Asynchrone Iterationsverfahren
zur
Lösung linearer Gleichungssysteme
auf
massiv parallelen Multiprozessorrechnern

Dissertation
zur
Erlangung des Doktorgrades
der Fakultät für Mathematik
der Universität Bielefeld

vorgelegt von
Klaus Jürgens

im Juni 1994

Einleitung

Iterative Verfahren zur Lösung linearer Gleichungssysteme sind ein wichtiges Werkzeug bei der numerischen Behandlung partieller Differentialgleichungen. Die einfachste, nichttriviale Aufgabe aus diesem Bereich ist das sogenannte "Modellproblem", das in die Klasse der elliptischen Randwertprobleme gehört.
Gesucht ist die Lösung der Poisson-Gleichung

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) \text{ für } (x,y) \in \Omega, \\ u(x,y) &= \varphi(x,y) \text{ für } (x,y) \in \partial\Omega \end{aligned}$$

auf dem Gebiet $\Omega = (0,1) \times (0,1)$. Zu gegebener Funktion $f(x,y)$ und gegebenen Randwerten $\varphi(x,y)$ soll die Funktion $u(x,y)$ auf dem Inneren von Ω berechnet werden. Δ bezeichnet in der obigen Gleichung den Laplace-Operator

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}.$$

Überzieht man das Gebiet Ω mit einem äquidistanten Gitter der Schrittweite $h = \frac{1}{N}$, so erhält man die diskrete Punktmenge

$$\Omega_h = \{(x,y) = (ih, jh) \quad | \quad 1 \leq i,j \leq N-1\}$$

als Teilmenge von Ω. Approximiert man die Poisson-Gleichung durch Differenzenquotienten, so ist die Diskretisierung der Differentialgleichung über dem Gitter Ω_h durch die Gleichungen

$$\frac{1}{h_2}(4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}) = f_{i,j}$$

für $1 \leq i,j \leq N-1$ gegeben. Hier wurden die Abkürzungen $u(ih,jh) = u_{i,j}$ bzw. $f(ih,jh) = f_{i,j}$ gewählt. Unter Berücksichtigung der Randwerte entsprechen die Diskretisierungen einem linearen Gleichungssystem $Au = f$ in $(N-1)^2$ Unbekannten $u_{i,j}$. Für eine lexikografische Anordnung der Unbekannten erhält die Koeffizientenmatrix des Gleichungssystems die Blockstruktur

$$A = \frac{1}{h^2}\begin{bmatrix} T & -I_{N-1} & & & \\ -I_{N-1} & T & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & T & -I_{N-1} \\ & & & -I_{N-1} & T \end{bmatrix}$$

mit den Blockdiagonaleinträgen

$$T = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 4 & -1 \\ & & & -1 & 4 \end{bmatrix} \in M_{N-1}(\mathbb{R}).$$

An der Koeffizientenmatrix des Gleichungsproblems werden sofort die Probleme deutlich, die dem Einsatz eines direkten Verfahrens zur Lösung des Gleichungssystems im Wege ständen. Die Anzahl der Unbekannten wächst quadratisch mit der Feinheit der Diskretisierung. Auch wenn zunächst nur $5N$ Elemente der Koeffizientenmatrix gespeichert werden müßten, so würde ein direktes Verfahren zu einem *fill in* führen und den Speicherbedarf erheblich vergrößern.
Dies ist anders bei iterativen Verfahren. Die Daten des Gleichungssystems können effizient unter Berücksichtigung des Nullschemas der Iterationsmatrix gespeichert werden und unterliegen während des Iterationsprozesses keinerlei Veränderungen. Der Speicherbedarf eines iterativen Verfahrens ist somit sehr gut kalkulierbar und stellt weniger Anforderungen an die vorhandene Rechnerausstattung als ein direktes Verfahren. Zusätzlich ist jeder Iterationsschritt sehr kostengünstig. Ist die Iterationsmatrix des Verfahrens bekannt, so besteht jeder Iterationsschritt im wesentlichen aus der Multiplikation einer dünnbesetzten Matrix mit einem Vektor.
Im Hinblick auf die zunehmende Bedeutung von Parallelrechnern zeigt sich ein weiterer Vorteil iterativer Verfahren. Sie lassen sich auf natürliche Art und Weise parallelisieren, indem die Multiplikation der Iterationsmatrix und der Iterierten zu gleichen Teilen auf mehrere Prozessoren verteilt wird. Da überwiegend *shared memory* Multiprozessorrechner im Einsatz sind, erfolgte die Untersuchung iterativer Verfahren auf diesen Systemen. Die geringe Skalierbarkeit und die hohen Kosten bei der Installation von *shared memory* Rechnern führten zur Entwicklung von sogenannten massiv-parallelen Multiprozessorrechnern, die aus einer Vielzahl preisgünstiger Prozessoren mit lokalem Speicher bestehen, die über ein konfigurierbares Netzwerk aus Datenleitungen verbunden sind. Aus dieser strukturellen Änderung der Multiprozessorrechner resultiert ein gänzlich verschiedenes Verhalten vieler Algorithmen. Erfolgte bisher der Datenaustausch zwischen mehreren Prozessoren über das *shared memory* des Rechners, so müssen auf einem massiv-parallelen Rechner zusätzlich zur Lösung des Problems Strategien für die Versendung von Nachrichten auf einem Netzwerk entwickelt werden, deren Anteil an der Gesamtlaufzeit des Algorithmus berücksichtigt werden muß.
Dieser für asynchrone Iterationsverfahren noch nicht betrachtete Aspekt motivierte die vorliegende Arbeit. Ausgehend von dem Verfahren der asynchronen Iteration [OW85] auf einem *shared memory* Multiprozessorrechner wird ein Modell für die asynchrone Iteration auf einem massiv-parallelen Multiprozessorrechner gegeben und das Konvergenzverhalten der asynchronen

Iteration untersucht. Kapitel 1 gibt zunächst eine allgemeine Einführung in die Theorie iterativer Verfahren und beschreibt unterschiedliche Verfahren für sequentielle und parallele Rechner. Kapitel 2 beschäftigt sich mit mathematischen Modellen für Multiprozessorrechner. Besondere Berücksichtigung findet in diesem Kapitel der Datenaustausch zwischen Prozessen über *shared memory* und über *message passing*. Auf dieser Grundlage wird in Kapitel 3 ein Modell für das asynchrone Iterationsverfahren auf einem massivparallelen Multiprozessorrechner entworfen, das zusätzlich zur Iteration auch die Kommunikation zwischen den Prozessoren berücksichtigt. Speziell für das asynchrone Block-Jacobi-Iterationsver-fahren werden in Kapitel 4 Aussagen bewiesen, die die Spektralradien der Iterationsmatrizen eines sequentiellen Block-Jacobi-Verfahrens und des Verfahrens aus Kapitel 3 zueinander in Relation setzen. Abschließend erfolgt in Kapitel 5 eine Darstellung der numerischen Experimente zur asynchronen Iteration auf einem Parsytec Supercluster64 Multiprozessorrechner.

An dieser Stelle bedanke ich mich bei den Herrn Dr. Gunnar Brinkmann, Ulf Jürgens und Dr. Gerd Krause für die vielen anregenden Gespräche. Mein besonderer Dank gilt Herrn Professor Dr. Ludwig Elsner für die geduldige und intensive Betreuung dieser Arbeit.

Inhaltsverzeichnis

Kapitel 1

Theorie iterativer Verfahren zur Lösung linearer Gleichungssysteme

1.1 Konvergenz und Konsistenz linearer iterativer Verfahren

Iterative Verfahren zur Lösung eines linearen Gleichungssystems

$$Ax = b \tag{1.1}$$

sind von Bedeutung für große, dünn besetzte Koeffizientenmatrizen A, wie sie z.B. bei der Diskretisierung partieller Differentialgleichungen entstehen. Ein iteratives Verfahren hängt von den Daten der Matrix A, des Vektors b und eines Startvektors x_0 ab. Allgemein versteht man unter einem linearen Iterationsverfahren eine Abbildung

$$\begin{aligned} \phi : \mathbb{R}^n \times \mathbb{R}^n &\longrightarrow \mathbb{R}^n \\ (x_k, b) &\longmapsto x_{k+1} = \phi(x_k, b) = Hx_k + Kb \quad k = 0, 1, 2, \ldots \quad , \end{aligned} \tag{1.2}$$

wobei H und K Matrizen aus $M_n(\mathbb{R})$ sind. H wird die Iterationsmatrix des Verfahrens genannt. Die Vektoren $x_k(x_0, b)$ werden als Iterierte bezeichnet. Ein Vektor $x^*(b)$ heißt Fixpunkt des Iterationsverfahrens, falls

$$x^*(b) = Hx^*(b) + Kb.$$

Die in (1.2) definierte lineare Abbildung ist im ersten Argument stetig, so daß eine konvergente Folge von Iterierten gegen einen Fixpunkt des Verfahrens konvergiert.
Im Hinblick auf das zu lösende Gleichungssystem (1.1) sind ausschließlich iterative Verfahren sinnvoll, deren Fixpunkte Lösungen des Gleichungssystems (1.1) sind. Kriterien für die Auswahl geeigneter Verfahren liefern die folgenden Definitionen:

Definition 1.1

> Ein Verfahren (1.2) heißt *konsistent* mit dem Gleichungssystem (1.1), falls jede Lösung des Gleichungssystems (1.1) ein Fixpunkt $x^*(b)$ des Verfahrens (1.2) ist.

Definition 1.2

> Ein Verfahren (1.2) heißt *konvergent*, falls für alle $b \in \mathbb{R}^n$ die Folge der Iterierten des Verfahrens (1.2) gegen einen vom Startwert x_0 unabhängigen Fixpunkt $x^*(b)$ konvergiert.

Ein mit dem gegebenen Gleichungssystem konsistentes sowie konvergentes Verfahren liefert für eine nichtsinguläre Matrix A die eindeutige Lösung des Gleichungssystems, wie der folgende Satz zeigt:

Satz 1.1

> A sei eine nichtsinguläre Matrix. Das iterative Verfahren (1.2) sei konsistent mit dem Gleichungssystem (1.1) und konvergent. Dann konvergiert die Folge der Iterierten des Verfahrens für alle Startwerte x_0 gegen die eindeutige Lösung des Gleichungssystems (1.1).

Beweis: siehe [You71].

Die Überprüfung eines gegebenen linearen Iterationsverfahrens auf Konvergenz erleichtert die folgende äquivalente Formulierung, die eine direkte Beziehung zwischen dem Spektralradius der Iterationsmatrix und der Konvergenz des Verfahrens herstellt.

Satz 1.2

> Das iterative Verfahren (1.2) ist konvergent dann und nur dann, wenn der Spektralradius der Iterationsmatrix kleiner als eins ist.

Beweis: siehe [You71].

1.2 Nichtnegative Matrizen und M-Matrizen

Die aus der Diskretisierung partieller Differentialgleichungen gewonnenen Koeffizientenmatrizen eines Gleichungssystems (1.1) und deren assozierte Iterationsmatrizen besitzen zumeist eine sehr spezielle Struktur, deren Verallgemeinerung zur Definition von Matrizenklassen geführt hat, die im Sinne von Abschnitt 1.1 vernünftige Iterationsverfahren liefern.

Auf der Menge der reellwertigen Matrizen $M_{m\times n}(\mathbb{R})$ definiert man die folgende Halbordnung:

Definition 1.3

$A, B \in M_{m\times n}(\mathbb{R})$ seien Matrizen, deren Koeffizienten mit a_{ij} und b_{ij} für $1 \leq i \leq m$ und $1 \leq j \leq n$ bezeichnet werden. Dann heißt

- $A > B$,
  falls $a_{ij} > b_{ij}$ für $i = 1, \ldots, m$ und $j = 1, \ldots, n$.
- $A \geq B$,
  falls $a_{ij} \geq b_{ij}$ für $i = 1, \ldots, m$ und $j = 1, \ldots, n$.

Eine Matrix $A \in M_{m\times n}(\mathbb{R})$ heißt *positiv*, falls $A > 0$.
Eine Matrix $A \in M_{m\times n}(\mathbb{R})$ heißt *nichtnegativ*, falls $A \geq 0$.

Analog definiert man $A < B$ und $A \leq B$. Diese Definition umfaßt auch den Vektorraum $\mathbb{R}^n = M_{1\times n}(\mathbb{R})$, so daß die Begriffe *positive* bzw. *nichtnegative* Vektoren erklärt sind.
Satz 1.2 zeigt, daß der Spektralradius der Iterationsmatrix für die Konvergenz eines iterativen Verfahrens von entscheidender Bedeutung ist. Für eine nichtnegative Matrix A erhält man bereits über ein Vergleichsargument eine Abschätzung für den Spektralradius $\rho(A)$ der Matrix A.

Satz 1.3

$A \in M_n(\mathbb{R})$ sei nichtnegativ, $u \in \mathbb{R}^n$ sei positiv und $\rho(A)$ bezeichne den Spektralradius der Matrix A. Dann gilt:

- $Au < u \Rightarrow \rho(A) < 1$
- $Au \leq u \Rightarrow \rho(A) \leq 1$
- $Au > u \Rightarrow \rho(A) > 1$
- $Au \geq u \Rightarrow \rho(A) \geq 1$

Beweis: siehe [Fro90].

Eine weitere Klassifizierung nichtnegativer Matrizen erhält man über die Eigenschaft der Reduzibilität bzw. Irreduzibilität einer Matrix.

Definition 1.4

Eine Matrix A heißt *reduzibel*, falls eine Teilmenge $\mathcal{J} \neq \emptyset, \mathcal{I}$ der Indexmenge $\mathcal{I} = \{1, \ldots, n\}$ existiert, so daß

$$a_{ij} = 0 \text{ für } i \in \mathcal{J} \text{ und } j \notin \mathcal{J}.$$

Andernfalls heißt die Matrix A *irreduzibel*.

Für nichtnegative, irreduzible Matrizen liefert der Satz von Perron-Frobenius eine Aussage über den betragsgrößten Eigenwert einer Matrix und den dazu korrespondierenden Eigenvektor.

Satz 1.4

> Die nichtnegative Matrix $A \in M_n(\mathbb{R})$ sei irreduzibel. Dann sind folgende Aussagen richtig:
>
> - $\rho(A)$ ist ein einfacher, positiver Eigenwert von A.
> - Zum Eigenwert $\rho(A)$ existiert ein positiver Eigenvektor u.

Beweis: siehe [Var62].

Der Satz von Perron-Frobenius dient als Grundlage für den sogenannten *Quotientensatz*, in dem obere und untere Schranken für den Spektralradius einer nichtnegativen, irreduziblen Matrix gegeben werden.

Satz 1.5

> Die nichtnegative Matrix $A \in M_n(\mathbb{R})$ sei irreduzibel. $u \in \mathbb{R}_n$ sei ein positiver Vektor. Dann ist entweder
>
> $$\frac{(Au)_i}{u_i} = \rho(A) \text{ für alle } i \in \{1, \ldots, n\}$$
>
> oder aber
>
> $$\min_{i \in \{1,\ldots,n\}} \frac{(Au)_i}{u_i} < \rho(A) < \max_{i \in \{1,\ldots,n\}} \frac{(Au)_i}{u_i}.$$

Beweis: siehe [Var62].

Sind die betrachteten Matrizen nicht mehr irreduzibel, aber immer noch nichtnegativ, so reduziert sich der Satz von Perron-Frobenius auf eine abgeschwächte Version.

Satz 1.6

> $A \in M_n(\mathbb{R})$ sei eine nichtnegative Matrix. Dann sind folgende Aussagen richtig:
>
> - $\rho(A)$ ist ein Eigenwert von A.
> - Zum Eigenwert $\rho(A)$ existiert ein nichtnegativer Eigenvektor u.

Beweis: siehe [Var62].

Auch der Quotientensatz gilt in abgeschwächter Form für nichtnegative Matrizen.

Satz 1.7

> $A \in M_n(\mathbb{R})$ sei nichtnegativ, $u \in \mathbb{R}$ ein positiver Vektor. Dann gilt
> $$\min_{i\in\{1,\ldots,n\}} \frac{(Au)_i}{u_i} \leq \rho(A) \leq \max_{i\in\{1,\ldots,n\}} \frac{(Au)_i}{u_i}. \tag{1.3}$$

Beweis: siehe [Var62].

Eine Folgerung aus dem Quotientensatz 1.7 sind die in Satz 1.8 gegebenen Abschätzungen für den Spektralradius.

Satz 1.8

> $A \in M_n(\mathbb{R})$ sei nichtnegativ, $u \in \mathbb{R}^n$ ein positiver Vektor. Weiterhin seien $\alpha \in \mathbb{R}_+ \cup \{0\}$ und $\beta \in \mathbb{R}_+$. Dann gelten die folgenden Implikationen:
> $$\begin{aligned} \alpha u \leq Au \quad &\Rightarrow \quad \alpha \leq \rho(A) \\ Au \leq \beta u \quad &\Rightarrow \quad \rho(A) \leq \beta. \end{aligned}$$

Beweis: siehe [BBG85].

Von besonderer Bedeutung für die Konstruktion iterativer Verfahren sind invertierbare Matrizen, deren Inverse nichtnegativ sind.

Definition 1.5

> Eine Matrix $A \in M_n(\mathbb{R})$ heißt *monoton*, falls A^{-1} existiert und A^{-1} nichtnegativ ist.

Eine Unterklasse der monotonen Matrizen bilden die sogenannten M-Matrizen, die in der Praxis sehr häufig aus Diskretisierungen partieller Differentialgleichungen entstehen.

Definition 1.6

> Eine monotone Matrix $A \in M_n(\mathbb{R})$ heißt *M-Matrix*, falls
> $$a_{ij} \leq 0 \text{ für } i \neq j.$$

1.3 Iterationsverfahren aus regulären Zerlegungen

Im linearen Gleichungssystem (1.1) werde im folgenden $A \in M_n(\mathbb{R})$ als nichtsingulär und $b \in \mathbb{R}^n$ angenommen. Ein konstruktiver Zugang zu iterativen Verfahren ergibt sich aus dem Begriff der regulären Zerlegung der Matrix A.

Definition 1.7

> Eine Zerlegung der Matrix A in zwei Summanden
>
> $$A = M - N \tag{1.4}$$
>
> heißt *regulär*, falls M monoton und $N \geq 0$ ist.

Noch schwächere Anforderungen an die Summanden der Zerlegung beschreibt die

Definition 1.8

> Eine Zerlegung (1.4) der Matrix A in zwei Summanden heißt *schwach regulär*, falls M monoton und $M^{-1}N \geq 0$ ist.

Existiert eine schwach reguläre Zerlegung der Matrix A, so erhält man aus dem Gleichungssystem (1.1) die Fixpunktgleichung

$$x = M^{-1}Nx + M^{-1}b$$

sowie das zugehörige Iterationsverfahren

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b. \tag{1.5}$$

Die Konsistenz des Verfahrens (1.5) mit dem Gleichungssystem (1.1) folgt sofort aus den Eigenschaften der schwach regulären Zerlegung. Ist A zusätzlich monoton, so ist der Spektralradius der Iterationsmatrix des Verfahrens (1.5) kleiner als eins.

Satz 1.9

> $A = M - N$ sei eine schwach reguläre Zerlegung. Dann sind die Aussagen 1 und 2 äquivalent:
>
> 1. A ist monoton.
> 2. $\rho(M^{-1}N) < 1$.

Beweis: siehe [OR67].

Aus Satz 1.2 geht somit sofort hervor, daß ein aus einer schwach regulären Zerlegung gewonnenes iteratives Verfahren für jeden beliebigen Startvektor $x_0 \in \mathbb{R}^n$ gegen eine eindeutige, von x_0 unabhängige Lösung konvergiert, falls A monoton ist.

Betrachtet man nun gleichzeitig m verschiedene schwach reguläre Zerlegungen

$$A = M_l - N_l \qquad l = 1, \ldots, m, \tag{1.6}$$

und sind zusätzlich m nichtnegative Diagonalmatrizen E_l mit der Eigenschaft

$$\sum_{l=1}^{m} E_l = I$$

gegeben, so entsteht durch eine Gewichtung der aus den Zerlegungen (1.6) gewonnenen Iterationsmatrizen mit diesen Diagonalmatrizen das Verfahren

$$x_{k+1} = \left(\sum_{l=1}^{m} E_l M_l^{-1} N_l \right) x_k + \left(\sum_{l=1}^{m} E_l M_l^{-1} b \right). \tag{1.7}$$

Im weiteren Verlauf dieser Arbeit sollen die Abkürzungen

$$\begin{aligned} H &= \sum_{l=1}^{m} E_l M_l^{-1} N_l, \\ H_l &= M_l^{-1} N_l \text{ für } l = 1, \ldots, m \end{aligned} \tag{1.8}$$

für die Iterationsmatrix des Verfahrens und die aus den Zerlegungen $A = M_l - N_l$ gewonnenen Komponenten der Iterationsmatrix benutzt werden. Auch für dieses Verfahren erhält man eine Aussage über den Spektralradius der Iterationsmatrix.

Satz 1.10

> A sei monoton und die Zerlegungen $A = M_l - N_l$ seien für $l = 1, \ldots, m$ schwach regulär. Dann gilt
>
> $$\rho(H) < 1.$$

Beweis: siehe [OW85].

Das Verfahren (1.7) ist auf natürliche Art und Weise parallelisierbar. Betrachtet man einen Rechner mit m Prozessoren $P_1, \ldots, P_m$, so berechnet jeder Prozessor P_l in einem Iterationsschritt eine neue Iterierte über der ihm zugewiesenen Zerlegung. Danach teilt jeder Prozessor den übrigen $m-1$ Prozessoren sein mit E_l gewichtetes Ergebnis mit. Ein Nachteil des Verfahrens (1.7) ist die notwendige Synchronisation der Ergebnisse nach jedem Iterationsschritt. Der langsamste der Prozessoren bestimmt die Geschwindigkeit des Verfahrens.
Das speziell für *shared-memory* Parallelrechner von O'Leary und White [OW85] vorgeschlagene Verfahren der asynchronen oder chaotischen Iteration behebt diesen Nachteil. Jedem Prozessor P_l wird eine schwach reguläre Zerlegung der Matrix A zugeteilt, die in jedem Iterationsschritt lokal zur Berechnung

der nächsten Iterierten benutzt wird. Das Ergebnis wird dann mit der Matrix E_l gewichtet und in den für alle Prozessoren zugänglichen gemeinsamen Speicher geschrieben. Im deutlichen Unterschied zum Verfahren (1.7) ist aber der Startvektor für die nächste Iteration der zu diesem Zeitpunkt im gemeinsamen Speicher befindliche Itererationsvektor und nicht die vollständige Iterierte des Zeitschrittes k aus dem Verfahren (1.7).
Eine formale Beschreibung des asynchronen Verfahrens ist durch die Gleichung

$$x_{k+r_k} = (I - E_{l_k})\,x_{k+r_k-1} + E_{l_k} H_{l_k} x_k + E_{l_k} M_{l_k}^{-1} b \tag{1.9}$$

für $k = 1, 2, 3, \ldots$ gegeben. r_k bezeichnet die Anzahl der durch andere Prozessoren erfolgten Schreibzugriffe auf den Vektor x_k während der lokalen Iterationsphase im Prozessor P_{l_k}. Die Indizes l_k sind beliebige ganze Zahlen im Bereich $1 \leq l_k \leq m$.
Ist die Folge der Prozessorschreibzugriffe auf den gemeinsamen Speicher *reguliert*, d. h. existiert eine Zahl $T \in N$, so daß in einem beliebigen Intervall von T aufeinanderfolgenden Elementen der Folge alle Zahlen zwischen 1 und m zu finden sind, so gibt es auch hier eine Konvergenzaussage.

Satz 1.11

> A sei eine monotone Matrix, und die Zerlegungen $A = M_l - N_l$ seien schwach regulär. Ist die Folge $\mathcal{F} = \{l_k\}_{k=1}^{\infty}$ mit $1 \leq l_k \leq m$ reguliert, so konvergiert das Verfahren (1.9) für einen beliebigen Startvektor x_0 gegen den Lösungsvektor des linearen Gleichungssystems (1.1).

Beweis: siehe [BEN88].

Kapitel 2

Mathematische Modelle für Parallelrechner

2.1 Abstrakte Parallelrechner

Die im letzten Kapitel vorgestellten Ansätze zur iterativen Lösung von Gleichungssystemen berücksichtigen in keiner Weise Zeitverluste, die auf realen Multiprozessorrechnern durch Synchronisation oder Kommunikation auftreten. Das Ziel dieses Kapitels ist, einen Überblick über die auf Multiprozessorrechnern möglichen Kommunikationsmechanismen zu geben und diese speziell am Beispiel eines Transputernetzwerkes zu studieren.

Multiprozessorrechner werden von Flynn [Fly72] als MIMD-Rechner (*multiple instruktion multiple data computer*) klassifiziert, wenn sie über mehrere Prozessoren verfügen, die unabhängig voneinander eine Instruktionsfolge bearbeiten und lediglich über ein Medium miteinander kommunizieren. Der Multiprozessorrechner wird als eng gekoppelt bezeichnet, wenn die Kommunikation über einen gemeinsamen Arbeitsspeicher aller Prozessoren erfolgt. Diese Rechnerklasse ist auch unter der Bezeichnung *shared memory* Rechner bekannt. Sind die Prozessoren lediglich über ein Netz von Datenleitungen miteinander verbunden, so spricht man von einer losen Koppelung. In beiden Fällen können die Prozessoren über einen lokalen Arbeitsspeicher verfügen.

Fortune und Wyllie entwickelten zur formalen Beschreibung der Klasse der *shared memory* Rechner ein abstraktes Modell, das als PRAM (*parallel random access machine*) bezeichnet wurde [FW78]. Eine PRAM der Größe P erfüllt folgende Eigenschaften.

Definition 2.1

Ein Computer heißt PRAM der Größe P, falls

- P nicht dynamisch modifizierbare Programme gleichzeitig nebeneinander laufen,
- ein Speicher existiert, der aus einer Folge von Wörtern der Größe eines Integers besteht,
- P Akkumulatoren existieren, die jeweils eines der Programme abarbeiten,
- ein read-only Eingabeband existiert und
- ein write-only Ausgabeband existiert.

(Quelle: [AG89])

Wie aus der Definition ersichtlich, verbindet eine PRAM den Computer und ein Programm zu einer Einheit. Die Programme bestehen aus elementaren Anweisungen, die auf dem Inhalt des Akkumulators oder einer Speicherzelle operieren. Jede Anweisung benötigt genau eine Zeiteinheit, so daß eine PRAM im Gegensatz zu realen *shared memory* Rechnern ein synchrones Rechnermodell darstellt. Der einzige Berührungspunkt der parallel arbeitenden Akkumulatoren ist der gemeinsame Speicherbereich des Computers. Kommunikation zwischen Prozessoren wird dementsprechend über Speicherzugriffe beschrieben, die als *read* Operationen oder als *write* Operationen auftreten. Gleichzeitige *read* Operationen mehrerer Prozessoren sind für den Ablauf eines Programmes unkritisch, da es sich hierbei um das Kopieren von Daten aus dem gemeinsamen Speicher in ein lokales Prozessorregister handelt. Der gleichzeitige Lesezugriff mehrerer Prozessoren auf eine Speicherzelle wird in der Literatur als *concurrent read* bezeichnet [AG89].

Schwieriger gestaltet sich die Definition einer *write* Operation mehrerer Prozessoren auf eine Speicherzelle. Ohne steuernde Maßnahmen ist der Inhalt der Speicherzelle nach Beendigung des Schreibzugriffes nicht exakt definiert und kann aus einer beliebigen, unter Umständen sinnlosen Mischung aller geschriebenen Daten bestehen. Fortune und Wyllie fordern in ihrem PRAM Modell den sofortigen Programmabbruch, falls mehrere Prozessoren gleichzeitig auf eine Speicherzelle schreiben [FW78]. Diese Bedingung entspricht der ausschließlichen Schreiberlaubnis für lediglich einen der beteiligten Prozessoren auf den gemeinsamen Speicher der PRAM (*mutual exclusion*). Die Steuerung des Speicherzugriffes muß also im Programm der PRAM formuliert werden. Hier wird deutlich, daß bei der Steuerung des Schreibzugriffes auf den gemeinsamen Speicher durch die notwendige *mutual exclusion* ein Engpaß entsteht, der zu einer sequentiellen Verarbeitung der Schreibzugriffe führt.

Ein lose gekoppelter Multiprozessorrechner läßt sich auf ähnliche Weise abstrahieren. Der gemeinsame Speicher einer PRAM wird durch ein zusammenhängendes Netzwerk aus Datenleitungen ersetzt. Formal wird dieser Rechnertyp als MP-RAM (*message passing random access machine*) klassifiziert.

Definition 2.2

Ein Computer heißt MP-RAM der Größe P, falls

- P nicht dynamisch modifizierbare Programme gleichzeitig nebeneinander laufen,
- P private, den Prozessoren zugeordnete Speicher existieren, die aus einer Folge von Wörtern der Größe eines Integers bestehen,
- P Akkumulatoren existieren, die jeweils eines der Programme abarbeiten,
- ein read-only Eingabeband existiert,
- ein write-only Ausgabeband existiert und
- die Prozessoren gemäß eines vorgegebenen Verknüpfungsgraphen angeordnet sind.

(Quelle: [AG89])

Ein wesentliches Merkmal dieses Parallelrechnertyps ist die Einschränkung des Datenaustausches auf direkt benachbarte Prozessoren. Kommunikation zwischen Prozessoren findet im MP-RAM Modell durch *message passing* statt, d.h. Nachrichten werden von einem Sender entlang der vorgegebenen Datenleitungen zu einem Empfänger geschickt. Der Befehlssatz eines MP-RAM Rechners muß dementsprechend um die zusätzliche Möglichkeit einer *send message* und einer *receive message* Operation erweitert werden. Kommunikation zwischen zwei Prozessoren in einem MP-RAM bedeutet die Synchronisation zweier Instruktionsfolgen für den Zeitpunkt des Datenaustausches. Wird die Effizienz des Datenaustausches beim PRAM Rechnermodell hauptsächlich durch eine mehr oder weniger geeignete Speicherverwaltung des gemeinsamen Speichers bestimmt, so wird sie beim MP-RAM Modell durch den Verknüpfungsgraphen der Maschine und den die Prozessorinteraktion beschreibenden Teil des Programms beeinflußt.

2.2 Parallele Prozesse

In Erweiterung der in Abschnitt 2.1 betrachteten Rechnermodelle sollen im folgenden auch mehrere Instruktionsströme auf einem Prozessor zugelassen werden. Die Parallelität auf der Hardwareebene wird somit durch eine Parallelität auf Prozeßebene ergänzt. Dieser Abschnitt beschäftigt sich mit der genauen Spezifizierung des Begriffes Prozeß und den Besonderheiten, die bei der Interaktion parallellaufender Prozesse zu beachten sind. Es soll verdeutlicht werden, daß in jedem Algorithmus mit mehreren parallellaufenden

Prozessen sowohl der Zugriff auf einen gemeinsamen Datenbereich als auch Kommunikation über *message passing* Einfluß auf die Zeitkomplexität des Algorithmus haben.
Ein Prozeß ist definiert als ein in Ausführung befindliches Programm. In einem einfachen Modell können sich Prozesse während eines Programmlaufes in drei unterschiedlichen Zuständen befinden, die als *running*, *ready* oder *blocked* bezeichnet werden. Die Verteilung der Systemzeit auf parallellaufende Prozesse und die Schaltung zwischen verschiedenen Zuständen wird in jedem Computersystem von einem *scheduler* gesteuert. In jedem Prozessor kann lediglich ein einziger Prozeß den Zustand *running* besitzen. Nach Ablauf einer bestimmten Ausführungszeit wird dieser Prozeß in den Zustand *ready* überführt und in eine Warteschlange eingereiht. Der nächste Prozeß aus der *ready* Warteschlange wird danach aktiviert. Gelangt ein Prozeß im Zustand *running* zu einer Instruktion, die ein noch nicht berechnetes Ergebnis verlangt, so blockiert der *scheduler* diesen Prozeß und überführt ihn in den Zustand *blocked*. Erst nachdem das geforderte Ergebnis vorliegt, kann dieser Prozeß wieder in die *ready* Warteschlange eingereiht werden. Diese Folge von möglichen Zustandsänderungen veranschaulicht die Abbildung 2.1.

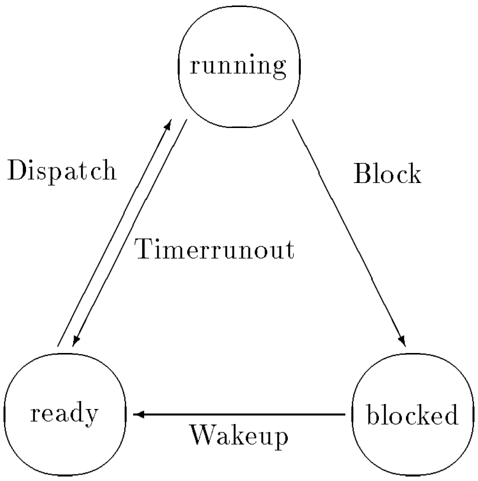

Abbildung 2.1: Prozeßzustände

Konflikte entstehen in diesem Modell, wenn mehrere Prozesse auf einen gemeinsamen Datenbereich zugreifen. Wird ein Prozeß während einer Schreiboperation vom Zustand *running* in den Zustand *ready* überführt, ohne daß die Operation abgeschlossen ist, so kann ein nachfolgender, aus diesem Speicherbereich lesender Prozeß einen undefinierten Speicherzustand vorfinden. Diesem Problem wird durch das Konzept der *mutual exclusion* vorgebeugt. Jeder Prozeß, der auf gemeinsame, mit anderen Prozessen geteilte Datenbereiche zugreift, verhindert aktiv den Zugriff gleichberechtigter Prozesse zu derselben Zeit. Der für eine *mutual exclusion* zuständige Bereich des Programmes wird als *critical section* bezeichnet und muß die nachstehend aufgeführten Postulate erfüllen:

- Zu jedem Zeitpunkt darf sich nur ein Prozeß in einer *critical section* befinden.
- Eine *critical section* soll so kurz wie möglich sein.
- Ein in seiner *critical section* befindlicher Prozeß darf nicht in den Zustand *blocked* übergehen.
- Terminiert ein Prozeß in seiner *critical section*, so muß das Betriebssystem für die Auflösung der aktiven Sperrung sorgen.

Eine Möglichkeit der Implementation der *mutual exclusion* bietet das Konzept des *monitors*, das von Hoare [Hoa74] und Brinch Hansen [Han73] entwickelt wurde. Hoare definiert einen Monitor wie folgt:

Definition 2.3

> Ein Monitor ist ein abstrakter Datentyp, innerhalb dessen Geltungsbereich nur ein Prozeß zu derselben Zeit aktiv sein darf.

Ein Monitor besteht aus den zu schützenden Resourcen sowie aus Prozeduren und Variablen, die für den Schutz und die Steuerung des Zugriffs auf diese Daten benötigt werden. Externe Prozesse erhalten Zugang zu den Resourcen des Monitors ausschließlich über *monitor entry* Prozeduren, die den exclusiven Zugriff eines Prozesses zum Monitor steuern. Erhält nun ein externer Prozeß über einen *monitor entry* Zugang zu den Monitorresourcen und stellt innerhalb des Monitors fest, daß die weitere Programmausführung Ergebnisse eines anderen Prozesses verlangt, so müßte gemäß des oben vorgestellten Multiprozeßmodells eine Transformation in den Zustand *blocked* erfolgen. Dies widerspricht jedoch den Anforderungen einer *critical section*. Einen Ausweg bietet die Zuordnung einer sogenannten *condition variable* zu jedem möglichen Grund einer Blockierung von Prozessen im Monitor. *Condition variables* sind Datenstrukturen innerhalb des Monitors, die für alle Prozeduren des Monitors erreichbar sind. Prozesse operieren auf *condition variables* mit Hilfe der Prozeduren *wait(condition variable)* und *signal(condition variable)*. Tritt ein Grund für eine Blockierung des Prozesses innerhalb des Monitors auf, so suspendiert der Prozeß mit dem Aufruf *wait(condition variable)* seine Ausführung, verläßt den Monitor und reiht sich in die Warteschlange der Prozesse ein, die auf die Erfüllung des mit *condition variable* verbundenen Ereignisses warten. Somit ist der Weg frei für den nächsten Prozeß, der die Resourcen des Monitors beansprucht. Der erste Prozeß, der die Bedingung für die *condition variable* erfüllt, meldet dies beim Verlassen des Monitors mit dem Aufruf der Prozedur *signal(condition variable)*. Gleichzeitig erhält genau ein Prozeß aus der mit *condition variable* gekoppelten Warteschlange Zugang zum Monitor. Neben den Operationen *wait* und *signal* wird die Funktion *queue (condition variable)* benutzt, um den Zustand der Warteschlange anzuzeigen. *Queue* liefert den Wert *TRUE*, falls mindestens ein Prozeß wartet, andernfalls liefert *queue* den Wert *FALSE*.
Ein typisches Beispiel für die Anwendung eines Monitors ist das sogenannte *reader-writer* Problem, das auch zur Verwaltung der gemeinsamen Speicherbereiche bei der asynchronen Iteration auf einem Transputernetzwerk benutzt wird. Daher soll es hier näher erläutert werden. Ein *reader-writer* Problem tritt auf, wenn mehrere konkurrierende Prozesse auf einen gemeinsamen Datenbereich lesend oder schreibend zugreifen wollen. Der Schreibzugriff muß aus oben geschilderten Gründen exklusiv sein. Es können jedoch gleichzeitig mehrere Lesezugriffe stattfinden. Eine Lösung des Problems zeigt die folgende Definition des Datentyps Monitor, die dem Buch [Sno92] entnommen wurde:

```
monitor readers_and_writers

begin

  var readercount:         integer;

      writing :            boolean;

      oktoWrite, oktoRead: condition;


  procedure RequestRead;

  begin

    if (writing OR queue(oktoWrite)) then

      wait (oktoRead);

    readercount := readercount + 1;

    signal (oktoRead);

  end;


  procedure ReleaseRead;

  begin

    readercount := readercount - 1;

    if ( readercount = 0) then

      signal (oktoWrite);

  end;


  procedure RequestWrite;

  begin
```

```
    if (writing OR (readercount != 0)) then
      wait (oktoWrite);
    writing := TRUE;
  end;

  procedure ReleaseWrite;
  begin
    writing := FALSE;
    if (queue(oktoREAD)) then
      signal (oktoRead);
    else
      signal (oktoWrite);
  end;

  /* Initialisierung des Monitors */
  readercount := 0;
  writing := false;
```

Die Prozesse *reader* und *writer* erhalten jeweils unterschiedliche *monitor entry* Prozeduren. Der *monitor entry* für einen *reader* übergibt den Zugriff auf die angeforderten Resourcen erst, wenn kein *writer* aktiv ist oder in einer*queue* wartet. Falls noch weitere *reader* in einer *queue* auf die Freigabe des Lesezugriffs warten, so werden diese aktiviert. Dies entspricht dem erlaubten parallelen Zugriff mehrerer *reader* auf einen gemeinsamen Datenbereich.
Ein *writer* wird durch den *monitor entry* so lange aufgehalten, bis kein Prozeß mehr Zugriff auf die Resourcen des Monitors hat. Anschließend erfolgt seine Aktivierung. Beim Verlassen des Monitors überprüft der *writer* zunächst, ob ein *reader* auf seine Aktivierung wartet, und erst danach, ob ein *writer* in der *queue* steht. Diese Konstruktion verhindert, daß weder *reader* noch *writer* unendlich lange in einer *queue* auf ihre Ausführung warten.

Die bisher betrachteten Datentypen erlauben eine indirekte Nachrichtenübermittlung zwischen Prozessen. Gerade im Hinblick auf lose gekoppelte MIMD Rechner ist es wichtig, direkten Nachrichtenaustausch zwischen zwei Prozessoren über *message passing* zu unterstützen.

Eine Datenübertragung zwischen zwei Prozessen ist nur dann sinnvoll, wenn der empfangende Prozeß eine an ihn adressierte Nachricht interpretieren kann. Dies erfordert zumindest Kenntnisse über den Datentyp der Nachricht. Als eine mögliche Lösung für dieses Problem findet man in den meisten parallelen Programmiersprachen den Datentyp *channel*. Ein *channel* ist eine unidirektionale, statische Verbindung zwischen zwei Prozessen, die lediglich den Transfer eines bestimmten, beiden Prozessen bekannten Datentyps zuläßt. Zugriff auf einen *channel* erhalten die Prozesse über die Operationen

- *send (channel, message)*
- *receive (channel, message).*

send arbeitet synchron, d.h. der aufrufende Prozeß wird solange blockiert, bis *send* die Nachricht übermittelt hat. Analog dazu blockiert *receive* solange den empfangenden Prozeß, bis auf dem bezeichneten *channel* eine Nachricht empfangen wurde.

Eine Nachrichtenübermittlung zweier Prozesse über einen *channel* stellt also einen Synchronisationspunkt der beteiligten Prozesse dar, der einer Zuweisungsoperation in einem sequentiellen Prozeß entspricht.

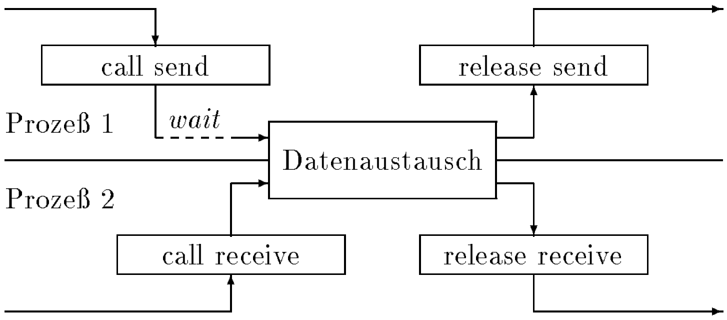

Abbildung 2.2: Synchronisation zweier Prozesse

2.3 Nachrichtenübermittlung per *broadcast*

Im Hinblick auf die Implementierung iterativer Verfahren auf *message passing* Multiprozessorrechnern ist besonders die Datenübermittlung durch *broadcasts* von Bedeutung.

Definition 2.4

> *Broadcast* bezeichnet den Prozeß der Informationsübermittlung in einem zusammenhängenden Netzwerk von m Teilnehmern, wobei ausgehend von einer Informationsquelle an alle übrigen Teilnehmer eine Nachricht verschickt wird. Ein *broadcast* ist abgeschlossen, sobald alle Teilnehmer des Netzwerkes die Nachricht erhalten haben.

Die Übertragung einer Nachricht erfolgt durch eine Sequenz von *calls*, die die folgenden Eigenschaften erfüllen sollen:

- Ein *call* benötigt genau eine Zeiteinheit.
- An jedem *call* sind exakt zwei Partner beteiligt, wobei einer als Sender und der andere als Empfänger bezeichnet wird.

Die weitere Klassifizierung des *broadcast* beruht auf der Basis der vorgegebenen Hardwarestruktur des Netzwerkes.

Definition 2.5

> Ein zusammenhängendes Netzwerk mit m Teilnehmern sei gegeben. Dann heißt ein *broadcast*
>
> - *processor bound*, falls jeder Teilnehmer des Netzwerkes pro Zeiteinheit lediglich an einem *call* beteiligt sein darf.
> - *link bound*, falls jeder Teilnehmer des Netzwerkes pro Zeiteinheit an einem *call* mit jedem seiner Nachbarn beteiligt sein darf.
> - *DMA bound*, falls jeder Teilnehmer des Netzwerkes pro Zeiteinheit an einem *call* mit maximal k seiner Nachbarn beteiligt sein darf. Hier ist k kleiner als die Anzahl der vorhandenen Nachbarn.

Die erfolgreiche Beendigung eines *broadcast* in einem Netzwerk mit m Teilnehmern erfordert mindestens $m - 1$ *calls*, da mit Ausnahme der Nachrichtenquelle jeder Teilnehmer mindestens einmal als Empfänger auftreten muß. Entsprechend folgt aus der Forderung nach einem zusammenhängenden Netzwerk, daß für m Teilnehmer mindestens $m-1$ Datenleitungen vorhanden sein müssen, um eine erfolgreiche Beendigung eines *broadcast* zu gewährleisten.
Ist ein zusammenhängendes Netzwerk mit m Teilnehmern gegeben, so ist die Anzahl der Zeiteinheiten, die ein von einer beliebigen Quelle im Netzwerk ausgehender *broadcast* benötigt, ein Kriterium für die Güte des Netzwerkes. Bestimmt man für jede mögliche Quelle des Netzwerkes die minimale Zeit für einen *broadcast*, so heißt das Maximum dieser Zeiten die *broadcast* Zeit des Netzwerkes. Die minimal erreichbare *broadcast* Zeit in einem Netzwerk mit m Knoten hängt direkt von der Hardware des Netzwerkes ab.

Den einfachsten Fall stellt ein *link bound broadcast* dar. Da jeder Teilnehmer mit allen seinen Nachbarn gleichzeitig kommunizieren kann, breitet sich die Nachricht ausgehend von der Nachrichtenquelle wellenförmig aus. Das Maximum über die kürzesten Wege von der Quelle zu den übrigen Teilnehmern ist also ein Maß für die Dauer des *broadcast*. Das optimale Netzwerk für einen *link bound broadcast* ist also ein vollständiger Graph, da hier ein Zeittakt ausreicht, um von einer beliebigen Quelle alle übrigen Teilnehmer zu informieren.

In einem *processor bound broadcast* kann sich die Anzahl der informierten Teilnehmer des Netzwerkes pro Zeitschritt maximal verdoppeln. Dies bedeutet, daß nach t Zeitschritten maximal $\sum_{i=0}^{t-1} 2^i$ Teilnehmer benachrichtigt sein können. Hieraus folgt sofort, daß ein Netzwerk mit m Teilnehmern in minimal

$$t_{opt} = \lceil log_2(m) \rceil \tag{2.1}$$

Zeitschritten informiert werden kann.

Die Berechnung der minimalen *broadcast* Zeit für einen *DMA bound broadcast* verläuft analog zum *processor bound broadcast*. Hier kann jedoch jeder Teilnehmer pro Zeitschritt maximal k weitere Knoten informieren, so daß letztendlich ein Netzwerk mit m Teilnehmern in minimal

$$t_{opt} = \lceil log_{k+1}(m) \rceil \tag{2.2}$$

Zeitschritten informiert werden kann.

2.4 Ein Netzwerk aus Transputern

Ein typisches Beispiel für einen frei konfigurierbaren, lose gekoppelten Multiprozessorrechner ist ein Transputernetzwerk, wie es auch der Fakultät für

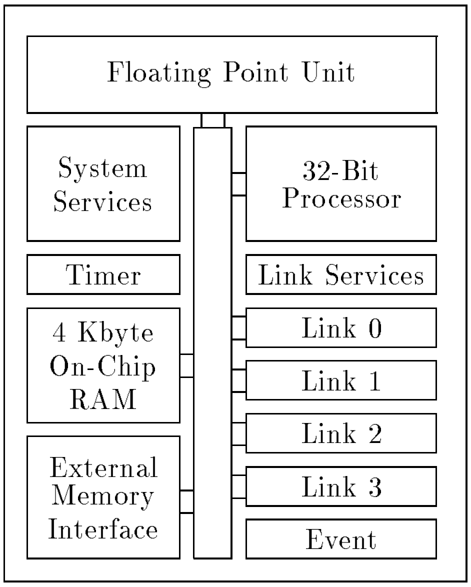

Abbildung 2.3: Blockschaltbild Transputer

Mathematik an der Universität Bielefeld zur Verfügung steht.
Transputer sind Prozessoren, die die Eigenschaften eines Mikrocomputers mit einer zusätzlichen Hochgeschwindigkeitsschnittstelle zur Datenübertragung kombinieren. Das Blockschaltbild eines Inmos T805 Transputers zeigt die nebenstehende Abbildung [Inm92]. Der Transputer ist ein vollwertiger 32-bit Prozessor, dessen CPU, FPU und 4 Kbyte schneller Speicher prozessorintern über einen 32-bit breiten Bus verbunden sind. Dank eines integrierten Hardwareschedulers ist der Inmos Transputer in der Lage, mehrere Prozesse gleichzeitig zu verwalten und im Vergleich zu anderen Prozessoren sehr schnell zwischen den einzelnen Prozessen zu schalten. Ein entscheidender Unterschied des Transputers im Vergleich zu herkömmlichen Prozessoren ist seine erweiterte Kommunikationsfähigkeit. Vier ebenfalls an den internen Bus des Prozessors gekoppelte serielle Schnittstellen geben dem Transputer die Möglichkeit des schnellen Datenaustausches mit anderen Transputern. Die Schnittstellen werden als *links* bezeichnet und sind bidirektional nutzbar. Die Übertragungsrate der seriellen Datenleitungen beträgt maximal 20 Megabit pro Sekunde. CPU und *links* können nahezu unabhängig voneinander arbeiten, da die *links* direkten Zugriff auf den Speicherbereich des Transputers haben (DMA-Modus) und die CPU lediglich zur Steuerung des Datentransfers benutzt wird. Arbeiten alle vier Links gleichzeitig, so führt dies zu einem maximalen Leistungsverlust der CPU von zehn Prozent [Ebe93].
Legt man die in Abschnitt 2.3 dargestellte Klassifizierung für Multiprozes-

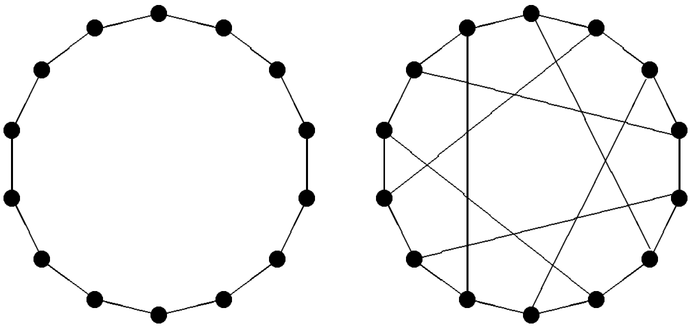

Abbildung 2.4: Beispielgraphen

sornetzwerke zugrunde, so ist ein *broadcast* auf einem Transputernetzwerk aufgrund dieser Leistungsdaten als *link bound* zu bezeichnen. Bei direkter

Vernetzung und Nutzung der Transputer ist der Grad des Prozessorgraphen maximal vier, da jeder Knoten über vier Schnittstellen verfügt. Beispiele für mögliche Prozessorgraphen, die auch für die numerische Anwendung in Kapitel 5 benutzt werden, sind 2-reguläre Ringgraphen und dreireguläre Graphen mit optimaler *broadcast* Zeit wie z.B. der Heawoodgraph (siehe Abbildung 2.4).
Durch Verschalten mehrerer Transputer zu einem größeren Knoten erhält man auch Prozessorgraphen höheren Grades. Zwei Transputer lassen sich so z. B. zu einem Knoten mit Kantengrad 6 verschalten (Abbildung 2.5).

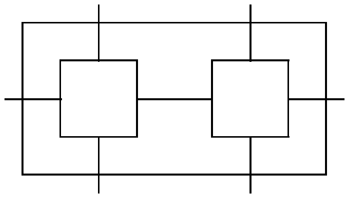

Abbildung 2.5: Beispiel für Mehrprozessorknoten

Obwohl die Transputer der Generation T800 technisch bereits überholt sind, bieten sie dennoch ein sehr flexibles Medium zur Implementation und Analyse paralleler Algorithmen, da die Programmierung auf einem sehr elementaren und transparenten Niveau erfolgen kann. Höhere Programmiersprachen für Transputersysteme sind zumeist um die in Abschnitt 2.2 vorgestellten Datentypen zur Synchronisation paralleler Prozesse erweitert, so daß der Programmierer die direkte Kontrolle über die Resourcen des Transputers hat.

Kapitel 3

Multisplitting-Verfahren auf Multiprozessorrechnern

3.1 Graphen und Multiprozessorrechner

Eine geeignete formale Beschreibung eines Multiprozessorrechners mit den Prozessoren $P_1, \ldots, P_m$ erhält man durch einen zusammenhängenden, gerichteten Graphen $G = (V, E)$ mit m Knoten. $V(G)$ bezeichnet die Menge der Knoten von G, $E(G)$ die Menge der Kanten in G. Besteht eine Datenleitung von Prozessor P_i zu Prozessor P_j, so wird dies im Modell durch eine gerichtete Kante vom Knoten i zum Knoten j dargestellt.

Für die Modellierung eines Multiprozessorrechners ist es weiterhin sinnvoll, den Graphen als einfach und symmetrisch anzunehmen.

Definition 3.1

- Ein gerichteter Graph G heißt *einfach*, falls keine Schleifen, d.h. Kanten von Knoten i zu Knoten i, und keine parallelen Kanten exitieren.
- Ein gerichteter Graph G heißt *symmetrisch*, falls zu jeder Kante von Knoten i zu Knoten j auch eine Kante von Knoten j zu Knoten i existiert.

Diese Einschränkungen an die betrachteten Graphen sind sinnvoll, da Schleifen und parallel liegende Kommunikationsverbindungen einen zusätzlichen Aufwand bei der Konstruktion des Multiprozessorrechners und somit Kosten erfordern, deren Nutzen fragwürdig ist. Die geforderte Symmetrie spiegelt wider, daß die Kommunikationsverbindungen in Multiprozessorsystemen zumeist bidirektional sind.

Auf der Basis dieses Rechnermodells für einen Multiprozessorrechner läßt sich nun ein Analogon zum asynchronen Iterationsverfahren auf einem *shared memory* Parallelrechner formulieren:

Verfahren 3.1

> Ein Multiprozessorrechner setze sich aus m Prozessoren $P_1, \ldots, P_m$ zusammen, die gemäß eines gerichteten, einfachen und symmetrischen Graphen G verschaltet sind.
>
> **Initialisierung:** Jedem Prozessor P_l werden über die vorhandenen Datenleitungen ein Startvektor x_0, eine Zerlegung $A = M_l - N_l$, der Vektor b und ein Gewicht E_l mitgeteilt.
>
> **Iterationsschritt:** Jeder Prozessor P_l berechnet lokal eine Iterierte aus der ihm zugewiesenen Zerlegung (1.6), skaliert das Ergebnis mit der Matrix E_l und verschickt die berechnete Information per *broadcast* an die übrigen Prozessoren. Empfängt der Prozessor P_l Nachrichten benachbarter Prozessoren, so beginnt er lokal mit der nächsten Iteration.

Eine Beschreibung dieses Verfahrens für einen Multiprozessorrechner erfordert eine exaktere Definition des Begriffes *broadcast*. Ein *broadcast* in einem vorgegebenen Graphen G entspricht der Ausbreitung einer Nachricht von einem Quellknoten l entlang eines gerichteten Spannbaumes zu allen übrigen Knoten im Graphen G. Ein Spannbaum ist ein spezieller Subgraph von G.

Definition 3.2

> Ein gerichteter *Spannbaum* ST_l eines Graphen G ist ein zusammenhängender Subgraph von G mit Wurzel im Knoten l, der alle Knoten von G umfaßt und keinen Kreis enthält. Außerdem existiert von der Wurzel l ein Weg zu jedem Knoten.

Da jeder Prozessor während des Iterationsverfahrens *broadcasts* durchführt, beschreibt eine Menge von m Spannbäumen ST_l in den Knoten $l = 1, \ldots, m$ alle für das Iterationsverfahren benötigten Kommunikationswege im Prozessorgraphen.

Definition 3.3

> Gegeben sei ein Prozessorgraph mit m Knoten. Dann heißt eine Menge $\{\mathrm{ST}_1, \ldots, \mathrm{ST}_m\}$ von m Spannbäumen des Prozessorgraphen eine *broadcast* Strategie des Prozessorgraphen.

In einem gegebenen Prozessorgraphen sind die Spannbäume und damit auch die *broadcast* Strategien nicht eindeutig.

Eine matrizentheoretische Darstellung aller Kommunikationswege ergibt sich aus dem Begriff der Adjazenzmatrix eines Graphen.

Definition 3.4

> G sei ein gerichteter Graph mit Knotenmenge $V(G)$ und Kantenmenge $E(G)$. Die Matrix $\mathcal{A}(G) = (a_{ij})$ heißt *Adjazenzmatrix* des Graphen G und ist durch die folgenden Einträge beschrieben:
>
> $$a_{ij} = \begin{cases} 1 & \text{falls eine Kante vom Knoten i zum Knoten j existiert} \\ 0 & \text{sonst}\,. \end{cases}$$

Speziell für einen Spannbaum ST_l erhält man besonders einfache, dünn besetzte Adjazenzmatrizen, da Spannbäume eines Graphen mit m Knoten genau $m-1$ Kanten haben. Betrachtet man als Prozessorgraph G den vollständigen Graphen K_m mit m Knoten, so ist für jeden Knoten l der Spannbaum minimaler Höhe durch die folgende Adjazenzmatrix gegeben:

$$\begin{array}{rcl} & \begin{array}{c} \text{Spalte } l \\ \downarrow \end{array} & \\ \mathcal{A}(\mathrm{ST}_l) = & \begin{bmatrix} 0 & & & & & & \\ & \ddots & & & & & \\ & & \ddots & & & & \\ 1 & \cdots & 1 & 0 & 1 & \cdots & 1 \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{bmatrix} & \leftarrow \text{Zeile } l \end{array} \tag{3.1}$$

Der in (3.1) beschriebene Spannbaum ermöglicht einen *link bound broadcast*, in dem die Wurzel l direkt mit allen Nachbarknoten kommuniziert. Ein linearer Prozessorgraph besitzt die in Abbildung 3.1 gezeigte Gestalt. Führt man auf einem linearen Prozessorgraphen einen *link bound broadcast* durch, so ergibt sich die Adjazenzmatrix des vom Knoten l ausgehenden Spannbaumes wie folgt:

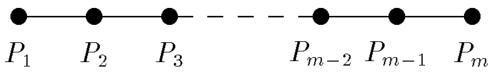

Abbildung 3.1: Linearer Graph mit m Knoten

$$
\begin{array}{cc}
 & \text{Spalte } l \\
 & \downarrow \\
\mathcal{A}(\mathrm{ST}_l) = & \left[\begin{array}{ccccccc}
0 & & & & & & \\
1 & \ddots & & & & & \\
 & \ddots & \ddots & & & & \\
 & & 1 & 0 & 1 & & \\
 & & & & \ddots & \ddots & \\
 & & & & & \ddots & 1 \\
 & & & & & & 0
\end{array}\right] \leftarrow \text{Zeile } l
\end{array} \tag{3.2}
$$

Die Anzahl der Nachbarn eines Knotens ist in einem linearen Prozessorgraphen auf maximal zwei beschränkt. Man sieht unmittelbar, daß ein *broadcast* in einem linearen Graphen maximal $m-1$ Zeiteinheiten in Anspruch nehmen wird.

Da ST_l zyklusfrei ist, aber ausgehend von der Wurzel l jeder Knoten erreicht werden muß, hat jede Spalte mit Ausnahme der Spalte l genau einen Eintrag. Dies bedeutet auch, daß jeder Knoten im Spannbaum ST_l mit Ausnahme der Wurzel genau einen Vorgänger hat. Für jeden Spannbaum ST_l sei nun φ_l die Funktion, die diese Eigenschaft des Spannbaumes beschreibt:

$$\varphi_l : \{1, \ldots, l-1, l+1, \ldots, m\} \to \{1, \ldots, m\}\,.$$

Gilt $\varphi_l(t) = s$, so heißt dies, daß der Knoten t im Spannbaum ST_l den Vorgänger s hat. In Bezug auf diese Funktion läßt sich die Adjazenzmatrix $\mathcal{A}(\mathrm{ST}_l)$ recht einfach als Summe von Elementarmatrizen $I_m^{(l,j)}$ beschreiben. $I_m^{(l,j)}$ hat lediglich an der Stelle (l,j) den Eintrag 1 und ist sonst überall gleich Null. Somit ergibt sich als äquivalente Schreibweise für die Adjazenzmatrix eines Spannbaumes

$$\mathcal{A}(\mathrm{ST}_l) = \sum_{\substack{j=1 \\ j \neq l}}^{m} I_m^{(\varphi_l(j),j)}\,. \tag{3.3}$$

Die Iterationsverfahren in Abschnitt 1.3 sind über dem Vektorraum $\mathbb{R}^n$ definiert. Die Auswirkungen der Kommunikation zwischen den Prozessoren sind in den Darstellungen der synchronen und asynchronen Iterationsverfahren aus Abschnitt 1.3 nicht meßbar, obwohl unterschiedliche Prozessorgraphen auch unterschiedliche Laufzeiten eines Programmes bedingen. Da jeder Prozessor in einem Multiprozessorrechner über einen lokalen Speicher verfügt, ist es sinnvoll, die in diesem Speicher vorliegenden Iterierten eines Verfahrens auch parallel darzustellen. Ein asynchrones, über dem Vektorraum $\mathbb{R}^n$ betrachtetes iteratives Verfahren auf einem Multiprozessorrechner mit m Prozessoren läßt sich somit anschaulich in den Vektorraum $\mathbb{R}^{mn}$ projizieren. Ein weiterer Vorteil dieser Darstellung ist die Einbindung der Eigenschaften des Prozessorgraphen in die Iterationsmatrix des Verfahrens. Dies veranschaulicht ein einfaches Beispiel.

Der gegebene Multiprozessorrechner bestehe aus drei Prozessoren, die als vollständiger Graph miteinander verschaltet sind.

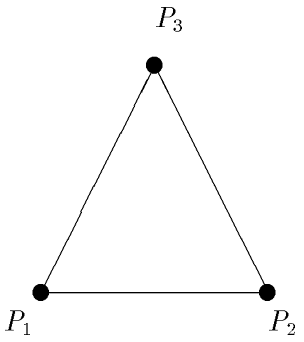

Abbildung 3.2: Vollständiger Graph mit drei Knoten

Nach dem Start des Iterationsverfahrens liegt zu einem Zeitpunkt k in jedem der Prozessoren ein vorläufiger Lösungsvektor $x_k^{(l)}$ für $l = 1, 2, 3$, der zur Berechnung der Iterierten zum Zeitpunkt $k + 1$ benutzt wird. Jeder Prozessor berechnet nun lokal die nächste Iterierte. Gleichzeitig werden die lokal berechneten Ergebnisse aus dem Zeitschritt k unter Berücksichtigung der prozessorspezifischen Gewichte E_l über die vorhandenen Datenleitungen an die Nachbarprozessoren verteilt, d.h. Prozessor P_1 informiert die Prozessoren P_2 und P_3, Prozessor P_2 informiert die Prozessoren P_1 und P_3, und Prozessor P_3 informiert P_1 und P_2.

Diesen Übergang vom Zeitschritt k zum Zeitschritt $k + 1$ verdeutlicht die folgende Graphik:

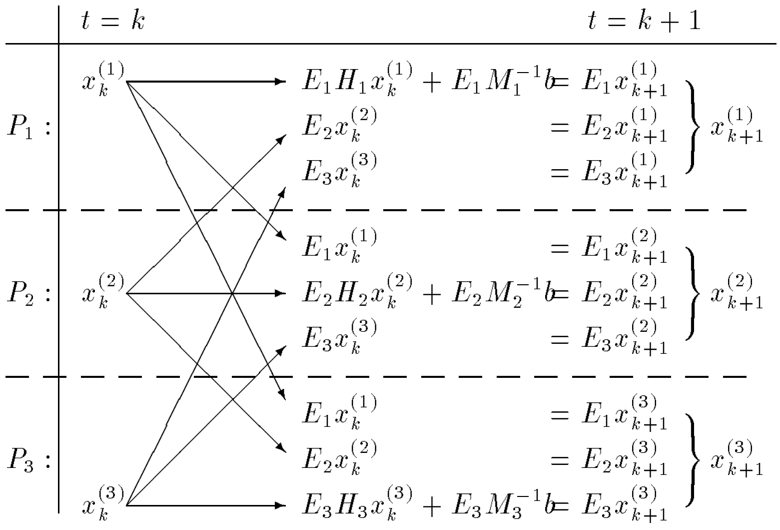

Abbildung 3.3: Zeitschritt von t = k auf t = k+1

Dieser Vorgang entspricht in einer Matrixdarstellung dem Iterationsverfahren

$$\begin{bmatrix} x_{k+1}^{(1)} \\ x_{k+1}^{(2)} \\ x_{k+1}^{(3)} \end{bmatrix} = \begin{bmatrix} E_1 H_1 & E_2 & E_3 \\ E_1 & E_2 H_2 & E_3 \\ E_1 & E_2 & E_3 H_3 \end{bmatrix} \begin{bmatrix} x_k^{(1)} \\ x_k^{(2)} \\ x_k^{(3)} \end{bmatrix} + \begin{bmatrix} E_1 M_1^{-1} b \\ E_2 M_2^{-1} b \\ E_3 M_3^{-1} b \end{bmatrix} \tag{3.4}$$

Die drei $n \times n$ Diagonalblöcke der Iterationsmatrix enthalten die skalierten Iterationsmatrizen $E_l H_l$ der den Prozessoren lokal bekannten schwach re-

gulären Zerlegungen. Dies entspricht dem Anteil der numerischen Iteration. Die Nichtdiagonalblöcke der Matrix beschreiben die Kommunikationskomponente des Algorithmus. Die Position der Gewichte E_l auf den Nichtdiagonalblöcken wird durch die Transponierten der Adjazenzmatrizen $\mathcal{A}(\mathrm{ST}_l)$ gegeben, die in diesem Beispiel gemäß (3.1) gewählt sind.
Eine Verallgemeinerung dieser Vorgehensweise für einen Multiprozessorrechner mit m Prozessoren wird durch die Einführung der folgenden Notationen erleichtert. $A \in M_n(\mathbb{R})$ und $B \in M_m(\mathbb{R})$ seien zwei Matrizen, deren Koeffizienten durch $A = (a_{ij})$ und $B = (b_{ij})$ gegeben sind. Das Kroneckerprodukt $A \otimes B$ dieser Matrizen ist dann definiert als

$$A \otimes B = \left[\begin{array}{ccc|c|ccc} a_{11}b_{11} & \cdots & a_{11}b_{1m} & & a_{1n}b_{11} & \cdots & a_{1n}b_{1m} \\ \vdots & & \vdots & \cdots & \vdots & & \vdots \\ a_{11}b_{m1} & \cdots & a_{11}b_{mm} & & a_{1n}b_{m1} & \cdots & a_{1n}b_{mm} \\ \hline & \vdots & & & & \vdots & \\ \hline a_{n1}b_{11} & \cdots & a_{n1}b_{1m} & & a_{nn}b_{11} & \cdots & a_{nn}b_{1m} \\ \vdots & & \vdots & \cdots & \vdots & & \vdots \\ a_{n1}b_{m1} & \cdots & a_{n1}b_{mm} & & a_{nn}b_{m1} & \cdots & a_{nn}b_{mm} \end{array}\right]. \tag{3.5}$$

Dies entspricht der Darstellung des Tensorproduktes zweier linearer Abbildungen bzgl. einer festen Basis, die in lexikographischer Ordnung angeordnet ist. Eine exakte Definition findet man in [Hal58].
Die endliche direkte Summe von m Kopien des Vektorraumes $\mathbb{R}^n$ werde mit $\bigoplus_{l=1}^m \mathbb{R}^n$ bezeichnet. Jedes Element $y \in \bigoplus_{l=1}^m \mathbb{R}^n$ ist ein m-Tupel $(y_1, \ldots, y_m)$ mit Komponenten $y_l \in \mathbb{R}^n$ für $1 \le l \le m$, das mit

$$y = \bigoplus_{l=1}^{m} y_l$$

bezeichnet wird.
Unter Verwendung dieser Schreibweise ergibt sich ein für einen MP-Rechner mit m Prozessoren verallgemeinertes Modell in der Gestalt eines traditionellen Iterationsverfahrens über dem Vektorraum $\bigoplus_{l=1}^m \mathbb{R}^n$

$$\tilde{x}_{k+1} = \hat{H}\tilde{x}_k + d, \tag{3.6}$$

$\hat{H}$ und d lassen sich aus den Adjazenzmatrizen der *broadcast* Strategie und den Matrizen der schwach regulären Zerlegungen von A konstruieren, so daß gilt:

$$\hat{H} = \sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l M_l^{-1} N_l \tag{3.7}$$

$$d = \bigoplus_{l=1}^{m} E_l M_l^{-1} b \tag{3.8}$$

Eine Teilmenge des $\bigoplus_{l=1}^m \mathbb{R}^n$ bilden die Vektoren $\bigoplus\limits_{l=1}^{m} y$, die aus m identischen Kopien des Vektors $y \in \mathbb{R}^n$ bestehen. Sie bilden einen Untervektorraum von $\bigoplus_{l=1}^m \mathbb{R}^n$, der im weiteren Verlauf der Arbeit mit

$$\mathcal{U} = \left\{ u \in \bigoplus_{l=1}^{m} \mathbb{R}^n \;\middle|\; u = \bigoplus_{l=1}^{m} y \text{ für ein } y \in \mathbb{R}^n \right\} \tag{3.9}$$

bezeichnet wird. Der Abschnitt 3.2 wird zeigen, daß lediglich Elemente aus $\mathcal{U}$ als Fixpunkte von (3.6) in Frage kommen.
Wird das in Kapitel 1 beschriebene Verfahren von O'Leary und White [OW85] in der Praxis auch tatsächlich asynchron ausgeführt, so enthält das Modell (3.6) für das asynchrone Verfahren wiederum eine Synchronität in den Zeittakten der Iteration. Dies bedeutet, daß der langsamste Prozessor die Dauer eines Zeittaktes bestimmt. Die Rechenlast sollte demnach möglichst homogen auf alle Prozessoren verteilt sein, wenn es nicht zu größeren Leerlaufzeiten zum Ende eines jeden Iterationsschritts kommen soll. Diese Annahme ist in der Realität oft erfüllt, da Koeffizientenmatrizen von großen Gleichungssystemen zumeist aus Diskretisierungen elliptischer Differentialgleichungen stammen, die in ihrer Struktur symmetrisch sind. Die Aufteilung der Rechenlast erfolgt somit sinnvollerweise über eine Zerlegung des Indexbereiches der Iterationsmatrix in m gleich große Teilmengen. Weiterhin wird in dem beschriebenen Modell angenommen, daß ein Kommunikationsschritt die gleiche Zeit benötigt wie ein lokaler, numerischer Iterationschritt. Dieser Ansatz ist idealisierend, da lokale Rechenzeit und Kommunikationszeit bei konstanter Problemgröße und wachsender Prozessorzahl gegenläufig sind. Einer sinkenden Rechenzeit steht eine wachsende Kommunikationszeit gegenüber. Wie die Analyse einiger Beispiele in Kapitel 5 jedoch zeigt, ist die Annahme in einem Bereich gültig, in dem die bestmöglichen Iterationszeiten erreicht werden.

3.2 Lösungsmenge und Konsistenz

Das im vorherigen Abschnitt definierte Verfahren entspricht rein formal einem linearen Iterationsverfahren, das aus den Gewichten und Iterationsmatrizen eines Multisplittingverfahrens im $\mathbb{R}^n$ konstruiert wurde. Im folgenden soll nun ein Gleichungssystem gegeben werden, das äquivalent zu der gestellten Anfangsaufgabe (1.1) und zu dem das Verfahren (3.6) konsistent ist.
Notwendige und hinreichende Voraussetzung für die Konvergenz der bisher betrachteten iterativen Verfahren war die Monotonie der Koeffizientenmatrix A des Gleichungssystems (1.1). Dies bedeutet jedoch auch die Existenz einer eindeutigen Lösung des Gleichungssystems, die mit $\xi(b) = A^{-1}b$ bezeichnet werde. Jeder Prozessor behält während des Iterationsvorganges einen temporären Lösungsvektor in seinem lokalen Speicher, der für alle Prozessoren gegen den Vektor $\xi(b)$ konvergieren muß. Dementsprechend definiert man ein Gleichungssystem

$$\hat{A} \bigoplus_{l=1}^{m} x_l = \bigoplus_{l=1}^{m} b \tag{3.10}$$

mit der Koeffizientenmatrix

$$\hat{A} = \sum_{l=1}^{m} I_m^{(l,l)} \otimes A, \tag{3.11}$$

das unter oben genannten Voraussetzungen an A die eindeutige Lösung

$$\bigoplus_{l=1}^{m} \xi(b) = \bigoplus_{l=1}^{m} A^{-1} b \tag{3.12}$$

besitzt. Es bleibt zu zeigen, daß das parallele iterative Verfahren (3.6) und das Gleichungssystem (3.10) konsistent sind und die Folge der Iterierten unabhängig vom Startvektor des Verfahrens gegen die eindeutige Lösung des Gleichungssystems (3.10) konvergiert. Gemäß Satz 1.2 reicht eine Abschätzung des Spektralradius der Iterationsmatrix für einen Konvergenznachweis des Verfahrens.

Satz 3.1

> A sei monoton, und die Zerlegungen $A = M_l - N_l$ seien für $l = 1, \ldots, m$ schwach regulär. Der gerichtete Prozessorgraph G sei einfach und symmetrisch. Eine *broadcast* Strategie auf dem Prozessorgraph sei durch die Spannbäume $\{\mathrm{ST}_1, \ldots, \mathrm{ST}_m\}$ gegeben. Dann gilt für den Spektralradius der Iterationsmatrix $\hat{H}$ aus Gleichung (3.7) die Ungleichung
>
> $$\rho(\hat{H}) < 1.$$

Beweis:

> Die Iterationsmatrix ist nichtnegativ, da nach Voraussetzung die Matrizen E_l sowie H_l für $l = 1, \ldots, m$ nichtnegativ sind. Das Ziel ist es nun, die Existenz eines positiven Vektors u mit der Eigenschaft $\hat{H} u < u$ zu zeigen. Hieraus folgt nach Satz 1.3 sofort $\rho(\hat{H}) < 1$.
>
> Sei nun der Vektor
>
> $$\hat{u} = \bigoplus_{i=1}^{m} A^{-1} \mathbb{1}_n$$
>
> gegeben, wobei $\mathbb{1}_n$ der Vektor der Länge n ist, dessen sämtliche Komponenten gleich 1 sind. $\hat{u}$ ist positiv, da A monoton ist; denn jede Zeile von A^{-1} hat zumindest einen positiven Eintrag, der dafür sorgt, daß der entsprechende Eintrag von $\hat{u}$ positiv ist. Für das Produkt aus der Iterationsmatrix $\hat{H}$ und dem Vektor $\hat{u}$ gilt somit

$$\hat{H} \hat{u} = \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l H_l \right) \hat{u}$$

$$
\begin{aligned}
&= \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l(I - M_l^{-1}A)\right) \hat{u} \\
&= \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l\right) \bigoplus_{l=1}^{m} A^{-1}\mathbb{1}_n \\
&\quad - \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l M_l^{-1} \bigoplus_{l=1}^{m} \mathbb{1}_n \\
&= \left(\sum_{l=1}^{m} (\mathcal{A}^T(\mathrm{ST}_l) + I_m^{(l,l)}) \otimes E_l\right) \bigoplus_{l=1}^{m} A^{-1}\mathbb{1}_n \\
&\quad - \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l M_l^{-1} \bigoplus_{l=1}^{m} \mathbb{1}_n \,.
\end{aligned}
$$

Der erste Term der Differenz entspricht dem Vektor $\hat{u}$, denn nach Definition steht in jeder Spalte der Matrix $\mathcal{A}(\mathrm{ST}_l)$ mit Ausnahme der Spalte l eine 1. Also folgt

$$
\begin{aligned}
\hat{H}\hat{u} &= \bigoplus_{l=1}^{m} \left(\sum_{l=1}^{m} E_l A^{-1}\mathbb{1}_n\right) - \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l M_l^{-1} \bigoplus_{l=1}^{m} \mathbb{1}_n \\
&= \bigoplus_{l=1}^{m} A^{-1}\mathbb{1}_n - \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l M_l^{-1} \bigoplus_{l=1}^{m} \mathbb{1}_n \qquad (3.13) \\
&\leq \hat{u} \,.
\end{aligned}
$$

Die Gleichung (3.13) gibt zusätzlich Aufschluß darüber, für welche Komponenten der Vektoren strikte Ungleichheit erfüllt ist. Für alle Indizes $k = (p-1)n + q$ mit $1 \leq p \leq m$ und $1 \leq q \leq n$ gilt

$$
\begin{aligned}
(\hat{H}\hat{u})_k &< \hat{u}_k, \text{ falls } (E_p)_{qq} \neq 0, \\
(\hat{H}\hat{u})_k &= \hat{u}_k, \text{ falls } (E_p)_{qq} = 0.
\end{aligned}
$$

Der Indexbereich $\mathcal{I} = \{1, \ldots, mn\}$ der Matrix $\hat{H}$ zerfällt in zwei disjunkte Teilmengen

$$
\begin{aligned}
\sigma &= \{k \in \mathcal{I} | k = (p-1)n + q \text{ und } (E_p)_{qq} \neq 0\}, \\
\tau &= \{k \in \mathcal{I} | k = (p-1)n + q \text{ und } (E_p)_{qq} = 0\}.
\end{aligned}
$$

Aus der Definition der Gewichte E_l folgt, daß die Menge σ nicht leer ist.

Mit der Bezeichnung $u = A^{-1}\mathbb{1}_n$ gilt $\hat{u} = \bigoplus_{l=1}^{m} u$. Da $\hat{u}$ positiv ist, existiert zu der Diagonalmatrix

$$
U = \sum_{l=1}^{m} I_m^{(l,l)} \otimes \mathrm{diag}(u)
$$

die Inverse U^{-1}, und aus $\hat{H}\hat{u} \leq \hat{u}$ folgt

$$U^{-1}\hat{H}U\mathbb{1}_{mn} = U^{-1}\hat{H}\hat{u} \leq U^{-1}\hat{u} = \mathbb{1}_{mn}. \tag{3.14}$$

Die Ähnlichkeitstransformation operiert auf den Zeilen $k \in \tau$ trivial, so daß die Matrix $\tilde{H} = U^{-1}\hat{H}U$ das schwache Zeilensummenkriterium

$$(\tilde{H}\mathbb{1}_{mn})_k < 1, \text{ falls } k \in \sigma, \tag{3.15}$$
$$(\tilde{H}\mathbb{1}_{mn})_k = 1, \text{ falls } k \in \tau \tag{3.16}$$

erfüllt. Gemäß einer Bemerkung von Elsner [Els69] erfüllt eine Matrix M, die das schwache Zeilensummenkriterium erfüllt, auch das verallgemeinerte Zeilensummenkriterium

$$Mx < x \text{ für einen Vektor } x > 0, \tag{3.17}$$

falls zu jeder Teilmenge $\tilde{\tau} \subseteq \tau$ ein Indexpaar (i,j) mit $i \in \tilde{\tau}$ und $j \notin \tilde{\tau}$ existiert, so daß $(\tilde{H})_{ij} \neq 0$ ist.

Kann man nun zeigen, daß das verallgemeinerte Zeilensummenkriterium für $\tilde{H}$ gilt, so folgt aus Satz 1.3 sofort $\rho(\tilde{H}) < 1$ und somit auch $\rho(\hat{H}) < 1$. Damit ist der Satz 3.2 bewiesen.

Unter obigen Voraussetzungen an $\tilde{H}$ nehme man an, daß die für das verallgemeinerte Zeilensummenkriterium benötigte Zusatzbedingung nicht erfüllt ist, d.h. für eine Teilmenge $\tilde{\tau} \subseteq \tau$ sei

$$(\tilde{H})_{ij} = 0 \text{ für } i \in \tilde{\tau}, j \notin \tilde{\tau}. \tag{3.18}$$

Aus dieser Teilmenge $\tilde{\tau} \subseteq \tau$ wähle man nun einen Index $i_0 \in \tilde{\tau}$. Mit geeigneten p_0 und q gilt

$$i_0 = (p_0 - 1)n + q, \quad 1 \leq p_0 \leq m, \quad 1 \leq q \leq n$$

und es folgt aus der Definition von τ $(E_{p_0})_{qq} = 0$. Da die Gewichte E_l nach Voraussetzung aber die Bedingung $\sum_{l=1}^{m} E_l = I_n$ erfüllen, gilt für das q-te Diagonalelement

$$(I)_{qq} = \sum_{(l-1)n+q \in \sigma} (E_l)_{qq},$$

d.h. es gibt mindestens eine Zahl $p(q)$ mit $(E_{p(q)})_{qq} \neq 0$. Da $p_0 \neq p(q)$ und in jeder Zeile der Matrix $\mathcal{A}^T(\mathrm{ST}_{p(q)})$ mit Ausnahme der Zeile $p(q)$ eine 1 steht, existiert ein Spaltenindex $i_1 = (p_1 - 1)n + q$ der Matrix $\tilde{H}$ mit $(\tilde{H})_{i_0 i_1} \neq 0$ und $1 \leq p_1 \leq m$. Es gilt $i_0 \neq i_1$, denn ansonsten wäre

$$\begin{aligned} 0 \neq (\tilde{H})_{i_0 i_0} &= \left(E_{p_0}\mathrm{diag}(u)^{-1}H_{p_0}\mathrm{diag}(u)\right)_{qq} \\ &= (E_{p_0})_{qq}\left(\mathrm{diag}(u)^{-1}H_{p_0}\mathrm{diag}(u)\right)_{qq} \\ &= 0 \end{aligned}$$

Aus der Annahme (3.18) folgt, daß $i_1 \in \tilde{\tau}$, so daß wiederum ein Index $i_2 = (p_2 - 1)n + q$ mit der Eigenschaft $i_2 \in \tilde{\tau} \backslash \{i_0, i_1\}$ konstruierbar ist. Dieser Prozeß ist beliebig oft fortsetzbar. Dies führt aber zum Widerspruch, da $\tilde{\tau}$ endlich ist. Also erfüllt $\tilde{H}$ das verallgemeinerte Zeilensummenkriterium und die Behauptung folgt.

$\boxed{\text{q.e.d.}}$

Es bleibt zu zeigen, daß das Verfahren konsistent ist. Sei nun

$$\xi = \bigoplus_{l=1}^{m} \xi(b) = \bigoplus_{l=1}^{m} A^{-1} b$$

die eindeutige Lösung des Gleichungssystems (3.10). Dann gilt mit (3.7) und (3.8)

$$\begin{aligned}
&\hat{H}\xi + d \\
&= \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l H_l \right) \bigoplus_{l=1}^{m} A^{-1} b + \bigoplus_{l=1}^{m} E_l M_l^{-1} b \\
&= \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l (I - M_l^{-1} A) \right) \bigoplus_{l=1}^{m} A^{-1} b \\
&\quad + \bigoplus_{l=1}^{m} E_l M_l^{-1} b \\
&= \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l \right) \bigoplus_{l=1}^{m} A^{-1} b \\
&\quad - \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l M_l^{-1} \right) \bigoplus_{l=1}^{m} A^{-1} b + \bigoplus_{l=1}^{m} E_l M_l^{-1} b \\
&= \left(\sum_{l=1}^{m} \left(\mathcal{A}^T(\mathrm{ST}_l) + I_m^{(l,l)} \right) \otimes E_l \right) \bigoplus_{l=1}^{m} A^{-1} b \\
&\quad - \bigoplus_{l=1}^{m} E_l M_l^{-1} b + \bigoplus_{l=1}^{m} E_l M_l^{-1} b \\
&= \left(\sum_{l=1}^{m} \left(\mathcal{A}^T(\mathrm{ST}_l) + I_m^{(l,l)} \right) \otimes E_l \right) \bigoplus_{l=1}^{m} A^{-1} b \,. \qquad (3.19)
\end{aligned}$$

Da nach Definition in jeder Zeile von $\mathcal{A}^T(\mathrm{ST}_l)$ mit Ausnahme der Zeile l eine 1 steht, vereinfacht sich (3.19) zu

$$\begin{aligned}
\hat{H}\xi + d &= \bigoplus_{i=1}^{m} \left(\sum_{l=1}^{m} E_l A^{-1} b \right) \\
&= \bigoplus_{i=1}^{m} \left(\sum_{l=1}^{m} E_l \right) A^{-1} b \\
&= \bigoplus_{i=1}^{m} A^{-1} b \,.
\end{aligned}$$

ξ ist also ein Fixpunkt des Verfahrens (3.6). Die letzte Umformung ist nur dann gültig, wenn Lösungen der Gleichung (3.10) aus dem in (3.9) definierten Vektorraum $\mathcal{U}$ stammen.

Kapitel 4

Block-Jacobi-Iterationsverfahren

4.1 Block-Jacobi-Zerlegungen

Aus praktischen Erwägungen ist eine möglichst einfache reguläre Zerlegung der Koeffizientenmatrix des Gleichungssystems (1.1) von Vorteil. Voraussetzung hierfür ist eine strengere Definition der bereits in Abschnitt 1.3 benutzten Gewichte E_l.

Definition 4.1

> Für $1 \leq l \leq m$ heißen Diagonalmatrizen E_l Gewichtsmatrizen des Block-Jacobi-Verfahrens, wenn sie die Bedingungen
>
> $$\sum_{l=1}^{m} E_l = I_n \,, \tag{4.1}$$
>
> $$E_i E_j = 0 \text{ für } i \neq j \tag{4.2}$$
>
> erfüllen.

Die Bedingung (4.2) impliziert eine Zerlegung des Indexbereiches $\mathcal{I} = \{1, \ldots, n\}$ in disjunkte Teilmengen S_l, die o.B.d.A. aus konsekutiven Indizes bestehen sollen.

Diese Zerlegung des Indexbereiches liefert einen Steuermechanismus für die Rechenauslastung der Prozessoren, da jeder Nulleintrag auf der Diagonale der Matrix E_l die Berechnung des korrespondierenden Koeffizienten in der Iterierten überflüssig macht.

Unter Benutzung dieser Zerlegungen des Indexbereiches ergeben sich Block-Jacobi-Zerlegungen als Spezialfälle der allgemeineren, oben vorgestellten, schwach regulären Zerlegungen. Besteht ein gegebener Multiprozessorrechner aus m Prozessoren, so beschreibt die folgende Definition die Block-Jacobi-Zerlegung einer Matrix A:

Definition 4.2

Die Koeffizientenmatrix $A = (a_{ij})$ eines Gleichungssystems (1.1) werde für $l = 1, \ldots, m$ in die Summen $A = M_l - N_l$ zerlegt. Erfüllen die Koeffizienten der Matrizen M_l die Bedingungen

$$m_{ij} = \begin{cases} a_{ij} & falls \quad i = j \quad oder \quad i,j \in S_l \\ 0 & sonst \end{cases},$$

so heißen die Zerlegungen *Block-Jacobi-Zerlegungen.*

Hieraus folgt entsprechend für die Matrizen N_l

$$n_{ij} = \begin{cases} 0 & falls \quad i = j \quad oder \quad i,j \in S_l \\ -a_{ij} & sonst \end{cases}.$$

Eine in den Sätzen dieses Kapitels oft benutzte Eigenschaft der Iterationsmatrix eines Block-Jacobi-Verfahrens beschreibt die folgende

Bemerkung 4.1

Für $1 \leq l \leq m$ seien $H_l = (h_{ij})$ die aus einer Block-Jacobi-Zerlegung gewonnenen Komponenten der Iterationsmatrix.

Dann ist

$$h_{ij} = 0$$

für alle Indexpaare mit $i, j \in S_l$.

Beweis:

Für ein Indexpaar (i, j) hat ein Eintrag der Matrix $H_l = M_l^{-1} N_l$ die Gestalt

$$(H_l)_{ij} = \sum_{k=1}^{n} (M_l^{-1})_{ik} (N_l)_{kj} \tag{4.3}$$

Da beide Indizes i und j zur Menge S_l gehören, folgt aus der Definition 4.2, daß $(M_l^{-1})_{ik} = 0$ für alle $k \notin S_l$ und $(N_l)_{kj} = 0$ für alle $k \in S_l$.

$\boxed{\text{q.e.d.}}$

4.2 Block-Jacobi-Multisplitting auf vollständigem Graphen

Ausgehend von einer monotonen Koeffizientenmatrix A des Gleichungssystems (1.1) erhält man aus m schwach regulären Block-Jacobi-Zerlegungen die Iterationsmatrix

$$H = \sum_{l=1}^{m} E_l H_l \tag{4.4}$$

eines Block-Jacobi-Verfahrens.
Dieselben Block-Jacobi-Zerlegungen betrachte man nun auf einem Multiprozessorrechner, dessen Prozessorgraph durch einen vollständigen Graphen mit m Knoten beschrieben ist. Erfolgt der Datentransfer gemäß einer *broadcast* Strategie, deren Elemente Adjazenzmatrizen der Gestalt (3.1) besitzen, so erhält man für die Iterationsmatrix $\hat{H}$ des parallelen Verfahrens aus Gleichung (3.7) die explizite Darstellung

$$\hat{H}_{K_m} = \begin{bmatrix} E_1H_1 & E_2 & \cdots & E_{m-1} & E_m \\ E_1 & E_2H_2 & & E_{m-1} & E_m \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ E_1 & E_2 & & E_{m-1}H_{m-1} & E_m \\ E_1 & E_2 & \cdots & E_{m-1} & E_mH_m \end{bmatrix} . \tag{4.5}$$

Der Index K_m weist darauf hin, daß (4.5) die Iterationsmatrix für einen vollständigen Graphen mit m Knoten beschreibt. Den Zusammenhang zwischen dem Spektrum der Matrix $\hat{H}_{K_m}$ und dem Spektrum der Iterationsmatrix H beschreibt der folgende Satz.

Satz 4.1

> A sei eine monotone Matrix aus $M_n(\mathbb{R})$. $A = M_l - N_l$ seien schwach reguläre Block-Jacobi-Zerlegungen für $l = 1, \ldots, m$. Die Gewichte E_l entsprechen der Definition 4.1 . Der Prozessorgraph G eines Multiprozessorrechners mit $m \geq 2$ Knoten sei vollständig und mit einer *broadcast* Strategie versehen, deren Elemente Adjazenzmatrizen der Gestalt (3.1) besitzen. Ist $\lambda \in \mathbb{C}\backslash\{0\}$ ein Eigenwert der Iterationsmatrix H des Block-Jacobi-Verfahrens, so sind die beiden Wurzeln $\sqrt{\lambda}$ Eigenwerte der Iterationsmatrix $\hat{H}_{K_m}$ des asynchronen Block-Jacobi-Verfahrens. Alle von Null verschiedenen Eigenwerte von $\hat{H}_{K_m}$ erhält man auf diese Weise.

Beweis:

Sei zunächst λ ein von Null verschiedener Eigenwert der Matrix H zum Eigenvektor $u \in \mathbb{R}^n$, d.h.

$$\left(\sum_{l=1}^{m} E_l H_l\right) u = \lambda u. \tag{4.6}$$

Dann folgt unter Benutzung der Orthogonalitätsbeziehung (4.2) aus der Eigenwertgleichung (4.6), daß

$$E_l H_l u = \lambda E_l u \tag{4.7}$$

für $l = 1, \ldots, m$ gilt.

Ist nun $\sqrt{\lambda}$ eine der beiden Wurzeln von λ, so sei der Vektor $\hat{u}$ definiert als

$$\hat{u} = \begin{bmatrix} \sqrt{\lambda} E_1 u + \sum\limits_{\substack{l=1 \\ l \neq 1}}^{m} E_l u \\ \vdots \\ \vdots \\ \sqrt{\lambda} E_m u + \sum\limits_{\substack{l=1 \\ l \neq m}}^{m} E_l u \end{bmatrix}. \tag{4.8}$$

Berechnet man nun das Produkt der Matrix $\hat{H}_{K_m}$ mit dem Vektor $\hat{u}$, so erhält man

$$\hat{H}_{K_m} \hat{u} = \begin{bmatrix} E_1 H_1 \left(\sqrt{\lambda} E_1 u + \sum\limits_{\substack{l=1 \\ l \neq 1}}^{m} E_l u \right) \\ + \sum\limits_{\substack{l=1 \\ l \neq 1}}^{m} E_l \left(\sqrt{\lambda} E_l u + \sum\limits_{\substack{k=1 \\ k \neq l}}^{m} E_k u \right) \\ \vdots \\ \vdots \\ E_m H_m \left(\sqrt{\lambda} E_m u + \sum\limits_{\substack{l=1 \\ l \neq m}}^{m} E_l u \right) \\ + \sum\limits_{\substack{l=1 \\ l \neq m}}^{m} E_l \left(\sqrt{\lambda} E_l u + \sum\limits_{\substack{k=1 \\ k \neq l}}^{m} E_k u \right) \end{bmatrix}. \tag{4.9}$$

Für jeden Index $1 \leq l \leq m$ taucht in (4.9) der Term $E_l H_l E_l$ auf, der jedoch gemäß Bemerkung 4.1 gleich Null ist. Aufgrund dieser Überlegung und der Beziehung (4.2) vereinfacht sich die Gleichung (4.9) zu

$$\hat{H}_{K_m}\hat{u} \quad = \quad \begin{bmatrix} E_1 H_1 u + \sqrt{\lambda} \sum\limits_{\substack{l=1\\ l\neq 1}}^{m} E_l u \\ \vdots \\ \vdots \\ E_m H_m u + \sqrt{\lambda} \sum\limits_{\substack{l=1\\ l\neq m}}^{m} E_l u \end{bmatrix}. \tag{4.10}$$

Einsetzen von (4.7) in (4.10) führt zu

$$\begin{aligned} \hat{H}_{K_m}\hat{u} \quad &= \begin{bmatrix} \lambda E_1 u + \sqrt{\lambda} \sum\limits_{\substack{l=1\\ l\neq 1}}^{m} E_l u \\ \vdots \\ \vdots \\ \lambda E_m u + \sqrt{\lambda} \sum\limits_{\substack{l=1\\ l\neq m}}^{m} E_l u \end{bmatrix} \\ &= \sqrt{\lambda} \begin{bmatrix} \sqrt{\lambda} E_1 u + \sum\limits_{\substack{l=1\\ l\neq 1}}^{m} E_l u \\ \vdots \\ \vdots \\ \sqrt{\lambda} E_m u + \sum\limits_{\substack{l=1\\ l\neq m}}^{m} E_l u \end{bmatrix}. \end{aligned}$$

Somit ist gezeigt, daß

$$\hat{H}_{K_m}\hat{u} = \sqrt{\lambda}\hat{u}, \tag{4.11}$$

d.h. $\sqrt{\lambda}$ ist ein Eigenwert von $\hat{H}_{K_m}$ zum Eigenvektor $\hat{u}$.

Sei nun umgekehrt $\lambda \in \mathbb{C}\backslash\{0\}$ ein Eigenwert der Matrix $\hat{H}_{K_m}$ zu einem Eigenvektor u, d.h.

$$\hat{H}_{K_m} u = \lambda u.$$

Zerlegt man den Eigenvektor u in m Blöcke u_i der Länge n, so erhält man eine ausführlichere Darstellung der Eigenwertgleichung

$$
\begin{bmatrix} E_1 H_1 u_1 + \sum\limits_{\substack{l=1 \\ l\neq 1}}^{m} E_l u_l \\ \vdots \\ \vdots \\ E_m H_m u_m + \sum\limits_{\substack{l=1 \\ l\neq m}}^{m} E_l u_l \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}. \tag{4.12}
$$

Für $1 \leq k, j \leq m$ ergeben sich aus (4.12) die Gleichungen

$$
\begin{aligned}
E_k H_k u_k &= \lambda E_k u_k, && (4.13) \\
E_k u_k &= \lambda E_k u_j \text{ falls } k \neq j, && (4.14)
\end{aligned}
$$

denn mit Hilfe der Orthogonalitätsrelationen (4.2) vereinfacht sich jede Blockzeile zu:

$$
E_k \left(E_j H_j u_j + \sum_{\substack{l=1 \\ l\neq j}}^{m} E_l u_l \right) = \begin{cases} E_k H_k u_k & \textit{falls} \quad k = j \\ E_k u_k & \textit{falls} \quad k \neq j \end{cases} .
$$

Ist $k = 1$, so folgt aus Gleichung (4.13)

$$
E_1 H_1 u_1 = \lambda^2 E_1 \frac{1}{\lambda} u_1
$$

bzw. unter Benutzung von (4.1) und der Bemerkung 4.1

$$
E_1 H_1 \left(\frac{1}{\lambda} E_1 u_1 + \sum_{l=2}^{m} E_l u_1 \right) = \lambda^2 E_1 \frac{1}{\lambda} u_1. \tag{4.15}
$$

Für $k \neq 1$ sollen nun in Gleichung (4.13) die Vektoren u_k durch den Vektor u_1 ersetzt werden. Wählt man speziell $j = 1$, so liefert (4.14)

$$
E_k H_k u_k = \lambda^2 E_k u_1. \tag{4.16}
$$

Für die linke Seite der Gleichung (4.16) erhält man mit (4.1), (4.14) und der Bemerkung 4.1 die Gleichung

$$
\begin{aligned}
E_k H_k u_k &= E_k H_k \left(\sum_{l=1}^{m} E_l u_k \right) && (4.17) \\
&= E_k H_k E_1 u_k + E_k H_k E_k u_k + E_k H_k \sum_{\substack{l=1 \\ l\neq 1,k}}^{m} E_l u_k \\
&= E_k H_k \frac{1}{\lambda} E_1 u_1 + E_k H_k \sum_{l=2}^{m} E_l u_1 \\
&= E_k H_k \left(\frac{1}{\lambda} E_1 u_1 + \sum_{l=2}^{m} E_l u_1 \right).
\end{aligned}
$$

Zusammengefaßt ergibt sich für jeden Index k mit $2 \leq k \leq m$ aus Gleichung (4.17) und (4.16) die Darstellung

$$E_k H_k \left(\frac{1}{\lambda} E_1 u_1 + \sum_{l=2}^{m} E_l u_1 \right) = \lambda^2 E_k u_1. \tag{4.18}$$

Summation der Gleichungen (4.15) und (4.18) für alle Indizes $2 \leq k \leq m$ ergibt die gewünschte Eigenwertgleichung

$$\left(\sum_{l=1}^{m} E_l H_l \right) \left(\frac{1}{\lambda} E_1 u_1 + \sum_{l=2}^{m} E_l u_1 \right) = \lambda^2 \left(\frac{1}{\lambda} E_1 u_1 + \sum_{l=2}^{m} E_l u_1 \right).$$

q.e.d.

Aus der Monotonie der Wurzelfunktion folgt, daß speziell der Spektralradius von $\hat{H}_{K_m}$ die Gleichung $\rho(\hat{H}_{K_m}) = \sqrt{\rho(H)}$ erfüllt. Diese Behauptung gilt nicht für Iterationsmatrizen, die aus allgemeineren schwach regulären Zerlegungen gewonnen werden. In diesem Fall berechnet sich das Produkt aus $\hat{H}_{K_m}$ und dem Vektor $\hat{u}$ aus (4.8) zu

$$\begin{aligned}
\hat{H}_{K_m} \hat{u} &= \begin{bmatrix} \sqrt{\rho(H)} E_1 H_1 E_1 u + E_1 H_1 \left(\sum_{\substack{l=1 \\ l \neq 1}}^{m} E_l u \right) \\ + \sqrt{\rho(H)} \left(\sum_{\substack{l=1 \\ l \neq 1}}^{m} E_l u \right) \\ \vdots \\ \vdots \\ \sqrt{\rho(H)} E_m H_m E_m u + E_m H_m \left(\sum_{\substack{l=1 \\ l \neq m}}^{m} E_l u \right) \\ + \sqrt{\rho(H)} \left(\sum_{\substack{l=1 \\ l \neq m}}^{m} E_l u \right) \end{bmatrix} \\
&= \begin{bmatrix} \sqrt{\rho(H)} E_1 H_1 E_1 u + E_1 H_1 u - E_1 H_1 E_1 u \\ + \sqrt{\rho(H)} \left(\sum_{\substack{l=1 \\ l \neq 1}}^{m} E_l u \right) \\ \vdots \\ \vdots \\ \sqrt{\rho(H)} E_m H_m E_m u + E_m H_m u - E_m H_m E_m u \\ + \sqrt{\rho(H)} \left(\sum_{\substack{l=1 \\ l \neq m}}^{m} E_l u \right) \end{bmatrix} \\
&= \left(\sqrt{\rho(H)} - 1 \right) \begin{bmatrix} E_1 H_1 E_1 u \\ \vdots \\ \vdots \\ E_m H_m E_m u \end{bmatrix} + \sqrt{\rho(H)} \hat{u}.
\end{aligned}$$

Der entscheidende Unterschied im Vergleich zum Satz 4.1 ist, daß bei der Berechnung von $\hat{H}_{K_m}\hat{u}$ für allgemeinere schwach reguläre Zerlegungen die Bemerkung 4.1 nicht mehr gilt.
Da $\sqrt{\rho(H)} < 1$ und die Matrizen H_l für $1 \leq l \leq m$ nichtnegativ sind, folgt

$$\hat{H}_{K_m}\hat{u} \leq \sqrt{\rho(H)}\hat{u}. \tag{4.19}$$

Die Voraussetzungen für Satz 1.8 sind somit erfüllt und es folgt $\rho(\hat{H}_{K_m}) \leq \sqrt{\rho(H)}$. Das folgende Beispiel zeigt, daß auch tatsächlich Matrizen mit $\rho(\hat{H}_{K_m}) < \sqrt{\rho(H)}$ existieren.
Die Matrix

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \in M_2(\mathbb{R})$$

ist eine M-Matrix, die sich wie folgt regulär zerlegen läßt:

$$A = M - N = \begin{bmatrix} 2 & -\frac{1}{2} \\ -\frac{1}{2} & 2 \end{bmatrix} - \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix}. \tag{4.20}$$

Sind die zwei Gewichtsmatrizen

$$E_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad E_2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.21}$$

gegeben, so ergibt sich die Iterationsmatrix

$$H = E_1 M^{-1} N + E_2 M^{-1} N = \begin{bmatrix} \frac{1}{15} & \frac{4}{15} \\ \frac{4}{15} & \frac{1}{15} \end{bmatrix} \tag{4.22}$$

mit den Eigenwerten $\lambda_1 = \frac{1}{3}$ und $\lambda_2 = -\frac{1}{5}$. Gemäß (3.7) berechnet sich die Iterationsmatrix für einen vollständigen Graphen aus zwei Prozessoren:

$$\hat{H}_{K_2} = \begin{bmatrix} E_1 M^{-1} N & E_2 \\ E_1 & E_2 M^{-1} N \end{bmatrix} = \left[\begin{array}{cc|cc} \frac{1}{15} & \frac{4}{15} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{4}{15} & \frac{1}{15} \end{array}\right]. \tag{4.23}$$

Die Matrix $\hat{H}_{K_2}$ hat die Eigenwerte $\lambda_1 = \frac{1}{30} + \frac{\sqrt{241}}{30}$, $\lambda_2 = \frac{1}{30} - \frac{\sqrt{241}}{30}$, $\lambda_3 = \frac{1}{30} + i\,\frac{\sqrt{239}}{30}$ und $\lambda_4 = \frac{1}{30} - i\,\frac{\sqrt{239}}{30}$.
Für beide Iterationsmatrizen ist λ_1 jeweils der Spektralradius, und es gilt die Beziehung

$$\sqrt{\rho(H)} = \frac{\sqrt{3}}{3} = \frac{10\sqrt{3}}{30} = \frac{\sqrt{3}}{30} + \frac{\sqrt{243}}{30} > \frac{1}{30} + \frac{\sqrt{241}}{30} = \rho(\hat{H}_{K_m}).$$

Da nun der Spektralradius des Iterationsverfahrens (3.6) auf einem vollständigen Prozessorgraphen bekannt ist, stellt sich die Frage nach dem Spektralradius der Iterationsmatrix $\hat{H}$ für eine beliebige *broadcast* Strategie. Zur Beantwortung dieser Frage wird das folgende Ergebnis aus der Theorie nichtnegativer Matrizen benötigt.

Satz 4.2

> A sei eine nichtnegative Matrix, $\rho(A) > 0$ sei Perron-Eigenwert von A zum positiven Eigenvektor z. Definiert man die Diagonalmatrix
>
> $$Z = \operatorname{diag}(z_1, \ldots, z_n),$$
>
> wobei die Diagonalelemente z_i die Komponenten des Eigenvektors z sind, so gilt:
>
> $$\frac{1}{\rho(A)} Z^{-1} A Z = P$$
>
> und $P = (p_{ij})$ ist eine stochastische Matrix, d.h. eine nichtnegative Matrix mit
>
> $$\sum_{j=1}^{n} p_{ij} = 1 \tag{4.24}$$
>
> für alle Zeilen $1 \leq i \leq n$.

Beweis: siehe [BP79].

Gleichung (4.24) und der Quotientensatz 1.5 zeigen, daß der Spektralradius einer stochastischen Matrix gleich 1 ist. Im folgenden soll zunächst noch einmal ein Block-Jacobi-Verfahren auf einem vollständigen Prozessorgraphen betrachtet werden, wobei aber die Matrix $H = \sum_{l=1}^{m} E_l H_l$ zusätzlich irreduzibel sei. Dann ist nach dem Satz von Perron-Frobenius $\rho(H) > 0$, und es existiert ein positiver Eigenvektor $u > 0$ zu $\rho(H)$. Aus dem oben bewiesenen Satz 4.1 folgt dann, daß die Matrix $\hat{H}_{K_m}$ einen Eigenwert $\rho(\hat{H}_{K_m}) = \sqrt{\rho(H)}$ hat. Ebenfalls aus Satz 4.1 ergibt sich die Existenz eines dazu korrespondierenden positiven Eigenvektors

$$\hat{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} \rho(\hat{H}_{K_m}) E_1 u + \sum\limits_{\substack{l=1 \\ l \neq 1}}^{m} E_l u \\ \vdots \\ \vdots \\ \rho(\hat{H}_{K_m}) E_m u + \sum\limits_{\substack{l=1 \\ l \neq m}}^{m} E_l u \end{bmatrix}.$$

Sei nun

$$\begin{aligned} D_i &= \operatorname{diag}(u_i) && (4.25) \\ &= \left(\rho(\hat{H}_{K_m}) E_i + \sum_{\substack{l=1 \\ l \neq i}}^{m} E_l \right) \operatorname{diag}(u) && (4.26) \end{aligned}$$

für $i = 1, \ldots, m$ sowie $\hat{U}$ die aus den Diagonalmatrizen D_i gebildete Blockdiagonalmatrix

$$\hat{U} = \begin{bmatrix} D_1 & & \\ & \ddots & \\ & & D_m \end{bmatrix}. \tag{4.27}$$

Wendet man nun den Satz 4.2 speziell auf die Matrix $\hat{H}_{K_m}$ an, so gilt

$$\begin{aligned} \hat{U}^{-1}\hat{H}_{K_m}\hat{U} &= \\ & \begin{bmatrix} D_1^{-1}E_1H_1D_1 & \cdots & D_1^{-1}E_mD_m \\ D_2^{-1}E_1D_1 & & D_2^{-1}E_mD_m \\ \vdots & \ddots & \vdots \\ D_{m-1}^{-1}E_1D_1 & & D_{m-1}^{-1}E_mD_m \\ D_m^{-1}E_1D_1 & \cdots & D_m^{-1}E_mH_mD_m \end{bmatrix} \\ &= \rho(\hat{H}_{K_m})\hat{P}_{K_m}. \end{aligned}$$

Die Nebendiagonalblöcke der Matrix $\hat{U}^{-1}\hat{H}_{K_m}\hat{U}$ lassen sich noch weiter vereinfachen. Zu einem Indexpaar (i,j) mit $i \neq j$ berechnet sich der entsprechende Blockeintrag als

$$\begin{aligned} D_i^{-1}E_jD_j &= \\ \operatorname{diag}(u)^{-1} & \left(\frac{1}{\rho(\hat{H}_{K_m})}E_i + \sum_{\substack{l=1 \\ i\neq l}}^{m} E_l\right) E_j \left(\rho(\hat{H}_{K_m})E_j + \sum_{\substack{l=1 \\ j\neq l}}^{m} E_l\right) \operatorname{diag}(u) \\ &= \operatorname{diag}(u)^{-1}\left(\frac{1}{\rho(\hat{H}_{K_m})}E_i + \sum_{\substack{k=1 \\ i\neq k}}^{m} E_k\right)\rho(\hat{H}_{K_m})E_j\operatorname{diag}(u) \\ &= \operatorname{diag}(u)^{-1}\rho(\hat{H}_{K_m})E_j\operatorname{diag}(u) \\ &= \rho(\hat{H}_{K_m})E_j. \end{aligned} \tag{4.28}$$

In jeder Blockzeile i der Matrix $\hat{P}_{K_m}$ findet man also alle Gewichte E_j mit Ausnahme der Matrix E_i. Da $\hat{P}_{K_m}$ stochastisch ist und die Diagonalelemente der Matrizen E_j gleich 1 sind und somit bereits der vollen Zeilensumme entsprechen, folgt, daß der Diagonalblock $D_i^{-1}E_iH_iD_i$ lediglich in den durch die Indexmenge S_i vorgegebenen Zeilen von Null verschiedene Einträge besitzt. Das folgende einfache Beispiel soll zur Verdeutlichung der Struktur von $\hat{P}_{K_m}$ dienen. Die Iterationsmatrix H eines Block-Jacobi-Verfahrens sei aus $M_6(\mathbb{R})$. Der Prozessorgraph des gegebenen Parallelrechners sei ein K_3. Der Indexbereich $I = \{1, \ldots, 6\}$ wird in die Teilmengen $S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$ und $S_3 = \{5, 6\}$ untergliedert, so daß die Gewichte die Gestalt

$$E_1 = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 0 & & & \\ & & & 0 & & \\ & & & & 0 & \\ & & & & & 0 \end{bmatrix} \qquad E_2 = \begin{bmatrix} 0 & & & & & \\ & 0 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 0 & \\ & & & & & 0 \end{bmatrix}$$

$$E_3 = \begin{bmatrix} 0 & & & & & \\ & 0 & & & & \\ & & 0 & & & \\ & & & 0 & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}$$

haben. Da der betrachtete Parallelrechner eine 3-Prozessor-Maschine ist, ist $\hat{P}_{K_3} \in M_{18}(\mathbb{R})$ und hat gemäß der oben ausgeführten Berechnungen die Gestalt:

$$\hat{P}_{K_3} = \left[\begin{array}{cccccc|cccccc|cccccc}
* & * & * & * & * & * & 0 & & & & & & 0 & & & & & \\
* & * & * & * & * & * & & 0 & & & & & & 0 & & & & \\
 & & & & & & & & 1 & & & & & & 0 & & & \\
 & & & & & & & & & 1 & & & & & & 0 & & \\
 & & & & & & & & & & 0 & & & & & & 1 & \\
 & & & & & & & & & & & 0 & & & & & & 1 \\
\hline
1 & & & & & & & & & & & & 0 & & & & & \\
 & 1 & & & & & & & & & & & & 0 & & & & \\
 & & 0 & & & & * & * & * & * & * & * & & & 0 & & & \\
 & & & 0 & & & * & * & * & * & * & * & & & & 0 & & \\
 & & & & 0 & & & & & & & & & & & & 1 & \\
 & & & & & 0 & & & & & & & & & & & & 1 \\
\hline
1 & & & & & & 0 & & & & & & & & & & & \\
 & 1 & & & & & & 0 & & & & & & & & & & \\
 & & 0 & & & & & & 1 & & & & & & & & & \\
 & & & 0 & & & & & & 1 & & & & & & & & \\
 & & & & 0 & & & & & & 0 & & * & * & * & * & * & * \\
 & & & & & 0 & & & & & & 0 & * & * & * & * & * & *
\end{array}\right]$$

Die mit $*$ gekennzeichneten Einträge sind nichtnegativ und reell. Berechnet man die Zeilensummen von $\hat{P}_{K_3}$, so ergeben die von Null verschiedenen Zeilenvektoren der Diagonalblöcke bereits den gesamten Beitrag zur Zeilensumme 1. Dies gilt auch für die von Null verschiedenen Zeilen der Nichtdiagonalblöcke.
Sei nun zu den oben beschriebenen Block-Jacobi-Verfahren ein beliebiger, zusammenhängender Prozessorgraph gegeben, so ist die Iterationsmatrix $\hat{H}$ gemäß Abschnitt (3.1) durch folgende Gleichung definiert:

$$\hat{H} = \sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l + \sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l H_l \, .$$

Die aus einem Eigenvektor der Matrix $\hat{H}_{K_m}$ konstruierte Matrix $\hat{U}$ läßt sich als Kroneckerprodukt darstellen

$$\hat{U} = \sum_{l=1}^{m} I_m^{(l,l)} \otimes D_l, \tag{4.29}$$

so daß das Matrixprodukt $\hat{P} = \frac{1}{\rho(\hat{H}_{K_m})}\hat{U}^{-1}\hat{H}\hat{U}$ in expliziter Form folgende Gestalt hat:

$$\begin{aligned}\hat{P} = &\frac{1}{\rho(\hat{H}_{K_m})}\hat{U}^{-1}\hat{H}\hat{U} = \\ &\frac{1}{\rho(\hat{H}_{K_m})}\left(\sum_{l=1}^{m}\sum_{\substack{j=1\\ j\neq l}}^{m}\sum_{k=1}^{m} I_m^{(l,l)}\mathcal{A}^T(\mathrm{ST}_k)I_m^{(j,j)} \otimes D_l^{-1}E_kD_j \right. \\ &\left. \qquad + \sum_{l=1}^{m} I_m^{(l,l)} \otimes D_l^{-1}E_lH_lD_l\right).\end{aligned} \tag{4.30}$$

Die Diagonalelemente der Matrix $\hat{P}$ im zweiten Term der Gleichung (4.30) unterscheiden sich nicht von denen eines vollständigen Prozessorgraphen. In beiden Fällen beschreiben sie den numerischen Anteil der Iteration, der lokal auf jedem Prozessor berechnet wird. Der erste Term der Gleichung (4.30) beschreibt die Nebendiagonalblöcke. Jedem Indexpaar (l, j) ist ein Block der Größe $n \times n$ zugewiesen. Da die Diagonalen der Adjazenzmatrizen $\mathcal{A}^T(\mathrm{ST}_k)$ identisch Null sind, reicht es, Indexpaare mit $j \neq l$ zu betrachten. Sei nun im folgenden ein Indexpaar (l, j) mit $l \neq j$ gegeben. Ist der Koeffizient $[\mathcal{A}^T(\mathrm{ST}_k)]_{lj}$ von Null verschieden, so entsteht im Block (l, j) der Matrix $\hat{P}$ ein Eintrag $D_l^{-1}E_kD_j$.
Aus der Definition von $\mathcal{A}^T(\mathrm{ST}_k)$ folgt, daß $k \neq l$ sein muß, da die Zeile k keine von Null verschiedenen Einträge hat. Es verbleiben also die zwei möglichen Fälle:

Fall 1: Sei $k = j$. Dann folgt aus obigen Betrachtungen für den vollständigen Prozessorgraphen

$$D_l^{-1}E_jD_j = \rho(\hat{H}_{K_m})E_j\,. \tag{4.31}$$

Fall 2: Sei $k \neq j$. Dann folgt bei Benutzung der Gleichung (4.28)

$$\begin{aligned} D_l^{-1}E_kD_j &= \frac{1}{\rho(\hat{H}_{K_m})}D_l^{-1}E_kD_k \\ &= \rho(\hat{H}_{K_m})\left(\frac{1}{\rho(\hat{H}_{K_m})}E_k\right).\end{aligned}$$

Die Matrix $\hat{P}$ ist also nicht mehr stochastisch, sondern hat entweder Zeilensummen vom Betrag 1 für die Diagonalblöcke und die Nebendiagonalblöcke aus Fall 1 oder Zeilensummen vom Betrag $\frac{1}{\rho(\hat{H})}$ für die Nebendiagonalblöcke aus Fall 2. Da $\rho(\hat{H}_{K_m}) < 1$, gilt $\frac{1}{\rho(\hat{H}_{K_m})} > 1$. Die Matrix $\hat{P}$ ist nichtnegativ. Also gilt nach dem verallgemeinerten Quotientensatz 1.7

$$1 \leq \rho(\hat{P}) \leq \frac{1}{\rho(\hat{H}_{K_m})}. \tag{4.32}$$

Ist $\hat{H}$ und somit auch $\hat{P}$ zusätzlich irreduzibel, so gilt nach Satz 1.5 sogar die strenge Ungleichung

$$1 < \rho(\hat{P}) < \frac{1}{\rho(\hat{H}_{K_m})}. \tag{4.33}$$

Da sich die Spektralradien der Matrizen $\hat{P}$ und $\hat{H}$ lediglich um den Faktor $\rho(\hat{H}_{K_m})$ unterscheiden, folgt aus (4.32)

$$\rho(\hat{H}_{K_m}) \leq \rho(\hat{H}_{K_m})\rho(\hat{P}) = \rho(\hat{H}) \leq 1 .$$

Da in Satz 3.1 bereits gezeigt wurde, daß die Spektralradien aller Matrizen $\hat{H}$ kleiner als 1 sind, folgt letztendlich

$$\rho(\hat{H}_{K_m}) \leq \rho(\hat{H}) < 1$$

und entsprechend in der verschärften Form für irreduzible $\hat{H}$

$$\rho(\hat{H}_{K_m}) < \rho(\hat{H}) < 1.$$

Somit ist der folgende Satz bewiesen.

Satz 4.3

> H sei die Iterationsmatrix eines Block-Jacobi-Verfahrens und irreduzibel. $\rho(\hat{H}_{K_m})$ bezeichne den Spektralradius der Iterationsmatrix $\hat{H}_{K_m}$ des asynchronen Block-Jacobi-Verfahrens auf einem Multiprozessorrechner mit vollständigem Prozessorgraphen. $\hat{H}$ sei die Iterationsmatrix eines asynchronen Block-Jacobi-Verfahrens auf einem Multiprozessorrechner mit beliebiger *broadcast* Strategie. Dann gilt

$$\begin{aligned} \rho(\hat{H}_{K_m}) &< \rho(\hat{H}) \text{ falls } \hat{H} \text{ irreduzibel und} \\ \rho(\hat{H}_{K_m}) &\leq \rho(\hat{H}) \text{ sonst.} \end{aligned}$$

Beliebige Prozessorgraphen G lassen sich über den Spektralradius der Iterationsmatrix $\hat{H}$ klassifizieren. Dazu soll zunächst der Begriff des Abstandes zweier Prozessoren in einem gerichteten Graphen definiert werden.
Wie bereits in Definition 3.2 erläutert, ist ein Spannbaum ST_l ein gerichteter Subgraph des Prozessorgraphen G, der ausgehend von der Wurzel l über $m-1$ Kanten alle Knoten des Graphen G erfaßt. Jedem der $m-1$ Zwischen-

und Blattknoten von ST_l läßt sich durch Abzählen der Kanten ein eindeutiger Abstand zur Wurzel l des Spannbaumes zuordnen. Allgemein bezeichne $\delta_l(k)$ den Abstand von der Wurzel l zum Knoten k, der als die Anzahl der gerichteten Kanten definiert sei, die den Weg von der Wurzel l zum Knoten k bilden. Da ST_l ein Baum ist, ist der Weg und somit auch $\delta_l(k)$ eindeutig. Eine geeignete *broadcast* Strategie auf dem Prozessorgraphen setzt sich aus m Spannbäumen zusammen. Das Maximum

$$d = \max_{i \in \{1,\ldots,m\}} \{\delta_i(k) \quad | \quad 1 \leq k \leq m\}$$

aller möglichen Weglängen in jedem dieser Spannbäume werde im folgenden als der Durchmesser der *broadcast* Strategie bezeichnet.
Unter diesen Voraussetzungen gilt der

Satz 4.4

> H sei die in Gleichung (4.4) definierte Iterationsmatrix des Block-Jacobi-Verfahrens und zusätzlich irreduzibel, $\rho(H)$ der Spektralradius von H zum Eigenvektor u. $\hat{H}$ bezeichne die Iterationsmatrix für das asynchrone Block-Jacobi-Verfahren auf einem Prozessorgraphen mit zugehöriger *broadcast* Strategie vom Durchmesser d.
>
> Dann ist
> $$\rho(\hat{H}) \leq \rho(H)^{\frac{1}{d+1}}. \tag{4.34}$$

Beweis:

Sei $\hat{u}$ der Vektor

$$\hat{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \tag{4.35}$$

mit den Komponenten

$$u_i = \sum_{l=1}^{m} \rho(H)^{\frac{d-\delta_l(i)}{d+1}} E_l u. \tag{4.36}$$

$\hat{u}$ ist positiv, da nach Satz 1.4 der Eigenvektor u zu $\rho(H)$ positiv und $\rho(H) > 0$ ist. Der Vektor $\hat{u}$ läßt sich alternativ als Kroneckerprodukt schreiben

$$\hat{u} = \sum_{i,j=1}^{m} \left(I_m^{(i,j)} \otimes \rho(H)^{\frac{d-\delta_j(i)}{d+1}} E_j \right) \bigoplus_{i=1}^{m} u. \tag{4.37}$$

Das Ziel dieses Beweises ist es, die Richtigkeit der Ungleichung

$$\hat{H}\,\hat{u} \leq \rho(H)^{\frac{1}{d+1}}\,\hat{u}$$

nachzuweisen. Dann folgt aus Satz 1.8 sofort die Behauptung.

Gemäß Gleichung (3.7) setzt sich $\hat{H}$ aus zwei Summanden zusammen. Entsprechend zerfällt das Produkt $\hat{H}\,\hat{u}$ in zwei Komponenten, die unabhängig voneinander betrachtet werden können. Sei also $\hat{H}\,\hat{u} = \hat{u}_1 + \hat{u}_2$ mit

$$\hat{u}_1 = \left(\sum_{l=1}^{m} \mathcal{A}^T(\mathrm{ST}_l) \otimes E_l \right) \hat{u} \tag{4.38}$$

und

$$\hat{u}_2 = \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l H_l \right) \hat{u}. \tag{4.39}$$

Zunächst betrachte man nun den Vektor $\hat{u}_1$, ersetze die Adjazenzmatrizen $\mathcal{A}^T(\mathrm{ST}_l)$ gemäß der Gleichung (3.3) und vertausche die Indizes der resultierenden Doppelsumme. Somit ist

$$\begin{aligned} \hat{u}_1 &= \left(\sum_{l=1}^{m} \sum_{\substack{k=1 \\ k \neq l}}^{m} I_m^{(k,\varphi_l(k))} \otimes E_l \right) \hat{u} && (4.40) \\ &= \left(\sum_{k=1}^{m} \sum_{\substack{l=1 \\ l \neq k}}^{m} I_m^{(k,\varphi_l(k))} \otimes E_l \right) \hat{u}\,. && (4.41) \end{aligned}$$

Das Kroneckerprodukt (4.37) ersetzt $\hat{u}$ in (4.41) und führt zu der Mehrfachsumme

$$\hat{u}_1 = \left(\sum_{k=1}^{m} \sum_{\substack{l=1 \\ l \neq k}}^{m} I_m^{(k,\varphi_l(k))} \otimes E_l \right) \left(\sum_{i,j=1}^{m} I_m^{(i,j)} \otimes \rho(H)^{\frac{d-\delta_j(i)}{d+1}} E_j \right) \bigoplus_{i=1}^{m} u\,. \tag{4.42}$$

Anwendung der Rechenregel $(A \otimes B)(C \otimes D) = AC \otimes BD$ und die Orthogonalitätsrelation (4.2) führen zu einer Vereinfachung der Gleichung (4.42), so daß nur noch über Indizes summiert wird, die die Nebenbedingungen $l = j$, $\varphi_l(k) = i$ und $l \neq k$ erfüllen. Es folgt also

$$\hat{u}_1 = \left(\sum_{k=1}^{m} \sum_{\substack{l=1 \\ l \neq k}}^{m} I_m^{(k,l)} \otimes \rho(H)^{\frac{d-\delta_l(\varphi_l(k))}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u$$

$$= \quad \rho(H)^{\frac{1}{d+1}} \left(\sum_{k=1}^{m} \sum_{\substack{l=1 \\ l \neq k}}^{m} I_m^{(k,l)} \otimes \rho(H)^{\frac{d-1-\delta_l(\varphi_l(k))}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u.$$

Da $\varphi_l(k)$ den Vorgänger des Knotens k im Spannbaum ST_l bezeichnet, gilt

$$\delta_l(\varphi_l(k)) + 1 = \delta_l(k)$$

und somit

$$\hat{u}_1 = \rho(H)^{\frac{1}{d+1}} \sum_{k=1}^{m} \left(\sum_{\substack{l=1 \\ l \neq k}}^{m} I_m^{(k,l)} \otimes \rho(H)^{\frac{d-\delta_l(k)}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u. \qquad (4.43)$$

Auf ähnliche Weise ersetzt man nun im Vektor $\hat{u}_2$ den Vektor $\hat{u}$ durch das Kroneckerprodukt (4.37) und vereinfacht $\hat{u}_2$ anschließend zu

$$\begin{aligned} &\hat{u}_2 \\ &\quad = \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l H_l \right) \left(\sum_{i,j=1}^{m} I_m^{(i,j)} \otimes \rho(H)^{\frac{d-\delta_j(i)}{d+1}} E_j \right) \bigoplus_{l=1}^{m} u \\ &\quad = \left(\sum_{l=1}^{m} \sum_{j=1}^{m} I_m^{(l,j)} \otimes E_l H_l \rho(H)^{\frac{d-\delta_j(l)}{d+1}} E_j \right) \bigoplus_{l=1}^{m} u. \end{aligned}$$

Da $\rho(H)^{\frac{d-\delta_j(l)}{d+1}}$ für alle Exponenten kleiner als 1 ist und alle $E_l H_l$ nichtnegativ sind, gilt die Abschätzung

$$\begin{aligned} \hat{u}_2 &\leq \left(\sum_{l,j=1}^{m} I_m^{(l,j)} \otimes E_l H_l E_j \right) \bigoplus_{l=1}^{m} u \\ &= \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes E_l H_l \right) \bigoplus_{l=1}^{m} u. \end{aligned}$$

Die Eigenwertgleichung (4.7) führt nun zu

$$\begin{aligned} \hat{u}_2 &\leq \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes \rho(H) E_l \right) \bigoplus_{l=1}^{m} u \\ &= \rho(H)^{\frac{1}{d+1}} \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes \rho(H)^{\frac{d}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u. \end{aligned}$$

Eine Addition der beiden Vektoren $\hat{u}_1$ und $\hat{u}_2$ ergibt

$$\begin{aligned}
\hat{H}\hat{u} &= \hat{u}_1 + \hat{u}_2 \\
&\leq \rho(H)^{\frac{1}{d+1}} \left(\sum_{\substack{k,l=1 \\ l \neq k}}^{m} I_m^{(k,l)} \otimes \rho(H)^{\frac{d-\delta_l(k)}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u \\
&\quad + \rho(H)^{\frac{1}{d+1}} \left(\sum_{l=1}^{m} I_m^{(l,l)} \otimes \rho(H)^{\frac{d}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u \\
&= \rho(H)^{\frac{1}{d+1}} \left(\sum_{k,l=1}^{m} I_m^{(k,l)} \otimes \rho(H)^{\frac{d-\delta_l(k)}{d+1}} E_l \right) \bigoplus_{l=1}^{m} u \\
&= \rho(H)^{\frac{1}{d+1}} \hat{u}
\end{aligned}$$

und somit ist gezeigt, daß

$$\hat{H}\hat{u} \leq \rho(H)^{\frac{1}{d+1}} \hat{u}$$

gilt und die Behauptung sofort folgt.

$\boxed{\text{q.e.d.}}$

Die im letzten Satz bewiesenen Schranken für den Spektralradius lassen sich unter den gegebenen Voraussetzungen nicht verbessern, denn für einen vollständigen Prozessorgraphen ist der Durchmesser der optimalen *broadcast* Strategie gerade 1. In diesem Fall reduziert sich die Abschätzung (4.34) auf

$$\rho(\hat{H}) \leq \sqrt{\rho(H)}.$$

Hier wurde aber bereits in Satz 4.1 die Gleichheit gezeigt. Analog dazu kann man den im Beweis zu Satz 4.4 definierten positiven Vektor (4.35) als Verallgemeinerung des Eigenvektors (4.8) zum Spektralradius der Matrix $\hat{H}_{K_m}$ betrachten, der sich für $d = 1$ auf den Vektor (4.8) reduziert.

Kapitel 5

Numerische Ergebnisse

5.1 Leistungsmessung auf MP-Rechnern

Die Qualität eines Algorithmus zur Lösung einer gegebenen berechenbaren Aufgabe wird zumeist durch die Kriterien Laufzeit und Platzbedarf charakterisiert. Im Vergleich mit direkten Verfahren zur Lösung von Gleichungssystemen ist der Platzbedarf iterativer Verfahren gering und ein Hauptargument für ihre Anwendung. Daher beschränken sich die anschließenden Ausführungen lediglich auf die Messung von Laufzeiten von Algorithmen. Die Darstellung dieser Theorie folgt den Ausführungen von [AG89].
Die betrachteten Aufgaben hängen von einem ganzzahligen Parameter N ab, der auch als die Größe einer Aufgabe bezeichnet wird. Ein Algorithmus zur Lösung einer Aufgabe der Größe N benötigt auf einem sequentiell arbeitenden Rechner mit einem Prozessor die Ausführungszeit $T_1(N)$. Ist derselbe Algorithmus parallelisierbar und für die Ausführung auf einem Multiprozessorrechner mit P Prozessoren geeignet, so ist die gemessene Ausführungszeit zusätzlich von einem zweiten ganzzahligen Parameter P abhängig, der die Größe des Multiprozessorrechners kennzeichnet. Die Parameter P und N sind im allgemeinen voneinander unabhängig, und die Ausführungszeit $T(P,N)$ ist auf Multiprozessorrechnern eine reellwertige Funktion in zwei ganzzahligen Variablen. Wird der Parameter P als konstante Größe betrachtet, so bezeichnet man die Ausführungszeit mit $T_P(N)$. Das Verhältnis der Aufgabengröße N zur Rechnergröße P heißt Ladungsfaktor

$$L = \left\lceil \frac{N}{P} \right\rceil .$$

Die Komplexitätstheorie beschreibt die Laufzeiten sequentieller Algorithmen durch eine Klassifizierung ihres asymptotischen Verhaltens in Abhängigkeit von der Größe der zu lösenden Aufgabe. Ist g eine positive, reellwertige Funktion einer ganzzahligen Variablen, so bezeichnet man mit $\Theta(g)$ die Menge aller positiven, reellwertigen Funktionen f, für die der Quotient f/g von oben und gegen Null beschränkt ist. f ist also Element von $\Theta(g)$, falls zwei positive Konstanten c und C existieren, so daß

$$0 < c\, g(n) < f(n) < C\, g(n)$$

für alle positiven n. An Stelle von $f \in \Theta(g)$ ist in der Komplexitätstheorie die Schreibweise $f = \Theta(g)$ üblich, die im folgenden auch benutzt wird. Für die Betrachtung paralleler Algorithmen auf Multiprozessorrechnern ist eine Erweiterung dieser Definition erforderlich, da hier die Ausführungszeit von zwei ganzzahligen Parametern abhängt. Im Fall der asynchronen Iteration ist die spezielle Situation eines "Rechners unter schwerer Last" von Bedeutung. Hier werden lediglich Zahlenpaare (P, N) betrachtet, für die der Ladungsfaktor groß ist. Diese Perspektive bedeutet keine Einschränkung, da in praktischen Anwendungen die Größe einer gestellten Aufgabe die Anzahl der vorhandenen Prozessoren zumeist um ein Vielfaches übertrifft.
In Analogie zu sequentiellen Algorithmen definiert man einen Komplexitätsbegriff für "Rechner unter schwerer Last". Sind f und g positive, reellwertige Funktionen, die jeweils von zwei positiven, ganzzahligen Parametern P und N abhängen, ist $N_0(P)$ eine ganzzahlige Funktion in P, und sind weiterhin c, C und P_0 Konstanten, so ist f ein Element der Funktionenklasse $\Theta_{HL}(g)$, falls

$$0 < c\,g(P, N) < f(P, N) < C\,g(P, N)$$

für alle $P \geq P_0$ und alle $N \geq N_0(P)$.
Zur Veranschaulichung der vorgestellten Komplexitätsbegriffe diene das folgende Beispiel der Addition von N Zahlen [AG89]. Ein sequentieller Rechner führt eine Addition nach der anderen aus, d.h. die benötigte Zeit ist proportional zur Zahl der Summanden. Da jede Addition eine konstante, vom Rechner abhängige Zeit zur Ausführung benötigt, gilt für die Zeitkomplexität des seriellen Additionsalgorithmus

$$T_1(N) = \Theta(N)\,.$$

Betrachtet man nun einen Multiprozessorrechner mit $P = N$ Prozessoren, dessen Prozessorgraph einem Binärbaum entspricht, so kann man jedem Pro-

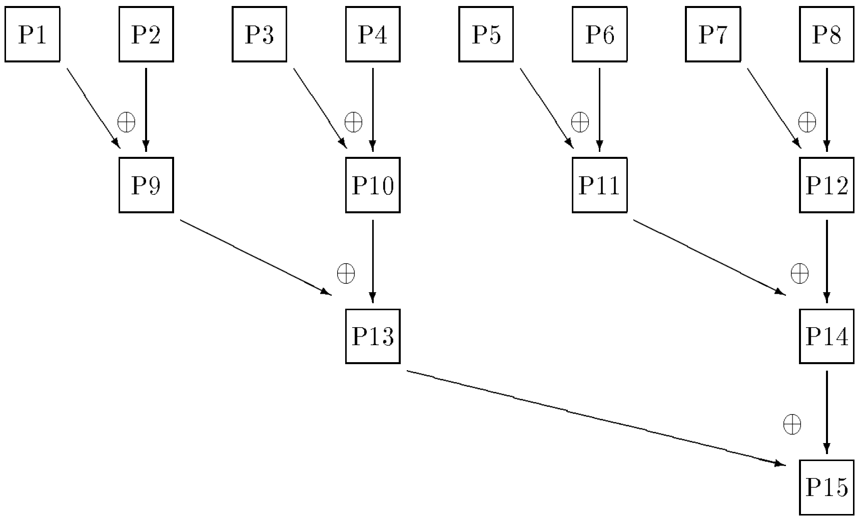

Abbildung 5.1: Kaskadierte Addition

zessor eine Zahl mitteilen und die Summe nach dem Prinzip der kaskadierten Addition berechnen. Die Blätter des Binärbaumes verschicken ihre Daten an ihre Eltern. Hier werden jeweils drei Zahlen addiert, und die Summe wird wiederum an die Eltern auf der nächsthöheren Ebene verschickt, so daß letztendlich die Wurzel des Baumes die Endsumme über alle N Zahlen enthält. Jeder Prozessor hat maximal zwei Additionen durchzuführen, die er in konstanter Zeit berechnet. Die Höhe des Binärbaumes $H = \log P$ bestimmt die Dauer der kaskadierten Addition, so daß

$$T(P, N) = \Theta(\log P) = \Theta(\log N).$$

Wählt man nun $N > P$ und übermittelt jedem Prozessor $L = \lceil \frac{N}{P} \rceil$ Zahlen, so erfolgt zunächst eine sequentielle Addition der L Summanden auf jedem Prozessor und erst danach die kaskadierte Addition. Die sequentielle Addition und die kaskadierte Addition laufen unabhängig voneinander ab, so daß die Komplexitäten der Einzeloperationen zur Gesamtkomplexität

$$T(P, N) = \Theta(L) + \Theta(\log P) = \Theta(L + \log P)$$

addiert werden können. Vergrößert sich der Ladungsfaktor L bei konstantem P, so dominiert $\Theta(L)$ asymptotisch den Term $\Theta(\log P)$, und die Komplexität "unter schwerer Last" geht im Grenzfall in

$$T(P, N) = \Theta_{HL}(L)$$

über. Der Übergang zu diesem asymptotischen Verhalten wird im Beispiel der Addition von N Zahlen für $L = \log P$ erreicht. Das für die kaskadierte Addition im Hinblick auf eine effiziente Parallelisierung interessante Gebiet ist die Menge aller Punkte (P, N) mit $P \geq P_0$ und $N \geq N_0(P) = P \log P$.
Die reellwertigen Funktionen $T(P, N)$ und die Möglichkeit ihrer Klassifizierung liefern die Grundlage für den Vergleich der Ausführungszeiten eines Algorithmus auf Multiprozessorrechnern mit unterschiedlicher Prozessorzahl. Ein klassisches Vergleichskriterium ist der sogenannte *speedup*

$$S_P(N) = \frac{T_1(N)}{T_P(N)}.$$

Die Ausführungszeit $T_1(N)$ eines Algorithmus auf einem sequentiellen Rechner wird als Maßstab benutzt, um sie mit der Ausführungszeit $T_P(N)$ desselben Algorithmus auf einem Multiprozessorrechner mit P Prozessoren zu vergleichen. In beiden Fällen werden Ausführungszeiten für eine Aufgabe der konstanten Größe N verglichen, so daß das Ziel der Untersuchung die Abschätzung eines möglichen Zeitgewinns durch einen Mehreinsatz an Prozessoren ist. Die Anzahl der eingesetzten Prozessoren ist eine obere Schranke für den *speedup*, die jedoch nur für ein vollständig parallelisierbares Programm erreicht wird.
Für das Beispiel der kaskadierten Addition gilt im Fall $N = P$

$$S_P(N) = \frac{T_1(N)}{T_P(N)} = \frac{\Theta(N)}{\Theta(\log N)} = \Theta\left(\frac{N}{\log N}\right).$$

Ein vermehrter Einsatz von Prozessoren bringt keinerlei Vorteil mit sich. Für wachsende N und P entfernt sich der erreichbare *speedup* beständig vom maximal erreichbaren Wert P. Vergleicht man dies mit dem *speedup* des Multiprozessorrechners unter "schwerer Last", so gilt

$$S_P(N) = \frac{T_1(N)}{T_P(N)} = \frac{\Theta(N)}{\Theta_{HL}(L)} = \Theta_{HL}(P)\,.$$

Das Problem ist asymptotisch betrachtet vollständig parallelisierbar, da der Kommunikationsaufwand der kaskadierten Addition vernachläßigbar klein ist im Vergleich zum Ladungsfaktor. Der oben beschriebene parallele Additionsalgorithmus wird aber erst dann eine zufriedenstellende Auslastung des Multiprozessorsystems gewährleisten, wenn die Anzahl der Summanden N größer als $N_0(P) = P \log P$ gewählt wird.
Bei der Definition des *speedup* werden die Größe der Aufgabe und die Anzahl der Prozessoren als voneinander unabhängig vorausgesetzt. Diese Voraussetzung und das Ziel des Zeitgewinns durch den vermehrten Einsatz von Prozessoren können zu einer sehr pessimistischen Einschätzung der Leistungsfähigkeit von Multiprozessorrechnern führen. Ist ein Algorithmus nicht vollständig parallelisierbar, so muß ein bestimmter Anteil von s Prozent sequentiell ausgeführt werden. Dieser Anteil soll unabhängig von der Problemgröße konstant sein. Der Einfachheit halber nehme man an, daß der parallelisierbare Anteil des Algorithmus gleichmäßig auf P Prozessoren verteilt werden kann. Ist die Ausführungszeit $T_1(N)$ des Algorithmus auf einem sequentiellen Rechner bekannt, so gilt auf einem Multiprozessorrechner mit P Prozessoren:

$$T_p(N) = s\,T_1(N) + (1-s)\frac{T_1(N)}{P}\,. \tag{5.1}$$

Für den *speedup* gilt also

$$S_P(N) = \frac{T_1(N)}{sT_1(N) + (1-s)\frac{T_1(N)}{P}} = \frac{1}{s + \frac{1-s}{P}}\,.$$

Betrachtet man den *speedup* als Funktion des sequentiellen Anteils s des Algorithmus und läßt die Anzahl der Prozessoren wachsen, so gilt im asymptotischen Fall

$$\lim_{P\to\infty} S_P(s,N) = \lim_{P\to\infty} \frac{P}{1 + s(P-1)} = \frac{1}{s}\,. \tag{5.2}$$

Dies bedeutet, daß der *speedup* eines Multiprozessorsystems eine beschränkte Funktion ist. Eine erfolgversprechende Strategie bei der Parallelisierung einer Aufgabe unter oben genannten Voraussetzungen liegt also nicht in der Steigerung der Prozessorzahlen, sondern in der Minimierung des sequentiellen Anteils des Algorithmus begründet.
Eine alternative Interpretation der Meßdaten für die Laufzeit eines Programmes führt zum Begriff des *skalierten speedup*. Die Grundlage dieser Analyse bildet die Annahme, daß ein gegebener Algorithmus über eine skalierbare Komponente verfügt, mit deren Vergrößerung die sequentielle Komponente

des Algorithmus relativ zur Gesamtaufgabe kleiner wird. Als Vergleichskriterium wählt man nun die Ausführungszeit dieses Algorithmus auf einem Multiprozessorsystem und fragt nach der Zeit, die ein sequentieller Rechner zur Lösung des Problems benötigt hätte. Ist $T_P(N)$ die bekannte Ausführungszeit des Algorithmus auf dem Multiprozessorrechner und beschreibt $s_P(N)$ den sequentiellen Anteil der Ausführungszeit, so gilt für die Ausführungszeit des Algorithmus auf einem sequentiellen Rechner

$$T_1(N) = s_P(N)T_P(N) + P(1 - s_P(N))T_P(N). \tag{5.3}$$

Analog zur Definition des *speedup* gilt für den *skalierten speedup*

$$SC_P(N) = \frac{T_1(N)}{T_P(N)} = \frac{s_P(N)T_P(N) + P(1 - s_P(N))T_P(N)}{T_P(N)}. \tag{5.4}$$

Die sequentielle Komponente des Algorithmus geht unter obigen Annahmen linear in den skalierten *speedup* ein, d.h. bei skalierbaren Algorithmen kann eine weitaus höhere Ausnutzung der Prozessoren erwartet werden.
Eine weitere interessante Frage bei skalierbaren Algorithmen ist, um welchen Faktor ein Problem auf einem Multiprozessorrechner größer sein darf, das in der gleichen Rechenzeit wie ein kleineres Problem auf einem sequentiellen Einprozessorrechner gelöst werden soll. Dieser Faktor wird als *scaleup* bezeichnet und ist definiert als

$$\mathrm{SCA}_P = \frac{N}{M},$$

falls $T_1(M) = T_P(N)$.

5.2 Spektralradius und Konvergenzgeschwindigkeit

Die enge Beziehung zwischen Spektralradius der Iterationsmatrix und der Konvergenz des entsprechenden Iterationsverfahrens ergibt sich bereits aus Satz 1.2. Wie die folgenden Überlegungen zeigen, ist der Spektralradius aber auch ein gutes Maß zur Beurteilung des asymptotischen Konvergenzverhaltens des Verfahrens. Entsprechend den Ausführungen von Hackbusch [Hac93] werde der Spektralradius der Iterationsmatrix als die Konvergenzrate des zugehörigen Verfahrens bezeichnet.
Die Güte eines iterativen Verfahrens zeigt sich in der Geschwindigkeit, mit der die Iterierten x_k gegen die gesuchte Lösung ξ konvergieren. Betrachtet man den nach k Iterationen auftretenden Fehler

$$e_k = x_k - \xi,$$

so läßt sich dieser auf den Ausgangsfehler $e_0 = x_0 - \xi$ zurückführen:

$$e_k = H^k e_0.$$

Bezüglich einer geeigneten Vektornorm $\| \bullet \|$ und der zugeordneten Matrixnorm berechnet sich die Reduktion des Fehlers relativ zum Ausgangsfehler nach k Iterationen als

$$\frac{\|e_k\|}{\|e_0\|} \leq \|H^k\|\,. \tag{5.5}$$

Da es zu jedem $\epsilon > 0$ eine ganze Zahl $k_0 \in \mathbb{N}$ gibt, so daß

$$\rho(H) \leq \|H^k\|^{\frac{1}{k}} \leq \rho(H) + \epsilon$$

für alle $k \geq k_0$, ergibt sich für den relativen Fehler (5.5) die Abschätzung

$$\frac{\|e_k\|}{\|e_0\|} \leq (\rho(H) + \epsilon)^k, \qquad k \geq k_0.$$

Diese Ungleichung führt zur Definition eines gebräuchlichen Geschwindigkeitsmaßes für das asymptotische Konvergenzverhalten eines Iterationsverfahrens. Betrachtet man den verallgemeinerten Reduktionsfaktor

$$\rho_{m+k,m}(x_0) = \left(\frac{\|e_{m+k}\|}{\|e_m\|} \right)^{\frac{1}{k}}$$

in Abhängigkeit vom gewählten Startwert x_0, so gilt die

Bemerkung 5.1

Für alle $m \in \mathbb{N}$ gilt

$$\lim_{k \to \infty} \max\{\rho_{m+k,m}(x_0) : x_0 \in \mathbb{R}^n\} = \rho(H)\,. \tag{5.6}$$

Beweis: siehe [Hac93].

Dies bedeutet, daß selbst für den ungeeignetesten Startwert und genügend große m der Reduktionsfaktor beliebig nahe am Spektralradius der Iterationsmatrix liegt. Somit kann im asymptotischen Fall die Reduktion des Fehlers nach k Iterationen durch $\rho(H)^k$ beschrieben werden. Legt man als Vergleichskriterium für die Güte des Verfahrens die Reduktion des Fehlers um den Faktor $\frac{1}{e}$ fest, so ergibt sich die Iterationszahl $\mathrm{It}(H)$ der hierfür benötigten Iterationen in Abhängigkeit vom Iterationsverfahren als

$$\mathrm{It}(H) = \left\lceil \frac{-1}{\ln(\rho(H))} \right\rceil .$$

Multipliziert man nun die Anzahl der Iterationsschritte mit dem Rechenaufwand des Verfahrens für einen Iterationsschritt, so erhält man die Zeitkomplexität des iterativen Verfahrens für eine Reduktion des Fehlers um den Faktor $\frac{1}{e}$.

5.3 *Broadcasts* auf s-regulären Graphen

Die gesamte Kommunikation zwischen den einzelnen Prozessoren wird im betrachteten Algorithmus der asynchronen Iteration über *broadcasts* abgewickelt. Daher ist es sinnvoll, ihr Laufzeitverhalten auf unterschiedlichen Prozessorgraphen zu untersuchen.

Der Prozessorgraph $G = (V(G), E(G))$ eines Multiprozessorrechners aus P Prozessoren heißt s-regulär, wenn zu jedem Prozessor genau s inzidente Kanten existieren. Bezeichnet $\gamma_G(p)$ den Kantengrad eines Prozessors $p \in V(G)$ und $\kappa(G)$ die Kardinalität der Menge $E(G)$, so berechnet sich die Anzahl der Kanten in G nach der Gleichung

$$\sum_{p \in V(G)} \gamma_G(p) = 2\kappa(G)\,. \tag{5.7}$$

Da G nach Voraussetzung s-regulär ist, gilt $\gamma_G(p) = s$ für alle Knoten aus $V(G)$ und somit

$$\kappa(G) = \frac{1}{2} s P\,. \tag{5.8}$$

Werden nun in allen P Knoten synchron *broadcasts* gestartet, so müssen bis zum Abschluß aller *broadcasts* insgesamt $P(P-1)$ Nachrichten verschickt werden. Dies entspricht einer durchschnittlichen Belastung jeder Kante mit

$$k = \frac{2}{s}(P-1) \tag{5.9}$$

Nachrichten. Dieser theoretische Wert weist auf ein lineares Wachstum der Kommunikationszeit bei steigender Anzahl von Prozessoren hin, falls die *broadcasts link bound* sind (siehe Definition 2.5).

Die Abbildung 5.2 bestätigt diese theoretische Vorhersage über das Wachstum der *broadcast* Zeit für Transputernetzwerke mit bis zu vierzig Prozessoren. Die Messungen wurden für wachsende Transputerzahlen P auf 2-regulären und 3-regulären Graphen durchgeführt. In einem 3-regulären Graphen sinkt die benötigte Kommunikationszeit entsprechend der Gleichung (5.9) um den Faktor $\frac{2}{3}$ gegenüber dem 2-regulären Graphen. Die markierten Punkte sind Mittelwerte, die nach der Auswertung von jeweils fünfzig Messungen errechnet wurden. Die Geraden sind lineare Regressionsgraphen der Meßpunkte.

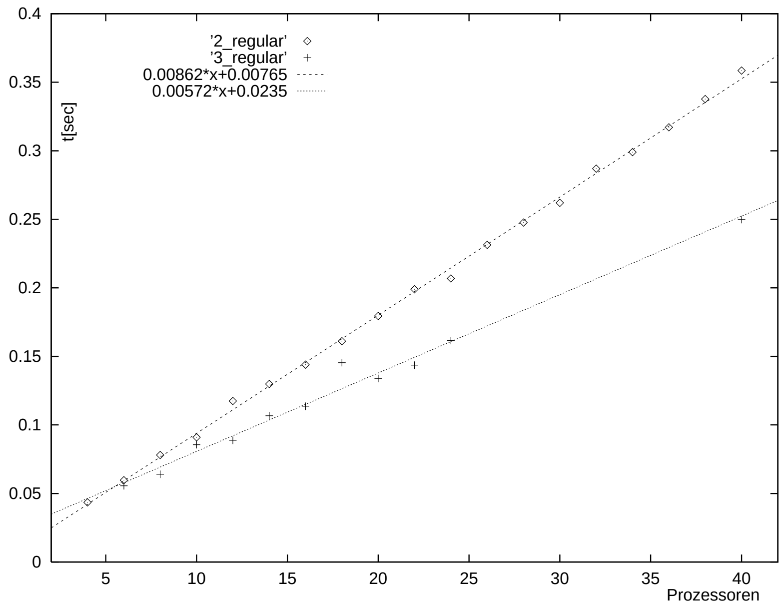

Abbildung 5.2: *Broadcast* auf s-regulären Graphen

Meßfehler bei der Zeitmessung treten aus drei Gründen auf:

- Ein Transputersystem erlaubt keine globale Zeitmessung. Jeder Transputer ist mit einem *timer* ausgerüstet, der nach dem Starten des Transputers aktiviert wird. Die Synchronisation der Transputer erfolgt über ein Programm, so daß geringfügige Zeitverschiebungen die Folge sind.
- Die Auslastung der Kanten in einem 3regulären Graphen entspricht nicht immer dem Idealfall der in Gleichung (5.9) errechneten durchschnittlichen Belastung.
- Das *routing* der Daten in einem Transputer wird über einen *reader-writer* Algorithmus gesteuert, so daß auch hier durch den sequentiellen Ablauf der Monitorzugriffe Verzögerungen auftreten.

Eine Tabelle der gemittelten Meßwerte mit den maximalen Abweichungen vom Mittelwert befindet sich in Anhang A.

5.4 Analyse eines Iterationsschrittes

Der Rechenaufwand für einen Iterationsschritt eines Block-Jacobi-Verfahrens mit nichtüberlappenden Blöcken hängt von zwei Parametern ab. N bezeichne die Größe der gegebenen Iterationsmatrix, P die Anzahl der Block-Jacobi-Zerlegungen, die im folgenden gleich der Anzahl der Prozessoren des Multiprozessorrechners sein soll. Je Iterationsschritt eines Block-Jacobi-Verfahrens wird ein Gleichungssystem

$$M_l x_{k+1} = N_l x_k + b_l \tag{5.10}$$

für jede Zerlegung $1 \leq l \leq m$ gelöst. Kalkuliert man den Aufwand für diese Operation nach Anwendung der Gewichtsmatrizen E_l, so erfordert die

Berechnung der rechten Seite von Gleichung (5.10) $L = \lceil \frac{N}{P} \rceil$ Multiplikationen zweier Vektoren der Länge N sowie eine Addition zweier Vektoren der Länge L. Anschließend ist ein Gleichungssystem der Größe L zu lösen. Liegt M_l bereits als LR-Zerlegung vor, so benötigt man für die Bewältigung dieser Aufgabe L Divisionen, $L(L-1)$ Multiplikationen und $L(L-1)$ Additionen. Insgesamt beträgt der Rechenaufwand je Iterationsschritt

- L Divisionen,
- $L(N+L-1)$ Multiplikationen,
- $L(N+L-1)$ Additionen.

Unter Vernachlässigung der Divisionen werden für jeden Iterationsschritt also

$$T^{(I)}(P,N) = t_1\,(L^2 - L + LN) \tag{5.11}$$

Zeiteinheiten benötigt. t_1 ist die konstante Zeit, die ein Prozessor zur Durchführung einer Multiplikation und einer Addition benötigt.

Die Iterationsmatrizen werden hier bewußt als vollbesetzte Matrizen vorausgesetzt, da bei den numerischen Experimenten der Einfluß des Prozessorgraphen auf die Iterationszeit im Vordergrund steht. Nur bei vollbesetzten Iterationsmatrizen ist gewährleistet, daß jeder Prozessor die Iterierten aller übrigen Prozessoren benötigt. Strukturierte Matrizen mit großen Nullblöcken benötigen speziell auf die Matrix abgestimmte Prozessorgraphen, wie noch aus den anschließend vorgestellten numerischen Experimenten ersichtlich wird.

Gleichzeitig zur Berechnung der Iterierten findet ein Informationsaustausch zwischen den Prozessoren per *broadcast* statt. Wie bereits in Abschnitt 5.3 gezeigt, steigt der Kommunikationsaufwand mit der Anzahl der Prozessoren. Besteht das gegebene Prozessornetz aus P Prozessoren, so verlangt jeder *broadcast* die Versendung von $P-1$ Nachrichten. Legt man das in Abschnitt 3.1 entwickelte Modell für die asynchrone Iteration zugrunde, so startet jeder Prozessor einen *broadcast* je Iterationsschritt. Hat der betrachtete Prozessorgraph den Durchmesser d, so befinden sich nach einer Einschwingdauer von $d-1$ Iterationsschritten gleichzeitig dP *broadcasts* in Ausführung. Reduziert man die Untersuchung wie in Abschnitt 5.3 auf s-reguläre Graphen, so ergibt dies je Iterationsschritt eine durchschnittliche Auslastung jeder Kante des Prozessorgraphen mit $\frac{2}{s}\,(P-1)$ Nachrichten. Jede Nachricht setzt sich aus der Initialisierungszeit und der Übertragungszeit zusammen. Die Initialisierungszeit ist die Zeitkomponente, die zum Aufbau einer Verbindung zwischen zwei Prozessoren benötigt wird. Ist t_2 die Zeit, die zur Versendung einer Gleitkommazahl mit 64 Bit Länge benötigt wird, und t_3 die konstante Initialisierungszeit, so ergibt sich je Iterationsschritt ein Kommunikationsaufwand von

$$T^{(K)}(P,N) = \frac{2}{s}(P-1)\,(t_2 L + t_3) \tag{5.12}$$

Zeiteinheiten.

Der sequentielle Anteil jedes Iterationsschrittes ergibt sich im wesentlichen aus den konkurrierenden Schreibzugriffen auf den in jedem Prozessor gespeicherten vorläufigen Lösungsvektor der Iteration. Zusätzlich zu den $\frac{1}{s}(P-1)$ Schreibzugriffen je Kante des Prozessorknotens beschreibt der numerische Iterationsprozeß je Iterationsschritt einmal diesen Vektor, so daß insgesamt

$$T^{(S)}(P,N) = t_4 N \tag{5.13}$$

Zeiteinheiten benötigt werden. t_4 ist die konstante Zeit, die der Prozessor zum Speichern einer Gleitkommazahl doppelter Genauigkeit benötigt.
Eine Zusammenfassung der einzelnen Laufzeitabschätzungen ergibt

$$T(P,N) = T^{(S)}(P,N) + \max\left(T^{(I)}(P,N), T^{(K)}(P,N)\right) \tag{5.14}$$

als Laufzeit eines Iterationsschrittes auf einem Multiprozessorrechner mit $P \geq 2$. Dieses Modell berücksichtigt keine Verzögerungen, die während eines Programmlaufes durch Funktionsaufrufe, Prozeßsynchronisation oder Kommunikationsengpässe auftreten.
Der zweite Summand der Gleichung (5.14) gibt einen Hinweis auf eine sinnvolle Wahl der Problemgröße in Abhängigkeit von der Zahl der zur Verfügung stehenden Prozessoren. Da die numerische Iteration die einzige produktive Operation der asynchronen Iteration ist, sind lediglich die Kombinationen aus der Größe der Aufgabe N und der Prozessoranzahl P sinnvoll, für die

$$T^{(I)}(P,N) \geq T^{(K)}(P,N) \tag{5.15}$$

ist. Das Ziel der folgenden Rechnung ist es, die Variable N zu isolieren und als Funktion der Prozessorzahl P zu schreiben. Auf diese Weise erhält man einen Wertebereich für die Problemgröße, in dem die Gesetze der Komplexität eines Systems "unter schwerer Last" gelten. Aus (5.11) und (5.12) folgt also

$$\begin{aligned} \frac{N^2}{P^2} - \frac{N}{P} + \frac{N^2}{P} \quad &\geq \quad \frac{2\,t_2}{s\,t_1}(P-1)\frac{N}{P} + \frac{2\,t_3}{s\,t_1}(P-1) && (5.16) \\ &= \quad \gamma_1 (P-1)\frac{N}{P} + \gamma_2 (P-1) && (5.17) \end{aligned}$$

mit den abkürzenden Bezeichnungen

$$\gamma_1 = \frac{2\,t_2}{s\,t_1} \qquad \gamma_2 = \frac{2\,t_3}{s\,t_1}\,. \tag{5.18}$$

Umordnen der Summanden in (5.17) führt zu einer quadratischen Ungleichung in der Unbekannten N

$$N^2 - \frac{P}{P+1}(1 + \gamma_1 (P-1))\,N - \gamma_2 \frac{P-1}{P+1} P^2 \geq 0. \tag{5.19}$$

Der negative Bereich der Lösungsmenge muß nicht weiter betrachtet werden, und die gesuchte Lösung wird schließlich durch die Ungleichung

$$N(P) \geq \frac{P}{2(P+1)}\left(1 + \gamma_1(P-1) + \sqrt{(1+\gamma_1(P-1))^2 + 4\gamma_2(P^2-1)}\right) \tag{5.20}$$

beschrieben. Bemerkenswert an dieser Ungleichung ist, daß alle maschinenabhängigen Parameter in den Konstanten γ_1 und γ_2 zusammengefaßt sind. Während die Rechenleistung des Prozessors und die Regularität des Prozessorgraphen kostendämpfend in die Konstanten γ_1 und γ_2 eingehen, so werden sie durch die Kommunikationsleistung der *broadcast* Strategie mit doppelter Gewichtung vergrößert.

Betrachtet man speziell ein Transputernetzwerk aus 2-regulären und 3-regulären Prozessorgraphen, so nehmen die Konstanten die in der nebenstehenden Tabelle gegebenen Werte an. Die Größe *clocktick* bezeichnet einen Arbeitstakt der CPU. Die Zeitwerte t_1 und t_4 für die Addition, Multiplikation und Speicherung von Fließkommazahlen doppelter Genauigkeit sind aus Beispielprogrammen berechnet, die in [Ebe93] als Assemblerroutinen vorliegen. Die Konstanten t_2 und t_3 beschreiben die in Megabit pro Sekunde gemessene Kommunikationsleistung des Transputers. Die Umrechnung in die Einheit *clocktick* erfolgte für eine CPU mit der Taktfrequenz von 25 MHz. Als Initialisierungszeit wurde eine Zeit von 50 μsec zugrunde gelegt [BRW93]. In die Berechnung der Übertragungszeit t_2 geht der Protokollaufwand von drei Bit Kontrollzeichen für eine Datenübertragung von acht Bit Daten ein.

Konstante	Ausführungszeit [clockticks]
t_1	63
t_2	110
t_3	1250
t_4	21

Tabelle 5.1: Transputerabhängige Konstanten

Die anschließenden Abbildungen 5.3 und 5.4 stellen die Iterationszeit und die Kommunikationszeit eines Iterationsschrittes als Funktion der Parameter N und P dar. Die Zeitachse ist mit der Quadratwurzelfunktion skaliert. Deutlich erkennt man, daß die Ausführungszeit der numerischen Iteration im Bereich einer Problemgröße bis $N = 300$ nur für relativ kleine Prozessorzahlen größer als die Kommunikationszeit ist. Die Anzahl der sinnvoll ein-

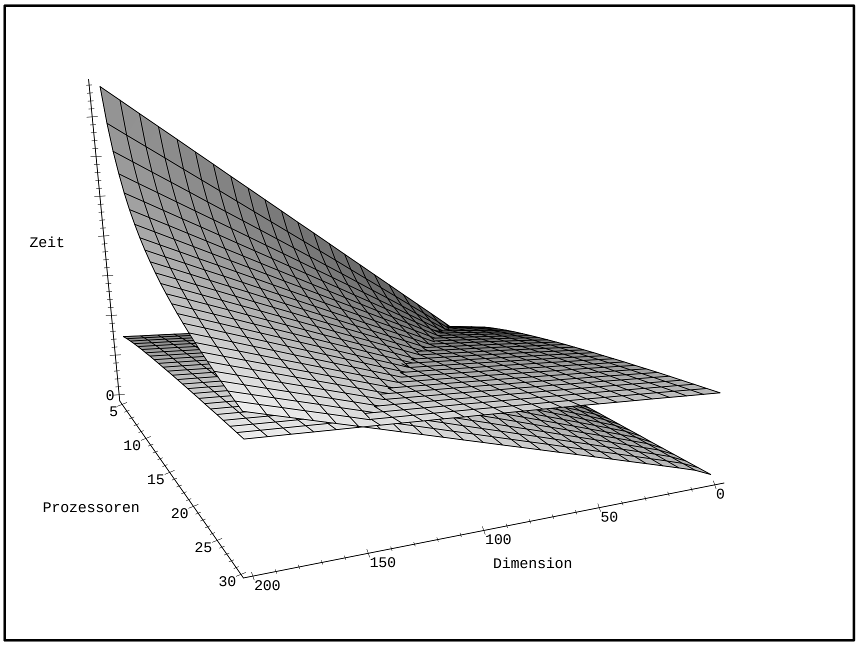

Abbildung 5.3: Iterationsschritt auf 2-regulären Graphen

setzbaren Prozessoren wächst aber mit der Problemgröße, da die rechte Seite der Ungleichung (5.20) für $P \geq 1$ streng monoton wächst. Es wird deutlich, daß die iterative Lösung eines linearen Gleichungssystems eine skalierbare Aufgabe ist, für die ein Mehreinsatz von Prozessoren erst mit wachsenden Problemgrößen Vorteile bringt.

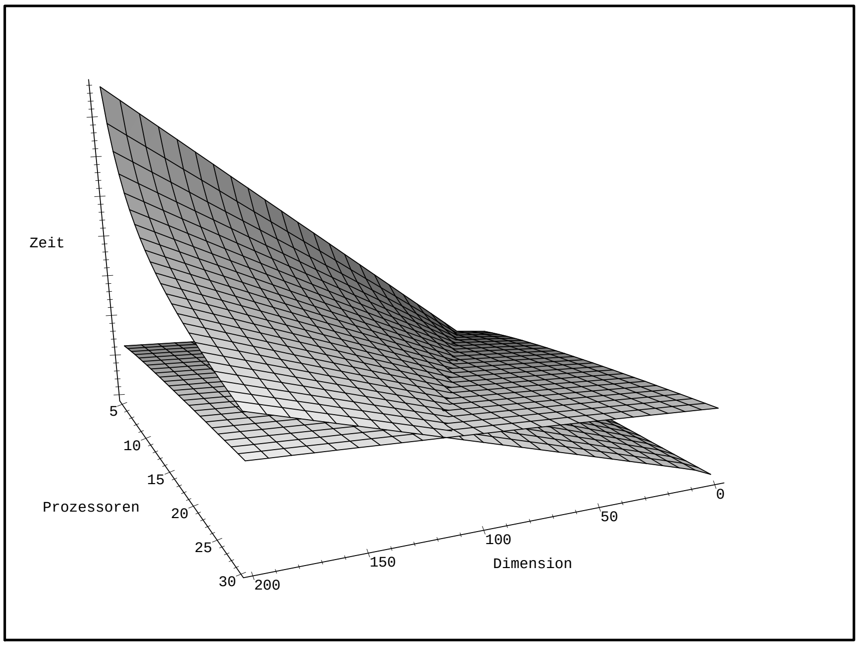

Abbildung 5.4: Iterationsschritt auf 3-regulären Graphen

Die Abbildungen 5.3 und 5.4 zeigen, daß 3-reguläre Graphen den Einsatz größerer Prozessorzahlen erlauben. Die benötigte Kommunikationszeit wird geringer, so daß die Schnittkante der Zeitebenen für Iteration und Kommunikation in Richtung größerer Prozessorzahlen verschoben ist.

5.5 Numerische Experimente

5.5.1 Testmatrizen

Numerische Experimente wurden für drei verschiedene Klassen von Testmatrizen durchgeführt. Ein Zufallsgenerator erzeugte vollbesetzte Matrizen der Dimensionen $N = 40, 60, 80, 100, 120$, deren Nichtdiagonalelemente aus dem Intervall $[0, 1]$ gewählt wurden. Die Diagonalelemente wurden nachträglich berechnet, so daß die Matrix strikt diagonaldominant war, der Spektralradius jedoch sehr nahe bei 1 lag. Die Matrizen dieses Typs gewährleisten, daß jeder Prozessor die Information aller übrigen Prozessoren zur Berechnung seiner Iterierten benötigt. Somit gehen die Weglängen aller Kommunikationswege in die Laufzeit einer Iteration ein.
Die zweite Serie numerischer Tests erfolgte für die Matrizen $A_1 \in M_{80}(\mathbb{R})$

$$A_1 = \begin{bmatrix} I_8 & & & -U & & & -U & & -U & \\ -U & I_8 & & -U & & & & & & -U \\ & & I_8 & & -U & & -U & & & \\ -U & & -U & I_8 & & & & -U & & \\ & & & -U & I_8 & & & & & -U \\ & & -U & & & I_8 & & & -U & \\ & & & & -U & & I_8 & -U & & \\ & -U & & & -U & & & I_8 & & \\ -U & & & & & -U & & & I_8 & \\ & & & & & & & -U & & I_8 \end{bmatrix} \tag{5.21}$$

und $A_2 \in M_{80}(\mathbb{R})$

$$A_2 = \begin{bmatrix} I_8 & & & -V & & & -V & & -V & \\ -V & I_8 & & -V & & & & & & -V \\ & & I_8 & & -U & & -U & & & \\ -V & & -V & I_8 & & & & -V & & \\ & & & -V & I_8 & & & & & -V \\ & & -V & & & I_8 & & & -V & \\ & & & & -U & & I_8 & -U & & \\ & -V & & & -V & & & I_8 & & \\ -V & & & & & -V & & & I_8 & \\ & & & & & & & -V & & I_8 \end{bmatrix}, \tag{5.22}$$

deren Einzelblöcke $U, V \in M_8(\mathbb{R})$ jeweils Inverse der Matrizen

$$U^{-1} = \begin{bmatrix} 6 & -1 & & & & & & \\ -1 & 6 & -1 & & & & & \\ & -1 & 6 & -1 & & & & \\ & & -1 & 6 & -1 & & & \\ & & & -1 & 6 & -1 & & \\ & & & & -1 & 6 & -1 & \\ & & & & & -1 & 6 & -1 \\ & & & & & & -1 & 6 \end{bmatrix}$$

und

$$V^{-1} = \begin{bmatrix} 10 & & & & & & & \\ & 20 & -2 & & & & & \\ & & 30 & & & & & \\ & & & 40 & & & & \\ & & -2 & & 50 & & -2 & \\ & & & & & 60 & & -10 \\ & & & & & & 70 & \\ & & -2 & & & & -2 & 80 \end{bmatrix}$$

sind. Diese Matrizen wurden bereits in der Arbeit [ENV91] benutzt, so daß hier eine Möglichkeit des Vergleiches mit bekannten Ergebnissen gegeben ist. Die Matrizen besitzen Blockstruktur, sind jedoch im Gegensatz zu den Matrizen des Modellproblems nicht bandstrukturiert.
Als dritte Matrizenklasse wurden Koeffizientenmatrizen des Modellproblems mit Gitterweiten für $n = 5, 9, 13, 17$ gewählt. Die Genauigkeit des Gitters bestimmt die Anzahl der gesuchten Unbekannten, die quadratisch mit der Anzahl der Teilintervalle wächst. Die Iterationsmatrizen des Modellproblems sind bandstrukturierte Blockmatrizen.

5.5.2 Untersuchung einer Reduktion um den Faktor $\frac{1}{e}$

Ein Vergleichskriterium für die Leistung eines Iterationsverfahrens ist der Zeitaufwand, der zur Reduktion eines Fehlers um den Faktor $\frac{1}{e}$ benötigt wird. Der Zeitaufwand berechnet sich aus der Zeitkomplexität eines Iterationsschrittes multipliziert mit der in Abschnitt 5.2 definierten Iterationszahl. Die Zeitkomplexität eines Iterationsschrittes ergibt sich aus Gleichung (5.14), wobei jedoch der Term $T^{(I)}(P, N)$ durch die Funktion

$$T^{(I)}(P, N) = 185 \left(\frac{N}{P}\right)^2 + 186\frac{N}{P} + 62N + 1500 \qquad (5.23)$$

ersetzt werden muß. Die numerische Iteration auf jedem Prozessor wird im benutzten Progamm (Anhang C) in der Funktion *iterate* beschrieben. Die

Umsetzung des C-Programmcodes in Maschinenbefehle des Transputers ergibt eine Befehlsfolge, deren Ausführungsverhalten durch die Funktion (5.23) beschrieben wird. Die Abweichungen zur theoretischen Komplexität (5.11) eines Iterationsschrittes ergeben sich durch einen zusätzlichen Aufwand für die Verwaltung von Subroutinen und Kontrollmechanismen.

Die Iterationszahl berechnet sich für Iterationsmatrizen, die aus Block-Jacobi-Zerlegungen der Testmatrizen gewonnen wurden. Für Einprozessor-

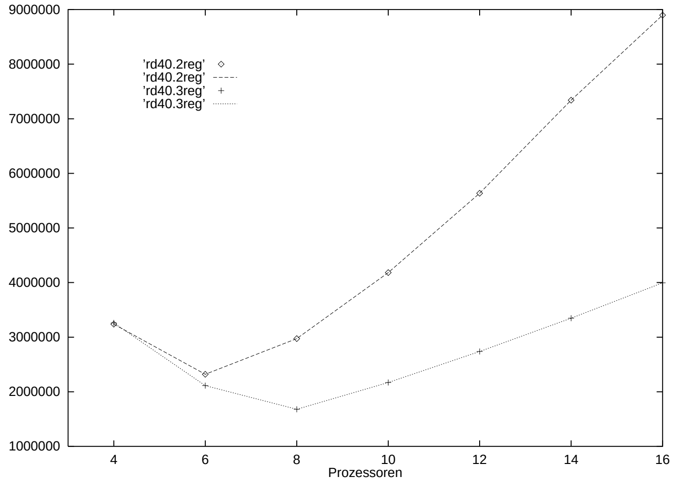

Abbildung 5.5: Vollbesetzte Matrix mit $N = 40$

systeme entspricht dies der Iterationsmatrix $H = \sum_{l=1}^{m} M_l^{-1} N_l$ eines sequentiellen Block-Jacobi-Verfahrens. Für das asynchrone Block-Jacobi-Verfahren auf einem Multiprozessorrechner wurde die Iterationsmatrix $\hat{H}$ aus Gleichung (3.7) betrachtet. Die aus den Spektralradien der Testmatrizen berechneten Iterationszahlen sowie die Zeitkomplexitäten der Verfahren für eine Fehlerreduktion um den Faktor $\frac{1}{e}$ geben die Tabellen in Anhang B wieder.

Die Umsetzung dieser Tabellen in die nachfolgenden $T(P)$-Diagramme zeigt

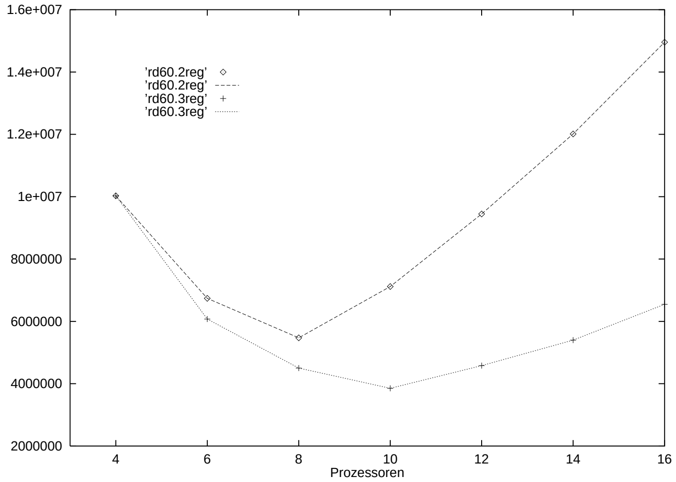

Abbildung 5.6: Vollbesetzte Matrix mit $N = 60$

das theoretische asymptotische Laufzeitverhalten der betrachteten Iterationsverfahren für die untersuchten Testmatrizen. Die Werte der Zeitachse sind in der Einheit *clocktick* angegeben. Die Kurven für 2-reguläre Graphen werden durch das Kürzel "th2", für 3-reguläre Graphen durch das Kürzel "th3" bezeichnet.

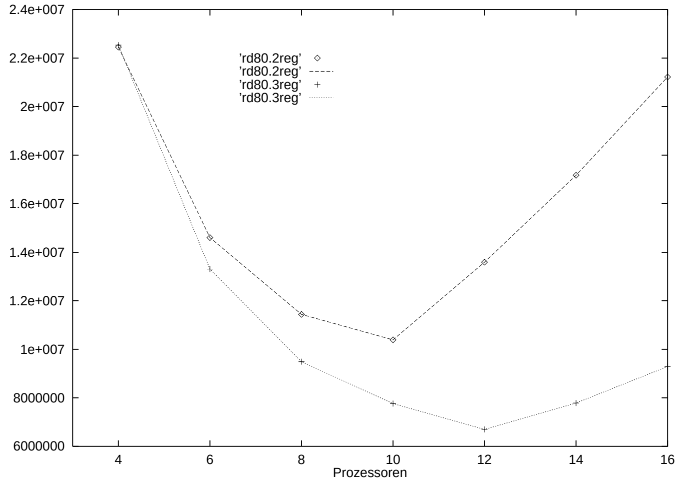

Abbildung 5.7: Vollbesetzte Matrix mit $N = 80$

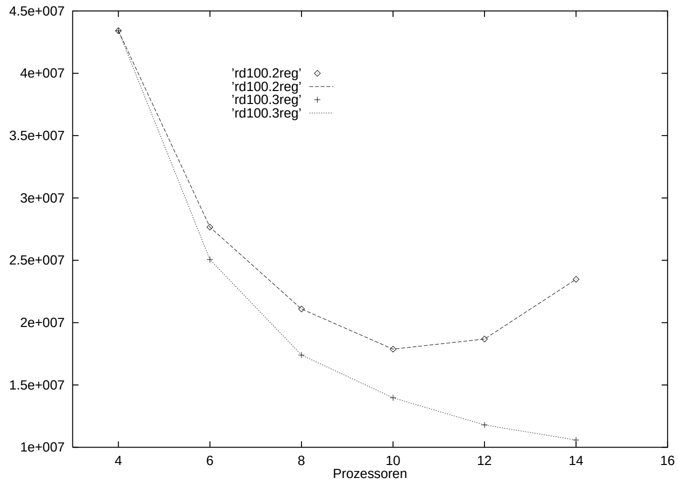

Abbildung 5.8: Vollbesetzte Matrix mit $N = 100$

Deutlich erkennt man in der Folge der Abbildungen 5.5 bis 5.9 die zunehmenden Möglichkeiten zur Parallelisierung für größer werdende Probleme, die sich besonders bei 3-regulären Graphen bemerkbar macht.

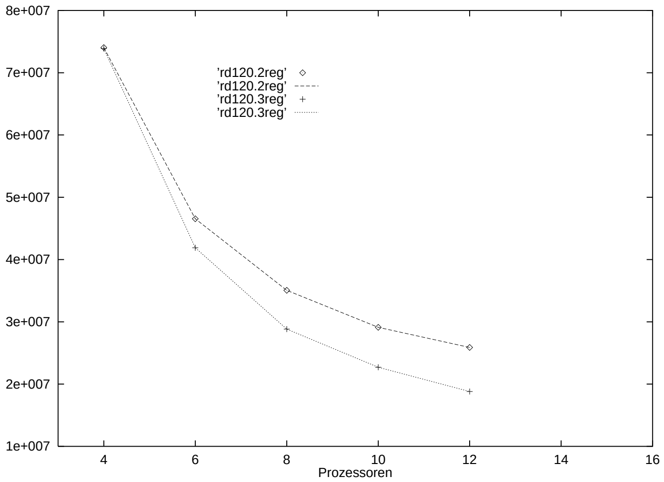

Abbildung 5.9: Vollbesetzte Matrix mit $N = 120$

Die Testmatrizen A_1 und A_2 zeigen ein ähnliches Verhalten wie die vollbesetzten Matrizen, auch wenn die Laufzeiten aufgrund des kleineren Spektralradius der Matrizen kleiner sind. Diese Beobachtung bestätigt sich auch für die Iterationsmatrizen zum Modellproblem.

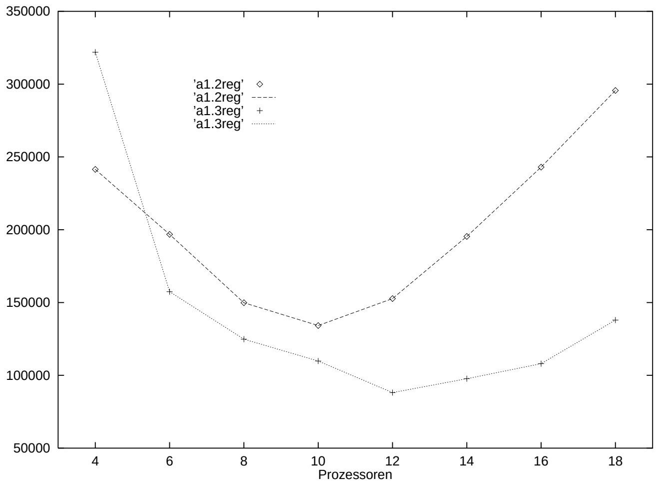

Abbildung 5.10: Matrix A_1

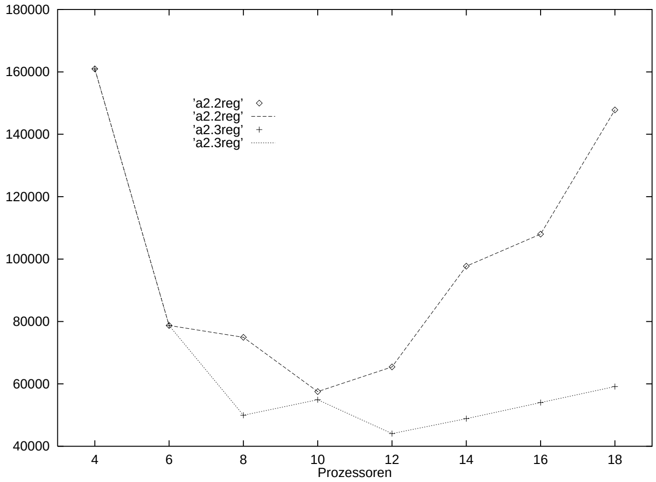

Abbildung 5.11: Matrix A_2

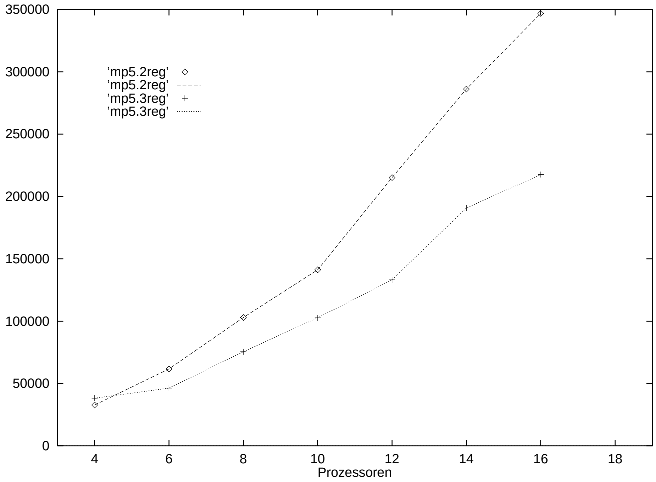

Abbildung 5.12: Modellproblem für $n = 5$

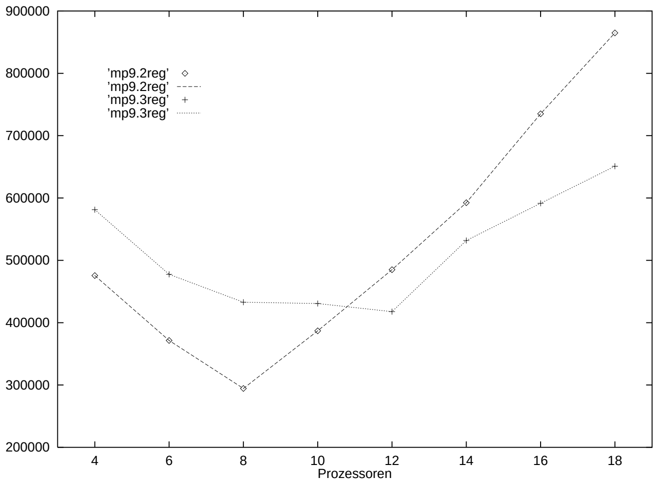

Abbildung 5.13: Modellproblem für $n = 9$

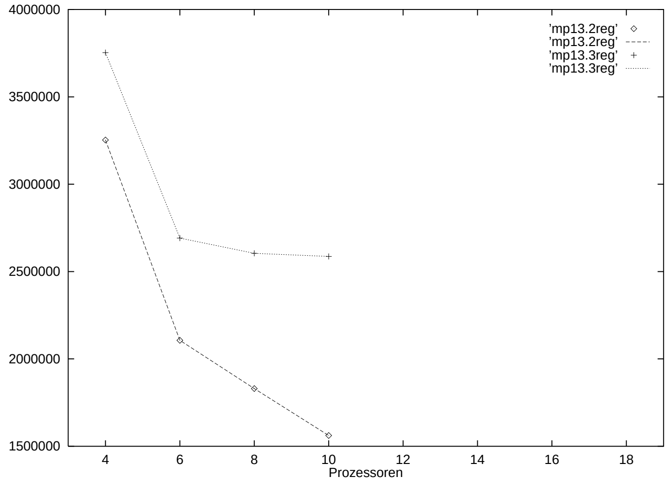

Abbildung 5.14: Modellproblem für = 13

Die theoretischen Untersuchungen zeigen, daß die Aufgabe der iterativen Lösung eines linearen Gleichungssystems skalierbar ist und die Auslastung eines Multiprozessorrechners mit wachsender Problemgröße steigt. Eine Beurteilung dieses Verhaltens erlaubt der in Abschnitt 5.1 definierte *skalierte speedup*. Wird die Rechenzeit auf einem Multiprozessorsystem durch (5.14) gegeben, so errechnet sich die Laufzeit der gleichen Aufgabe auf einem Einprozessorrechner aus der sequentiellen Komponente (5.13) und der Iterationszeit (5.11), die jedoch P mal ausgeführt wird. Somit ist der *skalierte speedup* durch die Gleichung

$$SC_P(N) = \frac{T_1(N)}{T_P(N)} = \frac{T_1^{(S)}(N) + P\,T_P^{(I)}(N)}{T_P^{(S)}(N) + \max(T_P^{(I)}(N), T_P^{(K)}(N))} \qquad (5.24)$$

gegeben.

Die Abbildung 5.15 demonstriert das Verhalten des *skalierten speedup* für 4, 8, 12 und 16 Prozessoren, die als Ringgraph verschaltet sind.

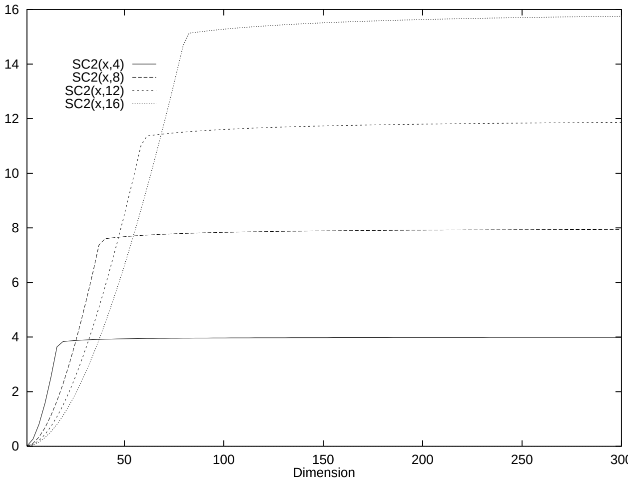

Abbildung 5.15: *Skalierter speedup* für Ringgraphen

Mit wachsender Problemgröße steigt der *speedup* an, erreicht jedoch erst für einen hohen Ladungsfaktor sein Maximum. Danach ändert sich die Kurve nur noch geringfügig. Der stufenförmige Aufbau der Kurven zeigt, daß ein Einsatz zusätzlicher Prozessoren zur Steigerung der Geschwindigkeit erst wieder für ein entsprechend größeres

Gleichungssystem sinnvoll ist.

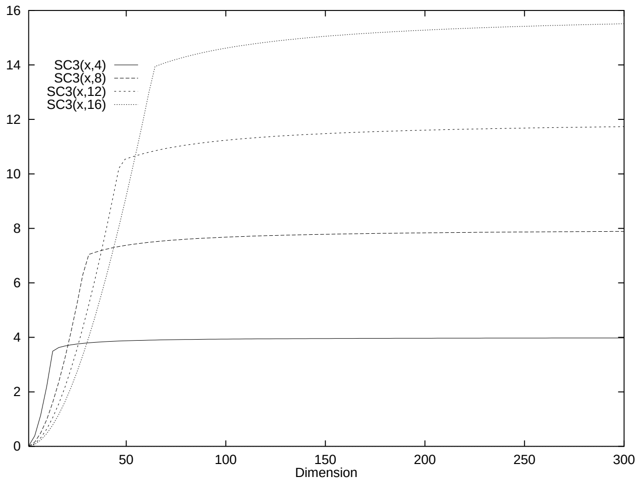

Abbildung 5.16: *Skalierter speedup* für 3-reguläre Graphen

Im Vergleich zu den Ringgraphen wird der maximale *speedup* für 3-reguläre Graphen schon für kleinere Problemgrößen erreicht.
Auch der *scaleup* für die iterative Lösung eines linearen Gleichungssystems auf einem Multiprozessorrechner läßt sich aus dem Modell (5.14) berechnen. Gesucht wird zu jedem N ein M, so daß $T_1(M) = T_P(N)$.

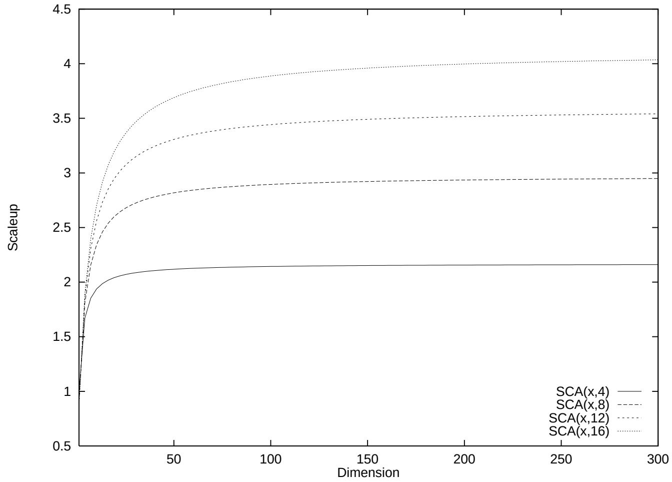

Abbildung 5.17: *Scaleup*

Betrachtet man für eine fest gewählte Prozessorzahl P nur den Bereich der Problemgrößen, in dem die Kommunikationszeit kleiner als die Iterationszeit ist, so ist

$$\begin{aligned} T_P(N) &= T_P^{(S)}(N) + T_P^{(I)}(N) \\ T_1(M) &= T_1^{(S)}(M) + P\,T_P^{(I)}(M). \end{aligned}$$

Gleichsetzen und Auflösen nach $M(N,P)$ ergibt

$$M(N,P) = \frac{(t_1 - t_4)P}{2t_1(P+1)} + \frac{\sqrt{(t_1-t_4)^2P^2 + (P+1)\left(4t_1t_4(2P-1)N + 4t_1^2\left(\frac{N^2}{P} - N + N^2\right)\right)}}{2t_1(P+1)}.$$

Die Abbildung 5.17 zeigt die Entwicklung des *scaleup* für die Prozessorzahlen $P = 4, 8, 12, 16$. Man erkennt, daß der *scaleup* in der Größenordnung von $\sqrt{P}$ wächst.

Sowohl der *speedup* als auch der *scaleup* zeigen, daß es zu einer gegebenen Problemgröße N eine bestimmte Anzahl Prozessoren gibt, für die die Laufzeit der Iteration optimal ist. Diese Aussage bestätigt sich auch in den anschließenden Messungen.

5.5.3 Messwerte aus Experimenten

Die numerischen Experimente wurden auf einem Parsytec Supercluster64 Transputersystem durchgeführt. Die Programmierung erfolgte mit dem PARC Compiler der Firma PARSEC. Dieser Compiler ist neben dem üblichen Sprachumfang der Programmiersprache C mit zusätzlichen Sprachelementen versehen, die parallele Prozesse sowie *message passing* unterstützen.

Das zu lösende Gleichungssystem (5.10) wurde für die oben definierten Klassen von Koeffizientenmatrizen derart konstruiert, daß der Vektor $\xi = \mathbb{1}_n$ Lösungsvektor des Systems ist. Als Startvektor für jede Iteration wurde $x_0 = (0.9, \ldots, 0.9)^T$ gewählt. Die Zeitmessung umfaßt lediglich die Iterationszeiten, nicht aber die Initialisierungszeit des Rechners. Sobald eine Iterierte x_k die Gleichung

$$\|x_k - \xi\| \leq 10^{-6}$$

bzgl. der Maximumnorm erfüllt, erfolgt der Abbruch der Iteration.

Die anschließenden $T(P)$-Diagramme zeigen das unter Berücksichtigung der oben beschriebenen Versuchsanordnung gemessene Verhalten für vollbesetzte Matrizen mit den Dimensionen $N = 40, 60, 80, 100, 120, 300$. Der Kurvenverlauf ist in allen Diagrammen nahezu identisch. Für konstante Prozessorzahlen steigt die Rechenzeit mit wachsender Problemgröße quadratisch an. Die Minima der Kurven wandern für wachsende N in Richtung größerer Prozessorzahlen. Die optimale Nutzung eines Transputernetzwerkes ist selbst bei großen Matrizen bereits für wenige Prozessoren erreicht, da der Kommunikationsaufwand die Rechenleistung sehr früh dominiert.

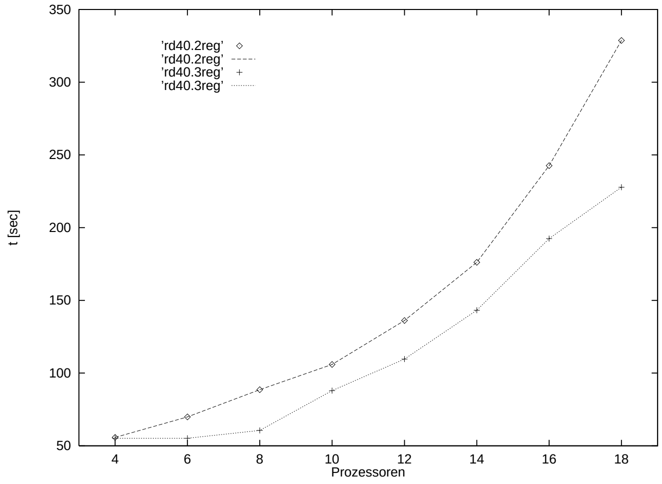

Abbildung 5.18: Vollbesetzte Matrix mit $N = 40$

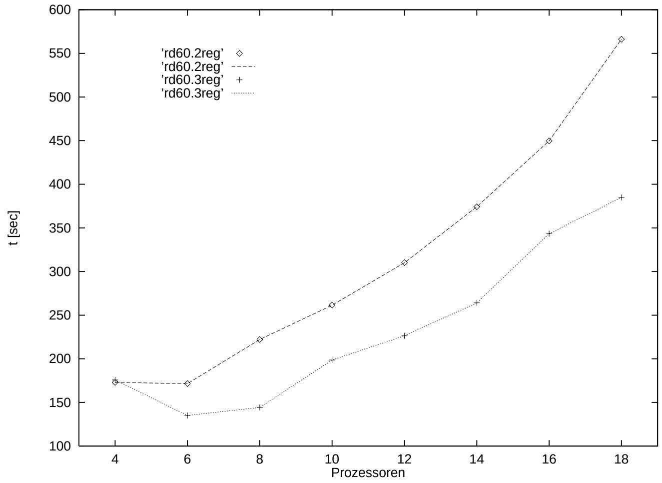

Abbildung 5.19: Vollbesetzte Matrix mit $N = 60$

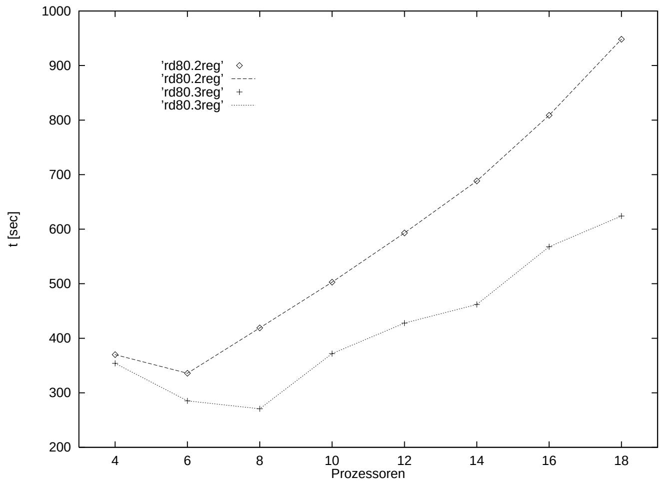

Abbildung 5.20: Vollbesetzte Matrix mit $N = 80$

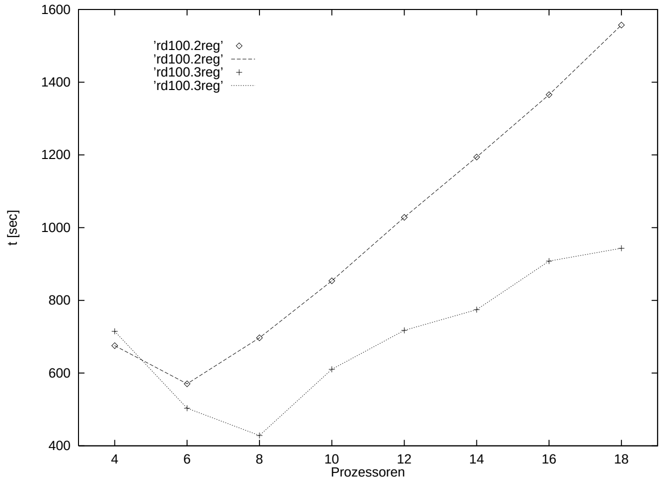

Abbildung 5.21: Vollbesetzte Matrix mit $N = 100$

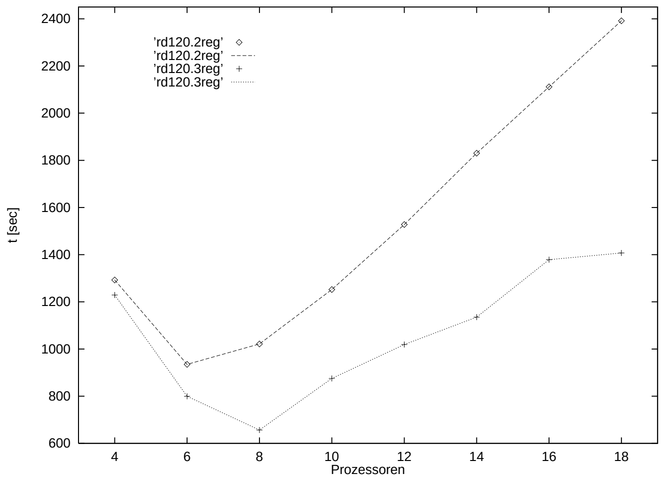

Abbildung 5.22: Vollbesetzte Matrix mit $N = 120$

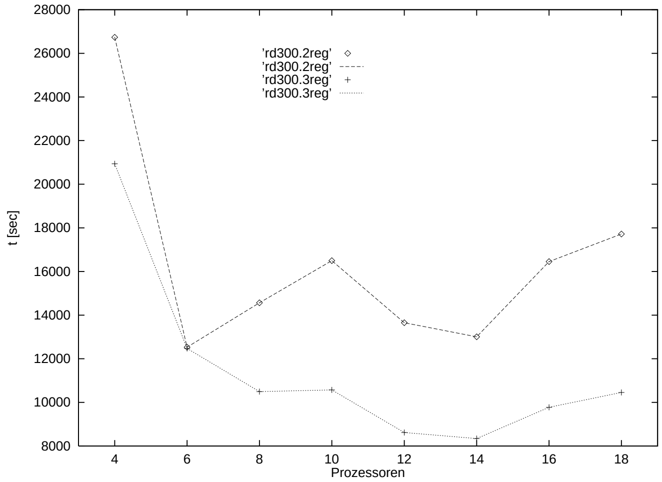

Abbildung 5.23: Vollbesetzte Matrix mit $N = 300$

Die Messungen bestätigen somit das theoretisch vorhergesagte Verhalten für vollbesetzte Matrizen für das in Abschnitt 3.1 beschriebene Modell der asynchronen Iteration.

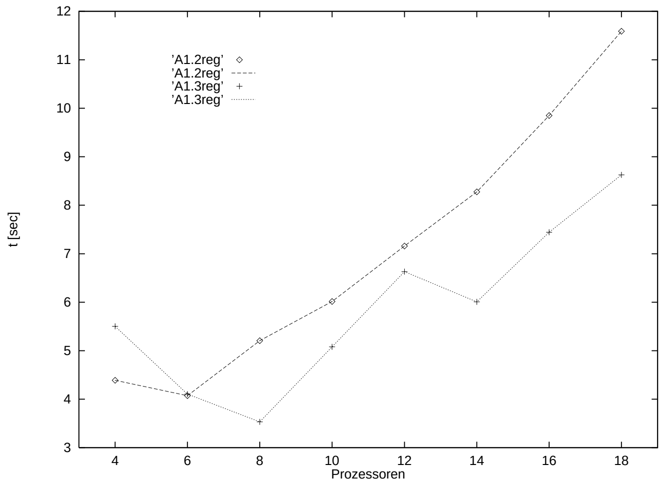

Abbildung 5.24: $T(P)$-Diagramm für A_1

Ähnlich wie im Fall der vollbesetzten Iterationsmatrix der Dimension 80 ist der Verlauf der Meßpunkte für die Matrizen A_1 und A_2. Die Iteration verläuft auf 3-regulären Graphen schneller als auf 2-regulären, ist jedoch sehr unregelmäßig.

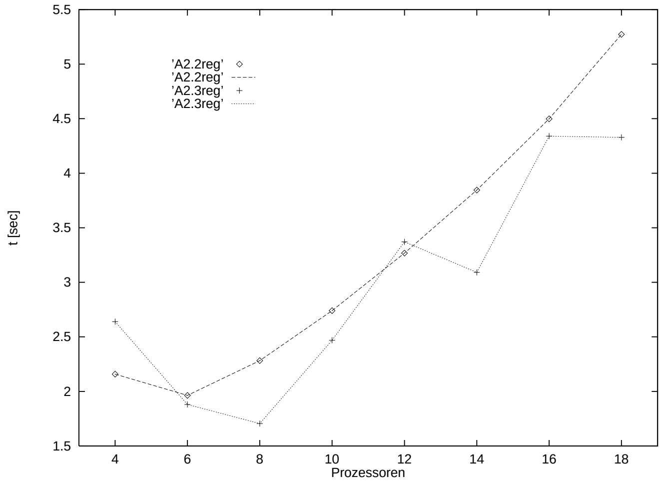

Abbildung 5.25: $T(P)$-Diagramm für A_2

Die Laufzeit auf 2-regulären Graphen ist für 6 Prozessoren sowie auf 3-regulären Graphen für 8 Prozessoren minimal. Dieser zeitliche Verlauf des $T(P)$-Diagramms korrespondiert mit den theoretischen Abschätzungen für die Laufzeit einer Fehlerreduktion um den Faktor $\frac{1}{e}$ (Anhang B).

Die Laufzeiten der Iterationen für die Matrix A_2 verhalten sich ähnlich wie die Laufzeiten für die Matrix A_1, auch wenn das Verfahren für alle P schneller konvergiert (siehe Spektralradien). Auffällig im Verlauf der Meßwerte beider Kurven ist ein zusätzliches lokales Minimum für 14 Prozessoren. Dieser Effekt ist bei einer vollbesetzten Matrix der Dimension 80 nicht zu beobachten und geht auch nicht aus den theoretischen Laufzeitabschätzungen für die

Matrizen A_1 und A_2 hervor.

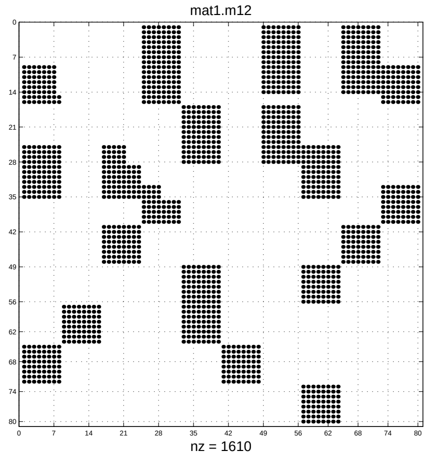

Abbildung 5.26: Nullmuster für 12 Prozessoren

Aus der bisherigen Analyse einer Iteration ergibt sich aber, daß die Laufzeiten der Iteration nach Durchlaufen des absoluten Minimums durch die Kommunikation zwischen den Prozessoren dominiert werden. Betrachtet man nun die Nullmuster der aus A_1 und A_2 gewonnenen Iterationsmatrizen für 12 und 14 Prozessoren, so kann man aus diesem Muster direkt die Abhängigkeiten zwischen den Prozessoren ablesen.

In den Abbildungen 5.26 und 5.27 werden von Null verschiedene Einträge der Iterationsmatrix durch einen Punkt gekennzeichnet. Lediglich diese Einträge liefern einen Beitrag bei der Berechnung der nächsten Iterierten. Dies heißt aber auch, daß ein Prozessor i die berechnete Iterierte eines Prozessors j nicht benötigt, wenn in der Blockzeile i der Block j kein von Null verschiedenes Element enthält. Analysiert man nun unter Berücksichtigung dieser Beobachtung die für einen vollständigen *broadcast* aller Prozessoren benötigten Nachrichten, so erhält man für 12, 14 und 16 Prozessoren die in der anschließenden Tabelle gegebenen Anzahlen von Fließkommazahlen, die für einen *broadcast* verschickt werden müssen.

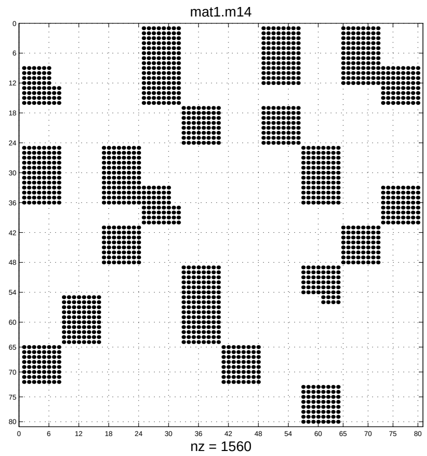

Abbildung 5.27: Nullmuster für 14 Prozessoren

Die Tabelle spiegelt qualitativ den Kurvenverlauf im $T(P)$-Diagramm der Matrizen A_1 und A_2 für 12, 14 und 16 Prozessoren wider. Hieraus wird deutlich, daß der Spektralradius einer Iterationsmatrix und die hieraus gewonnene Abschätzung für die Anzahl der Iterationen nicht allein maßgebend ist. Die spezielle Struktur einer Matrix muß bei der Wahl eines

Prozessoren	Zahlen
12	1033
14	984
16	1180

Tabelle 5.2: Verschickte FP-Zahlen je *broadcast*

Prozessorgraphen in dessen Struktur integriert werden.
Dieser Effekt wird noch deutlicher bei der Betrachtung der aus dem Modellproblem gewonnenen Iterationsmatrizen. Die theoretische Analyse aus Abschnitt 5.5.2 zeigt eine schnellere Laufzeit für 2-reguläre Graphen. Dieses Verhalten erkennt man auch in den Meßreihen wieder. Entsprechend der Vorhersage ist die Laufzeit bei 3-regulären Graphen schlechter als bei 2-regulären Graphen vergleichbarer Prozessorzahl.

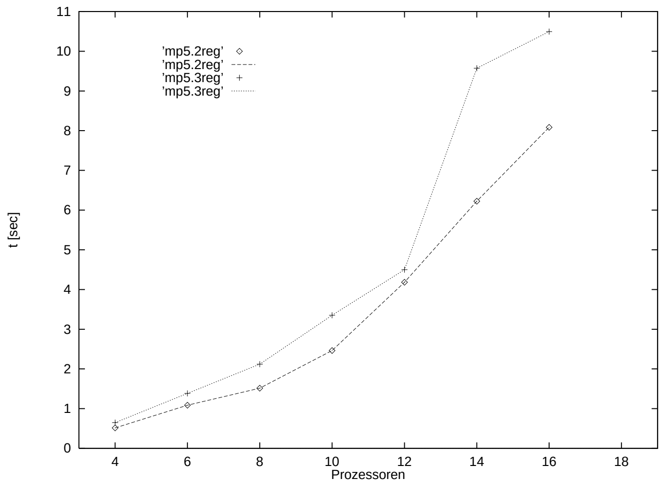

Abbildung 5.28: Modellproblem für $n = 5$

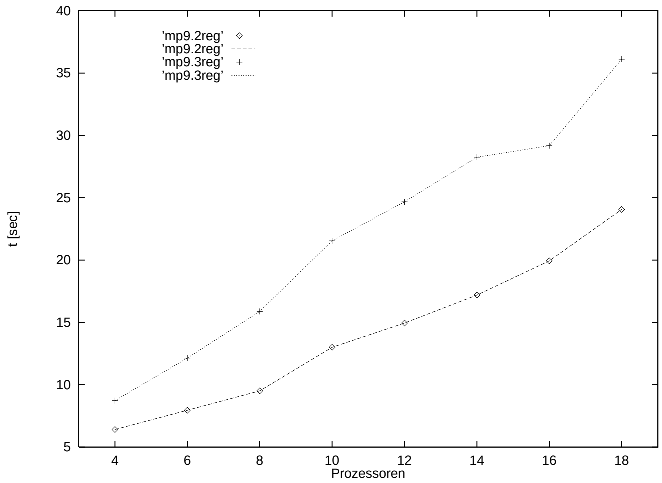

Abbildung 5.29: Modellproblem für $n = 9$

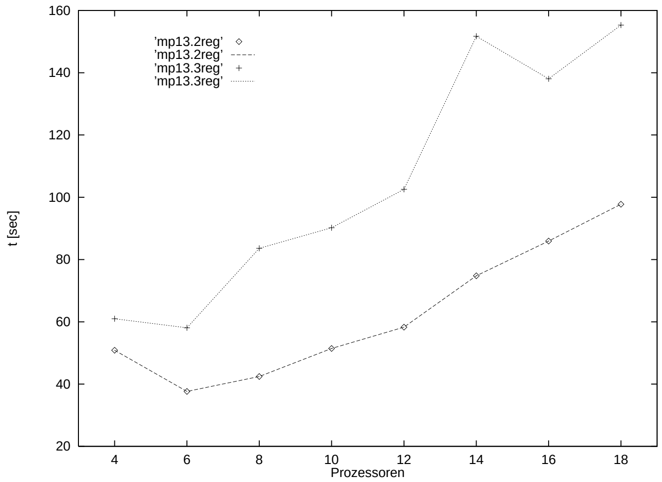

Abbildung 5.30: Modellproblem für $n = 13$

Der deutliche Sprung in der Laufzeit beim Übergang von 12 auf 14 Prozessoren tritt zum ersten Mal bei der Iterationsmatrix zum Modellproblem für

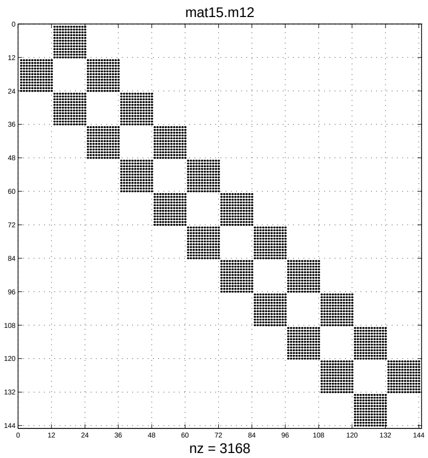

Abbildung 5.31: Nullmuster $P = 12$

$N = 13$ auf. Ein Vergleich der Abbildungen 5.31 und 5.32 zeigt, daß die Iterationsmatrix von der Blocktridiagonalgestalt für $P = 12$ in eine Blockpentadiagonalgestalt für $P = 14$ übergeht. Diese Blockstruktur erfordert einen größeren Kommunikationsaufwand als die Blocktridiagonalstruktur. Da bereits ab einer Prozessorzahl von $P = 6$ die Kommunikationszeit die Iterationszeit dominiert, resultiert aus dem Mehraufwand an Kommunikation sofort eine Verlängerung der Laufzeit. Dieses Beispiel zeigt wiederum sehr deutlich, daß jede Iterationsmatrix eine individuelle auf sie abgestimmte Struktur des Prozessorgraphen benötigt.

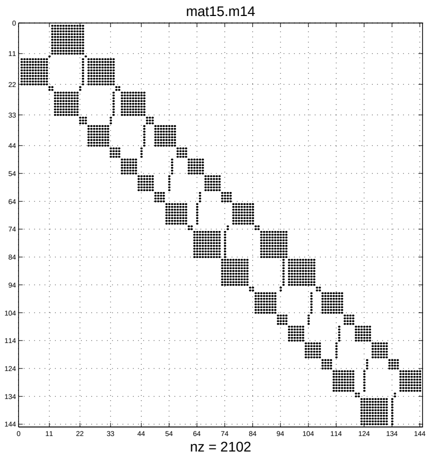

Abbildung 5.32: Nullmuster $P = 14$

Es zeigt sich, daß Ringgraphen bereits die optimale Struktur für Iterationsmatrizen des Modellproblems darstellen. Im Bereich kleiner Prozessorzahlen haben die Iterationsmatrizen Blocktridiagonalgestalt, d.h. jede Iteration benötigt nur die Information zweier benachbarter Prozessoren für die Berechnung der nächsten Iterierten. Diese Anforderungen an die Struktur des Prozessorgraphen werden aber genau durch einen Ringgraphen erfüllt. Erst größere Prozessorzahlen ergeben breitere Bandstrukturen für Iterationsmatrizen des Modellproblems.

Insgesamt zeigt ein Vergleich der theoretischen Berechnungen und der Meßergebnisse, daß das Modell aus Abschnitt 3.1 das Verhalten der asynchronen Iteration realistisch beschreibt. Das Minimum der Laufzeit wird korrekt ermittelt. Abweichungen ergeben sich im Bereich der Kommunikationszeiten. Treten große Diskrepanzen zwischen der Blockstruktur der Iterationsmatrix und der *broadcast* Strategie auf, so werden diese nicht im Modell berücksichtigt.

Anhang A

Tabellen zur Geschwindigkeit eines *broadcast*

Die Tabellen geben die über fünfzig Messungen gemittelte Laufzeit eines *broadcasts* auf 2- und 3-regulären Prozessorgraphen wieder. Die Spalten 3 und 4 enthalten die maximalen positiven bzw. negativen Abweichungen vom Mittelwert.

Prozessoren	Zeit [sec]	$\Delta+$ [sec]	$\Delta-$ [sec]
4	0.043	0.006	0.002
6	0.056	0.010	0.007
8	0.064	0.003	0.003
10	0.086	0.016	0.017
12	0.089	0.032	0.010
14	0.107	0.021	0.016
16	0.114	0.022	0.091
18	0.145	0.036	0.022
20	0.134	0.019	0.019
22	0.144	0.037	0.015
24	0.162	0.046	0.025
40	0.250	0.050	0.024

Tabelle A.1: Broadcastzeiten auf 3-regulären Graphen

Prozessoren	Zeit [sec]	$\Delta+$ [sec]	$\Delta-$ [sec]
4	0.043	0.006	0.002
6	0.060	0.002	0.002
8	0.078	0.007	0.005
10	0.091	0.008	0.009
12	0.117	0.008	0.013
14	0.130	0.030	0.012
16	0.144	0.030	0.012
18	0.161	0.027	0.016
20	0.179	0.022	0.017
22	0.199	0.017	0.009
24	0.207	0.023	0.016
26	0.231	0.032	0.016
28	0.248	0.029	0.028
30	0.262	0.038	0.017
32	0.287	0.022	0.019
34	0.299	0.031	0.017
36	0.317	0.028	0.013
38	0.338	0.026	0.027
40	0.358	0.024	0.034

Tabelle A.2: Broadcastzeiten auf Ringgraphen

Anhang B

Theoretische Laufzeitabschätzungen

Die nachfolgenden Tabellen vermitteln einen Eindruck vom theoretischen Laufzeitverhalten einer asynchronen Iteration gemäß der Gleichung (5.14). Für die in Abschnitt 5.5.1 beschriebenen Koeffizientenmatrizen wurden abhängig von der Prozessorenzahl Iterationsmatrizen berechnet. Aus den mit *Matlab* ermittelten Eigenwerten der Iterationsmatrizen für das sequentielle Block-Jacobi-Verfahren bzw. für das Modell der asynchronen Iteration ergaben sich die Iterationszahlen für die Reduktion des Fehlers um den Faktor $\frac{1}{e}$. Die Laufzeiten für die sequentielle bzw. parallele Iteration und die Kommunikation sind in der Einheit *clocktick*, d.h. ein CPU-Takt, angegeben.

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	7	7006272	13	3252912	203190
6	9	6318648	18	2106216	350100
8	13	7320144	26	1830036	587860
10	16	7808640	32	1561728	816192

Tabelle B.1: Modellproblem mit $N = 13$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	7	7006272	15	3753360	156300
6	9	6318648	23	2691276	298241
8	13	7320144	37	2604282	557701
10	16	7808640	53	2586612	901212

Tabelle B.2: Modellproblem mit $N = 13$ auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	5	1056880	9	475596	81270
6	7	1114176	14	371392	169610
8	9	1246752	17	294372	253470
10	10	1305200	22	287144	386892
12	10	1288320	24	257664	484968
14	10	1307460	26	242814	592514
16	11	1484032	29	244528	735150
18	11	1546380	31	242110	864807

Tabelle B.3: Modellproblem mit $N = 9$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	5	1056880	11	581284	66220
6	7	1114176	18	477504	145386
8	9	1246752	25	432900	248500
10	10	1305200	33	430716	386892
12	10	1288320	31	332816	417362
14	10	1307460	35	326865	531755
16	11	1484032	35	295120	591500
18	11	1546380	35	273350	650930

Tabelle B.4: Modellproblem mit $N = 9$ auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	3	65472	6	32736	30420
6	4	91464	8	30488	61720
8	4	103488	10	32340	102900
10	4	118680	11	32637	141174
12	5	169320	14	39508	215138
14	5	191380	16	43744	286208
16	5	214240	17	45526	346800

Tabelle B.5: Modellproblem mit $N = 5$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	3	65472	7	38192	23660
6	4	91464	9	34299	46287
8	4	103488	11	35574	75460
10	4	118680	12	35604	102672
12	5	169320	13	36686	133185
14	5	191380	16	43744	190800
16	5	214240	16	42848	217600

Tabelle B.6: Modellproblem mit $N = 5$ auf 3-regulären Graphen

B.1 Iterationszahlen und Laufzeiten für A_1 und A_2

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	2	643840	3	241440	31050
6	2	472344	5	196810	67924
8	2	399520	6	149820	98700
10	2	366520	7	128156	134190
12	2	352536	7	102823	152691
14	2	350196	8	100056	195416
16	2	354880	9	99810	243000
18	2	364284	10	101190	295630

Tabelle B.7: Dünnbesetzte Matrix A_1 auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	2	643840	4	321920	27600
6	2	472344	4	157448	36224
8	2	399520	5	124850	54835
10	2	366520	6	109848	76680
12	2	352536	6	88134	87252
14	2	350196	6	75042	97710
16	2	354880	6	66540	108000
18	2	364284	7	70833	137963

Tabelle B.8: Dünnbesetzte Matrix A_1 auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	1	321920	2	160960	20700
6	1	236172	2	78724	27170
8	1	199760	3	74910	49350
10	1	183080	3	54924	57510
12	1	176268	3	44067	65439
14	1	175098	4	50028	97708
16	1	177440	4	44360	108000
18	1	182142	5	50595	147815

Tabelle B.9: Dünnbesetzte Matrix A_2 auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	1	321920	2	160960	13800
6	1	236172	2	78724	18112
8	1	199760	2	49940	21934
10	1	183080	3	54924	38340
12	1	176268	3	44067	43626
14	1	175098	3	37521	48855
16	1	177440	3	33270	54000
18	1	182142	3	30357	59127

Tabelle B.10: Dünnbesetzte Matrix A_2 auf 3-regulären Graphen

B.2 Iterationszahlen und Laufzeiten für vollbesetzte Matrizen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	62	5577520	144	3238560	1015200
6	69	5054526	190	2319710	1883850
8	72	4959360	236	2031960	2973600
10	73	5069120	276	1916544	4182750
12	75	5435100	317	1914363	5634992
14	76	5844552	361	1982973	7339852
16	77	6331248	389	1999071	8898375
18	77	6785856	441	2159136	11200518

Tabelle B.11: Dimension $N = 40$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	62	5577520	145	3261050	681500
6	69	5054526	173	2112157	1143530
8	72	4959360	195	1678950	1638000
10	73	5069120	214	1486016	2169960
12	75	5435100	231	1395009	2737581
14	76	5844552	247	1356771	3348085
16	77	6331248	262	1346418	3995500
18	77	6785856	266	1302336	4503912

Tabelle B.12: Dimension $N = 40$ auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	91	17057040	214	10028040	1861800
6	101	14380380	284	6739320	3337000
8	107	13382704	350	5471900	5083750
10	109	12955740	414	4920804	7116660
12	111	13120200	477	4698450	9444600
14	113	13640004	537	4630014	12014301
16	114	14272800	600	4695000	14958000
18	115	15065460	658	4788924	18087762

Tabelle B.13: Dimension $N = 60$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	91	17057040	214	10028040	1241200
6	101	14380380	256	6074880	2005248
8	107	13382704	288	4502592	2788704
10	109	12955740	324	3851064	3713040
12	111	13120200	347	3417950	4580400
14	113	13640004	362	3121164	5399230
16	114	14272800	394	3083050	6548280
18	115	15065460	397	2889366	7275422

Tabelle B.14: Dimension $N = 60$ auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	120	38630400	279	22453920	2887650
6	133	31889676	371	14603302	5040035
8	140	27966400	458	11436260	7534100
10	144	26363520	542	9922936	10390140
12	147	25911396	623	9151247	13589499
14	149	26089602	703	8792421	17172181
16	150	26616000	786	8716740	21222000
18	151	27503442	863	8732697	25512869

Tabelle B.15: Dimension $N = 80$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	120	38630400	280	22534400	1932000
6	133	31889676	338	13304356	3060928
8	140	27966400	380	9488600	4167460
10	144	26363520	424	7762592	5418720
12	147	25911396	456	6698184	6631152
14	149	26089602	478	5978346	7784230
16	150	26616000	516	5722440	9288000
18	151	27503442	521	5271999	10268389

Tabelle B.16: Dimension $N = 80$ auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	151	75503400	352	43419200	4224000
6	167	59224212	468	27661608	7216560
8	175	51266600	576	21092544	10584000
10	181	47440100	682	17875220	14424300
12	184	45387648	784	16115904	18688208
14	186	44648184	887	15208502	23477116

Tabelle B.17: Dimension $N = 100$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	151	75503400	352	43419200	2816000
6	167	59224212	424	25060944	4358720
8	175	51266600	475	17394025	5818750
10	181	47440100	533	13969930	7515300
12	184	45387648	574	11799144	9121434
14	186	44648184	600	10287600	10587000

Tabelle B.18: Dimension $N = 100$ auf 3-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	181	127040280	422	74048340	5760300
6	200	99552000	561	46540560	9677250
8	211	85379040	693	35051940	14067900
10	216	76878720	818	29114256	18920340
12	220	72468000	943	25885350	24376550

Tabelle B.19: Dimension $N = 120$ auf 2-regulären Graphen

	sequentiell		parallel		
P	It(H)	Laufzeit	It(H)	Laufzeit	Kommunikation
4	181	127040280	421	73872870	3831100
6	200	99552000	505	41894800	5807500
8	211	85379040	570	28830600	7713810
10	216	76878720	638	22707696	9837960
12	220	72468000	685	18803250	11804605

Tabelle B.20: Dimension $N = 120$ auf 3-regulären Graphen

Anhang C

Programmbeschreibung

Das Programm zur asynchronen Iteration auf einem Transputernetzwerk besteht aus zwei voneinander unabhängigen Teilen. Die Initialisierungsphase erfolgt sequentiell und dient der ordnungsgemäßen Bereitstellung des Transputernetzwerkes und eines für die Iteration benötigten Datensatzes. Jeder an der Iteration beteiligte Prozessor erhält einen Teil der Iterationsmatrix, den Startvektor und Informationen über die *broadcast* Strategie, die im gegebenen Prozessorgraphen angewendet werden soll. Ist die Verteilung der Daten beendet, erfolgt eine Synchronisation aller Prozessoren, so daß die Iteration auf allen Prozessoren zur gleichen Zeit beginnen kann.
Der zweite Teil des Programms beschreibt die asynchrone Iteration. Nachdem die erste Iterierte in jedem Prozessor berechnet wurde, werden die Kommunikationsroutinen gestartet, die in jedem Transputer durch die gleichen Prozesse realisiert werden. In jedem Prozessor befindet sich ein temporärer Lösungsvektor, dessen Komponenten jedem Prozeß im Transputer zugänglich sein müssen. Dieser Datenbereich wird auf dem Transputer als *shared memory* behandelt und durch einen Monitor geschützt. Der Monitor erlaubt gleichzeitige Leseoperationen der aktiven Prozesse, verhindert aber gleichzeitige Schreibzugriffe. Der Prozeß *iterate* berechnet die Iterierten. Er liest zunächst den temporären Lösungsvektor, berechnet die nächste Iterierte und schreibt diese dann wieder in den Lösungsvektor zurück. Die über die neue Iterierte benachrichtigten Prozesse *get from monitor* lesen die neuen Daten und verschicken sie über das von ihnen verwaltete *link* des Transputers. Auf dem Nachbartransputer wartet der Prozeß *put to monitor* auf eine Nachricht und übergibt sie nach Erhalt an den Monitor. Gemäß der auf jedem Prozessor befindlichen Routingtabellen beauftragt er auch *get from monitor* Prozesse mit der Weiterverschickung dieser Daten. Der Prozeß *abbruch* ist der einzige Prozeß, der ausschließlich liest. In regelmäßigen Abstanden überprüft er, ob das geforderte Kriterium erfüllt ist. Ist dies der Fall, so stoppt *abbruch* die Zeit und bricht die Iteration ab. Das Zusammenspiel der Prozesse in einem Transputer zeigt Abbildung C.1. Die Pfeile symbolisieren den

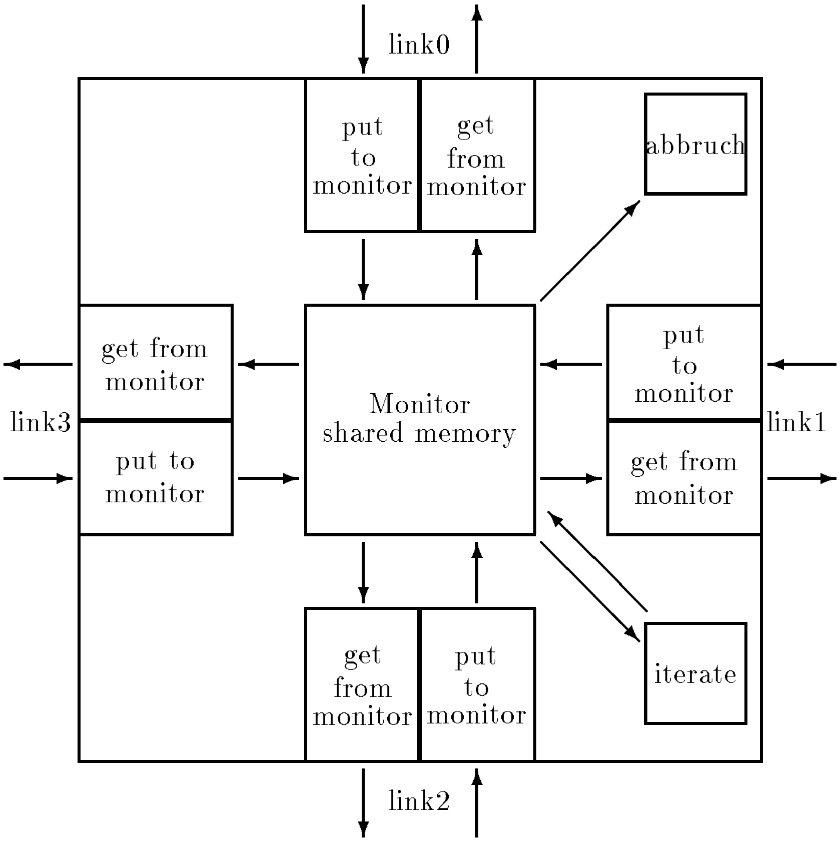

Abbildung C.1: Prozeßschema in jedem Transputer

Informationsfluß.
Den Programmtext zu den in der Abbildung C.1 benutzten Prozessen findet man nachfolgend:

```
/*******************************************************************/

/* numerik.c                                                       */

/*******************************************************************/

#include "par_defs.h"



/*******************************************************************/

/* Vorbereitung eines lokalen Iterationsschrittes                  */

/*******************************************************************/

void iterate()

{
```

```
  int i_link;

  request_read();
  read_info (glob_vek, global_x);
  release_read();
  block_iteration (d_form[p_name].size, d_form[p_name].offset,
   temp, global_x);
  request_write();
  cp_vek (d_form[p_name].size, temp, glob_vek[p_name]);
  if (_Tn == 1)
    set_bit (p_name, num_flag);
  par (i_link = 0; i_link < _LPT; i_link ++)
    {
      if (route[i_link][nr_of_proc + 1] == 1)
        set_bit (p_name, send_flag[i_link]);
    }
  release_write();
  signal (numerik, iterate);
}

/*****************************************************************/
/* Iteration                                                     */
/*****************************************************************/
void block_iteration(size, offset, re_seite, x_alt)
int size, offset;
VECTOR re_seite, x_alt;
{
```

```
  int i_dim, i_part;
  int index;

  for (i_part = 0; i_part < size; i_part ++)
    {
      re_seite[i_part] = local_data->b_teil[i_part];
      for (i_dim = size; i_dim < dim; i_dim ++)
        {
          index = (offset + i_dim) % dim;
          re_seite[i_part] -=
            local_data->block[i_part][index] * x_alt[index];
        }
    }
  gauss (size, offset, local_data->block, re_seite);
}

/*********************************************************************/
/* Loesung der lokalen Gleichungssysteme                             */
/*********************************************************************/
void gauss (size, offset, A, x)
int size, offset;
MATRIX A;
VECTOR x;
{
  int i, j;
```

```
  for (i = 1; i < size; i ++)
    {
      for (j = 0; j < i; j ++)
        x[i] -= A[i][offset + j] * x[j];
    }
  x[size - 1] =
    x[size - 1] / A[size - 1][offset + size - 1];
  for (i = size - 2; i >= 0; i --)
    {
      for (j = i + 1; j <= size - 1; j ++)
        x[i] -= A[i][offset + j] * x[j];
      x[i] /= A[i][offset + i];
    }
}

/******************************************************************/
/* communic.c                                                     */
/******************************************************************/
#include "par_defs.h"

/******************************************************************/
/* reader                                                         */
/* falls neue Daten im glob_vek stehen, werden diese gelesen und  */
/* auf einen Prozessorlink geschrieben                            */
/******************************************************************/
void get_from_monitor (link, proc)
int link, proc;
```

```
{
  VECTOR tmp_data;
  int send;

  tmp_data = (VECTOR) calloc (d_form[proc].size, dbl_size);
  while (TRUE)
    {
      request_read();
      if ((send = cmp_bit(proc, send_flag[link])) == TRUE)
        {
          cp_vek (d_form[proc].size, glob_vek[proc], tmp_data);
          unset_bit (proc, send_flag[link]);
        }
      release_read();
      if (send)
        {
          P (&s_link[link]);
          SendLink(link, &proc, i_size);
          SendLink(link, tmp_data, d_form[proc].size * dbl_size);
          V (&s_link[link]);
        }
    }
}

/*******************************************************************/
/* Writer                                                          */
```

```
/* erhaelt Daten ueber message-passing und schreibt sie in den     */
/* Iterationsvektor glob_vek                                       */
/******************************************************************/
void put_to_monitor (link)
int link;
{
  int i_link, id;
  VECTOR tmp;

  tmp = (VECTOR)calloc (max_size, dbl_size);
  while (TRUE)
    {
      RecvLink(link, &id, i_size);
      RecvLink(link, tmp, d_form[id].size * dbl_size);
      request_write();
      cp_vek (d_form[id].size, tmp, glob_vek[id]);
      par (i_link = 0; i_link < _LPT; i_link ++)
        {
          if (route[i_link][id] != (-1))
          set_bit (id, send_flag[route[i_link][id]]);
        }
      release_write();
      if (_Tn == 1)
        {
          request_write();
          set_bit (id, num_flag);
          if (memcmp (num_flag, mask, felder * 4) == 0)
```

```
          {
            raise (STOPTEST);
            memfill (num_flag, 0, 1, felder * 4);
          }
        release_write();
      }
    raise (NUMERIK);
  }
}
```

Literaturverzeichnis

[AG89] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., 1989.

[BBG85] Wolfgang Bunse and Angelika Bunse-Gerstner. *Numerische lineare Algebra.* Teubner Verlag-Stuttgart, 1985.

[BEN88] Rafael Bru, Ludwig Elsner, and Michael Neumann. Models of parallel chaotic iteration methods. *Linear Algebra and its Applications*, 103:175–192, 1988.

[BHLP92] Jean-Claude Bermond, Pavol Hell, Arthur L. Liestman, and Joseph G. Peters. Broadcasting in bounded degree graphs. *SIAM Journal of descrete mathematics*, 5(1):10–24, February 1992.

[Big74] Norman Biggs. *Algebraic graph theory.* Cambridge University Press, 1974.

[BP79] Abraham Berman and Robert J. Plemmons. *Nonnegative matrices in the mathematical sciences.* Academic Press - New York, 1979.

[Brä93] Thomas Bräunl. *Parallele Programmierung.* Vieweg - Braunschweig, Wiesbaden, 1993.

[BRW93] Georg Bader, Rolf Rannacher, and Gabriel Wittum, editors. *Numerische Algorithmen auf Transputer-Systemen.* B. G. Teubner Verlag - Stuttgart, 1993.

[Car89] Graham F. Carey, editor. *Parallel Supercomputing: Methods, algorithms and applications.* John Wiley and sons, 1989.

[Col81] Lothar Collatz. *Differentialgleichungen.* Teubner Verlag Stuttgart, 1981.

[Ebe93] Heinz Ebert. *Transputer und OCCAM.* Verlag Heinz Heise, 1993.

[Els69] Ludwig Elsner. Bemerkungen zum zeilensummenkriterium. *ZAMM*, 49:7, 1969.

[ENV91] Ludwig Elsner, Michael Neumann, and Birgit Vemmer. The effect of the number of processors on the convergence of the parallel block jacobi method. *Linear Algebra and its Applications*, 154–156:311–330, 1991.

[Fly72] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on computers*, 21:948–960, 1972.

[Fro90] Andreas Frommer. *Lösung linearer Gleichungssysteme auf Parallelrechnern.* Vieweg Verlag - Braunschweig, Wiesbaden, 1990.

[FW78] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of Tenth ACM Symposium on theory of computing*, pages 114–118, 1978.

[GMB89] John L. Gustafson, Gary R. Monty, and Robert E. Benner. Development of parallel methods for a 1024-processor hypercube. *SIAM Journal of sci stat comput.*, 9(4):609–638, 1989.

[Hac93] Wolfgang Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* B. G. Teubner Stuttgart, 1993.

[Hal58] Paul R. Halmos. *Finite-dimensional vector spaces.* Litton Educational Publishing, 1958.

[Hal89] Rudolf Halin. *Graphentheorie.* Wissenschaftliche Buchgesellschaft Darmstadt, 1989.

[Han73] Per Brinch Hansen. *Operating system principles.* Prentice Hall, 1973.

[HHL88] Sandra M. Hetetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[Hoa74] Charles A.R. Hoare. Monitors: An operating system structuring concept. *Communications of the ACM*, 17(10):549–555, October 1974.

[Hoa85] Charles A. R. Hoare. *Communicating sequential processes.* Prentice-Hall, 1985.

[Inm92] Inmos limited. *The Transputer Databook*, third edition, 1992.

[Laz92] E. Lazard. Broadcasting in dma-bound bounded degree graphs. *Discrete applied mathematics*, 37/38:387–400, 1992.

[LP88] Arthur L. Liestman and Joseph G. Peters. Broadcast networks of bounded degree. *SIAM Journal of descrete mathematics*, 1(4):531–540, November 1988.

[NP87] Michael Neumann and R. J. Plemmons. Convergence of parallel multisplitting iterative methods for m-matrices. *Linear algebra and its applications*, 88/89:559–573, 1987.

[Nut92] Gary J. Nutt. *Centralized and distributed operating systems.* Prentice Hall, 1992.

[OR67] James M. Ortega and Werner C. Rheinboldt. Monotone iterations for nonlinear equations with applications to gauss-seidel methods. *SIAM Journal of numerical analysis*, 4:171–190, 1967.

[OW85] D. P. O'Leary and R. E. White. Multisplittings of matrices and parallel solutions of linear systems. *SIAM Journal of algebraic discrete methods*, 6:630–640, 1985.

[RDP92] Roland Rössler, Dietmar Dietrich, and Karl-Friedrich Penning. *Transputer - Grundlagen, Programmierung und Anwendung.* Hüthig Buch Verlag Heidelberg, 1992.

[Sen73] Eugen Seneta. *Nonnegative matrices - An introduction to theory and applications.* George Allen & Unwin Ltd., 1973.

[Sno92] C. R. Snow. *Concurrent programming.* Cambridge University Press, 1992.

[Var62] Richard S. Varga. *Matrix iterative analysis.* Prentice Hall, 1962.

[You71] David M. Young. *Iterative solution of large linear systems.* Academic Press, Inc., 1971.