Institut für Angewandte Analysis
und Stochastik

im Forschungsverbund Berlin e.V.

Iterative Verfahren für lineare Gleichungssysteme auf Distributed Memory Systemen

R. Schlundt

submitted: 2nd July 1993

Institut für Angewandte Analysis
und Stochastik
Mohrenstraße 39
D – 10117 Berlin
Germany

Preprint No. 55
Berlin 1993

1991 Mathematics Subject Classification. 65 F 10, 65 F 50, 65 Y 05, 68 Q 22.
Key words and phrases. große lineare Systeme, iterative Verfahren, Krylov–Unterraum–Methoden, GMRES–Algorithmus, QMR–Verfahren, Distributed Memory Systeme.

Herausgegeben vom
Institut für Angewandte Analysis und Stochastik
Mohrenstraße 39
D – 10117 Berlin

Fax: + 49 30 2004975
e-Mail (X.400): c=de;a=d400;p=iaas-berlin;s=preprint
e-Mail (Internet): preprint@iaas-berlin.d400.de

Inhaltsverzeichnis

Zusammenfassung

Für die Lösung großer linearer Gleichungssysteme mit schwach bzw. voll besetzten Koeffizientenmatrizen für Distributed Memory Systeme werden das GMRES-Verfahren und die QMR-Methode vorgestellt. Beide iterativen Verfahren basieren auf *Krylov*-Unterraum-Methoden. Die Weiterentwicklung dieser beiden Verfahren für Distributed Memory Systeme erfolgt in zwei Richtungen. Die erste Variante beruht auf einer Parallelisierung der *Matrix∗Vektor*-Operation. Die zweite Richtung beinhaltet die Aufspaltung der Gesamtaufgabe in disjunkte bzw. sich überlappende Teilprobleme.

1 Einführung

1.1 Problembeschreibung

Es werden iterative Methoden für die Lösung von linearen Gleichungssystemen

$$A * x = b \tag{1}$$

für Distributed Memory Systeme betrachtet, wobei $A \in \mathrm{R}^{n \times n}$ nicht symmetrisch und schwach bzw. voll besetzt ist. Von besonderem Interesse sind hierbei die *Krylov*-Unterraum-Methoden. Ausgehend von einem Startwert x_0 für das lineare Gleichungssystem (1) wird mit der Projektionsmethode eine approximative Lösung x_m aus dem affinen Unterraum $x_0 + K_m$ der Dimension m gesucht, wobei die *Galerkin*-Bedingung

$$b - A * x_m \perp L_m \tag{2}$$

erfüllt sein muß.
L_m ist hierbei ein anderer Unterraum der Dimension m. Die *Krylov*-Unterraum-Methode ist eine Methode, in der der Unterraum K_m der *Krylov*-Unterraum

$$K_m(A, r_0) = span\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\} \tag{3}$$

ist, wobei

$$r_0 = b - A * x_0 \tag{4}$$

das Anfangsresiduum darstellt. Die verschiedenen Versionen von *Krylov*-Unterraum- Methoden hängen von der Wahl der Unterräume K_m und L_m und von der Art und Weise der Vorkonditionierung ab. Die häufigste Wahl von K_m und L_m ist in [5] angegeben. Ein wichtiger Faktor für die erfolgreiche Anwendung CG-ähnlicher Methoden ist die Präkonditionierungstechnik. Ein typisches Beispiel ist die Ersetzung des Originalsystems (1) durch das äquivalente System

$$W^{-1} A * x = W^{-1} * b. \tag{5}$$

Im klassischen Fall ist die Matrix W ein unvollständiger LU-Vorkonditionierer (ILU - incomplete LU preconditioning) der Form $W = LU$, wobei L eine untere Dreiecksmatrix und U eine obere Dreiecksmatrix ist. Die Dreiecksmatrizen L und U haben dieselbe Struktur wie der untere bzw. obere Dreieckteil der Matrix A.

1.2 Parallelisierungsaspekte

Die Vielfalt der Parallelrechner-Architekturen ist immer noch sehr ausgeprägt [4]. Die rasante Entwicklung auf dem Gebiet der Parallelrechner hat in einem starken Maße auch zu einer Neuorientierung bei numerischen Algorithmen geführt [3]. Fragen der Parallelisierung sind in das Zentrum des Interesses gerückt. Um einen parallelen Algorithmus effizient auf einer Parallelrechner-Architektur implementieren zu können, benötigt der Entwickler sowohl Kenntnisse der zugrundeliegenden Architektur als auch der Software. Die derzeitigen praxisrelevanten Parallelrechner-Architekturen kann man in verschiedene Klassen einteilen:

- SIMD-Parallelrechner
  (SIMD - Single-Instruction, Multiple-Data)
- MIMD-Parallelrechner mit gemeinsamem Speicher (Shared Memory Systeme)
  (MIMD - Multiple-Instruction, Multiple-Data)
- MIMD-Parallelrechner mit verteiltem Speicher
  (Distributed Memory Systeme)

SIMD-Parallelrechner sind dadurch gekennzeichnet, daß alle Prozessoren (in jedem Maschinentakt) dieselbe Operationsart eventuell aber auf unterschiedlichen Daten ausführen. Bei MIMD-Parallelrechnern mit gemeinsamem Speicher ist jeder Prozessor durch das Verbindungsnetzwerk mit jedem Speichermodul verbunden, so daß alle Prozessoren auf jedes Datum im Speicher zugreifen können. Bei MIMD-Parallelrechnern mit verteiltem Speicher kann jeder Prozessor auf Daten nur in dem ihm zugeordneten Speichermodul zugreifen. Werden Daten aus anderen Speichermoduln benötigt, so muß es durch Übertragung der entsprechenden Daten erfolgen. Diese Übertragung, auch Kommunikation genannt, erfolgt durch Botschaftsaustausch (Message Passing). Typische Vertreter des Message Passing-Modells sind die Elemente SEND für das Senden von Daten und RECEIVE für das Empfangen von Daten.
Ein effizienter Algorithmusentwurf für Parallelrechner basiert auf einer Reihe von Prinzipien, wie z.B.

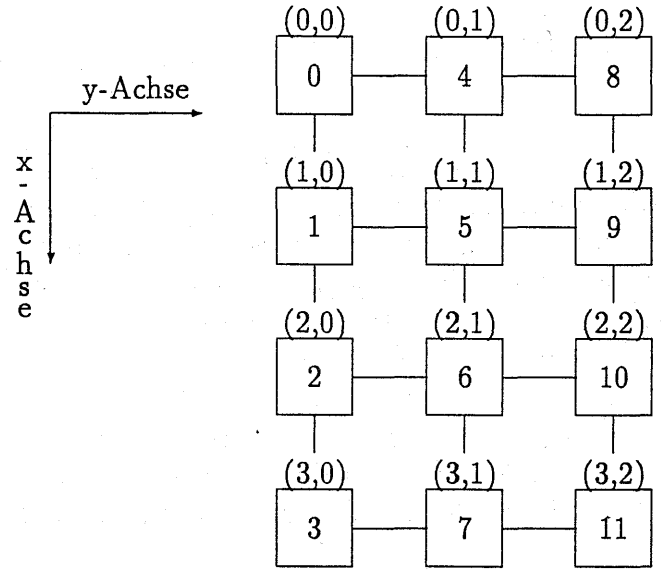
1. Kontrollschema:
   SIMD oder MIMD
2. Speicherzuordnung (physikalisch und logisch):
   lokaler oder globaler Speicher
3. Kommunikationsstruktur:
   Netzwerktopologie, Synchronisationsmechanismen, ...
4. Granularität und Skalierbarkeit:
   wenige starke oder sehr viele schwache Prozessoren
5. Lastverteilung

1.3 Speicherorganisation

Von besonderem Interesse bei der Parallelisierung von Algorithmen ist die Datenaufteilung. Da ein Distributed Memory System nur über lokale Speicher verfügt, müssen die Elemente der Matrix A auf die einzelnen Knoten (Prozessoren) verteilt werden. Die Datenverteilung sollte so vorgenommen werden, daß während des Rechenprozesses die Kommunikation so gering wie möglich ist. Für Distributed Memory Systeme sind die Prozessoren in bestimmter Weise konfiguriert. Es werden zwei unterschiedliche Konfigurationen behandelt.

1.3.1 Das zweidimensionale Prozessorgitter

Das zweidimensionale Prozessorgitter (d_x, d_y) sei auf die gedrehte (x, y)-Ebene abgebildet. Als Illustrationsbeispiel wird ein $(4, 3)$-Prozessorgitter gewählt.



Die Matrix A wird blockweise auf das zweidimensionale Prozessorgitter (d_x, d_y) aufgeteilt. Dazu wird die spaltenweise Speicherung mit Diagonalbehandlung [5] gewählt. Die Matrix A sei in einem zweidimensionalen Feld AC vom Typ (n, n_z) abgespeichert. Die Größe n_z gibt für schwach besetzte Matrizen die maximale Anzahl der Nichtnullelemente in einer Zeile von A an. Für voll besetzte Matrizen ist n_z gleich n. Die Teilblöcke sollten wegen der Auslastungsbalance der Prozessoren möglichst gleiche Größe besitzen. Mit den folgenden Abschätzungen kann man die Blockgröße (n^r, n_z^r) und das notwendige (reduzierte) Prozessorgitter (d_x^r, d_y^r) berechnen.

$$
\begin{aligned}
n^r &= int(\frac{n + d_x - 1}{d_x}) \\
n_z^r &= int(\frac{n_z + d_y - 1}{d_y}) \\
d_x^r &= int(\frac{n + n^r - 1}{n^r}) \\
d_y^r &= int(\frac{n_z + n_z^r - 1}{n_z^r})
\end{aligned}
$$

Im folgenden wird die Matrix A mit der spaltenweisen Speicherung mit Diagonalbehandlung, d.h. mit n_z, AC und KA identifiziert. Die unteren Randblöcke von A entlang der y-Richtung haben $n - n^r(d_x^r - 1)$ Zeilen und die rechten Randblöcke entlang der x-Richtung $n_z - n_z^r(d_y^r - 1)$ Spalten. Die folgenden Darstellungen verdeutlichen den Zusammenhang zwischen Prozessorgitter, Prozessornummer und Aufteilung der Matrix A auf die einzelnen

Prozessoren. Dazu wird die Beispielmatrix

$$A = \begin{pmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 & 0 & 0 & 0 & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & 0 & 0 & 0 \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} & a_{59} & 0 \\ 0 & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} & a_{69} & a_{610} \\ 0 & 0 & 0 & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} & a_{79} & a_{710} \\ 0 & 0 & 0 & 0 & 0 & a_{86} & a_{87} & a_{88} & a_{89} & a_{810} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{98} & a_{99} & a_{910} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{1010} \end{pmatrix}$$

mit $n = 10$ betrachtet.
Hieraus ergibt sich für die spaltenweise Speicherung mit Diagonalbehandlung $n_z = 9$,

$$\mathrm{AC} = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{22} & a_{21} & a_{23} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{33} & a_{31} & a_{32} & a_{34} & a_{35} & 0 & 0 & 0 & 0 \\ a_{44} & a_{41} & a_{42} & a_{43} & a_{45} & a_{46} & a_{47} & 0 & 0 \\ a_{55} & a_{51} & a_{52} & a_{53} & a_{54} & a_{56} & a_{57} & a_{58} & a_{59} \\ a_{66} & a_{62} & a_{63} & a_{64} & a_{65} & a_{67} & a_{68} & a_{69} & a_{610} \\ a_{77} & a_{74} & a_{75} & a_{76} & a_{78} & a_{79} & a_{710} & 0 & 0 \\ a_{88} & a_{86} & a_{87} & a_{89} & a_{810} & 0 & 0 & 0 & 0 \\ a_{99} & a_{98} & a_{910} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{1010} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

und

$$\mathrm{KA} = \begin{bmatrix} 1 & * & * & * & * & * & * & * & * \\ 2 & 1 & 3 & * & * & * & * & * & * \\ 3 & 1 & 2 & 4 & 5 & * & * & * & * \\ 4 & 1 & 2 & 3 & 5 & 6 & 7 & * & * \\ 5 & 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 \\ 6 & 2 & 3 & 4 & 5 & 7 & 8 & 9 & 10 \\ 7 & 4 & 5 & 6 & 8 & 9 & 10 & * & * \\ 8 & 6 & 7 & 9 & 10 & * & * & * & * \\ 9 & 8 & 10 & * & * & * & * & * & * \\ 10 & * & * & * & * & * & * & * & * \end{bmatrix}$$

mit $1 \leq * \leq n$.
Für das betrachtete Prozessorgitterbeispiel mit $d_x = 4$ und $d_y = 3$ ergeben sich dann $n^r = 3$, $n_z^r = 3$, $d_x^r = 4$ und $d_y^r = 3$. Hiermit erhält man dann die folgende Datenverteilung:

- für AC

	(0,0)			(0,1)			(0,2)	
a_{11}	0	0	0	0	0	0	0	0
a_{22}	a_{21}	a_{23}	0	0	0	0	0	0
a_{33}	a_{31}	a_{32}	a_{34}	a_{35}	0	0	0	0
	(1,0)			(1,1)			1,2)	
a_{44}	a_{41}	a_{42}	a_{43}	a_{45}	a_{46}	a_{47}	0	0
a_{55}	a_{51}	a_{52}	a_{53}	a_{54}	a_{56}	a_{57}	a_{58}	a_{59}
a_{66}	a_{62}	a_{63}	a_{64}	a_{65}	a_{67}	a_{68}	a_{69}	a_{610}
	(2,0)			(2,1)			(2,2)	
a_{77}	a_{74}	a_{75}	a_{76}	a_{78}	a_{79}	a_{710}	0	0
a_{88}	a_{86}	a_{87}	a_{89}	a_{810}	0	0	0	0
a_{99}	a_{98}	a_{910}	0	0	0	0	0	0
	(3,0)			(3,1)			(3,2)	
a_{1010}	0	0	0	0	0	0	0	0

- für KA

	(0,0)			(0,1)			(0,2)	
1	*	*	*	*	*	*	*	*
2	1	3	*	*	*	*	*	*
3	1	2	4	5	*	*	*	*
	(1,0)			(1,1)			1,2)	
4	1	2	3	5	6	7	*	*
5	1	2	3	4	6	7	8	9
6	2	3	4	5	7	8	9	10
	(2,0)			(2,1)			(2,2)	
7	4	5	6	8	9	10	*	*
8	6	7	9	10	*	*	*	*
9	8	10	*	*	*	*	*	*
	(3,0)			(3,1)			(3,2)	
10	*	*	*	*	*	*	*	*

Für einen Prozessor mit den Komponenten (i_x, i_y), wobei $i_x = 0, \ldots, d_x - 1$ und $i_y = 0, \ldots, d_y - 1$, läßt sich die Identifikationsnummer durch $d_x \cdot i_y + i_x$ bestimmen. Für die Aufteilung der Matrix A gilt:

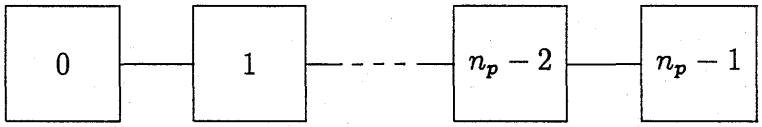
$$A = \begin{pmatrix} A_{0,0} & A_{0,1} & \cdots & A_{0,d_y^r-1} \\ A_{1,0} & A_{1,1} & \cdots & A_{1,d_y^r-1} \\ \vdots & \vdots & & \vdots \\ A_{d_x^r-1,0} & A_{d_x^r-1,1} & \cdots & A_{d_x^r-1,d_y^r-1} \end{pmatrix}$$

Die Rechteckmatrix $A_{i_x,i_y}, i_x = 0,\ldots,d_x-1, i_y = 0,\ldots d_y-1$, ist dem Prozessor mit den Komponenten (i_x, i_y) eindeutig zugeordnet.

$(0,0)$ $A_{0,0}$	$(0,1)$ $A_{0,1}$	$\cdots(0,i_y)\cdots$ $\cdots A_{0,i_y}\cdots$	$(0,d_y^r-1)$ A_{0,d_y^r-1}	$\cdots(0,d_y-1)$ $\cdots$
$(1,0)$ $A_{1,0}$	$(1,1)$ $A_{1,1}$	$\cdots(1,i_y)\cdots$ $\cdots A_{1,i_y}\cdots$	$(1,d_y^r-1)$ A_{1,d_y^r-1}	$\cdots(1,d_y-1)$ $\cdots$
$\vdots$ $(i_x,0)$ $A_{i_x,0}$ $\vdots$	$\vdots$ $(i_x,1)$ $A_{i_x,1}$ $\vdots$	$\vdots$ $\cdots(i_x,i_y)\cdots$ $\cdots A_{i_x,i_y}\cdots$ $\vdots$	$\vdots$ (i_x,d_y^r-1) A_{i_x,d_y^r-1} $\vdots$	$\vdots$ $\cdots(i_x,d_y-1)$ $\cdots$ $\vdots$
$(d_x^r-1,0)$ $A_{d_x^r-1,0}$	$(d_x^r-1,1)$ $A_{d_x^r-1,1}$	$\cdots(d_x^r-1,i_y)\cdots$ $\cdots A_{d_x^r-1,i_y}\cdots$	(d_x^r-1,d_y^r-1) $A_{d_x^r-1,d_y^r-1}$	$\cdots(d_x^r-1,d_y-1)$ $\cdots$
$\vdots$ $(d_x-1,0)$ $\cdots$	$\vdots$ $(d_x-1,1)$ $\cdots$	$\vdots$ $\cdots(d_x-1,i_y)\cdots$ $\cdots$	$\vdots$ (d_x-1,d_y^r-1) $\cdots$	$\vdots$ $\cdots(d_x-1,d_y-1)$ $\cdots$

1.3.2 Die Prozessorpipeline

Die Prozessorpipeline der Länge n_p genüge der Darstellung



mit n_p Prozessoren. Die Matrix A wird zeilenweise mit Vorsortierung und Diagonalbehandlung gespeichert [5]. Dazu werden die Nichtnullelemente von A durch die Felder ACP, KAP, IAP und JAP charakterisiert. Als Beispiel wird die Matrix aus 1.3.1 gewählt. Ausgehend von den Matrizen AC und KA werden die zweidimensionalen Matrizen AC' und KA' gleichen Typs gebildet. Sie entstehen dadurch, daß die Zeilen in den Feldern AC bzw. KA aufsteigend nach der Anzahl der Nichtnullelemente sortiert werden.

$$\text{AC'} = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{1010} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{22} & a_{21} & a_{23} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{99} & a_{98} & a_{910} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{33} & a_{31} & a_{32} & a_{34} & a_{35} & 0 & 0 & 0 & 0 \\ a_{88} & a_{86} & a_{87} & a_{89} & a_{810} & 0 & 0 & 0 & 0 \\ a_{44} & a_{41} & a_{42} & a_{43} & a_{45} & a_{46} & a_{47} & 0 & 0 \\ a_{77} & a_{74} & a_{75} & a_{76} & a_{78} & a_{79} & a_{710} & 0 & 0 \\ a_{55} & a_{51} & a_{52} & a_{53} & a_{54} & a_{56} & a_{57} & a_{58} & a_{59} \\ a_{66} & a_{62} & a_{63} & a_{64} & a_{65} & a_{67} & a_{68} & a_{69} & a_{610} \end{bmatrix}$$

$$\text{KA'} = \begin{bmatrix} 1 & * & * & * & * & * & * & * & * \\ 10 & * & * & * & * & * & * & * & * \\ 2 & 1 & 3 & * & * & * & * & * & * \\ 9 & 8 & 10 & * & * & * & * & * & * \\ 3 & 1 & 2 & 4 & 5 & * & * & * & * \\ 8 & 6 & 7 & 9 & 10 & * & * & * & * \\ 4 & 1 & 2 & 3 & 5 & 6 & 7 & * & * \\ 7 & 4 & 5 & 6 & 8 & 9 & 10 & * & * \\ 5 & 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 \\ 6 & 2 & 3 & 4 & 5 & 7 & 8 & 9 & 10 \end{bmatrix}$$

Hieraus werden Blöcke gleicher Spaltenbreite gebildet. Die Elemente eines jeden Blockes werden spaltenweise abgespeichert. Damit ergibt sich für die lineare Speicherung $n_b = 5$,

$$\begin{aligned} \text{ACP} &= [a_{11}, a_{1010}, a_{22}, a_{99}, a_{21}, a_{98}, a_{23}, a_{910}, a_{33}, a_{88}, a_{31}, a_{86}, a_{32}, a_{87}, a_{34}, a_{89}, a_{35}, \\ &\quad a_{810}, a_{44}, a_{77}, a_{41}, a_{74}, a_{42}, a_{75}, a_{43}, a_{76}, a_{45}, a_{78}, a_{46}, a_{79}, a_{47}, a_{710}, a_{55}, a_{66}, \\ &\quad a_{51}, a_{62}, a_{52}, a_{63}, a_{53}, a_{64}, a_{54}, a_{65}, a_{56}, a_{67}, a_{57}, a_{68}, a_{58}, a_{69}, a_{59}, a_{610}] , \end{aligned}$$

$$\begin{aligned} \text{KAP} &= [1,10,2,9,1,8,3,10,3,8,1,6,2,7,4,9,5,10,4,7,1,4,2,5,3, \\ &\quad 6,5,8,6,9,7,10,5,6,1,2,2,3,3,4,4,5,6,7,7,8,8,9,9,10] , \end{aligned}$$

$$\text{IAP} = [1,3,9,19,33,51] \qquad \text{und}$$

$$\text{JAP} = [1,3,5,7,9] .$$

Die Größe n_b gibt die Anzahl der Blöcke an. Das *INTEGER*-Feld IAP enthält die Startpositionen der einzelnen Blöcke, wobei die letzte Komponente von IAP die um Eins erhöhte Anzahl der Nichtnullelemente von A angibt. Schließlich sind in dem *INTEGER*-Feld JAP die Anzahl der Spalten für jeden Block enthalten.
Die Aufteilung der Matrix A auf die einzelnen Prozessoren bei dieser Speicherungsform ist wesentlich komplizierter als die unter Punkt 1.3.1 angegebene Aufteilung. Für jeden Prozessor müssen die Felder ACP, KAP, IAP und JAP erzeugt werden. Sie werden mit ACPP, KAPP, IAPP und JAPP bezeichnet. Hinzu kommt noch die Berechnung von n_{b_p}, d.h. die Anzahl der Blöcke für jeden Prozessor. Es muß gesichert sein, daß eine beliebige Zeile von Nichtnullelementen der Matrix A vollständig auf einem Prozessor enthalten ist. An der Beispielmatrix aus Punkt 1.3.1 soll dies verdeutlicht werden. Es wird eine Prozessorpipeline mit $n_p = 12$ ($n_p = d_x * d_y, d_x = 4, d_y = 3$) Prozessoren betrachtet. Die notwendige (reduzierte) Prozessorpipeline besteht aus $n_p^r = 7$ Prozessoren. Hiermit erhält man die folgende Datenverteilung:

- für Prozessor 0

$$\begin{aligned} \text{ACPP} &= [a_{11}, a_{1010}, a_{22}, a_{99}, a_{21}, a_{98}, a_{23}, a_{910}] \\ \text{KAPP} &= [1,10,2,9,1,8,3,10] \\ \text{IAPP} &= [1,3,9] \\ \text{JAPP} &= [1,3] \end{aligned}$$

- für Prozessor 1

$$\begin{aligned} \text{ACPP} &= [a_{33}, a_{31}, a_{32}, a_{34}, a_{35}] \\ \text{KAPP} &= [3, 1, 2, 4, 5] \\ \text{IAPP} &= [1, 6] \\ \text{JAPP} &= [5] \end{aligned}$$

- für Prozessor 2

$$\begin{aligned} \text{ACPP} &= [a_{88}, a_{86}, a_{87}, a_{89}, a_{810}] \\ \text{KAPP} &= [8, 6, 7, 9, 10] \\ \text{IAPP} &= [1, 6] \\ \text{JAPP} &= [5] \end{aligned}$$

- für Prozessor 3

$$\begin{aligned} \text{ACPP} &= [a_{44}, a_{41}, a_{42}, a_{43}, a_{45}, a_{46}, a_{47}] \\ \text{KAPP} &= [4, 1, 2, 3, 5, 6, 7] \\ \text{IAPP} &= [1, 8] \\ \text{JAPP} &= [7] \end{aligned}$$

- für Prozessor 4

$$\begin{aligned} \text{ACPP} &= [a_{77}, a_{74}, a_{75}, a_{76}, a_{78}, a_{79}, a_{710}] \\ \text{KAPP} &= [7, 4, 5, 6, 8, 9, 10] \\ \text{IAPP} &= [1, 8] \\ \text{JAPP} &= [7] \end{aligned}$$

- für Prozessor 5

$$\begin{aligned} \text{ACPP} &= [a_{55}, a_{51}, a_{52}, a_{53}, a_{54}, a_{56}, a_{57}, a_{58}, a_{59}] \\ \text{KAPP} &= [5, 1, 2, 3, 4, 6, 7, 8, 9] \\ \text{IAPP} &= [1, 10] \\ \text{JAPP} &= [9] \end{aligned}$$

- für Prozessor 6

$$\begin{aligned} \text{ACPP} &= [a_{66}, a_{62}, a_{63}, a_{64}, a_{65}, a_{67}, a_{68}, a_{69}, a_{610}] \\ \text{KAPP} &= [6, 2, 3, 4, 5, 7, 8, 9, 10] \\ \text{IAPP} &= [1, 10] \\ \text{JAPP} &= [9] \end{aligned}$$

Ein Vergleich beider Speicherungstechniken bzgl. der *Matrix∗ Vektor*-Operation ist in den Tabellen 1 und 2 enthalten.

2 Beschreibung der Verfahren

Ausgangspunkt ist das lineare Gleichungssystem (1) mit eventueller Vorkonditionierung (5). Ausgehend vom Startwert x_0 wird eine approximative Lösung x_m für (1) gesucht. Mit $x = x_0 + z$ geht (1) in das äquivalente System

$$A * z = r_0 \tag{6}$$

über. Sei K_m der *Krylov*-Unterraum (3), dann erhält man

$$x_m = x_0 + z_m \qquad \text{mit} \qquad z_m \in K_m, \tag{7}$$

so daß $(b - Ax_m) \perp K_m$ oder äquivalent $(r_0 - Az_m) \perp K_m$ gilt. Für beide Methoden muß das Minimumproblem

$$\|b - Ax_m\|_2 = \min_{x \in x_0 + K_m} \|b - Ax\|_2 \quad = \quad \min_{z \in K_m} \|r_0 - Az\|_2 \tag{8}$$

gelöst werden. Eine ausführliche Beschreibung des GMRES-Algorithmus und der QMR-Methode ist in [5] enthalten.

2.1 Der GMRES-Algorithmus

Gesucht wird eine l_2-Orthonormalbasis $V_m = \{v_1, \ldots, v_m\}$ von K_m. Die l_2-Orthonormalbasis kann sowohl durch die *Gram-Schmidt*-Orthogonalisierung als auch durch die *Householder*-Orthogonalisierung erzeugt werden. Aus der Konstruktion der Basisvektoren $v_i, i = 1, \ldots, m$, folgt

$$AV_m = V_{m+1}\bar{H}_m, \tag{9}$$

wobei $\bar{H}_m$ eine $(m+1) \times m$-*Hessenberg*-Matrix mit $\bar{H}_m = (h_{ij})$ für $1 \leq i \leq j+1$ und $1 \leq j \leq m$ ist. Mit $\bar{H}_m$ gilt für die Normabschätzung des Residuums

$$\|b - Ax_m\|_2 = \|\|r_0\|_2 e_1 - \bar{H}_m y_m\|_2, \tag{10}$$

wobei y_m die Lösung des Minimumproblems

$$\min_{y \in \mathrm{R}^m} \|\|r_0\|_2 e_1 - \bar{H}_m y\|_2 \tag{11}$$

ist. Mit y_m erhält man $z_m = V_m y_m$ und damit $x_m = x_0 + z_m$. Die *Hessenberg*-Matrix $\bar{H}_m$ wird durch QR-Faktorisierung zerlegt. Die Zerlegung wird mittels *Givens*-Rotationen durchgeführt. Mit den Nichtdiagonalelementen $s_i, i = 1, \ldots, m$, in den Drehungsmatrizen erhält man für das Residuum die Abschätzung

$$\|r_m\|_2 = \|r_0\|_2 |\prod_{i=1}^{m} s_i|. \tag{12}$$

2.2 Das QMR-Verfahren

Die *Lanczos*-Methode im klassischen Sinne bedeutet die Reduktion einer nichtsymmetrischen allgemeinen Matrix $A \in \mathrm{R}^{n \times n}$ auf Tridiagonalform. Der Algorithmus startet mit Vektoren $v_1 \in \mathrm{R}^n$ und $w_1 \in \mathrm{R}^n$, wobei $v_1 \neq 0$ und $w_1 \neq 0$. Dann werden Basen $\{v_i\}$ und $\{w_i\}$ für die *Krylov*-Unterräume $K_n(A, v_1)$ bzw. $K_n(A^T, w_1)$ mit der Biorthogonalitätsbedingung

$$(w_j, v_k) = \begin{cases} 1 & für \quad k = j \\ 0 & sonst \end{cases} \tag{13}$$

konstruiert. Ein möglicher irregulärer Abbruch der *Lanczos*-Methode ist $(w_m, v_m) = 0$, wobei weder $v_m = 0$ noch $w_m = 0$. Die Biorthogonalitätsbedingung (13) kann nicht erfüllt werden. Es kann aber sein, daß (13) für ein größeres m wieder erfüllt ist. Ein Algorithmus, der irgendwie auf ein solches Paar von *Lanczos*-Vektoren vorausschaut, wird *look-ahead Lanczos*-Algorithmus genannt. Die Grundidee ist die Abschwächung der Biorthogonalitätsbedingung (13).
Die grundlegende Idee des QMR-Verfahrens ist die Erzeugung der Matrix V_m mittels kurzer Rekursionen mit Hilfe des *look-ahead Lanczos*-Prozesses, wobei dann aber die Minimierungsbedingung (8) ein wenig abgeschwächt wird. Aus $x_m = x_0 + V_m y_m$ folgt $r_m = r_0 - AV_m y_m$ und damit

$$r_m = v_1 \|r_0\|_2 - V_{m+1} \bar{H}_m y_m = V_{m+1}(\|r_0\|_2 e_1 - \bar{H}_m y_m). \tag{14}$$

Da V_{m+1} nicht unitär ist, wird der Koeffizientenvektor y minimiert, d.h. y_m ist die Lösung des Minimumproblems

$$\|\|r_0\|_2 e_1 - \bar{H}_m y_m\|_2 \quad = \quad \min_{y \in \mathrm{R}^m} \quad \|\|r_0\|_2 e_1 - \bar{H}_m y\|_2. \tag{15}$$

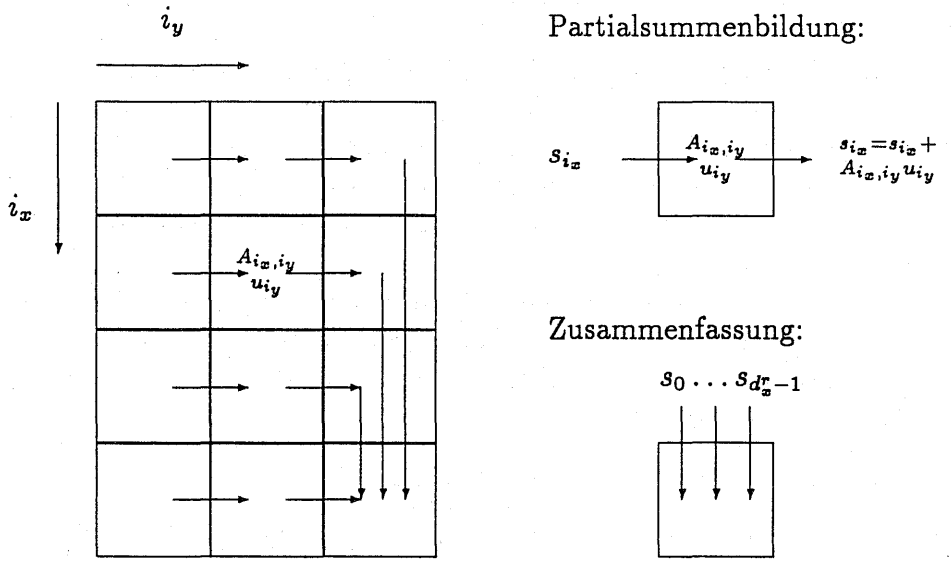
3 Parallele Varianten der Verfahren

Die Weiterentwicklung dieser beiden Verfahren für Distributed Memory Systeme erfolgt in zwei Richtungen. Die erste Variante beruht auf einer Parallelisierung der $Matrix*Vektor$-Operation. Hierzu wird die Matrix A blockweise auf ein zweidimensionales Prozessorgitter (d_x, d_y) aufgeteilt (siehe 1.3.1 mit den Feldern AC und KA) bzw. linear auf eine Prozessorpipeline abgebildet (siehe 1.3.2 mit den Feldern ACP, KAP, IAP und JAP). Die zweite Richtung beinhaltet die Aufspaltung der Gesamtaufgabe in disjunkte bzw. sich überlappende niederdimensionale Teilprobleme. Da von der algebraischen Gleichung (1) ausgegangen wird, beruht die Darstellung der Gebietszerlegungsmethode auf einer Zerlegung des Vektors x.

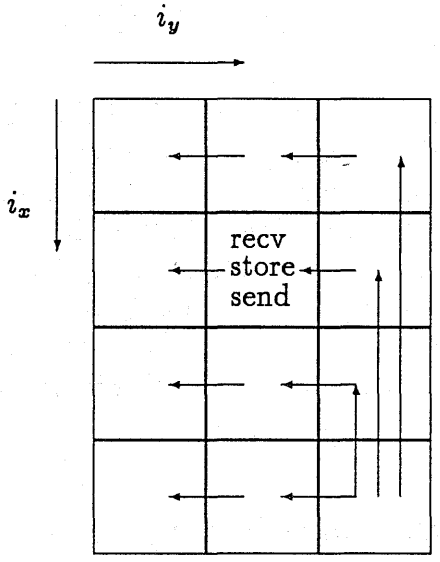
3.1 Matrix*Vektor-Parallelisierung

Ausgehend von den zwei verschiedenen Speicherungstechniken (siehe 1.3) werden auch unterschiedliche Realisierungen der *Matrix* $*$ *Vektor*-Parallelisierung vorgenommen. Zu-

nächst sei die Matrix A auf ein zweidimensionales Prozessorgitter abgebildet (siehe 1.3.1). Die Bildung des *Matrix* Vektor*-Produkts $A \cdot u = s$ mit $u \in \mathrm{R}^n$ und $s \in \mathrm{R}^n$ erfolgt dadurch, daß auf jedem Prozessor mit den Komponenten (i_x, i_y) Teilergebnisse $s_{i_x} = A_{i_x,i_y} \cdot u_{i_y}$ erzeugt werden, wobei $u = (u_0, \ldots, u_{d_y^r-1})^T$ mit $u_{i_y} \in \mathrm{R}^{n_z^r}$ für $i_y = 0, \ldots, d_y^r - 1$ und $s = (s_0, \ldots, s_{d_x^r-1})^T$ mit $s_{i_x} \in \mathrm{R}^{n^r}$ für $i_x = 0, \ldots, d_x^r - 1$. Die einzelnen Teilergebnisse werden mittels Kommunikation (Message Passing) bzgl. des Prozessorgitters (d_x^r, d_y^r) zeilenweise aufsummiert und zusammengefaßt.



Das Gesamtergebnis wird ebenfalls durch Kommunikation auf alle Prozessoren des Prozessorgitters (d_x^r, d_y^r) verteilt. Hierzu wird das Ergebnis der *Matrix * Vektor*-Operation s zuerst in der letzten Spalte des (reduzierten) Prozessorgitters (d_x^r, d_y^r) verschickt und dann parallel entlang der y-Richtung.

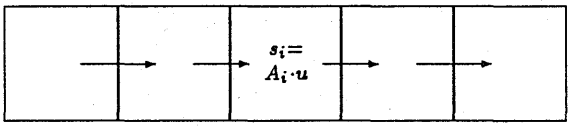


Bei der zweiten Speicherungstechnik (siehe 1.3.2) ist die Matrix A linear abgebildet auf

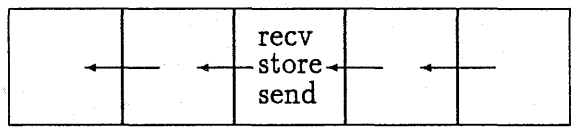
eine Prozessorpipeline. Die Bildung des *Matrix* $*$ *Vektor*-Produkts $A \cdot u = s$ erfolgt dadurch, daß auf jedem Prozessor unterschiedliche Komponenten des Ergebnisvektors s berechnet werden. Hierzu sei die Matrix A in der Gestalt

$$A = \begin{pmatrix} A_0 \\ \vdots \\ A_i \\ \vdots \\ A_{n_p^r-1} \end{pmatrix}$$

dargestellt. Die Größen $A_i, i = 0, \ldots, n_p^r - 1$, seien durch die Felder ACPP, KAPP, IAPP und JAPP repräsentiert. Die einzelnen Komponenten werden mittels Kommunikation bzgl. der Prozessorpipeline mit n_p^r Prozessoren zusammengefaßt.



Das Gesamtergebnis wird ebenfalls durch Kommunikation auf alle Prozessoren der Prozessorpipeline verteilt. Hierzu wird das Ergebnis der *Matrix*$*$*Vektor*-Operation s ausgehend vom Prozessor mit der Identifikation $n_p^r - 1$ an alle anderen Prozessoren der Pipeline verschickt.



3.2 Gebietszerlegungsmethode

Die Aufspaltung der Gesamtaufgabe (1) in disjunkte bzw. sich überlappende Teilprobleme beruht auf einer Zerlegung des Vektors x. Die einzelnen Lösungen der Teilprobleme lösen noch nicht das Gesamtproblem. Deshalb muß eine geeignete Kopplung der Einzelprobleme hergestellt werden. Die *additive Schwarz*-Iteration mit geeigneter Dämpfung bietet sich auf Grund ihrer Parallelität als Kopplung an. Eine ausführliche Beschreibung der *additiven Schwarz*-Iteration ist in [1] zu finden. Die Berechnung des optimalen Dämpfungsparameters ist eine wichtige Teilaufgabe für die Entwicklung eines effektiven GMRES- bzw. QMR-Algorithmus auf Distributed Memory Systemen.
Sei R^n der lineare Raum, der die Lösung x von (1) enthält. Die Teilaufgaben, die mit $i = 1, \ldots, n_p$ indiziert seien, entsprechen niederdimensionalen Aufgaben, die durch Vektoren $x^i \in \mathrm{R}^{n_i}, x = (x^1, \ldots, x^{n_p})^T$, repräsentiert sind. Die Lösung x wird aus den Einzellösungen $x^i, i = 1, \ldots, n_p$, zusammengesetzt. Dazu werden lineare und injektive Fortsetzungen (Prolongationen)

$$p_i : \mathrm{R}^{n_i} \longrightarrow \mathrm{R}^n \tag{16}$$

gewählt. Die Gesamtlösung $x = A^{-1}b$ wird in der Form

$$x = \sum_{i=1}^{n_p} p_i x^i \tag{17}$$

gesucht. Damit dies möglich ist, muß $\sum_{i=1}^{n_p} Bild(p_i) = \mathrm{R}^n$ gelten. Die Prolongation p_i aus (16) wird durch eine Rechteckmatrix repräsentiert. Für p_i^T gilt:

$$p_i^T : \mathrm{R}^n \longrightarrow \mathrm{R}^{n_i}. \tag{18}$$

Zu jedem $i, i = 1, \ldots, n_p$, seien die quadratischen Matrizen A_{ii} durch

$$A_{ii} = p_i^T A p_i \qquad \text{mit} \qquad A_{ii} \in \mathrm{R}^{n_i \times n_i} \tag{19}$$

definiert. Die niederdimensionalen Teilprobleme sind Gleichungen der Form

$$A_{ii} * c^i = d^i \tag{20}$$

mit $c^i \in \mathrm{R}^{n_i}$ und $d^i \in \mathrm{R}^{n_i}$.
Es gilt z.B.

$$\begin{aligned} c^i &\equiv x^i \\ d^i &= b^i - \sum_{\substack{j=1 \\ j \neq i}}^{n_p} A_{ij} x^j \qquad \text{mit} \\ b &= (b^1, \ldots, b^{n_p})^T, b^i \in \mathrm{R}^{n_i} \quad \text{und} \end{aligned}$$

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1n_p} \\ A_{21} & A_{22} & \ldots & A_{2n_p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n_p1} & A_{n_p2} & \ldots & A_{n_pn_p} \end{pmatrix}.$$

Es sei angenommen, daß Probleme der Form (20) exakt gelöst werden können. Assoziiert mit den Prolongationen p_i sind die Restriktionen

$$r_i = A_{ii}^{-1} p_i^T : \mathrm{R}^n \longrightarrow \mathrm{R}^{n_i} \tag{21}$$

und die Projektionen

$$P_i = p_i r_i A : \mathrm{R}^n \longrightarrow \mathrm{R}^n. \tag{22}$$

Eine erste Klassifizierung (der Prolongationen) ergibt die Fallunterscheidung

$$\sum_{i=1}^{n_p} n_i = n \qquad \text{(disjunkte Teilgebiete)} \tag{23}$$

$$\sum_{i=1}^{n_p} n_i > n \qquad \text{(überlappende Teilgebiete).} \tag{24}$$

Für (23) hat jedes x eine eindeutige Zerlegung (17), während für (24) mehrere Darstellungen (17) möglich sind. Für (23) hat die Matrix p_i die Gestalt

$$p_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ I \\ 0 \\ \vdots \\ 0 \end{bmatrix} \} \text{Block zum Index } i, \quad p_i^T = [0 \dots 0\, I\, 0 \dots 0],$$

wobei I die Einheitsmatrix vom Typ (n_i, n_i) ist.
Zu den Projektionen $P_i, i = 1, \dots, n_p$, gehört der Iterationsschritt

$$\Phi_i(x, b) = x - p_i r_i (Ax - b). \tag{25}$$

Ein Iterationsverfahren für (1) ist eine (lineare oder nichtlineare) Abbildung

$$\Phi : \mathrm{R}^n \times \mathrm{R}^n \longrightarrow \mathrm{R}^n$$

mit

$$x_{m+1} = \Phi(x_m, b),$$

wobei x_0 ein vorgegebener Startwert ist. Falls Φ linear ist, gibt es Matrizen M und N, so daß

$$\Phi(x, b) = M \cdot x + N \cdot b$$

gilt. Die Matrix M heißt hierbei Iterationsmatrix der Iteration Φ. Die zu (25) gehörende Iterationsmatrix ist $M_i = I - P_i$, denn

$$\begin{aligned} \Phi_i(x, b) &= x - p_i r_i Ax - p_i r_i b \\ &= x - P_i x - p_i r_i b \\ &= (I - P_i)x - p_i r_i b. \end{aligned}$$

Die Matrix N_i der zweiten Normalform von Φ_i ist $N_i = p_i r_i$. Als *additive Schwarz*-Iteration mit dem Dämpfungsparameter θ erhält man

$$\Phi_\theta(x, b) = x - \theta \sum_{i=1}^{n_p} p_i r_i (Ax - b). \tag{26}$$

Die Iterationsmatrix von Φ_θ lautet dann

$$M_\theta = I - \theta(\sum_{i=1}^{n_p} p_i r_i)A = I - \theta \sum_{i=1}^{n_p} P_i. \tag{27}$$

Die Matrix der zweiten Normalform von Φ_θ ist

$$N_\theta = \theta \sum_{i=1}^{n_p} p_i r_i. \tag{28}$$

Die *additive Schwarz*-Iteration ist besonders für Parallelrechner interessant, da sie eine natürliche Parallelität aufweist. Dazu werden die einzelnen Berechnungsschritte ausführlich erklärt.

1. Berechnung der partitionierten Defekte

$$d^i = p_i^T(Ax_m - b) \tag{29}$$

für alle $i = 1, \ldots, n_p$.
Dies bedeutet:

$$d^i = \begin{pmatrix} 0 \\ \vdots \\ I \\ \vdots \\ 0 \end{pmatrix}^T \left\{ \begin{pmatrix} A_{11} & \cdots & A_{1i} & \cdots & A_{1n_p} \\ \vdots & & \vdots & & \vdots \\ A_{i1} & \cdots & A_{ii} & \cdots & A_{in_p} \\ \vdots & & \vdots & & \vdots \\ A_{n_p1} & \cdots & A_{n_pi} & \cdots & A_{n_pn_p} \end{pmatrix} \begin{pmatrix} x_m^1 \\ \vdots \\ x_m^i \\ \vdots \\ x_m^{n_p} \end{pmatrix} - \begin{pmatrix} b^1 \\ \vdots \\ b^i \\ \vdots \\ b^{n_p} \end{pmatrix} \right\}$$

Hieraus folgt

$$d^i = \sum_{j=1}^{n_p} A_{ij}x_m^j - b^i. \tag{30}$$

Die nächsten Rechenschritte sind völlig unabhängig voneinander und können auf verschiedenen Prozessoren ohne Kommunikation untereinander durchgeführt werden.

(a) Bildung von $A_{ii}^{-1}d^i = A_{ii}^{-1}p_i^T(Ax_m - b) = r_i(Ax_m - b)$.
Dies bedeutet die Lösung des niederdimensionalen Gleichungssystems (31). Aus (30) folgt

$$A_{ii}A_{ii}^{-1}d^i - A_{ii}x_m^i = \sum_{\substack{j=1 \\ j\neq i}}^{n_p} A_{ij}x_m^j - b^i.$$

Mit $-x_{m+1}^i = A_{ii}^{-1}d^i - x_m^i$ und mit $A_{ii}^{-1}d^i = x_m^i - x_{m+1}^i = \delta x_m^i$ erhält man das Gleichungssystem

$$A_{ii}\delta x_m^i = \sum_{j=1}^{n_p} A_{ij}x_m^j - b^i. \tag{31}$$

(b) Bildung von $p_iA_{ii}^{-1}d^i = p_ir_i(Ax_m - b)$.
Die Lösung $\delta x_m^i \in \mathrm{R}^{n_i}$ wird auf den Raum R^n transformiert, d.h.

$$\delta x_m^i \longrightarrow \begin{pmatrix} 0 \\ \vdots \\ \delta x_m^i \\ \vdots \\ 0 \end{pmatrix}.$$

2. Durchführung des Korrekturschrittes

$$x_{m+1} = x_m - \theta \sum_{i=1}^{n_p} \delta x_m^i. \tag{32}$$

Der Schritt (32) kann parallel durchgeführt werden,wenn (23) gilt. In diesem Fall vereinfacht sich dann die Korrektur auf jedem Prozessor zu $x_{m+1}^i = x_m^i - \theta\delta x_m^i$. Im überlappenden Fall (24) ist weitere Kommunikation notwendig.

3. Verteilung der Größen x_{m+1}^i für alle $i = 1, \ldots, n_p$ über die lokalen Speicher der Prozessoren.

Für positiv definite Matrizen stimmt nach [1] die *additive Schwarz*-Iteration mit der *gedämpften Block-Jacobi*-Iteration überein und konvergiert für hinreichend kleines $\theta > 0$ (hinreichend ist stets $\theta < 2/n_p$). D.h., die *additive Schwarz*-Iteration Φ_θ kann bei geeigneter Dämpfung als konvergente Iteration verwendet werden. Ausgehend vom Korrekturschritt (32) für die Iteration (26) erhält man folgende Abschätzungen. Sei x_m die Lösung von (1), d.h. $\|Ax_m - b\|_2 \leq \varepsilon/2$. Damit gilt auch $\|Ax_{m+1} - b\|_2 \leq \varepsilon/2$ und somit

$$\begin{aligned} \|x_{m+1} - x_m\|_2 &= \|x_{m+1} - A^{-1}b + A^{-1}b - x_m\|_2 \\ &\leq \|x_{m+1} - A^{-1}b\|_2 + \|x_m - A^{-1}b\|_2 \\ &\leq \varepsilon. \end{aligned}$$

Für den optimalen Dämpfungsparameter θ ergibt sich für positiv definite Matrizen nach [1]

$$\theta = \frac{2}{\gamma + \Gamma} \leq 1 \tag{33}$$

mit

$$\gamma\theta W_\theta \leq A \leq \Gamma\theta W_\theta, \tag{34}$$

wobei W die Matrix der dritten Normalform ist, d.h. $W_\theta = N_\theta^{-1} = \frac{1}{\theta}(\sum_{i=1}^{n_p} p_i r_i)^{-1}$.
Sei K die maximale Anzahl von Teilblöcken der Matrix A, d.h. die maximale Anzahl von Prozessoren mit $K \leq n_p$, die miteinander verbunden sind, dann gilt (34) mit $\Gamma = K$. Γ gibt den Grad der gegenseitigen Verbundenheit der Teilgebiete an. Zur Berechnung einer unteren Schranke für γ bietet der folgende Satz eine Abschätzung (siehe [1]).
SATZ 1:
A sei positiv definit und symmetrisch. Sei c eine Konstante, so daß zu jedem $x \in \mathrm{R}^n$ *eine Darstellung* $x = \sum_{i=1}^{n_p} p_i x^i \quad (x^i \in \mathrm{R}^{n_i})$ *existiert mit*

$$\sum_{i=1}^{n_p} (Ap_i x^i, p_i x^i) \leq c(Ax, x). \tag{35}$$

Dann gilt die erste Ungleichung $\gamma\theta W_\theta \leq A$ *in (34) mit* $\gamma = 1/c$.
Für disjunkte Teilgebiete (23) erhält man nach dem obigen Satz $c = 1$ und damit $\gamma = 1$. Hiermit erhält man für θ eine erste Abschätzung.

$$\theta = \frac{2}{1 + \Gamma} \leq 1 \tag{36}$$

Wird für jedes Teilgebiet $i, i = 1, \ldots, n_p$, eine Größe Γ_i berechnet, so geht (36) in

$$\theta_i = \frac{2}{1 + \Gamma_i} \leq 1 \tag{37}$$

über. Die Größe Γ_i gibt den Grad der gegenseitigen Verbundenheit für das Teilgebiet mit dem Index i an. Mit $\theta = (\theta_1, \ldots, \theta_{n_p})$ und (37) wird die Korrekturvorschrift (32) erweitert zu

$$x_{m+1} = x_m - \sum_{i=1}^{n_p} \theta_i \delta x_m^i. \tag{38}$$

Für sich überlappende Teilprobleme, d.h. (24) gilt, ist (37) nicht geeignet. Hierzu ist weitere Kommunikation notwendig.
Sei $l \geq 0$ die Anzahl der Zeilen, die in aufsteigender oder abfallender Zeilennumerierung bzgl. der Matrix A von einem Teilproblem in das benachbarte bzw. die benachbarten Teilprobleme hineinragen. Hieraus wird der Grad der gegenseitigen Überlappung berechnet. Es gilt dann

$$k = min(n_p - 1, \frac{2 \cdot l + n^r - 1}{n^r}). \tag{39}$$

Zu jedem $x \in \mathrm{R}^n$ gibt es eine Darstellung

$$x = \sum_{i=1}^{n_p} \sum_{j=1}^{n_i} (1 + c_{i_j})^{-1} p_{i_j} x^{i_j} \tag{40}$$

mit $x^i \in \mathrm{R}^{n_i}$, $p_{i_j} : \mathrm{R} \longrightarrow \mathrm{R}^n$ und

$$p_{i_j} = \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{array} \right) \begin{array}{l} \\ \\ \\ i_j \\ \\ \\ \\ \\ \\ \end{array} \Bigg\} \text{Block zum Index } \; i \; \text{ mit } \; p_{i_j}(i_j) = 1$$

für $j = 1, \ldots, n_i$. Die Größe $c_{i_j}, 0 \leq c_{i_j} \leq k$ mit k aus (39), charakterisiert die wirkliche Überlappung für jedes $x^{i_j} \in \mathrm{R}$. Betrachtet man nun die Überlappung c_ν für jede Komponente $x_\nu \in \mathrm{R}$ für $\nu = 1, \ldots, n$ von $x \in \mathrm{R}^n$, so erhält man c_ν aus c_{i_j} dadurch, indem man die Überlappung c_{i_j} der mehrfach auftretenden Komponenten x^{i_j} von x wegfallen läßt. Hiermit erhält man die Abschätzung

$$\begin{aligned}
\sum_{i=1}^{n_p} \sum_{j=1}^{n_i} (A(1 + c_{i_j})^{-1} p_{i_j} x^{i_j}, (1 + c_{i_j})^{-1} p_{i_j} x^{i_j}) &= \sum_{i=1}^{n_p} \sum_{j=1}^{n_i} ((1 + c_{i_j})^{-1} p_{i_j} x^{i_j}, (1 + c_{i_j})^{-1} p_{i_j} x^{i_j})_A \\
&= \sum_{i=1}^{n_p} \sum_{j=1}^{n_i} (1 + c_{i_j})^{-2} (p_{i_j} x^{i_j}, p_{i_j} x^{i_j})_A \\
&= \sum_{\nu=1}^{n} (1 + c_\nu)^{-2} (1 + c_\nu)(e_\nu x_\nu, e_\nu x_\nu)_A \\
&\leq ((I + C)^{-1} Ax, x),
\end{aligned}$$

wobei

$$I+C=\begin{pmatrix} 1+c_1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & 1+c_\nu & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1+c_n \end{pmatrix}.$$

Der Vektor $e_\nu \in \mathrm{R}^n$ ist hierbei der Einheitsvektor, der an der Stelle ν eine 1 hat und sonst nur Nullelemente. Nun läßt sich Satz 1 zu dem folgenden Satz erweitern.

SATZ 2:

A sei positiv definit und symmetrisch. Sei C eine Diagonalmatrix, so daß zu jedem $x \in \mathrm{R}^n$ *eine Darstellung* $x = \sum_{i=1}^{n_p} \tilde{p}_i x^i = \sum_{i=1}^{n_p} \sum_{j=1}^{n_i} (1+c_{i_j})^{-1} p_{i_j} x^{i_j} \quad (x^i \in \mathrm{R}^{n_i})$ *existiert mit*

$$\sum_{i=1}^{n_p} (A\tilde{p}_i x^i, \tilde{p}_i x^i) \le ((I+C)^{-1}Ax, x). \tag{41}$$

Dann gilt die erste Ungleichung in (34) mit $\gamma = I + C$.

Beweis:

Der Beweis erfolgt analog zum Satz 1.

Wegen $P_i\tilde{p}_i = \tilde{p}_i$ und der *Cauchy-Scharz*-Ungleichung ist

$$\begin{aligned} ((I+C)^{-1}Ax, x) &= ((I+C)^{-1}Ax, \sum_{i=1}^{n_p} \tilde{p}_i x^i) = \sum_{i=1}^{n_p} ((I+C)^{-1}Ax, \tilde{p}_i x^i) \\ &= \sum_{i=1}^{n_p} ((I+C)^{-1}Ax, P_i\tilde{p}_i x^i) = \sum_{i=1}^{n_p} (A(I+C)^{-1}P_i x, \tilde{p}_i x^i) \\ &= \sum_{i=1}^{n_p} ((I+C)^{-1}P_i x, \tilde{p}_i x^i)_A \le \sum_{i=1}^{n_p} \|(I+C)^{-1}P_i x\|_A \cdot \|\tilde{p}_i x^i\|_A \\ &\le (\sum_{i=1}^{n_p} \|(I+C)^{-1}P_i x\|_A^2)^{1/2} \cdot (\sum_{i=1}^{n_p} (\|\tilde{p}_i x^i\|_A^2)^{1/2}. \end{aligned}$$

Voraussetzung (41) liefert

$$\sum_{i=1}^{n_p} \|\tilde{p}_i x^i\|_A^2 = \sum_{i=1}^{n_p} (A\tilde{p}_i x^i, \tilde{p}_i x^i) \le ((I+C)^{-1}Ax, x).$$

Hieraus ergibt sich

$$\begin{aligned} ((I+C)^{-1}Ax, x)^2 &\le \sum_{i=1}^{n_p} \|(I+C)^{-1}P_i x\|_A^2 \cdot \sum_{i=1}^{n_p} \|\tilde{p}_i x^i\|_A^2 \\ &\le ((I+C)^{-1}Ax, x) \sum_{i=1}^{n_p} ((I+C)^{-1}AP_i x, (I+C)^{-1}P_i x) \end{aligned}$$

und damit

$$((I+C)^{-1}Ax, x) \le \sum_{i=1}^{n_p} ((I+C)^{-2}AP_i x, x).$$

Dies beweist

$$(I+C)^{-1}A \le \sum_{i=1}^{n_p} (I+C)^{-2}AP_i$$

und damit

$$A \leq \sum_{i=1}^{n_p}(I+C)^{-1}AP_i \leq (I+C)^{-1}A\sum_{i=1}^{n_p}P_i \leq (I+C)^{-1}A\sum_{i=1}^{n_p}\tilde{p}_i\tilde{r}_iA,$$

woraus

$$A^{-1} \leq (I+C)^{-1}\sum_{i=1}^{n_p}\tilde{p}_i\tilde{r}_i = (I+C)^{-1}\theta N_\theta$$

und $\gamma = I + C$ folgen.
qed
Nach Satz 2 gilt mit $\gamma_{i_j} = 1 + c_{i_j}$ für die Berechnung von θ_{i_j},

$$\theta_{i_j} = \frac{2}{\gamma_{i_j} + \Gamma_i}. \tag{42}$$

Mit (42) geht die Korrektur (38) in die veränderte Berechnungsvorschrift

$$x_{m+1} = x_m - \sum_{i=1}^{n_p}\sum_{j=1}^{n_i}\theta_{i_j}\delta_m^{i_j} \tag{43}$$

über. Mit (39) und (42) gilt die folgende Abschätzung

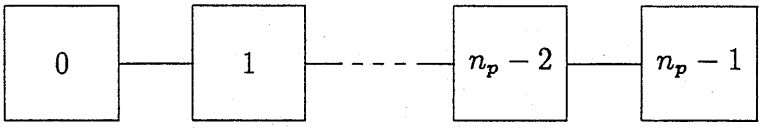
$$\begin{aligned}
\theta_{i_j} &= \frac{2}{1 + c_{i_j} + \Gamma_i} \\
&= \frac{2}{(1+k)\frac{(1+c_{i_j})}{(1+k)} + \Gamma_i} \\
&= \frac{2}{(1 + k + \Gamma_i\frac{(1+k)}{(1+c_{i_j})})} \cdot \frac{1}{(\frac{1+c_{i_j}}{1+k})} \\
&\leq min(1, \frac{2}{1+k+\Gamma_i} \cdot (\frac{1+k}{1+c_{i_j}})) \leq 1.
\end{aligned}$$

Erste praktische Rechnungen haben gezeigt, daß die Wahl von

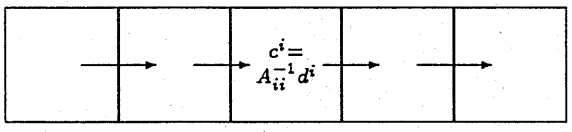
$$\theta_{i_j} = min(1, \frac{2}{1+k+\Gamma_i} \cdot (\frac{1+k}{1+c_{i_j}})) \leq 1 \tag{44}$$

für (43) effektiver ist als (42) für (43).
Für die rechentechnische Realisierung auf Distributed Memory Systemen wird das Gleichungssystem (1) in Teilprobleme (mit bzw. ohne Überlappung) der Gestalt (20) auf eine Prozessorpipeline mit n_p Prozessoren aufgeteilt. Nach der Lösung der niederdimensionalen Teilprobleme (20) werden die einzelnen Lösungen $x^i, i = 1, \ldots, n_p$, zum Prozessor mit der Identifikationsnummer $n_p - 1$ verschickt und zusammengefaßt. Dies bedeutet gleichzeitig eine Aufsummierung der sich überlappenden Komponenten. Der so gebildete Defekt in der Berechnungsvorschrift (43) wird ausgehend vom Prozessor mit der Identifikation $n_p - 1$ an alle anderen Prozessoren der Pipeline verschickt. Die folgenden Darstellungen sollen dies verdeutlichen.

Prozessorpipeline:

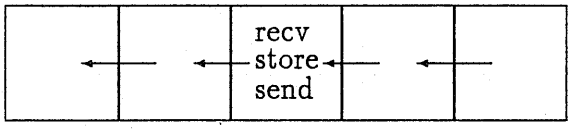


Zusammenfassung mit notwendiger Aufsummierung:



siehe (20) und (43)

Verteilung des Defekts:



siehe (43)

Eine notwendige Reduktion der Prozessorpipeline von n_p Prozessoren auf n_p^r Prozessoren ist auch vorgesehen.

4 Numerische Ergebnisse

An zwei Beispielen soll ein Vergleich der beiden Speicherungstechniken (siehe 1.3.1 bzw. 1.3.2) bzgl. der *Matrix∗Vektor*-Operation durchgeführt werden. Als erstes Beispiel wird die Beispielmatrix der Ordnung $(10, 10)$ aus Punkt 1.3.1 betrachtet. Die Matrix wird mit $A_{10,10}$ bezeichnet. Hierfür ist $n_z = 9$. Aus der Matrix $A_{10,10}$ wird die Vergleichsmatrix A für verschiedene Dimensionen n wie folgt aufgebaut:

$$A = \begin{pmatrix} A_{10,10} & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & A_{10,10} & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & A_{10,10} \end{pmatrix}.$$

Die Zeitmessungen wurden auf ein MultiCluster-2 System mit 32 Prozessoren vom Typ T800 mit rekonfigurierbarer Topologie unter dem Betriebssystem PARIX, Release 1.1, in ACE FORTRAN [2] durchgeführt und sind in μs angegeben. Das Übersetzen der einzelnen Routinen und das Linken zu einem Hauptprogramm erfolgte auf dem Hostrechner (SUN SPARCstation).

Die obere Zeile in Tabelle 1 für verschiedene Dimensionen n gibt die CPU-Zeiten für die unter Punkt 1.3.1 beschriebene Speicherungstechnik (A in AC und KA) an, während die untere Zeile die CPU-Zeiten für die unter Punkt 1.3.2 beschriebene Speicherungstechnik,

Tabelle 1: CPU-Zeiten für die *Matrix* Vektor*-Operation für unterschiedliche Speicherungstechniken bzgl. Beispiel 1

Dimension	Prozessorgitter							
	(1,1)	(2,1)	(2,2)	(4,1)	(4,2)	(8,1)	(4,4)	(5,5)
n= 1000	74960	49580	49484	55412	60189	95906	82771	160473
	40771	42902	76652	65404	136475	125763	286104	442837
n= 5000	376018	247307	246776	273141	292939	464957	345577	635756
	205115	214475	381374	323532	672263	615180	1411854	2185312
n=10000	723197	494141	493330	545268	584297	925809	673629	994007
	412342	429023	762821	645841	1343377	1226538	2819864	4361965

Tabelle 2: CPU-Zeiten für die *Matrix* Vektor*-Operation für unterschiedliche Speicherungstechniken bzgl. Beispiel 2

Dimension	Prozessorgitter							
	(1,1)	(2,1)	(2,2)	(4,2)	(8,1)	(4,4)	(16,1)	(5,5)
n= 100	75477	39062	22396	16138	24140	31402	56361	85495
	37136	21682	16957	19739	18946	34966	32709	55566
n= 500	377263	194682	110771	77949	94222	86376	141737	132537
	186309	105462	80833	90891	86011	156570	141966	232777
n=1000	722917	389114	221306	146039	181223	134778	251409	182984
	374191	210554	161104	179561	168157	308226	274851	457409

d.h. A in ACP, KAP, IAP und JAP angibt.
Die zweite Beispielmatrix wird ähnlich erzeugt wie die erste. Anstelle der Matrix $A_{10,10}$ wird eine Matrix $A_{100,100}$ verwendet, die analog wie die Matrix $A_{10,10}$ aufgebaut ist. Hierfür ist $n_z = 99$. Tabelle 2 ist dann analog zu Tabelle 1 zu lesen.

Als Ergebnis ist aus den beiden Tabellen 1 und 2 ersichtlich, daß die zweite Speicherungsart für wenige Prozessoren n_p sehr effektiv ist. Dies liegt daran, daß bei einer großen Prozessoranzahl die Kommunikation zwischen den Prozessoren sehr viel Zeit einnimmt, da ständig ein Austausch von n Elementen erfolgt, der nicht parallel durchgeführt werden kann. Die erste Speicherungstechnik ist bei einer höheren Prozessoranzahl sehr effektiv, wenn die Matrix A in ihrer Speicherungsform AC und KA "gut" auf das Prozessorgitter (d_x, d_y) abgebildet wird. Eine Prozessorpipeline, d.h. $d_x = 1$ oder $d_y = 1$, ist i. allg. nicht sinnvoll. Die Kommunikation in y-Richtung erfolgt "annähernd" parallel.
Jetzt soll noch an vier Beispielen die Leistungsfähigkeit der verschiedenen parallelen Varianten des GMRES-Algorithmus demonstriert werden. Es werden folgende Varianten verglichen:

GMRES(mv)	:	GMRES mit $Matrix * Vektor$-Parallelisierung
GMRES(dd)	:	GMRES angewandt auf die *additive Schwarz*-Iteration (ohne Überlappung)
GMRES(dd+1)	:	GMRES angewandt auf die *additive Schwarz*-Iteration (mit Überlappung)

Es werden folgende Gleichungssysteme betrachtet:

- Gleichungssystem 1:
  Verwendet wird die Beispielmatrix $A_{100,100}$, um das Gleichungssystem $Ax = b$ aufzustellen. Die Rechte Seite b ist so gewählt, daß alle Komponenten des Lösungsvektors x den Wert Eins haben, d.h. $x = (1, \ldots, 1)^T$.

- Gleichungssystem 2:
  Das zweite Gleichungssysytem setzt sich aus der *Grcar*-Matrix zusammen, d.h.

$$A = \begin{pmatrix} .9 & 1 & 1 & 1 & 0 & \cdots & \cdots & 0 \\ -1 & .9 & 1 & 1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & .9 & 1 & 1 & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & -1 & .9 & 1 & 1 & 1 \\ \vdots & & & \ddots & -1 & .9 & 1 & 1 \\ \vdots & & & & \ddots & -1 & .9 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & -1 & .9 \end{pmatrix}.$$

  Die Rechte Seite b ist hier auch wieder so gewählt, daß alle Komponenten des Lösungsvektors x den Wert Eins haben.

- Gleichungssystem 3:
  Als Matrix A für das dritte Gleichungssystem wird $A = (a_{ij})_{i=1,\ldots,n;j=1,\ldots,n}$ mit $a_{ij} = 1 + max(i, j)$ gewählt. Die Konditionszahl $\kappa(A)$ beträgt $4n(n+1)$.

$$A = \begin{pmatrix} 2 & 3 & 4 & \cdots & n \\ 3 & 3 & 4 & \ldots & n \\ \vdots & \vdots & \vdots & & \vdots \\ n & n & n & \ldots & n \end{pmatrix}$$

  Als Vektor der Rechten Seite wird der Vektor $b = (1, 2, 3, \ldots, n)^T$ gewählt. Damit ergibt sich der Lösungsvektor x zu $x = (1, 0, \ldots, 0, -\frac{1}{n+1})^T$.

- Gleichungssystem 4:
  Als letztes Beispiel wird die Matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 0 & 1 & \ddots & & & \vdots \\ 0 & -1 & 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 & 1 & 0 \\ \vdots & & & \ddots & -1 & 0 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & -1 & 0 \end{pmatrix}$$

  mit n gerade betrachtet.

  - Eigenwerte von$A$: $\lambda_k = i \cdot \cos(\frac{k\pi}{n+1})$ für $k = 1, \ldots, n$
  - Minimaler Eigenwert: $\lambda_{min} = i \cdot \cos(\frac{k\pi}{n+1})$ für $k = \frac{n}{2}$ oder $k = \frac{n+2}{2}$
  - Maximaler Eigenwert: $\lambda_{max} = i \cdot \cos(\frac{k\pi}{n+1})$ für $k = 1$ oder $k = n$
  - Konditionszahl $\kappa(A)$: $\kappa(A) = \cos(\frac{\pi}{n+1}) / \cos(\frac{n\pi}{2(n+1)})$

  Der Vektor der Rechten Seite ist durch $b = (2, 0, \ldots, 0, 2)^T$ beschrieben. Damit ist der Lösungsvektor x durch $x = (2, 2, \ldots, 2, 2)^T$ definiert.

Die Testrechnungen wurden einheitlich mit dem Startvektor $x_0 = (100, \ldots, 100)^T$ durchgeführt. Für die maximale Dimension des *Krylov*-Unterraumes wurde für die Gleichungssysteme 1 und 2 $m = 20$ gewählt. Das Gleichungssystem 3 wurde mit $m = 20$ und $m = 36$ untersucht. Für die maximale Dimension des *Krylov*-Unterraumes beim Gleichungssystem 4 wurde $m = 50$ gewählt. Für die Anzahl der Zeilen, die in aufsteigender oder abfallender Zeilennumerierung bzgl. der Matrix A von einem Teilproblem in das benachbarte hineinragen, wurde für die Gleichungssysteme 1, 2 und 3 $l = 10$ gewählt. Für das Gleichungssystem 4 ist $l = 2$. Die Iteration wird bendet, falls für die Iterierte x_m die Bedingung $\|Ax_m - b\|_2 \leq \varepsilon$ für die Toleranz $\varepsilon = 10^{-7}$ erfüllt ist.
Die Ergebnisse sind in den Tabellen 3 - 6 zu finden. Die obere Zeile für verschiedene Dimensionen n gibt die CPU-Zeiten in *sec* für die beschriebenen Verfahren an, während die untere Zeile die Iterationscharakteristika in der Form (it_1, it_2) für die $Matrix * Vektor$-Parallelisierung (GMRES(mv)) widerspiegelt. Die Größe $it_1 \geq 0$ gibt die Anzahl der "restarts" an, während it_2, $1 \leq it_2 \leq m$, die Dimension des *Krylov*-Unterraumes angibt, bei dem Konvergenz eintrat beim "restart" mit der Nummer it_1. Für die *additive Schwarz*-Iteration ist in der unteren Zeile die Anzahl der *Schwarz*-Iterationen angegeben.
Die Tabelle 3 zeigt deutlich, daß die $Matrix*Vektor$-Parallelisierung für große n_p (Anzahl der Prozessoren) sehr effetiv ist. Wird bei der *additiven Schwarz*-Iteration die Prozessoranzahl nicht richtig gewählt, tritt ein Effektivitätsverlust ein. Werden Blöcke vom Typ $(100, 100)$ zusammengefaßt, d.h. der Quotient $\frac{n}{n_p}$ ist ein Vielfaches von 100, so erhält man auch hier ein sehr effektives Verfahren. Das folgende Beispiel demonstriert dies. Werden für $n = 1400$ $14 = (14, 1)$ Prozessoren gewählt, so werden 3 *Schwarz*-Iterationen benötigt

Tabelle 3: CPU-Zeiten in *sec* und Iterationscharakteristika für das Gleichungssystem 1

Dimension	GMRES(mv) Prozessorgitter			GMRES(dd) Prozessoranzahl		GMRES(dd+1) Prozessoranzahl	
	(1,1)	(2,2)	(5,5)	4=(4,1)	25=(25,1)	4=(4,1)	25=(25,1)
n= 400	5.69	2.29	1.76	3.14	27.34	22.25	15.24
	(1,15)	(1,15)	(1,15)	2	84	16	32
n= 600	8.53	3.43	2.59	55.25	30.69	24.76	23.90
	(1,15)	(1,15)	(1,15)	23	61	16	35
n= 800	11.36	4.55	3.41	6.65	41.70	31.19	23.91
	(1,15)	(1,15)	(1,15)	2	60	16	29
n=1000	14.20	5.68	4.25	94.85	38.19	38.04	28.65
	(1,15)	(1,15)	(1,15)	23	43	16	28
n=1200	17.03	6.81	5.08	10.03	44.41	44.39	31.00
	(1,15)	(1,15)	(1,15)	2	41	16	28
n=1400	19.87	7.95	5.91	135.3	54.04	48.97	33.04
	(1,15)	(1,15)	(1,15)	23	41	16	27

Tabelle 4: CPU-Zeiten in *sec* und Iterationscharakteristika für das Gleichungssystem 2

Dimension	GMRES(mv) Prozessorgitter			GMRES(dd) Prozessoranzahl		GMRES(dd+1) Prozessoranzahl	
	(1,1)	(2,2)	(5,5)	4=(4,1)	25=(25,1)	4=(4,1)	25=(25,1)
n= 400	38.67	38.21	48.67	283.5	12.82	106.3	26.21
	(22,4)	(22,4)	(22,4)	57	59	17	20
n= 800	79.54	78.39	98.63	633.3	68.20	213.7	45.34
	(22,19)	(22,19)	(22,19)	57	59	17	18
n=1200	119.1	117.3	147.4	994.3	141.5	325.8	63.84
	(22,19)	(22,19)	(22,19)	57	59	17	18
n=1600	158.4	156.0	195.9	1355	201.0	437.9	81.74
	(22,19)	(22,19)	(22,19)	57	59	17	18
n=2000	197.7	194.8	244.1		261.3		99.98
	(22,19)	(22,19)	(22,19)	-	59	-	18
n=2400	237.8	233.9	293.4		321.8		119.1
	(22,19)	(22,19)	(22,19)	-	59	-	18

Tabelle 5: CPU-Zeiten in *sec* und Iterationscharakteristika für das Gleichungssystem 3

Dimension	GMRES(mv), m=20			GMRES(mv), m=36		
	Prozessorgitter			Prozessorgitter		
	(1,1)	(2,2)	(5,5)	(1,1)	(2,2)	(5,5)
n= 400	325.9 (14,20)	98.39 (14,14)	42.43 (14,11)	124.8 (3,34)	41.27 (3,33)	21.02 (3,34)
n= 800	-	648.9 (26,8)	171.7 (22,3)	-	211.6 (5,16)	76.29 (5,17)
n=1200	-	-	355.5 (24,16)	-	-	107.0 (6,19)
n=1600	-	-	729.6 (31,16)	-	-	249.9 (6,11)
n=1800	-	-	-	-	-	321.4 (6,25)
n=2000	-	-	-	-	-	384.2 (6,28)

bei einem CPU-Aufwand von 4.81 *sec*. Tabelle 4 verdeutlicht, daß die $Matrix * Vektor$-Parallelisierung bei diesem Beispiel nicht sehr effektiv ist. Dies liegt an der geringen Größe von $n_z = 5$. Auch bei einem Prozessorgitter von $(2, 1)$ erhält man keine wesentlichen Verbesserungen. Es wurden 224.6 *sec* benötigt. Dagegen erweist sich die *additive Schwarz*-Iteration mit Überlappung für eine hohe Prozessoranzahl als effektives Verfahren. Für die Ordnungen $n = 2000$ bzw. $n = 2400$ liefert die *additive Schwarz*-Iteration für vier Prozessoren zu große CPU-Zeiten.

Die *additive Schwarz*-Iteration sowohl mit als auch ohne Überlappung liefert für das Gleichungssystem 5 mit voll besetzter Matrix entweder gar keine Konvergenz oder konvergiert nur sehr langsam für kleine Ordnungen n und für eine kleine Prozessoranzahl n_p. Wie aus der Tabelle 5 ersichtlich ist, ist für dieses Beispiel die Wahl des Unterraumes, d.h. die Dimensionierung, von entscheidender Bedeutung. Die freien Plätze in Tabell 5 ergeben sich dadurch, daß der Speicherplatz pro Prozessor überschritten wird oder die CPU-Zeiten zu stark ansteigen.

Aus Tabelle 6 ist ersichtlich, daß die *additive Schwarz*-Iteration für wachsende Prozessoranzahl ein sehr effektives Verfahren ist. Auch bei diesem Beispiel ist die Wahl des *Krylov*-Unterraumes sowohl für GMRES(mv) als auch für GMRES(dd) bzw. GMRES(dd+1) von entscheidender Bedeutung. Für GMRES(mv) z.B. ergeben sich für $n = 100$ und $m = 20$ die folgenden Ergebnisse.

Tabelle 6: CPU-Zeiten in *sec* und Iterationscharakteristika für das Gleichungssystem 4

Dimension	GMRES(mv) Prozessorgitter			GMRES(dd) Prozessoranzahl		GMRES(dd+l) Prozessoranzahl	
	(1,1)	(2,2)	(5,5)	4=(4,1)	25=(25,1)	4=(4,1)	25=(25,1)
n= 100	140.6 (59,6)	143.8 (59,6)	155.2 (59,6)	n/4 unger.	4.45 106	n/4 unger.	3.91 72
n= 200	1116 (247,48)	1139 (247,48)	1231 (247,48)	51.55 39	9.19 105	216.3 19	6.06 55
n= 400	- -	-	-	-	22.39 105	-	14.21 54
n= 800	- -	-	-	-	70.39 105	-	45.25 54
n=1000	- -	-	-	-	112.1 104	-	71.01 54
n=1200	- -	-	-	-	166.9 104	-	618.6 54

Prozessorgitter		
(1,1)	(2,2)	(5,5)
179.8 (399,8)	190.1 (399,8)	224.4 (399,8)

Für die *additive Schwarz*-Iteration kann man dagegen feststellen, wenn die Ordnung der niederdimensionalen Teilprobleme die maximale Dimension m des *Krylov*-Unterraumes nicht übersteigt, so bekommt man ein sehr effektives Verfahren. So erhält man z.B. für $n = 1200$, $l = 2$ und $n_p = 25$ niederdimensionale Teilprobleme der Ordnung 50 bzw. 52. Wie aus Tabelle 6 zu erkennen ist, wirkt sich diese Konstellation negativ auf die benötigte CPU-Zeit aus. Wird für $n = 1200$, $l = 2$ und $n_p = 25$ die maximale Dimension $m = 52$ gewählt, so konvergiert GMRES(dd+l) nach 54 Iterationen, wobei 104.6 *sec* benötigt werden. Die freien Plätze in Tabelle 6 weisen darauf hin, daß die CPU-Zeit viel zu stark angestiegen ist.

Literatur

[1] W.Hackbusch, Iterative Lösung großer schwachbesetzter Gleichungssysteme , B.G.Teubner, Stuttgart (1991).

[2] PARIX, Release 1.1, Software Documentation, Manual Pages, Parsytec Computer GmbH, (1992).

[3] W.Rönsch, R.Reuter, Lineare Algebra für Parallelrechner, Scientific Center Technical Report, IBM Wissenschaftliches Zentrum Heidelberg, (Februar 1991).

[4] K.Solchenbach, Programmierung von Parallelrechnern, Erschienen in: Computerwoche Nr. 32, (7.August 1992).

[5] R.Schlundt, Iterative Verfahren für lineare Gleichungssysteme mit schwach besetzten Koeffizientenmatrizen, Preprint, Institut für Angewandte Analysis und Stochastik, (1992).

Veröffentlichungen des Instituts für Angewandte Analysis und Stochastik

Preprints 1992

1. D.A. Dawson, J. Gärtner: Multilevel large deviations.
2. H. Gajewski: On uniqueness of solutions to the drift–diffusion–model of semiconductor devices.
3. J. Fuhrmann: On the convergence of algebraically defined multigrid methods.
4. A. Bovier, J.-M. Ghez: Spectral properties of one–dimensional Schrödinger operators with potentials generated by substitutions.
5. D.A. Dawson, K. Fleischmann: A super–Brownian motion with a single point catalyst.
6. A. Bovier, V. Gayrard: The thermodynamics of the Curie–Weiss model with random couplings.
7. W. Dahmen, S. Prößdorf, R. Schneider: Wavelet approximation methods for pseudodifferential equations I: stability and convergence.
8. A. Rathsfeld: Piecewise polynomial collocation for the double layer potential equation over polyhedral boundaries. Part I: The wedge, Part II: The cube.
9. G. Schmidt: Boundary element discretization of Poincaré–Steklov operators.
10. K. Fleischmann, F.I. Kaj: Large deviation probability for some rescaled superprocesses.
11. P. Mathé: Random approximation of finite sums.
12. C.J. van Duijn, P. Knabner: Flow and reactive transport in porous media induced by well injection: similarity solution.
13. G.B. Di Masi, E. Platen, W.J. Runggaldier: Hedging of options under discrete observation on assets with stochastic volatility.
14. J. Schmeling, R. Siegmund–Schultze: The singularity spectrum of self-affine fractals with a Bernoulli measure.
15. A. Koshelev: About some coercive inequalities for elementary elliptic and parabolic operators.
16. P.E. Kloeden, E. Platen, H. Schurz: Higher order approximate Markov chain filters.

17. H.M. Dietz, Y. Kutoyants: A minimum–distance estimator for diffusion processes with ergodic properties.

18. I. Schmelzer: Quantization and measurability in gauge theory and gravity.

19. A. Bovier, V. Gayrard: Rigorous results on the thermodynamics of the dilute Hopfield model.

20. K. Gröger: Free energy estimates and asymptotic behaviour of reaction–diffusion processes.

21. E. Platen (ed.): Proceedings of the 1^{st} workshop on stochastic numerics.

22. S. Prößdorf (ed.): International Symposium "Operator Equations and Numerical Analysis" September 28 – October 2, 1992 Gosen (nearby Berlin).

23. K. Fleischmann, A. Greven: Diffusive clustering in an infinite system of hierarchically interacting diffusions.

24. P. Knabner, I. Kögel–Knabner, K.U. Totsche: The modeling of reactive solute transport with sorption to mobile and immobile sorbents.

25. S. Seifarth: The discrete spectrum of the Dirac operators on certain symmetric spaces.

26. J. Schmeling: Hölder continuity of the holonomy maps for hyperbolic basic sets II.

27. P. Mathé: On optimal random nets.

28. W. Wagner: Stochastic systems of particles with weights and approximation of the Boltzmann equation. The Markov process in the spatially homogeneous case.

29. A. Glitzky, K. Gröger, R. Hünlich: Existence and uniqueness results for equations modelling transport of dopants in semiconductors.

30. J. Elschner: The h-p–version of spline approximation methods for Mellin convolution equations.

31. R. Schlundt: Iterative Verfahren für lineare Gleichungssysteme mit schwach besetzten Koeffizientenmatrizen.

32. G. Hebermehl: Zur direkten Lösung linearer Gleichungssysteme auf Shared und Distributed Memory Systemen.

33. G.N. Milstein, E. Platen, H. Schurz: Balanced implicit methods for stiff stochastic systems: An introduction and numerical experiments.

34. M.H. Neumann: Pointwise confidence intervals in nonparametric regression with heteroscedastic error structure.

35. M. Nussbaum: Asymptotic equivalence of density estimation and white noise.

Preprints 1993

36. B. Kleemann, A. Rathsfeld: Nyström's method and iterative solvers for the solution of the double layer potential equation over polyhedral boundaries.

37. W. Dahmen, S. Prössdorf, R. Schneider: Wavelet approximation methods for pseudodifferential equations II: matrix compression and fast solution.

38. N. Hofmann, E. Platen, M. Schweizer: Option pricing under incompleteness and stochastic volatility.

39. N. Hofmann: Stability of numerical schemes for stochastic differential equations with multiplicative noise.

40. E. Platen, R. Rebolledo: On bond price dynamics.

41. E. Platen: An approach to bond pricing.

42. E. Platen, R. Rebolledo: Pricing via anticipative stochastic calculus.

43. P.E. Kloeden, E. Platen: Numerical methods for stochastic differential equations.

44. L. Brehmer, A. Liemant, I. Müller: Ladungstransport und Oberflächenpotentialkinetik in ungeordneten dünnen Schichten.

45. A. Bovier, C. Külske: A rigorous renormalization group method for interfaces in random media.

46. G. Bruckner: On the regularization of the ill–posed logarithmic kernel integral equation of the first kind.

47. H. Schurz: Asymptotical mean stability of numerical solutions with multiplicative noise.

48. J.W. Barrett, P. Knabner: Finite element approximation of transport of reactive solutes in porous media. Part I: Error estimates for non–equilibrium adsorption processes.

49. M. Pulvirenti, W. Wagner, M.B. Zavelani Rossi: Convergence of particle schemes for the Boltzmann equation.

50. J. Schmeling: Most β shifts have bad ergodic properties.

51. J. Schmeling: Self normal numbers.

52. D.A. Dawson, K. Fleischmann: Super–Brownian motions in higher dimensions with absolutely continuous measure states.

53. A. Koshelev: Regularity of solutions for some problems of mathematical physics.

54. J. Elschner and I.G. Graham: An optimal order collocation method for first kind boundary integral equations on polygons.