Institut für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Zur direkten Lösung linearer Gleichungssysteme auf Shared und Distributed Memory Systemen

G. Hebermehl

submitted: 18th December 1992

Institut für Angewandte Analysis
und Stochastik
Hausvogteiplatz 5-7
D – O 1086 Berlin
Germany

Preprint No. 32
Berlin 1992

1991 Mathematics Subject Classification. 65 F 05, 65 Y 05, 65 Y 10.
Key words and phrases. Gauß–Elimination, Rank–r LU Update Verfahren, blockzyklische Datenaufteilung, Topologie, Kommunikation, Shared Memory System, Distributed Memory System, Message Passing.

Herausgegeben vom
Institut für Angewandte Analysis und Stochastik
Hausvogteiplatz 5-7
D – O 1086 Berlin

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

Inhaltsverzeichnis

Zusammenfassung

Der Gaußsche Algorithmus zur Lösung linearer Gleichungssysteme $Ax = b$ wird für Shared Memory Systeme als Rank-r LU Update Verfahren und bei blockzyklischer Aufteilung von A für Distributed Memory Systeme als Message Passing Implementation vorgestellt.

1 Einführung

Vom technischen und algorithmischen Standpunkt können die modernen Supercomputer im wesentlichen in 2 Klassen eingeteilt werden:

- die Shared Memory Computer
- die Distributed Memory Computer

Virtual Shared Memory Systeme stellen eine Klammer beider Welten mit gewissen Effizienzverlusten und Einschränkungen in den Parallelisierungsmöglichkeiten dar.

Shared Memory Computer bestehen i.allg. aus einer relativ kleinen Anzahl von hochleistungsfähigen Prozessoren und einem gemeinsamen Speicher, auf den alle diese Prozessoren zugreifen können.
Durch die Bereitstellung von Distributed Memory Systemen werden neue Wege beschritten, um die Spitzenleistung von Supercomputern bezüglich Geschwindigkeit und Speicherkapazität zu steigern. Sie bestehen i.allg. aus einer größeren Anzahl billigerer Prozessoren niedriger oder mittlerer Leistung mit lokalem Speicher oder sind als Workstation -Cluster aus leistungsfähigen RISC - Prozessoren aufgebaut.
Die für die Anwendungen notwendigen Rechengeschwindigkeiten sind nicht nur durch die Hardware, sondern auch durch geeignete, auf die Supercomputer abgestimmte Algorithmenentwicklungen zu erbringen.
Der Entwicklung paralleler Algorithmen für innovative Rechnerarchitekturen wird ein reges Interesse entgegengebracht. Insbesondere die Lösung linearer Gleichungssysteme ermöglicht das Studium der Benutzung einfacher Programmkerne, von Kommunikation, Netzwerktopologie, Last- und Datenverteilung auf Superrechnern. Die Untersuchungen für die linearen Gleichungssysteme dienen als ein anerkannter Maßstab für die tatsächlich erreichbare Spitzenleistung der Computer.
In dieser Arbeit werden auf der Gaußelimination beruhende Algorithmen zur Lösung linearer Gleichungssysteme

$$Ax = b, \quad A \in \mathrm{R}^{n \times n}, \quad x, b \in \mathrm{R}^n \tag{1}$$

für Shared und Distributed Memory Systeme vorgestellt:

- der Rank-r LU Update Algorithmus mit Unrolling für Vektorrechner
- die LU-Dekomposition bei blockzyklischer Aufteilung der Matrix auf die lokalen Speicher eines Distributed Memory Systems

Die Algorithmen wurden auf einer IBM 3090 bzw. einem Transputernetz implementiert.
Der hier beschriebene Rank-r LU Update Algorithmus ist auf eine optimale Ausnutzung einer hierarchischen Speicherstruktur ausgerichtet.
Die LU-Dekomposition mit blockzyklischem Datenlayout der Matrix ist für beliebige Prozessorgitter (ρ, σ) und Blöcke der Dimension (p, q) ausgelegt und ermöglicht daher eine gute Anpassung an die Hardware und das Datenlayout anderer Routinen.
Die Randprozessoren des zweidimensionalen Prozessorgitters können in unterschiedlicher Weise verbunden sein.

2 Die Standard-Gauß-Elimination mit partieller Pivotisierung

Betrachtet wird das lineare Gleichungssystem (1). Wenn A nichtsingulär ist, existiert eine eindeutige Lösung $x \in \mathrm{R}^n$ für (1).

Die LU-Dekomposition der Matrix A in eine untere Dreiecksmatrix L und in eine obere Dreiecksmatrix U wird mit partieller Pivotisierung durchgeführt:

$$A = LU \tag{2}$$

Durch (2) wird das lineare Gleichungssystem (1) in 2 Dreieckssyteme transformiert, die durch Vorwärts- und Rückwärtseinsetzen gelöst werden können:

$$LUx = b, \quad Ly = b, \quad Ux = y \tag{3}$$

U ist eine obere Dreiecksmatrix mit den Pivotelementen in der Hauptdiagonale.
Die LU-Dekomposition erzeugt L^{-1} als das Produkt aus $n-1$ Permutationsmatrizen P_k und $n-1$ elementaren Eliminationsmatrizen L_k:

$$L^{-1} = L_{n-1}P_{n-1}...L_2P_2L_1P_1, \tag{4}$$

so daß

$$L^{-1}A = U \quad (siehe(2)). \tag{5}$$

P_k ist eine Matrix, die aus der Einheitsmatrix durch Vertauschung der k-ten und der l-ten Zeile hervorgeht. L_k unterscheidet sich von der Einheitsmatrix dadurch, daß unterhalb der Hauptdiagonale in der k-ten Spalte die negativen Multiplikatoren zur Erzeugung der Nullen im k-ten Eliminationsschritt stehen. Die LU-Dekomposition (2) ergibt sich durch die Anwendung des folgenden Algorithmus auf die Matrix $A = (a_{ij})$:
Für $k = 1$ bis $n-1$ werden nacheinander

- das Pivotelement

$$|a_{l,k}| = \max_{k \le i \le n} |a_{i,k}| \tag{6}$$

  mit dem Pivotindex l bestimmt, der in einem Pivotvektor abgespeichert wird.

- die Zeilen k und l vertauscht.

- die negativen Eliminationsfaktoren

$$a_{i,k} := -\frac{a_{i,k}}{a_{k,k}}, \quad i = k + 1(1)n \tag{7}$$

  berechnet. Sie werden im unteren Dreieck von A gespeichert.

- das Rank-One Updating durchgeführt:

$$a_{i,j} := a_{i,j} + a_{i,k}a_{k,j}, \quad i,j = k + 1(1)n \tag{8}$$

Das obere Dreieck von A wird mit U überschrieben.
Die Lösung des oberen und unteren Dreieckssystems (3) leistet der folgende Algorithmus.
Für $k = 1(1)n - 1$ führe durch

- Anwendung der Permutation P auf die rechte Seite b, d.h. Vertauschung von b_k und b_l.

- Vorwärtseinsetzen

$$b_i := b_i + a_{i,k} b_k, \quad i = k + 1(1)n \tag{9}$$

Für $k = n(-1)1$ führe durch

- Division durch das Diagonalelement

$$b_k := \frac{b_k}{u_{k,k}}, \quad u_{k,k} \in U \tag{10}$$

- Rückwärtseinsetzen

$$b_i := b_i - u_{i,k} b_k, \quad i = 1(1)k - 1 \tag{11}$$

Die Lösung x steht auf dem Platz von b.
Für innovative Rechnerarchitekturen ist die Standard-Gauß-Elimination optimal an die jeweilige Architektur anzupassen, um in den Bereich der möglichen Spitzenleistung dieser Computer zu kommen.

3 Shared Memory Systeme

Die hohe Leistung in Vektorrechnern wird dadurch erreicht, daß entsprechend ihrer Architektur die gleiche Operation mit einer großen Anzahl von Operanden gemacht wird. Die Operanden können zu Vektoren zusammengefaßt werden und werden nach einem Fließbandprinzip (pipelining) in einem Vektorprozessor verarbeitet. In modernen Vektorrechnern sind unterschiedliche Vektorprozessoren (z.B. für die Addition und Multiplikation) hintereinandergeschaltet (chaining), so daß auch komplexe Vektoroperationen wie die verkettete Triade (SAXPY) oder das Skalarprodukt, also

$$\alpha x_i + y_i, \quad i = 1(1)n, \qquad \text{oder} \qquad \sum_{i=1}^{n} x_i y_i, \tag{12}$$

nach einer Startup-Zeit pro Zyklus ein Resultat bringen. Entscheidend ist auch, daß Daten mit hoher Geschwindigkeit zwischen Speicher und Prozessor hin- und herbewegt werden können. Dazu dient das hierarchische Speicherprinzip, in dem zwischen dem Prozessor und dem Hauptspeicher noch schnelle Register und u.U. noch ein Cache (high speed storage) mit kürzeren Zugriffszeiten geschaltet werden.

3.1 Vektorisierung

Beispielsweise verfügt die IBM 3090 300 E neben dem Zentralspeicher über einen Cache von 64 kByte und 8 Vektorregister (double precision) für jeweils 128 Elemente (Registerlänge). So kann z.B. eine Vektoraddition für 128 Elemente durchgeführt werden.

Längere Vektoren sind in Gruppen zu 128 Elementen und einen Rest aufzuteilen. Der Anwender wird bei der Vektorisierung durch geeignete Compiler unterstützt, die im wesentlichen Schleifen analysieren und vektorisieren. Unter Ausnutzung solcher Möglichkeiten kann der Nutzer sein Programm auf einen Vektorrechner bringen, und er wird i.allg. eine bessere Effizienz als im Skalarmodus erreichen. Die Tabelle 1 enthält Effektivitätsvergleiche für die folgenden 3 Versionen der Standard-Gauß-Elimination (2-11):

(a) Inline-Implementation, d.h. es werden keine Elementarmoduln verwendet. (Skalarmodus).

(b) Implementation unter Verwendung von in Vektor-ASSEMBLER geschriebenen Elementarmoduln.

(c) Vektorisierung der Inline-Version durch Compilerdirektiven.

Bei den Elementarmoduln handelt es sich um einfache Programmkerne aus der Bibliothek NUMATH [8], die eine den BLAS, Level 1, [10] entsprechende Funktion haben.

Tabelle 1 Rechenzeiten auf der IBM 3090 mit Vector Facility in MFLOPS

Routine	n = 400	n = 1000
(a) Gauß, Inline, Skalarmodus	7.2	7.4
(b) Gauß, Vektor-Assembler-BLAS-1	18.5	21.2
(c) Gauß, Inline, Direktiven	18.4	21.2
Rank-r LU Update, r = 4	37.2	46.6
Rank-r LU Update, r = 8	43.4	58.0
Rank-r LU Update, r = 16	43.8	61.0

Wir erkennen, daß die Rechenzeiten für die Versionen (b) und (c) übereinstimmen, d.h., daß der Compiler im wesentlichen die Effektivität einer auf getunten BLAS, Level 1, basierenden Implementation erreicht.
Das Verhältnis von Vektorlade- zu Vektorrechen-Operationen ist für Vektorrechner bei den BLAS-1-Moduln ungünstig, was zur Entwicklung der BLAS-2- und BLAS-3-Routinen [5], [6] und zu neuen Modifikationen des Gaußschen Algorithmus geführt hat.
J.J. Dongarra u.a. haben in [4] den jik-SDOT-, den jki-GAXPY- und den kji-SAXPY-Algorithmus vorgeschlagen. Eine Weiterentwicklung des kji-SAXPY-Verfahrens, auch Rank-r LU Update Algorithmus genannt, wurde in [13, 3] vorgestellt.
Im folgenden Abschnitt wird diese Methode beschrieben.
Effizienzvergleiche zwischen einer Implementation dieser Methode und den Versionen (b) und (c) zeigen die Vorteile des Rank-r LU Update Verfahrens für Vektorrechner mit Cache (siehe Tabelle 1).

3.2 Der Rank-r LU Update Algorithmus

Der Schritt von dem Gaußschen Verfahren (2 - 8) zum Rank-r LU Update Algorithmus besteht im Übergang von der Behandlung einer einzelnen Spalte (Zeile) zu Blöcken von r Spalten (Zeilen). Es sei angenommen, daß wir bereits bis zum Schritt k gekommen sind. Die einzelnen Schritte des Rank-r LU Update Algorithmus sind dann:

1. for $\kappa = k - r(1)k$ do
   - Pivotsuche in der Spalte κ
   - Zeilenvertauschung für die Spalten $k - r$ bis k
   - Berechnung der Eliminationsfaktoren in der Spalte κ
   - Updating für die Spalten $\kappa + 1$ bis k

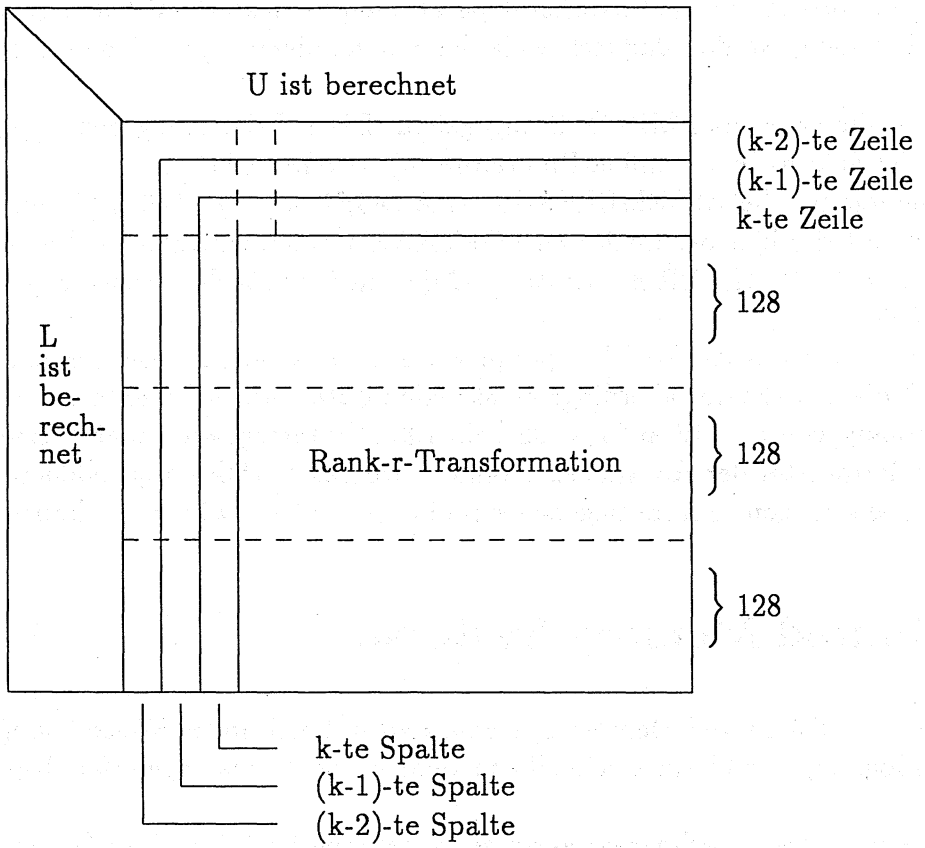
   Nach diesem Schritt ist ein Teil von L berechnet.
   Die Zeilenvertauschung wird also nur für die Spalten $k - r$ bis k vorgenommen. Die Vertauschung für die restlichen Spalten wird auf den Schritt 2 verschoben, um die Matrix nicht zu oft in den Cache laden zu müssen.

2. Updating und Transformation
   - Updating der Zeilen $k - r + 1$ bis k
     Nach diesem Schritt ist ein Teil von U berechnet.
   - Rank-r Transformation für den Rest der Matrix

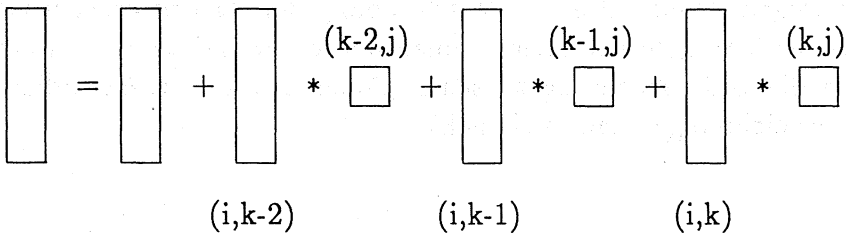
3. Aufhebung von Zeilenvertauschungen
   In diesem Schritt wird die temporäre Zeilenvertauschung rückgängig gemacht, die im Schritt 1 für alle r Elemente der Pivotzeile gemacht wurde und die nur für die Elemente unterhalb der Diagonale benötigt wird.

Im allgemeinen ist r kein Teiler der Matrixordnung n. Die Dekomposition des Restes wird mit dem Standard-Gauß-Verfahren durchgeführt. In den Routinen zur Dekomposition der Restmatrix und für das Vorwärts- und Rückwärtseinsetzen werden Vektor-ASSEMBLER-Elementarmoduln der Bibliothek NUMATH verwendet.
Der Hauptteil der Rechenzeit wird im Schritt 2 benötigt. Die vorbereitenden Operationen, das Updating der Zeilen $k - r$ bis k und die innere Schleife der Rank-r Transformation wurden entrollt.
Eine Momentaufnahme der Rank-r Transformation mit dem Unrolling zeigt Bild 1. Die Zahl 128 in der Zeichnung gibt die Registerlänge der Vektorregister an.

Bild 1 Momentaufnahme der Rank-r Transformation




```
C*VDIR: PREFER VECTOR
      do 120 i=k+1,n
         do 110 j=k+1,n
            a(i,j) = a(i,j) + a(i,k-2)*a(k-2,j)
                            + a(i,k-1)*a(k-1,j) + a(i,k)*a(k,j)
 110     continue
 120  continue
```



Durch die Verwendung der Blöcke und das Unrolling wird Arithmetik und Datenaustausch zwischen dem Cache und dem Zentralspeicher in großen Positionen ausgeführt und das Verhältnis von Vektorlade- zu Vektorrechen-Operationen verbessert. Mit den Daten, die sich im Cache befinden, werden dadurch viele der notwendigen Operationen auf einmal durchgeführt.
In Tabelle 1 sind die erreichten MFLOPS angegeben. Die Effizienzsteigerung gegenüber dem per Compiler vektorisierten Inline-Programm (c) ist offensichtlich.
Die Ausführungszeit für die ASSEMBLER-Routine DGEF aus der ESSL [7] wurde mit 100% angesetzt. Mit der hier besprochenen FORTRAN-Implementation bei Verwendung der oben erwähnten ASSEMBLER-Elementarmoduln wurden 92% der Leistung von DGEF erreicht, in [3] 83%.
Der Effektivitätsgewinn des Rank-r LU Update Verfahrens steigt mit wachsendem r. Für festes n läßt sich die Effektivität allerdings nicht beliebig steigern. So konnte für $n = 1000$ durch den Übergang von $r = 16$ auf $r = 32$ kein Effektivitätssgewinn mehr erzielt werden. Da r nicht Parameter der Routine sein kann, wäre das Verfahren gegebenenfalls als Polyalgorithmus auszulegen, um für gegebenes n ein optimales r wählen zu können.

4 Distributed Memory Systeme

Die Prozessoren der Distributed Memory Systeme sind in bestimmter Weise konfiguriert. Vielfach ist es möglich, virtuell unterschiedliche, dem Algorithmus angepaßte Topologien zu verwenden.
Die Programmierung dieser Rechner erfordert zunächst eine Verteilung der Daten auf die lokalen Speicher und Kommunikation zwischen den Prozessoren.
Parallele Algorithmen sind nach Möglichkeit so zu entwerfen, daß über den gesamten Rechenprozeß eine ausgeglichene Lastverteilung besteht und die im Verhältnis zu den arithmetischen Operationen teure Kommunikation gering gehalten wird.
In diesem Artikel wird ein frei konfigurierbares Transputersystem zugrundegelegt.
Implementationen des Gaußschen Algorithmus für Distributed Memory Systeme sind beispielsweise in [11], [12] und [2] und speziell für Transputersysteme in [1] und [9] beschrieben worden. In [9] wurde der Gaußsche Algorithmus für eine blockzyklische Datenaufteilung formuliert, die hier erweitert und auch im Zusammenhang mit dem Rank-r LU Update Algorithmus betrachtet werden soll.
Die in den vorherigen Abschnitten zur Beschreibung der Algorithmen verwendeten Bezeichnungen können hier nicht voll übernommen werden, weil die Beschreibung des Transputernetzes und die Unterscheidung zwischen globalen und lokalen Indizes ein starkes Anwachsen an Bezeichnungen nach sich zieht.

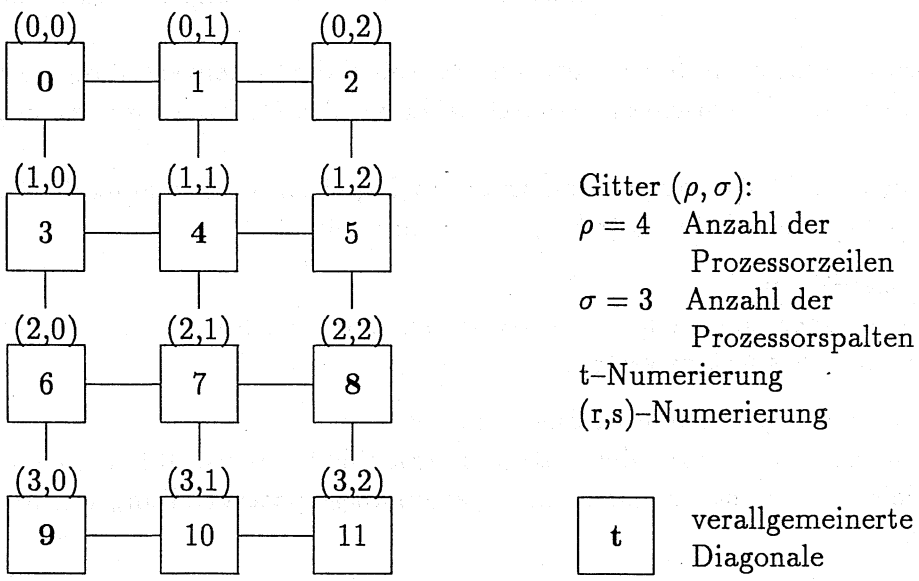
4.1 Konfiguration

Als Grundkonfiguration wird ein zweidimensionales Prozessornetz der Dimension (ρ, σ) (siehe Bild 2) gewählt, d.h. die Prozessoren sind nur mit ihren Nachbarn über Kom-

munikationskanäle verbunden. Die Randprozessoren können auf unterschiedliche Weise vernetzt sein:

- not wrapped around (Grid)
- Nord-Süd wrapped around
- Ost-West wrapped around
- Nord-Süd und Ost-West wrapped around (Torus)

Bild 2 Konfiguration



Aus programmtechnischen Gründen werden zwei Arten der Nummerierung verwendet. In der zeilenweisen Nummerierung erhält jeder Prozessor einen Identifikator t. Durch die Zeilen-Spalten-Nummerierung wird jedem Prozessor seine Zeilennummer r und seine Spaltennummer s zugewiesen. Der Identifikator wird mit (r, s) bezeichnet.
Beide Nummerierungen hängen wie folgt zusammen:

$$t = r\sigma + s \tag{13}$$

$$r = \left[\frac{t}{\sigma}\right] \qquad s = mod(t, \sigma) \tag{14}$$

Durch

$$mod(r, min(\rho, \sigma)) = mod(s, min(\rho, \sigma)) \tag{15}$$

ist eine verallgemeinerte Diagonale definiert (siehe Bild 2).
Sie spielt in bestimmten Fällen für das Vorwärts- und Rückwärtseinsetzen eine Rolle, das hier nicht erörtert wird.

4.2 Datenaufteilung

Da ein Distributed Memory System nur über lokale Speicher verfügt, müssen Daten, im vorliegenden Fall die Matrix A und die rechte Seite b aus (1), auf die einzelnen Knoten verteilt werden.
Um Aufgaben möglichst hoher Dimension lösen zu können, ist eine gleichmäßige Verteilung der Daten anzustreben. Andererseits sollte die Datenverteilung auch so erfolgen, daß die Lastverteilung auf die einzelnen Prozessoren während des Lösungsprozesses ausgeglichen ist.
Weil Kommunikation teurer als Rechenleistung ist, sollte die Datenverteilung so vorgenommen werden, daß der Nachrichtenaustausch während des Rechenprozesses so gering wie möglich ist.
Da der Algorithmus zur LU-Dekomposition zeilen- und spaltenweise ablaufende Prozesse enthält, ist eine orthogonale Datenaufteilung von A sinnvoll.

Orthogonale Datenaufteilung
Jede Zeile von A ist über eine Zeile von Prozessoren und jede Spalte von A über eine Spalte von Prozessoren des Netzes verteilt.

Die formulierten Forderungen führen zu Zielkonflikten.
Eine blockzyklische Datenaufteilung der Matrix A ermöglicht eine variable Anpassung an die genannten Ziele. Sie sichert vor allem eine gleichmäßige Lastverteilung während des Lösungsprozesses.

Blockzyklische Datenaufteilung
Die Matrix A wird bis auf Restblöcke in Blöcke E der Dimension (p, q) eingeteilt. Die Blöcke und Restblöcke werden zyklisch auf das Prozessorgitter abgebildet.

Die blockzyklische Datenaufteilung ist orthogonal.
Durch die blockzyklische Aufteilung zerfällt A in $\kappa \cdot \lambda$ Windows (siehe Bild 3)

$$W_{\mu\nu}, \quad \mu = 0(1)\kappa - 1, \quad \nu = 0(1)\lambda - 1. \tag{16}$$

Der Allgemeinheit halber sei hier $A \in \mathrm{R}^{m \times n}$ gewählt. Dann berechnen sich κ und λ wie folgt:

$$\kappa = \left\lceil \frac{m}{\rho p} \right\rceil, \qquad \lambda = \left\lceil \frac{n}{\sigma q} \right\rceil \tag{17}$$

mit

$$\left\lceil \frac{\Gamma}{\Lambda} \right\rceil = \begin{cases} \left[\frac{\Gamma}{\Lambda}\right], & wenn \quad \Gamma \ mod \ \Lambda = 0 \\ \left[\frac{\Gamma}{\Lambda}\right] + 1 & sonst \end{cases}$$

Jedes Window trägt mit einem Block $E^{r,s}_{\mu,\nu}$ zu der Teilmatrix $C^{r,s}$ (18,19) bei, die sich im lokalen Speicher des Prozessors (r,s) befindet.

Teilmatrizen von A

$$C^{r,s} = \left\{ E^{r,s}_{\mu,\nu} \right\}_{\mu=0(1)\kappa-1, \ \nu=0(1)\lambda-1}, \quad r = 0(1)\rho - 1, \quad s = 0(1)\sigma - 1 \tag{18}$$

mit

$$E^{r,s}_{\mu,\nu} \in \mathrm{R}^{p \times q} \qquad \text{oder} \qquad E^{r,s}_{\mu,\nu} \in \mathrm{R}^{\overline{p} \times \overline{q}} \qquad ((\overline{p}, \overline{q}) \text{ siehe } (22)). \tag{19}$$

Restblöcke von A

Wenn ρp kein Teiler von m bzw. σq kein Teiler von n ist, bleiben bei der Zerlegung von A in Blöcke $E_{(p,q)}$ u Restzeilen und v Restspalten übrig.
Die u Restspalten lassen sich so aufteilen, daß die ersten α Prozessorzeilen jeweils $g+1$ und die letzten $\rho - \alpha$ Prozessoren jeweils g zusätzliche Matrixzeilen erhalten (siehe (20)).
Die v Restspalten lassen sich so aufteilen, daß die ersten β Prozessorspalten jeweils $h+1$ und die letzten $\sigma - \beta$ Prozessorspalten jeweils h zusätzliche Matrixspalten erhalten (siehe (21)).
Offenbar gelten die Beziehungen

$$u = m \ mod \ (\rho p), \quad g = \left[\frac{u}{\rho}\right], \quad \alpha = u \ mod \ \rho \tag{20}$$

$$v = n \ mod \ (\sigma q), \quad h = \left[\frac{v}{\sigma}\right], \quad \beta = v \ mod \ \sigma \tag{21}$$

Die Restzeilen und -spalten der Matrix A definieren die Restblöcke

$$E_{\mu,\nu}^{r,s} \in \mathrm{R}^{\overline{p}\times\overline{q}} \qquad \text{mit} \qquad \begin{cases} \overline{p} = \begin{cases} g+1 & ,wenn \quad r < \alpha \\ g & ,wenn \quad r \geq \alpha \end{cases} \\ \\ \overline{q} = \begin{cases} h+1 & ,wenn \quad s < \beta \\ h & ,wenn \quad s \geq \beta \end{cases} \end{cases} \tag{22}$$

Im Bild 3 ist eine blockzyklische Aufteilung einer Matrix $A \in \mathrm{R}^{n\times n}$, $n = 20$, auf einem Transputernetz $(\rho, \sigma) = (4, 3)$ dargestellt. Die Dimension der Blöcke beträgt $(p, q) = (3, 3)$, die der Restblöcke $(\overline{p}, \overline{q}) = (3, 1)$, $(2, 3)$ oder $(2, 1)$.
Die Zeilen- und Spaltennummerierung von A ist an den Rändern angegeben. Die Ziffern in den Kästchen, die die zugehörigen Blöcke $E_{\mu,\nu}^{r,s}$ von A enthalten, kennzeichnen die Transputer (t-Nummerierung, siehe (13)). Die Windows $W_{\mu,\nu}$, $\mu = 0, 1$, $\nu = 0, 1, 2$ sind durch Doppellinien voneinander abgegrenzt.

Bild 3 Datenaufteilung

	0 1 2	3 4 5	6 7 8	9 10 11	12 13 14	15 16 17	18	19
0 1 2	0	1	2	0	1	2	0	1
3 4 5	3	4	5	3	4	5	3	4
6 7 8	6	7	8	6	7	8	6	7
9 10 11	9	10	11	9	10	11	9	10
12 13	0	1	2	0	1	2	0	1
14 15	3	4	5	3	4	5	3	4
16 17	6	7	8	6	7	8	6	7
18 19	9	10	11	9	10	11	9	10

Die Teilmatrizen $C^{r,s}$ aus (18) von A, die sich im Prozessor (r, s) befinden, ergeben sich aus A durch Schrumpfung von

$$\overline{C}^{r,s} = P^r A Q^s \tag{23}$$

um die Nullzeilen und -spalten.
Die Anzahl der Zeilen bzw. Spalten von $C^{r,s} \in \mathrm{R}^{\overline{m}\times\overline{n}}$ beträgt

$$\overline{m} = \left[\frac{m}{\rho p}\right] p + \overline{p} \tag{24}$$

$$\overline{n} = \left[\frac{n}{\sigma q}\right] q + \overline{q} \tag{25}$$

P^r und Q^s sind Diagonalmatrizen der Form

$$P^r = \{p_i^r\}_{i=0(1)m-1}, \quad p_i^r \quad \text{Diagonalelemente von} \quad P^r \tag{26}$$

mit

$$p_i^r = \begin{cases} 1, & wenn \quad i = rp + k, \quad k = 0(1)p - 1 \\ 1, & wenn \quad i = l(\rho p) + k, \quad l = 1(1)\left[\frac{m}{\rho p}\right], \quad k = 0(1)\overline{p} - 1 \\ 0 & sonst \end{cases} \tag{27}$$

und

$$Q^s = \left\{q_j^s\right\}_{j=0(1)n-1}, \quad q_j^s \quad \text{Diagonalelemente von} \quad Q^s \tag{28}$$

mit

$$q_j^r = \begin{cases} 1, & wenn \quad j = sq + k, \quad k = 0(1)q - 1 \\ 1, & wenn \quad j = l(\sigma q) + k, \quad l = 1(1)\left[\frac{n}{\sigma q}\right], \quad k = 0(1)\overline{q} - 1 \\ 0 & sonst \end{cases} \tag{29}$$

Beispiel

Im Bild 3 ist die Aufteilung der quadratischen Blöcke $E_{\mu,\nu}^{r,s} \in \mathrm{R}^{3\times3}$, und der Restblöcke der Matrix $A \in \mathrm{R}^{n\times n}$, $n = 20$, auf ein rechteckiges Prozessorgitter der Dimension $(\rho, \sigma) = (4, 3)$ dargestellt.
Die Teilmatrix

$$C^{0,0} = \begin{pmatrix} E_{0,0}^{0,0} & E_{0,1}^{0,0} & E_{0,2}^{0,0} \\ E_{1,0}^{0,0} & E_{1,1}^{0,0} & E_{1,2}^{0,0} \end{pmatrix}$$

der Matrix A aus Bild 3, die sich im Prozessor $(0, 0)$ befindet, setzt sich wie folgt zusammen:

$$C^{0,0} = \begin{pmatrix} a_{0,0} & a_{o,1} & a_{0,2} & a_{0,9} & a_{0,10} & a_{0,11} & a_{0,18} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,9} & a_{1,10} & a_{1,11} & a_{1,18} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,9} & a_{2,10} & a_{2,11} & a_{2,18} \\ a_{12,0} & a_{12,1} & a_{12,2} & a_{12,9} & a_{12,10} & a_{12,11} & a_{12,18} \\ a_{13,0} & a_{13,1} & a_{13,2} & a_{13,9} & a_{13,10} & a_{13,11} & a_{13,18} \end{pmatrix}$$

$$= \begin{pmatrix} c_{0,0} & c_{o,1} & c_{0,2} & c_{0,3} & c_{0,4} & c_{0,5} & c_{0,6} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} & c_{1,6} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} & c_{2,6} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} & c_{3,6} \\ c_{4,0} & c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} & c_{4,6} \end{pmatrix}$$

$C^{0,0}$ ergibt sich, wie in (23) bis (29) aufgeschrieben, aus A durch Schrumpfung von

$$\overline{C}^{0,0} = P^0 A Q^0$$

um die Nullzeilen und -spalten mit

$$P^0 = diag(1,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0)$$

$$Q^0 = diag(1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,0).$$

P^0 und Q^0 sind Diagonalmatrizen. Die zugehörigen Elemente sind in den Klammern von diag angegeben.

Spezialfälle

Durch die freie Wahl der Parameter (ρ, σ) und (p, q) werden wichtige Spezialfälle erfaßt.

(a) $(\rho, \sigma) = (1, \sigma)$, Ring bzw. Pipe,
(p, q)=(n, q).
Die Spaltenblöcke von A werden zyklisch auf dem Ring verteilt. Im Falle $q = 1$ reduzieren sich die Blöcke zu Spalten.

(b) $(\rho, \sigma) = (\rho, 1)$, Ring bzw. Pipe,
(p, q)=(p, n).
Die Zeilenblöcke von A werden zyklisch auf dem Ring (der Pipe) verteilt. Im Falle $p = 1$ reduzieren sich die Blöcke zu Zeilen.

(c) $(\rho, \sigma) = (\rho, \rho)$, quadratisches Netz,
$(p, q) = (p, p)$, quadratische Blöcke

(d) $(\rho, \sigma) = (\rho, \rho)$
$(p, q) = (1, 1)$
Die Marixelemente sind zyklisch über das quadratische Prozessornetz verteilt.

4.3 Indextransformationen

Durch die blockzyklische Aufteilung der Matrix

$$A = (a_{i,j}), \quad i = 0(1)m-1, \quad j = 0(1)n-1 \tag{30}$$

zerfällt A in die Teilmatrizen

$$C^{r,s} = (c^{r,s}_{k,l}), \quad k = 0(1)\overline{m}-1, \quad l = 0(1)\overline{n}-1, \tag{31}$$

die sich in den lokalen Speichern der Prozessoren

$$(r, s), \quad r = 0(1)\rho-1, \quad s = 0(1)\sigma-1 \tag{32}$$

befinden.
Für die Durchführung der LU-Dekomposition, die lokal auf den Prozessoren abläuft, sind daher verschiedene Indextransformationen von Bedeutung, die den Zusammenhang von globaler und lokaler Information herstellen.

Prozessor-Identifikation

Der Identifikator r der Prozessorzeile kann, wie man leicht nachprüft, wie folgt berechnet werden:

$$\gamma = \left[\frac{m}{\rho p}\right](\rho p) \tag{33}$$

a) $i < \gamma$:

$$r = mod\left(\left[\frac{i}{p}\right], \rho\right) \tag{34}$$

b) $i \geq \gamma$:

$$i_1 = i - \gamma, \quad i_2 = i_1 - \alpha(g+1) \quad (siehe(20)) \tag{35}$$

$$r = \begin{cases} \left[\frac{i_1}{g+1}\right], & wenn \quad \left[\frac{i_1}{g+1}\right] < \alpha \\ \alpha + \left[\frac{i_2}{g}\right], & wenn \quad \left[\frac{i_1}{g+1}\right] \geq \alpha \end{cases} \tag{36}$$

Entsprechend ergibt sich der Identifikator s der Prozessorspalte:

$$\delta = \left[\frac{n}{\sigma q}\right](\sigma q) \tag{37}$$

a) $\underline{j < \delta :}$

$$s = mod\left(\left[\frac{j}{q}\right], \sigma\right) \tag{38}$$

b) $\underline{j \geq \delta :}$

$$j_1 = j - \delta, \quad j_2 = j_1 - \beta(h+1) \quad (siehe(21)) \tag{39}$$

$$s = \begin{cases} \left[\frac{j_1}{h+1}\right], & wenn \quad \left[\frac{j_1}{h+1}\right] < \beta \\ \beta + \left[\frac{j_2}{h}\right], & wenn \quad \left[\frac{j_1}{h+1}\right] \geq \beta \end{cases} \tag{40}$$

Transformation der globalen Indizes (i, j) in die lokalen Indizes (k, l)

Die Transformation wird für $i \rightarrow k$ angegeben. Sie läßt sich formal auf die Transformation $j \rightarrow l$ übertragen.

a) $\underline{i < \gamma :}$

$$k = i - p(r + \tau(\rho - 1)) \tag{41}$$

mit

$$\tau = \left[\frac{i}{\rho p} - \frac{r}{\rho}\right] \tag{42}$$

b) $\underline{i \geq \gamma :}$

$$k = \left[\frac{m}{\rho p}\right] p + i_3 \tag{43}$$

mit (siehe (35))

$$i_3 = \begin{cases} i_1 - \left[\frac{i_1}{g+1}\right](g+1), & wenn \quad \left[\frac{i_1}{g+1}\right] < \alpha \\ i_2 - \left[\frac{i_2}{g}\right] g, & wenn \quad \left[\frac{i_1}{g+1}\right] \geq \alpha \end{cases} \tag{44}$$

Transformation der lokalen Indizes (k,l) in die globalen Indizes (i,j)

Die Transformation wird für $k \rightarrow i$ angegeben. Sie kann formal auf die Transformation $l \rightarrow j$ übertragen werden.
Gemäß (24,25) gilt $C^{r,s} \in \mathrm{R}^{\overline{m} \times \overline{n}}$.

Mit

$$\psi = \left[\frac{\overline{m}}{p}\right] p \qquad (siehe(24)) \tag{45}$$

berechnet sich i wie folgt aus k:

a) $k < \psi$:

$$i = \left(\left[\frac{k}{p}\right] \rho + r\right) p + k_p, \quad k_p = mod(k,p) \tag{46}$$

b) $k \geq \psi$:

$$i = i_4 + i_5 \tag{47}$$

mit

$$i_4 = \left[\frac{\overline{m}}{p}\right] (\rho p) \tag{48}$$

$$i_5 = \begin{cases} r(g+1) + k - \psi, & wenn \quad r < \alpha \\ (r-\alpha)g + \alpha(g+1) + k - \psi, & wenn \quad r \geq \alpha \end{cases} \tag{49}$$

4.4 Die LU-Dekomposition bei blockzyklischer Datenaufteilung

Die Durchführung der Standard-LU-Dekomposition (siehe (6-8)) erfordert bei blockzyklischer Datenverteilung von Blöcken der Dimension (p, q) auf ein Prozessornetz (ρ, σ) (siehe (17-29)) einige Modifikationen.
Es sei angenommen, daß wir in unserem Algorithmus bereits bis zur Stufe $\overline{j}$ gekommen sind, d.h. für die Stufe $\overline{j}$ sind im wesentlichen die folgenden 4 Schritte zu machen:

- Bestimmung des Pivotelementes
- Zeilenvertauschung
- Berechnung der negativen Eliminationsfaktoren
- Rank-one Updating für die Restmatrix

Bestimmung des Pivotelementes

Das zur Stufe $\overline{j}$ gehörige Element $a_{\overline{j},\overline{j}} \in A$ befinde sich im Prozessor $(r_{\overline{j}}, s_{\overline{j}})$ (siehe (13,14)), der Stufenprozessor genannt werde. Entsprechend werde die Prozessorzeile $r_{\overline{j}}$, die die Stufenzeile $\overline{j}$ enthält, als Stufen-Prozessorzeile und die Prozessorspalte $s_{\overline{j}}$, die die Stufenspalte $\overline{j}$ enthält und in der das Pivotelement zu suchen ist, als Stufen-Prozessorspalte bezeichnet. Der Stufenprozessor wird mit Hilfe der Beziehungen (33-40) ermittelt.
In jedem Prozessor $(r, s_{\overline{j}})$, $r = 0(1)\rho - 1$, der Stufen-Prozessorspalte wird simultan das lokale Pivotelement $\pi_{\overline{j}}$ des Teils der Spalte $\overline{j}$ von A bestimmt, der sich in der Teilmatrix $C^{r,s_{\overline{j}}}$ befindet.
Dazu ist der Spaltenindex $\overline{j}$ von $A = (a_{i,j})$ in den zugehörigen Spaltenindex $\overline{l}$ von $C^{r,s_{\overline{j}}} = (c_{k,l}^{r,s_{\overline{j}}})$ zu transformieren. Das geschieht mit Hilfe von Formeln, die den in (41-44) aufgeschriebenen Beziehungen für die Umwandlung von $i \rightarrow k$ entsprechen. Der lokale Pivotindex k_{lok} von $C^{r,s_{\overline{j}}}$ ist schließlich in den zugehörigen Index i_{lok} von A zu transformieren (siehe (45-49)).
Die lokalen Pivotelemente $\pi_{r,s_{\overline{j}}}$, $r \neq r_{\overline{j}}$, $r = 0(1)\rho - 1$, und die zugehörigen lokalen Pivotindizes werden zum Stufenprozessor $(r_{\overline{j}}, s_{\overline{j}})$ gesandt.
Dort wird das globale Pivotelement bestimmt.
Eine Modifikation dieses Vorgehens wird im Abschnitt 4.5 Kommunikation behandelt.
Um den Kommunikations-Overhead herabzusetzen, wird die Bestimmung des globalen Pivotelementes wahlweise mit Schwellenpivotisierung [9] durchgeführt, die für praktische Fälle oft ausreichend ist [14].
Vorgegeben sei die Schwelle $0 \leq \omega \leq 1$.
Wenn für alle $r \neq r_{\overline{j}}$

$$| \pi_{r_{\overline{j}},s_{\overline{j}}} | \geq \omega \, | \pi_{r,s_{\overline{j}}} | \tag{50}$$

gilt, ist die Stufen-Prozessorzeile auch auch die Pivot-Prozessorzeile, d.h. die Prozessorzeile, die die Pivotzeile von A zur Stufe $\overline{j}$ enthält, und eine Zeilenvertauschung ist nur in dieser Prozessorzeile notwendig. Eine Inter-Prozessor-Kommunikation findet nicht statt.
Wenn $\omega = 1$, entartet die Schwellenpivotisierung zur üblichen partiellen Pivotisierung.
Für $\omega = 0$ ergibt sich eine lokale Pivotisierung, bei der keine Inter-Prozessor-Kommunikation durchzuführen ist, weil (50) immer erfüllt ist.
Nach der Bestimmung des globalen Pivotelementes ist der globale Pivotindex i_{glob} im Stufenprozessor bekannt. Von dort wird er an alle anderen Prozessoren gesandt, damit alle Knoten darüber informiert sind, welche Zeilen zu vertauschen sind.
Mit Hilfe des globalen Pivotindexes i_{glob} wird die Pivot-Prozessorzeile r_{glob} ermittelt (siehe (33-36)).

Zeilenvertauschung

In der Pivot-Prozessorzeile werden 2 Vektoren r_1 und r_2 zusammengestellt.
r_2 besteht aus dem Pivotelement und dem Teil der Pivotzeile, die für das Rank-one Updating der Restmatrix benötigt wird. Dazu ist der globale Pivotindex i_{glob} bezüglich $A = (a_{i,j})$ in den zugehörigen Index k_{glob} bezüglich $C^{r_{glob},s_{\overline{j}}} = (c_{k,l}^{r_{glob},s_{\overline{j}}})$ zu transformieren (siehe (41-44)).
Der Rest der Zeile wird in r_1 zusammengefaßt.
r_2 wird über die Prozessorspalten an alle Prozessorzeilen gesandt.
r_1 wird an die Stufen-Prozessorzeile geschickt.
Die gesamte $\overline{j}$-te Zeile von A - sie befindet sich in der Stufen-Prozessorzeile - wird an die Pivot-Prozessorzeile gesandt.
(Die Kommunikationen erübrigen sich, wenn (50) immer erfüllt ist.)
Die Vertauschung der Pivotzeile i_{glob} mit der Stufenzeile $\overline{j}$ kann nun vollzogen werden, indem in der Stufen-Prozessorzeile die Zeile $\overline{j}$ ($\overline{l}$ bzgl. $C^{r_{\overline{j}},s}$) durch r_1 und r_2 und in der Pivot-Prozessorzeile die Zeile i_{glob} durch die $\overline{j}$-te Zeile ersetzt werden.

Berechnung der negativen Eliminationsfaktoren

In der $\overline{j}$-ten Spalte von A werden die negativen Eliminationsfaktoren berechnet, indem die Elemente unterhalb der Diagonale mit dem reziproken Wert des Pivotelementes skaliert werden. Das findet in der Stufen-Prozessorspalte statt.
Die negativen Eliminationsfaktoren werden von der Stufen-Prozessorspalte an die Prozessoren der Prozessorzeilen gesandt.

Rank-one Updating für die Restmatrix

Auf Grund der blockzyklischen Aufteilung ist die Restmatrix, wenn man von der Endphase des Algorithmus absieht, über alle Knoten verteilt.
Das Rank-one Updating findet daher in allen Prozessoren auf den Teilmatrizen $C^{r,s} = (c_{k,l}^{r,s})$ statt.

Um festzustellen, welche Elemente von $C^{r,s}$ dem Rank-one Updating zu unterziehen sind, müssen ständig Indextransformationen von (k,l) nach (i,j) vorgenommen werden (siehe (45-49)).

4.5 Kommunikation

Der im vorigen Abschnitt beschriebene Algorithmus wurde für ein MultiCluster-2 System mit 32 Prozessoren vom Typ T800 mit rekonfigurierbarer Topologie unter dem Betriebssystem PARIX, Release 1.1, in ACE FORTRAN77 [15] implementiert.
Das Übersetzen der einzelnen Routinen und das Linken zu einem Hauptprogramm erfolgt auf dem Hostrechner (SUN SPARCstation).
Nach dem Start der Applikation wird das Hauptprogramm vom Host an alle angeforderten Knoten gesandt.
Das Programm arbeitet auf jedem Prozessor auf einer anderen Datenmenge (Datenparallelität).
ACE FORTRAN77 ist gegenüber FORTRAN77 um Virtual Topology Libraries erweitert, die den Aufbau virtueller Topologieen und die Kommunikation durch Bibliotheksaufruf gestatten. Durch den Aufruf spezieller Routinen der Virtual Topology Libraries kann auch festgestellt werden, auf welchem Prozessor sich das Programm befindet. Mit Hilfe dieser Information wird die notwendige funktionelle Parallelität über geeignete IF-THEN-ELSE-Anweisungen hergestellt, so daß auf den einzelnen Knoten unterschiedliche Teile des Programms wirksam werden können.
Insbesondere können unter Verwendung der Virtual Topology Libraries für die verschiedenen Kommunikationen im Algorithmus angepaßte unterschiedliche Topologien aufgebaut und für den Nachrichtenaustausch genutzt werden.

Kommunikation bei der Bestimmung des Pivotelementes

Für die Kommunikation bei der Bestimmung des Pivotelementes sind 2 unterschiedliche Topologien implementiert worden.

(a) Für das Senden der lokalen absoluten Maxima und Pivotindizes an den Stufenprozessor ist die in der linken Zeichnung von Bild 4 dargestellte Topologie verwendet worden.
Das Senden des globalen Pivotindexes vom Stufenprozessor an alle Prozessoren erfolgt über die in der mittleren Zeichnung von Bild 4 angegebene Farmer-Worker-Topologie.
Die Topologie (a) ist vom Stufenindex $\overline{j}$ abhängig. Sie muß daher für jede Stufe neu erzeugt werden.

(b) Der Datenaustausch für die Pivotisierung erfolgt auf der Basis der 2D-Torus-Topologie (siehe auch Abschnitt 4.1 Konfiguration), die nur einmal generiert werden muß und

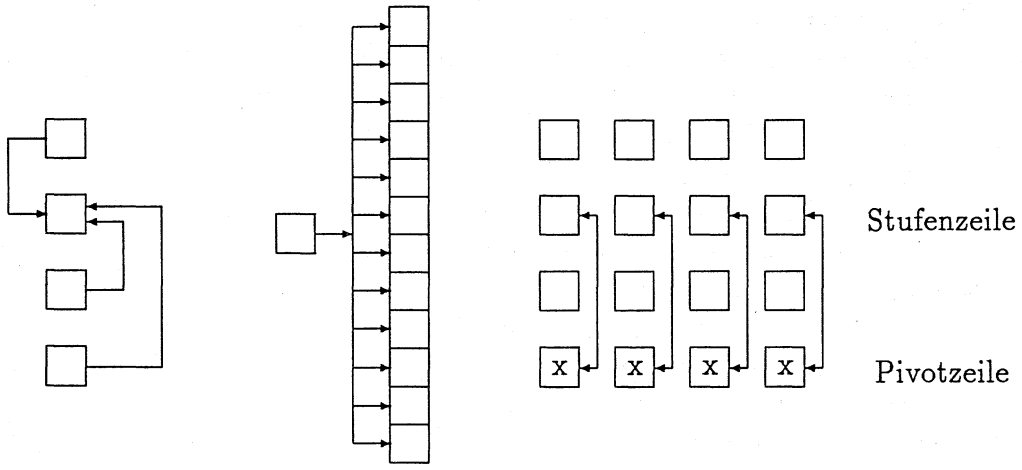
die auch für die Kommunikation zwischen den Schritten Zeilenvertauschung und Rank-one Updating benutzt wird.
Eine Stufen-Prozessorspalte stellt innerhalb der Torus-Topologie einen Ring dar, in dem die Lage des Stufenprozessors so interpretiert werden kann, daß die Anzahl der Prozessoren zu beiden Seiten gleich groß ist, wenn ρ ungerade, oder sich um 1 unterscheidet, wenn ρ gerade ist.
Die lokalen Pivotelemente und -indizes werden von den äußeren Prozessoren zu beiden Seiten des Stufenprozessors jeweils zu ihren Nachbarn in Richtung des Stufenprozessors gesandt. Weiter geschickt werden jeweils das dem Absolutbetrage nach größere Element und der zugehörige Index.
Da jeweils 2 Datenübertragungen parallel ablaufen, sind so $(\rho - 1)/2$, ρ ungerade, Übertragungen notwendig.
Die Verteilung des globalen Pivotindexes erfolgt entsprechend nach einer umgekehrten Strategie.
In der Stufen-Prozessorspalte wird der Index vom Stufenprozessor aus nach beiden Seiten von Nachbar zu Nachbar versandt.
Alle Knoten der Stufen-Prozessorspalte senden dann den Index auf die gleiche Weise in Zeilenrichtung.
Im Falle eines quadratischen Transputernetzes $(\rho, \sigma) = (\rho, \rho)$, ρ ungerade, wären auf Grund der parallelen Kommunikationen ρ Übertragungen durchzuführen.

Für die kleinen verwendeten Knotenanzahlen (maximal 32) ergaben sich leichte Vorteile für die Topologie (a).

Kommunikation bei der Zeilenvertauschung und dem Rank-one Updating

Die Kommunikation für die Zeilenvertauschung erfolgt über die in der rechten Zeichnung (Bild 4) dargestellten Topologie, die für jede Stufe neu zu generieren ist.
Der für das Rank-one Updating benötigte Teil der Pivotzeile wird von der Pivot-Prozessorzeile in Spaltenrichtung an alle Knoten gesandt.
Um das Rank-one Updating durchführen zu können, sind die negativen Eliminationsfaktoren von der Stufen-Prozessorspalte an die Knoten der Prozessorzeilen zu senden.
Beide Kommunikationen erfolgen im Rahmen einer Gitter- oder Torus-Topologie (siehe Abschnitt 4.1 Konfiguration).
Die auf Grund der kürzeren Wege beim Torus erwarteten Verbesserungen traten bei der zu Verfügung stehenden kleinen Prozessorzahl nicht ein.

Bild 4 Topologien



4.6 Bemerkungen

Im Unterschied zu HELIOS [16] ermöglicht das neue auf große Prozessorzahlen ausgerichtete Betriebssystem PARIX mit ACE FORTRAN77 die Formulierung von Algorithmus, Topologie und Kommunikation in einem Programm.
Der mit PARIX1.0 ausgelieferte ACE FORTRAN Compiler verfügte über keine Optimierungsstufe. Die mit PARIX1.1 bereitgestellte Optimierungsstufe brachte für die vorgestellte LU-Dekomposition eine Verbesserung der Effizienz von 20%, stellt aber noch nicht die endgültige Fassung dar, so daß hier noch keine quantitativen Angaben zur Effizienz gemacht werden sollen.
Qualitativ zeigte sich, daß die Effizienz bei quadratischen Netzen (ρ, ρ) besser als bei rechteckigen Netzen (ρ, σ) ist.
Auch quadratische Blöcke der Dimension (p, p) ergaben günstigere Zeiten als rechteckige (p, q).
Auf Grund der guten Lastverteilung führte die zyklische Verteilung der einzelnen Matrixelemente, d.h. $(p, q) = (1, 1)$, zu den besten Ergebnissen (siehe auch [9]).
Trotzdem dürften die allgemeineren Auslegungen (ρ, σ), $\rho \neq \sigma$, und (p, q), $p \neq q$, auf Grund ihrer Anpassungsfähigkeit an zur Verfügung stehende Knotenanzahlen und das Datenlayout anderer Routinen ihre Berechtigung haben.
Bei einer Beschränkung auf quadratische Netze könnten beispielsweise bei einer Anlage mit 32 Prozessoren maximal 25 genutzt werden.
Das blockzyklische Datenlayout mit seiner zugehörigen Kommunikationsstruktur bildet eine gute Basis für die Implementation des Rank-r LU Update Verfahrens auf Distributed Memory Systemen.
Die Wahl größerer Blöcke ist dann mit einem Datentransport in größeren Portionen verbunden. Verbesserungen in der Effizienz durch Verwendung des Rank-r LU Update Verfahrens sind in Zusammenhang mit den angekündigten T9000-Transputer-Systemen zu erwarten, die pro Knoten auch über einen für das Rank-r LU Update Verfahren wichtigen Cache verfügen werden.

Literatur

[1] Bader,G., Gehrke,E., On the Performance of Transputer Networks for Solving Linear Systems of Equations, Parallel Computing 17 (1991), pp. 1397-1407.

[2] Benzoni,A., Sales,M.L., Concurrent Matrix Factorization on Workstation Networks, Proceedings of the IMA Conference on Parallel Computation, Oxford, U.K. (18-20 September 1991).

[3] Carnevali,P., Radicati di Brozolo,G., Robert,Y., Sguazzero,P., Efficient FORTRAN Implementation of the Gaussian Elimination and Householder Reduction Algorithms on the IBM 3090 Vector Multiprocessor, IBM ECSEC Technical Report.

[4] Dongarra,J.J., Gustavson,F.G., Karp,A., Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine, SIAM Review 12, 1 (1984), pp. 91-112.

[5] Dongarra,J.J., DuCroz,J.J., Hammarling,S.J., Hanson,R.J., An Extendet Set of FORTRAN Basic Linear Algebra Subprogams. ACM Transaction Math. Software 14 (1988), pp. 1-17.

[6] Dongarra,J.J., DuCroz,J.J., Duff,I., Hammarling,S.J., A Set of Level 3 Linear Algebra Subprogramms, Argonne National Laboratory, Mathematics and Computer Science Division, Preprint No.1, August 1988.

[7] Engineering and Scientific Subroutine Library, IBM Branch Office, Order No SC23-0184-0, (1989).

[8] Hebermehl,G., Binder,B., Brysch,R. ,Burmeister,W., Creutziger,J., Ehlert,J., Fiedler,O., Funke,R., Grohmann,U.R., Hübner,F.K., Jahnke,R., Kehl,U., Keusch,C., Kleemann,B., Knauf,L., Luber,H., Marx,M., Pohl,W., Reinhardt,G., Sandmann,H., Schalm,G., Schlundt,R., NUMATH - Software for Numerical Mathematics, Report R-MATH-05/89, Karl-Weierstraß-Institut für Mathematik, Berlin, 1989, 180 Seiten.

[9] Hoffmann,W., Potma,K., Threshold Pivoting in Gaussian Elimination to Reduce Inter-Processor Communication, Technical Report CS-91-05, Department of Mathematics and Computer Science, University of Amsterdam (1991).

[10] Lawson,C.L., Hanson,R.H., Kincaid,D.R., Krogh,F.T., Basic Linear Algebra Subprograms for FORTRAN Usage, ACM Transactions on Math. Software 5 (1979), pp. 303-323.

[11] Marrakchi,M., Robert,Y., Optimal Algorithm for Gaussian Elimination on an MIMD Computer, Parallel Computing 12 (1989), pp. 183-194.

[12] Rönsch,W., Lösung von Gleichungssystemen, KfK-SUPRENUM-Seminar, Tagungsbericht, Kernforschungszentrum Karlsruhe, (19.-20.10.1989).

[13] Robert,Y., Sguazzero,P., The LU Decomposition Algorithm and its Efficient FORTRAN Implementation on the IBM 3090 Vector Multiprocessor, IBM ESSEC Technical Report (March 1987).

[14] Trefethen,L.N., Schreiber,R.S., Average-case Stability of Gaussian Elimination, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335-360.

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

[16] Helios Operating System, Manual, Parsytec-Helios Rel. 910701, PARSYTEC GmbH (1991).

Veröffentlichungen des Instituts für Angewandte Analysis und Stochastik

Preprints 1992

1. D.A. Dawson and 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 and J.-M. Ghez: Spectral properties of one–dimensional Schrödinger operators with potentials generated by substitutions.
5. D.A. Dawson and 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.