# Portabilität und Adaption von Software der linearen Algebra für Distributed Memory Systeme

Georg Hebermehl, * Friedrich-Karl Hübner †

*Weierstraß-Institut für Angewandte Analysis and Stochastik, Mohrenstr. 39, D-10117 Berlin, Germany*

24. April 1996

*1991 Mathematics Subject Classification.*
65Y05, 65Y10.
*Keywords.*
Linear algebra, communication routines, message passing, portability.

*e-mail: hebermehl@wias-berlin.de, URL: http://hyperg.wias-berlin.de
†e-mail: huebner@wias-berlin.de, URL: http://hyperg.wias-berlin.de

**Zusammenfassung**

Durch die Verwendung anerkannter Grundbausteine für elementare Operationen der linearen Algebra und von Kommunikationsroutinen sowie üblicher blockzyklischer Datenverteilungen können Algorithmen höheren Levels weitgehend portabel und optimal auf Distributed Memory Computern adaptiert werden. Insbesondere wird über die Bereitstellung der Kommunikationsbibliothek BLACS für PARSYTEC-Rechner berichtet.

# Inhaltsverzeichnis



# Abbildungsverzeichnis



# Tabellenverzeichnis



# 1 Einführung

Die numerische Behandlung von linearen Gleichungssystemen, Augleichsproblemen und Eigenwertaufgaben stellt auch für Distributed Memory Computer eine Grundaufgabe dar, die effizient gelöst werden muß. Einerseits müssen daher hardware- und betriebssystembedingte Besonderheiten des jeweiligen Distributed Memory Computers zur Erreichung einer optimalen Performance ausgenutzt werden, andererseits wird aber auch die Portabilität der zur

Lösung von Grundaufgaben der linearen Algebra verwendeten Software über eine große Klasse von Rechnern angestrebt.

Die Programmierung der Distributed Memory Computer erfordert die Ausnutzung der Hierarchie der Speicher, eine Verteilung der Daten und die Kommunikation zwischen den Prozessoren.

Durch die Verwendung anerkannter Grundbausteine für elementare Operationen der linearen Algebra (BLAS Level 1,2 und 3, [9], [5], [6] ) und von Kommunikationsroutinen (BLACS, [7]) sowie üblicher blockzyklischer Datenverteilungen, die eine große Klasse von Dekompositionen für regelmäßige Datenstrukturen unterstützen, können Algorithmen höheren Levels, die die elementaren Bausteine benutzen, weitgehend portabel und optimal adaptiert werden. Die Grundbausteine sind für die verschiedenen Plattformen zu optimieren.

Ein wenig nutzerfreundliches Programmiermodell ist das Message Passing, das aber, aufgabenspezifisch richtig angewendet, zu den effizientesten Lösungen führt und daher für Basissoftware nach wie vor genutzt wird. Da sich die Kommunikationsmodelle der verschiedenen Distributed Memory Systeme wesentlich voneinander unterscheiden, ist die Verwendung weithin anerkannter Grundbausteine für den Datenaustausch für die Portabilität der Software unerläßlich.

Insbesondere wird hier über Erfahrungen bei der Bereitstellung adaptierter BLACS für PARSYTEC-Rechner unter dem Betriebssystem PARIX berichtet, die die Programmierung der Kommunikation vereinfachen und eine Voraussetzung für die Nutzung der Public Domain Software ScaLAPACK [3], [4] (Distributed Memory System Version von LAPACK [1]) bilden.

# 2 Block-partitionierte Algorithmen und Datendekomposition

Weil der Zugriff auf die Daten im oberen Level (Cache) der Speicherhierarchie schneller ist als der auf die Daten in den unteren Levels (lokaler Speicher, Speicher der anderen Knoten), werden in der linearen Algebra für Distributed Memory Systeme block-partitionierte Algorithmen verwendet, die auf Blöcken von Matrizen statt auf einzelnen Elementen operieren und die daher die Daten, die sich im Cache befinden, wiederholt nutzen und den Datentransport in größeren Portionen durchführen. Solche Operationen mit Teilmatrizen werden von den parallelen BLAS (PBBLAS [2]) durchgeführt. Die PBBLAS rufen die sequentiellen BLAS für die lokalen Rechenoperationen und die BLACS für die Interprozessor-Kommunikation auf und werden in den High Level Routinen von ScaLAPACK verwendet. Als Basis für diese Aufgabenklassen dient das SPMD Programmiermodell (Single Program Multiple Data).

Die ScaLAPACK-, die PBBLAS- und BLACS-Routinen operieren auf 2D-Feldern, weil Matrizen eine zentrale Rolle in der linearen Algebra spielen. Die zweidimensionalen Felder werden auf ein zweidimensionales Prozessornetz (siehe Abbildung 1) der Dimension $(\rho, \sigma)$ abgebildet. Eindimensionale Felder (Vektoren, Skalare) bzw. eindimensionale Gitter ergeben sich als Spezialfälle.

Die Prozessoren werden durch ihre Zeilen- und Spaltennummer $(\gamma, \delta)$ gekennzeichnet . Die Matrix $A \in \mathrm{R}^{m \times n}$ wird bis auf Restblöcke $E \in \mathrm{R}^{\overline{p} \times \overline{q}}$ in Blöcke $E \in \mathrm{R}^{p \times p}$ eingeteilt. Die Blöcke und Restblöcke werden zyklisch auf das Prozessorgitter abgebildet. Wenn $\rho p$ kein Teiler von $m$ bzw. $\sigma p$ kein Teiler von $n$ ist, bleiben bei der Zerlegung von $A$ in Blöcke $E \in \mathrm{R}^{p \times p}$ $u$ Restzeilen und $v$ Restspalten übrig. Die $u$ Restzeilen werden so aufgeteilt, daß die ersten $\alpha$ Prozessorzeilen jeweils $p$, die Prozessorzeile $\alpha + 1$ $p_1$ und $\rho - (\alpha + 1)$ Prozessorzeilen keine Restzeilen erhalten. Entsprechendes gilt für die Restspalten.

$$u = m \bmod (\rho p), \quad p_1 = u \bmod (p), \quad v = n \bmod (\sigma p), \quad q_1 = v \bmod (p),$$

$$\alpha = \left\lfloor \frac{u}{p} \right\rfloor, \quad \beta = \left\lfloor \frac{v}{p} \right\rfloor \quad .$$

Die Restzeilen und -spalten der Matrix $A$ definieren die Restblöcke:

$$E_{\mu,\nu}^{\gamma,\delta} \in \mathrm{R}^{\overline{p} \times \overline{q}} \quad \text{mit} \quad \overline{p} = \begin{cases} p, & wenn \quad \gamma < \alpha, \\ p_1, & wenn \quad \gamma = \alpha, \\ 0, & wenn \quad \gamma > \alpha, \end{cases} \overline{q} = \begin{cases} p, & wenn \quad \delta < \beta, \\ q_1, & wenn \quad \delta = \beta, \\ 0, & wenn \quad \delta > \beta. \end{cases}$$

Durch die blockzyklische Aufteilung zerfällt $A = (a_{i,j})$ in $\kappa \cdot \lambda$ Templates

$$W_{\mu\nu}, \quad \mu = 0(1)\kappa - 1, \quad \nu = 0(1)\lambda - 1, \quad \kappa = \left\lceil \frac{m}{\rho p} \right\rceil, \quad \lambda = \left\lceil \frac{n}{\sigma p} \right\rceil \quad .$$

Jedes Template $W_{\mu,\nu}$ trägt mit einem Block $E_{\mu,\nu}^{\gamma,\delta}$ zu der Teilmatrix $C^{\gamma,\delta} = (c_{k,l}^{\gamma,\delta})$ bei, die sich im lokalen Speicher des Prozessors $(\gamma, \delta)$ befindet:

$$C^{\gamma,\delta} = \{E^{\gamma,\delta}_{\mu,\nu}\}_{\mu=0(1)\kappa-1, \quad \nu=0(1)\lambda-1} \quad ,\gamma = 0(1)\rho - 1, \quad \delta = 0(1)\sigma - 1 \ .$$

Die Anzahl der Zeilen bzw. Spalten von $C^{\gamma,\delta} \in \mathrm{R}^{\overline{m}\times\overline{n}}$ beträgt

$$\overline{m} = \left[\frac{m}{\rho p}\right] p + \overline{p}, \quad \overline{n} = \left[\frac{n}{\sigma p}\right] p + \overline{q} \ .$$

Die beschriebene Datenverteilung wird Block Cyclic Data Distribution [4] genannt und unterstützt eine große Klasse von Datendekompositionen.

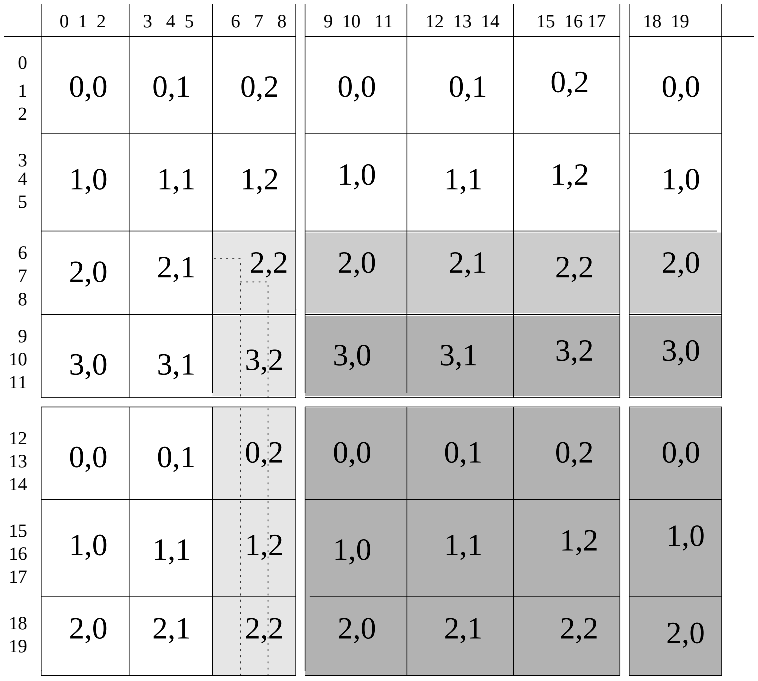


Abbildung 1: Block Cyclic Data Distribution

In Abbildung 1 ist die Datendekomposition für die verteilte blockorientierte LU-Dekomposition [8] einer Matrix $A \in \mathrm{R}^{n\times n}$, $n = 20$, auf einem Prozessornetz $(\rho, \sigma) = (4, 3)$ dargestellt. Die Identifikatoren $(\gamma, \delta)$ in den Kästchen kennzeichnen die Prozessoren, die die zugehörigen Blöcke $E^{\gamma,\delta}_{\mu,\nu}$ von $A$ enthalten. Die Templates $W_{\mu,\nu}$, $\mu = 0, 1$, $\nu = 0, 1, 2$, sind durch Doppellinien voneinander abgegrenzt.

Die Darstellung vermittelt auch einen Eindruck von der Nützlichkeit der Verwendung von 2D-Prozessornetzen für die Lösung linearer Gleichungssysteme. Wenn ein Distributed Memory Computer hardwareseitig als eindimensionales Prozessornetz aufgebaut ist, sollte eine logische Abbildung auf ein 2D-Gitter vorgenommen werden. Die blockzyklische Dekomposition der Koeffizientenmatrix garantiert bei der LU-Dekomposition neben der gleichmäßigen Verteilung der Daten auf die Knoten auch eine ausgeglichene Rechenlast für alle Prozessoren über den gesamten Lösungsprozeß.

# 3 Die Kommunikationsbibliothek BLACS

Die BLACS spielen in der Kommunikation eine ähnliche Rolle wie die BLAS als optimale Rechenroutinen. Im Gegensatz zu den BLAS, die auch direkt als FORTRAN- oder C-Routinen genutzt werden können, ist für die BLACS immer eine Adaption an die spezielle Plattform notwendig.

BLACS-Adaptionen sind erhältlich für PVM (Parallel Virtual Machine), Thinking Machine's (CM-5), die SP-Serie von IBM, die Intel Familie (iPSC2, iPSC/860, Touchstone Delta, Paragon XP/S), Cray, Meiko, für MPI (Message Passing Interface) und aufgrund der Implementation, über die hier berichtet wird, auch für PARSYTEC-Rechner (GC, PowerXplorer). Die BLACS sind in C implementiert. Interfaces ermöglichen den Aufruf aus FORTRAN77- und C-Programmen.

Die BLACS-Software besteht aus Kommunikationsprimitiven (Broadcast-Operationen, Knoten-Knoten-Kommunikation) und Combine Operationen für Daten der Typen Integer, Single Precision, Double Precision, Single Precision Complex und Double Precision Complex sowie Hilfsroutinen. Die Indizes $(\gamma, \delta)$ des Prozessornetzes werden als Ziel- oder Quellknoten verwendet.

Die Routinen operieren auf rechteckigen und trapezoidalen Matrizen. Rechteckige Matrizen sind als 2D-Felder abgespeichert und werden durch die Anzahl der Zeilen $M$, der Spalten $N$ und die führende Dimension $LDA$ beschrieben. Die führende Dimension gibt den Abstand zwischen aufeinanderfolgenden Spalten im Speicher an. Trapezoidale Matrizen werden durch $M, N, LDA$ und die zusätzlichen Parameter $UPLO$ (untere oder obere trapezoidale Matrix) und $DIAG$ beschrieben. Durch die Belegung von $DIAG$ kann festgelegt werden, ob die Diagonale bei der Kommunikation berücksichtigt werden soll. Dreiecksmatrizen sind spezielle trapezoidale Matrizen.

Bei den Broadcast-Operationen werden mehr als nur ein sendender und ein empfangender Knoten einbezogen. Durch einen Parameter $SCOPE$ kann festgelegt werden, ob die Operation über alle Knoten, eine Spalte oder eine Zeile des Netzes durchgeführt werden soll. Broadcast-Operationen können unter Verwendung spezieller Topologien (clockwise, counterclockwise und split ring, hypercube, tree, multi-path, master-slave) ausgeführt werden.

Combine Operationen (Maximum, Minimum, Summe) erzeugen Ergebnisse aus Daten, die sich auf verschiedenen Prozessoren befinden. Ein Beispiel wäre die Suche nach dem maximalen Element einer über mehrere Prozessoren verteilten Matrix.

Für die Ausführung des Datenaustausches müssen bestimmte parallele Umgebungen bereitgestellt werden, in denen die Kommunikationen stattfinden. In der BLACS-Bibliothek wird diese Aufgabe von Hilfsroutinen wahrgenommen. So wird beispielsweise im Betriebssystem PARIX die Gitterdimension durch das *run*-Kommando festgelegt. Der BLACS-Anwender erfährt durch den Aufruf der Hilfsroutine $BLACS\_PINFO$ die Anzahl der durch das *run*-Kommando bereitgestellten Prozessoren. Mit Hilfe der Routinen $BLACS\_GRIDINIT$ oder $BLACS\_GRIDMAP$ können in dieser Prozessormenge verschiedene Prozessornetze, die sich auch überlappen dürfen, definiert werden. Sie werden durch den Parameter $ICONTXT$ beschrieben. Durch den Aufruf von $BLACS\_GRIDEXIT$ können diese Netze wieder freigegeben werden, um Ressourcen zu sparen.

Parallelrechner unterscheiden sich wesentlich in der Art der Kommunikation. Man spricht von einer Globally-Blocking-Kommunikation, wenn die Anweisungen nach der *Send*-Operation erst dann ausgeführt werden, wenn die versendeten Daten von der zugehörigen *Receive*-Operation empfangen worden sind. Die PARSYTEC-Rechner haben diese Eigenschaft. Diese Art der Kommunikation kann zu Deadlocks führen.

Bei der Non-Blocking-Kommunikation können die Operationen nach den Kommunikationsanweisungen unabhängig von der Beendigung der Datenübertragung abgearbeitet werden. Das ermöglicht die Überlappung von Kommunikation und Rechnung, verlangt aber erhöhte Aufmerksamkeit in der Programmierung, um nicht auf falsche Daten zuzugreifen.

Weil in Programmen höheren Levels, wie den ScaLAPACK-Routinen, die Vermeidung möglicher Deadlocks sehr schwierig ist, unterstützt die BLACS-Bibliothek die Locally-Blocking-Kommunikation, bei der die Anweisungen nach der *Send*-Operation unabhängig von der Beendigung der Datenübertragung abgearbeitet werden. Für Distributed Memory Computer, die diese Eigenschaft hardwaremäßig nicht aufweisen, ist die Locally-Blocking-Kommunikation durch Pufferung zu simulieren, wenn mögliche Deadlocks in den Anwenderprogrammen vermieden werden sollen.

Die Übertragung der Daten in den BLACS wird für PARSYTEC-Rechner unter Verwendung der zum Betriebssystem PARIX gehörenden Kommunikationskommandos $PutMessage$ und $GetMessage$ realisiert, die die Daten in Portionen zu 1024 Bytes mit Pufferung senden bzw. empfangen, ohne daß die Abarbeitung der Kommandos nach dem Senden nicht ausgeführt wird, solange die Portionen auf der Gegenseite noch nicht entgegengenommen worden sind. Die Dimension der Felder beträgt $LDA * N$. Nur die in diesem Feld abgespeicherte Matrix der Dimension $M * N$ wird, um die Kommunikationszeiten zu reduzieren, tatsächlich übertragen. Sie wird dazu auf der sendenden und der empfangenden Seite in einem Puffer zwischengespeichert.

Die Deadlock-freie Programmunterstützung hat zusammen mit dem ohnehin vorhandenem Overhead der BLACS ihren Preis, wie Vergleiche mit 2 verschiedenen PARIX-Kommandos zeigen. Die PARIX-Anweisungen $SendLink$ und $RecvLink$ bewirken eine synchrone Kommunikation (globally blocking) auf der Basis vordefinierter Links. Die ebenfalls synchronen Kommandos $SendNode$ und $RecvNode$ benötigen lediglich die Ziel- bzw. Quelladressen.

In Anlehnung an [10] wird bei den Messungen der Kommunikationszeiten von dem linearen Modell $T = \alpha + \beta n$ ausgegangen. $2T$ $[ms]$ ist die Zeit, die für das Hin- und Hersenden einer Datenmenge zwischen 2 Prozessoren benötigt wird. $n$ steht für die Anzahl der Elemente (double precision), die übertragen werden. $\alpha$ gibt die Startupkosten (Latency) und $\beta$ die Kosten pro Element der Operation an.

Die in Tabelle 1 angegebenen Zahlen und die in den Abbildungen 2 und 3 dargestellten Werte beruhen auf jeweils 5 Zeitmessungen $T_{i,j}$, $i = 1, \cdots, 5$, für 10 verschiedene Datenmengen $n_j = 5000 + j5000$, $j = 0, \cdots, 9$.

Mit $T_{m_j}$ werde der Mittelwert der Zeitmessungen für eine Datenmenge bezeichnet. Die Werte $\alpha$ und $\beta$ in Tabelle 1 errechnen sich durch Least Squares Fitting. In der dritten Spalte der Tabelle ist der relative Fehler $E_{rel}$ der Messungen angegeben:

$$\sum_j (T_{m_j} - \alpha - \beta n_j)^2 \to min, \quad E_{rel} = \max_j \left\{ \frac{\max_i T_{i,j} - \min_i T_{i,j}}{T_{m_j}} \right\}.$$

Man erkennt aus Abbildung 2, daß die Kommunikation mit Hilfe der BLACS für Datenmengen, deren Umfang größer als 500 ist, etwa um 20%

| Call | $\alpha$ | $\beta$ | $E_{rel}$ |
|---|---|---|---|
| BLACS | 40 | 9.06 | 0.04 |
| $SendLink/RecvLink$ | 166 | 7.57 | 0.02 |
| $SendNode/RecvNode$ | 709 | 7.57 | 0.04 |

Tabelle 1: Least Squares Fitting

langsamer ist als die durch PARIX-Kommandos.

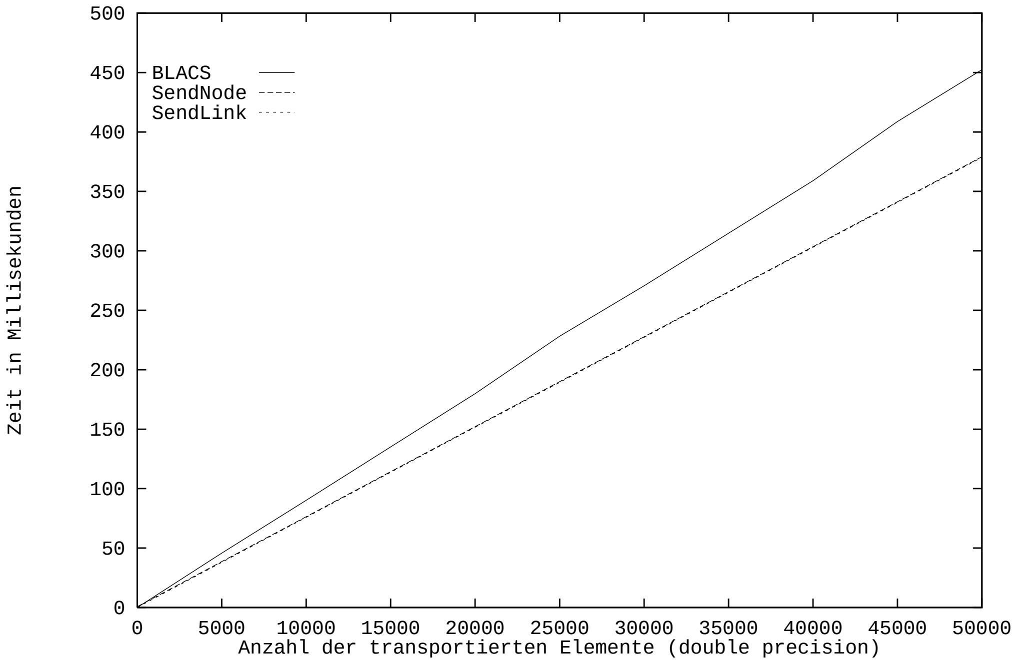


Abbildung 2: Knoten-Knoten-Kommunikation für große Datenmengen

Für Datenmengen, deren Umfang kleiner als 500 ist, ergibt sich die Reihenfolge: $SendLink/RecvLink$, BLACS, $SendNode/RecvNode$ (siehe Abbildung 3). Die in der Abbildung 3 bei der Kommunikation mit Hilfe der BLACS auftretenden Sprünge sind darauf zurückzuführen, daß die Daten portionsweise zu je 1024 Bytes transportiert werden. In dem Graph für die Kommunikationszeiten bei Verwendung von $SendNode/RecvNode$ tritt ein

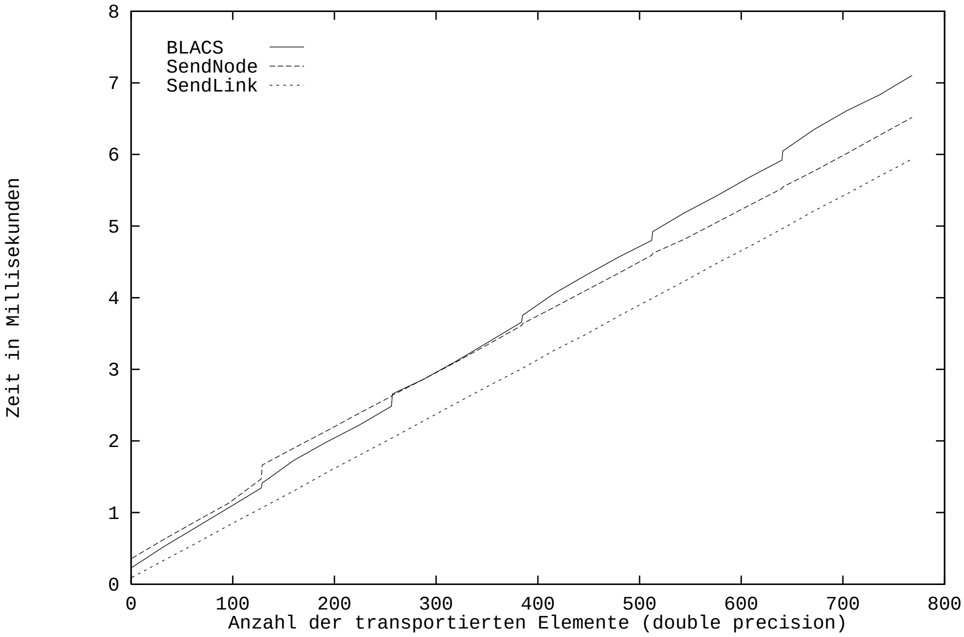


Abbildung 3: Knoten-Knoten-Kommunikation für kleine Datenmengen

Sprung bei 1024 Bytes auf, weil bei der Übertragung der Daten unterschiedliche Mechanismen angewendet werden je nachdem, ob der Datenumfang größer oder kleiner als 1024 Bytes ist.

# Literatur

[1] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D., *LAPACK Users' Guide*, Society for Industrial and and Applied Mathematics, Second Edition, Version 2.0, 1994.

[2] Choi, J., Dongarra, J. J., Ostrouchov, S., Petitet, A., Walker, D., Whaley, R. C., *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, LAPACK Working Note 100, University of Tennessee, Knoxville, CS-95-292, May 1995.

[3] Choi, J., Demmel, J., Dhillon, I., Dongarra, J. J., Ostrouchov, L. S., Petitet, A., Stanley, K., Walker, D., Whaley, R. C., *Installation Guide for ScaLAPACK*, LAPACK Working Note 93, University of Tennessee, Knoxville, CS-95-280, March 1995 (Version 1.0).

[4] Choi, J., Demmel, J., Dhillon, I., Dongarra, J. J., Ostrouchov, L. S., Petitet, A., Stanley, K., Walker, D., Whaley, R. C., *ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Compuers - Design Issues and Performance*, LAPACK Working Note 95, University of Tennessee, Knoxville, CS-95-283, March 1995.

[5] Dongarra, J. J., DuCroz, J., Duff, I and Hammarling, S., *A Extendet Set of FORTRAN Basic Linear Algebra Subroutines*, ACM Transactions on Mathematical Software, 14(1),1-17,March 1988.

[6] Dongarra, J. J., DuCroz, J., Hammarling, S. and Duff, I, *A Set of Level 3 Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16(1),1-17,1990.

[7] Dongarra, J. J., Whaley, R. C., *A User's Guide to the BLACS v1.0*, LAPACK Working Note 94, University of Tennessee, Knoxville, CS-95-281, June 7, 1995.

[8] Hebermehl, G., *Anpassungsfähige Datenstukturen und blockorientierte Algorithmen der linearen Algebra für Distributed Memory Systeme*, in Hektor, J. and Grebe, R. (Eds.), *Parallele Datenverarbeitung mit dem Transputer*, 5. Transputer-Anwender-Treffen TAT93 Aachen, 20.-22. September 1993, Springer-Verlag, Reihe Informatik aktuell, 34-53, 1994.

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

[10] Whaley, R. C., *Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures*, LAPACK Working Note 73,University of Tennessee, Knoxville, CS-94-234, June 10, 1994.