Diplomarbeit

Ein paralleles graphenbasiertes algebraisches Mehrgitterverfahren

Angefertigt am
Institut für Numerische Simulation

Vorgelegt der

Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

September 2004
Von
Bram Metsch
Aus
Haarlem

Inhaltsverzeichnis

Kapitel 1

Einleitung

Viele Probleme in den Naturwissenschaften und den Ingenieurwissenschaften lassen sich in einem mathematischen Modell darstellen. Ein solches Modell beschreibt die Beziehungen zwischen den für dieses Phänomen wichtigen Größen, wie zum Beispiel Kräfte, Geschwindigkeiten oder Materialeigenschaften, meist in Form einer Differenzialgleichung oder eines Systems von Differenzialgleichungen. Die Lösung solcher Differenzialgleichungen ist selten analytisch möglich, so dass auf eine numerische Behandlung zurückgegriffen werden muss. Dazu wird die Differenzialgleichung mittels finiter Differenzen, finiter Elemente oder finiter Volumen diskretisiert. Eine solche Diskretisierung führt zu einem algebraischen (linearen oder nichtlinearen) Gleichungssystem mit endlich vielen skalaren Unbekannten, welches auf einem Computer dargestellt werden kann.
Wir beschränken uns auf lineare Gleichungssysteme. Gegeben ist also ein Gleichungssystem der Form

$$Au = f$$

wobei $A \in \mathbb{R}^{N \times N}$ eine reellwertige $N \times N$-Matrix, $f \in \mathbb{R}^N$ die vorgegebene rechte Seite und $u \in \mathbb{R}^N$ der gesuchte Lösungsvektor ist.
Die Lösung eines solchen Gleichungssystems mittels einer direkten Methode, etwa der LU- oder der Cholesky-Zerlegung, erfordert einen Aufwand von $O(N^3)$ arithmetischen Operationen und einen Speicheraufwand von $O(N^2)$ reellen Werten. Nun sind aber diese aus einer Diskretisierung hervorgegangenen Systemmatrizen in der Regel *dünnbesetzt*, d.h. sie enthalten pro Matrixzeile unabhängig von der Problemgröße nur wenige Einträge $a_{ij} \neq 0$, was den Speicheraufwand für das lineare Gleichungssystem auf $O(N)$ reduziert. Für solche Systeme stellt sich eine direkte Lösung als ungünstig heraus. Der Speicheraufwand von $O(N^2)$ für die direkten Verfahren würde die maximale Problemgröße viel stärker begrenzen, als dies durch das Problem selbst gegeben wäre. Andererseits impliziert der Rechenaufwand von $O(N^3)$, dass

bei einer Verdopplung der Rechengeschwindigkeit die Problemgröße nur um den Faktor $\sqrt[3]{2} \approx 1.26$ anwachsen kann, wenn das Gleichungssystem in derselben Zeit gelöst werden soll. Nun geht jedoch meist mit der Verdopplung der Rechengeschwindigkeit auch eine Verdopplung des verfügbaren Speichers einher, so dass wir auch doppelt so große Systeme lösen wollen. Wir müssen also einen Lösungsalgorithmus entwickeln, der nur $O(N)$ arithmetische Operationen und $O(N)$ Speicherplatz beansprucht. Solche Verfahren werden als *optimal* bezeichnet.
Es bietet sich die Verwendung iterativer Löser, wie zum Beispiel das Gauß-Seidel-Verfahren oder (für symmetrische Matrizen) des Verfahrens der konjugierten Gradienten an. Diese Verfahren erfordern pro Iterationsschritt einen Rechenaufwand von $O(N)$ und gegebenenfalls einen ebenso großen Speicheraufwand zur Speicherung eines Residualvektors.
Die Konvergenzrate dieser Verfahren ist abhängig von der Kondition der Matrix, welche sich bei zunehmender Anzahl der Unbekannten verschlechtert. Ein Verfahren, welches in $O(N)$ angewendet werden soll, muss jedoch den Fehler innerhalb weniger Iterationsschritte unabhängig von N auf ein vorgegebenes Maß reduzieren.
Ein Ansatz zur Beschleunigung iterativer Lösungsverfahren stellen Mehrgitterverfahren ([Hac85]) dar. Sie nutzen eine bestimmte Eigenschaft der Fehlerreduktion klassischer Iterationsverfahren wie des Gauß-Seidel-Verfahrens oder des Jacobi-Verfahrens aus: Während niederfrequente Fehleranteile nur langsam gedämpft werden, verschwinden die hochfrequenten Fehleranteile schon nach wenigen Iterationsschritten. Wir sprechen von der *Glättung* des Fehlers und nennen diese Iterationsverfahren auch *Glättungsverfahren*. Nun kann der geglättete Fehler auf einem Gitter mit einer größeren Maschenweite schon hinreichend genau dargestellt werden. Es bietet sich damit an, das Residuum auf ein solches gröberes Gitter zu übertragen und dort als rechte Seite für ein entsprechend formuliertes Gleichungssystem zu verwenden. Die Lösung dieses Gleichungssystems transferieren wir wieder auf das feine Gitter zurück und datieren damit die Lösung auf. Dieser Vorgang wird *Grobgitter-Korrektur* genannt. Das Verfahren kann rekursiv fortgesetzt werden, bis auf einem Grobgitter mit nur wenigen Freiheitsgraden eine direkte Lösung ohne größere Kosten möglich ist. Bei geschickter Kombination von Glättung und Grobgitter-Korrektur können wir ein Verfahren erzielen, welches pro Iterationsschritt einen Rechenaufwand von $O(N)$ aufweist und den Fehler in $O(1)$, also unabhängig von der Problemgröße, um einen vorgegebenem Faktor verkleinert.
Die klassischen geometrischen Mehrgitterverfahren gehen dabei von einer vorgegebenen Hierarchie von Grobgittern aus, die zum Beispiel durch Verdopplung der Schrittweite $h \to 2h$ aus dem jeweils feineren Gitter erzeugt werden.

Die Interpolations- und Restriktionsoperatoren werden nur abhängig von den beiden beteiligten Gittern aufgestellt, während der Grobgitteroperator aus der Diskretisierung des Problems auf dem groben Gitter hervorgeht. Diese Vorgehensweise legt damit den Grobgitter-Korrekturschritt weitgehend fest, so dass für eine gute Fehlerreduktion der Glätter an das Problem angepasst werden muss. Es zeigt sich, dass sich für komplizierte geometrische Strukturen —insbesondere bei dreidimensionalen Problemen— die Konstruktion eines solchen Glätters sehr kompliziert gestaltet. Dieser Umstand legt nahe, nicht den Glätter, sondern die übrigen Komponenten —Grobgitter, Transferoperatoren und Grobgitteroperator— an das Problem anzupassen, während für die Glättung ein schnell auszuführendes Relaxationsverfahren (Gauß-Seidel-Verfahren oder Jacobiverfahren) eingesetzt wird.
Ein erster Schritt in die Richtung algebraischer Mehrgitterverfahren (engl. *Algebraic Multigrid Methods — AMG*) war die Entwicklung *operator-abhängiger Interpolationsoperatoren* ([ABDP], [Zee90]) Bei gegebenem Grobgitter wurde der Interpolationsoperator P abhängig nicht nur vom Grobgitter, sondern auch von der Systemmatrix aufgestellt. Hinzu kam die Aufstellung des Grobgitteroperators A_C nach dem *Galerkinansatz*, dass heisst in der Form $A_C := P^T AP$. Es zeigt sich, dass in diesem Fall die Grobgitter-Korrektur für symmetrisch, positiv definite Matrizen einem Variationsprinzip genügt, was zusammen mit der Fehlerreduktion durch die Glättung die Konvergenz des Mehrgitterverfahrens garantiert.
Algebraische Mehrgitterverfahren wurden zunächst von A. Brandt im Jahre 1982 ([BMR82], [BMR84], [Bra86]) vorgestellt. Zusätzlich zur operatorabhängigen Interpolation und dem Galerkinansatz kam jetzt die automatische operator-abhängige Grobgitterwahl. Diese Verfahren liefern für viele Probleme, in denen die Implementierung geometrischer Mehrgitterverfahren große Schwierigkeiten hervorruft, gute Ergebnisse. Insbesondere partielle Differenzialgleichungen, die zu M-Matrizen führen, können mit AMG effizient gelöst werden. Es ist sogar möglich, lineare Gleichungssysteme zu behandeln, die nicht der Diskretisierung einer Differenzialgleichung entspringen, weil der gesamte AMG-Algorithmus nur Informationen aus der Systemmatrix benötigt.
Algebraische Mehrgitterverfahren bestehen aus zwei Phasen: einer *Setupphase*, in der die Hierarchie der Grobgitter, der Transferoperatoren und der Grobgitteroperatoren konstruiert wird, und einer *Lösungsphase*, in der das Gleichungssystem mittels der zuvor konstruierten Hierarchie gelöst wird.
Um auch große Probleme mit einer Million oder mehr Unbekannten lösen zu können, sind parallele Lösungsverfahren erforderlich. Insbesondere gilt dies für den Einsatz des AMG-Algorithmus als Teil eines parallelen Lösungsverfahrens wie zum Beispiel ein Strömungslöser ([GNR98]). Die größte Heraus-

forderung bei der Parallelisierung der Setupphase stellt hierbei die parallele Grobgitterwahl dar. Die getrennte Auswahl eines Grobgitters auf jedem Prozessor führt nicht zwangsläufig zu einem geeigneten Grobgitter für das Gesamtproblem.
Ein erster Ansatz wurde von Krechel und Stüben in [KS99] im Jahre 1999 beschrieben. Hierbei wird die Setupphase auf den beteiligten Prozessoren weitgehend entkoppelt und der sequentielle Algorithmus kann angewendet werden. Mit dieser Entkopplung wird jedoch eine wichtige Eigenschaft eines algebraisch glatten Fehlers ignoriert, so dass dieses Verfahren schlechtere Konvergenzeigenschaften als der sequentielle Algorithmus für auf dasselbe Problem erzielt. In [HY01] werden Verfahren vorgestellt, die nach erfolgter Vergröberung auf jedem Prozessorteilgebiet eine gesonderte Behandlung auf den Teilrandgebieten vorgenommen. Diese Verfahren erzeugen jedoch Grobgitter, die entlang der Teilgebietsränder sehr viele Punkte aufweisen. Dies wirkt sich negativ auf den Gesamtspeicherverbrauch des Verfahrens auf.
In dieser Arbeit stellen wir eine Methode vor, die dieses Problem durch eine geschickte Grobgitterwahl auf den Teilgebieten umgeht. Wir erzeugen auf den einzelnen Prozessoren die Grobgitter in der Art, dass diese zu einem geeignetem Grobgitter für das gesamte Diskretisierungsgebiet zusammengesetzt werden können. Damit können wir auf eine teure und kommunikationsintensive Teilrandbehandlung verzichten.
In Kapitel 2 werden wir algebraische Mehrgitterverfahren vorstellen. Zunächst wiederholen wir die Funktionsweise geometrischer Mehrgitterverfahren als Beschleunigung iterativer Lösungsverfahren. Danach führen wir in das klassische AMG von Ruge und Stüben [Stü99b] ein. Wir definieren zunächt den „glatten“ Fehleranteil im Sinne des algebraischen Mehrgitterverfahrens (Abschnitt 2.3) und charakterisieren diesen für die wichtigsten Klassen von Systemmatrizen. Mit dem Wissen über das Verhalten dieses Fehleranteils können wir den Interpolationsoperator aufstellen. Wir stellen in Abschnitt 2.6 dazu zwei Möglichkeiten vor. Diese beiden Varianten stellen wiederum drei Bedingungen an die Grobgitterwahl, welche wir in Abschnitt 2.7 behandeln.
Nachdem wir alle Komponenten des Mehrgitterzyklus und des AMG formal vorgestellt haben, stellt sich die Frage nach der effizienten parallelen Implementierung. Zunächst wenden wir uns dem Mehrgitterzyklus zu. Wir stellen zunächst die parallele Implementierung des Matrix-Vektor-Produktes vor (Abschnitt 3.1) und führen eine Datenstruktur für parallele dünnbesetzte Matrizen ein. Der nächste Abschnitt behandelt die parallele Glättung, insbesondere die Schwierigkeiten bei der parallelen Durchführung des Gauß-Seidel-Verfahrens. Wir benutzen zur Parallelisierung des Mehrgitterzyklus die Datenstrukturen und Matrix-Vektor-Operationen der Programmbiblio-

thek PETSc ([BBG+03]).
Ein weiteres Problem besteht in der parallelen Aufstellung des Grobgitteroperators. Dieser Teil der Setupphase erfordert die meiste Kommunikation, es müssen auf jedem Level drei verteilt abgespeicherte Matrizen multipliziert werden.
Im 4. Kapitel beschäftigen wir uns mit der Parallelisierung der Setupphase. Nachdem wir zuerst kurz auf die parallele Aufstellung des Interpolationsoperators eingehen (Abschnitt 4.2), widmen wir uns dem Hauptproblem, der Grobgitterwahl. Wir fassen in Abschnitt 4.3 zunächst die bisherigen Parallelisierungsansätze zusammen und vergleichen ihre Stärken und Schwächen. Wir sehen, dass die Aufstellung des Interpolationsoperators und des Grobgitteroperators die weitaus rechenaufwändigsten Komponenten der Setupphase darstellen, während die Grobgitterwahl nur einen Bruchteil dieser Zeit beansprucht. Hierdurch motiviert stellen wir in Abschnitt 4.4 stellen wir mit CGC (Coarse-Grid Classification) eine neue parallele Vergröberungsstrategie vor. Diese zeichnet sich dadurch aus, dass statt einem Grobgitter pro Prozessor mehrere potenzielle Grobgitter erzeugt werden. Hiervon soll auf jedem Prozessor genau ein Gitter ausgewählt werden, so dass das resultierende Gesamtgitter keiner Teilrandbehandlung bedarf. Dazu stellen wir einen gewichteten, gerichteten Graphen auf, dessen Knoten die einzelnen potenziellen Gitter darstellen. Die Kantengewichte beschreiben, inwieweit die beiden beteiligten Knoten (Grobgitter auf benachbarten Prozessorteilgebieten) zusammengesetzt ein geeignetes Grobgitter für das zusammengesetzte Teilgebiet darstellen. Weil dieser Graph nur eine geringe Anzahl von Knoten und Kanten aufweist, kann die Auswahl der Grobgitter für die einzelnen Prozessoren sequentiell vorgenommen werden. Nach diesem Schritt erhalten wir ein Grobgitter über das gesamte Diskretisierungsgebiet, mit dem der Interpolationsoperator und der Grobgitteroperator aufgestellt werden können.
Die numerischen Vergleiche zwischen den einzelnen Vergröberungsverfahren werden in Kapitel 5 dargestellt. Wir untersuchen zunächst einige Modellprobleme mit einfacher Struktur und wenden uns danach komplizierteren Fällen wie springenden Koeffizienten zu. Wir benutzen dabei sowohl einen Cluster von Workstations als auch einen Großrechner mit mehreren Hundert Prozessoren. Es wird sich herausstellen, dass das CGC-Verfahren insbesondere beim Einsatz in einem Strömungslöser eine schnelle Lösung bei geringem Speicherverbrauch erzielt.

Kapitel 2

Algebraische Mehrgitterverfahren

In diesem Kapitel beschreiben wir die Funktionsweise von algebraischen Mehrgitterverfahren. Zuerst fassen wir in Abschnitt 2.1 die Funktionsweise klassischer (geometrischer) Mehrgitterverfahren zusammen und in den darauf folgenden Abschnitten geben wir eine Einführung in algebraische Mehrgitterverfahren.

2.1 Mehrgitterverfahren

Mehrgitterverfahren sind Methoden zur Beschleunigung eines iterativen Lösungsalgorithmus. Diese Algorithmen werden insbesondere bei dünnbesetzten linearen Gleichungssystemen (d.h. Anzahl der Matrixeinträge $\approx$ Anzahl der Unbekannten N) eingesetzt. Mit Hilfe von Mehrgitterverfahren kann ein für die Lösung eines solchen Gleichungssystems ein Rechenaufwand von $O(N)$ erreicht werden.
Die grundlegende Idee des Mehrgitteralgorithmus besteht darin, den Fehler nach einen oder mehreren Iterationsschritten aufzuspalten und die beiden Teile unterschiedlich zu behandeln. Um dies zu verdeutlichen, betrachten wir folgendes Modellproblem.

$$\begin{aligned} -\Delta \mathrm{u} &= \mathrm{f} \text{ auf } \Omega = [0,1]^2, \\ \mathrm{u} &= \mathrm{g} \text{ auf } \partial\Omega, \end{aligned}$$

mit stetigen Funktionen $\mathrm{f} \in C^0(\Omega)$, $\mathrm{g} \in C^0(\partial\Omega)$ und gesuchten Lösungsvektor $u \in C^2(\Omega)$. Wir diskretisieren den Laplaceoperator δ mit einem Differenzenverfahren (5-Punkte-Stern) auf einem Gitter der Größe $N = n \times n$ mit der

Maschenweite h.

$$\Omega^l := \{(i,j)\}_{i,j=1}^n$$

Es ergibt sich jetzt ein lineares Gleichungssystem der Form $Au = f$. Auf dieses wenden wir das gedämpfte Jacobiverfahren mit Dämpfungsparameter ω an, dabei bezeichne u^k die k-te Iterierte des Verfahrens:

$$u^{k+1} = u^k + \omega D^{-1}(f - Au^k)$$

Für den Fehler $e^k = u - u^k$ gilt:

$$e^{k+1} = e^k + \omega D^{-1} A e^k$$

Die Eigenvektoren ϕ_{ij} und Eigenwerte λ_{ij} $(i,j = 1,\dots n-1)$ von $D^{-1}A$ lauten:

$$\begin{aligned} \phi_{i,j} &= \sin(i\pi x)\sin(j\pi y), \qquad (x,y) \in \Omega^l \\ \lambda_{i,j} &= \left(1 - \frac{1}{2}\cos(i\pi h) - \frac{1}{2}\cos(j\pi h)\right) \end{aligned}$$

Wir betrachten nun den Anteil des Fehlers $\phi_{i,j}^k$ zum Eigenwert $\lambda_{i,j}$:

$$\phi_{i,j}^{k+1} = \left(1 + \omega\left(1 - \frac{1}{2}\cos(i\pi h) - \frac{1}{2}\cos(j\pi h)\right)\right)\phi^k$$

Wir beobachten, dass für kleine Werte von i und j sich der Fehler nach einigen Iterationsschritten kaum ändert. Diesen Anteil nennen wir den glatten oder langwelligen Fehler. Bei geschickter Wahl des Parameters ω kann jedoch für große Werte von i und j der Fehler nach wenigen Iterationsschritten fast eliminiert werden. Diesen Anteil nennen wir den oszillierenden Fehler.
Nun kann der glatte Fehleranteil schon auf einem gröberem Gitter, d.h. mit weniger Stützstellen, hinreichend genau dargestellt werden. Es bietet sich also an, diesen Fehler auf einem gröberen Gitter zu behandeln.
Wir bezeichnen ab jetzt mit Ω^l das feine und mit Ω^{l+1} das grobe Gitter der Größe N_l bzw. N_{l+1}. Die rechte Seite sei mit $f^l \in \mathbb{R}^{N_l}$, der Lösungsvektor mit $u^l \in \mathbb{R}^{N+l}$ und die Systemmatrix mit A^l bezeichnet. Mit $R^l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$ bezeichnen wir die *Restriktion*, mit $P^l : \mathbb{R}^{N_{l+1}} \to \mathbb{R}^{N_l}$ die *Prolongation* oder *Interpolation*.
Wir übertragen das Residuum $r^l := f^l - A^l u^l$ auf das gröbere Gitter:

$$f^{l+1} = Rr^l$$

und setzten dort den Algorithmus rekursiv auf die neue *Defektgleichung*

$$A^{l+1}u^{l+1} = f^{l+1}$$

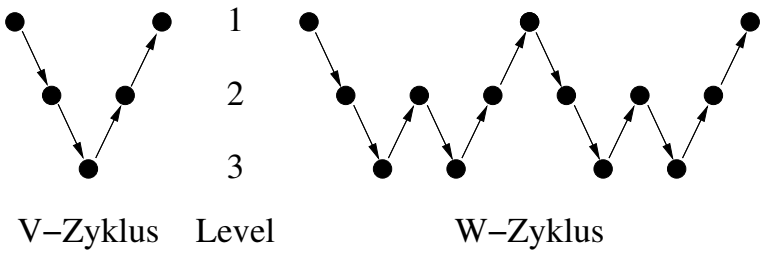


Abbildung 2.1: Durchlaufen der Level während eines V-Zykels (links) bzw. eines W-Zykels (rechts)

an. Das Ergebnis dieser *Grobgitterkorrektur* transferieren wir wieder zurück auf das feine Gitter und korrigieren damit die Feingitter-Lösung

$$u^l \leftarrow u^l + P^l u^{l+1}.$$

Die rekursive Anwendung kann jeweils einmal —*V-Zyklus*— oder zweimal —*W-Zyklus*— erfolgen. Diese Namensgebung wird aus Bild 2.1 deutlich, in dem das Durchlaufen der Level während eines Iterationsschrittes dargestellt worden ist. Der vollständige Algorithmus ist in Programm 2.1 dargestellt. Auf dem gröbsten Level lösen wir entweder mit einem direkten Verfahren (z.B. LU-Zerlegung) oder wir führen auch dort einige Glättungsschritte aus. Dieser Mehrgitterzyklus kann nun entweder alleine oder zur Beschleunigung eines Krylov-Lösungsverfahrens (zum Beispiel das Verfahren der konjugierten Gradienten) eingesetzt werden, wie es in Programm 2.1 beschrieben ist. Dabei bezeichne $KRYLOV(A, f, u)$ die Anwendung eines oder mehrerer Schritte des Krylovverfahrens angesetzt auf die Gleichung $Au = f$. Zur Konvergenz von Mehrgitterverfahren müssen wir die Qualität der Glättung (*Glättungseigenschaft*) und die der Grobgitterkorrektur (*Approximationseigenschaft*) untersuchen.

Definition 2.1 *[Hac85], Definition 6.1.3 (Glättungseigenschaft)*
Ein Iterationsverfahren

$$u^l \leftarrow \left(I^l - C^l A^l\right) u^l$$

erfüllt die Glättungseigeschaft*, wenn es ein* $\eta(\nu)$ *mit* $\lim_{\nu\to\infty} \eta(\nu) = 0$ *gibt, so dass für alle* $\nu \in \mathbb{N}_0$ *die Abschätzung*

$$\|A^l \left(I^l - C^l A^l\right)^\nu \|_2 \leq \eta(\nu)\|A^l\|_2$$

gilt.

Die Grobgitterkorrektur wird bei exaktem Lösen auf dem groben Level durch

$$u^l \leftarrow u^l - P^l \left(A^{l+1}\right)^{-1} R^l \left(A^l u^l - f^l\right)$$

Programm 2.1 Mehrgitter-Algorithmus $MG(A^l, f^l, u^l)$

```
begin
  for ν ← 1 to ν_1 do u^l ← S^l u^l; od;                    pre-smoothing
  r^l ← f^l − A^l u^l;                                      residual
  f^{l+1} ← R^l r^l;                                        restriction
                                                            coarse grid correction
  if l + 1 = l_max
    then
         u^{l+1} ← (A^{l+1})^{-1} f^{l+1};                  apply directly
     else
         for μ ← 1 to μ do
             MG(A^{l+1}, f^{l+1}, u^{l+1}));                solve recursively
         od;
  fi;
  u^l ← u^l + P^l u^{l+1};                                  update solution
  for ν ← 1 to ν_2 do u^l ← S^l u^l;                        post-smoothing
  end
```

Programm 2.2 MG-vorkonditioniertes Krylovverfahren

```
begin
  r ← f − Au;
  while ‖r‖ > ε_tol do
        MG(A, r, z);
        u ← u + z;
        KRYLOV(A, f, u);
        r ← f − Au;
  od;
end
```

beschrieben. Der Fehler e^l ändert sich während dieses Schrittes gemäß

$$e^l \leftarrow \left(I^l - P^l \left(A^{l+1}\right)^{-1} R^l A^l e^l\right).$$

Es soll also —für glatte Fehler— $\left(A^l\right)^{-1}$ möglichst gut durch $P^l \left(A^{l+1}\right)^{-1} R^l$ approximiert werden. Diese Forderung führt zur nachfolgenden Definition.

Definition 2.2 *[Hac85], Definition 6.1.6 (Approximationseigenschaft)*
Die Grobgitterkorrektur erfüllt die Approximationseigenschaft*, wenn für ein c unabhängig von l die Abschätzung:*

$$\| \left(A^l\right)^{-1} - P^l \left(A^{l+1}\right)^{-1} R^l \|_2 \leq \frac{c}{\|A^l\|_2}.$$

gilt. Zusammen ergibt sich die Konvergenz für das Zwei-Level-Verfahrens:

Satz 2.1 *(Konvergenz der Zwei-Level-Methode, [Hac85], Theorem 6.1.7)*
Es sei die Glättungseigenschaft und die Approximationseigenschaft erfüllt. Zu gegebenem $0 < \sigma < 1$ gibt es eine unter Schranke $\underline{\nu}$, so dass für alle $\nu > \underline{\nu}$ gilt:

$$\| \left(I^l - P^l \left(A^{l-1}\right)^{-1} R^l\right) \left(I^l - C^l A^l\right)^{\nu} \|_2 \leq c \cdot \eta(\nu) \leq \sigma$$

Mittels Rekursion lässt sich dieser Satz auch auf den allgemeinen Mehrgitterzyklus verallgemeinern, siehe [Hac85], Abschnitt 7.

2.2 Algebraische Mehrgitterverfahren

Wir wollen die Vorteile von Mehrgitterverfahren auch dann nutzen, wenn die geometrische Struktur des Diskretisierungsgebietes keine groben Gitter vorgibt oder wenn das Gleichungssystem gar nicht der Diskretisierung eines Gebietes entspringt. Wir müssen jetzt die groben „Gitter“ und die entsprechenden Gleichungssysteme nur mit Hilfe des Ausgangsgleichungssystems bestimmen. Die Verfahren, die dieses bewerkstelligen, heißen *algebraische Mehrgitterverfahren.* [Stü99b]
Diese Verfahren bestehen aus zwei Phasen:

1. Die *Setupphase*, in der die gröberen Gitter mitsamt ihren Gleichungssystemen und die zugehörigen Transferoperatoren R und P erzeugt werden

2. Die *Lösungsphase*, in der das Gleichungssystem mit Hilfe der zuvor erstellten Gitterhierarchie gelöst wird

Bemerkung 2.1 Obwohl wir von „Gittern“ sprechen, handelt es sich hierbei strikt genommen nur um Indexmengen. Der Begriff „Gitter“ hat sich aber in der Literatur eingebürgert.

Die Konstruktion der Mehrgitterhierarchie erfordert auf jedem Level $l = 1, \ldots, L_{max}$ die Lösung der folgenden Aufgaben:

- Die Aufteilung des Gitters Ω^l in die Menge der Grobgitterpunkte $C^l = \Omega^{l+1}$ und die Menge der Feingitterpunkte F^l
- Die Konstruktion des Interpolationsoperators $P^l : \mathbb{R}^{N_{l+1}} \to \mathbb{R}^{N_l}$ und des Restriktionsoperators $R^l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$
- Das Aufstellen des Grobgitteroperators $A^{l+1} = R^l A^l P^l$

in Abhängigkeit des Feingitteroperators A^l und eines vorgegebenen Glätters (z.B. gedämpftes Jacobiverfahren oder Gauß-Seidel-Verfahren). Wir besprechen in den nächsten Abschnitten detailliert diese drei Schritte. Zunächst führen wir noch einige Notationen ein.

2.2.1 Notation

Im folgenden seien die Level mit $l = 1, \ldots, L_{max}$ indiziert. Dabei bezeichne 1 das feinste Level, L_{max} das gröbste. Alle zu einem Level gehörenden Unbekannten, Konstanten, Mengen und Operatoren werden mit einem entsprechenden Index gekennzeichnet. Wenn keine Unterscheidung notwendig ist, lassen wir den Level-Index l weg.
Wir schreiben das lineare Gleichungssystem $Au = f$ in Komponenten

$$\sum_{j \in \Omega^l} a_{ij}^l u_j^l = f_i^l \quad \left(i \in \Omega^l\right).$$

Dabei bezeichne Ω^l die *Index*menge der Variablen auf Level l. Diese muss zur Erzeugung des nachstgröberen Gitters in die Menge der *Grobgitterpunkte* C^l und *Feingitterpunkte* F^l aufgeteilt werden

$$\Omega^l = C^l \dot{\cup} F^l.$$

Wir sprechen im Folgenden auch von einem *C/F-Splitting*. Wir übernehmen die Grobgitterpunkte als Punkte für das nächste Level

$$\Omega^{l+1} = C^l$$

und stellen auf dieses Gitter das entsprechende Gleichungssystem auf:

$$\begin{aligned} A^{l+1}u^{l+1} &= f^{l+1} \\ \text{mit } A^{l+1} &= R^l A^l P^l \end{aligned}$$

wobei $R^l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$ die *Restriktion* und $P^l : \mathbb{R}^{N_{l+1}} \to \mathbb{R}^{N_l}$ die *Prolongation* oder *Interpolation* bezeichne.
Wir führen die *Nachbarpunkte* von $i \in \Omega^l$ ein

$$N_i = N_i^l = \{j \in \Omega^l : j \neq i,\ a_{ij}^l \neq 0\}, \tag{2.2}$$
$$N_i^- = \{j \in N_i : a_{ij} < 0\}, \tag{2.3}$$
$$N_i^+ = \{j \in N_i : a_{ij} > 0\}. \tag{2.4}$$

Weiterhin unterscheiden wir zwischen positiven und negativen Werten

$$\begin{aligned} a^+ &= \max(0, a), \\ a^- &= \min(0, a); \end{aligned}$$

entsprechend für Matrixeinträge a_{ij}.
Zwei wichtige Qualitätskriterien für die konstruierte Mehrgitterhierarchie sind die *Operatorkomplexität*

$$c_A := \frac{\sum_{l=1}^{L_{max}} nonzeros\left(A^l\right)}{nonzeros\left(A^1\right)} \tag{2.5}$$

und die *Gitterkomplexität*

$$c_G := \frac{\sum_{l=1}^{L_{max}} |\Omega^l|}{|\Omega^1|} \tag{2.6}$$

Beide Größen liefern ein Maßdafür, um welchen Faktor der Speicherbedarf gegenüber dem des zu lösenden Gleichungssystems anwächst.
Um die Glättung beschreiben zu können, führen wir drei innere Produkte ein.

Definition 2.3

$$(u, v)_0 := (Du, v) \tag{2.7}$$
$$(u, v)_1 := (Au, v) \tag{2.8}$$
$$(u, v)_2 := (D^{-1}Au, Av) \tag{2.9}$$

Die zugehörigen Normen seien mit $\|u\|_0$, $\|u\|_1$ und $\|u\|_2$ bezeichnet.

Im folgenden Lemma geben wir einige Beziehungen zwischen diesen Normen an.

Lemma 2.1 *([Stü99b], Lemma 3.1) Sei A symmetrisch und positiv definit. D bezeichne die Diagonale von A. Dann gilt für alle e:*

$$\|e\|_1^2 \leq \|e\|_0\|e\|_2, \quad \|e\|_2^2 \leq \rho(D^{-1}A)\|e\|_1^2, \quad \|e\|_1^2 \leq \rho(D^{-1}A)\|e\|_0^2$$

Für die Eigenvektoren ϕ_i von $D^{-1}A$ und ihre zugehörigen Eigenwerte λ_i gilt

$$\|\phi_i\|_2^2 = \lambda_i\|\phi_i\|_1^2, \text{ und } \|\phi_i\|_1^2 = \lambda_i\|\phi_i\|_0^2. \tag{2.10}$$

Mit diesen Bezeichnungen können wir jetzt die Mehrgitter-Hierarchie konstruieren.

2.3 Glättung

Zunächst untersuchen wir die Auswirkung des Glättungsoperators $S = S^l$ auf den Fehler $e = e^{it} = u - u^{it}$, wobei u^{it} eine iterierte Approximation an der exakten Lösung u bezeichne.

Definition 2.4 *Ein Fehler e heisst* algebraisch glatt*, falls er durch Anwendung des Glättungsoperators S langsam abfällt, d.h. $Se \approx e$.*

Wir entwickeln den Fehler in den Eigenvektoren ϕ_i von S. Die zugehörigen Eigenwerte erfüllen $\lambda_i^* \approx 1$. Im Falle der Jacobi-Relaxation $S = I - \omega D^{-1}A$ entsprechen diese Eigenwerte wiederum den kleinen Eigenwerten λ_i von $D^{-1}A$. Für die Gauß-Seidel-Relaxation gilt ein ähnliches Resultat, siehe [Stü99b], Seite 27.
Bemerkung: Ein algebraisch glatter Fehler ist nicht zwangsläufig geometrisch „glatt“ und umgekehrt, siehe [Stü99b], Example 3.2.

Definition 2.5 *([Stü99b], Abschnitt 3.2) Ein Operator S erfüllt die* Glättungseigenschaft *bezüglich einer Matrix A, wenn*

$$\|Se\|_1^2 \leq \|e\|_1^2 - \sigma\|e\|_2^2 \tag{2.11}$$

für ein $\sigma > 0$ unabhängig von e gilt.

Ein solcher Glätter reduziert diejenigen Fehleranteile e, für die $\|e\|_2 \ll \|e\|_1$ gilt, nur sehr langsam. Diese Anteile gehören zu kleinen Eigenwerten von $D^{-1}A$, siehe (2.10) und sind damit nach obigen Überlegungen algebraisch glatt.

Satz 2.2 *([Stü99b], Theorem 3.1)*
Sei A symmetrisch, positiv definit und w ein beliebiger Vektor mit $w_i > 0$ für alle i. Wir definieren die Größen

$$\gamma_- := \max_i \{\frac{1}{w_i a_{ii}} \sum_{j<i} w_j |a_{ij}|\} \text{ und } \gamma_+ := \max_i \{\frac{1}{w_i a_{ii}} \sum_{j>i} w_j |a_{ij}|\}.$$

Dann erfüllt das Gauß-Seidel-Verfahren die Glättungseigenschaft (2.11) mit $\sigma = \frac{1}{(1+\gamma_-)(1+\gamma_+)}$.

Satz 2.3 *([Stü99b], Theorem 3.2)*
Sei A symmetrisch positiv definit und $\eta \geq \rho(D^{-1}A)$. Dann erfüllt Jacobi-Relaxation mit Relaxationsparameter $0 < \omega < \frac{2}{\eta}$ die Glättungseigenschaft (2.11) mit $\sigma = \omega(2 - \omega\eta)$. Der optimale Relaxationsparameter ist durch $\omega^* = \frac{1}{\eta}$ gegeben, in diesem Fall ist $\sigma = \frac{1}{\eta}$.

Bemerkung 2.2 Für die meisten Anwendungen (Diskretisierung partieller Differenzialgleichungen) ist $\sum_{j \neq i} |a_{ij}| \approx a_{ii}$. In diesen Fällen beträgt der Parameter $\sigma \approx \frac{1}{4}$ für das Gauß-Seidel-Verfahren und $\sigma \approx \frac{1}{2}$ für das Jacobiverfahren mit Dämpfungsparameter $\omega = \frac{1}{2}$.

Wir betrachten nun die algebraisch glatten Fehleranteile, d.h. diejenigen e mit $\|e\|_2 \ll \|e\|_1$. Für das Residuum $r = Ae$ gilt damit

$$(D^{-1}r, r) \ll (e, r).$$

Das skalierte Residuum $D^{-1}r$ ist also wesentlich kleiner als der eigentliche Fehler e.
Für die Gauß-Seidel-Relaxation und das Jacobiverfahren können wir dies auch direkt herleiten. Ein Gauß-Seidel-Relaxationsschritt $u^{it} \mapsto u^{it+1}$ für die i-te Komponente ist durch:

$$u_i^{it+1} = \frac{1}{a_{ii}} \left(f_i - \sum_{j<i} a_{ij} u_j^{it+1} - \sum_{j>i} a_{ij} u_j^{it} \right) = u_i^{it} + \frac{r_i^{it}}{a_{ii}}$$

gegeben. Dabei bezeichnet r_i^{it} die i-te Komponente des Residuums, nach Relaxation von u_{i-1}^{it} und vor Relaxation von u_i^{it}. (Im Falle des Jacobiverfahrens ist r_i^{it} gleich dem Residuum vor Anfang des Iterationsschrittes.) Für den Fehler $e_i^{it+1} = u_i - u_i^{it+1}$ heisst dies:

$$e_i^{it+1} = e^{it} - \frac{r_i}{a_{ii}}$$

Ist nun der Fehler algebraisch glatt, dass heisst $e^{it+1} \approx e^{it}$, so sehen wir, dass

$$|r_i| \ll a_{ii}|e_i|$$

gilt. Wir können damit den Fehler in der Komponente i durch

$$(r_i =)\, a_{ii}e_i + \sum_{j \neq i} a_{ij}e_j \approx 0$$

approximieren. Diese Gleichung liefert die Basis für die Interpolation eines algebraisch glatten Fehlers. Zunächst wollen wir aber den Charakter eines glatten Fehlers genauer untersuchen.

2.3.1 M-Matrizen

Definition 2.6 *(M-Matrix) Eine Matrix $A \in R^{n\times n}$ heisst* M-Matrix*, wenn die folgenden Bedingungen erfüllt sind:*

- $a_{ii} > 0$ *für alle* $i = 1, \ldots, n$,
- $a_{ij} \leq 0$ *für alle* $i, j = 1, \ldots n,\ i \neq j$,
- *A ist regulär,*
- *die Komponenten von A^{-1} sind nichtnegativ.*

Für diese Matrizen können wir den algebraisch glatten Fehler wie folgt charakterisieren:
Aus $\|e\|_2 \ll \|e\|_1$ (2.11) folgt wegen $\|e\|_1^2 \leq \|e\|_0\ \|e\|_2$

$$\|e\|_1 \ll \|e\|_0 \text{ für } e \neq 0.$$

Wir schreiben $\|e\|_1$ aus und erhalten:

$$\begin{aligned} \|e\|_1^2 &= \sum_{i,j} a_{ij}e_ie_j = \frac{1}{2}\sum_{i,j} -a_{ij}(e_i - e_j)^2 + \sum_i s_i e_i^2, \\ \text{wobei } s_i &:= \sum_j a_{ij} \end{aligned}$$

und damit

$$\sum_{j\neq i} -\frac{a_{ij}(e_i - e_j)^2}{a_{ii}e_i^2} \ \ll\ 1$$

Dies bedeutet, dass ein algebraisch glatter Fehler entlang der Kopplungen a_{ij} mit $|a_{ij}| \gg 0$ nur wenig variiert.

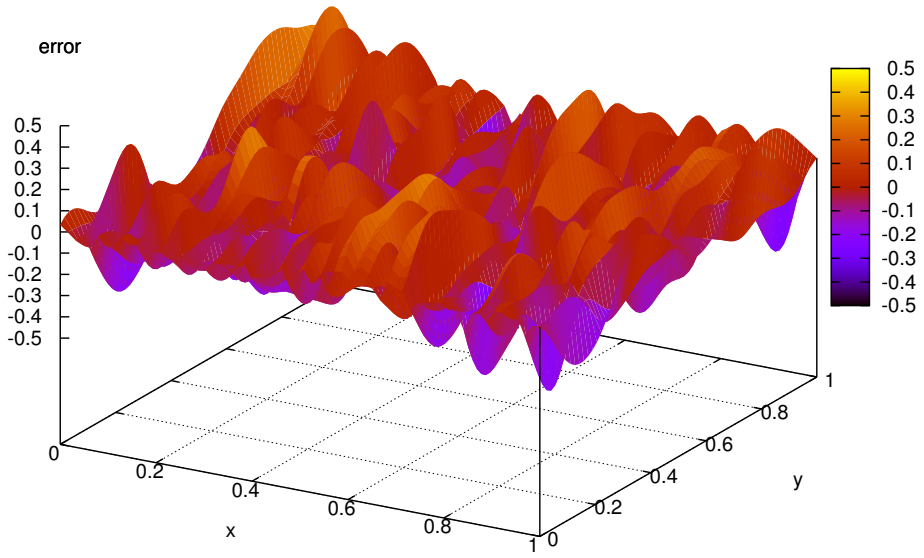


Abbildung 2.2: Fehler nach 10 Gauß-Seidel-Schritten für den Operator $-u_{xx} - 0.001 \cdot u_{yy}$ und zufälligen, auf 1 normierten Startvektor.

Beispiel 2.1 Wir betrachten das anisotrope Problem

$$\begin{aligned} -\epsilon \mathrm{u}_{xx} - \mathrm{u}_{yy} &= \mathrm{f} \text{ in } \Omega = (0,1)^2 \\ \mathrm{u} &= \mathrm{g} \text{ auf } \partial\Omega \end{aligned}$$

für ein $\epsilon \ll 1$. Dann variiert der Fehler in y-Richtung nach Glättung nur geringfügig, während er in x-Richtung stark oszilliert, siehe Abbildung 2.2

2.3.2 Wesentlich positive Matrizen

Wir untersuchen jetzt eine zweite wichtige Klasse von Matrizen, die vor allem bei der Diskretisierung von partiellen Differenzialgleichungen mit gemischten Ableitungen auftreten.

Definition 2.7 *(wesentlich positive Matrix) ([Stü99b], Anschnitt 3.3.2) Eine Matrix $A \in \mathbb{R}^{n\times n}$ heisst* wesentlich positiv*, wenn es eine Konstante $c > 0$ gibt, so dass für alle $e \in \mathbb{R}^n$*

$$\sum_{i,j} -a_{ij}(e_i - e_j)^2 \geq c \sum_{i,j} -a_{ij}^-(e_i - e_j)^2$$

gilt.

In diesem Fall enthält jede Zeile A_i der Matrix mindestens ein negatives Element a_{ij}.
Analog zum Fall der M-Matrizen erhalten wir folgende Abschätzung

$$\frac{c}{2}\sum_{i,j} -a_{ij}^{-}(e_i - e_j)^2 + \sum_i s_i e_i^2 \ll \sum_i e_i^2$$

und sehen wiederum, dass der glatte Fehleranteil entlang der negativen großen Kopplungen $a_{ij}^{-} \ll 0$ nur geringfügig variiert.

Beispiel 2.2 Wir betrachten die 9-Punkte-Diskretisierung des Operators $-\Delta u + u_{xy}$ auf einem uniformen Gitter der Maschenweite h. Diese ist durch den Stern

$$\frac{1}{h^2}\begin{bmatrix} -0.25 & -1 & +0.25 \\ -1 & 4 & -1 \\ +0.25 & -1 & -0.25 \end{bmatrix}_h$$

gegeben. Die resultierende Matrix erfüllt die obige Bedingung mit $c = 0.5$. Wir sehen in Abbildung 2.3, dass die gegenüber der 5-Punkte-Diskretisierung von Δu hinzugefügten *kleinen* positiven Kopplungen keinen Einfluss auf den glatten Fehler haben.

2.3.3 Große positive Kopplungen

Wir untersuchen nun das Oszillationsverhalten eines algebraisch glatten Fehlers entlang großer positiver Kopplungen, insbesondere für Matrizen A, die fast schwach diagonaldominant sind, d.h. $a_{ii} - \sum_{j\neq i} |a_{ij}| \approx 0$. Wir erhalten

$$\sum_{j\neq i} -\frac{a_{ij}^{-}(e_i - e_j)^2}{a_{ii}e_i^2} + \sum_{j\neq i} \frac{a_{ij}^{+}(e_i + e_j)^2}{a_{ii}e_i^2} \ll 1.$$

Für positive a_{ij}^{+}, die relativ groß gegenüber a_{ii} sind, ergibt sich $e_i \approx -e_j$. In diesem Fall oszilliert der Fehler stark entlang der großen positiven Kopplungen.

Beispiel 2.3 Wir betrachten die Diskretisierung der Poissongleichung mit

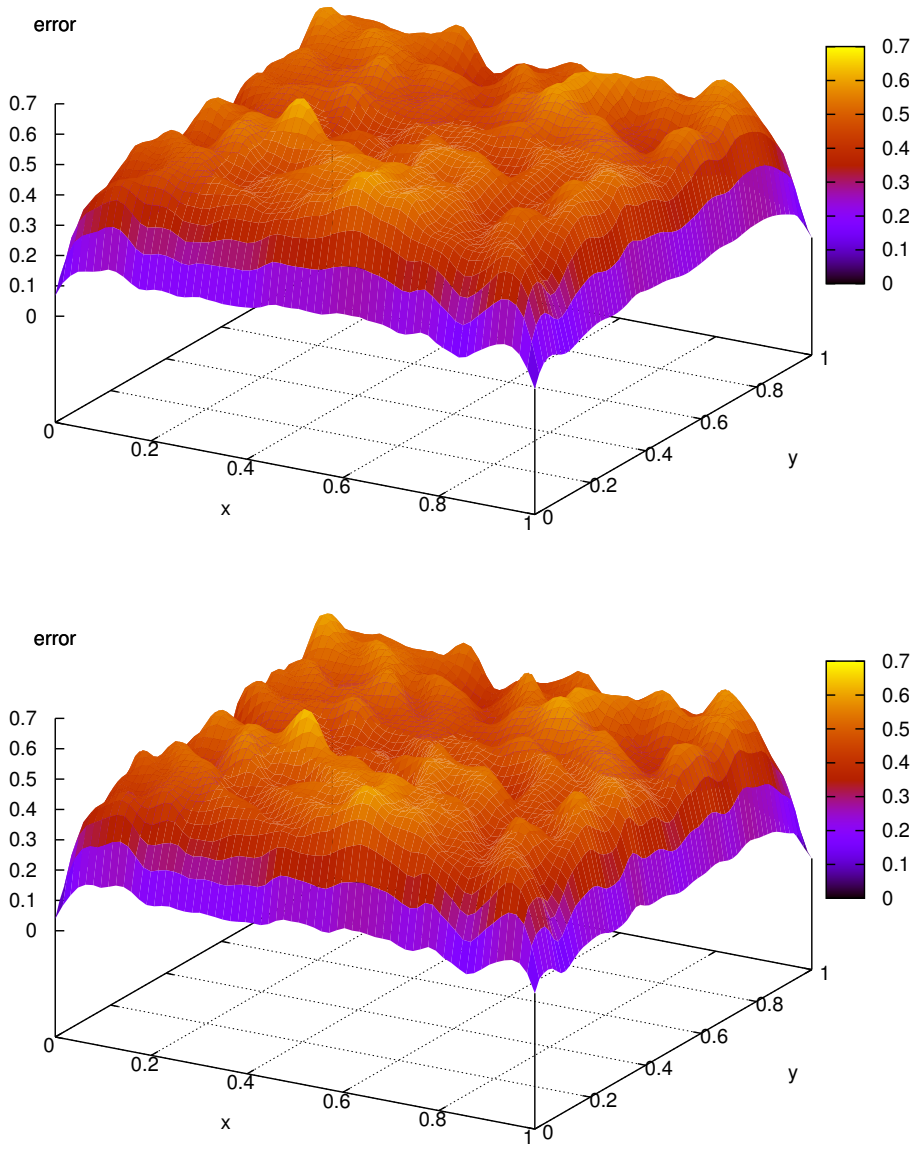


Abbildung 2.3: Fehler nach 5 Gauß-Seidel-Schritten für die Diskretisierung des Laplaceoperators mittels eines 5-Punkte-Stern(oben) und die Diskretisierung von $-\Delta u + u_{xy}$ (unten) jeweils mit zufällig gewähltem, auf 1 normierten Startvektor

antiperiodischen Randbedingungen, deren Systemmatrix durch

$$A = \begin{pmatrix} B & -I & 0 & \cdots & \cdots & 0 & I \\ -I & B & -I & 0 & \cdots & \cdots & 0 \\ 0 & -I & B & -I & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & \cdots & 0 & -I & B & -I \\ I & 0 & \cdots & \cdots & 0 & -I & B \end{pmatrix}$$

$$\text{mit } B = \begin{pmatrix} 4 & -1 & 0 & \cdots & \cdots & 0 & 1 \\ -1 & 4 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & 4 & -1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & -1 & 4 & -1 & 0 \\ 0 & \cdots & \cdots & 0 & -1 & 4 & -1 \\ 1 & 0 & \cdots & \cdots & 0 & -1 & 4 \end{pmatrix}$$

gegeben ist. In diesem Fall treten für die Kopplungen über den Rand hinweg einige positive Nicht-Diagonaleinträge auf. Der Effekt der positiven Matrixeinträge ist in Bild 2.4 deutlich zu sehen. Der algebraisch glatte Fehler enthält einen Sprung über den Gebietsrand hinweg.

2.4 Starke Kopplungen

Wie wir in Abschnitt 2.3.1 gesehen haben, variiert ein algebraisch glatter Fehler zwischen zwei Punkten i und j, für die a_{ij} relativ großist, nur geringfügig. Dies motiviert den Begriff der *starken Kopplung*.

Definition 2.8 *(starke Kopplung, [Stü99b], Abschnitt 7.1*
Sei $0 < \alpha < 1$ *fest gewählt.*

1. *Ein Punkt* j *ist mit einem Punkt* $i \neq j$ stark gekoppelt, *wenn*

$$|a_{ij}| \geq \alpha \cdot \max_{k \neq i} |a_{ik}|. \tag{2.12}$$

   gilt.

2. *Weiterhin bezeichne*

$$S_i := \{j \in \Omega : \ |a_{ij}| \geq \alpha \cdot \max_{k \neq i} |a_{ik}|\} \text{ die starken Kopplungen von } i,$$
$$S_i^T := \{j \in \Omega : i \in S_j\} \text{ die Menge aller mit } i \text{ stark gekoppelten Punkte.}$$

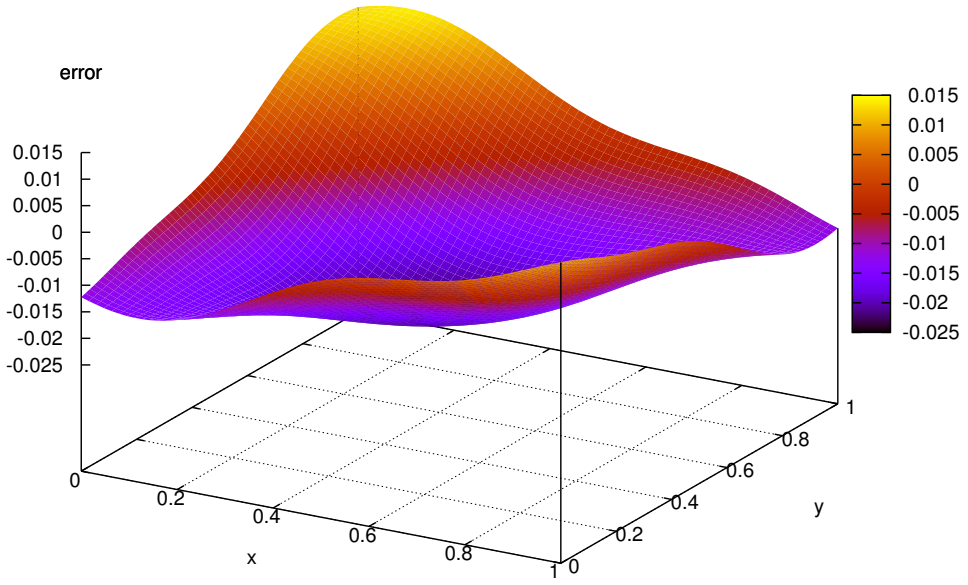


Abbildung 2.4: algebraisch glatter Fehler für den Laplaceoperator mit antiperiodischen Randbedingungen, zufällig gewählter, normierter Startvektor.

Anschaulich beschreibt S_i die *Abhängigkeiten* von i, d.h. der algebraisch glatte Fehler e_i am Punkt i hängt insbesondere vom algebraisch glatten Fehler e_j an den Punkten $j \in S_i$ ab. Entsprechend nennen wir S_i^T auch den *Einfluss* von i.
Im AMG-Algorithmus benutzen wir üblicherweise einen Wert von $\alpha = 0.25$. Der Algorithmus zur Bestimmung der starken Kopplungen ist in Programm 2.3 dargestellt. Der Rechenaufwand für diesen Algorithmus beträgt $O\left(\sum_{i=1}^{N} N_i\right)$,

Programm 2.3 *AmgStrongCouplings*(A, S, S^T)

```
begin
  for i ← 1 to N do
      for j ∈ N_i do
          if |a_ij| ≥ α · max_{k∈N_i} |a_ik|
            then
                    S_i ← S_i ∪ {j};
                    S_j^T ← S_j^T ∪ {i};
          fi;
      od;
  od;
end
```

also $O(N)$, wenn A dünnbesetzt ist. Der Speicheraufwand ist ebenfalls durch diesen Wert begrenzt.

2.5 Variationsprinzip

Bevor wir uns der Konstruktion der Interpolation und der Grobgitterwahl zuwenden, formulieren wir zunächst das Variationsprinzip für einen Zwei-Level-Zyklus. Es wird sich zeigen, dass in diesem Fall die Grobgitterkorrektur der Lösung einer Variationsaufgabe entspricht. Im folgenden sei der Gitteroperator A^l ein symmetrisch, positiv definiter Operator. Wir gehen von einem gegebenen C/F-Splitting aus und sortieren die Einträge des Gitteroperators und der Prolongation um, so dass die zu C-Variablen gehörenden Zeilen und Spalten zuerst kommen. Wir erhalten die Blockmatrizen

$$A^l = \begin{pmatrix} A_{CC} & A_{CF} \\ A_{FC} & A_{FF} \end{pmatrix}, \quad P^l = \begin{pmatrix} P_{CC} \\ P_{FC} \end{pmatrix}.$$

Wir setzen nun $P_{CC} = I$, d.h. die Werte an Grobgittervariablen werden einfach vom nächstgröberem Level übernommen. Der Grobgitteroperator A^{l+1}

sei durch

$$A^{l+1} = \left(P^l\right)^T A^l P^l$$

definiert (sog. *Galerkinansatz*). Damit wird ein Grobgitter-Korrekturschritt gegeben durch den *Korrekturoperator*:

$$K^l = I - P^l \left(A^{l+1}\right)^{-1} R^l A^l.$$

Der Fehler e^l wird jetzt durch Vorglättung, Grobgitterkorrektur und Nachglättung gemäß

$$e \mapsto \left(S^l\right)^{\nu_2} K^l \left(S^l\right)^{\nu_1} e$$

verändert. Wir untersuchen nun den Korrekturoperator näher. Es zeigt sich, dass dieser ein orthogonaler Projektor bezüglich des 1-Produktes ist.

Satz 2.4 *([Stü99b], Corollary 2.1)*
Sei ein beliebiges C/F-Splitting $\Omega = C \dot\cup F$ gegeben. Sei A^l symmetrisch positiv definit und P^l ein beliebiger Interpolationsoperator mit vollem Rang. Dann ist der durch den Galerkin-Ansatz gegebene Korrekturoperator ein Orthogonalprojektor bezüglich $(\cdot,\cdot)_1$, d.h.

$$\begin{aligned} \mathcal{R}(K^l) &\perp_1 \mathcal{R}(P^l), \\ \text{für } u^l \in \mathcal{R}(K^l),\ v^l \in \mathcal{R}(P^l) \text{ gilt } \|u^l + v^l\|_1^2 &= \|u^l\|_1^2 + \|v^l\|_1^2, \\ \|K^l\|_1 &= 1, \\ \text{für alle } e^l \text{ gilt } \|K^l e^l\|_1 &= \min_{\tilde e^{l+1}} \|e^l - P^l \tilde e^{l+1}\|_1. \end{aligned}$$

Die Grobgitterkorrektur minimiert also den Fehler über das Bild von P^l in der Energienorm des Operators. Ebenfalls garantiert dieser Satz die Konvergenz der Zweigittermethode, solange die Energienorm des Glätters $\|S^l\| \leq 1$ erfüllt (dies gilt zum Beispiel für das Gauß-Seidel-Verfahren und das gedämpfte Jacobiverfahren). Um dieses Resultat auf den gesamten V-Zyklus auszudehnen, benötigen wir noch folgendes Lemma.

Lemma 2.2 *([Stü99b], Lemma 2.2)*
Wir ersetzen die exakte Grobgitterkorrektur e^{l+1} durch eine Approximation $\tilde e^{l+1}$ mit $\|e^{l+1} - \tilde e^{l+1}\|_1 \leq \|e^{l+1}\|_1$, wobei $\|\cdot\|_1$ die Energienorm des Grobgitteroperators A^{l+1} bezeichne. Dann erfüllt der approximierende Zwei-Level-Korrekturoperator $\tilde K^l$ noch $\|\tilde K^l\|_1 \leq 1$.

Hiermit ist die Konvergenz des Mehrgitterverfahrens für jede Interpolation mit vollem Rang gesichert. Gleichzeitig haben wir auch eine Rechtfertigung für den Galerkinansatz $R^l A^l P^l$ als Grobgitteroperator.

2.6 Interpolation

Wie wir im vorigen Abschnitt gesehen haben, führt jeder Interpolationsoperator mit vollem Rang zu einem konvergenten Verfahren. Dies sagt jedoch nichts über die Konvergenzgeschwindigkeit und die Kosten des Verfahrens aus, so dass die Konstruktion des Interpolationsoperators trotzdem keine triviale Aufgabe ist.
Wir werden zunächst eine Interpolation beschreiben, die eine Zweigittermethode in einen direkten Löser verwandelt. Diese Interpolation ist jedoch in der Praxis nicht brauchbar, so dass wir sie nur approximieren können.
Wir definieren zunächst einen „Glättungsprozess“, welcher nur die Feingittervariablen relaxiert,

$$u \mapsto \overline{u}, \quad , A_{FF}\overline{u}_F + A_{FC}u_C = f_F, \quad \overline{u}_C = u_C.$$

Der Fehler nach diesem Glättungsschritt beträgt dementsprechend

$$e \mapsto \overline{e}, \quad , A_{FF}\overline{e}_F + A_{FC}e_C = 0, \quad \overline{e}_C = e_C.$$

Der Glättungsoperator lautet damit

$$S^l = \begin{pmatrix} I & 0 \\ -A_{FF}^{-1}A_{FC} & 0 \end{pmatrix}. \tag{2.13}$$

Wir geben nun zwei mögliche Transferoperatoren an:

$$\begin{aligned} \overline{P}_{FC} &= -A_{FF}^{-1}A_{FC} \\ \overline{R}_{CF} &= -A_{CF}A_{FF}^{-1} \end{aligned}$$

Wie vorher ist stets $P_{CC} = R_{CC} = I$.

Satz 2.5 *([Stü99b], Theorem 2.1)*
Sei A^l nichtsingulär und sei ein C/F-Splitting gegeben, so dass A_{FF} nichtsingulär ist. Sei die Glättung gegeben durch (2.13)*. Dann gilt*

1. *Für $P_{FC} = \overline{P}_{FC}$ und beliebigem R_{CF} ist A^{l+1} nichtsingulär und $K^l S^l = 0$.*
2. *Für $R_{CF} = \overline{P}_{CF}$ und beliebigem R_{FC} ist A^{l+1} nichtsingulär und $S^l K^l = 0$.*
3. *In beiden Fällen ist der Galerkinoperator gleich dem Schurkomplement*

$$A^{l+1} = A_{CC} - A_{CF}A_{FF}^{-1}A_{FC}.$$

Wir können also —je nach Wahl des Interpolationsoperators— mit nur einem Vor- oder Nachglättungsschritt ein direktes Verfahren konstruieren. Diese Überlegungen können wir auch für einen kompletten V-Zyklus durchführen, welcher ebenfalls in einem Iterationsschritt konvergieren würde. In der Praxis ist dieses Verfahren aber nicht geeignet, da die Invertierung von A_{FF} benötigt wird.
Wir betrachten nochmals die Interpolation $\overline{P}_{FC} = -A_{FF}^{-1}A_{FC}$. Damit folgt für den Fehler e_F

$$A_{FF}e_F + A_{CF}e_C = 0. \tag{2.14}$$

Zur Konstruktion der Interpolation werden wir diese Gleichung approximieren. Weiterhin beschränken wir uns jetzt wieder auf diejenigen Glätter, für die die Glättungseigenschaft (2.11)

$$\|Se\|_1^2 \leq \|e\|_1^2 - \sigma\|e\|_2^2$$

erfüllt ist. Wenn nun außerdem die Interpolation so gegeben ist, dass der Korrekturoperator K die Ungleichung

$$\|Ke\|_1^2 \leq \tau\|Ke\|_2^2 \tag{2.15}$$

für ein $\tau > 0$ unabhängig von e erfüllt, so gilt für den Fehler nach Grobgitterkorrektur und Nachglättung:

$$\|Ke\|_1^2 \leq \left(1 - \frac{\sigma}{\tau}\right)\|e\|_1^2. \tag{2.16}$$

Eine hinreichende Bedingung für (2.15) wird im folgenden Satz charakterisiert.

Satz 2.6 *([Stü99b], Theorem 4.2)*
Sei ein C/F-Splitting und eine Interpolation P_{FC} dergestalt, dass für alle e und ein $\tau > 0$ unabhängig von e

$$\|e_F - P_{FC}e_C\|_{0,F}^2 \leq \tau\|e\|_1^2 \tag{2.17}$$

gilt. Dann ist 2.15 erfüllt.

Wir fordern hiermit, dass der Fehler nach einem Grobgitter-Korrekturschritt durch Anwendung des Glätters weiter reduziert werden kann. Damit stellt dies eine Verallgemeinerung des oben angegebenen Glättungs- und Interpolationsverfahren dar, welches den Fehler vollständig eliminiert.
Wir bemerken noch, dass Bedingung (2.17) insbesondere für algebraisch glatte Fehler eine möglichst exakte Interpolation fordert. Dazu schreiben wir den

Fehler in den Eigenvektoren von A und erhalten für jeden Eigenvektor ϕ und den zugehörigen Eigenwerten λ

$$\|\phi_F - P_{FC}\phi_C\|_{0,F}^2 \leq \tau\lambda\|\phi\|_0^2,$$

d.h. für die kleinen Eigenwerte von A, die den glatten Fehleranteilen zugeordnet sind, wird eine starke Reduktion gefordert.
Mit diesen Überlegungen können wir nun Interpolationsoperatoren konstruieren, die mit einem Rechenaufwand von $O(N)$ aufgestellt und angewendet werden können.

2.6.1 Direkte Interpolation

Wir beschränken uns zunächst auf M-Matrizen. Wir haben in Abschnitt 2.3.1 gesehen, dass für diese Klasse von Matrizen der glatte Fehler entlang der (negativen) starken Kopplungen nur geringfügig variiert. Es liegt deshalb nahe, auch entlang dieser starken Kopplungen zu interpolieren. Wir approximieren deshalb Gleichung (2.14) mittels

$$e_i = -\frac{1}{a_{ii}} \cdot \frac{\sum_{j\in N_i} a_{ij}}{\sum_{j\in S_i\cap C} a_{ij}} \sum_{j\in S_i\cap C} a_{ij}e_j. \tag{2.18}$$

Die Gewichtung mittels

$$\frac{\sum_{j\in N_i} a_{ij}}{\sum_{j\in S_i\cap C} a_{ij}}$$

gewährleistet dabei, dass die Zeilensumme erhalten bleibt. Damit werden im Fall von Matrizen mit verschwindender Zeilensumme konstante Fehlervektoren exakt interpoliert, welche Forderung sich zwingend aus Gleichung (2.17) ergibt.

Satz 2.7 *([Stü99b], Theorem 4.3)*
Sei A eine symmetrische M-Matrix mit $\sum_j a_{ij} \geq 0$. Sei $\tau \geq 1$ fest und sei ein C/F-Splitting gegeben, so dass es für jeden Feingitterpunkt i

$$\sum_{j\in S_i\cap C} |a_{ij}| \geq \frac{1}{\tau} \sum_{j\in N_i} |a_{ij}|$$

gilt. Dann erfüllt die direkte Interpolation 2.18 die Ungleichung 2.17.

Dieses Ergebnis kann auch auf symmetrische M-Matrizen mit negativer Zeilensumme erweitert werden, wenn die Eigenwerte gleichmäßig von 0 weg beschränkt sind, siehe [Stü99b], Theorem 4.4.

Wenn die Matrizen positive Einträge aufweisen, müssen wir die Interpolation nach Vorzeichen aufsplitten. Wir interpolieren dann gemäß

$$e_i = -\frac{1}{a_{ii}} \cdot \left(\frac{\sum_{j \in N_i} a_{ij}^-}{\sum_{j \in S_i \cap C} a_{ij}^-} \sum_{j \in S_i \cap C} a_{ij}^- e_j + \frac{\sum_{j \in N_i} a_{ij}^+}{\sum_{j \in S_i \cap C} a_{ij}^+} \sum_{j \in S_i \cap C} a_{ij}^+ e_j \right) \tag{2.19}$$

Sind in einer Zeile nur positive Nichtdiagonaleinträge a_{ij}^+ oder nur negative Nichtdiagonaleinträge a_{ij}^- vorhanden, so entfällt der entsprechend andere Summand. Für diese Interpolation gilt ein ähnliches Resultat wie oben, siehe [Stü99b], Theorem 4.6.
Der Algorithmus zur Konstruktion des Interpolationsoperators ist in den Programmen 2.4 und 2.5 angegeben. Dabei haben wir zu Gunsten einer einfacheren Darstellung darauf verzichtet, die Grobgitterpunkte im groben Level anders zu indizieren. Der Rechenaufwand für diesen Algorithmus beträgt

Programm 2.4 direkte Interpolation *AmgInterpolationDirect*(A, S, C, F, P)

```
begin
  for i ← 1 to N do
    if i ∈ C
      then
          p_ii ← 1;
       else
          AmgInterpolationDirectRow(A_i, S_i, C, F, P_i);
    fi;
  od;
end
```

wiederum

$$O(\left(\sum_{i=1}^{N} N_i \right) = O\left(N\right).$$

Dieser Interpolationsoperator ist jedoch nicht stabil, weil starke Kopplungen zu anderen Feingitterpunkten $j \in S_i \cap F$ nicht berücksichtigt werden. Wir sehen aber aus Satz 2.7, dass je weniger der stark gekoppelten Nachbarn $j \in S_i$ in die Menge C aufgenommen worden sind, um so größer der Parameter τ sein muss, um die Bedingung (2.17) zu erfüllen. Größere Werte für τ führen jedoch nach (2.16) zu einer langsameren Konvergenz.

Programm 2.5 *AmgInterpolationDirectRow*(A_i, S_i, C, F, P_i)

```
begin
  for j ← 1 to N do
    if j ∈ S_i ∩ C
      then
        if a_ij < 0
          then
            p_ij ← (1/a_ii) (Σ_{k∈N_i} a⁻_ik) / (Σ_{k∈S_i∩C} a⁻_ik) · a⁻_ij;
        elsif a_ij > 0
          then
            p_ij ← (1/a_ii) (Σ_{k∈N_i} a⁺_ik) / (Σ_{k∈S_i∩C} a⁺_ik) · a⁺_ij;
        fi;
    fi;
  od;
end
```

2.6.2 Standard-Interpolation

Die Standardinterpolation behebt —wenn auch verbunden mit höheren Kosten— den soeben beschriebenen Nachteil der direkten Interpolation. Wir berücksichtigen jetzt *alle* starken Kopplungen eines F-Punktes i, auch diejenigen, die zu anderen F-Punkten j führen. Die Summe dieser Beträge dieser Kopplungen macht dann einen Großteil der Gesamtzeilensumme aus.
Sei $i \in F$ ein Feingitterpunkt und $j \in S_i \cap F$. Wir eliminieren zuerst diese Kopplung

$$a_{ij}e_j \mapsto -\frac{a_{ij}}{a_{jj}} \sum_{k \in N_j} a_{ik}e_k$$

und erhalten eine modifizierte Zeile

$$\hat{A}_i = (\hat{a}_{i1}, \cdots, \hat{a}_{iN}) \tag{2.20}$$

Auf diese wenden wir nun das direkte Interpolationsverfahren, aber mit einer erweiterten Menge von Interpolationspunkten

$$\mathcal{P}_i := \left(S_i \cup \bigcup_{j \in S_i \cap F} S_j \right) \cap C, \tag{2.21}$$

an. Wir erhalten

$$e_i = -\frac{1}{\hat{a}_{ii}} \cdot \left(\frac{\sum_{j \in N_i} \hat{a}^-_{ij}}{\sum_{j \in \mathcal{P}_i} \hat{a}^-_{ij}} \sum_{j \in \mathcal{P}_i} \hat{a}^-_{ij} e_j + \frac{\sum_{j \in N_i} \hat{a}^+_{ij}}{\sum_{j \in \mathcal{P}_i} \hat{a}^+_{ij}} \sum_{j \in \mathcal{P}_i} \hat{a}^+_{ij} e_j \right), \tag{2.22}$$

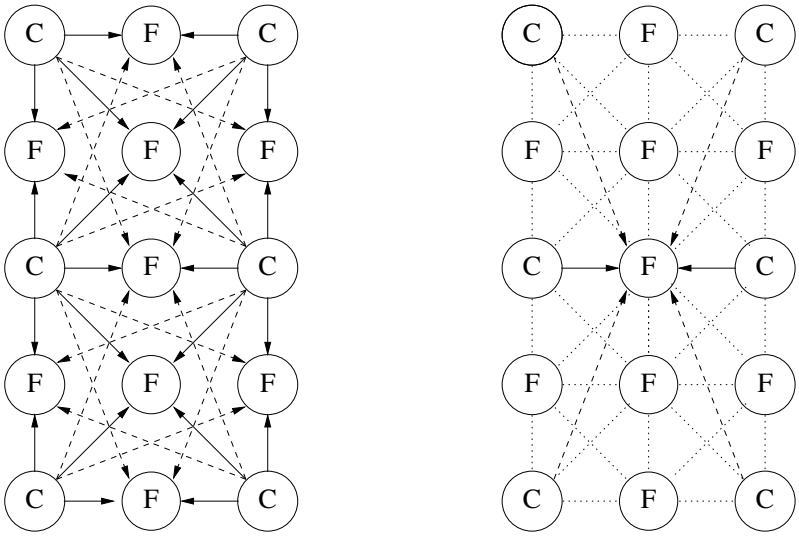


Abbildung 2.5: Direkte und Standardinterpolation für einen 9-Punkte-Stern. Die ununterbrochenen Pfeile markieren die direkte Interpolation, bei der Standardinterpolation kommen die unterbrochenen Verbindungen hinzu. Links ist die Interpolation für alle Feingitterpunkte dargestellt, rechts ist zur besseren Erkennbarkeit nur ein Punkte hervorgehoben. Hier sind zusätzlich die starken Kopplungen als gepunktete Linien dargestellt

wobei gegebenenfalls wieder ein Summand wegfällt.
Die Standardinterpolation führt zu besseren Konvergenzraten als die direkte Interpolation, weil jetzt alle starken Verbindungen berücksichtigt werden. Wir betrachten als Beispiel einen 9-Punkte-Stern in zwei Raumdimensionen, siehe Abbildung 2.5. Wir sehen, dass der mittlere Punkt bei der direkten Interpolation nur von seinem linken und rechtem Nachbar interpoliert. Bei der Standardinterpolation kommen vier weitere Punkte hinzu, insbesondere werden jetzt auch die starken Kopplungen nach oben und unten berücksichtigt. (Die Interpolation berücksichtigt also nicht nur eine Richtung). Die Standardinterpolation hat jedoch einen gravierenden Nachteil: Es werden Kopplungen zu Punkten hergestellt, die vorher nicht mit dem Feingitterpunkt verbunden waren. Bei der Aufstellung des Grobgitteroperators kann dies zu einem wesentlichem *fill-in* führen. Deshalb werden wir in der Praxis den Standard-Interpolationsoperator abbrechen oder *trunkieren*. Wir berücksichtigen dazu nur solche Kopplungen $\hat{a_{ij}}$, für die

$$|\hat{a}_{ij}| \geq \epsilon_{tr} \max_{k \in \mathcal{P}_i} |\hat{a}_{ik}|$$

gilt, d.h.

$$\mathcal{P}_i \leftarrow \mathcal{P}_i \setminus \{j : \ |\hat{a}_{ij}| < \max_{k \in \mathcal{P}_i} |a_{ik}|\}.$$

Kleinere Werte haben nur einen geringen Einfluss auf den Fehler und können daher weggelassen werden, ohne das Konvergenzverhalten allzusehr zu beeinträchtigen. (Wir erinnern noch einmal, dass jeder Interpolationsoperator mit vollem Rang zu einem konvergenten Verfahren führt, siehe Lemma 2.2.) Als Wert für ϵ_{tr} hat sich 0.2 bewährt.
Abschließend geben wir noch in den Programmen 2.6, 2.7 und 2.8 den Algorithmus zur Konstruktion der Interpolation an.

Programm 2.6 *AmgInterpolationStandard*(A, S, C, F, P)

```
begin
  for i ← 1 to N do
      if i ∈ C
        then
              p_ii ← 1;
         else
              AmgInterpolationTransform(A_i, S, C, F, Â_i, P_i);
              AmgInterpolationStandardRow(Â_i, P_i, P_i);
      fi;
  od;
end
```

Der Rechenaufwand diese Verfahrens beträgt pro Zeile $O\left(N_i + \sum_{j \in S_i \cap F} N_j\right)$. Insgesamt kommen wir so immer noch auf einen Aufwand von $O(N)$.

2.7 Vergröberung

In den vorherigen Abschnitten haben wir die Glättung und die Interpolation untersucht und sind dabei stets von einem gegebenem C/F-Splitting ausgegangen. Allerdings haben wir festgestellt, dass die Qualität der Interpolation für einen Feingitterpunkt i davon abhängt, inwieweit dieser Punkt starke Kopplungen zu Grobgitterpunkten j aufweist. Insbesondere für die direkte Interpolation sind direkte Kopplungen zu Grobgitterpunkten notwendig, bei der Standardinterpolation kann auch indirekt über stark gekoppelte Feingitterpunkte interpoliert werden. Die Aufstellung der Interpolation ist hier jedoch mit einem grösseren Aufwand verbunden und führt —wenn wir keine geeignete Trunkation vorsehen— zu einer erhöhten Operatorkomplexität (2.5)

$$c_A = \frac{\sum_{l=1}^{L_max} nonzeros\left(A^l\right)}{nonzeros\left(A^1\right)}.$$

Programm 2.7 *AmgInterpolationStandardTransform*$(A_i, S, C, F, \hat{A}_i, \mathcal{P}_i)$

```
begin
  P_i ← ∅;
  for j ← 1 to n do â_ij ← 0 od;
  for j ← 1 to N do
      if j ∈ S_i
        then
            if j ∈ C
              then                                          coarse grid point
                  â_ij ← â_ij + a_ij;
                  P_i ← P_i ∪ {j}
               else                                         eliminate
                  for k ∈ N_j do
                      â_ik ← â_ik + (a_ij · a_jk) / a_jj;
                      if k ∈ S_j ∩ C then P_i ← P_i ∪ {k}; fi;
                  od;
            fi;
         else                                  no strong connection: do not eliminate
            â_ij ← â_ij + a_ij;
      fi;
  od;
end.
```

Programm 2.8 *AmgInterpolationStandardRow*$(\hat{A}_i, \mathcal{P}_i, P_i)$

```
begin
  for j ← 1 to N do
      if j ∈ P_i
        then
            if â_ij < −ε_tr · max_{k∈P_i} |a⁻_ik|
              then
                  p_ij ← (Σ_{k∈N_i} â⁻_ik / Σ_{k∈P_i} â⁻_ik) â⁻_ij;
            elsif â_ij > ε_tr · max_{k∈P_i} |â⁺_ik|
                then
                    p_ij ← (Σ_{k∈N_i} â⁺_ik / Σ_{k∈P_i} â⁺_ik) â⁺_ij;
            fi;
      fi;
  od;
end
```

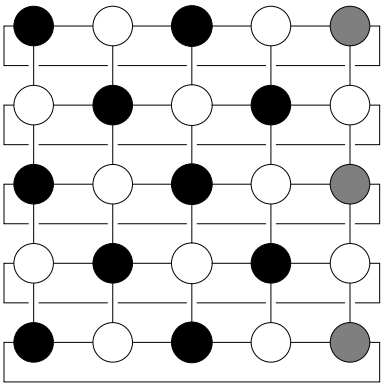


Abbildung 2.6: Vergröberung bei periodischen Randbedingungen

Wir können damit drei Bedingungen an das C/F-Splitting formulieren:

C1 Sei $i \in F$. Dann ist für jedes $j \in S_i$ entweder $j \in C$ oder $S_j \cap C_i \neq \emptyset$.

C2 Für $i \in C$ und $j \in S_i \cup S_i^T$ ist $j \notin C$. ($C \subset \Omega$ ist eine *unabhängige* oder *stabile* Menge.)

C3 C ist eine maximale Menge mit den vorigen beiden Eigenschaften.

Allerdings sind die ersten beiden Bedingungen nicht immer gleichzeitig erfüllbar.

Beispiel 2.4 Wir betrachten die Diskretisierung eines 5-Punkte-Sterns auf einem Gebiet von 5×5 Punkten mit periodischen Randbedingungen in x-Richtung. Wenn wir die grau markierten Grobgitterpunkte in F aufnehmen, ergibt sich ein Widerspruch zu Forderung C1. Fügen wir sie C hinzu, so ist die Forderung C2 verletzt.

Wir müssen also diese beiden Forderungen gewichten. Wir räumen dabei Forderung C1 ein größeres Gewicht als Forderung C2 ein, weil die Stabilität der Interpolation darauf beruht. Damit nehmen wir auch eine höhere Operator- und Gitterkomplexität in Kauf.
Der von Ruge und Stüben entwickelte Vergröberungsalgorithmus ([Stü99b], $\mathbf{S}_7$.1.1) läuft in zwei Phasen ab. Wir definieren eine Menge U der noch nicht zugeordneten Punkte. Anfangs wird sie gleich dem ganzen Gitter Ω gesetzt. Die erste Phase ordnet jedem Punkt i ein Gewicht λ_i zu, welches von der Anzahl der noch nicht zugeordneten stark gekoppelten Nachbarn und von der Anzahl der benachbarten Feingitterpunkte abhängt. Der Punkt mit dem jeweils höchsten Gewicht wird zum Grobgitterpunkt, alle seine stark gekoppelten Nachbarn $j \in S_i \cap U$ werden zu Feingitterpunkten. Die Gewichte

der noch nicht zugeordneten stark gekoppelten Punkte $k \in S_j^T \cap U$ werden erhöht, da diese gute Interpolationspunkte für die jetzt zu Feingitterpunkten gewordenen j darstellen. Die Gewichte der Punkte $j \in S_i \cap U$ werden erniedrigt, weil der Punkt i keine Interpolationspunkte braucht. Wir brechen das Verfahren ab, wenn es keine Punkte in U mit positivem Gewicht mehr gibt. Eventuell übriggebliebene Punkte werden den F-Punkten zugeordnet. Der genaue Ablauf ist in Programm 2.9 dargestellt.

Bei einer geeigneten Datenstruktur für U und λ (das heisst eine, die die Bestimmung von $\arg\max_{j \in U} \lambda_j$ in $O(1)$ erlaubt) ist dieser Algorithmus in $O(N)$ durchführbar. Wir verwenden eine aus Warteschlangen bestehende Struktur, die jedem vorkommenden Wert von λ eine Warteschlange Q_λ mit dementsprechend gewichteten Punkten $i \in U,\ \lambda_i = \lambda$ zuweist. Zusätzlich speichern wir für jedes $i \in U$ seine Position innerhalb der jeweiligen Warteschlange, um ihn bei jeder Änderung seines Gewichtes mit einem Aufwand von $O(1)$ in eine andere Warteschlange eingefügen zu können.

Dieses Verfahren alleine gewährleistet nicht, dass die Bedingung C1 auch wirklich erfüllt wird. Es wird die in Bild 2.6 grau markierten Punkte zu Feingitterpunkten machen, da sie über den Rand hinweg bereits mit C-Punkten

Programm 2.9 *AmgPhaseI*(Ω, S, S^T, C, F)

```
begin
  U ← Ω;
  C ← ∅;
  F ← ∅;
  for i ∈ Ω do λ_i ← |S_i^T|; od;
  while max_{i∈U} λ_i ≠ 0 do
        i ← arg max_{j∈U} λ_j;
        C ← C ∪ {i};
        for j ∈ S_i^T ∩ U do
            F ← F ∪ {j};
            for k ∈ S_j ∩ U do
                λ_k ← λ_k + 1;
            od;
        od;
        for j ∈ S_i ∩ U do
            λ_j ← λ_j − 1;
        od;
  od;
  F ← F ∪ U;
end
```

gekoppelt sind. Diese Fehler werden mit Hilfe einer zweiten Phase behoben. Wir durchlaufen alle $F-F$-Kopplungen, (Paare von stark gekoppelten Feingitterpunkten), d.h. alle $i, j \in F$ mit $j \in S_i$ und untersuchen, inwieweit jeweils j von den Interpolationspunkten von i abhängt, d.h. wir vergleichen

$$\frac{1}{\max_k |a_{jk}|} \sum_{k \in N_j \cap S_i \cap \left(C \cup \tilde{C}\right)} a_{jk} \text{ und } \beta \cdot \frac{a_{ij}}{\max_k |a_{ik}|}$$

Der Parameter β beträgt meist 0.35. Ist der erste Term kleiner oder gleich dem zweiten, so hängt j nur geringfügig von den Interpolationsvariablen von i ab, i dahingegen hängt stark von j ab. Damit schleichen sich bei der Interpolation des Wertes an i über j Werte ein, zu denen i keine direkte Verbindung aufweist. Insbesondere trifft dies zu, wenn wie im obigen Beispiel $S_j \cap S_i \cap C = \emptyset$ ist, d.h. wenn die beiden Feingitterpunkte keine starken Kopplungen zu einem gemeinsamen Grobgitterpunkt haben. Wir fügen in diesem Falle j den Grobgitterpunkten hinzu. Um den Anstieg der Grobgitterpunkte zu vermeiden, fügen wir in dem Fall, dass für einen Punkt i mehrere solche Punkte j eingefügt werden müssten, stattdessen i selbst hinzu. Der genaue Ablauf ist in Programm 2.10 dargestellt.

2.8 Setupphase

Wir haben nun alle für die Setupphase erforderlichen Komponenten konstruiert. Wir fassen diese zusammen und erhalten Programm 2.11, der zu einem gegebenem Operator A das C/F-Splitting und die Transferoperatoren P und R bestimmt. Zum Schluss betrachten wir das Gesamtprogramm 2.12. Zu gegebenem Feingitteroperator A, Interpolationstyp int, minimale Gittergröße N_{min} und maximale Levelanzahl L_{max} liefert dieser die Anzahl der erzeugten Level L, die Operatoren $\{A^l\}_{l=1}^{L}$ sowie die Transferoperatoren $\{P^l\}_{l=1}^{L-1}$ und $\{R^l\}_{l=1}^{L-1}$. Mit diesen Daten können wir dann den klassischen Mehrgitterzyklus, siehe Programm 2.1 zur iterativen Lösung starten.

Programm 2.10 *AmgPhaseII*$(\Omega, A, S, S^T, C, F)$

```
begin
  for i ∈ F do
      C̃ ← ∅;
      for j ∈ S_i ∩ F do
          if 1/(max_k |a_jk|) Σ_{k∈N_j∩S_i∩(C∪C̃)} a_jk ≤ β · a_ij/(max_k |a_ik|)
            then
                  if C̃ ≠ ∅
                    then change i, because otherwise 2 points need to be changed
                         C ← C ∪ {i};
                         F ← F \ {i};
                         GOTO NEXTi;
                     else          j could be an interpolation point for i
                         C̃ ← {j};
                  fi;
          fi;
      od;
      C ← C ∪ C̃;
      F ← F \ C̃;
      NEXTi :
  od;
end
```

Programm 2.11 *AmgLevel*$(\Omega, A, int, C, F, P, R, N_{coarse})$

```
begin
  AmgStrongCouplings (A, S, S^T);
  AmgPhaseI(Ω, S, C, F);
  AmgPhaseII(Ω, A, S, C, F);
  if int = standard
    then
         AmgInterpolationStandard(A, S, C, F, P);
     else
         AmgInterpolationDirect(A, S, C, F, P);
  fi;
  R ← P^T;
end.
```

Programm 2.12 *AmgSetup*$(\Omega, A, int, N_{min} L_{max}, L, \{A^l\}_{l=1}^{L}, \{P^l\}_{l=1}^{L-1} \{R^l\}_{l=1}^{L-1})$

begin
  $\Omega^1 \leftarrow \Omega$;
  $A^1 \leftarrow A$;
  for $l \leftarrow 1$ **to** $L_{max} - 1$ **do**
    AmgLevel$(\Omega^l, A^l, int, C^l, F^l, P^l, R^l)$;
    $\Omega^{l+1} \leftarrow C^l$;
    $A^{l+1} \leftarrow P^l A^l R^l$;
    if $|\Omega^{l+1}| \leq N_{min}$ **then** *break*; **fi**;
  od;
  $L \leftarrow l + 1$;
end.

Kapitel 3

Datenstrukturen und Algorithmen für den parallelen Mehrgitterzyklus

Nachdem wir alle Komponenten der (sequentiellen) Setupphase vorgestellt haben, wollen wir uns mit der effizienten Umsetzung auf Parallelrechnern befassen. Bevor wir uns mit der Parallelisierung der Setupphase beschäftigen, werfen wir zunächst einen Blick auf den Mehrgitterzyklus. Auf jedem Level müssen ein oder mehrere Glättungsschritte sowie zur Berechnung des Residuums, zur Restriktion und zur Interpolation jeweils eine Matrix-Vektor-Multiplikation parallel durchgeführt werden. Schließlich betrachten wir noch die parallele Aufstellung des Grobgitteroperators $A^{l+1} = R^l A^l P^l$.
Bei der Wahl einer geeigneten Datenstruktur für die dünnbesetzten parallelen Matrizen müssen wir neben einer effizienten Speicherung auch die möglichst

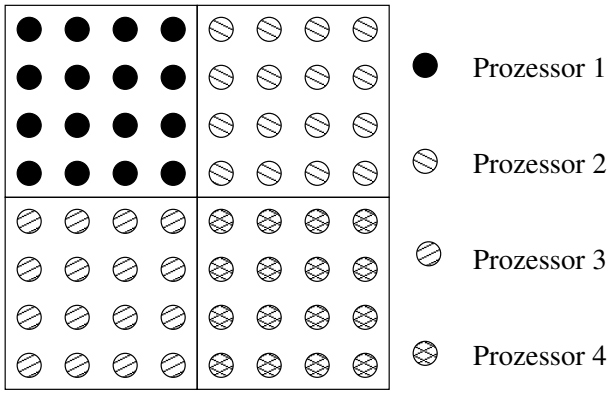


Abbildung 3.1: Disjunkte Partitionierung eines Diskretisierungsgebietes.

effiziente Durchführung der erforderlichen Operationen im Blick behalten. Im Folgenden sei $A = (a_{ij})_{i,j=1,\ldots,N}$ eine Matrix mit reellen Einträgen a_{ij}. Die Zeilen der Matrix seien jeweils mit $A_i = (a_{i,1}, \ldots, a_{i,N})$ bezeichnet.
$u = (u_j)_{j=1,\ldots,N}$ und $v = (v_j)_{j=1,\ldots,N}$ seien reelle Vektoren der Länge N. Die beteiligten Prozesse seien jeweils mit $q = 1, \ldots, Q$ nummeriert.
Wir ordnen jedem Index $j \in [1, \ldots, N]$ genau einen Prozessor $q \in \{1, \ldots, Q\}$ zu, auf dem die Werte der j-ten Komponenten der Vektoren u und v abgespeichert werden. Wir bezeichnen mit Ω_q^l oder abkürzend Ω_q die Menge der Indizes derjenigen Punkte $i \in \Omega^l$, die vom Prozess q verwaltet werden. Entsprechend sei $u_{(q)} = (u_i)_{i \in \Omega_q}$ der auf Prozessor q gespeicherte Teilvektor von u. Wir schreiben weiterhin

$$A = \begin{pmatrix} A_{(1)} \\ A_{(2)} \\ \vdots \\ A_{(Q)} \end{pmatrix}$$

wobei $A_{(q)}$ jeweils die auf Prozessor q gespeicherten Zeilen bezeichnet

$$A_{(q)} = (A_i)_{i \in \Omega_q}.$$

Die Verteilung der Variablen auf die Prozessoren wird in der Regel durch Zerlegung des zugrundeliegenden Diskretisierungsgebietes vorgenommen, wie es in Abbildung 3.1 für 64 Punkte und 4 Prozessoren vorgenommen worden ist. Bei komplexen geometrischen Strukturen muss ein Gebietszerlegungsverfahren angewendet werden, zum Beispiel raumfüllende Kurven ([Zum01], Kapitel 4). Eine andere Möglichkeit ist der Einsatz von Metis [KK98a], [KK98b] oder dessen parallele Variante, ParMetis [KK98c]. Diese Programme bieten neben der Gebietszerlegung auch die Möglichkeit, die Variablen nur anhand des durch die Feingittermatrix gegebenen Graphen in Teilgebiete aufzuteilen, können also auch dann verwendet werden, wenn dem Gleichungssystem kein Diskretisierungsgebiet zugrunde liegt.

3.1 Paralleles Matrix-Vektor-Produkt

Die Anwendung der Restriktion $R_l : \Omega^l \to \Omega^{l+1}$ und der Interpolation $P^l : \Omega^{l+1} \to \Omega^l$ sowie das Berechnen des Residuums $r^l \leftarrow f^l - A^l u^l$ erfordert die Anwendung eines Matrix-Vektor-Produktes:

$$v \leftarrow Au.$$

Der Rechenaufwand zur Berechnung dieses Produktes beträgt im Allgemeinen für volle Matrizen $O(N^2)$. Für einen Mehrgitter-Vorkonditionierer, der in $O(N)$ angewendet werden soll, ist das jedoch zu viel. Die Transferoperatoren sind jedoch dünnbesetzte Matrizen, d.h. sie enthalten nur $O(N)$ Nicht-Null-Einträge. Damit ist eine Matrix-Vektor-Multiplikation mit Rechenaufwand $O(N)$ möglich. Zur Speicherung der Matrix brauchen wir ebenfalls nur $O(N)$ Speicherplatz, wenn wir die Nulleinträge nicht abspeichern.
Wir verwenden dazu für jede Zeile i der Matrix zwei Arrays ([Bun88]):

- Ein Array enthält diejenigen Indizes j, für die $a_{ij} \neq 0$ ist.
- Das andere Array enthält die Werte a_{ij}.

Weiterhin wird für jede Zeile noch die Anzahl der Nicht-Nulleinträge gespeichert. Die so gespeicherten Zeilen der Matrix werden über die Prozessoren verteilt. Auf jedem Prozessor q speichern wir die Zeilen $A_i,\ i \in \Omega_q$ ab. Eine parallele Matrix-Vektor-Multiplikation läuft dann wie in Programm 3.1 ab. Der Kommunikationsaufwand hängt proportional von der Anzahl der Indizes

Programm 3.1 Paralleles Matrix-Vektor-Produkt $MatVec(A_{(q)}, u_{(q)}, v_{(q)})$

begin
$J_q \leftarrow \{j \notin \Omega_q : \exists i \in \Omega_q,\ a_{ij} \neq 0\}$;
$u_{J,q} \leftarrow (u_j)_{j \in J_q}$; communication
$v_{(q)} \leftarrow A_{(q)} \cdot (u_{(q)}, u_{J,q})$;
end

$j \notin \Omega_q$ mit $a_{ij} \neq 0$ für ein $i \in \Omega_q$ ab. Eine Interpolation in einem (geometrischen) Mehrgitterverfahren auf einem zweidimensionalen Gitter erfordert für jeden Prozess den Austausch von $O(\sqrt{N/Q})$ Werten, Im dreidimensionalen Fall beträgt der Aufwand $O((N/Q)^{2/3})$.

3.2 Glätter

Wir untersuchen jetzt die Parallelisierbarkeit der für das AMG gebräuchlichen Glätter, das heisst des Jacobi-Verfahrens und des Gauß-Seidel-Verfahrens. Dazu sei im Folgenden $A = L + D + U$, wobei D die Diagonalelemente a_{ii}, L die Elemente $a_{ij},\ j < i$ im linken unteren Teil und U die Elemente $a_{ij},\ j > i$ der Matrix A enthält.
Zunächst betrachten wir das Jacobiverfahren, dessen Iterationsvorschrift durch

$$u^{it+1} \leftarrow u^{it} + D^{-1}(f - Au^{it})$$

gegeben ist ($it \geq 0$). Die Anwendung des Jacobi-Verfahrens besteht demnach aus der Anwendung eines Matrix-Vektor-Produktes (siehe 3.1), einer Diagonalskalierung und zwei Additionen, beinhaltet also einen Kommunikationsschritt. Diesen müssen vor jedem weiteren Iterationsschritt it wiederholen. Dennoch ist dieses Verfahren einfach parallel auszuführen.
Das Gauß-Seidel-Verfahren

$$u^{it+1} \leftarrow u^{it} + (L + D)^{-1}(f - Au^{it})$$

ist leider nicht direkt parallelisierbar. Ein einzelner Gauß-Seidel-Schritt, angewandt auf die Variable u_i^{it+1}

$$u_i^{it+1} \leftarrow \frac{1}{a_{ii}} \left(f_i - \sum_{j<i} a_{ij} u_j^{\mathbf{it+1}} + \sum_{j>i} a_{ij} u_j^{it} \right)$$

erfordert die Kenntnis aller *neuen* Iterierten u_j^{it+1}, $j < i$. Insbesondere heisst dies, dass Prozess q erst anfangen kann, wenn Prozess $q-1$ den Gauß-Seidel-Schritt für *alle* seine Unbekannten durchgeführt hat. Dieses Verfahren ist also inhärent sequentiell. Eine mögliche Abhilfe ist hier multi-color-relaxation ([Zum01], Abschnitt 5.2.2), bei der zunächst alle Punkte gefärbt werden, so dass keine zwei Indizes i und j mit $a_{ij} \neq 0$ dieselbe Farbe aufweisen. Während eines Gauß-Seidel-Schrittes können alle Variablen der gleichen Farbe gleichzeitig relaxiert werden. Vor der Relaxation der nächsten Farbe ist eine Aufdatierung notwendig.
Dieses Verfahren erfordert damit pro Farbe einen Kommunikationsschritt. Außerdem muss bei unstrukturierten Gittern zunächst ein paralleles Graphfärbungsverfahren angewendet werden.
Wir wollen diese Kosten vermeiden, aber trotzdem auf die besseren Glättungseigenschaften des Gauß-Seidel-Verfahrens (SOR-Verfahrens) auch beim parallelisierten Mehrgitterverfahren nicht verzichten, vergleiche 2.2 . Dafür verwenden wir eine Technik der Gebietszerlegung, das *Block-Jacobi-Verfahren* ([BBG+03]). Dieses besteht aus

- einer äußeren Iteration, in der das Jacobi-Verfahren auf Blöcken (statt auf einzelnen Variablen) angewandt wird,

- einer inneren Iteration oder einem direkten Verfahren, welches die Anwendung von D_i^{-1} für jeden Diagonalblock approximiert oder durchführt.

Wir schreiben dazu die Matrix A in $Q \times Q$ Blöcken:

$$A = \begin{pmatrix} D_{(1)} & U_{(1,2)} & \cdots & \cdots & U_{(1,Q)} \\ L_{(2,1)} & D_{(2)} & U_{(2,3)} & \cdots & U_{(2,Q)} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & U_{(Q-1,Q)} \\ L_{(Q,1)} & \cdots & \cdots & L_{(Q,Q-1)} & D_{(Q)} \end{pmatrix}$$

Dabei enthält die quadratische Matrix $D_{(q)}$ für jedes $q = 1, \ldots, Q$ die Einträge $a_{ij},\ i, j \in \Omega_q$. Entsprechend enthalten die Matrizen $U_{(q,q^*)}$ $(q^* > q)$ und L_{q,q^*} $(q^* < q)$ die Einträge a_{ij} mit $i \in \Omega_q$ und $j \in I_{q^*}$.
Damit erhält das Block-Jacobi-Verfahren für die Unbekannten in $u_{(a)}$ folgende Gestalt:

$$u_{(a)}^{it+1} \leftarrow D_{(q)}^{-1} \left(f_{(q)} - \sum_{q^*<q} L_{(q,q^*)} u_{(q^*)}^{it} - \sum_{q^*>q} U_{(q,q^*)} u_{(q^*)}^{it} \right).$$

Wir müssen also dieselben Unbekannten kommunizieren, die auch für das Jacobiverfahren benötigt werden.
Die Anwendung der Matrix $D_{(q)}^{-1}$ wird jetzt mit einem inneren Verfahren approximiert. Wir verwenden dafür in der Regel einen oder mehrere Gauß-Seidel- (bzw. SOR-) Schritte.
Auf diese Weise werden die inneren Variablen (d.h. diejenigen u_i, für die $a_{ij} \neq 0$ nur für $i, j \in \Omega_q$) vollständig mit dem Gauß-Seidel-Verfahren relaxiert. Solange die Randvariablen nur einen geringen Teil der Gesamtvariablen ausmachen, sind die Glättungseigenschaften des Block-Jacobi-Verfahrens mit innerer Gauß-Seidel-Iteration kaum schlechter als die des Gauß-Seidel-Verfahrens, vgl. auch 3.2.

Wir müssen beachten, dass bei steigendem Level der Anteil der Randvariablen zunimmt. Weil die Relaxation dieser Variablen teilweise auf noch nicht aufdatierte Werte zurückgreift, entspricht das Dämpfungsverhalten des Fehlers an diesen Punkten dem des Jacobiverfahrens. Insgesamt gilt für die Glättungseigenschaft des Block-Jacobi-Verfahrens folgender Satz.

Satz 3.1 *([Yan04])*
Sei A symmetrisch, positiv definit und $\eta = \rho(\tilde{D}^{-1}A)$, dabei bezeichnet

$$\tilde{D} = \begin{pmatrix} D_{(1)} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & D_{(Q)} \end{pmatrix}.$$

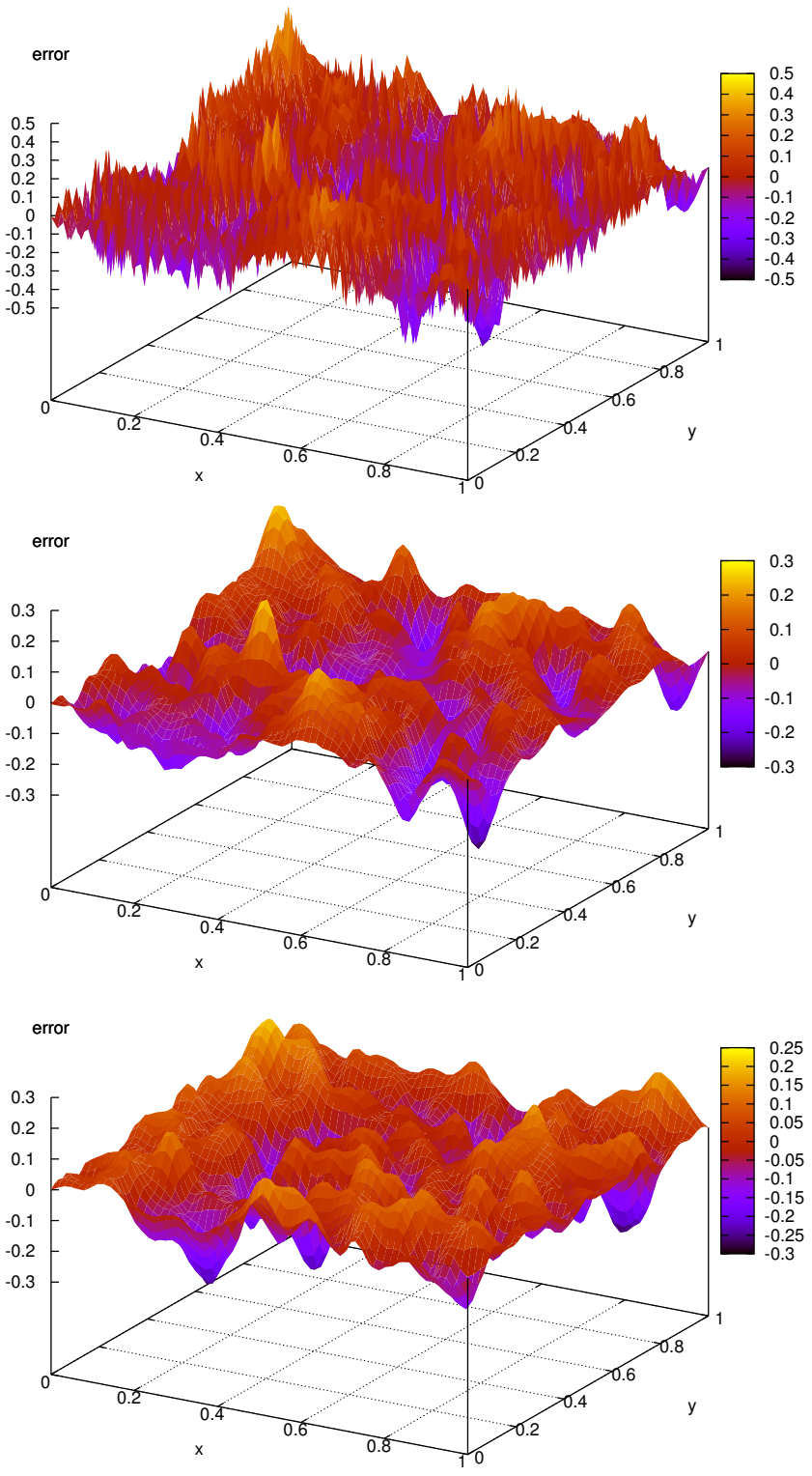


Abbildung 3.2: Diskretisierung eines 5-Punkte Sternes auf einem Gebiet von 64×64 Punkten. Die Bilder zeigen von oben nach unten den Fehler nach jeweils 5 Jacobi-, Gauß-Seidel- und Block-Jacobi-Iterationen. Beim Block-Jacobi-Verfahren wurde das Diskretisierungsgebiet in 4 Teilgebiete aufgeteilt, auf denen ein inneres Gauß-Seidel-Verfahren angewendet wurde. Der Startwert war wiederum zufällig gewählt und auf 1 normiert.

Dann erfüllt Block-Jacobi-Relaxation mit Relaxationsparameter $0 < \omega < \frac{2}{\eta}$ *die Glättungseigenschaft (2.11) mit* $\sigma = \omega(2 - \omega\eta)$*. Der optimale Relaxationsparameter ist durch* $\omega^* = \frac{1}{\eta}$ *gegeben, in diesem Fall ist* $\sigma = \frac{1}{\rho(D^{-1}\tilde{D})\rho(\tilde{D}^{-1}A)}$.

Eine Abschätzung für $\rho(\tilde{D}^{-1}A)$ können wir zum Beispiel durch die Anwendung einiger CG-Schritte angewendet auf das mittels $\tilde{D}$ vorkonditionierte System $Au = b$ erhalten.
Wir müssen also auch das Block-Jacobi-Verfahren gegebenenfalls dämpfen, um ein konvergentes Verfahren zu erhalten.

3.3 Lösung auf dem gröbsten Gitter

Für eine gute Konvergenz des Mehrgitteralgorithmus ist eine möglicht genaue Lösung des Problems auf dem gröbsten Gitter wünschenswert. Im sequentiellen Fall können wir bis zu einer geringen Anzahl von Punkten ($\sim 1 - 50$) pro Prozessor vergröbern und das sich ergebene lineare Gleichungssystem ohne grösseren Aufwand exakt lösen.
Wie wir im nächsten Kapitel sehen werden, gilt dies nicht für alle parallelen Vergröberungsalgorithmen. Bei grösseren Prozessoranzahlen können auch auf dem gröbsten Level noch mehrere Hundert Unbekannte auftreten. In diesem Fall reicht die Durchführung einiger weniger (Block-)Jacobi-Schritte nicht aus, um den Fehler hinreichend zu reduzieren. Wir müssen also ein paralleles direktes Lösungsverfahren benutzen, welches jedoch sehr aufwändig ist. Um diesen Aufwand zu vermeiden, fassen wir auf gröbereren Leveln mehrere Teilgebiete zu einem zusammenzufassen und behandeln das gröbste Gitter mit nur einem Prozessor. Dann lässt sich auch wieder eine kleinere Anzahl von Grobgitterpunkten erreichen, außerdem steigt der Anteil der Randvariablen im Laufe der Vergröberung dann weniger stark an oder nimmt auf den gröbsten Leveln sogar wieder ab.

3.4 Paralleles Triple-Matrix-Produkt

Die obigen Überlegungen legen die Verwendung von Datenstrukturen aus der Bibliothek PETSc (Portable, Extensible Toolkit for Scientific Computation) [BBG+03] nahe. Dieses Programmpaket bietet unter Anderem Datenstrukturen für zeilenorientiert abgespeicherte parallele Matrizen, parallele Vektoren, Algorithmen für das Matrix-Vektor-Produkt und das Skalarprodukt. Weiterhin stellt PETSc eine große Auswahl von Krylovverfahren (u. A. Richardson-Relaxation, CG, BiCG(-STAB), GMRES) und Vorkonditionierern/Glättern

(ILU, (Block-)Jacobi, Gauß-Seidel, (S)SOR), direkten Lösern (LU, Cholesky) und ein Grundgerüst für (geometrische) Mehrgitterverfahren bereit, die meisten davon sind auch parallelisiert. Eine Zerlegung unstrukturierter Diskretisierungsgebiete mittels ParMetis [KK98c] ist ebenfalls möglich.
Zur Kommunikation benutzt PETSc den Standard MPI [Mes95], die sowohl auf einem Netzwerk von Workstations als auch auf Großrechnern implementiert worden ist. Damit ist eine weitgehende Portabilität des Programmcodes gegeben.
Ein paralleles Triple-Matrix-Produkt gehört nicht zum Funktionsumfang von PETSc, in geometrischen Mehrgitterverfahren wird es auch nicht benötigt. Wir brauchen aber zur Aufstellung des Grobgitteroperators $A^{l+1} = R^l A^l P^l$ eine schnelle Multiplikationsroutine. Für eine möglichst effiziente Umsetzung müssen wir zunächst genau untersuchen, wie eine parallele Matrix unter PETSc organisiert ist. Programm 3.2 zeigt einen Auszug aus der Definition einer parallelen Matrix in PETSc.

Der auf dem Prozessor q abgespeicherte Block $A_{(q)}$ der parallelen Ma-

Programm 3.2 Datenstruktur einer parallelen Matrix

```
typedef struct {
  int           *rowners,*cowners;
                  /* ranges owned by each processor */
  int           rstart,rend;
                  /* starting and ending owned rows */
  int           cstart,cend;
                  /* starting and ending owned columns */
  Mat           M,N;
                  /* local submatrices: M (diag part),
                                        N (off-diag part) */
  ...
  int           *garray;
                  /* global index of all off-processor
                     columns */
  ...
} Mat_MPIAIJ;
```

trix A ist also zweigeteilt: Der mit M bezeichnete Teil enthält die Einträge $a_{ij},\ i,j \in \Omega_q$, der mit N bezeichnete Block enthält die Einträge $a_{ij},\ i \in \Omega_q,\ j \notin \Omega_q$. Wir sortieren die Spalten in $A_{(q)}$ um:

$$A_{(q)} \cdot C_q = \left(A_{(q),M} A_{(q),N}\right)$$

wobei C_q die Spaltentransformation ist, die jede Spalte $j \in 1, \ldots, |\Omega_q|$ mit der j-ten Spalte aus Ω_q vertauscht:

$$C_q = \begin{pmatrix} 0 & \cdots & \cdots & 0 & I & 0 & \cdots & \cdots & 0 \\ 0 & I & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & & & & & \vdots \\ 0 & \cdots & 0 & I & 0 & \cdots & \cdots & \cdots & 0 \\ I & 0 & \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & I & 0 & \cdots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & I & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & I \end{pmatrix}$$

Auf Prozessor q sind damit folgende Daten verfügbar:

$$\left(R_{(q),M}\ R_{(q),N}\right), \quad \left(A_{(q),M}\ A_{(q),N}\right) \text{ und } \left(P_{(q),M}\ P_{(q),N}\right)$$

Wir betrachten jetzt den Ablauf auf Prozessor q und schreiben abkürzend $A_M = A_{(q),M}$ und $A_N = A_{(q),N}$. Die Spalten- und Zeilenindizes dieser Matrizen beginnen jeweils bei 1 und sind fortlaufend nummeriert (lokale Indizierung). Weiterhin enthalte $A.\texttt{garray}$ zu jedem Spaltenindex in A_N den globalen Spaltenindex, das heisst denjenigen in A. Wir bezeichnen die einzelnen Werte mit $A.\texttt{garray}[i]$. Wir führen die Multiplikation in folgender Reihenfolge aus:

1 Zunächst multiplizieren wir A_M (deren Spaltenindizes genau den lokalen Zeilenindizes von P entsprechen) mit P_M und P_N. Der Faktor A_N wird nach einer Kommunikationsphase mit den entsprechenden (nichtlokalen) Zeilen von P (diejenigen in $A.\texttt{garray}$) multipliziert. Das Produkt wird mit T bezeichnet und ist wieder wie oben beschrieben in T_M und T_N aufgeteilt, wobei die Spalten von T_M denen in P_M entsprechen.

2 In einem zweiten Schritt multiplizieren wir jetzt R_M mit T_M und T_N sowie (wiederum nach einem Kommunikationsschritt) R_N mit den entsprechenden Zeilen von T.

Der genaue Ablauf ist in Programm 3.3 dargestellt. Diese Vorgehensweise „von hinten nach vorne“ hat vor allem bei der Agglomeration mehrerer Prozessorteilgebiete einen großen Vorteil gegenüber einem Verfahren, bei dem zuerst R mit A und dann dieses Produkt mit P multipliziert wird. Bei der Agglomeration enthält nämlich die Matrix R (Restriktion) auf den aussteigenden (auf dem nächsten Level nicht mehr beteiligten) Prozessoren keine

Programm 3.3 Triple-Matrix-Produkt

```
begin
  for i := 1 to size(A.garray) do
      K_i = P_{M.garray[i]};                          communication
  od;
  allocate space for T_A and T_B;
  T_A ← M_A · N_A + M_B · K_A;
  T_B ← M_A · N_B + M_B · K_B;
  destroy(K);
  for i := 1 to size(T.garray) do
      K_i = T_{R.garray[i]};                          communication
  od;
  allocate space for A_M and A_N;
  A_M ← R_M · T_M + R_N · T_M;
  A_N ← R_M · T_N + R_N · T_N;
  destroy(K);
end
```

lokalen Zeilen. Ein Vorgehen „von vorne nach hinten“ ließe diesen Prozessor völlig unbeschäftigt, und alle lokalen Zeilen des Interpolations- und des Feingitteroperators müssten an einen anderen Prozess übergeben werden. Bei dem von uns benutzten Verfahren multipliziert der aussteigende Prozessor zunächst noch seinen Anteil des Feingitteroperators mit den entsprechenden Zeilen der Interpolationsmatrix, bevor dieses Resultat verschickt wird. Es muss also nicht der komplette lokale Block der Interpolationsmatrix kommuniziert werden, sondern nur die in `garray` auftauchenden Zeilen.

Bemerkung 3.1 Es sei noch darauf hingewiesen, dass eine möglichst genaue Präallokation des von der Grobgittermatrix benötigten Speichers unerlässlich ist. Wird dieser unterschätzt, so beansprucht eine Erweiterung u.U. mehr Zeit als die eigentliche Multiplikation. Eine allzu großzügige Allokation erhöht wiederum unnötigerweise den gesamten Speicheraufwand für das Programm. Wir lösen dieses Problem, indem wir die Schleifen in den sequentiellen Multiplikationsalgorithmen zweimal durchlaufen. Zuerst zählen wir für jede Zeile der Produktmatrix die Anzahl der Einträge, danach alloziieren wir den benötigten Speicher. Im zweiten Durchlauf führen wir dann die eigentliche Multiplikation aus. Diese Vorgehensweise führt zwar zu etwas höheren Rechenkosten, es hat sich jedoch vor allem bei unstrukturierten Gittern oder springenden Koeffizienten als die schnellste Lösung herausgestellt.

Kapitel 4

Parallelisierung der Setupphase

Im Folgenden bezeichne $\Omega_p = \Omega_p^l \subset \Omega^l$ die Menge derjenigen Gitterpunkte, die vom Prozessor $p \in \{1, \ldots, P\}$ verwaltet werden. Wir definieren weiterhin die äußeren Variablen $\partial\Omega_p$(auch Randvariablen genannt) und die inneren Variablen $\overset{\circ}{\Omega}_p$ eines Prozessors p

$$\begin{aligned} \partial\Omega_p &= \{i \in \Omega_p : \ a_{ij} \neq 0 \text{ für ein } j \in \Omega_q, \ q \neq p\}, \\ \overset{\circ}{\Omega}_p &= \Omega_p \setminus \partial\Omega_p. \end{aligned}$$

Weiterhin bezeichnen wir mit

$$\begin{aligned} C_p &:= C \cap \Omega_p, \\ F_p &:= F \cap \Omega_p \end{aligned}$$

die Mengen der Grob- und Feingitterpunkte auf Prozessor p. In den Mengen $S_{(p)}$ und $S_{(p)}^T$ fassen wir alle starken Kopplungen von Punkten aus Ω_p zusammen,

$$\begin{aligned} S_{(p)} &:= \{S_i\}_{i \in \Omega_p}, \\ S_{(p)}^T &:= \{S_i^T\}_{i \in \Omega_p}. \end{aligned}$$

Wie im vorigen Kapitel bezeichnet A_i die i-te Zeile der Matrix A und $A_{(p)} = (A_i)_{i \in \Omega_p}$ den auf Prozessor p gespeicherten Teil der Matrix A.
Wir beschreiben jetzt die Parallelisierung aller für der Setupphase notwendigen Komponenten, insbesondere die Problematik bei der parallelen Grobgitterwahl.

4.1 Kopplungen

Die starken Kopplungen S können wir auf jedem Prozessor p völlig lokal ermitteln, da für die Bestimmung der Zugehörigkeit von $j \in N_i$ zu S_i nur die

Zeile A_i vorhanden sein muss. Für das Einfügen von i in S_j^T, $j \notin \Omega_p$ wäre ein Kommunikationsschritt erforderlich. Die Datenstruktur S^T wird jedoch lediglich benötigt, um ausgehend von einem Grobgitterpunkt i die benachbarten Punkte $j \in S_i^T$ zu Feingitterpunkten zu machen (vergleiche Programm 2.9). Wie wir sehen werden, führen die parallelen Vergröberungsalgorithmen diesen Schritt jedoch nicht für $j \notin \Omega_p$ durch, womit wir auf das Einfügen von i in S_j^T, $j \notin \Omega_p$, und die damit einhergehende Kommunikation verzichten können. Wir müssen lediglich die Werte $|S_j^T|$ übertragen, um die korrekten Startwerte für die Gewichte λ_j zu erhalten. In Programm 4.1 ist der parallele

Programm 4.1 *ParallelStrongCouplings*$(A_{(p)}, S_{(p)}, S_{(p)}^T, \{\lambda\}_{i\in\Omega_p})$

begin
  StrongCouplings$(A_{(p)}, S, S^T, \{\lambda_i\}_{i\in\Omega_p})$;
  for $i \notin \Omega_p,\ S_i^T \neq \emptyset$ **do**
    $\mu_i \leftarrow |S_i^T|$;
    send μ_i *to* $q,\ i \in I_q$; communication
  od;
  for $i \in \Omega_p$ **do**
    $\lambda_i \leftarrow |S_i^T|$;
  od;
  for *received* μ_i **do**
    $\lambda_i \leftarrow \lambda_i + \mu_i$;
  od;
end.

Algorithmus zur Ermittlung der starken Kopplungen dargestellt.

4.2 Interpolation

Die direkte Interpolation (Abschnitt 2.6.1) ist sofort parallelisierbar. Wir müssen nur vor der Anwendung von *AmgInterpolationDirect* (2.4) für alle Punkte $i \in \partial\Omega_p$ die Zugehörigkeit von $j \in S_i \setminus \Omega_p$ zu C oder F ermitteln. Aufgrund der in Abschnitt 2.6.1 beschriebenen Instabilität dieses Interpolationsoperators reicht diese jedoch nicht für eine schnelle Konvergenz aus. Wir wenden uns deshalb der Standardinterpolation (Abschnitt 2.6.2) zu.
Die Aufstellung dieses Interpolationsoperators für den Feingitterpunkt $i \in F_p$ erfordert die Transformation der Matrixzeile A_i. Hierzu benötigen wir die Matrixzeilen A_j für alle $j \in S_i \cap F$, also auch diejenigen, für die $j \notin \Omega_p$ ist und die erst von einem anderen Prozessor q übertragen werden müssen.

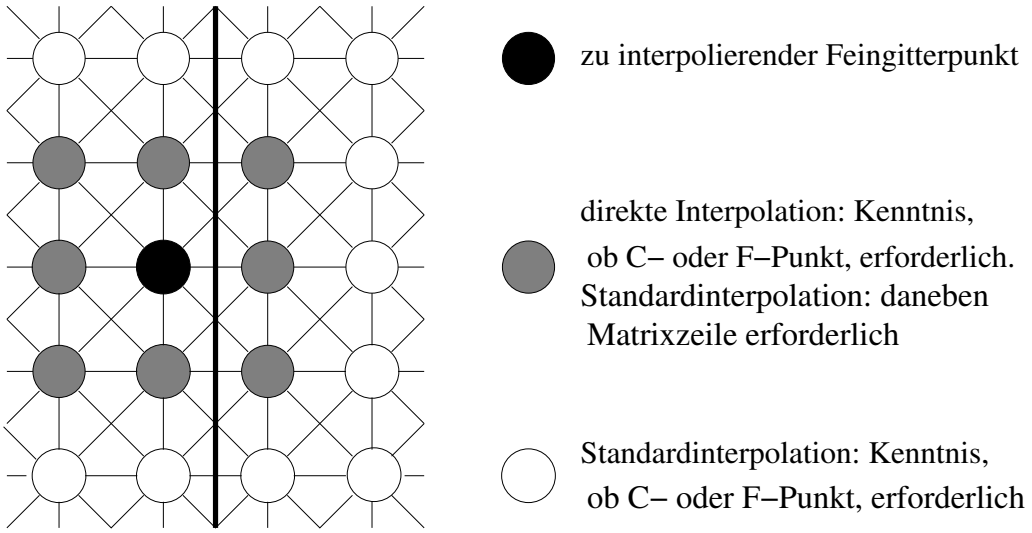


Abbildung 4.1: Für die Interpolation notwendige Daten

Auf den übertragenen Zeilen j wenden wir dann *AmgStrongCouplings* (Programm 2.3) an, um die starken Kopplungen S_j zu ermitteln (dies ist in der Praxis günstiger, als in einem weiteren Kommunikationsschritt die Mengen S_j von Prozessor q auf Prozessor p zu übertragen). Danach müssen wir für $k \in S_j \setminus \Omega_p$ noch ermitteln, ob $k \in C$ oder $k \in F$ ist, soweit die C/F-Zuordnung von k dem Prozessor p noch nicht bekannt ist.
Wir betrachten dazu beispielsweise die Diskretisierung eines Laplaceoperators mittels eines 9-Punkte-Sternes. Die direkte Interpolation erfordert für jeden Feingitterpunkt am Rand die Kommunikation von 3 Werten ((C/F)-Zuordnung der stark gekoppelten Punkte im anderen Prozessorteilgebiet). Für die Standardinterpolation benötigen wir zusätzlich bis zu drei Zeilen des Operators auf dem Nachbargebiet (je nach Zugehörigkeit der Nachbarpunkte zu C oder F), und die Kommunikation von sieben weiteren C/F-Zuordnungen (siehe Abbildung 4.1). Dabei enthalten die zu übertragenden Zeilen wiederum jeweils neun Spaltenindizes und neun Matrixeinträge. Die meisten dieser Variablen werden mehrfach benötigt, so dass wir für einen Teilgebietsrand der Größe n insgesamt $4n$ C/F-Werte und —je nach C/F-Splitting auf dem Nachbargebiet— $18n$ bis $36n$ Matrixwerte versenden müssen. Die direkte Interpolation kommt mit der Kommunikation von $2n$ (C/F)-Werten aus.
Ein weiteres Problem bei der Standardinterpolation über Gebietsränder hinweg erwächst daraus, dass der Interpolationsoperator schon zwei „Schichten“ in das andere Teilgebiet hinein ragen kann. Dasselbe gilt demzufolge auch

für die Restriktion. Dieses führt zu einem hohen Kommunikationsaufwand während der Aufstellung des Grobgitteroperators, welcher damit bis weit in das Nachbargebiet hineinragt. Damit nimmt auch der Anteil der Teilgebietsrandvariablen $j \in \partial\Omega_p$ zu. Diese können jedoch nicht so gut geglättet werden wie die inneren Variablen (siehe Abschnitt 3.2). Das Abbrechen (und Reskalieren) des Interpolationsoperators vermindert zwar dieses zuletzt genannte Problem, jedoch bleibt erstmal der erhöhte Kommunikationsaufwand während des Aufstellen des Interpolationsoperators.
Für unsere Experimente haben wir über den Rand hinweg deshalb eine direkte Interpolationsformel verwendet. Bei einer guten Agglomeration und einem nicht zu hohem Randvariablenanteil lassen sich damit sowohl die Setupphase als auch die Lösungsphase beschleunigen, ohne dass die Qualität der Konvergenz wesentlich zurückgeht.
Wir eliminieren dazu für den Punkt $i \in F_p$ zunächst die Kopplungen

$$a_{ij}e_j \mapsto -\frac{a_{ij}}{a_{jj}} \sum_{k \in N_j} a_{ik}e_k$$

für $j \in S_i \cap F_p$. Wir erhalten eine neue Matrixzeile $\hat{A}_i$

$$\hat{A}_i = (\hat{a}_{i1}, \cdots, \hat{a}_{iN}),$$

welche sich von der in (2.20) dargestellten Zeile dadurch unterscheidet, dass die Einträge a_{ij} für $j \in S_i \cap F_q$, $q \neq p$, nicht eliminiert sind.
Die Menge der Interpolationsvariablen $\mathcal{P}_i$ ergibt sich damit im Unterschied zu 2.21 gemäß

$$\mathcal{P}_i := \left(S_i \cup \bigcup_{j \in S_i \cap F_p} S_j \right) \cap C.$$

Die Interpolationsvorschrift ergibt sich dann weiter analog zu 2.22, d.h.

$$e_i = -\frac{1}{\hat{a}_{ii}} \cdot \left(\frac{\sum_{j \in N_i} \hat{a}_{ij}^-}{\sum_{j \in \mathcal{P}_i} \hat{a}_{ij}^-} \sum_{j \in \mathcal{P}_i} \hat{a}_{ij}^- e_j + \frac{\sum_{j \in N_i} \hat{a}_{ij}^+}{\sum_{j \in \mathcal{P}_i} \hat{a}_{ij}^+} \sum_{j \in \mathcal{P}_i} \hat{a}_{ij}^+ e_j \right),$$

wobei nur ein Summand vorhanden ist, falls die Matrix keine positiven (oder negativen) Kopplungen aufweist.

4.3 Klassischen parallele Vergröberungsstrategien

Die Grobgitterwahl stellt die größte Herausforderung bei der Parallelisierung der Setupphase dar ([CFHJ98]). Die Arbeitsweise des klassischen Ver-

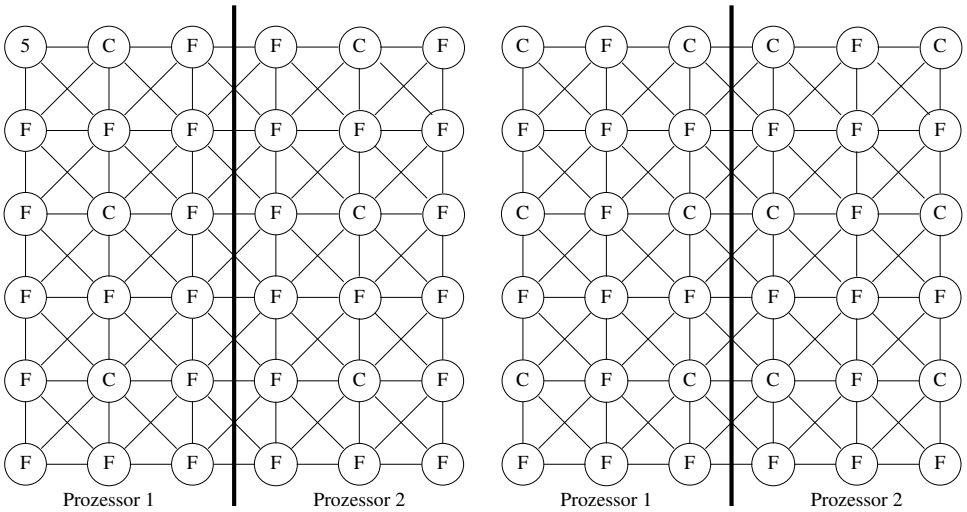


Abbildung 4.2: Zwei verschiedene, parallel erstellte Grobgitter für einen 9-Punkte-Stern. Das linke Gitter führt zu einer instabilen Interpolation, das rechte Gitter zu einer erhöhten Operatorkomplexität.

gröberungsalgorithmus nach Ruge und Stüben ist inhärent sequentiell, da die Aufnahme eines Punktes in das grobe Gitter die Wahl für jeden noch nicht zugeordneten Punkt beeinflussen kann. Wenn wir nun beispielsweise den klassischen Algorithmus auf die einzelnen Prozessorteilgebiete ansetzen, erfüllt das sich ergebene zusammengesetzte Grobgitter nicht zwangsläufig die Bedingungen C1-C3. Abbildung 4.3 zeigt zwei mögliche Grobgitterwahlen, die von jeweils zwei Prozessoren vorgenommen worden sind und die daraus resultierenden Probleme. Der erste Fall führt zu einer instabilen Interpolation, weil über den Rand hinweg jeweils zwei Feingitterpunkte stark gekoppelt sind, ohne dass diese von einem gemeinsamen Grobgitterpunkt interpolieren. Im zweiten Fall tritt dieses Problem zwar nicht auf, es werden aber mehr Grobgitterpunkte als nötig ausgewählt, was die Operatorkomplexität erhöhen kann.

4.3.1 Subdomain Blocking

Ziel des in [KS99] vorgestellten Subdomain Blockings ist es, die Vergröberungsverfahren auf den Teilgebieten zu entkoppeln. Wir ordnen zunächst die Variablen auf den Teilgebietsrändern dem groben oder dem feinen Gitter zu und können dann den klassischen Algorithmus auf dem Teilgebietsinnern ansetzen. Zunächst betrachten wir das *Full Subdomain Blocking*. Hierbei werden alle Randvariablen eines Prozessorgebietes zu C-Variablen, siehe Programm

4.2 und Abbildung 4.4 (b). Hiermit sind alle Interpolationsoperatoren nur

Programm 4.2 *AmgFSB*(Ω_p, C_p, F_p)

begin
  for $i \in \partial\Omega_p$ **do**
    $C_p \leftarrow C_p \cup \{i\}$;
  od

noch lokal. Damit findet während der Setupphase bis auf die Aufstellung des Grobgitteroperators keine Kommunikation statt. Dieses Verfahren hat aber einen großen Nachteil: Alle Randvariablen werden immer wieder in die nächste Ebene übernommen. Dies führt zu einer hohen Gitter- und Operatorkomplexität und damit zu einem hohen Speicherbedarf. Die verhältnismäßig vielen Randpunkte führen außerdem zu einer schlechten und kommunikationsintensiven Glättung, siehe Abschnitt 3.2. Wir sehen in Abbildung 4.4 (b) die Anwendung dieses Verfahrens auf die 5-Punkte-Diskretisierung des zweidimensionalen Laplaceoperators. Wir sehen, dass die kompletten Teilgebietsränder in das grobe Gitter aufgenommen worden sind.
Eine Verbesserung lässt sich dadurch erzielen, dass ab einem gewissen Level l_a mehrere Teilgebiete zusammengefasst und auf einem Prozessor bearbeitet werden. Einige der Randvariablen werden dadurch zu inneren Variablen und müssen nicht mehr bei jeder Vergröberung erhalten bleiben. Diese Agglomeration wird soweit wiederholt, bis nur noch ein Prozessor übrigbleibt.
Ein besseres Verfahren ist das *Minimum Subdomain Blocking*. Anstatt alle Randvariablen zu C-Variablen zu machen wird hierbei der Vergröberungsalgorithmus einmal auf die Menge der Randvariablen angewendet. Wiederum werden alle starken Verbindungen zu anderen Teilgebieten hierbei ignoriert, siehe Programm 4.3. Dieses Verfahren ist jedoch nicht in der Lage, bis zu ei-

Programm 4.3 *AmgMSB*$(\Omega_p, A_p, S_{(p)}, S_{(p)}^T, C_p, F_p)$

begin
  AmgPhaseI$(\partial\Omega_p, S_{(p)}, S_{(p)}^T, C_p, F_P)$;
  AmgPhaseII$(\partial\Omega_p, A_p, S_{(p)}, C_p, F_p)$;
end

ner vorgegebenen Punkteanzahl zu vergröbern. Pro Prozessor wird nämlich auf jedem Level mindestens ein Grobgitterpunkt gewählt, so dass die Anzahl der Prozessoren eine untere Schranke für die Anzahl der Grobgitterpunkte darstellt. Dieses Problem lösen wir wiederum dadurch, dass wir mehrere Prozessorteilgebiete mit steigendem Level zusammenfassen. Ein weiterer, noch

gewichtigerer Nachteil besteht darin, dass dieses Verfahren nicht überprüft, ob zwei stark gekoppelte Feingitterpunkte $i \in \Omega_p$ und $j \in S_i \cap \Omega_q$, $q \neq p$, eine starke Kopplung zu einem gemeinsamen Grobgitterpunkt aufweisen. Dies kann die Konvergenzgeschwindigkeit des Verfahrens beeinträchtigen, auch wenn für jeden Feingitterpunkt eine Interpolationsformel aufgestellt werden kann.
Abbildung 4.4 (c) zeigt das erste Level nach Vergröberung mittels des MSB-Verfahrens, angewendet auf die 5-Punkte-Diskretisierung des Laplaceoperators. Wir sehen, dass die $F-F$-Kopplungen über die Teilgebietsgrenzen des rechten oberen Quadranten nicht aufgelöst wurden. Es wurde dasselbe Gitter wie in Abbildung 4.4 (a) (ohne Randbehandlung) erzeugt.

4.3.2 RS3-Algorithmus

Dieses Verfahren wird in [HY01], Abschnitt 4.2 vorgestellt. Wie bei dem Subdomain Blocking führt auch hier jeder Prozessor das klassische Vergröberungsverfahren lokal auf seinen Knoten durch. Die Randbehandlung erfolgt jedoch nicht vor, sondern nach der Behandlung der inneren Punkte. Zunächst wird von jedem Prozessor p die Aufteilung in C_p und F_p im gesamten Teilgebiet Ω_p, also inklusive der Teilgebietsränder $\partial\Omega_p$ bestimmt.
Dabei ist jedoch das Auftreten von starken $F-F$-Kopplungen über den Teilgebietsrand hinweg nicht ausgeschlossen. Insbesondere können starke $F-F$-Kopplungen auftreten, bei denen die beteiligten Knoten nicht von einem gemeinsamen Grobgitterpunkt interpolieren können. Abhilfe schafft hier die Anwendung der zweiten Phase des klassischen Ruge-Stüben-Vergröberungsalgorithmus (Programm 2.10) , dieses mal aber angewendet auf die erweiterten Teilgebietsränder $\overline{\partial\Omega}_p$. Diese setzen sich aus dem Teilgebietsrändern $\partial\Omega_p$ und die stark gekoppelten Punkte auf anderen Prozessoren zusammen,

$$\overline{\partial\Omega}_p := \partial\Omega_p \cup \left(\cup_{i\in\Omega_p} S_i \setminus \Omega_p\right).$$

Die Menge der in dieser *RS3-Phase (Ruge-Stüben-Algorithmus, 3. Phase)* hinzugefügten C-Punkte bezeichnen wir mit $\tilde{C}_p$.
Die Prozessoren p und q können dabei zu unterschiedlichen C/F-Zuordnungen für die Punkte $i \in \overline{\partial\Omega}_p \cap \overline{\partial\Omega}_q$ kommen. Um hier zu einer einheitlichen Lösung zu kommen, können wir eine der drei folgenden Strategien anwenden.

- Wir akzeptieren jeden Punkt, der von einem der beteiligten Prozessoren ausgewählt worden ist, als C-Punkt, d.h. $i \in F_p \cap \partial\Omega_p$ wird zu einem Grobgitterpunkt genau dann, wenn

$$\exists q \in \{1,\ldots,P\} : \; i \in \tilde{C}_q$$

gilt. Dies führt zu vielen C-Punkten entlang der Teilgebietsränder, weil für die Beseitigung einer $F-F$-Kopplung *beide* beteiligten Punkte in das Grobgitter aufgenommen werden können.

- Wir nehmen als C-Punkte nur solche, die von allen beteiligten Prozessoren ausgewählt worden sind, d.h. $i \in F_p \cap \partial\Omega_p$ wird in C_p übernommen, genau dann, wenn

$$\forall q \text{ mit } i \in \overline{\partial\Omega}_q \text{ gilt: } \; i \in \tilde{C}_q$$

  Dies führt im Allgemeinen aber nicht zur Auflösung aller $F-F$-Kopplungen ohne gemeinsamen C-Punkt.

- Für den gemeinsamen Rand $\overline{\partial\Omega}_p \cap \overline{\partial\Omega}_q$ wird die Auswahl $\tilde{C}_p$ für den niedrigsten nummerierten Prozessor $q < p$ übernommen. Der Punkt $i \in F_p \cap \partial\Omega_p$ wird damit zu einem Grobgitterpunkt genau dann, wenn

$$i \in \tilde{C}_q, \text{ wobei } q = \min_{i \in \overline{\partial\Omega}_{q^*}} q^*$$

  gilt. Hier kann dieselbe Problematik wie bei der vorherigen Methode auftreten.

In der Praxis hat sich hierbei eine Kombination aus der ersten und der dritten Methode bewährt ([HY01], Abschnitt 4.2): Jeder Prozessor p übernimmt die von niedriger nummerierten Prozessoren $q < p$ ausgewählten Grobgitterpunkte $\tilde{C}_q \cap \Omega_q$ und fügt seine eigene Auswahl $\tilde{C}_p$ hinzu, d.h. $i \in F_p$ wird zu einem Grobgitterpunkt genau dann, wenn

$$\exists q \leq p : i \in \tilde{C}_q$$

gilt. Dies kann zwar zu einem Lastungleichgewicht führen (höher nummerierte Prozesse verwalten mehr Randpunkte), aber es werden weniger Punkte als mit der ersten Methode ausgewählt, während die Bedingung C1 erfüllt bleibt.
Wie bei den Subdomain Blocking-Verfahren gilt auch hier, dass ohne eine Agglomeration von Prozessorteilgebieten die Anzahl der Grobgitterpunkte auf dem gröbsten Level nicht beliebig klein werden kann. In Abbildung 4.4 (d) sind die von der RS3-Phase eingefügten Punkte entlang der Teilgebietsgrenzen des rechten oberen Quadranten zu erkennen. In diesem Fall wurde dort jeder Feingitterpunkt zu einem Grobgitterpunkt umgewandelt, um die in Abbildung 4.4 (a) erkennbaren $F-F$-Kopplungen zu beseitigen.

Programm 4.4 *AmgRS3*$(\Omega_p, A_p, C_p, F_p)$

```
begin
  AmgPhaseII(∂Ω̄_p, A_p, S_(p), C, F);
  for q > p do
      send C̃_p ∩ Ω_q to processor q;                    communication
      for q < p do
          receive C̃_q ∩ Ω_p from processor q;           communication
          C_p ← C̃_q ∩ Ω_p;
      od;
  end
```

4.3.3 CLJP-Algorithmus

Einen nicht auf den klassischen Ruge-Stüben-Algorithmus basierenden Ansatz zur parallelen Erstellung des Grobgitters stellt der in [HY01], Abschnitt 4.1 vorgestellte CLJP-Algorithmus dar. Er wurde von A. J. Cleary auf Basis der Graphpartitionierungsmethoden von M. Luby [Lub86], M. T. Jones und P. E. Plassman [JP93] entwickelt.
Der CLJP-Algorithmus erzeugt nicht, wie der klassische Ruge-Stüben-Algorithmus, eine unabhängige Menge $C \subset \Omega$. Stattdessen wird in mehreren Durchläufen jeweils eine unabhängige Menge $D \subset \Omega$ im durch den Operator induzierten Graphen A gewählt. Die Vereinigung dieser Mengen D bildet dann das Grobgitter C und ist in der Regel nicht mehr unabhängig.
Zunächst definieren wir die *Einflussmatrix* $S = (S_{ij})_{i,j=1}^{N}$ mit den Komponenten S_{ij},

$$S_{ij} := \begin{cases} 1 & \text{falls } j \in S_i, \\ 0 & \text{sonst.} \end{cases} \tag{4.2}$$

Die i-te Zeile enthält somit alle Punkte, von denen i stark abhängt, während die i-te Spalte alle Punkte enthält, die von i stark abhängen. Diese Matrix definiert einen gerichteten Graphen mit der Knotenmenge Ω.
Jedem Knoten wird —analog zum klassischem Ruge-Stüben-Algorithmus— ein Gewicht w_i zugeordnet. Wir initialisieren dieses Gewicht mit $|S_i^T|$. Dieses Gewicht repräsentiert den Wert von i als Grobgitterpunkt. Punkte mit hohem Gewicht sollen in das Grobgitter aufgenommen werden, Punkte mit Gewicht 0 werden zu Feingitterpunkten. Wir müssen zwischen benachbarten Punkten mit gleichem eingehenden Kantengewicht eine Rangfolge herstellen. Deshalb addieren wir noch auf jedes Gewicht eine Zufallszahl $\sigma_i \in [0, 1)$, d.h. $w_i \leftarrow |S_i^T| + \sigma_i$.
Wir konstruieren jetzt eine unabhängige Menge D, deren Elemente im späte-

ren Verlauf des Algorithmus zu der Menge der Grobgitterpunkte C hinzugefügt werden. Hierzu fügen wir einen Punkt i zu D hinzu, wenn sein Gewicht w_i größer als die Gewichte w_j aller seiner Nachbarn $j \in S_i \cup S_i^T$ ist. Nach Erstellung dieser Menge D werden die Gewichte der anderen Punkte anhand der nachfolgenden Heuristiken modifiziert.

H1 Da C-Punkte nicht interpoliert werden, vermindern wir das Gewicht eines Punktes j, der einen D-Punkt i beeinflusst, um 1 und entfernen die Kante S_{ij} aus dem Graphen.

H2 Wenn sowohl k als auch j von einem D-Punkt i abhängen ($i \in S_j \cap S_k$) und k von j abhängt ($j \in S_k$), dann vermindern wir das Gewicht von j um 1 und entfernen die Kante S_{kj} aus dem Graphen, da der Wert an k direkt von dem an i interpoliert werden kann.

Wenn diese Heuristiken für alle Punkte durchgeführt worden sind, werden alle ein- und ausgehenden Kanten von D-Punkten aus dem Graphen entfernt und die Punkte in D zu C hinzugefügt. Punkte, deren Gewicht w_i unter 1 fällt, benötigen wir nach den obigen Heuristiken nicht zur Interpolation und werden zu der Menge der Feingitterpunkte hinzugefügt.
Das Verfahren wird dann mit einer neuen Wahl von D fortgesetzt, solange es noch unbestimmte Punkte gibt. Das gesamte Verfahren ist in Programm 4.3.3 dargestellt. Abbildung 4.3 zeigt das Gitter einer 9-Punkte-Diskretisierung jeweils nach Anwendung der ersten und der zweiten Heuristik für alle Punkte. Wir sehen, dass nach der Anwendung der zweiten Heuristik nicht alle zu D-Punkten benachbarten Punkte in die Menge F der Feingitterpunkte aufgenommen sind. Dies führt dazu, dass im nächsten Iterationsschritt auch direkt zu C-Punkten benachbarte Punkte in die Menge D aufgenommen werden können.
Die parallele Durchführung dieses Verfahrens wirft ein Problem auf: Bei der Anwendung der Heuristik H2 braucht der Prozess, der den Punkt i verwaltet, die Kenntnis über den Vektor S_j^T für alle $j \in S_i^T$. Weil die Vektoren sich im Laufe der Iteration ändern, reicht es nicht aus, sie einmal am Anfang zu übertragen.
In unserer Implementierung haben wir einen modifizierten Algorithmus verwendet, der von S^T nur die lokalen *Spalten* benötigt, welche den lokalen Zeilen von S entsprechen. Der Algorithmus *AmgStrongCouplings* (Programm 2.3) stellt diese Strukturen ohne Kommunikation bereit (siehe Abschnitt 4.1). Zusätzlich speichern wir für jeden nicht-D-Punkt, von welchem D-Punkt er beeinflusst wird. Weil diese Marke m_i pro Gitterpunkt nur einen Wert gleichzeitig erfasst, müssen unter Umständen einige D-Punkte bis zum nächsten

Programm 4.5 CLJP-Algorithmus $CLJP(\Omega, S, S^T, C, F)$

```
begin
  C ← ∅;
  F ← ∅;
  for i ∈ Ω do
      w(i) = |S_i^T| + σ(i);
  od;
  while C ∪ F ≠ Ω do
        D ← ∅
        for i ∈ Ω do
            if w(i) > w(j) for all j ∈ S_i ∪ S_i^T
              then
                    D ← D ∪ {i};
            fi;
        od;
        for i ∈ D do
            for j ∈ S_i do                                  Heuristik H1
                w(j) ← w(j) − 1;
                S_i ← S_i\{j};
            od;
            for j ∈ S_i^T do                                Heuristik H2
                for k ∈ S_j^T do
                    if k ∈ S_i^T
                      then
                            w(j) ← w(j) − 1;
                            S_j^T ← S_j^T\{k};
                    fi;
                od;
                S_i^T ← S_i^T\{j};
            od;
        od;
        for i ∈ Ω do
            if w(i) < 1 then F ← F ∪ {i} fi;
        od;
        C ← C ∪ D;
  od;
end;
```

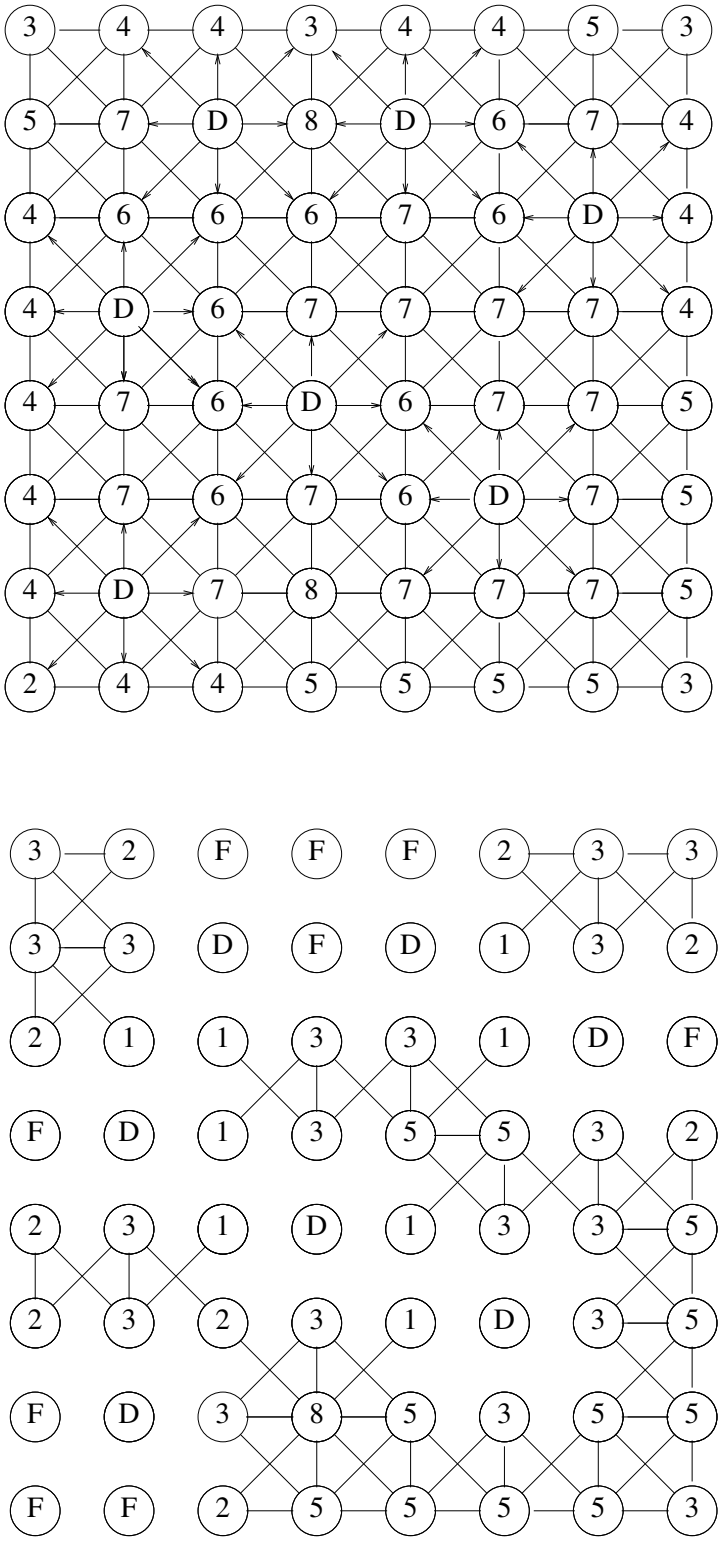


Abbildung 4.3: Gitter eines 9-Punkte Sternes nach Anwendung der ersten (oben) und zweiten (unten) Heuristik des CLJP-Verfahrens. Die Zahlen geben die Gewichte w_i (ohne die Zufallszahl) an. Die Pfeile zeigen die Richtung des Einflusses an, Linien ohne Pfeil stellen Einflüsse in beiden Richtungen dar.

Iterationsschritt wieder aus dieser Menge entfernt werden. Am Gesamtergebnis ändert dies jedoch nichts, weil ein solcher D-Punkt j niemals direkt mit einem anderen D-Punkt verbunden ist und dessen Wert w_j im Laufe dieses Iterationsschrittes damit auch nicht geändert werden kann. Die Werte seiner Nachbarn werden nur erniedrigt, so dass dieser Punkt zu Anfang der nächsten Iteration wieder in die Menge D aufgenommen wird.
Der modifizierte Algorithmus erfordert nur die Kommunikation der Gewichte w_i, der D-Punkte und der Marken m_i (für nicht D-Punkte). Der Ablauf ist in Programm 4.3.3 beschrieben.
Dieses Verfahren kann vollständig parallel durchgeführt werden. Es produziert außerdem unabhängig von der Anzahl der Prozessoren und der konkreten Gebietszerlegung dasselbe C/F-Splitting (unter der Voraussetzung, dass jedem Punkt immer dieselbe Zufallszahl zugeordnet wird). Außerdem ist es möglich, auch auf verteilten Gebieten bis auf einen einzigen Punkt zu vergröbern. Da jedoch nicht jeder Nachbar eines Grobgitterpunktes sofort zu einem Feingitterpunkt wird (wie beim klassischen Algorithmus), kann dieser in einem späteren Iterationsschritt noch zu einem Grobgitterpunkt werden. Damit erzeugt dieses Verfahren viel mehr Grobgitterpunkte als erforderlich, wie in Abbildung 4.5 (a) zu sehen ist.

4.3.4 Falgout-Vergröberung

Dieser in [HY01], Abschnitt 4.3 vorgestellte Algorithmus setzt wieder auf den klassischen Ruge-Stüben-Vergröberungsalgorithmus (Abschnitt 2.7) auf. Wir benutzen jetzt zur Randbehandlung den CLJP-Algorithmus.
Zunächst führt jeder Prozessor p auf seinen lokalen Punkten Ω_p das klassische Ruge-Stüben-Vergröberungsverfahren durch. Die dadurch erhaltenen C-Punkte im Teilgebiets*innern* $\overset{\circ}{\Omega}_p$ dienen dann als Startwerte (D-Punkte) für den CLJP-Algorithmus, mit dem die Teilgebietsrandpunkte eingeteilt werden
Mit dieser Wahl der D-Punkte wird das Verfahren im Teilgebietsinneren ähnliche Ergebnisse wie das klassische Verfahren produzieren, weil die nur mit inneren Punkten inzidenten Kanten im ersten CLJP-Durchlauf größtenteils entfernt und die Gewichte der beteiligten Punkte unter 1 fallen werden. Die Ränder werden jedoch vollständig vom CLJP-Algorithmus vergröbert, d.h. die Auswahl der D-Punkte erfolgt hier wie in Abschnitt 4.3.3 beschrieben. Dies beschränkt die große Zahl der durch den CLJP-Algorithmus verursachten $C-C$-Kopplungen auf die Randbereiche. Auf höheren Leveln, wenn (fast) jeder Punkt eine starke Verbindung zu einem Punkt auf einem anderen Prozessor hat, bedeutet dies nur noch eine Anwendung des CLJP-Algorithmus.

Programm 4.6 Paralleler CLJP-Algorithmus

```
begin
  C_p ← ∅;
  U ← Ω;
  for i ∈ Ω_p ∪ ⋃_{i∈Ω_p} S_i do w_i = |S_i^T| + σ(i); od;
  while U ≠ ∅ do
        D ← U;
        for i ∈ D ∩ (Ω_p ∪ ⋃_{i∈Ω_p} S_i) do
            for j ∈ S_i^T do
                if w_i > w_j then D ← D \ {j};
                elsif w_i < w_j then D ← D \ {i}; fi;
            od;
        od;
        update D;                                   communication step 1
        for i ∈ D ∩ (Ω_p ∪ ⋃_{i∈Ω_p} S_i) do
            for j ∈ S_i^T do if m_j ≠ 0
                              then j already influenced by another D-point
                                   D ← D \ {i};
                                   for k ∈ S_i^T, m_k = i do m_k = 0; od;
                               else m_j = i;              mark influence of i
                fi;
            od;
        od;
        update D and m;                             communication step 2
        for j ∈ ((Ω_p ∪ ⋃_{i∈Ω_p} S_i) ∩ U) \ D do
            for k ∈ S_j^T do
                if k ∈ D
                  then                                  Heuristik H1
                       w_j ← w_j − 1;
                       S_j^T ← S_j^T \ {k};
                elsif m_k = m_j
                     then                               Heuristik H2
                          w_j ← w_j − 1;
                          S_j^T ← S_j^T \ {k};
                fi;
            od;
        od;
        for i ∈ Ω_p ∩ D do S_i^T ← ∅; od;
        update w;                                   communication step 3
        C ← C ∪ D;
        for i ∈ Ω do if w_i < 1 then F ← F ∪ {i} fi; od;
        U ← U \ (C ∪ F);
  od;
end
```

Damit lässt sich auch wieder eine kleine Punkteanzahl des gröbsten Gitters erreichen. Der Kommunikationsaufwand ist jedoch pro Iterationsschritt nicht geringer als derjenige des CLJP-Algorithmus.

Abbildung 4.5 (b) zeigt die Anwendung des Falgout-Algorithmus auf den

Programm 4.7 *AmgFalgout*$(\Omega_p, A_p, S_{(p)}, S_{(p)}^T, C_p, F_p)$

begin
  AmgPhaseI$(\Omega_p, S_{(p)}, S_{(p)}^T, C_p, F_P)$;
  AmgPhaseII$(\Omega_p, A_p, S_{(p)}, C_p, F_p)$;
  $D \leftarrow \emptyset$;
  for $i \in C_p$ **do**
    if $N_i = \emptyset$ **then** $D \leftarrow D \cup \{i\}$;
    od;
    $C_p \leftarrow \emptyset$;
    $F_p \leftarrow \emptyset$;
    AmgCLJP$(\Omega_p, S_p, S_p^T, C_p, F_p)$;
  end

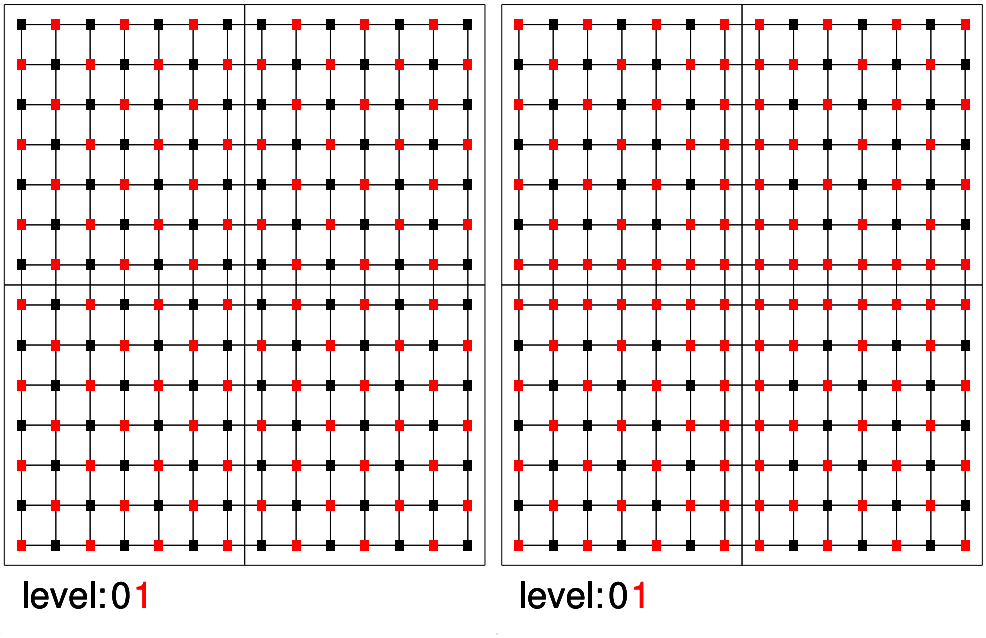
zweidimensionalen Laplaceoperator. Wir sehen in den Randbereichen des rechten oberen Quadranten eine Häufung der Grobgitterpunkte.

Wir fassen abschließend in Tabelle 4.1 die Vorteile und Nachteile des Mini-

	Minimum Subdomain Blocking	RS3	CLJP	Falgout
Vorteile	Keine Kommunikation zur Vergröberung notwendig	Nur zur Interpolation wirklich benötigte Punkte werden zusätzlich eingefügt	parallele Grobgitterwahl unabhängig von der Gebietszerlegung	Beschleunigung des CLJP-Verfahrens, weniger Grobgitterpunkte im Teilgebietsinnern
Nachteile	keine Kontrolle der starken $F-F$-Kopplungen	Übertragung von Teilen der Grobgittermatrix erforderlich, in ungünstigen Situationen Auswahl von vielen Randpunkten erforderlich	sehr großzügige Auswahl des Grobgitters, mehrere Kommunikationsschritte im Laufe des Verfahrens	fügt am Rand mehr Grobgitterpunkte ein als nötig, erfordert mehrere Kommunikationsschritte

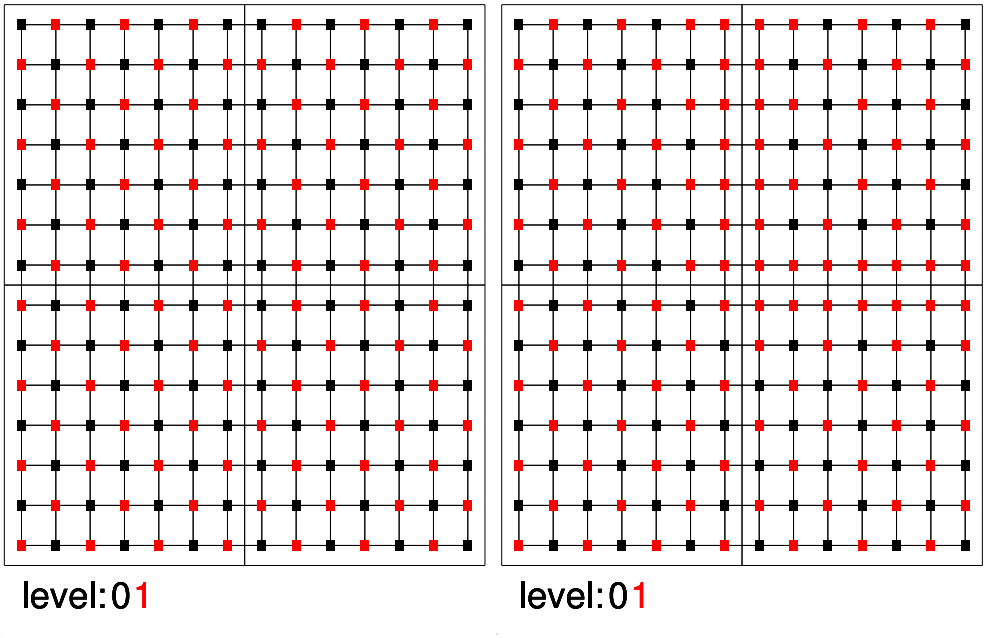
Tabelle 4.1: Vorteile und Nachteile der Vergröberungsalgorithmen

mum Subdomain Blockings, des RS3- des CLJP- und des Falgout-Verfahrens zusammen. Mit diesem Wissen wollen wir einen Algorithmus konstruieren, welches die Nachteile der Verfahren vermeidet, die Vorteile jedoch kombiniert.



(a) Keine Randbehandlung

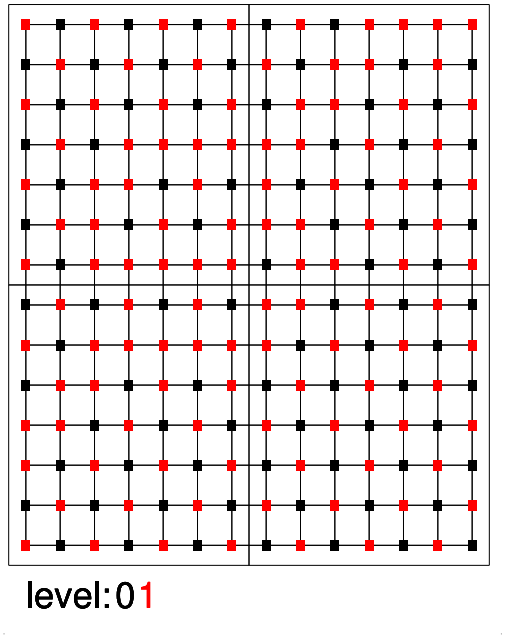
(b) Full subdomain blocking



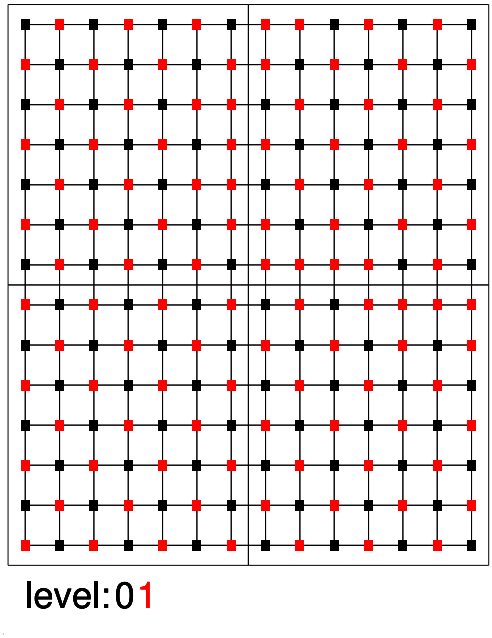
(c) Minimum subdomain blocking

(d) RS3-Algorithmus

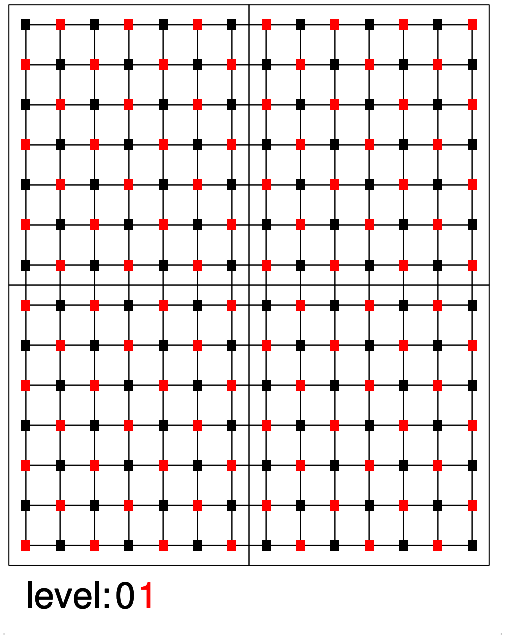
Abbildung 4.4: Anwendung der verschiedenen Vergröberungsstrategien auf die 5-Punkte-Diskretisierung des Laplaceoperators.



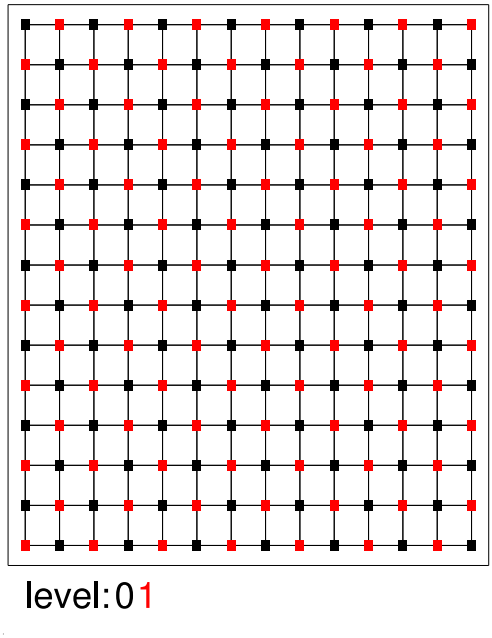
(a) CLJP-Algorithmus



(b) Falgout-Algorithmus



(c) CGC-Algorithmus



(d) sequentielle Vergröberung

Abbildung 4.5: Anwendung der verschiedenen Vergröberungsstrategien auf die 5-Punkte-Diskretisierung des Laplaceoperators.

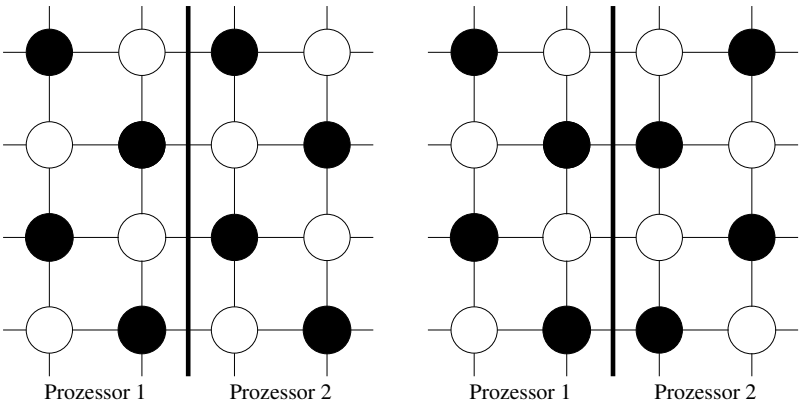


Abbildung 4.6: Zwei mögliche Grobgitterkonstellationen für einen 5-Punkte Stern auf zwei Prozessoren. Die schwarz markierten Punkte sind die Grobgitterpunkte

4.4 Eine neue parallele Vergröberungsstrategie: Coarse-Grid Classification (CGC)

Wir haben im vorangehenden Abschnitt verschiedene Methoden zur parallelen Grobgitterwahl diskutiert. Dabei stellte sich heraus, dass das RS3-Verfahren Grobgitter produziert, die eine stabile Interpolation ermöglichen, jedoch steigt der Speicher- und Rechenaufwand an, wenn entlang eines Teilgebietsrandes viele $F-F$-Kopplungen untersucht und gegebenenfalls eliminiert werden müssen.
Hierdurch motiviert ist es nun unser Ziel, auf jedem Prozessorteilgebiet ein solches Gitter zu erzeugen, dass möglichst ohne Randbehandlung zu einem Gitter auf einem anderen Teilgebiet „passt“, d.h. ein Gitter, welches keine oder nur sehr wenige $F-F$-Kopplungen über den Teilgebietsrand hinweg aufweist. Wie wir bereits in Abschnitt 4.2 gesehen haben, erfordert die stabile Behandlung einer starken $F-F$-Kopplung die Standard-Interpolationsformel über den Teilgebietsrand hinweg, während zusätzlich garantiert sein muss, dass die beiden Feingitterpunkte eine starke Kopplung zu einem gemeinsamen Grobgitterpunkt aufweisen.
Als Motivation betrachten wir die möglichen Vergröberungen bei der Benutzung eines 5-Punkte-Diskretisierungssterns. Wir sehen in Abbildung 4.6 zwei verschiedene Grobgitterkonstellationen am Teilgebietsrand zweier Prozessoren. Die zweite Variante weist entlang des Teilgebietsrandes $C-C$-Kopplungen und $F-F$-Kopplungen auf, die erste nicht. Wir wollen nun ein Verfahren entwickeln, das für jeden Prozessor ein geeignetes Grobgitter auswählt. Dabei soll ein Prozessor aber nicht auf einen anderen warten, bis

die Aufteilung der Randpunkte bekannt ist, sondern alle Prozessoren sollen gleichzeitig ihre Grobgitterwahl treffen. Dazu wählt jeder Prozessor mehrere potenzielle Grobgitter aus und wählt dann —in Abhängigkeit von den benachbarten Prozessoren— eines davon als Grobgitter aus.
Wir nutzen bei dieser Vorgehensweise die Tatsache aus, dass der Ruge-Stüben-Vergröberungsalgorithmus ein Grobgitter in Abhängigkeit eines Startpunktes bestimmt. Unterschiedliche Wahlen für den Startpunkt können zu verschiedenen Grobgittern führen, wie es in Abbildung 4.4 für einen 9-Punkte-Stern dargestellt ist. Wir sehen, dass das Ergebnis der letzten Grobgitterwahl mit dem der ersten Wahl identisch ist, nur die Reihenfolge der Punktauswahl war unterschiedlich. Bei insgesamt 25 verschiedenen Möglichkeiten, den Startpunkt zu wählen (alle inneren Punkte haben ja dasselbe eingehende Kantengewicht) , ergeben sich hier nur 4 verschiedene Gitter. Wir sprechen deshalb von Klassen von Gittern.
Wir wählen diese Gitter aus, indem wir den klassischen Ruge-Stüben-Algorithmus mehrfach auf das feine Gitter ansetzen, jeweils mit einem anderen Startpunkt. In der Regel reichen bereits vier Grobgitter auf jedem Prozessorteilgebiet aus, um eine gute Auswahl zu ermöglichen. Sobald keine Punkte mehr zur Verfügung stehen, die das maximale eingehende Kantengewicht aufweisen, wird das Verfahren ebenfalls beendet. Wir haben damit für jeden Prozessor p jeweils ng_p potenzielle Grobgitter $\{C_{(p),i}\}_{i=1}^{ng_p}$ erzeugt. Aufgabe ist es, für jeden Prozessor eines auszuwählen und daraus ein Grobgitter über alle Prozessoren zu erzeugen.
Wir bezeichnen im folgenden mit $\mathcal{S}_p$ die Menge der Nachbargebiete von Ω_p,

$$\mathcal{S}_p := \{q \neq p : \ \exists i \in \Omega_p, \ j \in \Omega_q : \ j \in S_i\}.$$

Wenn einmal ein Grobgitter auf einem Prozessor p ausgewählt ist, dann soll auf den benachbarten Prozessoren $q \in \mathcal{S}_p$ jeweils ein Gitter ausgewählt werden, welches möglichst wenig $F-F$-Kopplungen über den Teilgebietsrand hinweg erzeugt.
Wir stellen zur definitiven Auswahl der Grobgitter einen gerichteten, gewichteten Graphen $G = (V, E)$ auf, dessen Knotenmenge aus der Menge aller potenzieller Grobgitter C_i besteht,

$$\begin{aligned} V_p &:= \{C_{(p),i}\}_{i=1,\ldots,ng_p}, \\ V &:= \cup_{p=1}^{P} V_p. \end{aligned}$$

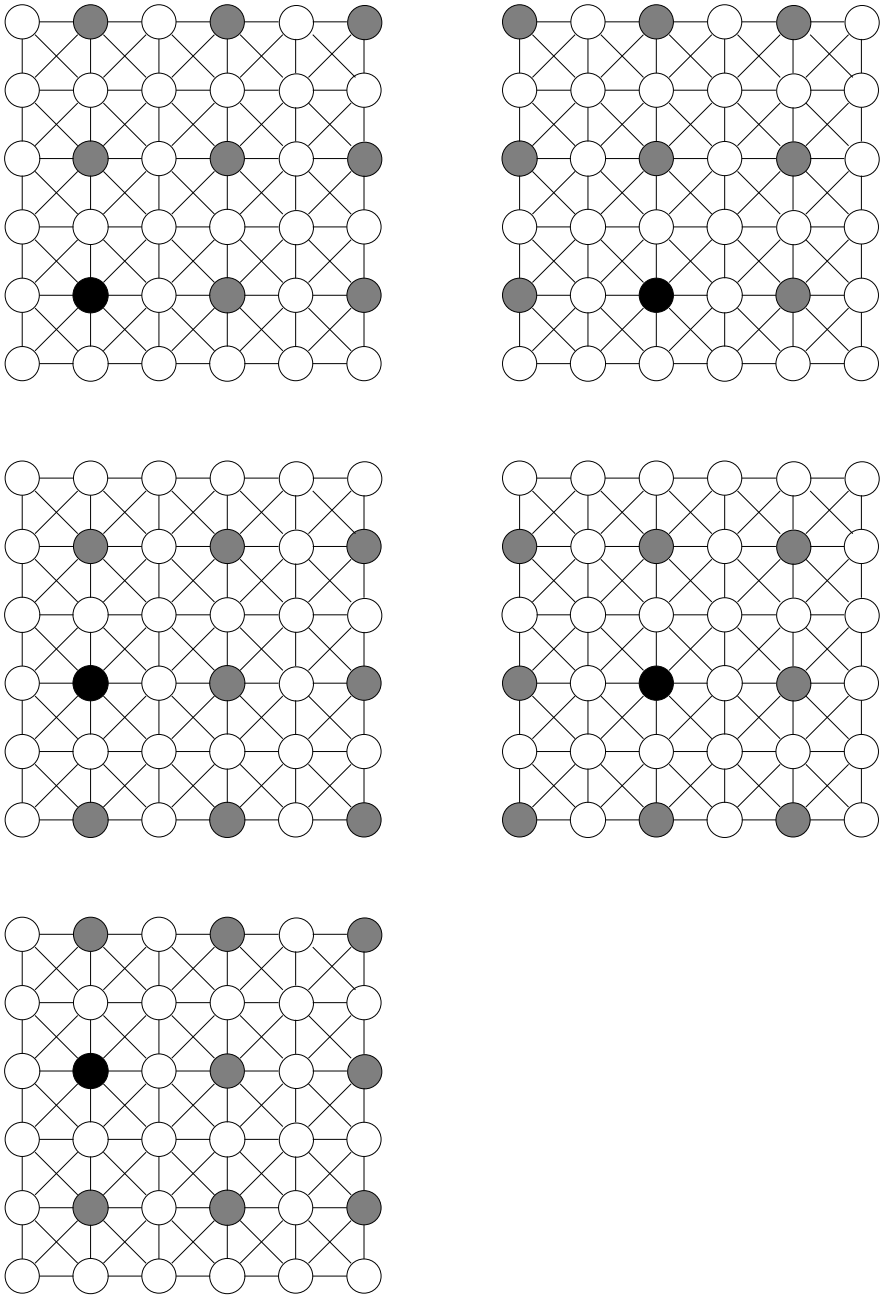


Abbildung 4.7: Fünf mögliche Grobgitter für einen 9-Punkte-Stern. Die grauen Punkte sind die Grobgitterpunkte, der schwarze Punkt ist der jeweils zuerst gewählte Grobgitterpunkt.

Programm 4.8 CGC-Algorithmus $CGC(S, S^T, ng, \{C_i\}_{i=1,\ldots,ng}, \{F_i\}_{i=1,\ldots,ng})$

```
for j ← 1 to |Ω| do
    λ_j ← |S_j^T|;
od;
C_0 ← ∅;
λ ← arg max_{k∈Ω} λ_k;
do
    U ← Ω \ ⋃_{i≤it} C_i;
    if max_{k∈U} λ_k < λ then break; fi;
    it ← it + 1;
    F_it ← ∅;
    C_it ← ∅;
    do
        j ← arg max_{k∈U} λ_k;
        if λ_j = 0 then break; fi;
        C_it ← C_it ∪ {j};
        λ_j ← 0;
        for k ∈ S_j^T ∩ U do
            F_it ← F_it ∪ {k};
            λ_k ← 0;
            for l ∈ S_k ∩ U do
                λ_l ← λ_l + 1;
            od;
        od;
        for k ∈ S_j ∩ U do
            λ_k ← λ_k − 1;
        od;
    od;
od
ng ← it;
```

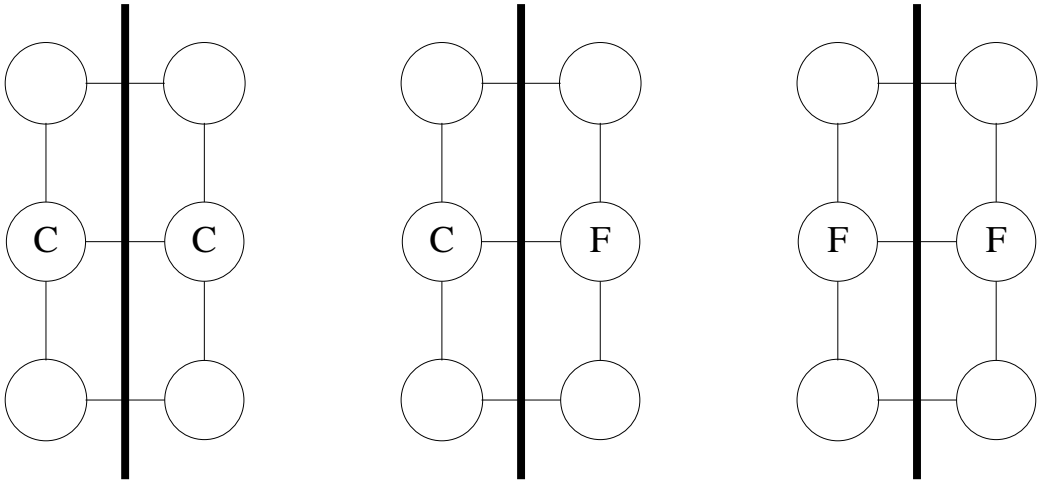


Abbildung 4.8: Drei mögliche C/F-Konstellationen entlang einer Teilgebietsgrenze.

Die Kantenmenge E besteht aus allen Paaren (v, u) mit $v \in V_p$ und $u \in V_q$, wobei $q \in \mathcal{S}_p$ ein zu p benachbarter Prozessor ist, .

$$\begin{aligned} E_p &:= \{\cup_{q\in\mathcal{S}_p} \cup_{v\in V_p,\ u\in V_q} (v,u)\}, \\ E &:= \cup_{p=1}^{P} E_p. \end{aligned}$$

Wir betrachten zur Bestimmung der Kantengewichte γ für $q \in \mathcal{S}_p$ die Knoten $v \in V_p$ und $u \in V_q$, die jeweils ein C/F-Splitting der Form (C_p, F_p) beziehungsweise (C_q, F_q) auf ihrem Prozessor darstellen. Zusammen erzeugen diese Knoten ein C/F-Splitting auf dem Gebiet $\Omega_p \cup \Omega_q$. Am gemeinsamen Teilgebietsrand können dabei drei Situationen auftreten, die in Abbildung 4.4 dargestellt worden sind. Für jede solche Situation bestimmen wir ein Gewicht γ_{CC}, γ_{CF} oder γ_{FF}, welches auf das Kantengewicht $\gamma(e)$ der Kante $e = (v, u)$ addiert wird.
Wichtig ist vor allem die in der dritten Abbildung dargestellte Situation. In diesem Fall treten zwei stark gekoppelte Feingitterpunkte $i \in F_p$ und $j \in F_q$ auf. Dies führt zu zwei Problemen: Erstens ist es möglich, dass diese beiden Punkte nicht von einem gemeinsamen Grobgitterpunkt her interpolieren, so dass C1 nicht erfüllt ist. Andererseits ist für die Aufstellung einer stabilen (Standard-)Interpolation der beteiligten Punkte —selbst bei Erfüllung der Forderung C1— die gegenseitige Übertragung der Matrixzeilen A_i und A_j erforderlich.
Wir setzen deshalb für diese Situation den Summand $\gamma_{F,F}$ auf einen betragsmäßig großen, negativen Wert. In der Praxis hat sich die Addition eines

Wertes von $\gamma_{F,F} = -8$ als praktikabel herausgestellt.
Die in der ersten Abbildung dargestellte Situation sollte ebenfalls vermieden werden, weil sie die Operator- und Gitterkomplexität erhöhen kann. Sie ist allerdings nicht so gravierend wie das vorhin beschriebene Problem, wir setzen deshalb $\gamma_{C,C} = -1$.
Tritt die in der mittleren Abbildung gezeigte Situation auf, so erhöhen wir das Kantengewicht $\gamma(e)$ um den Wert $\gamma_{C,F} = 1$.
Wir haben jetzt einen gerichteten, gewichteten Graphen mit den potenziellen Grobgittern als Knoten erzeugt. Abbildung 4.9 zeigt am Beispiel eines 5-Punkte-Sternes in zwei Raumdimensionen den sich ergebenden Graphen mit den zugehörigen Kantengewichten. Mit Hilfe dieses Graphens können wir nun geeignete Gitter auswählen. Die Knotenmenge dieses Graphens hat eine Mächtigkeit in Größenordnung zu der Anzahl der beteiligten Prozessoren P, ist also viel kleiner als die Dimension des zu lösenden Gleichungssystems N. Da nur zwischen benachbarten Prozessoren Kanten erzeugt worden sind, ist die Mächtigkeit der Kantenmenge E ebenfalls gering. Damit können wir den gesamten Graphen mit geringem Aufwand auf einen Prozessor transferieren .
Auf diesem Prozessor wählen wir jetzt aus jeder Teilmenge $V_p \subset V$ genau einen Knoten v aus. Dabei bezeichne $\delta^+(v)$ die Menge der von $v \in V$ ausgehenden Kanten und $\Gamma(v)$ die Menge der zu $v \in V$ adjazenten Knoten

$$\begin{aligned} \delta^+(v) &:= \{e \in E : \ e = (v, u) \text{ für ein } u \in V\} \\ \Gamma(v) &:= \{u \in V : \ \exists e = (v, u) \in E\} \end{aligned}$$

Wir gehen wie folgt vor:

1 Zunächst weisen wir jedem Knoten v ein Gewicht $w(v)$ zu, welches wir mit $\sum_{e \in \delta^+(v)} \gamma(e)$ initialisieren.

2 Wir wählen den Knoten v mit dem höchsten Gewicht aus, nehmen ihn in $\mathcal{C}$ auf und entfernen die Knotenmenge V_p für die $v \in V_p$ ist, aus dem Graphen.

3 Wir betrachten jetzt diejenige Kante $e \in \delta^+(v)$, die das höchste Gewicht $\gamma(e)$ aufweist. Den Endpunkt dieser Kante u nehmen wir in $\mathcal{C}$ auf und wir entfernen u aus der entsprechenden Menge V_q. Gibt es mehrere Kanten mit maximalem Gewicht, so wählen wir denjenigen Knoten u aus, der das höchste Gewicht $w(u)$ aufweist. Wir wiederholen diesen Schritt für den Knoten u. Stehen keine Kanten mehr zur Verfügung, so kehren wir zu Schritt 2 zurück. Wir beenden das Verfahren, sobald aus jeder Menge V_p ein Knoten v ausgewählt worden ist.

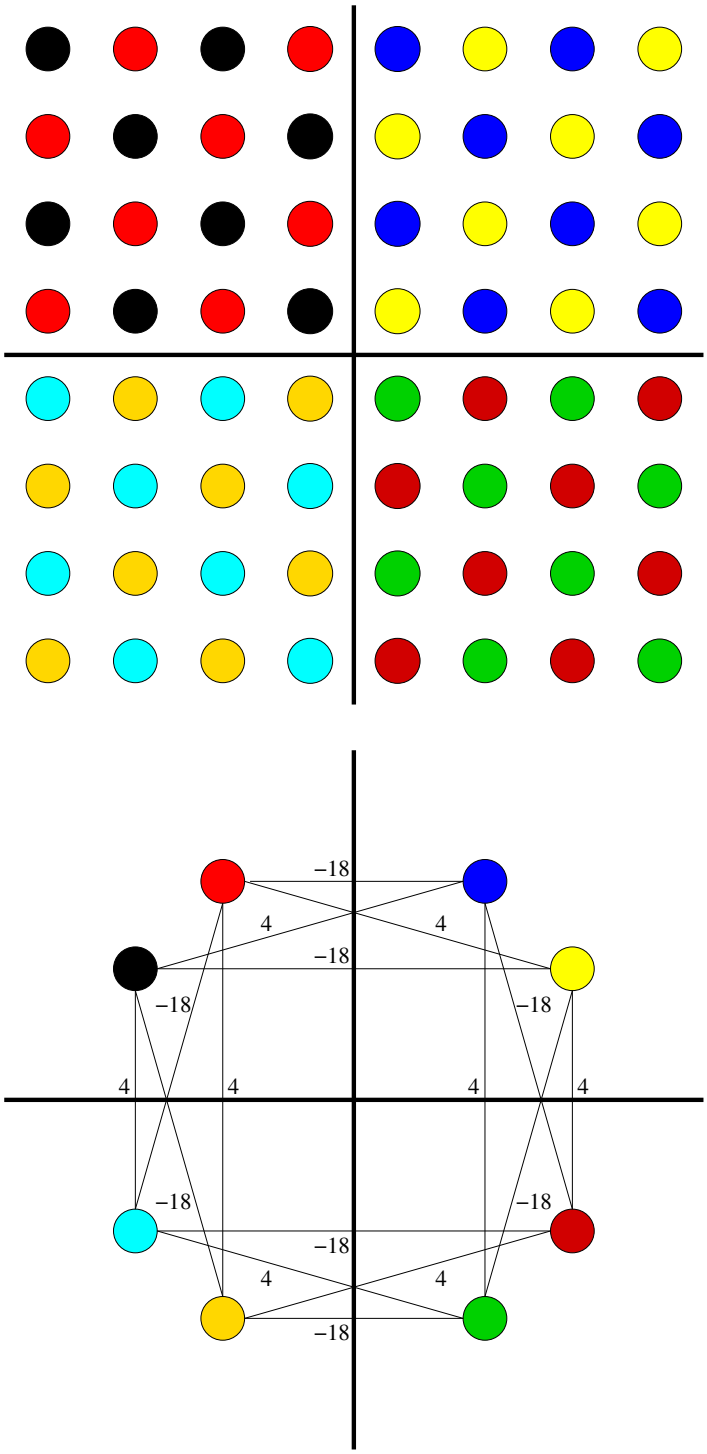


Abbildung 4.9: Anwendung des CGC-Algorithmus auf die 5-Punkte-Diskretisierung des Laplaceoperators, auf 4 Prozessoren verteilt. Die obere Abbildung zeigt die Zuordnung zu den einzelnen Gittern, die untere Abbildung zeigt den gewichteten Graphen

Programm 4.9 *AmgCGCChoose*$(V, E, \mathcal{C})$

begin
  $\mathcal{C} \leftarrow \emptyset$;
  for $v \in V$ **do** $w(v) \leftarrow \sum_{e \in \delta^+(v)} \gamma(e)$; **od**;
  while $V \neq \emptyset$ **do**
    $v \leftarrow \arg\max_{u \in V} w(u)$;
    mark :
    $p \leftarrow \arg\max_{q=1,\ldots,P} |\Omega_q \cap \{v\}|$;
    $v_p \leftarrow v$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{v_p\}$;
    $V \leftarrow V \setminus V_p$;
    $E \leftarrow E \setminus \{(u_1, u_2)\}_{u_1 \in V_p \vee u_2 \in V_p}$;
    $U_v \leftarrow \{\arg\max_{u \in \Gamma(v)} \gamma((v, u))\}$;
    if $U_v \neq \emptyset$
      then
        $v \leftarrow \arg\max_{u \in U_v} w(u)$;
        GOTO mark;
    fi;
  od;
end.

Der genaue Ablauf ist in Programm 4.4 dargestellt. Dieses Vorgehen benötigt P Iterationen, um für jeden Prozessor $p = 1, \ldots, P$ ein Gitter v auszuwählen. Schließlich transferieren wir an jeden Prozessor p den Punkt $v \in \mathcal{C} \cap V_p$.
Abbildung 4.5 (c) zeigt die Wirkung des CGC-Algorithmus: Das ausgewählte Gesamtgrobgitter hat dieselbe Struktur wie ein sequentiell erstelltes Grobgitter (Abbildung 4.5 (d)). Alle Prozessoren haben ein Grobgitter ausgewählt, welches Grobgitterpunkte am Teilgebietsrand aufweist. Es müssen keine weiteren Punkte eingefügt werden.
Sollten jetzt noch $F - F$-Kopplungen über den Gebietsrand hinweg übrig geblieben sein, so müssen wir sicherstellen, dass für die beteiligten Punkte eine Interpolation möglich ist. Dazu untersuchen wir lediglich, ob ein Feingitterpunkt $i \in F \cap \Omega_p$, der eine starke Kopplung zu einem nichtlokalem Feingitterpunkt $j \in F \setminus \Omega_p$ aufweist, auch mit einem Grobgitterpunkt $k \in C$ stark gekoppelt ist. Ist dies nicht der Fall, so fügen wir diesen Punkt i der Menge der Grobgitterpunkte hinzu. Damit ist sichergestellt, dass für diesen Punkt auf jeden Fall eine Interpolationsvorschrift gegeben ist. Hat ein Feingitterpunkt gar keine starke Kopplungen —weder innerhalb des Teilgebiets noch zu einem Punkt auf einem anderen Prozessor—, so bleibt er Feingitterpunkt, weil der Fehler an einem solchen Punkt durch Glättung reduziert werden kann. Dieses Verfahren erzielt nicht die strikte Einhaltung der Be-

Programm 4.10 $CGCCheck(\Omega, S, C, F)$

```
begin
  for i ∈ ∂Ω_p do
      if S_i ≠ ∅ ∧ S_i ∩ C = ∅
        then
              C ← C ∪ {i};
              F ← F \ {i};
      fi;
  od;
end
```

dingung C1. Wenn zwei Feingitterpunkte i und j über eine Teilgebietsgrenze hinweg stark gekoppelt sind und jeweils starke Kopplungen zu Grobgitterpunkten aufweisen, jedoch nicht von einem gemeinsamen Grobgitterpunkte interpolieren, so wird diese Situation nicht korrigiert.
Abhilfe schafft in solchen Fällen die Anwendung des RS3-Algorithmus anstatt des in Programm 4.10 beschriebenen Verfahrens. Da das CGC-Verfahren die starken $F - F$-Kopplungen über den Teilgebietsrand weitgehend vermeidet, ist der Aufwand für diesen Schritt geringer als der Aufwand bei einem klassischen RS3-Ansatz. In der Praxis hat sich jedoch gezeigt, dass auch der

einfachere Ansatz bei vielen Problemen zu einer guten Konvergenzrate führt.

4.5 Abschätzung des Kommunikationsaufwandes

Wir wollen jetzt den Kommunikationsaufwand für die Setupphase abschätzen. Zunächst betrachten wir die allen Parallelisierungsverfahren gemeinsame Kommunikation.
In jedem Verfahren —außer dem Full Subdomain Blocking— werden zunächst die Kopplungen S_i und S_i^T bestimmt. Wie wir in Abschnitt 4.1 gesehen haben, erfordert die Ermittlung der Startwerte λ_i für die Vergröberung 2.9 die Kommunikation von $|S_i^T|$ auf denjenigen Prozessor, der den Punkt i verwaltet.
Wir betrachten die Punkte j auf Prozessor p, für die $S_j \setminus \Omega_p \neq \emptyset$ gilt. Genau diese Punkte j werden in ein S_i mit $i \notin \Omega_p$ eingefügt. Damit muss der Prozessor p insgesamt

$$O(\#\{j \in \partial\Omega_p : \ S_j \setminus \Omega_p \neq \emptyset\})$$

Werte verschicken. Dieser Wert ist durch $O(|\partial\Omega_p|)$ beschränkt, wobei dieser Wert für isotrope Probleme, bei denen jede Kopplung $a_{ij} \neq 0$ stark ist, wie zum Beispiel einer 5- oder 9-Punkte Diskretisierung eines Laplaceoperators, auch angenommen wird.
Nach erfolgter Vergöberung müssen wir jedem Grobgitterpunkt i einen eindeutigen Index cg_i zuordnen, um die Interpolationsmatrix aufstellen zu können. Diese Indizes werden fortlaufend vergeben und sind genauso sortiert wie die Indizes des feinen Gitters. Der Startwert für die Indizierung auf Prozessor P lautet demnach $\sum_{q<p} |C_q|$, die Ermittlung dieses Wertes erfordert pro Prozessor —außer dem ersten— einen Kommunikationsaufwand von

$$O(1).$$

Für die Aufstellung der Interpolationsformel eines Feingitterpunktes $i \in F_p$, der starke Kopplungen zu Punkten $j \in S_i \setminus \Omega_p$ aufweist, müssen wir die Zugehörigkeit dieser Punkte j zu C oder F sowie deren Index cg_j im Grobgitter kennen. Wir erhalten somit einen Kommunikationsaufwand von

$$O\left(\left| \cup_{i \in F_p} (S_i \setminus \Omega_p)\right|\right), \tag{4.3}$$

um die Interpolationsformeln aller Feingitterpunkte $i \in F_p$ auf Prozessor p aufstellen zu können.

Wenn wir mit der Standardinterpolation (Abschnitt 2.6.2) über den Rand hinweg interpolieren, so vergrößert sich der Kommunikationsaufwand erheblich. Nach dem oben genannten Kommunikationsschritt müssen wir die zu $j \in S_i \cap F_q$, $q \neq p$, gehörenden Zeilen A_j des Operators übertragen. Bezeichne $|A_i|_{nz}$ die Anzahl der Nicht-Nulleinträge in der Zeile A_i, so erhalten wir für einen Kommunikationsaufwand von

$$O\left(\sum |A_j|_{nz} : \; j \in \bigcup_{i \in F_p,\; q \neq p} (S_i \cap F_q)\right)$$

Matrixeinträgen. Diese Zeilen enthalten wiederum auch starke Kopplungen zu Punkten $j \in \Omega_q,\; j \notin \bigcup_{i \in F_p} S_i$. Für diese Punkte müssen wir wiederum die Zugehörigkeit zu F oder C und den entsprechenden Index cg_j auf Prozessor p übertragen, also nochmal einen Aufwand von

$$O\left(\left| \bigcup_{i \in F_p,\; q \neq p} \; \bigcup_{j \in S_i \cap F_q} S_j \setminus (\Omega_p \cup B_p) \right|\right).$$

Dabei bezeichnet $B_p := \cup_{i \in F_p} S_i \setminus \Omega_p$ die Menge aller stark gekoppelten Nachbarpunkte j , deren Index wir bereits im ersten Schritt ermittelt haben.
Wir gehen jetzt davon aus, dass die Menge der Nicht-Null-Einträge pro Zeile $|A_i|_{nz}$ unabhängig von i und N durch einen kleinen Wert c beschränkt wird (dünnbesetzte Matrix). Damit ist auch $|S_i| < |A_i|_{nz} \leq c$. In diesem Fall lässt sich der Gesamtkommunikationsaufwand für die Bereitstellung der notwendigen Daten auf Prozessor p durch

$$O\left(|\partial \Omega_p|\right)$$

abschätzen.
Eine genaue Präallokation für die Interpolation ist wichtig im Hinblick auf den Gesamtspeicherverbrauch des Verfahrens (siehe auch die Diskussion in Abschnitt 3.4). Während die Bestimmung der Anzahl der Nicht-Nulleinträge für die Interpolation keine weitere Kommunikation erfordert, müssen wir vor der Aufstellung der Restriktion einige Daten austauschen. Interpoliert ein Wert am Punkt $i \in F_p$ von einem Wert an $j \in C_q$, $q \neq p$, so bedeutet dies einen nicht-Nulleintrag in der j-ten Zeile der Restriktion. Unter Benutzung der direkten Interpolation muss Prozessor p darum

$$O\left(\left| \cup_{i \in F_p} S_i \cap C_q \right|\right)$$

Werte an Prozessor q übergeben.
Bei der Standardinterpolation handelt es sich um

$$O\left(\left| \cup_{i \in F_p} (S_i \cap C_q) \cup \left(\cup_{j \in S_i \cap F_q} (S_j \cap C_q)\right) \right|\right)$$

Werte. Wir können diese Terme wiederum durch $O(|\partial\Omega_p|)$ abschätzen.
Bei einer Gebietsagglomeration erhöht sich diese Kommunikation natürlich erheblich. In diesem Fall muss ein Prozessor p, der am weiteren Vergröberungsverfahren nicht mehr teilnimmt, zusätzlich die Anzahl der Nicht-Nulleinträge der Restriktion für alle $|C_p|$ der an den Prozessor p übermitteln, der die Verwaltung dieser Punkte im nächsten Level übernimmt.
Bezüglich des Kommunikationsaufwandes für die Aufstellung des Grobgitteroperators betrachten wir zunächt die Multiplikation des Feingitteroperators mit der Interpolationsmatrix $T = A \cdot P$. Wir müssen diejenigen Matrixzeilen A_j, $j \notin \Omega_p$, auf Prozessor p übertragen, für die es einen Eintrag $a_{ij} \neq 0$, $i \in \Omega_p$, gibt. Wir gehen wieder von einer beschränkten Anzahl der Nichtnull-Einträge pro Zeile aus und können damit schließen, dass wir

$$O\left(|\partial\Omega_p|\right)$$

Zeilen übertragen müssen.
Für die zweite Multiplikation müssen die Zeilen T_j mit $r_{ij} \neq 0$, $i \in \Omega_p$ und $j \notin \Omega_p$, auf Prozessor p übertragen werden.
Wir beschränken uns jetzt auf symmetrische Operatormatrizen und betrachten die direkte Interpolation über die Teilgebietsgrenze hinweg. Dann lässt sich der Aufwand für das Triple-Matrix-Produkt abschätzen durch

$$O\left(|\partial\Omega_p|\right).$$

Während eines Agglomerationschrittes ist die zu übertragende Datenmenge bedeutend größer. In diesem Fall muss ein Prozessor q, der im nächsten Level nicht mehr beteiligt ist, insgesamt $O(|\Omega_q|)$ Zeilen der Matrix T verschicken.

4.5.1 Kommunikationsaufwand für das Subdomain Blocking

Bei den Varianten des Subdomain Blockings fällt außer den in vorigen Abschnitt genannten Kommunikationsschritten keine weitere Datenübertragung an. Beim Full Subdomain Blocking können wir sogar auf die Kommunikation der Werte $|S_j^T|$ verzichten, weil alle Variablen $i \in \partial\Omega_p$ in das grobe Gitter aufgenommen werden. Insbesondere muss der Wert an i nicht von demjenigen an $j \in S_i \setminus \Omega_p$ interpoliert werden, die Abhängigkeit des Punktes i vom Punkt j soll sich also nicht im Wert λ_j niederschlagen.

4.5.2 Kommunikationsaufwand für den RS3-Algorithmus

Der Kommunikationsaufwand für den RS3-Algorithmus ist vergleichbar mit dem für das Aufstellen eines Standard-Interpolationsoperators über eine Prozessorgrenze hinweg. Nach Anwendung der zweiten Phase des klassischen Vergröberungsalgorithmus und vor der RS3-Phase übertragen wir zunächst für alle Feingitterpunkte $i \in F_p$ die Zugehörigkeit von $j \in S_i \setminus \Omega_p$ zu C oder F. Dies erfordert einen Aufwand von

$$O\left(\left|\cup_{i\in F_p}\left(S_i \setminus \Omega_p\right)\right|\right),$$

siehe 4.3. Danach muss für alle Feingitterpunkte $j \in S_i \cap F_q$, $q \neq p$, die Matrixzeile A_j übertragen werden. Es ergibt sich wiederum ein Aufwand von

$$O\left(\sum |A_j|_{nz} : \; j \in \bigcup_{i\in F_p} \left(S_i \cap F \setminus \Omega_p\right)\right). \tag{4.4}$$

Nach erfolgter RS3-Phase überträgt Prozessor p diejenigen Indizes $i \in \Omega_q$, $q > p$, die von Prozessor p in das Grobgitter eingefügt worden sind, an Prozessor q. Der Aufwand hierfür ist beschränkt durch

$$O\left(\left|\cup_{i\in F_p}\left(S_i \setminus \Omega_p\right)\right|\right), \tag{4.5}$$

insgesamt erhalten wir (bei begrenzter Anzahl von Nicht-Nulleinträgen pro Matrixzeile) einen Aufwand pro Prozessor von

$$O\left(|\partial\Omega_p|\right).$$

4.5.3 Kommunikationsaufwand für den CLJP- und den Falgout-Algorithmus

Anders als bei den bisher besprochenen Verfahren findet die Kommunikation für den CLJP-Algorithmus nicht nur vor oder nach der Anwendung des Verfahrens statt. Während eines einzelnen CLJP-Iterationsschrittes finden insgesamt drei Kommunikationsschritte statt, siehe Programm 4.3.3. Die ersten beiden Schritte dienen der Aufdatierung von D und m, der letzte der Aktualisierung der Gewichte w.
Nun lassen sich D und m in dasselbe Array speichern, indem wir wie folgt vorgehen:

$$m_j = \begin{cases} 1 \text{ falls } j \in D, \\ -i \text{ falls } j \in S_i^T, \\ 0 \text{ sonst.} \end{cases}$$

Um den Kommunikationsaufwand des Verfahrens abzuschätzen, erinnern wir noch einmal daran, das jeder Prozessor p diese Marke m nicht nur für seine lokalen Punkte $i \in \Omega_p$ abspeichert, sondern auch für alle Punkte $i \in \cup_{j \in \partial\Omega_p} S_j \setminus \Omega_p$ und die Heuristiken entlang der (lokal ermittelten) Kopplungen S_i^T anwendet.
Wir untersuchen zuerst Kommunikationsschritt 1. In diesem Schritt übertragen zunächst alle Prozessoren q, die einen Wert m_i für den Punkt $i \in \Omega_p$ gespeichert haben, den Index i an Prozessor p, falls i nicht in der Menge D verbleiben soll. Danach sendet Prozessor p den aktualisierten Wert m_i an alle oben genannten Prozessoren q. Der letztgenannte Schritt erfordert auf jedem Prozessor q einen Kommunikationsaufwand von

$$O\left(\left|\cup_{i \in F_q} (S_i \setminus \Omega_q)\right|\right)$$

m-Werten, der Aufwand des ersten Schrittes ist abhängig von der Verteilung der σ-Werte und wird ebenfalls durch den oben genannten Term beschränkt. Dieselben Überlegungen gelten auch für Kommunikationsschritt 2. Zunächst übertragen alle Prozessoren q die Indizes zu löschender D-Punkte i auf denjenigen Prozessor p mit $i \in \Omega_p$, danach überträgt Prozessor p die Marke m_i —die auch die Zugehörigkeit zu D enthalten— wieder zurück.
Kommunikationsschritt 3 besteht ebenfalls aus zwei Teilschritten. Im ersten Schritt akkumulieren wir die Werte w_i auf Prozessor p mit $i \in \Omega_p$, danach verteilen wir diese Werte zurück an die Prozessoren q. Der Kommunikationsaufwand pro Prozessor q beträgt wiederum

$$O\left(\left|\cup_{i \in F_q} (S_i \setminus \Omega_q)\right|\right).$$

Damit erfordert der Austausch der Werte w und m pro Iterationsschritt jeweils einen Kommunikationsaufwand in der Größenordnung

$$O\left(|\partial\Omega_q|\right).$$

Eine Abschätzung für die Gesamtzahl der Iterationsschritte dieser Verfahren ist nicht bekannt, deshalb ist auch eine weitergehende Abschätzung für den Gesamtkommunikationsaufwand nicht möglich.

4.5.4 Kommunikationsaufwand für den CGC-Algorithmus

Bei der Anwendung des CGC-Algorithmus findet der erste Kommunikationsschritt nach den Durchläufen des klassischen Vergröberungsalgorithmus statt. Um den Graphen aufzustellen, benötigt Prozessor p über jeden stark

gekoppelten Nachbarpunkt $j \in \cup_{i \in \partial\Omega_p} S_i \setminus \Omega_p$ die Kenntnis, zu welchem Gitter C_i dieser Punkt gehört. Der hierfür erforderliche Kommunikationsaufwand beträgt

$$O\left(\left|\cup_{i \in F_p} (S_i \setminus \Omega_p)\right|\right),$$

vergleiche (4.3).
Nach Austausch dieser Daten stellt jeder Prozessor lokal seinen Teil des Graphen auf. Der gesamte Graph wird dann auf den ersten Prozessor übertragen. Dies erfordert einen Kommunikationsaufwand von $O(|E_p|)$ pro Prozessor, also insgesamt einen Aufwand von $O(|E \setminus E_1|)$. (Die Daten auf Prozessor 1 müssen nicht übertragen werden).
Die Anzahl der erzeugten Gitter pro Prozessor lässt sich bei den meisten Problemen auf einen kleinen Wert begrenzen. Damit wird auch die Mächtigkeit der Kantenmenge E_p auf $O\left(|\mathcal{S}_p|\right)$ begrenzt, wobei der Wert

$$\mathcal{S}_p = \{q \neq p : \ \exists i \in \Omega_p, \ j \in \Omega_q : \ j \in S_i\}$$

abhängig von dem gegebenem Problem und der konkreteten Gebietszerlegung ist. Die Kommunikation nach der Durchführung des Programms 4.4 erfordert lediglich die Übertragung von insgesamt P Werten.
Vor der Anwendung des Korrekturschrittes (Programm 4.10) übertragen wir für alle Punkte

$$j \in \cup_{i \in \partial F_p} S_i \setminus \Omega_p$$

die Zugehörigkeit von j zu C oder F, also insgesamt wieder

$$O\left(\left|\cup_{i \in F_p} (S_i \setminus \Omega_p)\right|\right),$$

Werte. Eine weitere Kommunikation wie beim RS3-Verfahren nach erfolgtem Durchlauf findet nicht statt, da jeder Prozessor nur lokale Punkte manipuliert. Insgesamt übersteigt der Kommunikationsaufwand für Prozessor p wiederum nicht

$$O\left(|\partial\Omega_p|\right).$$

Wir haben gesehen, dass für alle Vergröberungsverfahren der Kommunikationsaufwand pro Prozessor p durch $O\left(|\partial\Omega_p|\right)$ abgeschätzt werden kann, wobei dieser Wert für isotrope Probleme, bei denen jede Kopplung stark ist, auch angenommen wird. Eine genauere Unterscheidung ist nur anhand gegebener Beispiele möglich. Im nächsten Kapitel werden wir die Laufzeiten der Verfahren in der Praxis anhand einiger Beispiele untersuchen.

Kapitel 5

Numerische Ergebnisse

Wir wollen nun die verschiedenen Verfahren in den Punkten Laufzeit, Speicherplatz und Skalierbarkeit miteinander vergleichen. Der Speicheraufwand wird charakterisiert durch die *Operatorkomplexität*

$$c_A := \frac{\sum_{l=1}^{L_{max}} nonzeros\left(A^l\right)}{nonzeros\left(A^1\right)}$$

und die *Gitterkomplexität*

$$c_G := \frac{\sum_{l=1}^{L_{max}} |\Omega^l|}{|\Omega^1|}.$$

Diese Werte beschreiben, um welchen Faktor der Speicherbedarf gegenüber dem des zu lösenden Gleichungssystems anwächst, vergleiche (2.5) und (2.6). Zur parallelen Leistungsmessung führen wir jetzt den *speed-up* S_P und die *parallele Effizienz* E_P ein. Wir bezeichnen mit $T_P(N)$ die Laufzeit des Codes auf P Prozessoren bei der Problemgröße N. Der *speed-up*

$$S_P(N) := \frac{T_1(N)}{T_P(N)} \leq P$$

charakterisiert die Beschleunigung des parallelen Codes angesetzt auf eine feste Problemgröße N. Der ideale Wert für S_P beträgt P, d.h. auf P Prozessoren wird das Problem P-mal so schnell gelöst wie auf einem Prozessor.
Mit dem speed-up verwandt ist die *parallele Effizienz*

$$E_P(N) := \frac{S_p(N)}{P} \leq 1.$$

Der ideale Wert für die parallele Effizienz beträgt 1, was einem speed-up von $S_P = P$ entspricht. Für eine ausführliche Diskussion verweisen wir auf

[GO93], Kapitel 3.

Wir benutzen als Parameter für die erste Phase des klassischen Algorithmus den Parameter $\alpha = 0.25$, für die zweite Phase $\beta = 0.35$. Als Interpolation benutzen wir die Standardinterpolation, welche wir wie in Abschnitt 2.6.2 beschrieben mit dem Trunkationsparameter $\epsilon_{tr} = 0.2$ abschneiden. Über die Teilgebietsränder hinweg interpolieren wir der wesentlich niedrigeren Kosten wegen mit der direkten Interpolation (Abschnitt 4.2). Wir fassen jeweils zwei Prozessorteilgebiete zu einem zusammen, wenn der Anteil der Teilgebietsrandvariablen an der Gesamtzahl der Punkte 70% übersteigt (soweit nicht anders angegeben). Die Setupphase wird beendet, wenn entweder nur noch ein Punkt vorhanden ist oder die Anzahl der Punkte durch die Vergröberung nicht mehr reduziert wird.
Wir starten die Iterationen mit einem zufällig gewählten, auf 1 normierten Startvektor. Wir messen das Residuum in der l^2-Norm und brechen die Iteration ab, wenn die Norm des Residuums r_{it} unter 10^{-10} fällt. Die Konvergenzrate berechnet sich gemäß

$$\rho = \left(\frac{r_{it}}{r_1} \right)^{\frac{1}{it-1}},$$

dabei ignorieren wir den Übergang vom Anfangsresiduum r_0 zum Residuum der ersten Iterierten r_1, um störende Starteffekte auszuschließen.
In den Tabellen und Grafiken benutzen wir folgende Abkürzungen:

- P: Anzahl der Prozessoren,
- NONE: Anwendung des klassischen Algorithmus auf die Prozessorteilgebiete nach Aufdatierung der Gewichte λ,
- FSB: full subdomain blocking (Abschnitt 4.3.1)
- MSB : minimum subdomain blocking (Abschnitt 4.3.1),
- RS3 : RS3-Algorithmus (Abschnitt 4.3.2,
- CLJP : CLJP-Algorithmus (Abschnitt 4.3.3),
- Falgout : Falgout-Algorithmus (Abschnitt 4.3.4),
- CGC : CGC-Verfahren (Abschnitt 4.4),
- CGC-RS3 : CGC-Verfahren mit RS3-Randbehandlung (Abschnitt 4.4).

Wir benutzen für die numerischen Experimente zwei verschiedene Rechner, einerseits den Linux-Cluster Parnass2 [SZG99], welcher mit 80 Pentium2/400-MHz-Prozessoren und insgesamt 4 GB RAM ausgestattet ist. Die Kommunikation zwischen den Dualprozessor-Knoten findet mittels Myrinet statt. Diese Netzwerkarchitektur ermöglicht eine Kommunikation mit 1.28 Gb/s.
Um Ergebnisse auf einer größeren Prozessorzahl zu erzielen, rechnen wir zusätzlich auf einem Cluster aus 41 IBM p690 mit jeweils 32 Prozessoren und 128 GB Hauptspeicher. Jeder dieser Knoten hat eine Leistung von 218 GFLOPS. Die Knoten sind über ein HPS (High Perfomance Switch)-Netzwerk verbunden, welches eine Bandbreite von über 1200 MB/s pro Link und eine Latenzzeit von unter 11μs aufweist. Die peak performance des gesamten Rechners beträgt 8.9 TFLOPS, die LINPACK-Performance 5.568 TFLOPS.

5.1 Laplaceoperator

Wir betrachten zunächst das Dirichletproblem in zwei Raumdimensionen

$$-\Delta u \;=\; f \text{ in } \Omega \tag{5.2}$$

auf einem rechteckigem Gebiet $\Omega \subset \mathbb{R}^2$ mit Nullrandbedingungen. Wir diskretisieren die Gleichung mittels eines 5-Punkte Finite-Differenzen-Schemas,

$$\frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}_h .$$

Zuerst betrachten wir das speed-up-Verhalten für 1 bis 64 Prozessoren und einem Gebiet von 512×512 Punkten. Aus Tabelle 5.1 und Abbildung 5.1 (a)

P	NONE	FSB	MSB	RS3	CLJP	Falgout	CGC
1	76.0	76.0	76.0	76.0	105	76.0	77.6
2	39.4	38.4	66.1	37.3	62.6	43.8	41.1
4	19.8	27.7	36.1	20.6	30.6	22.2	20.2
8	10.3	20.4	15.4	10.9	16.0	11.6	10.6
16	5.55	19.3	7.08	6.06	8.81	6.39	5.70
32	3.20	18.7	3.78	3.63	5.65	4.16	3.45
64	2.40	14.7	2.57	2.81	5.22	3.80	2.62

Tabelle 5.1: Setupzeit in Sekunden für Problem (5.2)

wird sofort das schlechte speed-up-Verhalten des full subdomain blockings ersichtlich. Alle anderen Verfahren weisen bis zu 16 Prozessoren eine gute parallele Effizienz über 70% (etwas weniger für das minimum subdomain blocking) auf, darüber hinaus nimmt der Anteil der der Teilgebiets-Randvariablen zu und die Effizienz sinkt ab.

Die Operatorkomplexität für das full subdomain blocking wird durch die

P	NONE	FSB	MSB	RS3	CLJP	Falgout	CGC
1	2.60	2.60	2.60	2.60	3.87	2.60	2.60
2	2.59	2.76	2.62	2.62	4.00	2.62	2.59
4	2.61	3.14	2.66	2.67	3.97	2.65	2.60
8	2.62	4.11	2.68	2.70	4.32	2.70	2.62
16	2.65	4.74	2.70	2.71	4.26	2.74	2.64
32	2.68	5.96	2.76	2.76	4.12	2.81	2.68
64	2.73	6.99	2.80	2.80	4.10	2.89	2.71

Tabelle 5.2: Operatorkomplexität für Problem (5.2)

vielen Randvariablen stark erhöht (Tabelle 5.2). Das CLJP-Verfahren, welches mehr Punkte in das grobe Gitter aufnimmt als der klassische Ruge-Stüben-Algorithmus, weist ebenfalls eine hohe Operatorkomplexität auf, allerdings steigt sie bei größerer Prozessorzahl nicht so stark an. Obwohl der CLJP-Algorithmus für eine beliebige Prozessorzahl dasselbe Grobgitter erzeugen kann, ist eine einheitliche Operatorkomplexität für alle Level des CLJP-Verfahrens hier nicht gegeben, weil die Interpolation über die Teilgebietsränder hinweg anders verläuft als im Teilgebietsinnern und damit die Grobgitteroperatoren für die verschiedenen Level eine andere Gestalt annehmen. Die Operatorkomplexität für das Falgout-Verfahren liegt bedingt durch die CLJP-Schritte höher als derjenige des MSB- und des RS3-Verfahrens. Wird keine gesonderte Randbehandlung durchgeführt, also keine zusätzlichen Punkte eingefügt, so lässt sich die niedrigste Operatorkomplexität erzielen (Spalten NONE und CGC).

Wir sehen in Tabelle 5.3 die Bedeutung der Parallelisierungsstrategie für die Konvergenzgeschwindigkeit des Verfahrens. Der klassische Ruge-Stüben-Algorithmus erzielt eine um bis zu 30% schlechtere Fehlerreduktion als das RS3- und das Falgout-Verfahren. Es zeigt sich, dass die vielen Randvariablen des FSB-Algorithmus die Konvergenz stark verschlechtern. Das MSB-Verfahren zeigt ebenfalls verschlechterte Konvergenzraten auf, weil die Anwesenheit von starken $F-F$-Kopplungen über den Teilgebietsrand bei diesem Verfahren nicht überprüft wird.

Die Mehrgitter-Lösungszeit (Tabelle 5.4) und die Gesamtzeit (Tabelle 5.5) bestätigen die bisherigen Ergebnisse. Das Falgout-Verfahren ermöglicht die

P	NONE	FSB	MSB	RS3	CLJP	Falgout	CGC
1	0.13	0.13	0.13	0.13	0.16	0.13	0.13
2	0.19	0.25	0.26	0.13	0.17	0.13	0.13
4	0.19	0.25	0.28	0.13	0.17	0.13	0.13
8	0.19	0.36	0.27	0.14	0.16	0.13	0.18
16	0.18	0.37	0.26	0.14	0.16	0.14	0.14
32	0.19	0.39	0.28	0.14	0.16	0.14	0.18
64	0.19	0.35	0.26	0.14	0.19	0.14	0.15

Tabelle 5.3: Konvergenzraten für Problem (5.2)

P	NONE	FSB	MSB	RS3	CLJP	Falgout	CGC
1	45.2	45.0	45.2	45.2	68.1	45.2	44.9
2	37.5	47.7	45.9	32.9	48.9	33.7	31.3
4	18.6	27.7	22.9	15.8	21.7	15.8	15.7
8	8.99	26.8	12.2	8.32	12.8	7.96	8.89
16	4.53	24.6	5.78	4.23	6.46	4.09	4.17
32	2.56	24.0	3.43	2.30	3.31	2.32	2.56
64	1.61	19.7	1.89	1.41	1.98	1.36	1.41

Tabelle 5.4: Mehrgitter-Lösungszeit in Sekunden für Problem (5.2)

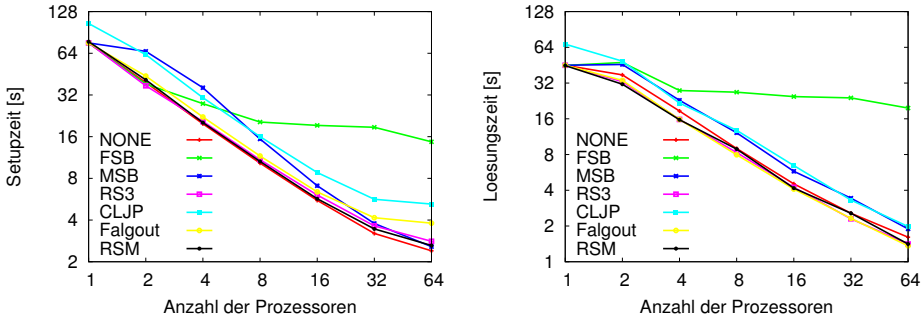
schnellste Lösung, jedoch führen die erhöhten Kosten während der Setupphase zu einer längeren Gesamtzeit.

In Abbildung 5.2 ist das speed-up-Verhalten noch einmal in Graphenform

P	NONE	FSB	MSB	RS3	CLJP	Falgout	CGC
1	121	121	121	121	174	121	123
2	76.9	86.1	111	70.2	111	77.5	72.4
4	39.4	53.8	59.0	36.3	52.3	38.0	35.9
8	19.3	47.8	27.6	19.2	28.8	19.6	19.5
16	10.1	44.0	12.4	10.9	15.3	10.5	9.87
32	5.76	42.6	7.20	5.93	8.96	6.64	4.17
64	4.01	34.3	4.46	4.22	7.20	5.16	4.03

Tabelle 5.5: Gesamtzeit in Sekunden für Problem (5.2)

zusammengefasst. Deutlich erkennbar ist das sehr schlechte Resultat des full subdomain blockings, welches schon bei 8 und mehr Prozessoren eine parallele Effizienz von unter 50% aufweist. Die anderen Verfahren —bis auf das minimum subdomain blocking— weisen bei dieser Prozessorzahl noch eine Effizienz von über 75% auf. In den folgenden Beispielen werden wir auf die Verwendung des full subdomain blockings verzichten.

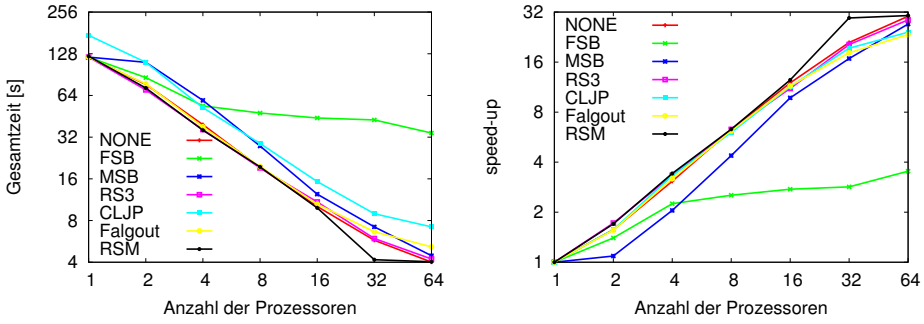


(a) Setupzeit (b) Mehrgitter-Lösungszeit

Abbildung 5.1: Setup- und Mehrgitter-Lösungszeit für Problem (5.2)

P	NONE	FSB	MSB	RS3	CLJP	Falgout	CGC
1	1	1	1	1	1	1	1
2	1.57	1.40	1.09	1.72	1.57	1.56	1.70
4	3.07	2.25	2.05	3.33	3.32	3.18	3.42
8	6.27	2.53	4.38	6.30	6.04	6.17	6.31
16	12.0	2.75	9.75	11.1	11.4	11.5	12.5
32	21.0	2.84	16.8	20.4	19.4	18.2	29.5
64	30.2	3.53	27.1	28.7	24.2	23.4	30.5

Tabelle 5.6: speed-up für Problem (5.2)



(a) Gesamtzeit in Sekunden (b) speed-up

Abbildung 5.2: speed-up-Verhalten des Dirichletproblems (5.2)

Wir betrachten nun das Skalierungsverhalten der Verfahren. Dazu führen wir die Diskretisierung auf jeweils 128×128 Punkten pro Prozessor aus.
Tabelle 5.7 und Abbildung 5.3 (a) zeigen die Setupzeiten für dieses Problem,

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	1.14	1.14	1.14	1.36	1.14	1.15
2	1.38	1.76	1.40	1.70	1.56	1.42
4	1.42	2.14	1.52	1.80	1.60	1.49
8	1.49	2.13	1.57	1.90	1.71	1.71
16	1.74	2.46	1.86	2.26	2.03	1.82
32	1.99	2.70	2.08	2.70	2.27	2.04
64	2.58	3.29	2.73	4.05	3.13	2.63
128	3.68	4.49	3.90	6.21	4.84	3.73
256	5.93	6.37	6.13	11.3	8.69	6.33

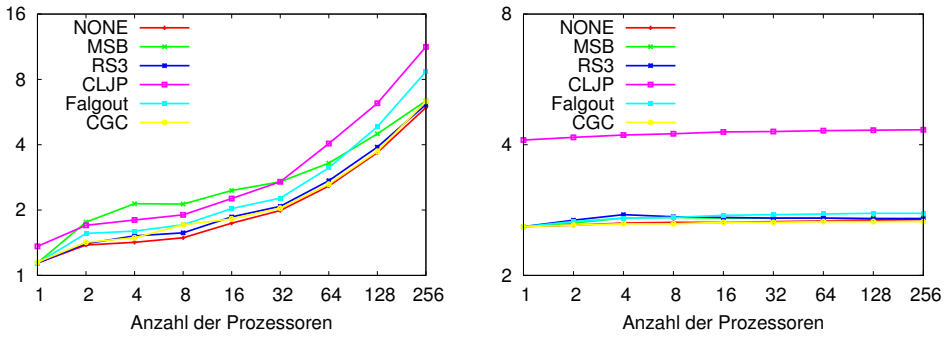
Tabelle 5.7: Setupzeit in Sekunden für Problem (5.2)

gerechnet auf 1 bis 256 Prozessoren. Das in dieser Hinsicht teuerste Verfahren ist bis zu 16 Prozessoren das MSB-Verfahren, darüber hinaus das CLJP-Verfahren. Das MSB-Verfahren verursacht wegen der ungünstigen Randsituationen höhere Kosten beim Aufstellen des Grobgitteroperators, bei höheren Prozessorzahlen wird dies jedoch durch den geringeren Kommunikationsaufwandes während des Vergröberns ausgeglichen. Der CGC-Algorithmus erfordert bis auf eine Ausnahme nicht mehr als ca. 5% mehr Zeit als der klassiche Ruge-Stüben-Algorithmus.
In Tabelle 5.8 sind die Operatorkomplexitäten zusammengefasst. Auffällig

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	2.59	2.59	2.59	4.10	2.59	2.59
2	2.61	2.64	2.68	4.16	2.66	2.61
4	2.64	2.71	2.76	4.21	2.71	2.63
8	2.65	2.72	2.73	4.24	2.72	2.63
16	2.65	2.71	2.72	4.28	2.75	2.65
32	2.66	2.71	2.71	4.29	2.76	2.65
64	2.67	2.71	2.71	4.31	2.77	2.66
128	2.68	2.71	2.70	4.32	2.78	2.66
256	2.69	2.71	2.70	4.33	2.78	2.66

Tabelle 5.8: Operatorkomplexität für Problem (5.2)

sind vor allem die Werte für den CLJP-Algorithmus, welcher mit 2.15 ebenfalls die höchsten Gitterkomplexität erzeugt. Der auf den CLJP-Algorithmus basierende Falgout-Algorithmus erzeugt bedingt durch die vielen benachbarten Grobgitterpunkte an den Teilgebietsrändern ebenfalls eine etwas teurere



(a) Setupzeit (b) Operatorkomplexität

Abbildung 5.3: Setupzeit und Operatorkomplexität für Problem (5.2)

Operatorhierarchie, jedoch übersteigt dessen Gitterkomplexität —wie auch bei den anderen Verfahren— den Wert 1.7 nicht. Der Grund hierfür liegt in der „klassischen“ Vergröberung für innere Punkte, welche die Mehrzahl aller Punkte ausmachen.

In Tabelle 5.9 sind die Konvergenzraten festgehalten. Wir sehen, dass der

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	0.13	0.13	0.13	0.13	0.13	0.13
2	0.16	0.25	0.13	0.13	0.13	0.13
4	0.19	0.28	0.13	0.13	0.13	0.14
8	0.19	0.27	0.13	0.14	0.13	0.14
16	0.18	0.26	0.14	0.16	0.14	0.14
32	0.18	0.27	0.14	0.16	0.14	0.14
64	0.19	0.26	0.14	0.17	0.14	0.14
128	0.19	0.27	0.14	0.19	0.14	0.14
256	0.21	0.27	0.15	0.19	0.14	0.16

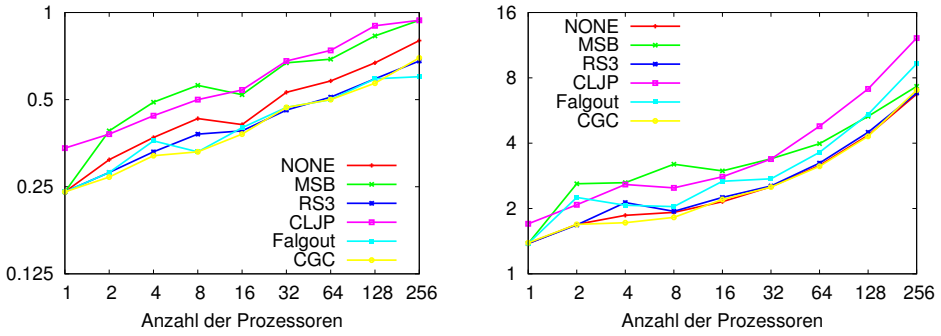
Tabelle 5.9: Konvergenzraten für Problem (5.2)

MSB-Algorithmus einige $F - F$-Kopplungen über den Teilgebietsrand nicht beachtet hat und deshalb die Konvergenzrate dieses Verfahrens gegenüber derjenigen der sequentiellen Lösung auf einem Prozessor nahezu verdoppelt ist. Damit funktioniert dieses Verfahren noch schlechter als ein „naiver“ Ruge-Stüben-Ansatz, für den dieser Wert sich um 50% erhöht.

Das Verhalten der Lösungszeit (Tabelle 5.10 und Abbildung 5.4 (a)) entspricht im wesentlichen dem Verhalten der Konvergenzrate. Einzige Ausnahme ist dabei, dass das MSB-Verfahren nicht langsamer als das CLJP-

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	0.24	0.24	0.24	0.34	0.24	0.24
2	0.31	0.39	0.28	0.38	0.28	0.27
4	0.37	0.49	0.33	0.44	0.36	0.32
8	0.43	0.56	0.38	0.50	0.33	0.33
16	0.41	0.52	0.39	0.54	0.40	0.38
32	0.53	0.67	0.46	0.68	0.47	0.47
64	0.58	0.69	0.51	0.74	0.50	0.50
128	0.67	0.83	0.59	0.90	0.59	0.57
256	0.80	0.94	0.68	0.94	0.60	0.69

Tabelle 5.10: Mehrgitter-Lösungszeit für Problem (5.2)



(a) Mehrgitterzeit

(b) Gesamtzeit

Abbildung 5.4: Mehrgitter-Lösungszeit und Gesamtzeitaufwand für Problem (5.2)

Verfahren ist. Der Grund hierfür liegt in den kleineren Gittern des MSB-Verfahrens, wodurch die Glättungsschritte schneller angewendet werden können. Der Gesamtzeitverbrauch (Tabelle 5.11 und Abbildung 5.4 (b)) wird hauptsächlich durch die Setupzeit bestimmt, so dass auch hier bei kleinen Prozessorzahlen das MSB-Verfahren und bei größeren Prozessorzahlen das CLJP-Verfahren die höchsten Werte aufweisen. Das RS3- und das CGC-Verfahren weisen trotz schnellerer Lösung keine Vorteile gegenüber dem klassischen Ruge-Stüben-Algorithmus auf, weil ihre Setupphasen höhere Kosten verursacht. Die teure Setupphase des Falgout-Verfahrens impliziert auch eine höhere Gesamtlaufzeit. Wir werden auch das CLJP-Verfahren ab sofort nicht mehr berücksichtigen, weil sich die Lösung schon für dieses einfache Problem als sehr aufwändig herausgestellt hat.

P	NONE	MSB	RS3	CLJP	Falgout	CGC
1	1.38	1.38	1.38	1.70	1.38	1.39
2	1.69	2.60	1.68	2.08	2.25	1.69
4	1.86	2.63	2.13	2.58	2.07	1.72
8	1.92	3.20	1.94	2.49	2.04	1.82
16	2.15	2.98	2.25	2.80	2.67	2.20
32	2.52	3.39	2.54	3.38	2.74	2.51
64	3.16	3.98	3.24	4.79	3.63	3.13
128	4.35	5.32	4.49	7.11	5.43	4.30
256	6.73	7.31	6.81	12.8	9.29	7.02

Tabelle 5.11: Gesamtzeitaufwand für Problem (5.2)

P	NONE	MSB	RS3	Falgout	CGC
1	0.97	0.97	0.97	0.97	0.99
2	1.39	1.92	1.75	2.40	1.36
4	1.72	2.60	3.83	4.27	1.56
8	2.90	4.05	11.6	12.7	3.08
16	3.65	5.36	27.5	24.4	3.56
32	5.41	7.42	44.0	52.8	4.40
64	12.2	15.1	—	—	9.29
128	22.7	25.7	—	—	17.7
256	40.7	35.6	—	—	29.7

Tabelle 5.12: Setupzeit in Sekunden für den dreidimensionalen Laplaceoperator

Als nächstes Beispiel betrachten wir die Diskretisierung des Laplaceoperators im dreidimensionalen Raum mittels finiter Differenzen in einem 7-Punkte-Schema. Wir untersuchen nur das Skalierungs-Verhalten des MSB-Verfahrens, des RS3-Verfahrens, der Falgout-Vergröberung sowie des CGC-Verfahrens und vergleichen die Ergebnisse mit der klassischen Ruge-Stüben-Vergröberung ohne Randbehandlung. Wir diskretisieren dabei das Gebiet mit $16 \times 16 \times 16$ Punkten pro Prozessor. Eine solche Verteilung impliziert einen hohen Randvariablenanteil auf dem Ausgangsgitter: Auf den „inneren“ Prozessoren weisen 1352 von 4096 Punkten eine starke Kopplung zu einem Punkt auf einem anderen Prozessor auf. In diesem Fall findet deshalb eine Agglomeration bereits statt, wenn der Anteil der Randpunkte 60% übersteigt.

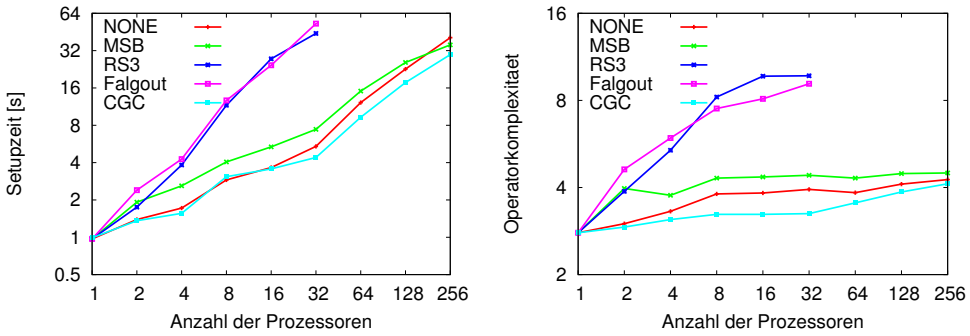
Wir sehen in diesem Fall eine sehr hohe Setupzeit für das RS3- und das Falgout-Verfahren (Tabelle 5.12). Grund hierfür ist das Einfügen vieler zusätzlicher Punkte am Gebietsrand, was den Randvariablenanteil im nächsten Level spürbar erhöht. So enthält die zweite Ebene des RS3-Algorithmus auf 32

P	NONE	MSB	RS3	Falgout	CGC
1	2.79	2.79	2.79	2.79	2.79
2	3.00	3.97	3.88	4.62	2.92
4	3.31	3.76	5.38	5.93	3.10
8	3.80	4.31	8.22	7.50	3.23
16	3.83	4.35	9.70	8.10	3.23
32	3.94	4.41	9.73	9.13	3.25
64	3.84	4.31	—	—	3.54
128	4.11	4.47	—	—	3.86
256	4.26	4.49	—	—	4.12

Tabelle 5.13: Operatorkomplexität für das dreidimensionale Dirichletproblem

Prozessoren insgesamt 27497 Unbekannte, von denen 96% zu den Randvariablen gehören. Deshalb wird das nächste Level —mit 16738 Unbekannten— auf 16 Prozessoren behandelt. Im Gegensatz dazu weist das zweite Level des klassischen Algorithmus auf 32 Prozessoren lediglich 15225 Unbekannte auf, das nächste Level wird auf 16 Prozessoren behandelt und enthält 5063 Unbekannte.

Die hohe Randvariablenanzahl verursacht für das RS3- und das Falgout-



(a) Setupzeit (b) Operatorkomplexität

Abbildung 5.5: Setupzeit und Operatorkomplexität für das dreidimensionale Dirichletproblem

Verfahren eine hohe Operatorkomplexität (Tabelle 5.13), welche gegenüber dem klassischen Algorithmus ab 8 Prozessoren mehr als verdoppelt wird. Bei dieser Prozessorzahl weist das RS3-Verfahren eine Gitterkomplexität von 2.23 auf, während sie für das klassische Verfahren 1.7 nicht überschreitet. Deutlicher als im zweidimensionalem Fall (Tabelle 5.8) führt hier die Verwendung

P	NONE	MSB	RS3	Falgout	CGC
1	0.13	0.13	0.13	0.13	0.14
2	0.17	0.20	0.23	0.33	0.17
4	0.21	0.30	0.48	0.46	0.22
8	0.30	0.32	1.23	1.20	0.27
16	0.35	0.40	2.11	1.58	0.32
32	0.58	0.64	3.26	3.37	0.46
64	0.87	0.94	—	—	0.71
128	5.65	4.24	—	—	4.02
256	4.46	4.16	—	—	4.24

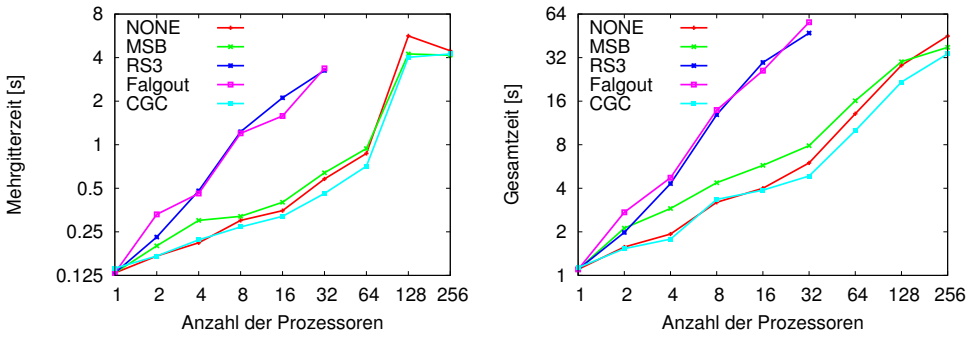
Tabelle 5.14: Mehrgitter-Lösungszeit für das dreidimensionale Dirichletproblem

P	NONE	MSB	RS3	Falgout	CGC
1	1.14	1.14	1.14	1.14	1.13
2	1.57	2.84	1.98	2.73	1.53
4	1.93	2.90	4.31	4.73	1.78
8	3.20	4.37	13.3	13.9	3.35
16	4.00	5.76	29.6	26.0	3.88
32	5.99	7.88	47.3	56.2	4.86
64	13.1	16.1	—	—	10.0
128	28.3	30.0	—	—	21.7
256	45.1	37.7	—	—	34.0

Tabelle 5.15: Gesamtzeitaufwand für das dreidimensionale Dirichletproblem

des CGC-Verfahrens zu besser zusammenpassenden Gittern und damit zu einem geringerem Speicheraufwand. Der Unterschied in der Operatorkomplexität beträgt bis zu 30%.
Große Unterschiede in der Konvergenzrate gibt es bei diesem Problem nicht. Sie steigt von 0.12 im sequentiellen Fall auf 0.15 für 32 Prozessoren und 0.18 für 256 Prozessoren an. Grund für die Verschlechterung der Konvergenzrate bei steigender Prozessorzahl auch unter Benutzung einer Parallelisierungsstrategie ist der hohe Randvariablenanteil, welcher die Qualität der Glättung verschlechtert (vergleiche auch Satz 3.1).

Betrachten wir die Mehrgitter-Lösungszeit (Tabelle 5.1) und den Gesamtzeitaufwand des AMG-Verfahrens (Tabelle 5.1), so sehen wir bei 64 und mehr Prozessoren einen starken Anstieg dieser Werte. Hier kommt die Rechnerarchitektur ins Spiel, die zwischen jeweils 32 Prozessoren auf einem Knoten eine schnelle Kommunikation erlaubt, darüber hinaus jedoch nur eine wesentlich langsamere. Weil der hohe Randvariablenanteil diese Problems den Kommunikationsaufwand stark erhöht, zeigt sich dieser Effekt im dreidimensionalem



(a) Mehrgitter-Lösungszeit (b) Gesamtzeitaufwand in Sekunden

Abbildung 5.6: Mehrgitter-Lösungszeit und Gesamtzeit für das dreidimensionale Dirichletproblem in Sekunden

Fall wesentlich stärker als im zweidimensionalen Fall. In Abbildung 5.6 ist dies nochmal graphisch dargestellt.

5.2 Anisotrope Diffusion

Wir betrachten jetzt die Gleichung

$$-\epsilon u_{xx} - u_{yy} = f \tag{5.3}$$

auf dem Gebiet $\Omega = [0,1]^2$ mit Nullrandbedingungen. Wir diskretisieren diese Gleichung wiederum mit einem 5-Punkte-Stern auf 128×128 Punkten pro Prozessor. Der Diskretisierungsstern damit damit die Form

$$\frac{1}{h^2}\begin{bmatrix} 0 & -1 & 0 \\ -\epsilon & 2+2\epsilon & -\epsilon \\ 0 & -1 & 0 \end{bmatrix}_h .$$

Der Wert für den Parameter ϵ beträgt 0.001.
Wir betrachten in diesem Fall zusätzlich das CGC-Verfahren mit nachträglicher Randbehandlung durch das RS3-Verfahren.
Die Diskretisisierung dieses Verfahrens führt auf dem feinsten Level dazu, dass alle starken Kopplungen in y-Richtung liegen. Damit gibt es Teilgebietsränder, über die keine starken Kopplungen hinweg auftreten. Wie wir sehen werden, führt dies zu einigen interessanten Effekten.

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.51	0.51	0.51	0.51	0.54	0.54
2	0.77	0.82	0.77	0.91	0.76	0.79
4	0.83	1.73	0.82	0.94	0.83	0.81
8	0.92	1.84	0.94	1.06	0.96	0.92
16	1.17	3.86	1.29	1.26	1.16	1.15
32	1.63	6.46	1.80	1.61	1.66	1.74
64	2.59	15.9	2.86	2.44	2.30	2.39
128	4.25	—	5.07	4.20	3.84	3.86
256	6.64	—	7.98	8.58	6.39	6.58

Tabelle 5.16: Setupzeiten für das anisotrope zweidimensionale Problem (5.3)

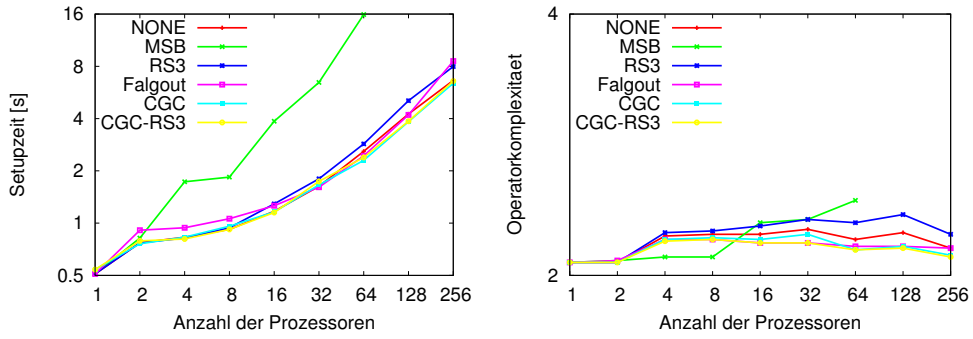
In Tabelle 5.2 sind die Setupzeiten aufgelistet. Der MSB-Algorithmus schneidet hier besonders schlecht ab. Die in x-Richtung orientierten Teilgebietsränder enthalten nämlich keine starken Kopplungen innerhalb des Randes. Diese Teilgebietsränder werden vom MSB-Algorithmus vollständig in das Grobgitter übernommen, was zu einer langsamen Vergröberung führt. Bei mehr als 64 Prozessoren ermöglicht das CGC-Verfahren trotz der zusätzlichen Vergröberungsdurchläufe eine schnellere Setupphase, weil die Gitter besser zusammenpassen und dadurch das Aufstellen des Grobgitteroperators verbilligt wird.

In Tabelle 5.17 fällt der hohe Wert für die Operatorkomplexität des MSB-

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	2.07	2.07	2.07	2.07	2.07	2.07
2	2.07	2.08	2.07	2.08	2.07	2.07
4	2.22	2.10	2.24	2.20	2.20	2.19
8	2.23	2.10	2.25	2.20	2.21	2.20
16	2.23	2.30	2.28	2.18	2.20	2.18
32	2.26	2.32	2.32	2.18	2.23	2.18
64	2.20	2.44	2.30	2.16	2.14	2.14
128	2.24	—	2.35	2.16	2.16	2.15
256	2.15	—	2.23	2.15	2.11	2.10

Tabelle 5.17: Operatorkomplexität für das zweidimensionale anisotrope Problem (5.3)

Verfahrens bei 64 Prozessoren auf, welche durch die langsame Vergröberung hervorgerufen wird. Die Gitterkomplexität liegt mit 2.13 für diese Prozessorzahl ebenfalls deutlich über diejenigen der anderen Verfahren, die bis auf den RS3-Algorithmus lediglich eine Gitterkomplexität von 2.02 aufweisen. Der RS3-Algorithmus weist eine maximale Gitterkomplexität von 2.05 ein, fügt



(a) Setupzeit (b) Operatorkomplexität

Abbildung 5.7: Setupzeit und Operatorkomplexität für das zweidimensionale anisotrope Problem (5.3)

also relativ viele zusätzliche Randpunkte ein. Dies drückt sich auch in den Werten für die Operatorkomplexität aus. Dahingegen erzielt das kombinierte CGC-RS3-Verfahren ab 16 Prozessoren die niedrigsten Operatorkomplexitäten, weil die Auswahl der Gitter durch den CGC-Algorithmus nur wenige zu korrigierende $F - F$-Kopplungen übriglässt. Ohne eine solche Korrektur würden jedoch –bedingt durch die Orientierung der starken Kopplungen nur in y-Richtung– viele stark gekoppelte Paare von Feingitterpunkten i und j übrigbleiben, die zwar jeweils von einem Grobgitterpunkt c und d her interpolieren, aber keinen gemeinsamen Grobgitterpunkt aufweisen. Jedoch führt dies im nächsten Level zu einem Matrixeintrag $a_{cd} \neq 0$, so dass sich die Operatorkomplexität erhöht.
Die Sprünge in den Werten für die Operatorkomplexität sind durch die unterschiedlichen Gebietsstrukturen zu erklären. Bei quadratischer Prozessorzahl sind auch die Teilgebiete quadratisch. In diesem Fall sind die horizontal (in x-Richtung) orientierten Ränder insgesamt ebenso lang wie die vertikal (in y-Richtung) orientierten Ränder. Im anderen Fall gibt es zweimal so viele vertikal orientierten wie horizontal orientierten Ränder. Weil über die vertikal orientierten Ränder keine starken Kopplungen verlaufen, können hier viele benachbarte —schwach gekoppelten— Grobgitterpunkte auftreten, was die Operatorkomplexität erhöht.
Tabelle 5.18 zeigt die Konvergenzraten für dieses Problem. Wir sehen, dass ohne Randbehandlung eine bis zu dreimal größere Rate erzielt wird. Auch das CGC-Verfahren alleine vermag dieses Problem nicht zu lösen, es ist auf jeden Fall ein zusätzlicher RS3-Schritt erforderlich. Die schlechteren Kon-

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.14	0.14	0.14	0.14	0.14	0.14
2	0.14	0.14	0.14	0.14	0.14	0.14
4	0.42	0.14	0.14	0.14	0.42	0.14
8	0.42	0.14	0.14	0.14	0.42	0.14
16	0.43	0.18	0.14	0.14	0.34	0.14
32	0.44	0.17	0.14	0.14	0.34	0.14
64	0.43	0.16	0.14	0.14	0.34	0.14
128	0.43	—	0.14	0.14	0.34	0.14
256	0.42	—	0.14	0.14	0.34	0.14

Tabelle 5.18: Konvergenzraten für das zweidimensionale anisotrope Problem (5.3)

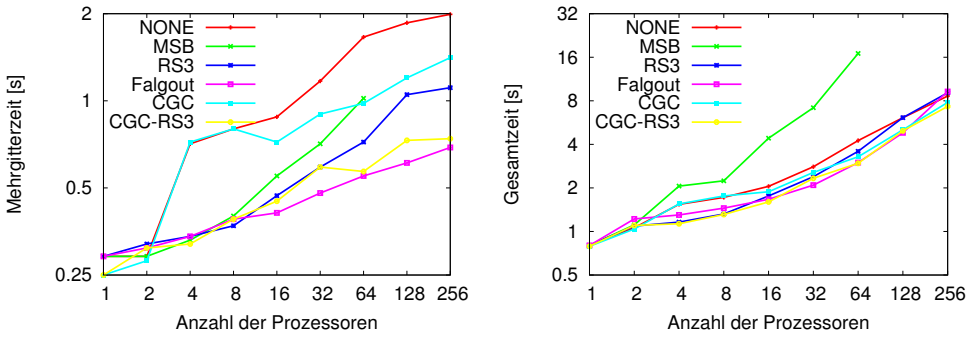
P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.29	0.29	0.29	0.29	0.25	0.25
2	0.29	0.29	0.32	0.31	0.28	0.31
4	0.71	0.33	0.34	0.34	0.72	0.32
8	0.80	0.40	0.37	0.39	0.80	0.39
16	0.88	0.55	0.47	0.41	0.72	0.45
32	1.17	0.71	0.59	0.48	0.90	0.59
64	1.66	1.02	0.72	0.55	0.98	0.57
128	1.86	—	1.05	0.61	1.20	0.73
256	1.99	—	1.11	0.69	1.41	0.74

Tabelle 5.19: Mehrgitter-Lösungszeit für das zweidimensionale anisotrope Problem (5.3)

vergenzraten wirken sich auch auf die für den Mehrgitterzyklus benötigte Zeit aus. Ohne Randbehandlung erfordert die Lösung mehr als doppelt so viel Zeit als mit dem schnellsten Verfahren, der Falgout-Vergröberung. Dieses Verfahren verdankt seine Geschwindigkeit der niedrigen Gesamtlevelzahl von nicht mehr als 20 Ebenen gegenüber bis zu 24 für das RS3-Verfahren. Die

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.80	0.80	0.80	0.80	0.79	0.79
2	1.06	1.11	1.09	1.22	1.04	1.10
4	1.54	2.06	1.16	1.30	1.55	1.13
8	1.72	2.24	1.32	1.45	1.76	1.31
16	2.05	4.41	1.76	1.67	1.88	1.60
32	2.80	7.17	2.39	2.09	2.56	2.33
64	4.25	17.0	3.58	2.99	3.28	2.96
128	6.11	—	6.12	4.81	5.04	4.96
256	8.63	—	9.09	9.27	7.80	7.32

Tabelle 5.20: Gesamtzeitaufwand für das anisotrope zweidimensionale Problem (5.3)



(a) Mehrgitterzeit (b) Gesamtzeitaufwand

Abbildung 5.8: Mehrgitter-Lösungszeit und Gesamtzeit für das zweidimensionale anisotrope Problem (5.3)

Falgout-Vergröberung erzielt jedoch nicht für alle Prozessorzahlen die geringsten Gesamtkosten (Tabelle 5.20), weil die Setupphase zu viel Zeit erfordert. Bei quadratischen Prozessorzahlen ist das kombinierte CGC-RS3-Verfahren schneller.

Als nächstes betrachten wir die 7-Punkte-Diskretisierung des Problems

$$-\epsilon u_{xx} - u_{yy} - u_{zz} = f \tag{5.4}$$

wiederum mit Nullrandbedingungen. Wir wählen wieder $\epsilon = 0.001$.
Tabelle 5.21 zeigt die Setupzeiten für dieses Problem. Anders als beim isotro-

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.43	0.43	0.43	0.45	0.43	0.43
2	0.54	0.56	0.54	0.63	0.55	0.56
4	0.64	1.13	0.69	0.76	0.64	0.67
8	0.79	1.33	1.06	1.04	0.80	0.84
16	0.79	1.36	1.11	1.08	0.82	0.90
32	0.95	1.53	1.32	1.35	1.02	1.20
64	1.42	2.50	2.11	2.05	1.36	1.61
128	1.90	2.57	2.89	3.41	1.79	2.18
256	2.90	3.43	4.12	5.63	2.85	3.16

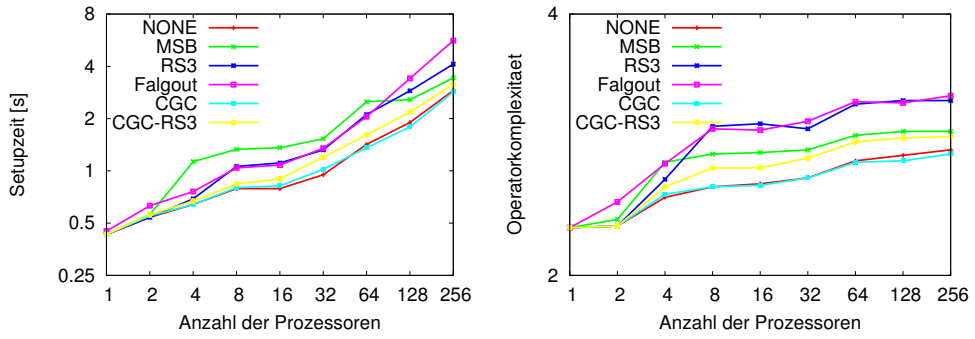
Tabelle 5.21: Setupzeit in Sekunden für das anisotrope dreidimensionale Problem (5.4)

pen Fall sehen wir hier keine so schwerwiegende Verlangsamung der Setupphase für das RS3- und das Falgout-Verfahren bei kleinen Prozessorzahlen. Der Grund hierfür liegt darin, dass in x-Richtung auf dem feinsten Level überhaupt keine starken Kopplungen vorhanden sind, die Variablen in den entsprechenden Teilrandflächen wie innere Variablen behandelt werden.
Die Operatorkomplexität 5.22 des RS3- und des Falgout-Verfahrens lie-

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	2.27	2.27	2.27	2.27	2.27	2.27
2	2.28	2.32	2.28	2.43	2.28	2.28
4	2.46	2.70	2.58	2.69	2.48	2.53
8	2.53	2.76	2.97	2.95	2.53	2.66
16	2.55	2.77	2.99	2.94	2.54	2.66
32	2.59	2.79	2.95	3.01	2.59	2.73
64	2.71	2.90	3.15	3.17	2.70	2.85
128	2.75	2.93	3.18	3.16	2.71	2.88
256	2.79	2.93	3.18	3.22	2.76	2.89

Tabelle 5.22: Operatorkomplexität für das dreidimensionale anisotrope Problem (5.4)

gen mit Werten über 2.9 bereits ab 8 Prozessoren deutlich über denjenigen der anderen Verfahren. Wie auch im isotropen Fall fügen diese Verfahren viele zusätzliche Grobgitterpunkte an den Teilgebietsrändern ein und



(a) Setupzeit (b) Operatorkomplexität

Abbildung 5.9: Setupzeit und Operatorkomplexität für das dreidimensionale anisotrope Problem (5.4)

erhöhen damit die Operatorkomplexität gegenüber dem klassischen Ruge-Stüben-Verfahren. Im Gegensatz zum zweidimensionalen Fall zeigt das MSB-Verfahren keinen starken Anstieg bezüglich der Setupzeit und der Operatorkomplexität. In diesem Fall bestehen die Teilgebietsränder aus Flächen (statt aus Linien wie im zweidimensionalen Fall), die auch starke Kopplungen innerhalb dieser Ränder aufweisen. Solche Flächen können vom MSB-Algorithmus gut vergröbert werden, insbesondere die senkrecht zur x-Achse stehenden, bei denen alle starken Kopplungen der enthaltenen Punkte innerhalb der Fläche bleiben. Anders als im zweidimensionalen Fall führt das CGC-Verfahren ohne RS3-Schritt zu einer bis zu 5% geringeren Operatorkomplexität als das Verfahren mit RS3-Randbehandlung, weil letztgenannter Algorithmus bei diesem Problem mit dem hohen Randvariablenanteil viele zusätzliche Punkte in das Grobgitter aufnimmt. Die Gitterkomplexitäten der verschiedenen Verfahren verhalten sich wie die Operatorkomplexitäten. Für klassische Ruge-Stüben-Verfahren und das CGC-Verfahren erhöht sie sich sich von 1.73 auf 1.77 bei steigender Prozessorzahl, für das CGC- und das kombinierte CGC-RS3-Verfahren auf 1.81, für das RS3-Verfahren auf 1.92 und für das Falgout-Verfahren auf bis zu 1.87.

Tabelle 5.23 zeigt die Konvergenzraten. Das schlechte Abschneiden des MSB-Verfahrens begründet sich wiederum in den $F-F$-Kopplungen über die Teilgebietsränder in y- und z-Richtung, die dieses Verfahren nicht eliminiert. Wie im zweidimensionalem Fall gilt auch hier, dass die beiden Verfahren ohne eine spezielle Randbehandlung —klassischer Ruge-Stüben-Algorithmus und CGC-Algorithmus— keine so gute Konvergenz bieten wie die Verfahren mit

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.12	0.12	0.12	0.12	0.12	0.12
2	0.12	0.12	0.12	0.12	0.12	0.12
4	0.14	0.14	0.13	0.13	0.15	0.13
8	0.19	0.21	0.13	0.15	0.18	0.14
16	0.19	0.20	0.13	0.15	0.18	0.14
32	0.19	0.20	0.14	0.15	0.19	0.15
64	0.21	0.23	0.15	0.15	0.20	0.15
128	0.20	0.23	0.15	0.15	0.20	0.15
256	0.21	0.23	0.15	0.15	0.21	0.15

Tabelle 5.23: Konvergenzraten für das dreidimensionale anisotrope Problem (5.4)

Randbehandlung, jedoch unterscheiden sich die Konvergenzraten für dieses Problem gegenüber dem RS3- und dem Falgout-Verfahren nur um 30%, da die Teilrandvariablen in zwei Raumrichtungen starke Kopplungen aufweisen und die starke Kopplung über den Teilgebietsrand hinweg einen geringeren Anteil aller starken Kopplungen ausmacht.

Wir betrachten jetzt die Mehrgitter-Lösungszeit der verschiedenen Verfah-

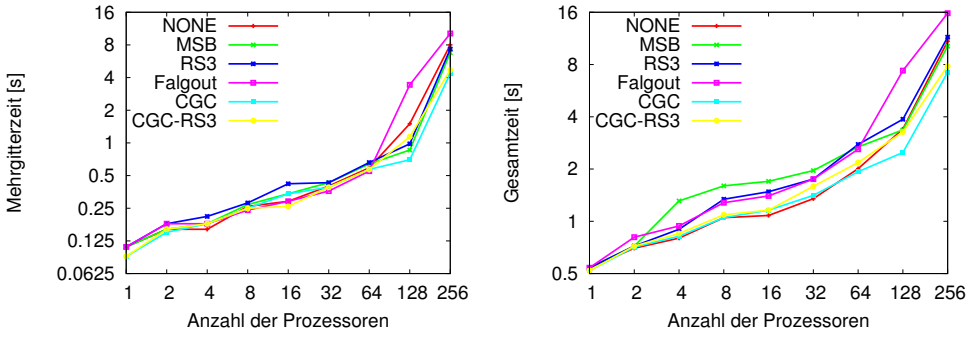
P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.11	0.11	0.11	0.11	0.09	0.09
2	0.16	0.16	0.18	0.18	0.15	0.16
4	0.16	0.18	0.21	0.18	0.18	0.18
8	0.26	0.27	0.28	0.24	0.25	0.25
16	0.29	0.34	0.42	0.29	0.34	0.26
32	0.40	0.43	0.43	0.36	0.39	0.39
64	0.59	0.64	0.66	0.55	0.57	0.57
128	1.50	0.86	0.98	3.43	0.70	1.14
256	8.01	6.78	7.34	10.2	4.37	4.69

Tabelle 5.24: Mehrgitter-Lösungszeit für das dreidimensionale anisotrope Problem (5.4)

ren. In diesem dreidimensionalem Problem mit einem großen Randvariablenanteil zeigt sich der Vorteil der CGC-Varianten und deren Gitterauswahl. Trotz schlechterer Konvergenzrate stellt sich das CGC-Verfahren als eines der schnellsten heraus, weil die beteiligten Operatoren wenig Einträge aufweisen und schnell anzuwenden sind. Die relativ langsame Kommunikation zwischen den einzelnen Knoten verursacht wiederum den Sprung bei großen Prozessorzahlen.

P	NONE	MSB	RS3	Falgout	CGC	CGC-RS3
1	0.54	0.54	0.54	0.54	0.52	0.52
2	0.70	0.72	0.72	0.81	0.71	0.72
4	0.80	1.31	0.90	0.94	0.82	0.85
8	1.05	1.60	1.34	1.28	1.05	1.09
16	1.08	1.70	1.48	1.40	1.16	1.16
32	1.35	1.96	1.75	1.75	1.41	1.59
64	2.01	2.68	2.77	2.60	1.93	2.18
128	3.40	3.36	3.87	7.38	2.49	3.27
256	10.9	10.2	11.5	15.8	7.22	7.85

Tabelle 5.25: Gesamtzeitaufwand für das dreidimensionale anisotrope Problem (5.4)



(a) Mehrgitterzeit (b) Gesamtzeitaufwand

Abbildung 5.10: Mehrgitter-Lösungszeit und Gesamtzeit für das dreidimensionale anisotrope Problem (5.4)

5.3 Navier-Stokes-Gleichungen

Wir wenden uns jetzt einer Anwendung aus dem Bereich der Strömungsmechanik zu. Wir verwenden das AMG, um die Druckgleichung einer inkompressiblen Zweiphasenströmung in einem dreidimensionalen, quaderförmigen Gebiet zu lösen. Eine Zweiphasenströmung zeichnet sich dadurch aus, dass innerhalb des Simulationsgebietes zwei Fluide mit unterschiedlicher Dichte vorhanden sind, deren Lage zueinander sich im Laufe der Zeit ändert. Der hierbei auftretende Dichtesprung verschlechtert die Kondition der Systemmatrix für die diskretisierte Druckgleichung.

5.3.1 Mathematische Modellierung einer Zweiphasenströmung

Das mathematische Modell einer Zweiphasenströmung wird in [Cro02] beschrieben. Wir gehen aus von einem dreidimensionalem, quaderförmigen Gebiet Ω, welches in Abhängigkeit der Zeit $t \in [t_o, t_{end}]$ disjunkt in zwei Teilgebiete zerlegt wird

$$\Omega = \Omega_g(t) \dot{\cup} \Omega_l(t).$$

In jedem dieser Teilgebiete $\Omega_i(t)$ befindet sich ein Fluid mit einer spezifischen Dichte ρ_i, wie es in Abbildung 5.11 für zwei Raumdimensionen dargestellt ist. Es findet kein Austausch von Material zwischen den Gebieten statt, jedoch verändern sich die Teilgebiete im Laufe der Zeit $t \in [t_0, t_{end}]$. Den *freien Rand* zwischen den beiden Gebieten bezeichnen wir mit

$$\Gamma_f(t) := \bar{\Omega}_g(t) \cap \bar{\Omega}_l(t)$$

Auf jedem der beiden Teilgebiete $i \in \{g, l\}$ stellen wir nun die Impulsgleichung in integraler Form

$$\int_{\Omega_i(t)} \rho_i \frac{D\vec{u}_i}{Dt} d\vec{x} = \int_{\partial\Omega_i(t)} \mathrm{T}_i \cdot \vec{n} dF + \int_{\Omega_i} \rho_i \vec{g} d\vec{x} \tag{5.5}$$

und die Kontinuitätsgleichung für inkompressible Fluide in differenzieller Form

$$\nabla \cdot \vec{u}_i = 0 \ \in \Omega_i$$

auf. Dabei bezeichnet

$$\mathrm{T}_i := -pI + \mu_i \mathrm{D}_i$$

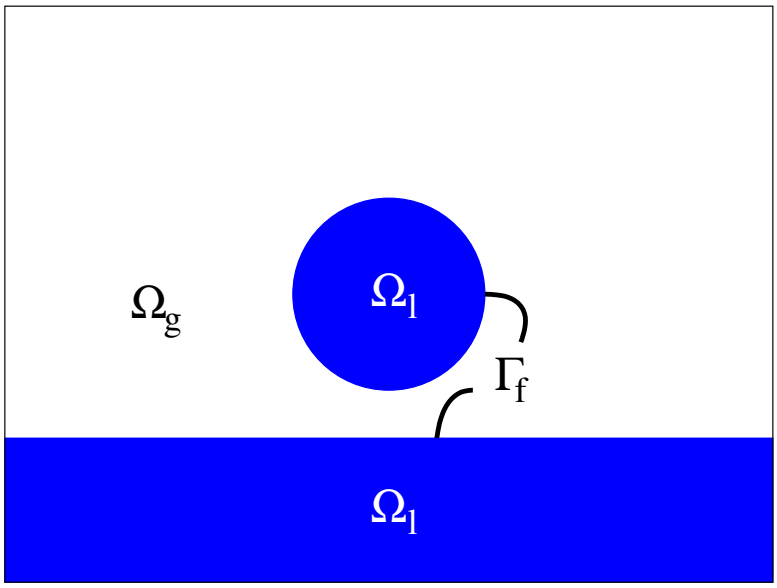


Abbildung 5.11: Simulationsgebiet mit zweiphasiger Strömung

den *Spannungstensor* für inkompressible, viskose Newtonsche Fluide. D_i ist der *Deformationstensor*

$$D_i := \nabla \vec{u}_i + (\nabla \vec{u}_i)^T ,$$

I die Einheitsmatrix und μ_i die *dynamische Viskosität*, welche innerhalb eines Teilgebietes Ω_i jeweils als konstant vorausgesetzt wird.
Die Kopplung der Gleichungssysteme für die beiden Phasen geschieht über die Oberflächenspannung am freien Rand $\Gamma_f(t)$. An diesem Rand gelten folgende Bedingungen:

- *Regularitätsbedingung:* Die Geschwindigkeiten sind stetig, d.h.$\vec{u}_l = \vec{u}_g$ auf $\Gamma_f(t)$.

- *Kinematische Bedingung:* Die freie Oberfläche bildet eine scharfe Trennfläche, durch die keine Masse fließt.

- *Dynamische Bedingung:* Die Bilanzierung der viskosen Reibungskräfte entlang des freien Randes ist gegeben durch

$$[T] \cdot \vec{n} = \sigma \kappa \vec{n}. \tag{5.6}$$

  Dabei bezeichne σ den Koeffizienten der Oberflächenspannungskraft, $\vec{n}$ den Normalenvektor sowie κ die lokale Krümmung der Grenzfläche und

$[\mathrm{T}] := T_l - T_g$ den Sprung über den freien Rand. Die viskosen Spannungskräfte in Normalenrichtung sind also proportional zur mittleren Krümmung.

Wir addieren die Impulsgleichungen (5.5) für die beiden Teilgebiete, transformieren das Randintegral entlang des festen Gebietsrandes und setzen am freien Gebietsrand $\Gamma_f(t)$ die Bedingung (5.6) ein. Wir erhalten

$$\int_\Omega \rho \frac{D\vec{u}}{Dt} d\vec{x} = \int_\Omega \nabla \cdot \mathrm{T}s\vec{x} - \int_{\Gamma_f(t)} \sigma\kappa\vec{n} dF + \int_\Omega \rho\vec{g} d\vec{x}.$$

Die Randbedingungen der freien Oberfläche sind in dieser Beschreibung bereits enthalten. Am äußeren Gebietsrand wählen wir je nach physikalischer Situation eine der folgenden Bedingungen:

- *Haftbedingungen:* Das Fluid haftet am Rand Γ_H, d.h. $\vec{u}|_{\Gamma_H} = 0$.

- *Fixe Ein-/Ausströmbedingungen:* Das Fluid strömt mit einer vorgegebenen Geschwindigkeit über den Rand Γ_{EA} ein oder aus, d.h. $\vec{u}|_{\Gamma_{EA}} = \vec{u}_o$.

- *Rutschbedingungen:* Das Fluid gleitet reibungsfrei am Rand Γ_R entlang. Die Geschwindigkeiten entlang der Tangentenvektoren $\vec{s}$ bleiben erhalten.

$$(\vec{u} \cdot \vec{n})|_{\Gamma_R} = 0, \ (\partial_n(\vec{u} \cdot \vec{s}))|_{\Gamma_R} = 0$$

- *Natürliche Randbedingungen:* Die Fluidgeschwindigkeit ändert sich nicht in Normalenrichtung des Randes Γ_A, d.h. $(\partial_n \vec{u})|_{\Gamma_A} = 0$. (Hierbei muss wegen der Massenerhaltung darauf geachtet werden, dass der Massenausfluss gleich dem Masseneinfluss ist!)

Die Modellierung des freien Randes $\Gamma_f(t)$ erfolgt mittels einer *Level-Set-Funktion* $\phi(\vec{x}, t)$, deren Nullstellenmenge zu allen Zeiten $t \in [t_0, t_{end}]$ gleich $\Gamma_F(t)$ ist. ([Cro02], Abschnitt 4.2). Mit Hilfe dieser Funktion lässt sich die Impulsgleichung in differenzieller Form umschreiben zu

$$\rho(\phi)\left(\partial_t \vec{u} + \nabla \cdot (\vec{u} \otimes \vec{u})\right) + \nabla p = \nabla \cdot (\mu(\phi)\mathrm{D}) - \sigma\kappa(\phi)\delta(\phi)\nabla\phi + \rho(\phi)\vec{g},$$

wobei $\delta : \mathbb{R} \to \bar{\mathbb{R}}$ die eindimensionale Dirac-Distribution bezeichnet. (Herleitung siehe [Cro02], Abschnitt 4.4).

5.3.2 Diskretisierung der zweiphasigen Navier-Stokes-Gleichungen

Die numerische Lösung der zweiphasigen Navier-Stokes-Gleichungen erfordert die Diskretisierung in Raum und Zeit. Das Simulationsgebiet Ω wird dazu uniform in quaderförmige Zellen $\Omega_{i,j,k}$ aufgeteilt, an deren Ränder die Geschwindigkeitsvektoren

$$\vec{u} = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

in Normalenrichtung ausgewertet werden (siehe [Cro02], Abschnitt 3.2).
Eine äußere Iteration behandelt die Zeitschritte $t = t_0, \ldots, t_{end}$ ausgehend von einem vorgegebenem Geschwindigkeitsfeld $\vec{u}|_{t=t_0} = \vec{u}_0$ zum Zeitpunkt t_0 mit der Zeitschrittweite δt. Der Übergang vom Zeitschritt n zum Zeitschritt $n+1$ besteht dabei aus folgenden Schritten:
Zuerst bestimmen wir die neue Schätzgeschwindigkeit

$$\begin{aligned}\vec{u}^* = \vec{u}^n &+ \delta t \left(-\nabla \cdot (\vec{u}^n \otimes \vec{u}^n) + \vec{g}\right) \\ &+ \delta t \left(+ \frac{1}{\rho(\phi^n)} \left(\nabla \cdot (\mu(\phi^n) \mathrm{D}^n) - \sigma \kappa(\phi^n) \delta(\phi^n) \nabla \phi^n\right)\right)\end{aligned}$$

und ermitteln anschließend eine neue Level-Set-Funktion ϕ^{n+1}, die die Lage des Randes $\Gamma_f(t_n)$ im Zeitschritt n beschreibt (siehe [Cro02], Abschnitt 4.6).
Wir müssen jetzt zur Bestimmung der neuen Geschwindigkeit $\vec{u}^{n+1}$ noch den Druckgradienten ∇p^{n+1} berücksichtigen. Dazu bemerken wir, dass $\vec{u}^{n+1}$ das Gleichungssystem

$$\begin{aligned}\frac{\vec{u}^{n+1} - \vec{u}^*}{\delta t} + \frac{\nabla p^{n+1}}{\rho(\phi^{n+1})} &= 0 \\ \nabla \cdot \vec{u}^{n+1} &= 0\end{aligned}$$

erfüllen soll. Wenn wir die Divergenz aus der ersten Gleichung, bilden und die zweite darin einsetzen, so sehen wir, dass die Lösung des Gleichungssystems die numerische Lösung der Poissongleichung

$$\nabla \cdot \left(\frac{1}{\rho(\phi^{n+1})} \nabla p^{n+1} \right) = \nabla \cdot \frac{\vec{u}^*}{\delta t} \tag{5.7}$$

erfordert. Ist der Druckwert p^{n+1} auf diese Weise berechnet worden, so ergibt sich die Geschwindigkeit im nächsten Zeitschritt gemäß

$$\vec{u}^{n+1} = \vec{u}^* - \frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1}.$$

Wir diskretisieren die Gleichung (5.7) mit einem 7-Punkte-Schema der Form

$$\begin{aligned}\left[\nabla\cdot\frac{1}{\rho(\phi)}\nabla p^{n+1}\right]_{i,j,k} &= \frac{1}{(\delta x)^2}\left(\frac{p^{n+1}_{i+1,j,k}-p^{n+1}_{i,j,k}}{\rho(\phi_{i+\frac{1}{2},j,k})}-\frac{p^{n+1}_{i,j,k}-p^{n+1}_{i-1,j,k}}{\rho(\phi_{i-\frac{1}{2},j,k})}\right)\\ &+ \frac{1}{(\delta y)^2}\left(\frac{p^{n+1}_{i,j+1,k}-p^{n+1}_{i,j,k}}{\rho(\phi_{i,j+\frac{1}{2},k})}-\frac{p^{n+1}_{i,j,k}-p^{n+1}_{i,j-1,k}}{\rho(\phi_{i,j-\frac{1}{2},k})}\right)\\ &+ \frac{1}{(\delta x)^2}\left(\frac{p^{n+1}_{i,j,k+1}-p^{n+1}_{i,j,k}}{\rho(\phi_{i,j,k+\frac{1}{2}})}-\frac{p^{n+1}_{i,j,k}-p^{n+1}_{i,j,k-1}}{\rho(\phi_{i,j,k-\frac{1}{2}})}\right)\end{aligned}$$

und die rechte Seite durch zentrale Differenzen

$$[\nabla\cdot\vec{u}^*]_{i,j,k}=\frac{u^*_{i+\frac{1}{2},j,k}-u^*_{i-\frac{1}{2},j,k}}{\delta x}+\frac{v^*_{i,j+\frac{1}{2},k}-v^*_{i,j-\frac{1}{2},k}}{\delta y}+\frac{w^*_{i,j,k+\frac{1}{2}}-w^*_{i,j,k-\frac{1}{2}}}{\delta z}$$

und homogene Neumann-Randwerte. Es ergibt sich ein dünnbesetztes Gleichungssystem.
Die Kondition der Matrix hängt von der Größe des Dichtesprunges, d.h. dem Sprung von $\rho(\phi^n)$ über Γ_f ab. Sie wird schlechter, je größer dieser Unterschied wird. In dieser Hinsicht stellt das Problem eine interessante Herausforderung für den AMG-Vorkonditionierer dar. Die Größe des Diskretisierungsgebietes erfordert außerdem eine parallele Vorgehensweise.

5.3.3 Numerische Ergebnisse

Wir betrachten in einer Box der Grösse $2 \times 2 \times 1$ cm einen kugelförmigen Wassertropfen, der ähnlich der in Abbildung 5.11 dargestellten Situation in ein Wasserbecken fällt. An der unteren Seite der Box gelten Haftrandbedingungen, an den übrigen Seiten Rutschbedingungen. Die physikalischen Parameter lauten wie folgt:

- Oberflächenspannung $\sigma = 0.07275$,
- Dynamische Viskosität des Wassers $\mu_l = 1.787e^{-3}$,
- Dynamische Viskosität der Luft $\mu_l = 1.71e^{-5}$,
- Dichte des Wassers $\rho_l = 999.9$,
- Dichte der Luft $\rho_g = 1.293$.

Die Abbildungen 5.12 bis 5.16 zeigen die von den verschiedenen Verfahren erzeugten Grobgitter. Die Größe des Diskretisierungsgebietes beträgt $64 \times$

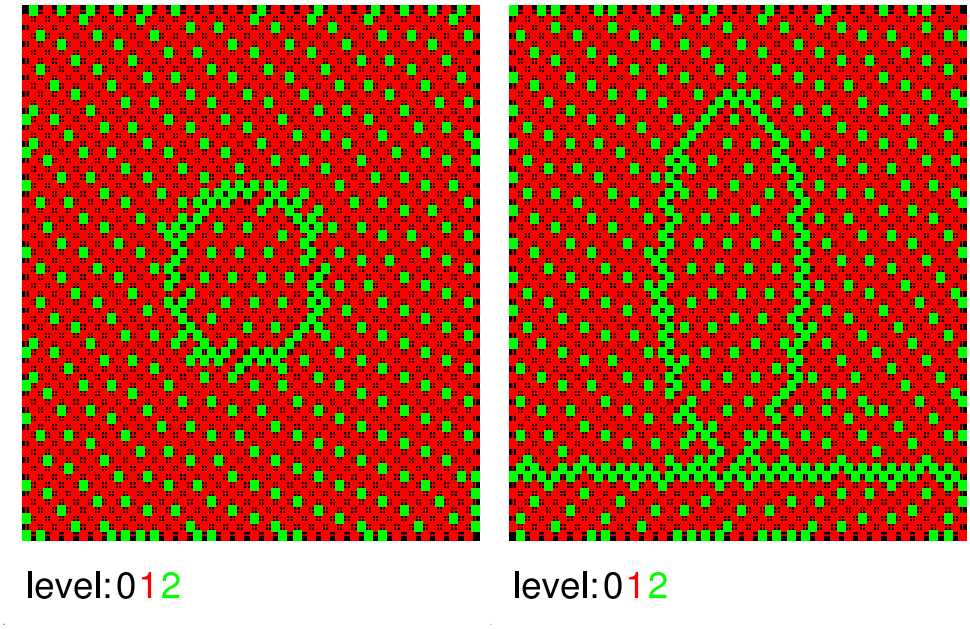


Abbildung 5.12: Vergröberung auf einem Prozessor

64×64 Punkte. Die jeweils linke Abbildung zeigt einen $x-y$-Querschnitt an der 31-ten Stelle in z-Richtung, die rechte Abbildung einen $x-z$-Querschnitt an der Stelle $y = 31$.
In Abbildung 5.12, welcher die sequentielle Vergröberung zeigt, sind die Umrisse des Wassertropfens und der Wasseroberfläche zu erkennen. Die Phasengrenzen zeichnen sich dadurch aus, dass die Diskretisierungssterne an diesen Stellen starke Anisotropien aufweisen. Dies führt zu einer Aufnahme vieler Punkte in das Grobgitter. Im vertikalen Schnitt ist der Tropfen aufgrund der verschiedenen Schrittweiten in x- und z-Richtung verzerrt dargestellt.
Abbildung 5.13 zeigt, dass der klassische Ruge-Stüben-Algorithmus angewendet auf das verteilte Gebiet keine zueinander passende Gitter erzeugt. An den Teilgebietsgrenzen sind deutlich die $C-C$- und $F-F$-Kopplungen auf Level 2 zu erkennen.
Die Abbildungen 5.14 und 5.15 zeigen, dass das RS3-Verfahren und der Falgout-Algorithmus viele zusätzliche Punkte an den Teilgebietsrändern einfügen. Der Falgout-Algorithmus wählt außerdem —bedingt durch die CLJP-artige Vorgehensweise— entlang des freien Randes ein breites Band von Grobgittervariablen aus.
Das CGC-Verfahren (Abbildung 5.16) wählt weitgehend zueinander „passende“ Gitter aus. Lediglich das in der $x-y$-Sicht links unten gezeigte Gitter weist eine Verschiebung gegenüber dem in Abbildung 5.12 gezeigtem „idealem“ Gitter auf.

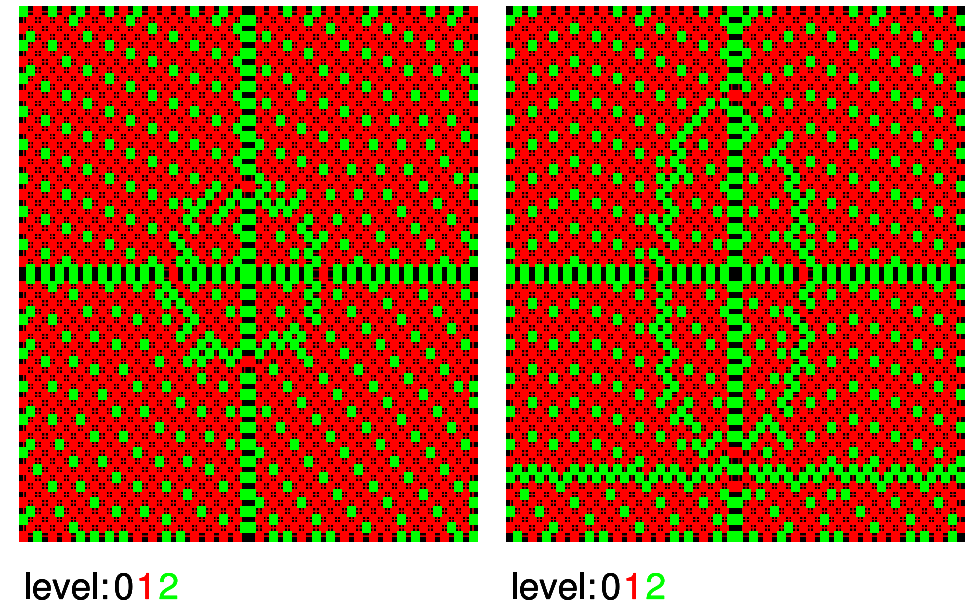


Abbildung 5.13: 4 Prozessoren, keine Randbehandlung

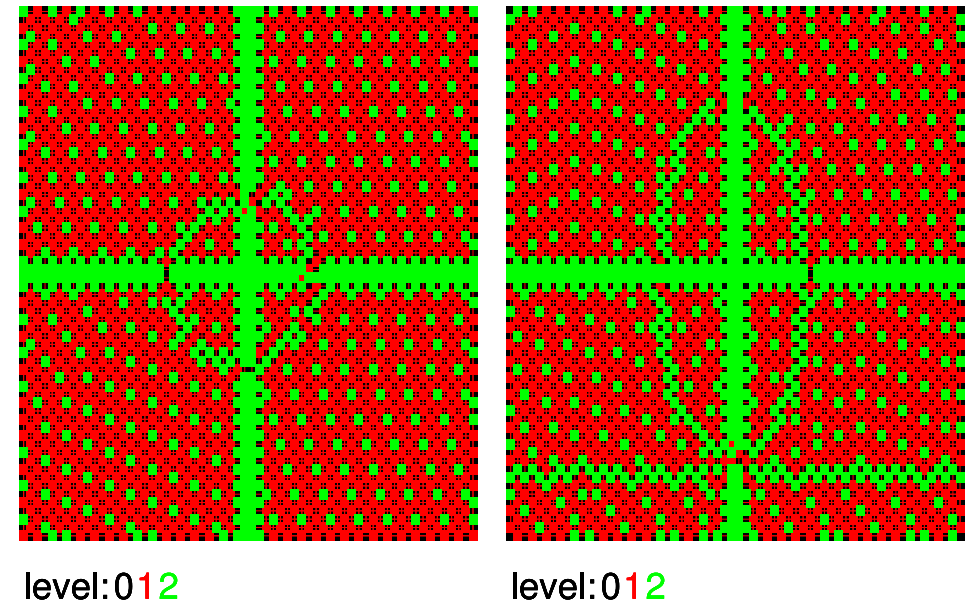


Abbildung 5.14: 4 Prozessoren, RS3-Algorithmus

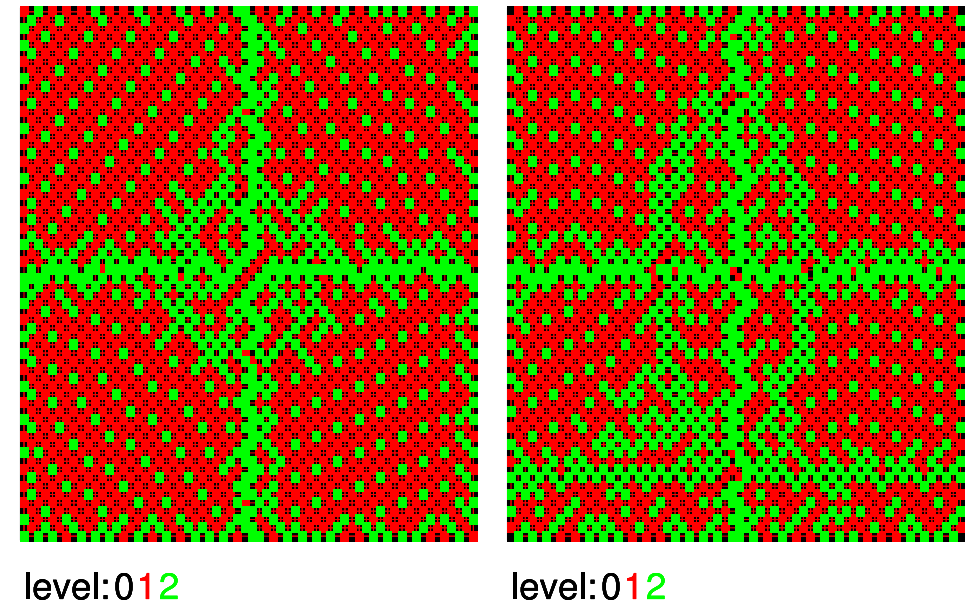


Abbildung 5.15: 4 Prozessoren, Falgout-Algorithmus

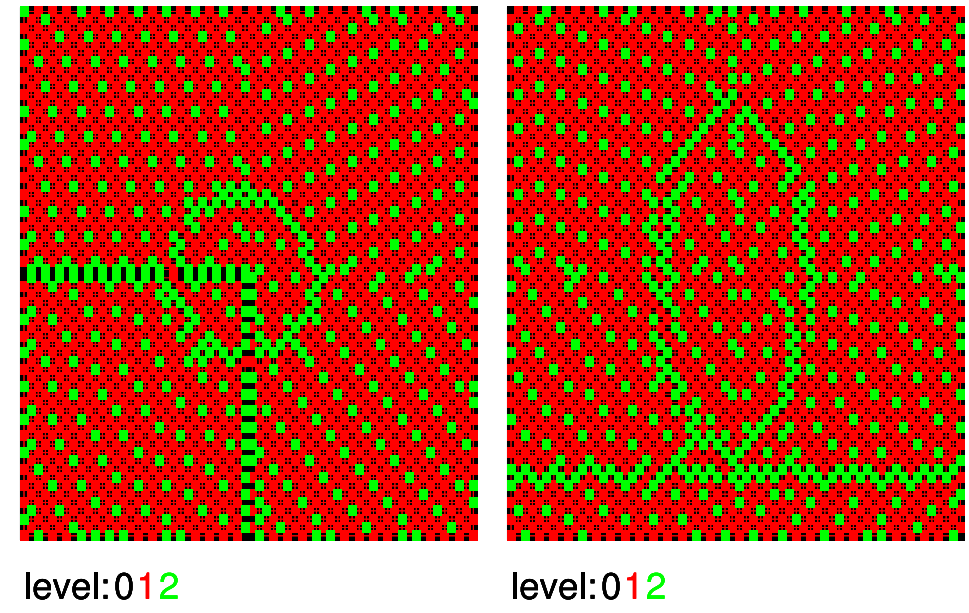


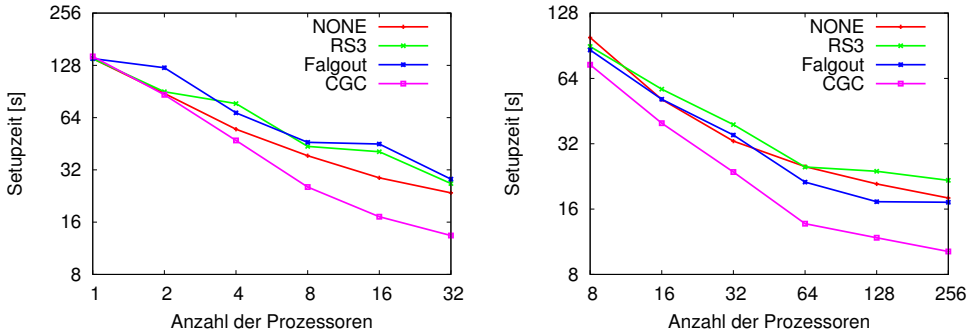
Abbildung 5.16: 4 Prozessoren, CGC-Algorithmus

Wir betrachten im Folgenden zwei Problemgrößen: Einmal ein Gebiet von $64 \times 64 \times 64$ Punkte. Dieses Problem lösen wir auf 1 bis 32 Prozessoren des Parallelrechners Parnass2. Das zweite Problem hat die Größe $128 \times 128 \times 128$ und wird auf dem IBM-Cluster berechnet.
Die Setupzeit ist in Tabelle 5.26 dargestellt. Wir sehen für das CGC-Verfahren

P	NONE	RS3	Falgout	CGC
1	140	140	140	144
2	88.2	90.2	124	86.6
4	54.9	77.1	68.2	47.3
8	38.6	43.7	46.1	25.5
16	28.8	40.7	45.1	17.2
32	23.6	26.7	28.3	13.4

P	NONE	RS3	Falgout	CGC
8	98.7	89.8	86.5	74.0
16	51.2	57.1	51.3	39.8
32	32.9	39.2	35.1	23.7
64	25.1	25.0	21.3	13.7
128	20.9	23.9	17.3	11.8
256	18.0	21.7	17.2	10.2

Tabelle 5.26: Setupzeit in Sekunden für die Druckgleichung (5.7) auf $64 \times 64 \times 64$ (links) und $128 \times 128 \times 128$ (rechts) Punkten



(a) $64 \times 64 \times 64$ Punkte (b) $128 \times 128 \times 128$ Punkte

Abbildung 5.17: Setupzeit in Sekunden für die Druckgleichung (5.7)

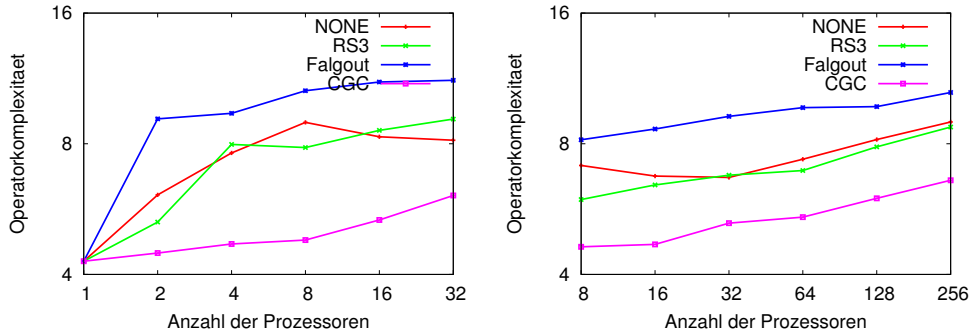
bei steigender Prozessorzahl eine zunehmende Beschleunigung gegenüber dem klassischen Ruge-Stüben-Algorithmus, welche bis zu 45% betragen kann. Die besser zusammenpassenden Gitter des CGC-Algorithmus ermöglichen eine schnellere Berechnung des Grobgitteroperators. So beträgt auf 64 Prozessoren der gesamte (über alle Level addierte) Zeitaufwand hierfür 17 Sekunden für den sequentiellen Algorithmus, jedoch nur 8 Sekunden für den CGC-Algorithmus.
Die Begründung für die hohe Setupzeit des RS3- und des Falgout-Algorithmus liegt vor allem in der teuren Randbehandlung, die die zur Vergröberung notwendigen Zeit gegenüber dem sequentiellen Algorithmus annähernd verdop-

pelt. Die zusätzlichen Vergröberungsdurchläufe des CGC-Algorithmus fallen dagegen kaum ins Gewicht.
Beim Übergang von 128 zu 256 Prozessoren ist kaum noch eine Beschleunigung möglich. Dieses Phänomen ist in der Rechnerarchitektur des IBM-Clusters begründet.

P	NONE	RS3	Falgout	CGC
1	4.29	4.29	4.29	4.29
2	6.10	5.28	9.13	4.48
4	7.62	7.97	9.40	4.70
8	8.96	7.84	10.6	4.80
16	8.30	8.59	11.1	5.34
32	8.15	9.12	11.2	6.08

P	NONE	RS3	Falgout	CGC
8	7.13	5.95	8.17	4.63
16	6.74	6.43	8.65	4.69
32	6.69	6.77	9.25	5.25
64	7.37	6.94	9.69	5.42
128	8.18	7.87	9.74	5.99
256	8.98	8.74	10.5	6.59

Tabelle 5.27: Operatorkomplexität für die Druckgleichung (5.7) auf $64 \times 64 \times 64$ (links) und $128 \times 128 \times 128$ (rechts) Punkten



(a) $64 \times 64 \times 64$ Punkte (b) $128 \times 128 \times 128$ Punkte

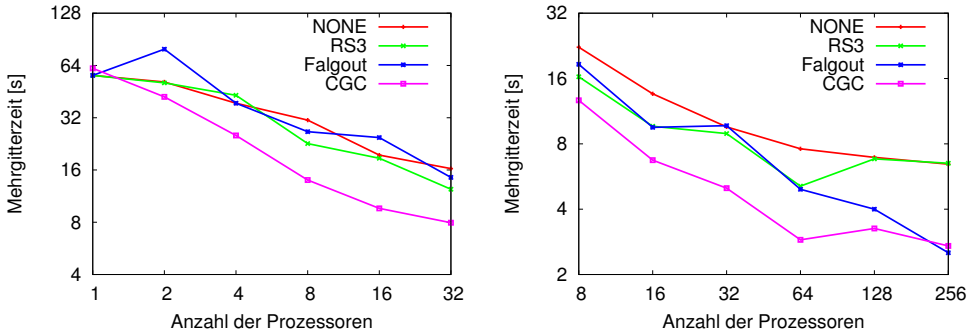
Abbildung 5.18: Operatorkomplexität für die Druckgleichung (5.7)

Bei der Operatorkomplexität (Tabelle 5.27) fällt ebenfalls wieder der Vorsprung für das CGC-Verfahren auf, welches zwischen 1 und 16 Prozessoren einen Anstieg um rund 25% aufweist, während dieser Wert für das RS3-Verfahren verdoppelt wird und für das Falgout-Verfahren noch darüber hinaus geht. Hier zeigt sich wieder die auch aus Abbildung 5.15 ersichtliche Effekt der CLJP-Vergröberung in den Randbereichen, welche sich zudem in einer erhöhten Gitterkomplexität von 2.05 für 16 Prozessoren niederschlägt. Das CGC-Verfahren weist lediglich einen moderaten Anstieg von 1.7 auf 1.77 beim Übergang von 1 zu 16 Prozessoren auf.
Die Konvergenzraten der verschiedenen Verfahren sind nahezu identisch,

P	NONE	RS3	Falgout	CGC
1	56.1	56.1	56.1	61.6
2	51.4	50.8	79.4	42.2
4	38.8	43.1	38.8	25.3
8	31.0	22.7	26.6	14.0
16	19.5	18.7	24.6	9.62
32	16.3	12.4	14.5	7.95

P	NONE	RS3	Falgout	CGC
8	22.3	16.3	18.6	12.7
16	13.6	9.64	9.53	6.73
32	9.58	8.92	9.70	5.00
64	7.59	5.11	4.94	2.89
128	6.93	6.82	4.00	3.26
256	6.45	6.51	2.52	2.71

Tabelle 5.28: Mehrgitter-Lösungszeit für die Druckgleichung (5.7) auf $64 \times 64 \times 64$ (links) und $128 \times 128 \times 128$ (rechts) Punkten in Sekunden



(a) $64 \times 64 \times 64$ Punkte

(b) $128 \times 128 \times 128$ Punkte

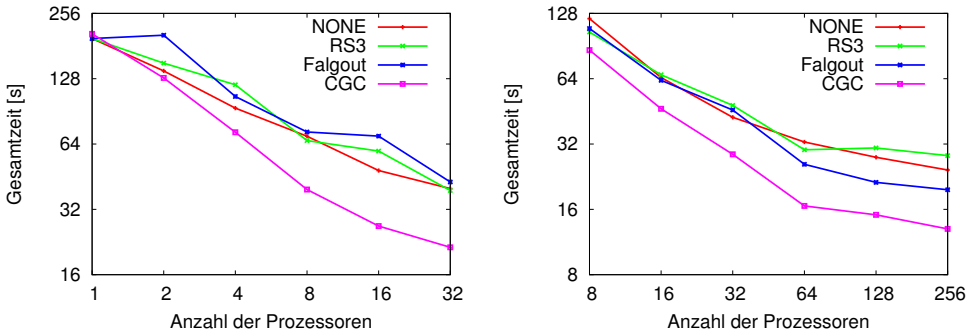
Abbildung 5.19: Mehrgitter-Lösungszeit für die Druckgleichung (5.7) in Sekunden

sie erhöhen sich —für beide Problemegrößen— bei steigender Prozessorzahl lediglich von 0.06 auf 0.07. Die zur Lösung benötigten Zeiten unterscheiden sich aber erheblich (Tabelle 5.28). Das CGC-Verfahren erzielt eine bis zu doppelt so schnelle Lösung wie das RS3-Verfahren und der Falgout-Algorithmus. Letzterer benötigt sogar auf zwei Prozessoren mehr Zeit als auf einem Prozessor, weil die vielen Punkte auf den feineren Gittern die parallele Matrixmultiplikation (erforderlich für die Glättung und die Bestimmung des Residuums) verzögern. Die Verlangsamung beim Übergang von 64 auf 128 Prozessoren des RS3- und des CGC-Verfahrens ist wiederum auf die Rechnerarchitektur zurückzuführen. Bei größeren Prozessorzahlen (ab 64) ermöglicht auch der Falgout-Algorithmus eine schnelle Lösung, hier ermöglichen die vielen Randpunkte eine schnell durchzuführende Interpolation. In der Gesamtzeit (Tabelle 5.29) dominiert jedoch die für die Setupphase benötigte Zeit, so dass dort ein klarer Vorsprung für das CGC-Verfahren erkennbar ist.

P	NONE	RS3	Falgout	CGC
1	196	196	196	206
2	139	151	203	129
4	93.7	120	106	72.5
8	69.5	66.4	72.7	39.5
16	48.3	59.4	69.6	26.8
32	39.9	39.1	42.8	21.4

P	NONE	RS3	Falgout	CGC
8	121	105	109	86.7
16	64.8	66.7	62.9	46.5
32	42.5	48.2	45.9	28.7
64	32.7	30.1	25.8	16.6
128	27.8	30.7	21.3	15.1
256	24.3	28.3	19.7	13.0

Tabelle 5.29: Gesamtzeitaufwand in Sekunden für die Druckgleichung (5.7) auf $64 \times 64 \times 64$ (links) und $128 \times 128 \times 128$ (rechts) Punkten



(a) $64 \times 64 \times 64$ Punkte (b) $128 \times 128 \times 128$ Punkte

Abbildung 5.20: Gesamtzeitaufwand in Sekunden für die Druckgleichung (5.7)

Kapitel 6

Zusammenfassung und Ausblick

In dieser Arbeit haben wir ein neues paralleles algebraisches Mehrgitterverfahren vorgestellt. Insbesondere sind wir dabei auf die parallele Vergröberung eingegangen, wobei der Aspekt einer effizienten parallelen Aufstellung des Interpolationsoperators und des Grobgitteroperators auch nicht vernachlässigt werden darf.
Nach einer Einführung in algebraische Mehrgitterverfahren und parallele Mehrgitterverfahren haben wir uns zunächst mit der Aufstellung dieser Operatoren beschäftigt. Hier stellte besonders die parallele Aufstellung des Grobgitteroperators eine besondere Herausforderung dar. Bei der Aufstellung der Interpolation haben wir uns für einen gemischten Ansatz aus direkter und Standardinterpolation entschieden, welcher schneller als die parallele Standardinterpolation durchführbar ist, jedoch nicht die schlechte Konvergenzeigenschaft der direkten Interpolation aufweist.
Für die Konstruktion eines parallelen Vergröberungsalgorithmus haben wir zunächst die bestehenden Parallelisierungsansätze untersucht. Als erstes haben wir die Varianten des Subdomain Blockings besprochen, welche die Vergröberungsprozesse auf den beteiligten Prozessoren entkoppelten. Diese Entkoppelung führt jedoch zu einem schlechten Konvergenzverhalten.
Ein anderer Ansatz besteht in der Behandlung der Prozessorteilgebietsränder nach erfolgter Vergröberung im Teilgebietsinneren. Die als RS3-Phase bezeichnete Randbehandlung besteht darin, im Falle einer drohenden instabilen Interpolation zusätzliche Punkte in das Grobgitter aufzunehmen. Diese Methode führt zwar zu einer guten Konvergenzrate des Verfahrens, jedoch stellt sich die Randbehandlung als sehr kommunikationsintensiv heraus und erhöht in ungünstigen Situationen die Anzahl der Grobgitterpunkte und damit den Speicherverbrauch des Verfahrens erheblich.
Das CLJP-Verfahren stellt eine Möglichkeit dar, ein Grobgitter parallel zu wählen. Ein solches Grobgitter weist jedoch zu viele Grobgitterpunkte auf,

so dass dieses Verfahren zu einem viel zu hohen Speicherverbrauch führt. Wird es —im Rahmen des Falgout-Algorithmus— nur zur Randbehandlung eingesetzt, so erhalten wir ähnliche Ergebnisse wie mit dem RS3-Verfahren. Wir haben mit Coarse-Grid Classification eine neue Methode entwickelt, die ungünstige Situationen am Teilgebietsrand weitgehend vermeidet. Dazu müssen wir zunächst mehrere potenzielle Grobgitter auf jedem Teilgebiet auswählen, aus denen dann ein passendes Grobgitter für das gesamte Diskretisierungsgebiet zusammengesetzt werden soll. Hierzu definieren wir einen gewichteten, gerichteten Graphen, der die Beziehung der einzelnen potenziellen Grobgitter zueinander beschreibt. Das Verfahren wählt nun für jedes Prozessorteilgebiet ein Grobgitter so aus, dass eine Randbehandlung, wenn überhaupt, nur für wenige Punkte notwendig ist und in vereinfachter Form, d.h. mit geringem Kommunikationsaufwand, durchgeführt werden kann.
Wir haben die Vergröberungsstrategien zuerst anhand einiger Modellprobleme miteinander und mit dem klassischen Ruge-Stüben-Vergröberungsalgorithmus angesetzt auf die einzelnen Prozessorteilgebiete verglichen. Dabei fielen sofort die Nachteile der subdomain-blocking-Varianten und des CLJP-Verfahrens auf, während das RS3- und das Falgout-Verfahren bei einem hohen Randvariablenanteil eine teure Setupphase aufwiesen. Die Setupphase des CGC-Verfahren lief wesentlich schneller ab, erzielte aber dieselben Konvergenzraten. Vor allem bei dreidimensionalen Problemen zeichnete sich der CGC-Algorithmus durch einen geringeren Speicherverbrauch aus. Schließlich haben wir das RS3-Verfahren, das Falgout-Verfahren und das CGC-Verfahren beim Einsatz als Löser für die Druckgleichung eines Strömungslösers getestet. Hier zeichnete sich das CGC-Verfahren durch eine besonders schnelle Setupphase und einen geringen Speicherverbrauch gegenüber den anderen Parallelisierungsstrategien aus.

Die Entwicklung paralleler algebraischer Mehrgitterverfahren ist hiermit nicht abgeschlossen. Noch immer ist kein Verfahren in der Lage, auf einem verteilten Diskretisierungsgebiet genau dasselbe Gitter zu produzieren, welches auf demselben Diskretisierungsgebiet von einem Prozessor erzeugt werden würde. Dabei müssen allerdings die Besonderheiten der Glättung und der Interpolation an den Teilgebietsrändern berücksichtigt werden. In dieser Arbeit haben wir uns mit dem Block-Jacobi-Verfahren und der direkten Interpolation über Teilgebietsränder hinweg für einfach anzuwendende Methoden entschieden, jedoch erfordern diese unter Umständen eine spezielle Lage der Grobgitterpunkte an den Teilgebietsrändern. Ist ein uniformes Gitter über alle Prozessorteilgebiete hinweg gegeben, so muss unter Umständen ein aufwändiger Interpolationsoperator verwendet werden, während die Glättung eine bestimmte Durchlaufreihenfolge erfordert.

Eine interessante Anwendungsmöglichkeit stellt die Verwendung des parallelen AMG als Löser für die Druckgleichung im parallelen Strömungslöser für dreidimensionale Zweiphasenfluide ([Cro02]) dar. Im letzten numerischen Beispiel haben wir sehr gute Resultate mit dem CGC-Verfahren erzielt, jedoch ist die Setupphase verhältnismäßig teuer. Es muss also untersucht werden, inwieweit auf die erneute Durchführung der Setupphase für jeden einzelnen Zeitschritt verzichtet werden kann, wenn sich die geometrische Konfiguration von Zeitschritt zu Zeitschritt nur geringfügig ändert.
Eine Erweiterungsmöglichkeit des parallelen AMG ist die Verbindung mit dem Block-AMG ([Oel01], [GOS03]). Das Block-AMG ist ein algebraisches Mehrgitterverfahren zur Lösung von *Systemen* partieller Differenzialgleichungen, wie sie zum Beispiel in der Elastizitätstheorie auftreten. Bei Systemen von partiellen Differenzialgleichungen tauchen im Hinblick auf das AMG zwei Probleme auf. Einerseits können die Systemmatrizen für die einzelnen Gleichungen zu unterschiedlichen Wahlen für das Grobgitter führen, andererseits weist die zusammengesetzte Systemmatrix keine M-Matrix-Eigenschaft auf und herkömmliche Relaxationsverfahren verlieren damit ihre Glättungseigenschaften. Das Block-AMG-Verfahren löst diese Probleme, indem die jeweils zu einem Gitterpunkt gehörenden Unbekannten in Blöcken zusammengefasst werden. Die starken Kopplungen zwischen zwei Blöcken i und j werden über eine Matrixnorm des Blockes A_{ij} definiert. Die Aufstellung der Interpolation geschieht ebenfalls über die Blöcke der Systemmatrix. Als Glätter wird ein Block-Gauß-Seidel-Verfahren verwendet.

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. Griebel für die Überlassung des Themas und die Betreuung während der Arbeit bedanken. Bei Roberto Croce und Martin Engel bedanke ich mich für die Bereitstellung der Beispiele aus dem Navier-Stokes-Code. Schließlich seien vor allem Daniel Oeltz und Marc Alexander Schweitzer gedankt, die mir viele Anregungen zur Diplomarbeit gegeben haben und immer mit Rat und Tat zur Seite standen.

Verzeichnis der Programme

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ABDP] Alcouffe, R.E., Brandt, A., Dendy, J.E., and Painter, J.W. The multi-grid method for the diffusion equation with strongly discontinuous coefficients. *SIAM J. Sci. Stat. Comput.*, 2:430–454.

[BBG+03] Satish Balay, Kris Buschelman, William Gropp, Dinesh Kaushik, Lois Curfman McInnes Matt Knepley, Barry Smith, and Hong Zhang. *PETSc Users Manual*. Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, January 2003. ANL-95/11 - Revision 2.1.5.

[BMR82] Brandt, A., McCormick, S.F., and Ruge, J. Algebraic Multigrid (AMG) for automatic multigrid solution with application to geodetic computations, 1982.

[BMR84] Brandt, A., McCormick, S.F., and Ruge, J. Algebraic Multigrid (AMG) for sparse matrix equations. In D.J. Evans, editor, *Sparsity and its Applications*, pages 257–284. Cambridge University Press, Cambridge, 1984.

[Bra86] Brandt, A. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comput.*, 19(1-4):23–56, 1986.

[Brö03] Oliver Bröker. *Parallel Multigrid Methods using Sparse Approximate Inverses*. PhD thesis, ETH, Zürich, May 2003.

[Bun88] Hans Bungartz. Entwurf von Datenstrukturen und Algorithmen für algebraische Mehrgitterverfahren. Diplomarbeit, Technische Universität München, 1988.

[CFH+00] Cleary, Andrew J., Falgout, Robert D., Henson, Van Emden, Jones, Jim E., Manteuffel, Thomas A., McCormick, Stephen F., Miranda, Gerald N., and Ruge, John W. Robustness and Scalability of Algebraic Multigrid. *SIAM J. Sci. Comput.*, 21(5):1886–1908, 2000.

[CFHJ98] Cleary, Andrew J., Falgout, Robert D., Henson, Van Emden, and Jones, Jim E. Coarse-grid Selection for Parallel Algebraic Multigrid. In *Proc. of the Fifth International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 104–115. Springer Verlag, 1998. Also available as Lawrence Livermore National Laboratory technical report UCRL-JC-130893.

[Cro02] R. Croce. Ein paralleler, dreidimensionaler Navier-Stokes-Löser für inkompressible Zweiphasenströmungen mit Oberflächenspannung, Hindernissen und dynamischen Kontaktflächen. Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, Bonn, Germany, 2002.

[GNR98] M. Griebel, T. Neunhoeffer, and H. Regler. Algebraic multigrid methods for the solution of the Navier-Stokes equations in complicated geometries. *Int. J. Numer. Meth. Fluids*, 26:281–301, 1998. Also as SFB Bericht 342/1/96A, Institut für Informatik, TU München, 1996.

[GO93] Gene Golub and James M. Ortega. *Scientific Computing.* Academic Press, 1993.

[GOS03] M. Griebel, D. Oeltz, and M. A. Schweitzer. An Algebraic Multigrid Method for Linear Elasticity. *SIAM J. Sci. Comp.*, 25(2):385–407, 2003.

[Hac85] Wolfgang Hackbusch. *Multi-Grid Methods and Applications.* Springer series in Computational Mathematics. Springer-Verlag, Berlin, Heidelberg, 1985.

[Hac96] Wolfgang Hackbusch. *Theorie und Numerik elliptischer Differentialgleichungen.* Teubner Studienbücher: Mathematik. Teubner, Stuttgart, 2. auflage edition, 1996.

[HY01] Van Emden Henson and Ulrike Meier Yang. *BoomerAMG:* a Parallel Algebraic Multigrid Solver and Preconditioner. Technical Report UCRL-JC-141495, Lawrence Livermore National Laboratory, March 2001.

[JED82] Jr. J. E. Dendy. Black box multigrid. *J. Comp. Phys.*, 48:366–386, 1982.

[JP93] Mark T. Jones and Paul E. Plassmann. A parallel graph coloring heuristic. *SIAM Journal on Scientific Computing*, 14(3):654–669, 1993.

[KK98a] George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graph. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[KK98b] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96 – 129, 1998.

[KK98c] George Karypis and Vipin Kumar. Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs. Technical Report 96-036, University of Minnesota, Department of Computer Science / Army HPC Research Center, Minneapolis, MN 55455, March 1998.

[KS99] Arnold Krechel and Klaus Stüben. Parallel Algebraic Multigrid Based on Subdomain Blocking. Technical Report REP-SCAI-1999-71, GMD, December 1999.

[KV00] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Algorithms and Combinatorics. Springer, Berlin, Heidelberg, 2000.

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comp.*, 15:1036–1053, 1986.

[Mes95] Message Passing Interface Forum. *MPI: A Message-Parsing Interface Standard*. University of Tennessee, Knoxville, Tennessee, 1.1 edition, June 1995.

[MK01] Irene Moulitsas and George Karypis. Multilevel algorithms for generating coarse grids for multigrid methods. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 45–45. ACM Press, 2001.

[Oel01] D. Oeltz. Algebraische Mehrgittermethoden für Systeme partieller Differentialgleichungen. Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, 2001.

[Stü99a] Klaus Stüben. A Review of Algebraic Multigrid. Technical Report REP-AiS-1999-69, GMD, November 1999.

[Stü99b] Klaus Stüben. Algebraic Multigrid (AMG): An Introduction with Applications. Technical report, GMD - Forschungszentrum Informationstechnik GmbH, March 1999.

[SZG99] M. A. Schweitzer, G. W. Zumbusch, and M. Griebel. Parnass2: A cluster of dual-processor PCs. In W. Rehm and T. Ungerer, editors, *Proceedings of the 2nd Workshop Cluster-Computing, Karlsruhe*, number CSR-99-02 in Chemnitzer Informatik Berichte, pages 45–54, Chemnitz, Germany, 1999. TU Chemnitz.

[Yan04] Ulrike Meyer Yang. On the Use of Relaxation Parameters in Hybrid Smoothers. *Numerical Linear Algebra With Applications*, 11:155–172, 2004. Lawrence Livermore National Laboratory technical report UCRL-JC-151575, May 2003.

[Zee90] P. M. de Zeeuw. Matrix-dependent prolongations and restrictions in a black-box multigrid solver. *J. Comp. and Appl. Math.*, 33:1–27, 1990.

[Zum01] G. W. Zumbusch. *Adaptive Parallel Multilevel Methods for Partial Differential Equations*. Habilitation, Universität Bonn, 2001.