $\boldsymbol{v}$ zur Berechnung der akkumulierten Vektoren $\boldsymbol{s}$ und $\boldsymbol{p}$.

Die übrigen Rechenschritte werden analog zu den Algorithmen aus Kapitel 3 durchgeführt. Nachfolgend werden die vier Algorithmen zur Schurkomplement-Methode anhand von Struktogrammen dargestellt.

- Anwendung des Schurkomplements auf das CG-Verfahren:

Algorithmus 3-5: Schur-CG-Verfahren

$\boldsymbol{x}_0^{akk.} = 0\,; \quad \varepsilon = 0{,}001$	
$\boldsymbol{r}_0 = \boldsymbol{b}^c - \boldsymbol{A}^{cc}\,\boldsymbol{x}_0^c - \boldsymbol{A}^{cs}\,\boldsymbol{A}^{ss^{-1}}\,\boldsymbol{b}^s \quad \rightarrow \quad \boldsymbol{w}_0^{akk.}$	
$\boldsymbol{s}_0^{akk.} = \boldsymbol{w}_0^{akk.}$	
$\gamma_0 = \boldsymbol{r}_0^T\,\boldsymbol{w}_0^{akk.} \quad \rightarrow \quad \gamma_0^{akk.}$	
Iteration für m = 1, 2, ...	
J $\rightarrow$ $\rightarrow$ $\dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$ N	
$\boldsymbol{v}_m = \left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\,\boldsymbol{A}^{ss^{-1}}\,\boldsymbol{A}^{sc}\right)\boldsymbol{s}_{m-1}^{akk.}$	STOP
$\sigma_m = \boldsymbol{v}_m^T \boldsymbol{w}_{m-1}^{akk.} \quad \rightarrow \quad \sigma_m^{akk.}$	
$\alpha_m^{akk.} = \dfrac{\gamma_{m-1}^{akk.}}{\sigma_m^{akk.}}$	

$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\, \boldsymbol{s}_{m-1}^{akk.}$	
$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} - \alpha_m^{akk.}\, \boldsymbol{v}_m \quad \rightarrow \quad \boldsymbol{w}_m^{akk.}$	
$\gamma_m = \boldsymbol{r}_m^T\, \boldsymbol{w}_m^{akk.} \quad \rightarrow \quad \gamma_m^{akk.}$	
$\beta_m^{akk.} = \dfrac{\gamma_m^{akk.}}{\gamma_{m-1}^{akk.}}$	
$\boldsymbol{s}_m^{akk.} = \boldsymbol{w}_m^{akk.} - \beta_m^{akk.}\, \boldsymbol{s}_{m-1}^{akk.}$	

- Anwendung des Schurkomplements auf das stabilisierte BiCGSTAB-Verfahren:

Algorithmus 3-6: Schur-StabBiCGSTAB-Verfahren

$\boldsymbol{x}_0^{akk.} = 0\,; \quad \varepsilon = 0{,}001; \quad \tilde{\varepsilon} = 10^{-9}$		
$\boldsymbol{r}_0 = \boldsymbol{b}^c - \boldsymbol{A}^{cc}\, \boldsymbol{x}_0^c - \boldsymbol{A}^{cs}\, \boldsymbol{A}^{ss^{-1}}\, \boldsymbol{b}^s \quad \rightarrow \quad \boldsymbol{p}_0^{akk.}$		
Iteration für m = 1, 2, ... Solange $\rightarrow \quad \rightarrow \quad \dfrac{\left\lVert \boldsymbol{r}_{m-1}^{akk.} \right\rVert_2}{\left\lVert \boldsymbol{r}_o \right\rVert_2} > \varepsilon$		
$\boldsymbol{v}_m = \left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\, \boldsymbol{A}^{ss^{-1}}\, \boldsymbol{A}^{sc}\right) \boldsymbol{p}_{m-1}^{akk.} \quad \rightarrow \quad \boldsymbol{v}_m^{akk.}$		
$\sigma_m = \boldsymbol{v}_m^T\, \boldsymbol{p}_0^{akk.} \quad \rightarrow \quad \sigma_m^{akk.}$		
J	$m = = 1$	N
$\alpha_m = \dfrac{\boldsymbol{p}_{m-1}^{akk.T}\, \boldsymbol{r}_0}{\sigma_m^{akk.}} \quad \rightarrow \quad \alpha_m^{akk.}$	$\alpha_m = \dfrac{\boldsymbol{r}_{m-1}^{akk.T}\, \boldsymbol{r}_0}{\sigma_m^{akk.}} \quad \rightarrow \quad \alpha_m^{akk.}$	
	$m = = 1$	
$\boldsymbol{s}_m^{akk.} = \boldsymbol{p}_{m-1}^{akk.} - \alpha_m^{akk.}\, \boldsymbol{v}_m^{akk.}$	$\boldsymbol{s}_m^{akk.} = \boldsymbol{r}_{m-1}^{akk.} - \alpha_m^{akk.}\, \boldsymbol{v}_m^{akk.}$	
J	$\rightarrow \quad \left\lVert \boldsymbol{s}_m^{akk.} \right\rVert_2 > \varepsilon$	N
$\boldsymbol{t}_m = \left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\, \boldsymbol{A}^{ss^{-1}}\, \boldsymbol{A}^{sc}\right) \boldsymbol{s}_m^{akk.} \quad \rightarrow \quad \boldsymbol{t}_m^{akk.}$		$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\, \boldsymbol{p}_{m-1}^{akk.}$
$Z_m = \boldsymbol{t}_m^T\, \boldsymbol{s}_m^{akk.}$		$\boldsymbol{r}_m^{akk.} = \boldsymbol{s}_{m-1}^{akk.}$
$N_m = \boldsymbol{t}_m^T\, \boldsymbol{t}_m^{akk.}$		$m = m + 1$
$\rightarrow \quad \omega_m^{akk.} = \dfrac{Z_m^{akk.}}{N_m^{akk.}}$		
$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\, \boldsymbol{p}_{m-1}^{akk.} + \omega_m^{akk.}\, \boldsymbol{s}_m^{akk.}$		

	$\boldsymbol{r}_m^{akk.} = \boldsymbol{s}_m^{akk.} - \omega_m^{akk.}\ \boldsymbol{t}_m^{akk.}$	
	$Z2_m = \boldsymbol{r}_m^{akk.T}\ \boldsymbol{r}_0$	
	J — $m = = 1$ — N	
	$N2_m = \boldsymbol{p}_{m-1}^{akk.T}\ \boldsymbol{r}_0$ | $N2_m = \boldsymbol{r}_m^{akk.T}\ \boldsymbol{r}_0$	
	$\rightarrow \quad \delta_m^{akk.} = \dfrac{Z2_m^{akk.}}{N2_m^{akk.}}$	
	$\beta_m^{akk.} = \dfrac{\alpha_m^{akk.}}{\omega_m^{akk.}}\ \delta_m^{akk.}$	
	$\boldsymbol{p}_m^{akk.} = \boldsymbol{r}_m^{akk.} - \beta_m^{akk.}\left(\boldsymbol{p}_{m-1}^{akk.} - \omega_m^{akk.}\ \boldsymbol{v}_m^{akk.}\right)$	
	$m = m+1$	

- Anwendung des Schurkomplements auf das PCG-Verfahren:

Algorithmus 3-7: Schur-PCG-Verfahren

$\boldsymbol{x}_0^{akk.} = 0\,; \quad \varepsilon = 0{,}001$		
$\boldsymbol{r}_0 = \boldsymbol{b}^c - \boldsymbol{A}^{cc}\ \boldsymbol{x}_0^c - \boldsymbol{A}^{cs}\ \boldsymbol{A}^{ss^{-1}}\ \boldsymbol{b}^s$		
$\hat{\boldsymbol{r}}_0 = \boldsymbol{P}\,\boldsymbol{r}_0$	$\rightarrow$	$\hat{\boldsymbol{w}}_0^{akk.}$
$\hat{\boldsymbol{s}}_0^{akk.} = \hat{\boldsymbol{w}}_0^{akk.}$		
$\gamma_0 = \boldsymbol{r}_0^T\ \hat{\boldsymbol{w}}_0^{akk.}$	$\rightarrow$	$\gamma_0^{akk.}$
Iteration für $m = 1, 2, ..., n$		
	J — $\rightarrow \dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$ — N	
	$\boldsymbol{v}_m = \left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\ \boldsymbol{A}^{ss^{-1}}\ \boldsymbol{A}^{sc}\right)\hat{\boldsymbol{s}}_{m-1}^{akk.}$	STOP
	$\sigma_m = \boldsymbol{v}_m^Z \hat{\boldsymbol{s}}_{m-1}^{akk.}$ $\rightarrow$ $\sigma_m^{akk.}$	
	$\alpha_m^{akk.} = \dfrac{\gamma_{m-1}^{akk.}}{\sigma_m^{akk.}}$	
	$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} - \alpha_m^{akk.}\ \hat{\boldsymbol{s}}_{m-1}^{akk.}$	
	$\boldsymbol{r}_m = \boldsymbol{r}_{m-1} + \alpha_m^{akk.}\ \boldsymbol{v}_m$	

	$\hat{\boldsymbol{r}}_m = \boldsymbol{P}\,\boldsymbol{r}_m \quad \rightarrow \quad \hat{\boldsymbol{w}}_m^{akk.}$	
	$\gamma_m = \boldsymbol{r}_m^T\,\hat{\boldsymbol{w}}_m^{akk.} \quad \rightarrow \quad \gamma_m^{akk.}$	
	$\beta_m^{akk.} = \dfrac{\gamma_m^{akk.}}{\gamma_{m-1}^{akk.}}$	
	$\hat{\boldsymbol{s}}_m^{akk.} = \hat{\boldsymbol{w}}_m^{akk.} + \beta_m^{akk.}\,\hat{\boldsymbol{s}}_{m-1}^{akk.}$	

- Anwendung des Schurkomplements auf das präkonditionierte stabilisierte BiCGSTAB-Verfahren:

Algorithmus 3-8: Schur-PStabBiCGSTAB-Verfahren

$\boldsymbol{x}_0 = 0;\quad \varepsilon = 0{,}001;\quad \tilde{\varepsilon} = 10^{-9}$			
$\boldsymbol{r}_0 = \boldsymbol{b}^c - \boldsymbol{A}^{cc}\,\boldsymbol{x}_0^c - \boldsymbol{A}^{cs}\,\boldsymbol{A}^{ss^{-1}}\,\boldsymbol{b}^s$			
$\hat{\boldsymbol{p}}_0 = \boldsymbol{P}\,\boldsymbol{r}_0 \quad \rightarrow \quad \hat{\boldsymbol{w}}_0^{akk.}$			
$\hat{\boldsymbol{p}}_0^{akk.} = \hat{\boldsymbol{w}}_0^{akk.}$			
Iteration für m = 1, 2, ... Solange $\rightarrow \quad \rightarrow \quad \dfrac{\lVert \boldsymbol{r}_{m-1} \rVert_2}{\lVert \boldsymbol{r}_o \rVert_2} > \varepsilon$			
	$\boldsymbol{v}_m = \left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\,\boldsymbol{A}^{ss^{-1}}\,\boldsymbol{A}^{sc}\right)\hat{\boldsymbol{p}}_{m-1}^{akk.}$		
	$\hat{\boldsymbol{v}}_m = \boldsymbol{P}\,\boldsymbol{v}_m \quad \rightarrow \quad \hat{\boldsymbol{v}}_m^{akk.}$		
	$\sigma_m = \hat{\boldsymbol{v}}_m^T\,\hat{\boldsymbol{w}}_0^{akk.} \quad \rightarrow \quad \sigma_m^{akk.}$		
	J	$m = = 1$	N
	$\alpha_m = \dfrac{\hat{\boldsymbol{w}}_{m-1}^{akk.\,T}\,\hat{\boldsymbol{p}}_0}{\sigma_m^{akk.}} \quad \rightarrow \quad \alpha_m^{akk.}$	$\alpha_m = \dfrac{\hat{\boldsymbol{r}}_{m-1}^T\,\hat{\boldsymbol{p}}_0}{\sigma_m^{akk.}} \quad \rightarrow \quad \alpha_m^{akk.}$	
	$\boldsymbol{s}_m = \boldsymbol{r}_{m-1} - \alpha_m^{akk.}\,\boldsymbol{v}_m$		
	$\hat{\boldsymbol{s}}_m = \boldsymbol{P}\,\boldsymbol{s}_m \quad \rightarrow \quad \hat{\boldsymbol{s}}_m^{akk.}$		
	J	$\rightarrow \quad \lVert \hat{\boldsymbol{s}}_m \rVert_2 > \tilde{\varepsilon}$	N
	$\boldsymbol{t}_m = \left(\boldsymbol{A}^{cc} - \boldsymbol{A}^{cs}\,\boldsymbol{A}^{ss^{-1}}\,\boldsymbol{A}^{sc}\right)\hat{\boldsymbol{s}}_m^{akk.}$		$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.}\,\boldsymbol{p}_{m-1}^{akk.}$
	$\hat{\boldsymbol{t}}_m = \boldsymbol{P}\,\boldsymbol{t}_m \quad \rightarrow \quad \hat{\boldsymbol{t}}_m^{akk.}$		$\boldsymbol{r}_m = \boldsymbol{s}_m$

	$Z_m = \hat{\boldsymbol{t}}_m^T \, \hat{\boldsymbol{s}}_m^{akk.}$	$m = m + 1$
	$N_m = \hat{\boldsymbol{t}}_m^T \, \hat{\boldsymbol{t}}_m^{akk.}$	
	$\rightarrow \quad \omega_m^{akk.} = \dfrac{Z_m^{akk.}}{N_m^{akk.}}$	
	$\boldsymbol{x}_m^{akk.} = \boldsymbol{x}_{m-1}^{akk.} + \alpha_m^{akk.} \, \boldsymbol{p}_{m-1}^{akk.} + \omega_m^{akk.} \, \hat{\boldsymbol{s}}_m^{akk.}$	
	$\boldsymbol{r}_m = \boldsymbol{s}_m - \omega_m^{akk.} \, \boldsymbol{t}_m$	
	$\hat{\boldsymbol{r}}_m^{akk.} = \hat{\boldsymbol{s}}_m^{akk.} - \omega_m^{akk.} \, \hat{\boldsymbol{t}}_m^{akk.}$	
	$Z2_m = \hat{\boldsymbol{r}}_m^{akk.T} \, \boldsymbol{p}_0$	
	J $m = = 1$ N	
	$\gamma_m = \hat{\boldsymbol{w}}_{m-1}^{akk.T} \, \boldsymbol{p}_0$ | $\gamma_m = \hat{\boldsymbol{r}}_{m-1}^T \, \boldsymbol{p}_0$	
	$\rightarrow \quad \delta_m^{akk.} = \dfrac{Z2_m^{akk.}}{\gamma_m^{akk.}}$	
	$\beta_m^{akk.} = \dfrac{\alpha_m^{akk.}}{\omega_m^{akk.}} \, \delta_m^{akk.}$	
	$\boldsymbol{p}_m^{akk.} = \hat{\boldsymbol{r}}_m^{akk.} + \beta_m^{akk.} \left(\boldsymbol{p}_{m-1}^{akk.} - \omega_m^{akk.} \, \hat{\boldsymbol{v}}_m^{akk.} \right)$	
	$m = m+1$	

4 Präkonditionierer

4.1 Funktion von Präkonditionierern

In Kapitel 2.3 wurde schon erwähnt, dass die konjugierten Gradientenverfahren mit der größtmöglichen Konvergenzgeschwindigkeit rechnen, wenn sich die Eigenwerte einer Steifigkeitsmatrix nur geringfügig voneinander unterscheiden, das heißt die Konditionszahl $\kappa(\boldsymbol{A}) \approx 1$ ist. Im Idealfall sind alle Eigenwerte gleich ($\kappa(\boldsymbol{A}) = 1$), und die iterativen Verfahren liefern schon im ersten Iterationsschritt die Lösung. Dieser Idealfall kann jedoch nur durch eine Multiplikation der Steifigkeitsmatrix mit ihrer Inversen $\boldsymbol{A}^{-1}$ erzielt werden. Wie bereits erwähnt, würde die Invertierung der Steifigkeitsmatrix, abgesehen vom enormen Speicherplatzbedarf, zu unakzeptablen Rechenzeiten führen. Um die Konditionszahl der Steifigkeitsmatrizen zu verringern, werden reguläre Präkonditionierungsmatrizen eingesetzt, die das Gleichungssystem $\boldsymbol{Ax} = \boldsymbol{b}$ in die äquivalente Form $\boldsymbol{PAx} = \boldsymbol{Pb}$ transformieren. Diese Transformation wird als Linkspräkonditionierung bezeichnet. Neben dieser Technik existieren auch die rechtsseitige und beidseitige Präkonditionierung. Sie sind jedoch nicht so verbreitet wie die Linkspräkonditionierung. Aus [7] geht auch hervor das sich keine gravierenden Unterschiede zwischen einer linksseitigen und rechtsseitigen Präkonditionierung ergeben. Die beidseitige Präkonditionierung wird für symmetrische, positiv definite Matrizen verwendet, da bei einer einseitigen Präkonditionierung unsymmetrische Matrizen entstehen können. Durch die Transformation $\boldsymbol{P}_{\mathrm{L}}\boldsymbol{AP}_{\mathrm{R}}$ (mit $\boldsymbol{P}_R = \boldsymbol{P}_L^T$) werden die Eigenschaften Symmetrie und positive Definitheit bewahrt. Da die skalierenden, unvollständigen und der SSOR-Präkonditionierer (Kapitel 4.2, 4.4 und 4.3.2) die Symmetrie trotz linksseitiger Transformation nicht verletzen, wird auch die beidseitige Präkonditionierungsart nicht näher betrachtet.

Folgende Anforderungen werden an die Präkonditionierungsmatrizen gestellt:

- Einfache Berechnung (Rechenzeit)
- Geringen Speicherplatzbedarf
- Gute Approximation an die Inverse der Steifigkeitsmatrix $\boldsymbol{A}$ → leichte Invertierbarkeit (Effektivität)
- Breites Anwendungsgebiet (Flexibilität und Stabilität)

Bei Erfüllung dieser Anforderungen führt eine Präkonditionierung außerdem zu einer Stabilisierung der numerischen Verfahren.

In den folgenden Kapiteln werden ausgewählte Präkonditionierungstechniken vorgestellt. Eine Auswertung bezüglich oben genannter Eigenschaften und eine vergleichende Bewertung mit Hilfe konkreter Beispiele beinhaltet Kapitel 6.

4.2 Skalierende Präkonditionierer

Für die skalierenden Präkonditionierer (kurz: Skalierung) sprechen neben ihrer leichten Berechnung der geringe Speicherplatzbedarf. Als eventuell auftretender Nachteil lässt sich nur die sehr grobe Approximation an die Inverse $\boldsymbol{A}^{-1}$ und die unter Umständen damit verbundene geringfügige Beschleunigung der Konvergenzgeschwindigkeit nennen.

Aus [7] geht folgende Definition der Skalierung hervor:

„Eine reguläre Diagonalmatrix $\boldsymbol{D} = \mathrm{diag}\{d_{11}, ..., d_{nn}\} \in \mathbb{R}^{n \times n}$ heißt Skalierung“ (190).

Voraussetzung aller Skalierungen ist, dass die Diagonalelemente der Steifigkeitsmatrix nicht Null sind: $a_{ii} \neq 0$. Für alle Skalierungen entspricht die Präkonditionierungsmatrix $\boldsymbol{P}$ der inversen Diagonalmatrix $\boldsymbol{D}^{-1}$. Lediglich die Einträge der Diagonalen unterscheiden sich voneinander.

Die Tatsache, dass die Skalierer nur Diagonaleinträge enthalten, ermöglicht eine Speicherung in Vektorform, wodurch der Speicherplatzbedarf dieser Präkonditionierer auf ein Minimum reduziert wird und die eigentliche Matrix-Vektor-Multiplikation bei der Präkonditionierung als Vektor-Vektor-Multiplikation durchgeführt werden kann.

4.2.1 Skalierung mit dem Diagonalelement (Jakobi-Skalierer)

Dieser Präkonditionierer setzt sich nur aus der inversen Diagonalen von $\boldsymbol{A}$ zusammen,

$$d_{ii} = \frac{1}{a_{ii}} \qquad \text{für } i = 1, ..., n. \tag{4.1}$$

Aufgrund der einfachen Erzeugung von $\boldsymbol{D}^{-1}$, lässt sich ein Gleichungssystem der Form $\hat{\boldsymbol{f}} = \boldsymbol{P}\boldsymbol{f}$ sehr schnell lösen. Dabei ist $\boldsymbol{f}$ ein beliebiger Vektor, der präkonditioniert werden soll.

Man sollte jedoch bemerken, dass die Diagonalskalierung ausschließlich bei Matrizen mit unterschiedlichen Diagonalelementen zu einer Veränderung der Konditionszahl führt.

4.2.2 Zeilen-/Spaltenskalierung bezüglich der Betragssummennorm

Bei dieser Skalierung werden die Beträge der Matrixkoeffizienten zeilen- bzw. spaltenweise aufsummiert. Die Kehrwerte der Summen je Zeile bzw. Spalte werden diagonal geordnet und bilden somit die diagonale Präkonditionierungsmatrix.

$$d_{ii} = \frac{1}{\sum_{j=1}^{n} \left| a_{ij} \right|} \text{ beziehungsweise } d_{jj} = \frac{1}{\sum_{i=1}^{n} \left| a_{ij} \right|} \tag{4.2}$$

für $i = 1, ..., n$ beziehungsweise $j = 1, ..., n$.

4.2.3 Zeilen-/Spaltenskalierung bezüglich der euklidischen Norm

Hier wird die Länge jeder Zeile i bzw. Spalte j ermittelt und die Inverse an die Stelle ii bzw. jj der Präkonditionierungsmatrix gesetzt.

$$d_{ii} = \frac{1}{\left(\sum_{j=1}^{n} \left| a_{ij} \right|^2\right)^{\frac{1}{2}}} \text{ beziehungsweise } d_{jj} = \frac{1}{\left(\sum_{i=1}^{n} \left| a_{ij} \right|^2\right)^{\frac{1}{2}}} \tag{4.3}$$

für $i = 1, ..., n$ beziehungsweise $j = 1, ..., n$.

4.2.4 Zeilen-/Spaltenskalierung bezüglich der Maximumsnorm

Bei der Skalierung bezüglich der Maximumsnorm wird nur der betragsmäßig größte Wert in jeder Zeile bzw. Spalte entnommen. Die Kehrwerte der Maxima werden auf die entsprechende Diagonalstelle der Präkonditionierungsmatrix gesetzt.

$$d_{ii} = \frac{1}{\max_{j=1,...,n} \left| a_{ij} \right|} \text{ beziehungsweise } d_{jj} = \frac{1}{\max_{i=1,...,n} \left| a_{ij} \right|} \tag{4.4}$$

für $i = 1, ..., n$ beziehungsweise $j = 1, ..., n$

4.3 Splitting-assoziierte Präkonditionierer

Die Splitting-Verfahren basieren auf einer Aufteilung der Steifigkeitsmatrix $\boldsymbol{A}$ in folgende Form:

$$\boldsymbol{A} = \boldsymbol{B} + (\boldsymbol{A} - \boldsymbol{B}). \tag{4.5}$$

Wobei $\boldsymbol{B}$ eine leicht invertierbare Approximation an $\boldsymbol{A}$ darstellt und bei der resultierenden Iterationsmatrix $\boldsymbol{M} = \boldsymbol{B}^{-1}(\boldsymbol{B}\text{-}\boldsymbol{A})$ zu einer möglichst kleinen Konditionszahl führen soll. Die Eigenschaften der Matrix $\boldsymbol{B}$ entsprechen den gewünschten Forderungen an Präkonditionierer, so dass $\boldsymbol{B}$ sich als mögliche Präkonditionierungsmatrix für andere iterative Verfahren (hier CG, stabilisiertes BiCGSTAB) anbietet. In [7] wird folgendes definiert:

> „Sei durch $x_{j+1} = Mx_j + Nb$ eine Splitting-Methode zur Lösung von $Ax = b$ mit einer regulären Matrix N gegeben, dann heißt $P = N$ [$N = B^{-1}$] der zur Splitting-Methode assoziierte Präkonditionierer“ (196).

Verglichen mit den Skalierern wird bei den splitting-assoziierten Präkonditionierern eine bessere Approximation an die Steifigkeitsmatrix $\boldsymbol{A}$ erzielt, wodurch eine kleinere Kondition und folglich eine schnellerer Konvergenz erwartet werden darf.

Wie in den Kapiteln 4.3.1 und 4.3.2 zu sehen sein wird, setzten sich die Splitting –assoziierten Präkonditionierer aus der Diagonalmatrix $\boldsymbol{D}$ und den Dreiecksmatrizen $\boldsymbol{L}$ und gegebenenfalls $\boldsymbol{R}$ der Steifigkeitsmatrix $\boldsymbol{A}$ zusammen. Somit kann aufgrund der Diagonal- oder Dreiecksgestalt zur

Invertierung der Matrizen bei der Matrix-Vektor-Multiplikation eine Vorwärts- und Rückwärtselimination durchgeführt werden.

Als Vorraussetzung für diese Präkonditionierer muss die Regularität der Diagonalmatrix $\boldsymbol{D}$ erfüllt sein, $a_{ii} \neq 0$. Die Matrizen $\boldsymbol{L}$ und $\boldsymbol{R}$ haben eine strikte Dreiecksgestalt, das heißt, dass die Diagonalelemente nicht belegt sind.

4.3.1 SOR- und Gauß-Seidel-Präkonditionierer

Das zum assoziierten Präkonditionierer zugehörige SOR-Verfahren (engl. Successive Over-Relaxation) wird auch Überrelaxationsverfahren genannt und das Gauß-Seidel-Verfahren findet man auch unter der Bezeichnung Relaxationsverfahren.

Der SOR-Präkonditionierer ist für das CG-Verfahren nicht anwendbar, da bei seiner Anwendung die Symmetrie der Matrix $\boldsymbol{PA}$ verloren geht. Das heißt, dass der nachfolgend dargestellte Präkonditionierer nur für Gleichungssysteme mit nicht symmetrischen Steifigkeitsmatrizen Verwendung finden kann.

$$\boldsymbol{P}_{SOR} = \omega\left(\boldsymbol{D} + \omega\,\boldsymbol{L}\right)^{-1} \tag{4.6}$$

Der Parameter ω wird als Relaxationsparameter bezeichnet und führt bei optimaler Wahl zu einer Konvergenzverbesserung der iterativen Verfahren, jedoch nur für Werte $0 < \omega < 2$. Für $0 < \omega < 1$ spricht man von einer Unterrelaxation und für $1 < \omega < 2$ von einer Überrelaxation.

Für $\omega = 1$ stimmt der SOR-Präkonditionierer mit dem Gauß-Seidel-Präkonditionierer überein:

$$\boldsymbol{P}_{GS} = \left(\boldsymbol{D} + \boldsymbol{L}\right)^{-1}. \tag{4.7}$$

4.3.2 SSOR- und Symmetrischer Gauß-Seidel-Präkonditionierer

Da das CG-Verfahren die Symmetrie und positive Definitheit der Steifigkeitsmatrix $\boldsymbol{A}$ voraussetzt, muss auch das präkonditionierte System aus einer positiv definiten Matrix $\tilde{\boldsymbol{A}}$ bestehen:

$$\underbrace{\boldsymbol{B}^{-1/2}\,\boldsymbol{A}\,\boldsymbol{B}^{1/2}}_{\tilde{A}}\,\boldsymbol{x} = \boldsymbol{B}^{-1/2}\,\boldsymbol{b}\,. \tag{4.8}$$

Die Erhaltung der Symmetrieeigenschaften wird durch die Anwendung der assoziierten Präkonditionierer von symmetrischen Splitting-Methoden gewährleistet.

Die Variante des im vorigen Kapitel vorgestellten SOR-Präkonditionierers für symmetrische Matrizen (SSOR-Verfahren) sieht folgendermaßen aus:

$$\boldsymbol{P}_{SSOR} = \omega\left(2 - \omega\right)\left(\boldsymbol{D} + \omega\,\boldsymbol{R}\right)^{-1}\boldsymbol{D}\left(\boldsymbol{D} + \omega\,\boldsymbol{L}\right)^{-1}. \tag{4.9}$$

Für ω = 1 stimmt das SSOR-Verfahren mit der symmetrischen Gauß-Seidel-Methode überein:

$$\boldsymbol{P}_{SGS} = \left(\boldsymbol{D} + \boldsymbol{R}\right)^{-1} \boldsymbol{D}\left(\boldsymbol{D} + \boldsymbol{L}\right)^{-1}. \tag{4.10}$$

Der Mehraufwand des SSOR-Präkonditionierers gegenüber dem SOR-Präkonditionierer liegt in der erweiterten Multiplikation mit der Diagonalmatrix und der Rückwärtselimination mit der rechten unteren Dreiecksmatrix ($\boldsymbol{D} + \boldsymbol{R}$), für deren Erzeugung auch zusätzlicher Speicher benötigt wird.

4.4 Unvollständige Zerlegungen

Eine Zerlegung der Steifigkeitsmatrix $\boldsymbol{A}$ in eine linke untere Dreiecksmatrix $\boldsymbol{L}$ und rechte obere Dreiecksmatrix $\boldsymbol{R}$, $\boldsymbol{A} = \boldsymbol{LR}$, ist als Präkonditionierer für Systeme mit schwachbesetzten Matrizen ungeeignet, da die Matrizen $\boldsymbol{L}$ und $\boldsymbol{R}$ weitaus mehr Elemente ungleich Null enthalten als die Ursprungsmatrix und folglich mehr Speicher- und Rechenaufwand erfordern. Aus diesem Grund wurde die unvollständige Zerlegung entwickelt, bei der ein Auffüllen der Dreiecksmatrizen verhindert wird. Durch das Blockieren der Auffüllung (hier wird das Besetzungsmuster der Steifigkeitsmatrix $\boldsymbol{A}$ bei der Ermittlung von $\boldsymbol{L}$ und $\boldsymbol{R}$ beibehalten, das heißt, es „dürfen Einträge ungleich Null nur für Positionen aus dem Muster auftreten“) stimmt die Gleichung $\boldsymbol{A} = \boldsymbol{LR}$ nicht mehr exakt, sondern geht in folgende Form über:

$$\boldsymbol{A} = \boldsymbol{LR} + \boldsymbol{F} \approx \boldsymbol{LR}. \tag{4.11}$$

$\boldsymbol{F}$ bezeichnet die Fehler- bzw. Restmatrix. Unter Vernachlässigung der Fehlermatrix ergibt sich eine leicht zu berechnende Approximation an $\boldsymbol{A}$, deren Inverse einen geeigneten Präkonditionierer liefert:

$$\boldsymbol{P} = \boldsymbol{R}^{-1}\boldsymbol{L}^{-1}. \tag{4.12}$$

Die Inversen der Matrizen $\boldsymbol{L}$ und $\boldsymbol{R}$ müssen jedoch nicht explizit (zu aufwendig und speicherintensiv) bei der Matrix-Vektor-Multiplikation $\hat{\boldsymbol{f}} = \boldsymbol{Pf}$ berechnet werden, sondern nur deren Wirkung auf den zu präkonditionierten Vektor $\boldsymbol{f}$. Dies geschieht auf Basis einer Vorwärts- und anschließenden Rückwärtselimination, welche in den nachfolgenden Algorithmen (Algorithmus 4-1 bis Algorithmus 4-3) dargestellt sind. Aus diesem Grund werden die unvollständigen Zerlegungen auch als implizite Präkonditionierer bezeichnet.

4.4.1 Die unvollständige LU-Zerlegung (ILU)

Das I der Abkürzung ILU steht für Incomplete, das L („Lower“) für die linke untere Dreiecksmatrix und das U („Upper“) für die rechte obere Dreiecksmatrix.

Nachfolgendes Struktogramm verdeutlicht die LU-Zerlegung:

Algorithmus 4-1: ILU-Präkonditionierer

Für $k = 0, ..., < n - 1$	Durchführung der LU-Zerlegung			
	Für $i = k + 1, ..., < n$			
		$a_{ik} = a_{ik} / a_{kk}$		
		Für $j = k + 1, ..., < n$		
		J	$a_{ij} \neq 0$	N
		$a_{ij} = a_{ij} - a_{ik}\, a_{kj}$		$a_{ij} = 0$
Für $k = 0, ..., < n$	Durchführung der Vorwärtselimination			
	Für $i = 0, ..., < k$			
		$f_k = f_k - a_{ki}\, f_i$		
Für $k = n - 1, ..., \geq 0$	Durchführung der Rückwärtselimination			
	Für $i = k + 1, ..., < n$			
		$f_k = f_k - a_{ki}\, \hat{f}_i$		
	$\hat{f}_k = f_k / a_{kk}$			

Bei der LU-Zerlegung werden die beiden Dreiecksmatrizen nicht jeweils einzeln gespeichert, sondern in eine Matrix zusammen abgelegt. Somit wird nur ein zu $\boldsymbol{A}$ identischer Speicherplatzbedarf benötigt. Bei der Vorwärtselimination wird der Rechenschritt $\boldsymbol{f} = \boldsymbol{L}^{-1}\, \boldsymbol{f}$ und bei der Rückwärtselimination der Rechenschritt $\hat{\boldsymbol{f}} = \boldsymbol{U}^{-1}\, \boldsymbol{f}$ durchgeführt.

4.4.2 Die unvollständige Cholesky-Zerlegung (IC)

Die in Kapitel 4.4.1 vorgestellte ILU-Zerlegung kann sowohl für nicht symmetrische als auch für symmetrische, positiv definite Steifigkeitsmatrizen $\boldsymbol{A}$ als Präkonditionierer angewendet werden. Bei Vorliegen einer symmetrischen Matrix lässt sich jedoch zusätzlich der Rechenaufwand und Speicherbedarf der ILU-Zerlegung reduzieren, indem die Symmetrieeigenschaften ausgenutzt werden. Die Zerlegung von $\boldsymbol{A}$ lässt sich dann wie folgt formulieren:

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}} + \boldsymbol{F}. \tag{4.13}$$

Der zugehörige Präkonditionierer erhält dann eine zu Gleichung (4.12) analoge Form:

$$\boldsymbol{P} = \boldsymbol{L}^{-T}\boldsymbol{L}^{-1}. \tag{4.14}$$

Der nachfolgend dargestellte Algorithmus wird als unvollständige Cholesky-Zerlegung (engl. Incomplete Cholesky) bezeichnet:

Algorithmus 4-2: IC-Präkonditionierer

Für $k = 0, ..., < n$	Durchführung der Cholesky-Zerlegung		
	Für $j = 0, ..., < k$		
		$a_{kk} = a_{kk} - a_{kj}\, a_{kj}$	
	$a_{kk} = \sqrt{a_{kk}}$		
	Für $i = k + 1, ..., < n$		
	J	$a_{ik} \neq 0$	N
	Für $j = 0, ..., <k$		$a_{ik} = 0$
		$a_{ik} = a_{ik} - a_{ij}\, a_{kj}$	
	$a_{ik} = a_{ik} / a_{kk}$		
Für $k = 0, ..., < n$	Durchführung der Vorwärtselimination		
	Für $i = 0, ..., < k$		
		$f_k = f_k - f_i\, a_{ki}$	
	$f_k = f_k / a_{kk}$		
Für $k = n - 1, ..., \geq 0$	Durchführung der Rückwärtselimination		
	Für $i = k + 1, ..., < n$		
		$f_k = f_k - a_{ik}\, \hat{f}_i$	
	$\hat{f}_k = f_k / a_{kk}$		

Auch hier wird, wie in Kapitel 4.4.1, nur ein zu $\boldsymbol{A}$ identischer Speicherplatzbedarf benötigt und die Vorwärtselimination $\boldsymbol{f} = \boldsymbol{L}^{-1}\boldsymbol{f}$ stimmt exakt überein. Bei der Rückwärtselimination wird statt einer explizit berechneten oberen Dreiecksmatrix der Rechenschritt $\hat{\boldsymbol{f}} = \boldsymbol{L}^{-T}\boldsymbol{f}$ mit der linken unteren Dreiecksmatrix (Symmetrieausnutzung) durchgeführt.

4.4.3 Die unvollständige QR-Zerlegung (IQR)

Auch bei dieser Zerlegung wird bei der Erstellung einer regulären Matrix $\boldsymbol{Q}$ ($\boldsymbol{Q}$ ist eine orthogonale Matrix) und $\boldsymbol{R}$ die Besetzungsstruktur der Steifigkeitsmatrix $\boldsymbol{A}$ beibehalten mit der Folge, dass eine Fehlermatrix entsteht:

$$\boldsymbol{A} = \boldsymbol{QR} + \boldsymbol{F}. \tag{4.15}$$

Der Vorteil dieser unvollständigen Zerlegung liegt in der Orthogonalität der Matrix $\boldsymbol{Q}$. Da eine orthogonale Matrix immer eine Konditionszahl $\kappa(\boldsymbol{Q}) = 1$ besitzt, wird bei dieser Zerlegung, im Gegensatz zu den in Kapitel 4.4.1 und 4.4.2 vorgestellten Varianten, die Kondition der Ursprungsmatrix $\boldsymbol{A}$ nicht verschlechtert. Somit hat die Matrix $\boldsymbol{R}$ die gleiche Kondition wie die Steifigkeitsmatrix $\boldsymbol{A}$.

Der zugehörige IQR-Präkonditionierer ist:

$$\boldsymbol{P} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{\mathrm{T}}. \tag{4.16}$$

Nachfolgend wird die unvollständige QR-Zerlegung in Form des Gram-Schmidt-Verfahrens dargestellt:

Algorithmus 4-3: IQR-Präkonditionierer

- Für $k = 0, ..., < n$ — Durchführung der QR-Zerlegung
  - Für $i = 0, ..., < k$
    - $r_{ik} = 0$
    - $a_{ik} \neq 0$
      - J:
        - Für $j = 0, ..., <n$
          - $r_{ik} = r_{ik} + a_{ji}\, a_{jk}$
      - N:
        - $r_{ik} = 0$
  - Für $i = 0, ..., < k$
    - Für $j = 0, ..., <n$
      - $a_{jk} \neq 0$
        - J:
          - $a_{jk} = a_{jk} - r_{ik}\, a_{ji}$
        - N:
          - $a_{jk} = 0$
    - $r_{kk} = 0$
    - Für $j = 0, ..., <n$
      - $r_{kk} = r_{kk} + a_{jk}\, a_{jk}$
    - $r_{kk} = \sqrt{r_{kk}}$

		Für $j = 0, ..., <n$
		$a_{jk} = a_{jk} / r_{kk}$
Für $k = 0, ..., < n$		Durchführung der Matrixmultiplikation $\tilde{\boldsymbol{f}} = \boldsymbol{Q}^T \boldsymbol{f}$
	$\tilde{f}_k = 0$	
	Für $i = 0, ..., < n$	
		$\tilde{f}_k = \tilde{f}_k + a_{ik} f_i$
Für $k = n - 1, ..., \geq 0$		Durchführung der Rückwärtselimination $\hat{\boldsymbol{f}} = \boldsymbol{R}^{-1} \tilde{\boldsymbol{f}}$
	Für $i = k + 1, ..., < n$	
		$\tilde{f}_k = \tilde{f}_k - r_{ki} \hat{f}_i$
	$\hat{f}_k = \tilde{f}_k / r_{kk}$	

Da $\boldsymbol{Q}$ keine Dreiecksgestalt hat, wird zur Ermittlung von $\boldsymbol{R}$ und $\boldsymbol{Q}$ jeweils zusätzlicher Speicherplatz angefordert. Nach Aussage von [7] liegt der Speicherbedarf geringfügig über dem 1,5-fachen der Steifigkeitsmatrix.

Der Rechenaufwand ist im Vergleich zum Algorithmus 4-1 bei einer großen Anzahl von Unbekannten nach [7] ungefähr dreimal so hoch.

Der Algorithmus 4-3 ist eine modifizierte Form der QR-Zerlegung. Bei der nicht modifizierten QR-Zerlegung geht mit steigender Iterationszahl die Orthogonaltät der Matrix $\boldsymbol{Q}$ vollkommen verloren und mit ihr auch die Verbesserung der Konditionseigenschaften. Dies würde dazu führen, dass bei Anwendung dieses Präkonditionierers die iterativen Verfahren verschlechtert werden.
Trotz der Modifikation des Algorithmus 4-3 geht aufgrund der unvollständigen Zerlegung die Orthogonalität der Matrix $\boldsymbol{Q}$ ebenfalls verloren, so dass mit $\boldsymbol{Q}^T$ auch nur eine Approximation an $\boldsymbol{Q}^{-1}$ erzielt wird. Nach einem Testlauf hat sich herausgestellt, dass mit steigender Zahl der Unbekannten die Abweichung von der Qrthogonalität ansteigt und das verwendete iterative Verfahren sogar instabil wird.
Somit ist die IQR-Zerlegung als unvollständiger Präkonditionierer ungeeignet.

4.5 Sonstige Präkonditionierer

4.5.1 Die unvollständige LU-Zerlegung mit fill-in (ILUT)

Bei der ILU-Zerlegung (Kapitel 4.4.1) kann das Anpassen der Dreieckmatrizen an die Besetzungsstruktur der Steifigkeitsmatrix zu einer so groben Approximation führen, dass die iterativen Verfahren bezüglich der Rechengeschwindigkeit nicht verbessert, sondern sogar verschlechtert werden.

Um eine bessere Approximation an die Matrix $\boldsymbol{A}$, und damit eine stabilere Form der ILU-Zerlegung, zu erzielen, kann bei der unvollständigen LU-Zerlegung ein fill-in (Auffüllen der Matrix) erlaubt werden. Hierbei wird die Abfrage $a_{ij} \neq 0$ in Kapitel 4.4.1 durch folgende ersetzt:

$$a_{ij} \geq \left\| \boldsymbol{a}_i \right\|_2 \tau \tag{4.17}$$

τ stellt einen Toleranzfaktor dar und kann zwischen 0 und 1 gewählt werden. Aus diesem Grund wird der Abkürzung ILU das „T“ angehängt. $\left\| \boldsymbol{a}_i \right\|_2$ entspricht der Länge der i-ten Reihe der Matrix.

Nachdem die neuen Werte $a_{ij} = a_{ij} - a_{ik}\, a_{kj}$ berechnet wurden, wird die i-te Reihe erneut der Abfrage $a_{ij} \geq \left\| \boldsymbol{a}_i \right\|_2 \tau$ unterzogen ($\left\| \boldsymbol{a}_i \right\|_2$ wird nicht mit den neuen Werten berechnet). Bei einem letzten Durchlauf der Reihe i werden die p größten Werte (p kann frei gewählt werden) als fill-in belassen und die restlichen auf Null gesetzt. Dadurch wird gewährleistet, dass die Dreiecksmatrizen nicht zu stark aufgefüllt werden.

Die exaktere Approximation an die Steifigkeitsmatrix erfordert, neben der aufwendigen Implementierung, eine erhöhte Rechenzeit und zusätzlichen Speicherplatz. So dass die ILUT-Version nur angewendet werden sollte, falls die Präkonditionierung mit ILU (Kapitel 4.4.1) eine schlechtere Iterationszahl liefert als der zugehörige Grundalgorithmus (CG, stabilisierter BiCGSTAB). Zudem wird in [7] darauf hingewiesen, dass sich die ILUT nur für Matrizen mit einer kleinen Bandbreite eignet. Da bei sehr großen Gleichungssystemen zur Netzverfeinerung die p-Version angewendet wird, verliert die Steifigkeitsmatrix ihre Bandbreitenstruktur, so dass die in diesem Kapitel vorgestellte Präkonditionierung voraussichtlich eine unzureichende Konvergenzgeschwindigkeit erzielt.

4.5.2 Die modifizierte ILU-Zerlegung (MILU)

Um eine verbesserte Approximation an $\boldsymbol{A}$ zu bekommen, ist bei der modifizierten ILU-Zerlegung der Aufwand geringer als bei ILUT. Die MILU-Zerlegung wird folgendermaßen umgesetzt:

Im Gegensatz zur ILU-Zerlegung werden bei der MILU-Zerlegung die Einträge, die außerhalb des Besetzungsmusters von $\boldsymbol{A}$ liegen, berechnet und mit einem Vorfaktor $0 < \omega < 1$ auf den

Diagonaleintrag der jeweiligen Zeile / Spalte aufaddiert. Um jedoch diese Einträge zu bekommen, müsste zunächst eine vollständige LU-Zerlegung durchgeführt und extra gespeichert werden. So dass diese Variante bezüglich des Rechenaufwands zur Erzeugung des Präkonditionierers ineffizient ist und deshalb nicht implementiert wurde.

4.5.3 Die unvollständige Frobenius-Inverse

Die Definition einer Frobenius-Inversen lautet [7]:

„Ein Muster M heißt regulär im $\mathbb{R}^{n \times n}$, wenn

$$P_M = \{P \in \mathbb{R}^{n \times n} \mid P \text{ ist regular und } M^P = M\} \neq \emptyset$$

gilt. Vorausgesetzt, daß zu gegebenem regulären Muster M das folgende Minimum existiert, dann heißt

$$P_M^R \in \arg\min_{P \in P_M} \left\| AP - I \right\|_F^2$$

unvollständige Frobenius-Inverse (Frobenius-Rechstinverse) der Matrix A zum Muster M“ (203).

Die Ermittlung der unvollständigen Fobenius-Inversen basiert auf dem QR-Verfahren (siehe Kapitel 4.4.3). Dadurch entsteht das selbe Problem wie bei der IQR-Zerlegung, nämlich der Verlust der Orthogonalität von $\boldsymbol{Q}$ (Die Matrix $\boldsymbol{A}$ besteht aus n linear unabhängigen Spalten.) mit ansteigender Anzahl der Unbekannten. Folglich kann erwartet werden, dass diese erheblich aufwendigere Präkonditionierungsart zu keiner Konvergenzverbesserung führen wird, so dass sie im Rahmen dieser Arbeit nicht implementiert wurde.

4.5.4 Polynomiale Präkonditionierer

In [7] werden die polynomialen Präkonditionierer folgendermaßen beschrieben:

„Die grundlegende Idee polynomialer Präkonditionierer beruht auf der Darstellung der Inversen einer Matrix in der Form einer Neumannschen Reihe.

Satz 5.6 *Sei $\rho(I - A) < 1$, dann ist die Matrix $A \in \mathbb{R}^{n \times n}$, und die Inverse A^{-1} besitzt die Darstellung in der Form einer Neumannschen Reihe*

$$A^{-1} = \sum_{k=0}^{\infty} (I - A)^k \text{ “ (193).}$$

Da diese Präkonditionierungstechnik jedoch in [7] als äußerst ineffizient eingestuft wird (erhöhte Anzahl an Matrix-Vektor-Multiplikationen und sehr hohe Iterationszahlen), wurde auf eine Implementierung verzichtet.

4.5.5 Elementweise Vorkonditionierung (EBE)

Die bisher vorgestellten Präkonditionierer wurden auf Gesamtsteifigkeitsmatrizen (bei der parallelen Anwendung auf Teilsteifigkeitsmatrizen) angewendet. Die Idee einer elementweisen Präkonditionierungstechnik ist, die Präkonditionierung an den einzelnen Elementsteifigkeits-

matrizen durchzuführen. Dadurch soll bei den parallelen Berechnungen Kommunikationsaufwand gespart werden.

4.5.5.1 Sortierte Cholesky-EBE Präkonditionierung

Bei der Cholesky-EBE-Präkonditionierung (engl. Element-By-Element) wird die Gesamtsteifigkeitsmatrix $\boldsymbol{A}$ durch folgendes Produkt approximiert:

$$\boldsymbol{A} \approx \boldsymbol{M} = \boldsymbol{D}^{0,5} \prod_{e=1}^{nel} \boldsymbol{L}^e \prod_{e=nel}^{1} \boldsymbol{U}^e \, \boldsymbol{D}^{0,5} \tag{4.18}$$

mit

$$\boldsymbol{A}^e = \boldsymbol{L}^e \boldsymbol{U}^e = \boldsymbol{I} + D^{-0,5} B^{e^T} \left(A^e - diag\left(A^e \right) \right) B^e D^{-0,5} \tag{4.19}$$

A^e stellt hier die Elementsteifigkeitsmatrix dar. B ist ein Zuordnungsoperator und $\boldsymbol{I}$ die Einheitsmatrix.

„Die Elemente der Diagonalmatrix sind die den globalen Freiheitsgraden des Elements zugeordneten Einträge diag$\boldsymbol{A}^{-0,5}$“ ([5],139). Durch diese Gleichung wird die Stabilität der Choleskyzerlegung gesichert.

Der Präkonditionierer entspricht der Inversen von M:

$$\boldsymbol{P} \approx \boldsymbol{M}^{-1} = \boldsymbol{D}^{-0,5} \prod_{e=nel}^{1} \left(\boldsymbol{U}^e \right)^{-1} \prod_{e=1}^{nel} \left(\boldsymbol{L}^e \right)^{-1} \boldsymbol{D}^{-0,5} \tag{4.20}$$

Diese Gleichung muss nicht explizit berechnet werden. Auch hier gilt, dass nur die Wirkung des Präkonditionierers auf einen Vektor $\boldsymbol{f}$ benötigt wird. So dass die elementweise Präkonditionierung $\hat{\boldsymbol{f}} = \boldsymbol{Pf}$ durch folgende Teilschritte berechnet wird:

$$\boldsymbol{f}' = \boldsymbol{D}^{-0,5} \, \boldsymbol{f} \tag{4.21}$$

$$\boldsymbol{f}'' = \prod_{e=nel}^{1} \left(\boldsymbol{L}^e \right)^{-1} \boldsymbol{f}' = \left(\boldsymbol{L}^{nel} \right)^{-1} * \left(\boldsymbol{L}^{nel-1} \right)^{-1} * \cdots * \left(\boldsymbol{L}^1 \right)^{-1} \boldsymbol{f}' \tag{4.22}$$

$$\boldsymbol{f}''' = \prod_{1}^{e=nel} \left(\boldsymbol{U}^e \right)^{-1} \boldsymbol{f}'' = \left(\boldsymbol{U}^1 \right)^{-1} * \left(\boldsymbol{U}^2 \right)^{-1} * \cdots * \left(\boldsymbol{U}^{nel} \right)^{-1} \boldsymbol{f}'' \tag{4.23}$$

$$\hat{\boldsymbol{f}} = \boldsymbol{D}^{-0,5} \boldsymbol{f}''' \tag{4.24}$$

Dieses Verfahren hat folgende Nachteile:

- Zusätzlicher Speicherbedarf für die Elementsteifigkeitsmatrizen
- Zusätzlicher Rechenaufwand in der Größenordnung der Elementanzahl

Aufgrund dieser Nachteile wird auf eine Implementierung im Rahmen dieser Arbeit verzichtet. Verbesserungen dieser Version (z.B. Reduzierte Global extraction-EBE Präkonditionierung) sind nur unter programmiertechnisch höchsten Aufwand umsetzbar, da eine Kommunikation zwischen Matrizen erforderlich wäre.

4.6 Parallele Anwendung der Präkonditionierer

Für eine parallele Anwendung der Präkonditionierer, müssen diese in akkumulierter Form in jedem Prozessor für die Matrix-Vektor-Multiplikation zur Verfügung stehen.

Da der Jakobi-Präkonditionierer, als Inverse der Diagonalmatrix von $\boldsymbol{A}$, in Vektorenform gespeichert werden kann, erfolgt die Ermittlung des akkumulierten Präkonditionierers mit einer Vektortypumwandlung. So dass ein geringer Mehraufwand entsteht. Der parallele Jakobi-Präkonditionierer ist sowohl bei einer Netzverfeinerung mit der h-Version als auch mit der p-Version anwendbar.

Zwar werden die übrigen Skalierer aufgrund ihrer Diagonalgestalt ebenfalls als Vektoren gespeichert, die Werte der Diagonalen setzen sich aber im Gegensatz zum Jakobi-Präkonditionierer für:

- den Betragssummen-Skalierer aus der Summe der Beträge der jeweiligen Zeile / Spalte der Gesamtsteifigkeitsmatrix,
- den Euklid-Skalierer aus der Länge der jeweiligen Zeile / Spalte der Gesamtsteifigkeitsmatrix,
- den Maximum-Skalierer aus dem betragsmäßig maximalen Wert der Zeile / Spalte der Gesamtsteifigkeitsmatrix

zusammen. So dass bei der Aufsplittung der Gesamtsteifigkeitsmatrix, zur parallelen Berechnung, eine Matrizen-Kommunikation notwendig ist, um die akkumulierte Form der Skalierer berechnen zu können.

Die splitting-assoziierten und unvollständigen Präkonditionierer sind ebenfalls auf eine Matrizenkommunikation angewiesen, um in akkumulierte Matrizen umgewandelt zu werden.
Bei der Ermittlung der Präkonditionierer für die Gesamtsteifigkeitsmatrix wird ein Eintrag jeweils aus den in den Spalten bzw. Zeilen voranstehenden Einträgen bestimmt (siehe Kapitel 4.3 und 4.4). Bei der Anwendung der nicht-überlappenden Gebietszerlegung auf verteilten Arbeitsplatzrechnern entstehen additiv verteilte Gesamtsteifigkeitsmatrizen, so dass eine Matrizen-Kommunikation zur Erstellung dieser Präkonditionierer ebenfalls erforderlich wird.

Die Implementierung der Matrizenkommunikation ist jedoch mit einem unverhältnismäßig großen Aufwand verbunden, so dass außer dem Jakobi-Skalierer die übrigen Präkonditionierer nicht parallel angewendet werden können.

Folglich ist für eine parallele Berechnung eines Gleichungssystems der am einfachsten und schnellsten zu erzeugende Jakobi-Skalierer, trotz gröbster Approximation an die Inverse der Steifigkeitsmatrix, besonders gut geeignet.

5 Anwendungsbeispiele

5.1 Girkmann-Scheibe

Das nachfolgende Beispiel einer Girkmann-Scheibe liefert eine symmetrische Steifigkeitsmatrix und kann sowohl mit dem CG-Verfahren als auch mit dem stabilisierten BiCGSTAB-Verfahren gelöst werden. Dieses Beispiel bietet sich zum Vergleich der beiden Verfahren an. Als Girkmann-Scheibe versteht man eine dreiseitig gelagerte Scheibe, die an ihren seitlichen Kanten jeweils vertikal verschieblich gelagert, an der unteren Kante eingespannt und an der oberen Kante durch eine Streckenlast belastet ist.

Nachfolgend wird die Girkmann-Scheibe in zwei Diskretisierungs-Varianten berechnet. Bei der Auswertung wird jedoch nicht das Ergebnis der Verschiebungen betrachtet, sondern die Berechnung an sich, das heißt die Konditionsverbesserung durch die Präkonditionierer anhand der Iterationszahlen beurteilt. In Abbildung 5-1 ist die Scheibe in 100 Elemente diskretisiert und in Abbildung 5-2 auf 400 Elemente verfeinert.

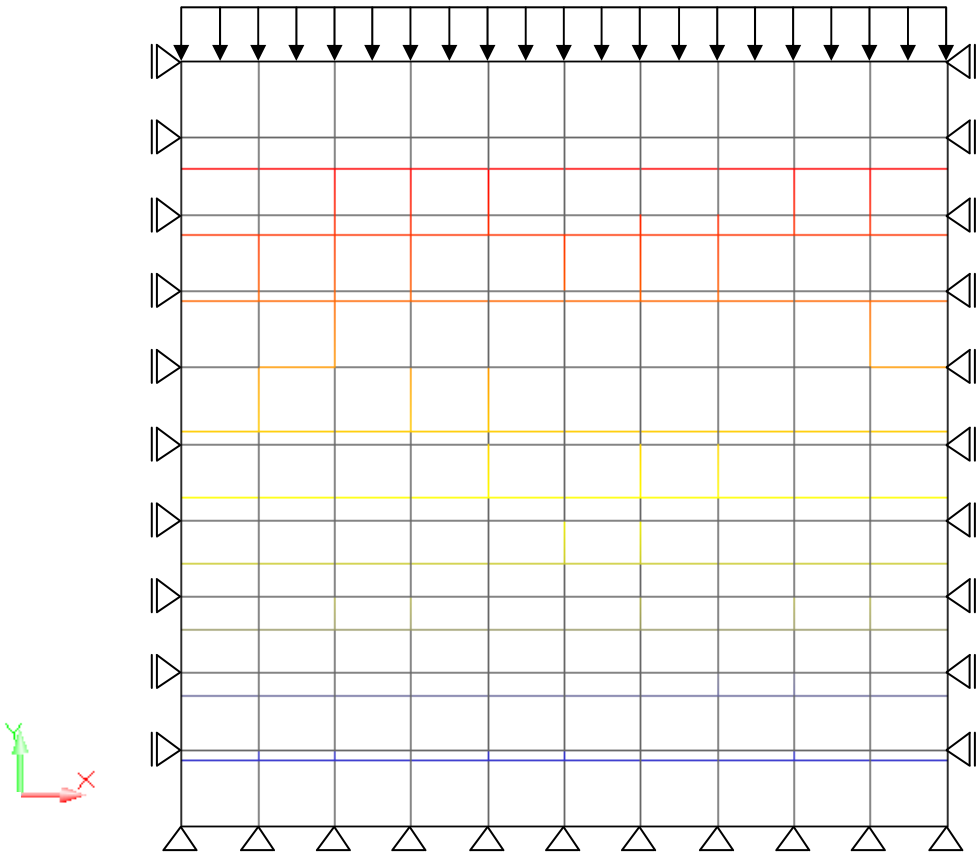

Abbildung 5-1: Girkmann-Scheibe mit 100 Elementen und 121 Knoten

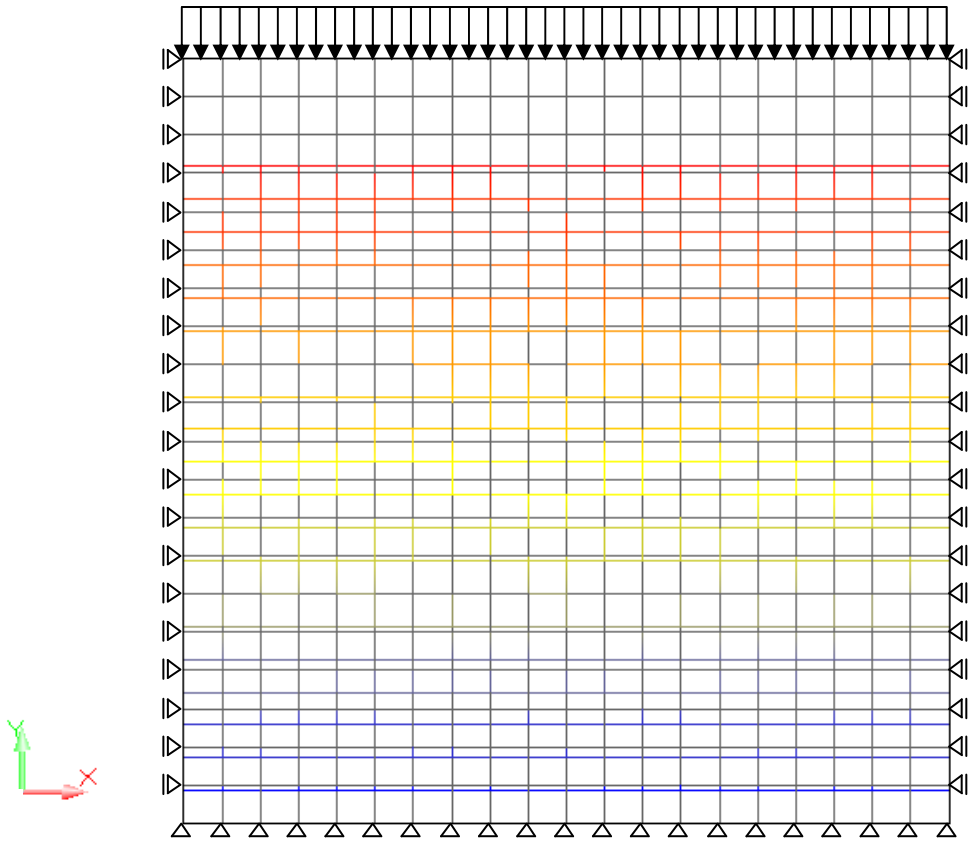

Abbildung 5-2: Girkmann-Scheibe mit 400 Elementen und 441 Knoten

Folgende Werkstoffeigenschaften und Lasten wurden für beide Varianten gewählt:

Elastizitätsmodul:	$E = 10000\ N/m^2$
Querkontraktionszahl:	$\nu = 0{,}2$
Scheibendicke:	$d = 1\ m$
Eigengewicht:	wird vernachlässigt
Streckenlast:	$p = 1000\ N/m$

Um die Wirkung der Präkonditionierung auf die durch die p-Version (Netzverfeinerung) veränderte Steifigkeitsmatrix zu überprüfen, wurden die oben dargestellten Scheiben jeweils mit dem Polynomgrad 3 verfeinert. Die hinzukommenden Freiheitsgrade werden nachfolgend dargestellt:

Für 2-dimensionale 4-Knoten-Elemente gilt:

Polynomgrad 1	Anzahl der Knoten · 2
Polynomgrad 2	Anzahl der Knoten · 2 + Anzahl der Kanten · 2
Polynomgrad 3	Anzahl der Knoten · 2 + Anzahl der Kanten · 2 + Anzahl der Kanten · 2

Der Faktor 2 bei der Multiplikation resultiert aus den beiden Translations-Freiheitsgraden in x- und in y-Richtung.

Für die Scheibe mit 100 Elementen ergeben sich somit für:

Polynomgrad 1: 242 Freiheitsgrade und

Polynomgrad 3: 1122 Freiheitsgrade.

Für die Scheibe mit 400 Elementen ergeben sich für:

Polynomgrad 1: 882 Freiheitsgrade und

Polynomgrad 3: 4242 Freiheitsgrade.

Des Weiteren wird auch der Effekt normierter Polynome für den Polynomgrad 3 untersucht. Eine Normierung der Polynome durch Multiplikation mit einem Vorfaktor führt dazu, dass die Spalten der Matrix unabhängig voneinander werden (senkrecht aufeinander stehen) und somit die Steifigkeitsmatrix eine bessere Konditionszahl bekommt. Die normierten Polynome unterscheiden sich von den nicht normierten Polynomen erst ab dem Polynomgrad 3. Die Normierung (Vorfaktor) ist in den nachfolgend, bis Polynomgrad 4, dargestellten Formfunktionen grau schattiert hervorgehoben:

$P_1(\xi) = 1/2\ (1 - \xi)$

$P_2(\xi) = 1/2\ (1 + \xi)$

$P_3(\xi) = 1/4\ (6)^{1/2}\ (\xi^2 - 1)$

$P_4(\xi) = 1/4\ (10)^{1/2}\ (\xi^2 - 1)\ \xi$

5.2 Belasteter Boden-Ausschnitt

Die Diskretisierung von Boden-Material liefert bei der Berücksichtigung der Interaktion zwischen mehreren Phasen (hier: Festkörper und Fluid) eine nicht symmetrische Steifigkeitsmatrix. So dass nur das stabilisierte BiCGSTAB-Verfahren zur Lösung des Problems angewendet werden kann.

Folgendes Beispiel mit einer Diskretisierung in 126 Elemente wird zur vergleichenden Bewertung der Präkonditionierer berechnet:

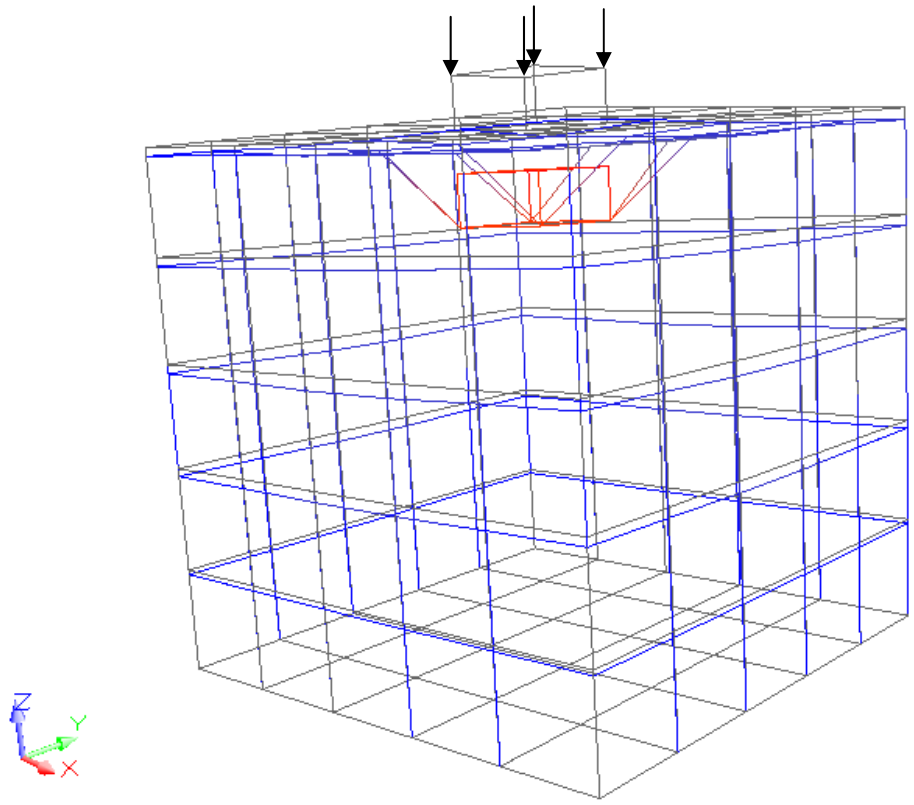

Abbildung 5-3: Boden-Ausschnitt mit 126 Elemente und 220 Knoten

Aus Gründen der Übersichtlichkeit wurde in diesem dreidimensionalen Beispiel auf die Darstellung der Lagersymbole verzichtet. Die Seitenflächen sind in jedem Knoten, analog zu den zweidimensionalen Beispielen in Abbildung 5-1 und Abbildung 5-2, vertikal verschieblich und horizontal unverschieblich gelagert. Die Unterseite ist vertikal und horizontal unverschieblich gelagert. Das Fundament auf der Oberfläche stellt eine flächig verteilte vertikale Last dar.

Folgende Werkstoffeigenschaften und Lasten wurden gewählt:

	Boden (Sand):	Fundament (Beton):
Elastizitätsmodul:	$E = 100000$ N/m²	$E = 34000000$ N/m²
Querkontraktionszahl:	$\nu = 0{,}2$	$\nu = 0{,}2$
Wichte des Festkörpers:	$\gamma_S = -18$ N/m³	$\gamma_S = -15$ N/m³
Fluid-Anteil:	$\theta_F = 0{,}3$	$\theta_F = 0{,}1$
Kompressibilität:	$Q = 7{,}86*10^8$ kN/m²	$Q = 1{,}89*10^7$ kN/m²
Wichte des Fluids:	$\gamma_F = 10$ N/m³	$\gamma_F = 15$ N/m³
Durchlässigkeitsbeiwert:	$k_f = 1*10^{-5}$ m/s	--
Dichte des Festkörpers:	$\rho_S = 1835{,}5$ kg/m³	$\rho_S = 2550$ kg/m³
Dichte des Fluids:	$\rho_F = 1000$ kg/m³	$\rho_F = 1000$ kg/m³
Vertikallast auf den vier oberen Knoten des Quaders:		1000 N

Die Netzverfeinerung des oben dargestellten 3-dimensionalen Boden-Modells mit der p-Version führt auf folgende Berechnung der zusätzlichen Freiheitsgrade:

Polynomgrad 1: Anzahl der Knoten · 6

Polynomgrad 2: Anzahl der Knoten · 6 + Anzahl der Kanten · 6

Polynomgrad 3: Anzahl der Knoten · 6 + Anzahl der Kanten · 6 + Anzahl der Kanten · 6

Der Faktor 6 aus der Multiplikation mit der Knoten- und Kanten-Anzahl resultiert aus den Translations-Freiheitsgraden in x-, y- und z-Richtung jeweils für den Festkörper- und Fluid-Anteil getrennt, dass heißt, dass bei einer Lagerung des Bodenelements der Fluid-Anteil explizit gelagert werden muss, um ein Durchsickern des Fluids durch den Boden zu vermeiden.

Somit ergeben sich für dieses Beispiel folgende Freiheitsgrade:

Polynomgrad 1: 1320 Freiheitsgrade

Polynomgrad 3: 7896 Freiheitsgrade

6 Vergleich der Präkonditionierer

In den Auswertungs-Tabellen werden folgende Abkürzungen verwendet:

PK	Präkondititonerer
t-PK	Zeit zur Aufstellung des Präkonditionierers
t-Lösung	Zeit zur Lösung des Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$
redA	reduzierte Steifigkeitsmatrix $\boldsymbol{A}$
Betrag	Skalierung bezüglich der Betragssummennorm
Euklid	Skalierung bezüglich der euklidischen Norm
Jakobi	Skalierung mit dem Diagonalelement / Jakobi-Skalierer
Maximum	Skalierung bezüglich der Maximumsnorm
StabBiCGSTAB	stabilisiertes BiCGSTAB-Verfahren
SchurCG	Anwendung der Schurkomplement-Methode auf das CG-Verfahren
SchurStabBiCGSTAB	Anwendung der Schurkomplement-Methode auf das stabilisierte BiCGSTAB-Verfahren

Die Scheibe in Abbildung 5-1 wird nachfolgend auch als Scheibe-100 und das Beispiel aus Abbildung 5-2 als Scheibe-400 bezeichnet.

Die Anwendungsbeispiele mit sequentiellen Verfahren wurden auf einem Rechner mit 1,13 GHz Intel Pentium III-Mobile Prozessor und 256 MB Arbeitsspeicher berechnet. Mit parallelen Verfahren wurde auf Rechnern mit Intel Celeron 466 MHz Prozessoren und 128 MB Arbeitsspeicher gerechnet.

Für alle Beispiele wurde das Abbruchkriterium auf $\varepsilon = 0{,}001$ gesetzt.

Da die Berechnung der Konditionszahl vor und nach der Präkonditionierung für sehr große Systeme zu zeitintensiv ist, wird auf die explizite Angabe dieser verzichtet. Stattdessen wird die Auswertung anhand der benötigten Iterationsschritte vor und nach der Präkonditionierung durchgeführt.

Die für die Aufstellung der Präkonditionierer benötigte Zeit beträgt selbst bei dem größten in dieser Arbeit berechneten Gleichungssystem lediglich 30 Sekunden (mit Ausnahme des ILU-Präkonditionierers) im Vergleich zu einer insgesamt für die Lösung benötigte Zeit von ungefähr

2 Stunden und 52 Minuten. Die Zeiten für die Aufstellung der Präkonditionierer sind somit prinzipiell vernachlässigbar.

Mit der Kardinalität (siehe Beschriftungen der Tabellen) wird die Anzahl der Elemente in der reduzierten Steifigkeitsmatrix, die ungleich Null sind, bezeichnet.

6.1 Auswertung der Girkmann-Scheiben

6.1.1 Vergleich der Präkonditionierer für Polynomgrad 1

In den nachfolgenden beiden Tabellen sind die Ergebnisse für die Scheibe-100 mit dem Polynomgrad 1 (242 Freiheitsgrade), berechnet mit dem CG- und StabBiCGSTAB-Verfahren dargestellt.

Tabelle 6-1: Scheibe mit 100 Elementen, CG-Verfahren,
Polynomgrad 1, redA = 200, Kardinalität: 2890

PK	t-PK	t-Lösung	Iterationen
ohne	--	771ms	26
Betrag	0,05 s	671ms	21
Euklid	0,06 s	631ms	17
Jakobi	0,01 s	421ms	9
Maximum	0,07 s	1s, 382ms	9
ILU	0,842s	1s, 593ms	5
IC	0,12 s	570ms	5
SSOR			
$\omega = 1$	0,04 s	661ms	9
$\omega = 1{,}05$	0,07 s	1s, 31ms	9
$\omega = 1{,}1$	0,04 s	821ms	9
$\omega = 1{,}2$	0,03 s	681ms	9
$\omega = 1{,}3$	0,03 s	641ms	8
$\omega = 1{,}4$	0,06 s	921ms	8
$\omega = 1{,}5$	0,03 s	651ms	8
$\omega = 1{,}6$	0,04 s	741ms	9
$\omega = 0{,}9$	0,03 s	661ms	9
$\omega = 0{,}8$	0,05 s	650ms	8

Tabelle 6-2: Scheibe mit 100 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 1, redA = 200, Kardinalität: 2890

PK	t-PK	t-Lösung	Iterationen
ohne	--	1s, 803ms	20
Betrag	0,04 s	831ms	14
Euklid	0,07 s	751ms	10
Jakobi	0,01 s	991ms	9
Maximum	0,06 s	651ms	9
ILU	0,881 s	1s, 552ms	2
IC	0,100 s	791ms	2
SSOR			
$\omega = 1$	0,04 s	891ms	5
$\omega = 1{,}05$	0,05 s	1s, 252ms	5
$\omega = 1{,}1$	0,06 s	1s, 402ms	6
$\omega = 1{,}2$	0,05 s	932ms	5
$\omega = 1{,}3$	0,03 s	862ms	5
$\omega = 1{,}4$	0,03 s	891ms	5
$\omega = 1{,}5$	0,03 s	771ms	4
$\omega = 1{,}6$	0,04 s	1s, 162ms	7
$\omega = 0{,}9$	0,04 s	771ms	4
$\omega = 0{,}1$	0,03 s	1s, 261ms	9

In diesen beiden Tabellen ist trotz der relativ kleinen Steifigkeitsmatrix bereits sehr gut zu erkennen, dass alle präkonditionierten Algorithmen zur Lösung des Gleichungssystems eine geringere Iterationszahl benötigen als der Grundalgorithmus. Alle Präkonditionierer erzielen somit eine Konditionsverbesserung der Steifigkeitsmatrix. Die kleinste Konditionszahl wurde sowohl beim CG- als auch beim StabBiCGSTAB-Verfahren mit den ILU- und IC-Präkonditionierern erreicht. Bezüglich der Rechenzeit kann aufgrund der geringen Freiheitsgrade keine vergleichende Aussage getroffen werden, da die Unterschiede zwischen den Lösungsvarianten im Bereich von zehntel Sekunden liegen.

In Tabelle 6-3 und Tabelle 6-4 sind die Ergebnisse für die Scheibe-400 mit Polynomgrad 1 zusammengefasst.

Tabelle 6-3: Scheibe mit 400 Elementen, CG-Verfahren,
Polynomgrad 1, redA = 800, Kardinalität: 10970

PK	t-PK	t-Lösung	Iterationen
ohne	--	13s, 119ms	52
Betrag	0,330 s	10s, 285ms	40
Euklid	0,481 s	8s, 472ms	32
Jakobi	0,010 s	4s, 817ms	19
Maximum	0,371 s	5s, 358ms	19
ILU	36,312 s	45s, 416ms	11
IC	0,911 s	6s, 790ms	11
SSOR			
$\omega = 1$	0,301 s	9s, 474ms	16
$\omega = 1{,}05$	0,340 s	9s, 473ms	16
$\omega = 1{,}1$	0,310 s	9s, 213ms	16
$\omega = 1{,}2$	0,300 s	9s, 93ms	15
$\omega = 1{,}3$	0,301 s	8s, 12ms	14
$\omega = 1{,}4$	0,300 s	7s, 781ms	13
$\omega = 1{,}5$	0,310 s	7s, 591ms	13
$\omega = 1{,}6$	0,310 s	7s, 621ms	13
$\omega = 0{,}9$	0,311 s	7s, 791ms	13
$\omega = 0{,}8$	0,300 s	8s, 71ms	14

Tabelle 6-4: Scheibe mit 400 Elementen, StabBiCGSTAB-Verfahren,
Polynomgrad 1, redA = 800, Kardinalität: 10970

PK	t-PK	t-Lösung	Iterationen
ohne	--	19s, 348ms	41
Betrag	0,301 s	13s, 530ms	26
Euklid	0,451 s	9s, 834ms	19
Jakobi	0,010 s	9s, 443ms	19
Maximum	0,361 s	11s, 687ms	19
ILU	34,90 s	47s, 408ms	6
IC	0,902 s	9s, 143ms	6
SSOR			
$\omega = 1$	0,301 s	10s, 145ms	7
$\omega = 1{,}05$	0,310 s	11s, 247ms	7
$\omega = 1{,}1$	0,501 s	15s, 943ms	7
$\omega = 1{,}2$	0,531 s	22s, 783ms	10
$\omega = 1{,}3$	0,561 s	18s, 496ms	8
$\omega = 1{,}4$	0,300 s	11s, 897ms	8
$\omega = 1{,}5$	0,320 s	10s, 395ms	7
$\omega = 1{,}6$	0,301 s	12s, 448ms	9
$\omega = 0{,}9$	0,320 s	11s, 726ms	8
$\omega = 0{,}1$	0,351 s	24s, 666ms	18

Die Verfeinerung der Diskretisierung liefert 882 Freiheitsgrade, dies entspricht ungefähr dem 3,6-fachen der Scheibe mit 100 Elementen. Die Unterschiede zwischen den Zeiten zur Lösung des Gleichungssystems (t-Lösung) liegen im Sekunden-Bereich, mit Ausnahme des ILU-Präkonditionierers, der die mit Abstand längste Zeit benötigt. Auch bei der Scheibe-400 führen alle Präkonditionierungs-Techniken zu einer Konditionsverbesserung. Alle Iterationen liegen unter der benötigten Iterationszahl des Grundalgorithmus. Tendenziell kann festgestellt werden, dass die unvollständigen Präkonditionierer ILU und IC die wenigsten Iterationen benötigen, da sie die beste Approximation an die Inverse der Steifigkeitsmatrix liefern. Eine etwas schlechtere Iterationszahl liefert der SSOR-Präkonditionierer, gefolgt von den skalierenden Präkonditionierern. Erkennbar ist, dass die benötigte Zeit zum Aufstellen des ILU-Präkonditionierers (t-PK) länger ist als die Lösungszeit (t-Lösung) des Grundalgorithmus. Dies lässt die Vermutung zu, dass der ILU-Präkonditionierer bei großen Gleichungssystemen ineffizient wird. Aus den Ergebnissen ist kein direkter Zusammenhang zwischen der Anzahl der Iterationen und der Lösungszeit feststellbar.

6.1.2 Wirkung von Relaxationsparametern

Durch optimale Wahl des Relaxationsparameters wird die Konditionszahl der Steifigkeitsmatrix so stark reduziert, dass das SSOR-Verfahren in Kombination mit dem StabBiCGSTAB-Verfahren nur eine Iteration mehr liefert als die unvollständigen Präkonditionierer (Tabelle 6-4). In dieser Arbeit wurde der Parameter ω durch Probieren zum Optimum variiert, was für große Systeme zu Aufwendig ist. In den nachfolgenden Tabellen wird daher für die normierten Polynome der Parameter ω ausschließlich zu 1,0 gewählt.

6.1.3 Wirkung der Präkonditionierer bei Anwendung der p-Version

Um zu untersuchen, ob bei einer Netzverfeinerung mit der p-Version die Zusammensetzung der Steifigkeitsmatrix bessere Konditionseigenschaften zur Folge hat, und die Präkonditionierer dadurch überflüssig wären, wurden die Beispiele bis zum Polynomgrad 3 verfeinert. Dadurch bekommt die Scheibe-100 1122 und die Scheibe-400 4242 Freiheitsgrade. Die Ergebnisse dieser Berechnungen sind in den nachfolgenden vier Tabellen dargestellt.

Tabelle 6-5: Scheibe mit 100 Elementen, CG-Verfahren,
Polynomgrad 3, redA = 1000, Kardinalität: 22170

PK	t-PK	t-Lösung	Iterationen	t-PK	t-Lösung	Iterationen
	nicht normierte Polynome			normierte Polynome		
ohne	--	13s,189ms	31	--	12s, 668ms	31
Betrag	0,490 s	14s, 231ms	30	0,601 s	13s, 500ms	30
Euklid	0,731 s	11s, 26ms	24	0,691 s	11s, 717ms	24
Jakobi	0,020 s	7s, 440ms	17	0,040 s	7s, 350ms	17
Maximum	0,501 s	8s, 232ms	17	0,521 s	7s, 711ms	17
ILU	74,437s	1m, 22s, 979ms	6	82,208 s	1m, 30s, 650ms	6
IC	1,372 s	8s, 172ms	7	1,092 s	8s, 603ms	7
SSOR						
$\omega = 1$	0,792 s	10s, 365ms	10	0,510 s	9s, 724ms	10
$\omega = 1{,}05$	0,921 s	10s, 605ms	10	--	--	--
$\omega = 1{,}1$	0,571 s	11s, 357ms	11	--	--	--
$\omega = 1{,}2$	0,500 s	11s, 126ms	11	--	--	--
$\omega = 1{,}3$	0,511 s	10s, 105ms	10	--	--	--
$\omega = 1{,}4$	0,791 s	11s, 106ms	11	--	--	--
$\omega = 1{,}5$	0,511 s	11s, 607ms	12	--	--	--
$\omega = 0{,}9$	0,781 s	10s, 805ms	10	--	--	--
$\omega = 0{,}8$	0,791 s	9s, 944ms	9	--	--	--
$\omega = 0{,}7$	0,781 s	9s, 613ms	9	--	--	--

Tabelle 6-6: Scheibe mit 100 Elementen, StabBiCGSTAB-Verfahren,
Polynomgrad 3, redA = 1000, Kardinalität: 22170

PK	t-PK	t-Lösung	Iterationen	t-PK	t-Lösung	Iterationen
	nicht normierte Polynome			normierte Polynome		
ohne	--	17s, 946ms	21	--	17s, 385ms	21
Betrag	0,491 s	15s, 152ms	17	0,501 s	14s, 962ms	17
Euklid	0,801 s	14s, 50ms	16	0,711 s	14s, 190ms	16
Jakobi	0,020 s	11s, 66ms	12	0,010 s	10s, 805ms	12
Maximum	0,511 s	11s, 46ms	12	0,600 s	13s, 429ms	12
ILU	73,726s	1m, 25s, 613ms	3	77,791s	1m, 31s, 30ms	3
IC	1,102 s	9s, 353ms	3	1,092 s	9s, 293ms	3
SSOR						
$\omega = 1$	0,500 s	12s, 918ms	5	0,581 s	13s, 520ms	5
$\omega = 1{,}05$	0,491 s	13s, 990ms	5	--	--	--
$\omega = 1{,}1$	0,500 s	13s, 18ms	5	--	--	--
$\omega = 1{,}2$	0,511 s	13s, 390ms	5	--	--	--
$\omega = 1{,}3$	0,511 s	15s, 102ms	6	--	--	--
$\omega = 1{,}4$	0,531 s	15s, 72ms	6	--	--	--
$\omega = 0{,}9$	0,521 s	13s, 650ms	5	--	--	--

Tabelle 6-7: Scheibe mit 400 Elementen, CG-Verfahren,
Polynomgrad 3, redA = 4000, Kardinalität: 86330

PK	t-PK	t-Lösung	Iterationen	t-PK	t-Lösung	Iterationen
	nicht normierte Polynome			normierte Polynome		
ohne	--	5m, 33s, 510ms	56	--	5m, 34s, 261ms	56
Betrag	8,081 s	5m, 21s, 172ms	53	7,731 s	5m, 22s, 995ms	53
Euklid	11,346 s	5m, 39s, 729ms	43	11,607 s	5m, 56s, 503ms	43
Jakobi	0,091 s	4m, 42s, 767ms	29	0,06 s	3m, 21s, 750ms	29
Maximum	8,152 s	3m, 5s, 86ms	29	7,811 s	3m, 4s, 415ms	29
ILU	nach 20 m ist PK immer noch nicht erstellt			nach 20 m ist PK immer noch nicht erstellt		
IC	19,277 s	3m, 2s, 883ms	12	17,395 s	2m, 59s, 829ms	12
SSOR						
$\omega = 1$	7,951 s	4m, 13s, 224ms	14	11,297 s	4m, 34s, 84ms	14
$\omega = 1{,}05$	8,663 s	3m, 18s, 405ms	14	--	--	--
$\omega = 1{,}1$	7,911 s	3m, 39s, 516ms	17	--	--	--
$\omega = 1{,}2$	11,767 s	5m, 39s, 248ms	17	--	--	--
$\omega = 1{,}3$	8,222 s	3m, 32s, 55ms	16	--	--	--
$\omega = 1{,}4$	7,891 s	3m, 25s, 606ms	15	--	--	--
$\omega = 1{,}5$	11,416 s	5m, 12s, 790ms	16	--	--	--
$\omega = 0{,}9$	7,901 s	3m, 20s, 98ms	15	--	--	--
$\omega = 0{,}8$	8,032 s	2m, 51s, 557ms	13	--	--	--
$\omega = 0{,}7$	8,412 s	3m, 4s, 435ms	14	--	--	--

Tabelle 6-8: Scheibe mit 400 Elementen, StabBiCGSTAB-Verfahren,
Polynomgrad 3, redA = 4000, Kardinalität: 86330

PK	t-PK	t-Lösung	Iterationen	t-PK	t-Lösung	Iterationen
	nicht normierte Polynome			normierte Polynome		
ohne	--	7m, 59s, 970ms	40	--	8m, 25s, 727ms	40
Betrag	7,621 s	7m, 19s, 101ms	36	7,590 s	8m, 8s, 492ms	36
Euklid	12,227 s	6m, 28s, 248ms	31	12,789 s	7m, 16s, 488ms	31
Jakobi	0,071 s	4m, 49s, 507ms	23	0,050 s	4m, 56s, 296ms	23
Maximum	8,302 s	7m, 9s, 557ms	23	10,605 s	5m, 23s, 345ms	23
ILU	nach 20 m ist PK immer noch nicht erstellt			nach 20 m ist PK immer noch nicht erstellt		
IC	18,016 s	4m, 2s, 629ms	6	17,285 s	4m, 2s, 88ms	6
SSOR						
$\omega = 1$	7,801 s	4m, 25s, 41ms	8	7,631 s	4m, 22s, 477ms	8
$\omega = 1{,}05$	7,701 s	3m, 57s, 141ms	7	--	--	--
$\omega = 1{,}1$	7,581 s	3m, 57s, 61ms	7	--	--	--
$\omega = 1{,}2$	7,581 s	4m, 2s, 769ms	7	--	--	--
$\omega = 1{,}3$	7,561 s	4m, 29s, 378ms	8	--	--	--
$\omega = 1{,}4$	7,541 s	4m, 59s, 571ms	9	--	--	--
$\omega = 0{,}9$	7,571 s	4m, 24s, 560ms	8	--	--	--

Wie aus Tabelle 6-5 bis Tabelle 6-8 zu erkennen ist, wirkt sich eine Präkonditionierung der iterativen Verfahren bei einer Netzverfeinerung mit der p-Version positiv bezüglich der Konditionszahl auf die Steifigkeitsmatrix aus. Die unvollständigen Präkonditionierer werden als beste Approximation an die Steifigkeitsmatrix bestätigt und liefern, trotz Erhöhung der Freiheitsgrade um das 4,6–fache bei der Scheibe-100 und 4,8-fache bei der Scheibe-400 gegenüber Polynomgrad 1, eine maximale Iterationszahl von 12. Je größer die Steifigkeitsmatrizen jedoch werden, desto länger wird die Zeit zur Aufstellung des ILU-Präkonditionierers. In Tabelle 6-5 und Tabelle 6-6 ist zu erkennen, dass die Erstellung des ILU-Präkonditionierers um 60 beziehungsweise 56 Sekunden länger dauert als die Lösungszeit des Grundalgorithmus. Bei den Beispielen aus Tabelle 6-7 und Tabelle 6-8 wurde sogar der Rechenvorgang abgebrochen, da der ILU-Präkonditionierer selbst nach 20 Minuten noch nicht erstellt war und der Grundalgorithmus nach 5 Minuten und 33 Sekunden (CG-Verfahren) beziehungsweise nach 8 Minuten (StabBiCGSTAB-Verfahren) die Lösung berechnet hatte. Somit ist der ILU-PK ab einer Unbekanntenzahl von etwa 1000 Unbekannten unbrauchbar. Die Begründung hierfür liegt bei der Anwendung der p-Version selbst. Die p-Version als Netzverfeinerung schließt eine Bandstruktur der Steifigkeitsmatrix aus, welche für die ILU-Zerlegung bezüglich der Zeit von entscheidender Bedeutung ist. Modifizierte Formen des ILU-Präkonditionierers mit fill-in (Kapitel 4.5.1 und 4.5.2) würden somit zur Anwendung ohnehin nicht in Frage kommen. Der IC-Präkonditionierer liefert für die hier berechneten Beispiele aufgrund der Symmetrie-Ausnutzung, obwohl es sich um einen unvollständigen Präkonditionierer handelt, die schnellsten Lösungszeiten. Nach einem Testlauf für ein System mit 20000 Freiheitsgraden stellte sich allerdings der IC-Präkonditionierer bezüglich des Zeitbedarfs für die Staffelung der Steifigkeitsmatrix ebenfalls als ineffizient heraus. Somit ist auch dieser Präkonditionierer für sehr große Systeme ungeeignet.

6.1.4 Effekt der Parallelisierung

Die nachfolgenden Tabellen stellen die Ergebnisse der parallel durchgeführten Berechnungen für beide Scheiben-Varianten sowohl für das CG- als auch für das StabBiCGSTAB-Verfahren dar.
Als Effizienz der einzelnen Prozessoren wird der prozentuale Anteil der Rechenoperationen, die keine Kommunikation benötigen, bezeichnet.

Tabelle 6-9: Scheibe mit 100 Elementen, CG-Verfarhen, Polynomgrad 3, Prozessoren: 2

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Effizienz [%] Prozessor 2
		nicht normierte Polynome				
CG						
	ohne	--	39s, 98ms	31	70,75	75,06
	Jakobi	0,056 s	15s, 902ms	17	76,17	78,29
SchurCG						
	ohne	--	1m, 40s, 143ms	10	82,51	97,35
	Jakobi	--	1m, 31s, 666ms	1	99,00	98,91
		normierte Polynome				
CG						
	ohne	--	30s, 106ms	31	71,84	73,63
	Jakobi	0,986 s	17s, 678ms	17	77,3	77,72
SchurCG						
	ohne	--	1m, 34s, 195ms	6	83,97	97,88
	Jakobi	--	1m, 30s, 783ms	1	98,85	99,00

Tabelle 6-10: Scheibe mit 100 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, Prozessoren: 2

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Effizienz [%] Prozessor 2
		nicht normierte Polynome				
StabBiCGSTAB						
	ohne	--	57s, 605ms	21	67,24	73,97
	Jakobi	0,152 s	23s, 6ms	12	71,82	75,41
SchurStabBiCGSTAB						
	ohne	--	1m, 40s, 653ms	6	83,16	96,68
	Jakobi	--	1m, 32s, 505ms	1	98,41	98,68
		normierte Polynome				
StabBiCGSTAB						
	ohne	--	37s, 437ms	21	71,13	73,36
	Jakobi	0,174 s	23s, 603ms	12	71,77	74,78
SchurStabBiCGSTAB						
	ohne	--	1m, 36s, 870ms	4	83,63	97,39
	Jakobi	--	1m, 34s, 241ms	1	98,23	98,50

Tabelle 6-11: Scheibe mit 400 Elementen, CG-Verfarhen, Polynomgrad 3, Prozessoren: 2

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Effizienz [%] Prozessor 2
		nicht normierte Polynome				
CG						
	ohne	--	8m, 58s, 846ms	56	93,58	96,93
	Jakobi	0,102 s	2m, 54s, 27ms	29	93,24	97,81
		normierte Polynome				
CG						
	ohne	--	5m, 29s, 969ms	56	93,88	97,08
	Jakobi	0,086 s	2m, 56s, 806ms	29	68,49	95,97

Tabelle 6-12: Scheibe mit 400 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, Prozessoren: 2

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Effizienz [%] Prozessor 2
		nicht normierte Polynome				
StabBiCGSTAB						
	ohne	--	11m, 29s, 664ms	40	93,70	96,54
	Jakobi	1,325 s	5m, 3s, 814ms	23	93,43	96,33
		normierte Polynome				
StabBiCGSTAB						
	ohne	--	7m, 22s, 387ms	40	93,85	96,70
	Jakobi	0,302 s	5m, 4s, 635ms	23	92,90	95,69

Tabelle 6-13: Scheibe mit 100 Elementen, CG-Verfarhen, Polynomgrad 3, Prozessoren: 4

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Effizienz [%] Prozessor 2	Effizienz [%] Prozessor 3	Effizienz [%] Prozessor 4
		nicht normierte Polynome						
CG								
	ohne	--	1m, 4s, 425ms	31	80,95	78,47	79,15	79,23
	Jakobi	3,248 s	27s, 868ms	17	84,99	83,15	83,42	83,12
SchurCG								
	ohne	--	31s, 817ms	17	85,32	85,56	87,18	87,33
	Jakobi	--	23s, 5ms	10	88,35	89,05	89,10	87,84
		normierte Polynome						
CG								
	ohne	--	32s, 763ms	31	80,29	79,29	80,00	79,98
	Jakobi	0,157 s	18s, 241ms	17	82,12	81,32	81,51	81,35
SchurCG								
	ohne	--	21s, 886ms	10	86,70	86,07	90,12	89,92
	Jakobi	--	22s, 619ms	10	88,99	88,52	88,90	89,06

Tabelle 6-14: Scheibe mit 100 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, Prozessoren: 4

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Effizienz [%] Prozessor 2	Effizienz [%] Prozessor 3	Effizienz [%] Prozessor 4
		nicht normiert Polynome						
StabBiCGSTAB								
	ohne	--	1m, 40s, 195ms	21	80,92	77,93	78,31	78,47
	Jakobi	0,593 s	30s, 824ms	12	80,70	80,05	80,02	79,68
SchurStabBiCGSTAB								
	ohne	--	42s, 344ms	11	82,24	82,86	84,48	78,83
	Jakobi	--	24s, 277ms	6	87,96	88,15	88,49	88,38
		normierte Polynome						
StabBiCGSTAB								
	ohne	--	45s, 960ms	21	78,82	78,48	79,44	79,68
	Jakobi	0,207 s	26s, 116ms	12	80,02	80,10	80,49	80,32
SchurStabBiCGSTAB								
	ohne	--	21s, 700ms	6	85,71	85,64	89,96	89,94
	Jakobi	--	25s, 490ms	6	87,78	87,83	88,14	88,05

Tabelle 6-15: Scheibe mit 400 Elementen, CG-Verfarhen, Polynomgrad 3, Prozessoren: 4

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Prozessor 2	Prozessor 3	Prozessor 4
		nicht normierte Polynome						
CG								
	ohne	--	3m, 8s, 81ms	56	90,00	89,53	91,80	92,16
	Jakobi	0,085 s	59s, 741ms	29	91,40	91,21	93,09	93,62
		normierte Polynome						
CG								
	ohne	--	3m, 272ms	56	85,89	94,83	87,08	87,20
	Jakobi	8,315 s	1m, 35s, 541ms	29	88,82	95,64	89,95	89,74
SchurCG								
	ohne	--	20m, 8s, 638ms	13	87,20	99,77	89,24	89,44
	Jakobi	--	19m, 26s, 390ms	14	99,29	99,77	99,21	99,22

Tabelle 6-16: Scheibe mit 400 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, Prozessoren: 4

Iterativer Löser	PK	t-PK	t-Lösung	Iterationen	Effizienz [%] Prozessor 1	Prozessor 2	Prozessor 3	Prozessor 4
		nicht normiert Polynome						
StabBiCGSTAB								
	ohne	--	3m, 52s, 583ms	40	89,39	88,81	91,01	91,41
	Jakobi	1,288 s	1m, 46s, 691ms	23	90,49	90,47	91,96	92,53
		normierte Polynome						
StabBiCGSTAB								
	ohne	--	4m, 17s, 956ms	40	85,35	94,80	86,70	86,85
	Jakobi	0,544 s	3m, 23s, 13ms	23	87,22	87,18	90,92	87,78

Aus den Tabellen geht hervor, dass eine Präkonditionierung mit dem Jakobi-Skalierer immer zu einer Konditionsverbesserung und beschleunigten Konvergenz führt. Eine zusätzliche Normierung der Formfunktionen für Polynomgrad 3, wie im Kapitel 6.1.3 bereits erwähnt, bringt keinen Vorteil.

Die Verwendung der Schurkomplement-Methode für die Scheibe-100 reduziert zwar die Iterationsschritte mindestens um das 3,1-fache (Tabelle 6-9, Tabelle 6-10), benötigt jedoch zur Lösung der Anwendungsbeispiele eine längere Rechenzeit, so dass auf die Berechnung der Scheibe-400 mit der Schurkomplement-Methode auf zwei Prozessoren verzichtet wurde. Da die Iterationszahlen geringer sind, liegt der Grund für die längere Berechnungszeit in der direkten Berechnung der Innenknoten-Verschiebungen, die die Mehrzahl der Unbekannten bilden. Bei Erhöhung der Prozessorzahl auf vier für die Scheibe-100 (Tabelle 6-13, Tabelle 6-14), ergaben sich bei Anwendung des Schurkomplements gleiche und teilweise geringfügig geringere Lösungszeiten als ohne Anwendung des Schurkomplements. Dies liegt vermutlich daran, dass bei einer Vierteilung der Scheiben die Anzahl der Innenknoten gering genug ist, dass die direkte Lösung in der selben Zeit zum Ergebnis führt. Bei Berechnung der Scheibe-400 mit der SchurCG-Methode (Tabelle 6-15) ist die Anzahl der Innenknoten wiederum so groß, dass trotz Vierteilung die 7-fache (ohne PK) beziehungsweise 19-fache (Jakobi-PK) Lösungszeit benötigt

wird. Da ein analoges Verhalten bei Anwendung des SchurStabBiCGSTAB-Verfahrens zu erwarten ist, wurde für dieses auf die Berechnung der Scheibe-400 verzichtet. Der große Nachteil der Schurkomplement-Methode ist die Berechnung der Innenknoten-Verschiebungen mit direkten Verfahren, die zu zeitintensiv ist. Die Schurkomplement-Methode ist somit, trotz starker Verringerung der Iterationszahlen, als ineffizient einzustufen.

Die Beispiele der Scheibe-100 und Scheibe-400 zeigen noch nicht die Zeitersparnis, die durch Aufteilung in mehrere Teilsysteme beim parallelen Rechnen erwartet wurde. Der Grund hierfür liegt darin, dass der Kommunikationsaufwand bei kleinen Systemen verhältnismäßig hoch ist und somit kein Zeitvorteil zu einer sequentiellen Lösung erzielt wird.

6.1.5 Effekt der normierten Polynome

In Kapitel 5.1 wurde schon erwähnt, dass normierte Polynome die Iterationszahlen verbessern. Aus Tabelle 6-5 bis Tabelle 6-8 geht jedoch hervor, dass die normierten und nicht normierten Formfunktionen für Polynomgrad 3 die selben Iterationszahlen liefern. Bei Wahl höherer Polynomgrade hat sich jedoch erwiesen, dass die normierten Polynome eine geringere Iterationszahl liefern. Im Vergleich zu einer Präkonditionierung ist der Effekt einer Normierung allerdings sehr gering. Am Beispiel der Schurkomplement-Methode ist der positive Effekt der normierten Formfunktionen auf die Teilsteifigkeitsmatrix $\boldsymbol{A}^{cc}$ deutlich zu erkennen.

6.1.6 Vergleich zwischen CG-Verfahren und stabilisiertem StabBiCGSTAB-Verfahren

Wie in Kapitel 2.2 schon erwähnt, benötigt das stabilisierte StabBiCGSTAB-Verfahren gegenüber dem CG-Verfahren eine längere Zeit zur Lösung des Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$, da mehrerer Rechenschritte pro Iteration durchgeführt werden. So dass das stabilisiert StabBiCG-Verfahren nur für nicht symmetrische Steifigkeitsmatrizen Verwendung finden sollte.
Je größer die Anzahl der Freiheitsgrade desto stärker zeichnet sich die bessere Konvergenzgeschwindigkeit des StabBiCGSTAB-Verfahrens gegenüber dem CG-Verfahren aus. Dies liegt womöglich in der stabilisierten Form begründet.

6.2 Auswertung des Boden-Modells

Da für den Polynomgrad 1 das Boden-Modell geringfügig mehr Freiheitsgrade (1320 FG) liefert als die Scheibe-100 mit Polynomgrad 3 (1122 FG), wurde der Boden nur für den Polynomgrad 3 mit 7896 Freiheitsgraden ausgewertet. Für das sequentielle Lösen des Problems wurde, da sich der Jakobi-Präkonditionierer unter den Skalierern als Präkonditionierer mit geringster

Iterationszahl herausgestellt hat und im allgemeinen die kürzeren Lösungszeiten liefert, auf die Anwendung der übrigen Skalierer verzichtet. Die SOR-Präkonditionierung wurde nur für den Relaxationsparameter $\omega = 1$ durchgeführt, da zu erkennen war, dass auch ohne die zeitaufwendige Suche des optimalen Relaxationsparameters bessere Iterationszahlen erzielt werden als mit den Skalierern. Des weiteren wurde die parallele Anwendung des StabBiCGSTAB-Verfahrens nur mit normierten Formfunktionen ausgeführt, da bei den Scheiben deutlich wurde, dass bis zum Polynomgrad 3 keine nennenswerte Konditionsverbesserung erzielt werden kann. Auf die Schurkomplement-Methode wurde ebenfalls für dieses Beispiel verzichtet, da sie bereits bei den Scheiben-Beispielen mit geringerer Unbekanntenzahl ineffizient arbeitet.

In den nachfolgenden drei Tabellen ist die Auswertung des Boden-Modells zusammengefasst.

Tabelle 6-17: Boden mit 126 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, redA: 5744, Kardinalität: 1481449

PK	t-PK	t-Lösung	Iterationen	t-PK	t-Lösung	Iterationen
	nicht normierte Polynome			normierte Polynome		
ohne	--	2h, 48m, 25s, 51ms	167	--	2h, 52m, 49s, 821ms	167
Jakobi	0,822 s	1h, 18m, 14s, 540ms	70	0,160 s	1h, 5m, 6s, 527ms	70
ILU	nach 20 m ist PK immer noch nicht erstellt			nach 20 m ist PK immer noch nicht erstellt		
SOR $\omega = 1$	23,023 s	2h, 48m, 37s, 559ms	68	30,364 s	2h, 51m, 37s, 357ms	68

Tabelle 6-18: Boden mit 126 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, Prozessoren 2

PK	t-PK	t-Lösung	Iterationen	Effizienz [%]	
				Prozessor 1	Prozessor 2
	normierte Polynome				
ohne	--	48m, 26s, 7ms	167	96,80	86,84
Jakobi	0,426 s	43m, 45s, 862ms	70	74,01	98,9

Tabelle 6-19: Boden mit 126 Elementen, StabBiCGSTAB-Verfahren, Polynomgrad 3, Prozessoren 4

PK	t-PK	t-Lösung	Iterationen	Effizienz [%]				
				Prozessor 1	Prozessor 2	Prozessor 3	Prozessor 4	Prozessor 5
	normiert Polynome							
ohne	--	1h, 1m, 10s, 472ms	167	83,32	96,97	96,99	98,06	85,04
Jakobi	5,291 s	27m, 48s, 800ms	70	82,71	98,14	97,66	98,20	97,51

Auch für das unsymmetrische Beispiel liefern die Präkonditionierer geringere Iterationszahlen (2,4-fache Konvergenzgeschwindigkeit bei sequentieller Ausführung) und somit eine Konditionsverbesserung der Steifigkeitsmatrix. Der ILU-Präkonditionierer wurde auch hier, wie erwartet, nach 20 Minuten nicht aufgestellt. Auch hier bestätigt sich, dass er als Präkonditionierer ungeeignet ist. Der SOR-Präkonditionierer liefert sogar zwei Iterationen weniger als der Jakobi-Präkonditionierer, jedoch entspricht die Lösungszeit dem Grundalgorithmus, so dass dieser Präkonditionierer für große Systeme nicht sinnvoll anwendbar ist. Der Jakobi-Skalierer liefert bezüglich Iterationszahl und Lösungszeit von allen betrachteten Präkonditionierern die besten Ergebnisse. Bei paralleler Lösung des Boden-Modells mit zwei Prozessoren wird die Rechenzeit um das 3,5–fache (ohne PK) beziehungsweise 1,8-fache (Jakobi-PK) verkürzt. Bei Aufteilung des Boden-Modells in fünf Teilgebiete liefert zwar der parallele Grundalgorithmus eine kürzere Lösungszeit als die sequentielle Lösung, benötigt jedoch die doppelte Zeit wie bei der Aufteilung in zwei Teilgebiete. Daraus lässt sich schließen, dass bei einer zu hohen Aufteilung eines Systems das Verhältnis von Kommunikationszeit zu Berechnungszeit ungünstig wird. Führt man allerdings eine Präkonditionierung am Boden-Modell bestehend aus fünf Teilgebieten durch, so verkürzt sich die Berechnungszeit gegenüber dem zweiteiligen System um das 1,6-fache. Somit ist bei großen Systemen eine Präkonditionierung verbunden mit einer großen Parallelität (viele Teilgebiete) die effizienteste und beste Methode der Gleichungslösung.

7 Implementierung der parallelen Rechenoperationen

Die Umsetzung der in den vorherigen Kapiteln vorgestellten Algorithmen erfolgte mit der Programmiersprache Java. Die großen Vorteile dieser Sprache sind die Unabhängigkeit von Betriebssystemen, wie zum Beispiel Windows oder Linux (Plattformunabhängikeit), und die gute Skalierbarkeit (10). Diese Eigenschaften können insbesondere bei der parallelen Lösung von Gleichungssystemen ausgenutzt werden.

Bei der Berechnung von Systemen mit Hilfe der FEM entstehen üblicherweise Gleichungssysteme mit schwachbesetzten Steifigkeitsmatrizen. Für die Programmiersprache Java stehen als Bibliothek verschiedene Implementierungen zur Speicherung von Matrizen zur Verfügung. Bezüglich schwach besetzter Matrizen erwies sich die Matrizenbibliothek COLT [11] als gut geeignet, da sie den Verbrauch von Speicher für Null-Elemente verhindert und trotzdem einen schnellen Zugriff auf einzelne Elemente gewährt.

Der Aufbau des Programms ist in Abbildung 7-1 auf nachfolgender Seite in Form eines Klassendiagramms dargestellt. Darin ist ein Ausschnitt der implementierten Gleichungslöser sowie der zugehörigen Klassen wie z.B. die Präkonditionierer zu sehen. Die eingeführten Schnittstellen - am vorangestellten „I“ (Interface) zu erkennen - ermöglichen eine übersichtliche Gliederung des Programms zur Wahl eines geeigneten Gleichungslösers vor einer Berechnung, ohne Änderungen am Programmkern vorzunehmen, und bieten eine leichte Erweiterungsmöglichkeit.

Wie schon aus Kapitel 3 hervorgegangen ist, ist trotz des hohen Parallelisierungsgrads (10) bei einer verteilten Berechnung eine Kommunikation zur Lösung des Gleichungssystems $\boldsymbol{Ax} = \boldsymbol{b}$ nötig, mit dem Ziel additiv verteilte Daten in akkumulierte Daten umzuwandeln. Dies resultiert aus der Anwendung der nicht-überlappenden Gebietszerlegung, bei der die finiten Elemente einem Teilgebiet zugeordnet, und die Knoten in Innen- (10) und Koppelknoten (10) unterteilt werden. Dadurch werden bei der parallelen Berechnung auf den einzelnen Prozessoren für die Koppelknoten Daten ermittelt, die zur Weiterrechnung erst durch einen Datenaustausch und Aufsummierung mit dem Host-Agenten einer Typumwandlung unterzogen werden müssen. Auf die Umsetzung der Typumwandlung wird am Ende dieses Kapitels ausführlicher eingegangen.

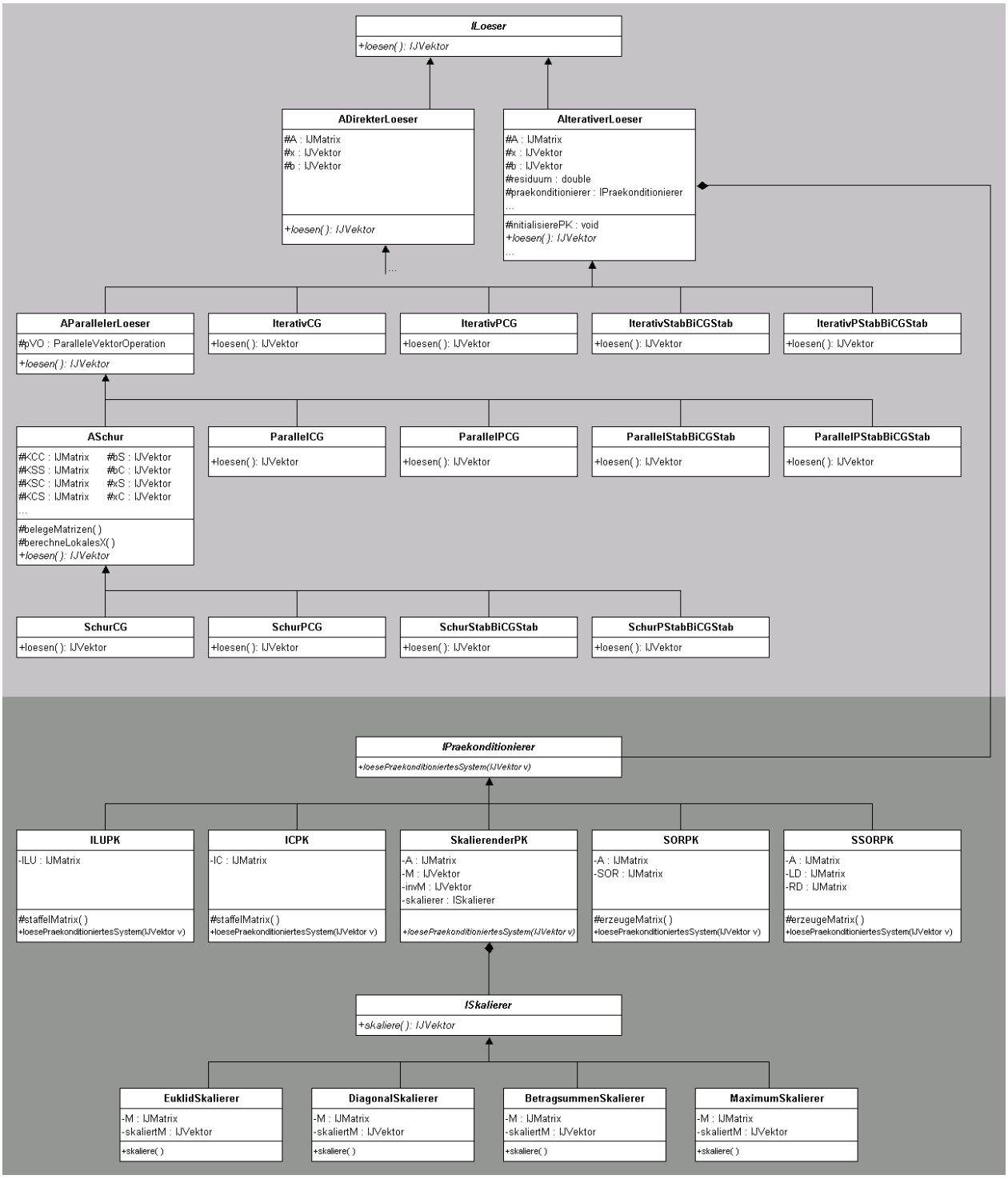

Abbildung 7-1: Klassendiagramm (Ruben)

Die Umsetzung der Parallelisierung erfolgt mit Hilfe von sogenannten mobilen Software-Agenten. Für jedes Teilgebiet werden auf dem „Startrechner“ - die einzelnen Rechner kommunizieren jeweils nur mit dem Host (Sterntopologie) - Berechnungsagenten erzeugt, die zur Durchführung der Berechnung auf einen anderen Rechner migrieren. Nach der Migration können je nach Bedarf zusätzlich benötigte Klassen von der Agentenplattform über das Netz nachgeladen werden [12].

Als Plattform hat sich AGLETS [13] als äußert zuverlässig erwiesen, da sie differenziert einstellbare Zugriffsberechtigungen zulässt, und somit im schlimmsten Fall das Ausführen von zerstörerischen Codes der Agenten verhindert werden.

Die einfache Bereitstellung von Rechnerkapazitäten mittels Agententechnologie - die Einrichtung einer Agentenplattform genügt - ermöglicht eine effiziente Nutzung großer Rechnerressourcen für numerische Berechnungen auch außerhalb der Arbeitszeiten, zum Beispiel nachts oder an Wochenenden.

Die in diesem Kapitel zuvor beschriebenen Techniken wurden im verteilten Multi-Agentensystem PAR.A.FEM (PARallele Agenten FEM) auf der Basis von mobilen Softwareagenten umgesetzt. PAR.A.FEM wird auf einem beliebigen Rechner, der als Host dient, gestartet. Nachdem der Nutzer die nötigen Daten des zu berechnenden Systems eingegeben hat, erzeugt PAR.A.FEM einen Host-Agenten. Der Host-Agent erzeugt eine vom Nutzer vorgegebene Anzahl von Client-Agenten. Diese wandern, nachdem sie die Netzinformationen für ihr Teilgebiet eingelesen haben, auf einen zur Verfügung stehenden Rechner und starten die parallele Berechnung. Solange vollkommen parallel, das heißt ohne jegliche Kommunikation der Agenten - wie zum Beispiel beim Aufstellen der Teilsteifigkeitsmatrizen - gerechnet wird, arbeiten die Prozessoren mit maximaler Effizienz. Bei der Erzeugung der Client-Agenten wird auf dem Host ein virtueller Cluster instanziiert. Dieser ist dafür zuständig für jeden Client eine Prozessor-Instanz zu erzeugen, die als Schnittstelle zwischen Host- und jeweiligem Client-Agenten die in Kapitel 3 erforderlichen Typumwandlungen der Daten steuert.
Die Clients kommunizieren mit dem Host-Agenten ausschließlich über eine Instanz der Klasse *ParalleleVektorOperation* (PVO). In dieser Klasse sind die Methoden implementiert, die die notwendigen Operationen für die parallele Gleichungslösung durchführen.
Dabei wird die Aufsummierung der entsprechenden Daten für die Typumwandlung durch eine global eindeutige Bezeichnung jedes einzelnen Werts gewährleistet. Die Aufgabe der eindeutigen Identifizierung übernimmt die Klasse *Zuordnungszeile*, in der folgende Attribute der Freiheitsgrade in einer Liste zusammengestellt werden:

1. Angabe des Substrukturtyps (Knoten, Kante, Fläche oder Körper; im folgenden als Zeile bezeichnet) und der angrenzenden Knoten. (Die Substrukturen Kante Fläche und Körper erscheinen nur bei Anwendung der p-Version.)
2. Freiheitsgrad der Zeile.
3. Identifizierung der Koppelwerte.

4. Angabe des Polynomgrads der Zeile bei der Anwendung der p-Version.
5. Polynomzähler zur Unterscheidung der verschiedenen Flächenpolynome (ebenfalls nur bei Anwendung der p-Version notwendig).

Wie schon erwähnt erfolgt die Aufstellung der Blockmatrizen für die Schurkomplement-Methode ebenfalls mit Hilfe der Klasse *Zuordnungszeile*.

Zu Beginn einer parallelen Gleichungslösung sendet jeder Client eine Liste mit Zuordnungszeilen an den Host, der, nachdem er alle Listen empfangen hat, Regeln zur Summation bei der Typumwandlung aufstellt. Diese Regeln gelten solange die Clients neue Zuordnungszeilen für eine neue Typumwandlung senden. Die Verantwortung einer dynamischen Anpassung des Systems liegt somit bei den Clients.

Die Verantwortung des Hosts liegt darin, nachstehend aufgezählte Methoden, die von den PVOs der einzelnen Clients übertragen werden, durchzuführen.

Tabelle 7-1: Methoden der Klasse *Host*

addKoppelzeilen	Fügt die Zuordnungszeilen des Senders zum globalen Zuordnungsvektor des Hosts hinzu.
vektorTypUmwandlung	Wandelt den gesendeten additiv verteilten Vektor in einen akkumulierten um.
normAddierdenerVektor	Berechnet die Norm eines additiv verteilten Vektors.
normÜberlappenderVektor	Berechnet die Norm eines akkumulierten Vektors.
addiereZahl	Wandelt eine verteilte Zahl in eine akkumulierte Zahl um.
addiereBruch	Wandelt einen verteilten Bruch in eine akkumulierte Zahl um.

Bei der Ausführung der Operationen aus Tabelle 7-1 wird der Synchronisation besondere Aufmerksamkeit beigemessen, um eventuelle Deadlocks (10) zu vermeiden. Bestimmte Zustände, die sich im Laufe der Berechnung ergeben, werden sowohl im Host als auch in den Clients durch bool`sche Variablen abgebildet und durch Kontroll-Threads verwaltet.

In Abbildung 7-2 wird die parallele Vektortypumwandlung in einer Prinzipskizze und in Abbildung 7-3 der Nachrichtenaustausch in einem Sequenzdiagramm dargestellt.

Für die anderen Typumwandlungen gemäß Tabelle 7-1 funktioniert der Nachrichtenaustausch analog zu Abbildung 7-3.

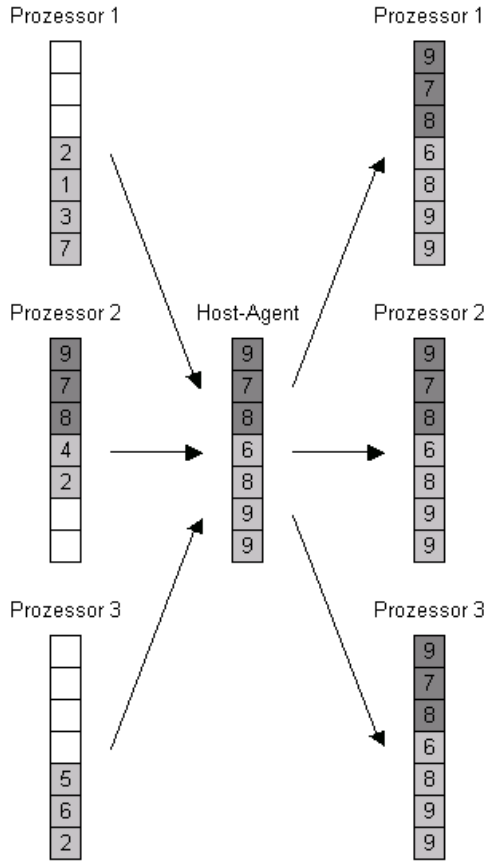

Abbildung 7-2: Prinzipskizze zur Vektortypumwandlung

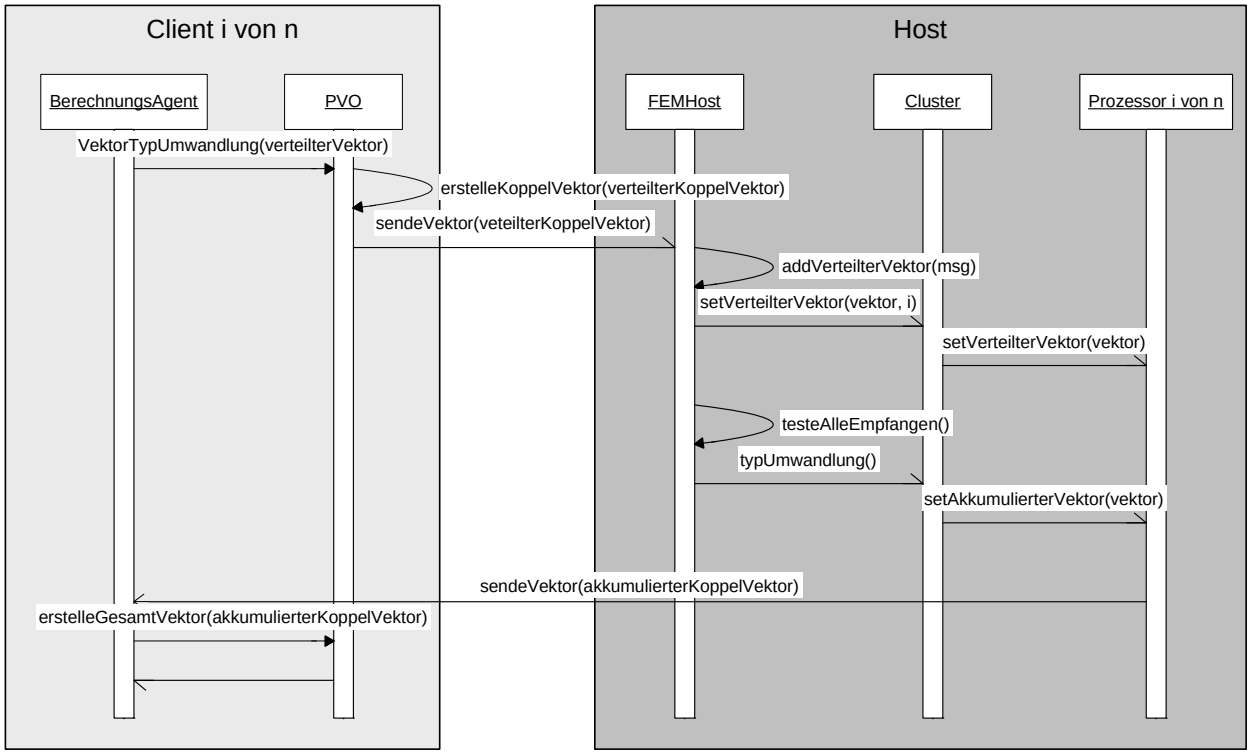

Abbildung 7-3: Sequenzdiagramm (Ruben)

8 Zusammenfassung und Ausblick

Im Rahmen diese Diplomarbeit wurden verschiedene Präkonditionierungs-Techniken im Hinblick auf Konditionsverbesserung der Steifigkeitsmatrix, Rechengeschwindigkeit, Parallelisierungsmöglichkeit, Flexibilität und Anwendungsbereich untersucht und beurteilt. Des Weitern wurde die Schurkomplement-Methode zur Effizienzsteigerung von parallelen Gleichungslösern vorgestellt und bewertet.

Die iterativen Gleichungslöser aus der Gruppe der konjugierten Gradientenverfahren eignen sich gleichermaßen gut für eine Parallelisierung von großen Gleichungsssystemen. Im Vergleich zur sequentiellen Implementierung besteht der zusätzliche Aufwand in der Kommunikation und Addition von gekoppelten Teilergebnissen. Die erwünschte Rechenzeitverkürzung wird jedoch erst ab einer Unbekanntenzahl von mehr als 4000 deutlich. Ist ein System zu klein, ist der Zeitbedarf für die Kommunikation bei einer Aufteilung in mehrere Gebiete größer als die benötigte Zeit bei einer sequentiellen Lösung.

Bei einer Netzverfeinerung und damit verbundenen Erhöhung der Freiheitsgrade eines Systems mit der p-Version wirkt sich eine Normierung der Formfunktionen positiv auf die Konvergenzgeschwindigkeit der iterativen Verfahren aus. Jedoch ist dieser Effekt nicht mit einer Präkonditionierung zu vergleichen, da die Normierung in Verbindung mit einer Präkonditionierung bei den hier ausgewerteten Beispielen keine Auswirkung hatte.

Die Wahl des Präkonditionierers ist von ausschlaggebender Bedeutung für die Effizienz des resultierenden iterativen Verfahrens.
Unter den skalierenden Präkonditionierern haben der Jakobi- und der Maximum-Skalierer die beste Konditionsverbesserung zur Folge und benötigen die kürzesten Lösungszeiten. Erheblicher Vorteil des Jakobi- gegenüber dem Maximum-Skalierer ist die kommunikationsarme Implementierung für eine parallele Anwendung. Dagegen liefern die Skalierer bezüglich der Betragssummennorm und der euklidischen Norm, neben der geringfügig höheren Lösungszeit, die schlechteste Konditionsverbesserung.

Bei optimaler Wahl des Relaxationsparameters ω resultieren aus der SSOR-Präkonditionierung für symmetrische Matrizen und der SOR-Präkonditionierung für unsymmetrische Matrizen kleinere Iterationszahlen als aus der Jakobi-Skalierung. Der große Nachteil dieser Methoden ist

jedoch die aufwendige Suche nach dem optimalen Relaxationsparameter, der für jedes System und jede Netzverfeinerung neu ermittelt werden muss. Da man anhand einer groben Diskretisierung für ein bestimmtes System nicht auf den optimalen Relaxationsparameter des feiner diskretisierten Systems schließen kann, sind Relaxations-Präkonditionierer durch den Zeitaufwand bei der notwendigen Suche des optimalen Relaxationsparameters nicht sinnvoll anwendbar. Aufgrund der besseren Approximation an die Inverse der Steifigkeitsmatrix liefern das SOR- (unsymmetrische Matrizen) und das SSOR-Verfahren (symmetrische Matrizen) für $\omega = 1$, wenn es sich dabei nicht schon um den optimalen Relaxationsparameter handelt, ebenso bessere Iterationszahlen als die Skalierer. Anhand des Boden-Modells (redA: 5744 FG) hat sich jedoch herausgestellt, dass sich bei größeren Systemen die Iterationszahl zwischen Jakobi- und splitting-assoziierten Präkonditionierern nur unwesentlich voneinander unterscheidet. Zudem benötigt der SOR- und somit auch der SSOR-Präkonditionierer bei ähnlich großen Systemen die gleiche Rechenzeit wie der Grundalgorithmus. Es wird die Vermutung aufgestellt, dass die splitting-assoziierten Präkonditionierer bei über 6000 Freiheitsgraden sogar längere Rechenzeiten benötigen als die jeweiligen Grundalgorithmen.

Bei symmetrischen Matrizen hat sich bestätigt, dass der IC-Präkonditionierer die selbe Anzahl an Iterationsschritten wie der ILU-Präkonditionierer benötigt, jedoch aufgrund der Symmetrie-ausnutzung bei der Implementierung einen Geschwindigkeitsvorteil hat und sogar im Rahmen der Auswertung teilweise die kürzeste Lösungszeit verglichen mit den übrigen hier erwähnten Präkonditionierern liefert. Folglich wird der ILU-Präkonditionierer zur Lösung von symmetrischen Gleichungssystemen nicht empfohlen. Als unter den hier aufgeführten Präkonditionierern beste Approximation an die Inverse der Steifigkeitsmatrix ist die ILU-Zerlegung beziehungsweise die IC-Zerlegung für größere Systeme jedoch als Präkonditionierer ungeeignet. Der Grund dafür ist, dass für Steifigkeitsmatrizen ohne Bandstruktur - p-Version und Steifigkeitsmatrizen mit Bandstruktur schließen sich gegenseitig aus - die Staffelung in eine linke untere und rechte obere Dreiecksmatrix länger dauert als die Lösung des linearen Gleichungssystems mit dem Grundalgorithmus.
Aus dem selben Grund kann die Schurkomplement-Methode, trotz Erhöhung des Parallelisierungsgrads und niedrigster Iterationszahlen, nicht effizient bei Anwendung der p-Version zur Netzverfeinerung eingesetzt werden, da die Berechnung der Innenknoten-Verschiebungen mit einem direkten, und für große Systeme deshalb zeitaufwendigen Verfahren (hier der LU-Algorithmus), erfolgt.

Schließlich lässt sich sagen, dass alle untersuchten Präkonditionierer zu einer Konditionsverbesserung der Steifigkeitsmatrix führen. Jedoch hat der Jakobi-Skalierer die eingangs (Kapitel 4) genannten Anforderungen:

- einfache Berechnung (Rechenzeit),
- geringer Speicherplatzbedarf,
- gute Approximation an die Inverse der Steifigkeitsmatrix $\boldsymbol{A}$ → leichte Invertierbarkeit (Effektivität),
- breites Anwendungsgebiet (Flexibilität und Stabilität) und
- leichte Implementierung zur parallelen Anwendung

am besten erfüllt.

Die Schurkomplement-Methode ist bei parallelen Rechnungen prinzipiell sehr empfehlenswert, hat allerdings aufgrund der direkten Lösungsverfahren für die Innenknoten noch erhebliche Geschwindigkeitsnachteile. Es bleibt zu prüfen, ob die Anwendung von iterativen Lösungsverfahren für die Innenknoten bei der Schurkomplement-Methode diese Geschwindigkeitsnachteile beheben kann.

9 Verwendete Symbole

Griechische Symbole:

$\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}}$	Konditionszahl der Steifigkeitsmatrix $\boldsymbol{A}$
λ	Eigenwert einer Matrix
ν	Querkontraktionszahl
ω	Relaxationsparameter

Lateinische Symbole:

$\boldsymbol{A}$	Steifigkeitsmatrix
$\boldsymbol{A}^* = (\bar{a}_{ji})$	adjungierte Matrix
$\boldsymbol{D} = \mathrm{diag}\{a_{11},\ldots,a_{nn}\}$	Diagonalmatrix
E	Elastizitätsmodul
$\boldsymbol{F}$	Fehlermatrix bzw. Restmatrix
G	Schubmodul
$\boldsymbol{I}$	Einheitsmatrix
$\boldsymbol{L}$	reguläre linke untere Dreiecksmatrix
$\boldsymbol{P}$	Prä- bzw. Vorkonditionierungsmatrix
$\boldsymbol{Q}$	orthogonale Matrix
$\boldsymbol{R}$ bzw. $\boldsymbol{U}$	reguläre rechte obere Dreiecksmatrix
$\boldsymbol{U}$ bzw. $\boldsymbol{R}$	reguläre rechte obere Dreiecksmatrix

$\boldsymbol{b}$	Lastvektor
n	Anzahl der Freiheitsgrade eines Systems
$\boldsymbol{r}$	Residuenvektor
$\lVert \boldsymbol{r} \rVert_2$	Residuum, euklidische Norm, Länge des Vektors r
$\boldsymbol{x}$	Unbekanntenvektor bzw. Verschiebungsvektor

10 Begriffserklärungen

Additiv verteilte Daten:
Beim parallelen Rechnen werden die Verschiebungen der Koppelknoten durch mehrere Prozessoren berechnet. Die Gesamtverschiebung eines Koppelknotens ergibt sich aus der Addition der Teilergebnisse jedes Prozessors. Die Teilergebnisse werden als additiv verteilte Daten bezeichnet. (siehe *Akkumulierte Daten*)

Akkumulierte Daten (bzw. global verteilte Daten):
Akkumulierte Daten sind auf das Gesamtsystem bezogene Ergebnisse, die aus den Teilergebnissen der betreffenden Prozessoren zusammengesetzt wurden. Liegen die Ergebnisse des Gesamtsystems auf den Clients vor, spricht man von einer akkumulierten bzw. globalen Datenhaltung. (siehe *Additiv verteilte Daten*)

Deadlock:
Beim Datenaustausch zwischen einem Sender und Empfänger werden folgende Prozesse durchlaufen: Senden, Empfangen, Bestätigung senden, Bestätigung empfangen.
Wenn sich zwei Prozessoren gegenseitig und exakt gleichzeitig Daten senden, kommt es zu einer „Datenkollision“. Beide Prozessoren sind dadurch gleichzeitig Sender und Empfänger und durch die Kollision können keine Empfangsbestätigungen gesendet werden, wodurch der Prozess zum Stillstand gebracht wird.

Direkte Verfahren:
Direkte Verfahren liefern nach einer vorher bekannten Anzahl von Rechenschritten eine bis auf die Rundungsfehler exakte Lösung. Das bekannteste direkte Verfahren ist die Gaußelimination. (siehe *Iterative Verfahren*)

Euklidische Norm:
Die Länge eines Vektors wird als euklidische Norm bezeichnet.

Frobeniusmatrix:
„Derartige Matrizen, die sich höchstens in einer Spalte von der Einheitsmatrix unterscheiden werden als Frobeniusmatrizen bezeichnet“ ([7], 33).

Funktional:
Die Formulierung eines Funktionals beschreibt die Menge der Funktionen, die die Randbedingungen des Systems erfüllen. Durch Minimierung des Funktionals erhält man die Lösungsfunktion der Differentialgleichung, die die gegebenen Randbedingungen erfüllt.

h-Version:
Mit der h-Version bezeichnet man eine Netzverfeinerung, bei der die Elemente verkleinert und somit ihre Anzahl und der Rechenaufwand erhöht wird. (siehe *p-Version*)

Innenknoten:
Knoten, die bei einer Gebietszerlegung vollständig innerhalb eines Teilgebiets liegen. (siehe *Koppelknoten*)

Iterative Verfahren:
Iterative Verfahren berechnen, ausgehend von einem Startvektor, eine Folge von Iterierten.
Sie benötigen ein Abbruchkriterium, dass eine gewünschte Genauigkeit der iterierten Lösung an die exakte Lösung vorgibt, da niemals die Lösung exakt berechnet wird, und somit das Verfahren unendlich rechnen würde. (siehe *Direkte Verfahren*)

Koppelknoten:
Knoten, die bei einer Gebietszerlegung auf den Rändern angrenzender Teilgebiete liegen. (siehe *Innenknoten*)

Orthogonale Matrix:
Eine orthogonale Matrix $\boldsymbol{Q}$ besitzt folgende Eigenschaften:
- Die Inverse der Matrix entspricht der Transponierten der Matrix: $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^{T}$.
- Die Transponierte der Matrix multipliziert mit der Matrix entspricht somit auch der Einheitsmatrix: $\boldsymbol{Q}^{T}\boldsymbol{Q} = \boldsymbol{I}$.
- Eine orthogonale Matrix besitzt immer eine Konditionszahl von $\kappa(\boldsymbol{Q}) = 1$.

Parallelisierung:
Bei einer Parallelisierung wird ein zu lösendes System in mehrere Teilgebiete aufgeteilt. Die Teilgebiete werden je von einzelnen Prozessoren (Clients) gelöst. Nach Abschluss des Rechenablaufs werden die Ergebnisse vom Hostagenten wieder zusammengesetzt.

Parallelisierungsgrad:
Je höher der Parallelisierungsgrad einer Berechnung ist, desto geringer ist der Kommunikationsaufwand zwischen den einzelnen Prozessoren.

p-Version:
Bei Anwendung der p-Version zur Netzverfeinerung werden die finiten Elemente nicht verkleinert, sondern ein höherer Polynomgrad für die Ansatzfunktion gewählt. Das heißt, daß die Anzahl der Elemente unverändert bleibt, jedoch die Freiheitsgrade pro Element ansteigen. (siehe *h-Version*)

Skalierbarkeit:

Bei einer fehlenden Skalierbarkeit gibt es keinen proportionalen Zusammenhang zwischen Rechenleistung und Berechnungszeit. Das heißt, ein Hinzufügen von zusätzlicher Rechenleistung zur Berechnung eines Gleichungssystems verkürzt die Rechenzeit nicht.

11 Literaturverzeichnis

[1] Düster, A.: *High order finite elements for three-dimensional, thin-walled nonlinear continua.* Shaker Verlag, Aachen, 2002

[2] Greb, S.: *Entwicklung eines Softwaretools zur Bewertung von Partitionierungen.* Institut für Numerische Methoden und Informatik im Bauwesen, TU-Darmstadt, 1999

[3] Gruttmann, F.: *Finite Elemente in der Baustastik I.* Skriptum, Institut für Statik, TU-Darmstadt, 2001

[4] Hackbusch, W.: *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* 2. Auflage, B. G. Teubner Verlag, Stuttgart, 1993

[5] Lämmer, L.: *Parallelisierung von Anwendungen der Finite-Element-Methode im Bauingenieurwesen.* Dissertation, Institut für Numerische Methoden und Informatik im Bauwesen, TH-Darmstadt, 1996

[6] Meißner, U.F., Maurial, A.: *Die Methode der finiten Elemente. Eine Einführung in die Grundlagen.* 2. Auflage, Springer Verlag, Heidelberg, 2000

[7] Meister, A.: *Numerik linearer Gleichungssysteme. Eine Einführung in moderne Verfahren.* Vieweg Verlag, Hamburg, 1999

[8] Müller, M., Ruben, J.: *Aspekte der Agentenbasierten adaptiven FE-Simulation.* Entwurf, Institut für Numerische Methoden und Informatik im Bauwesen, TU-Darmstadt, 2003

[9] Saad, Y.: *Iterativ methods for sparse linear systems*. PWS Publishing Company, Boston, 1996

[10] Shewchuk, J.R.: *An introduction to the conjugate gradient method without the agonizing pain.* School of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA 15213, 1994

[11] http://hoschek.home.cern.ch/hoscheck/colt/, Referenziert in [8]

[12] Meißner, U.F., Rüppel, U., Theiß, M.: *Verteilte Brandschutzmodellierung auf der Basis von Software-Agenten.* In: Tagungsband der Fachtagung „Bauen mit Computern – Kooperation in IT-Netzwerken“, Bonn 2002, Fortschritt-Bericht VDI, April 2002, S. 557-569, Referenziert in [8]

[13] http://www.trl.ibm.com/aglets, Referenziert in [8]

Anhang

Abbildungen

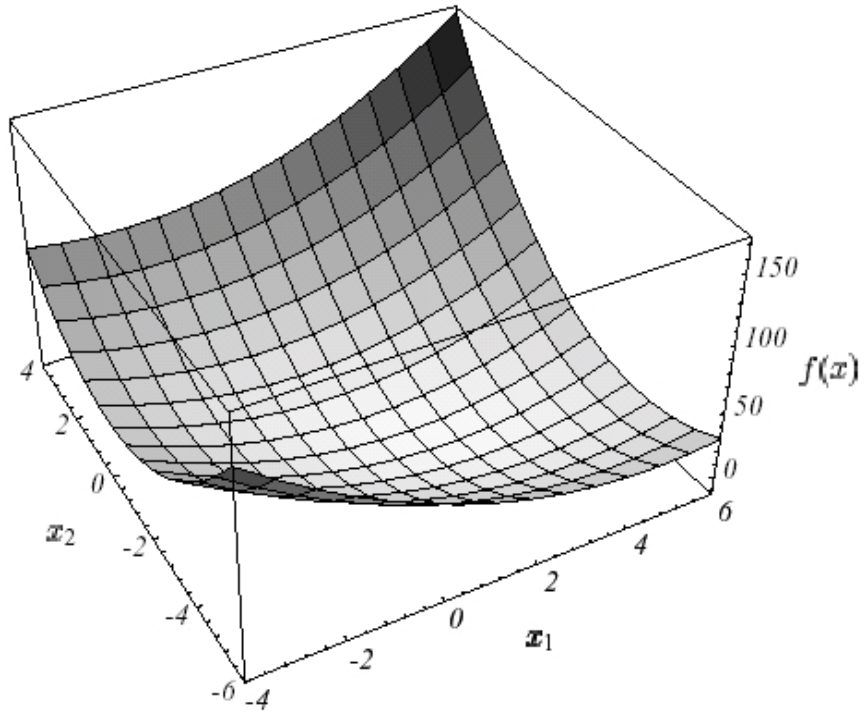

Abbildung 1: Graph einer quadratischen Gleichung f(x). Das Minimum des Paraboloids entspricht der Lösung von ***Ax*** = ***b***; aus [10]

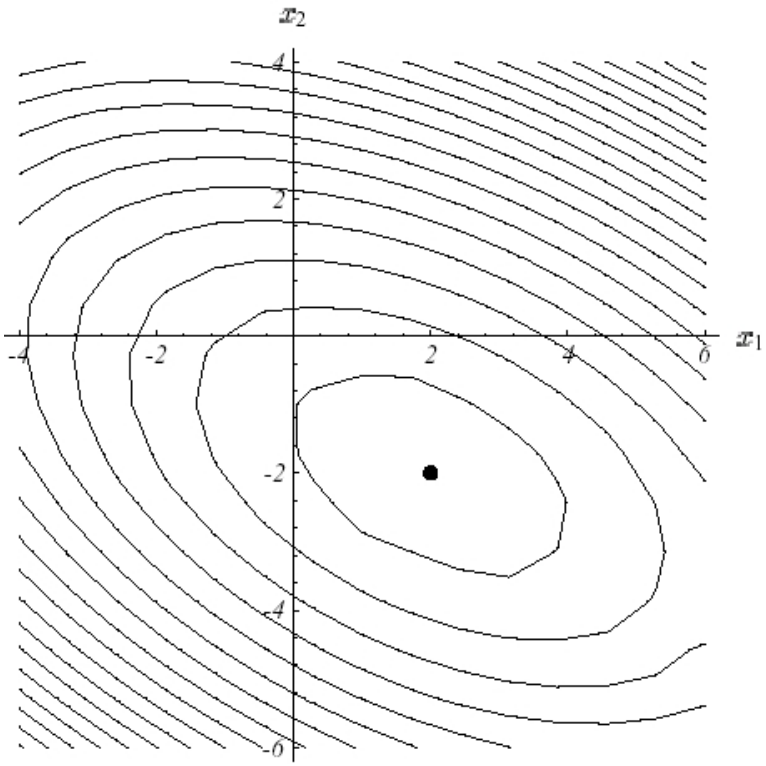

Abbildung 2: Draufsicht auf das Paraboloid aus Abbildung 1; aus [10]

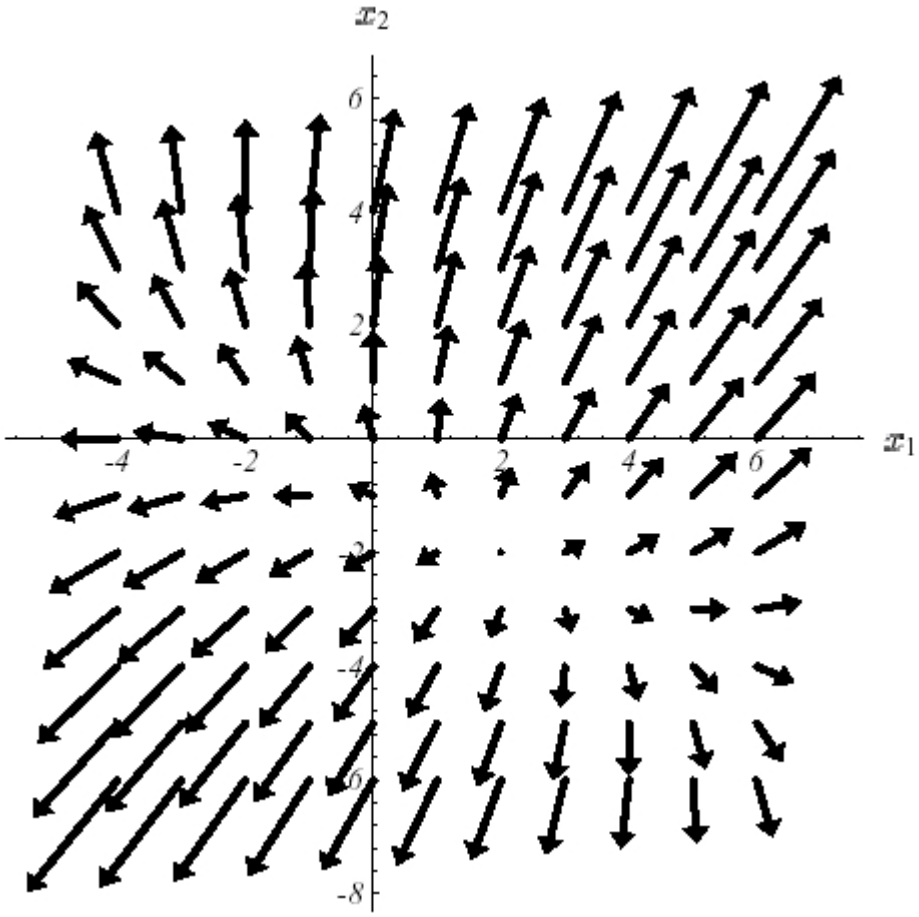

Abbildung 3: Gradienten des Paraboloids; aus [10]

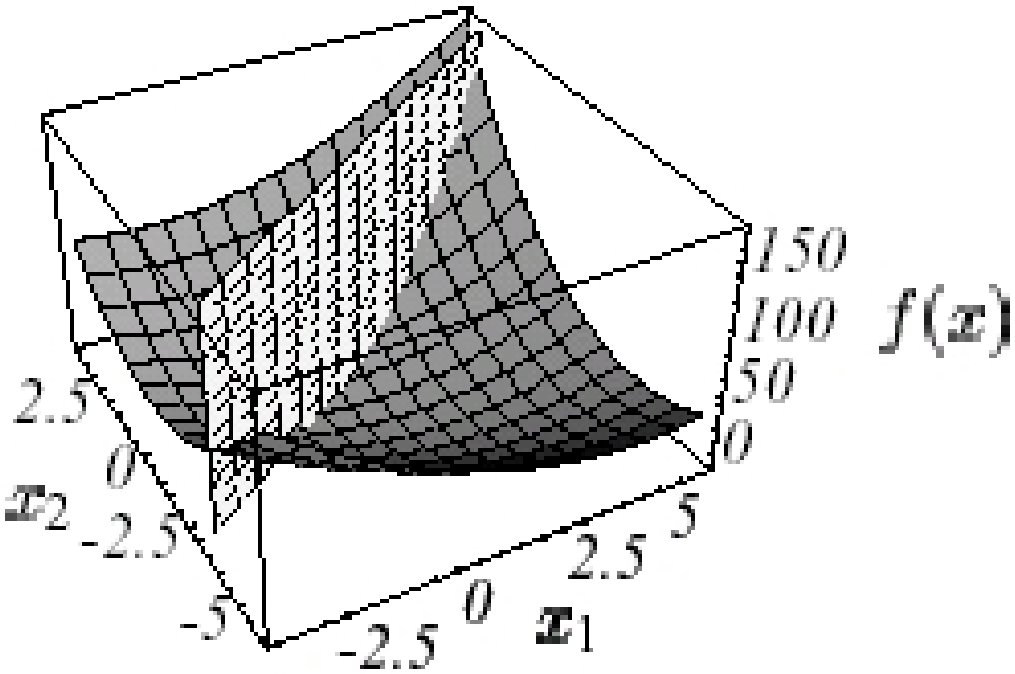

Abbildung 4: Schnittmenge zwischen Paraboloid und Suchvektor; aus [10]

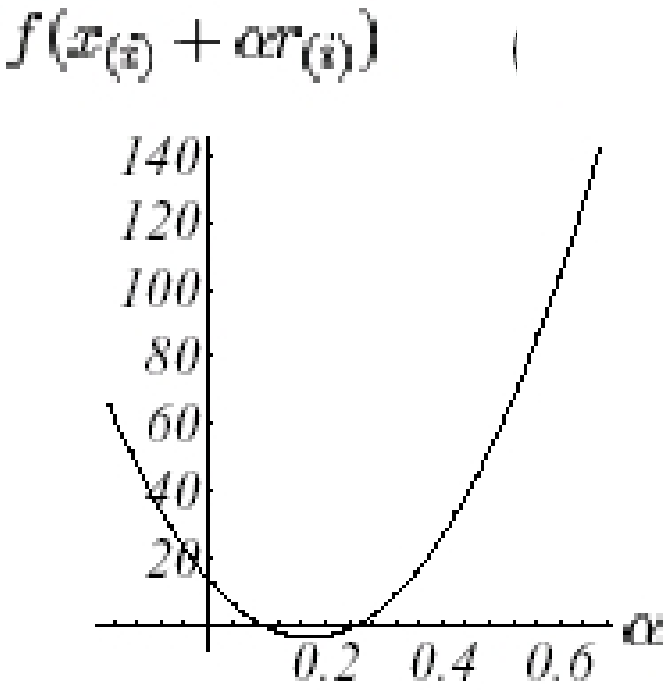

Abbildung 5: Darstellung der Schnittmenge aus Abbildung 4 als Seitenansicht; aus [10]

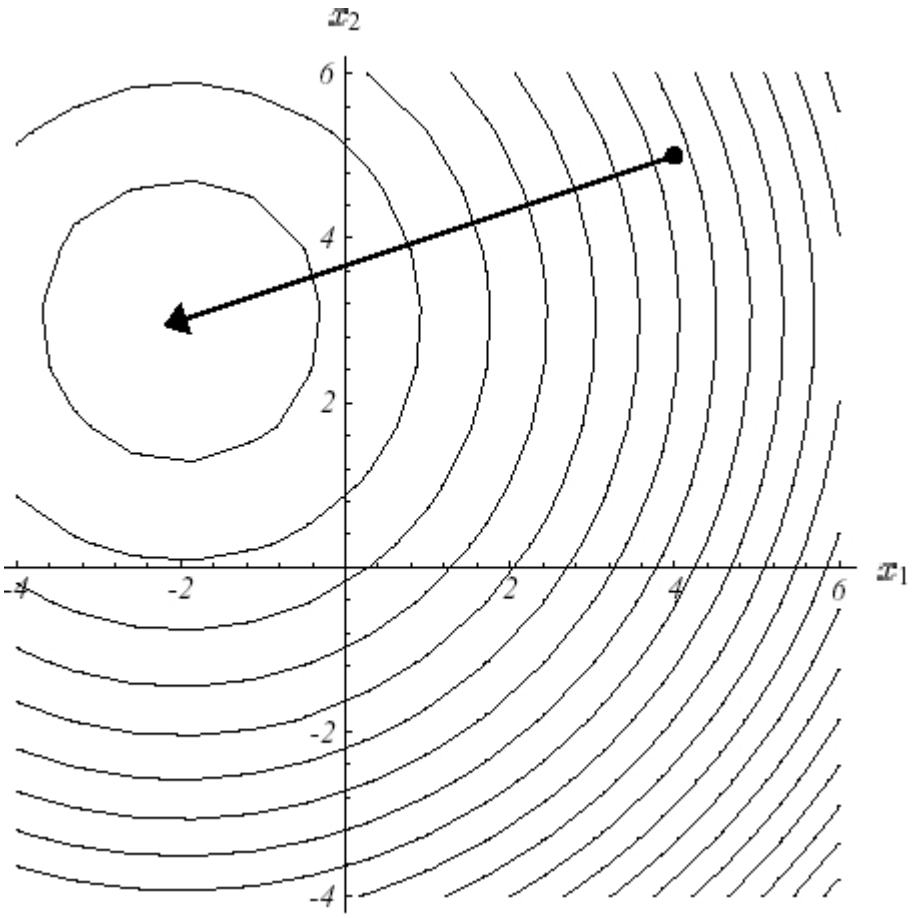

Abbildung 6: Sphärische Isolinien des Paraboloids bei gleichen Eigenwerten, κ = 1; aus [10]