Technische Universität Chemnitz

Sonderforschungsbereich 393

Numerische Simulation auf massiv parallelen Rechnern

T.Ermer, L. Grabowsky

Objektorientierte Implementierung eines PPCG–Verfahrens

Preprint SFB393/98-10

Zusammenfassung

Ein üblicher Ansatz bei der Parallelisierung von FEM–Verfahren ist die Gebietszerlegung. Typisch hierbei ist, daß die beteiligten Prozessoren weitgehend lokal arbeiten können und nur an wenigen Punkten eine sogenannte "Koppelrandkommunikation" erforderlich ist.

Während sich ein diesbezüglicher Algorithmus recht einfach formal angeben läßt, bedingen die von prozeduralen Sprachen bereitgestellten Mittel eine Anpassung der Programmstruktur an konkrete Kommunikationsbibliotheken und Hardware–Systeme.

Abhilfe bringen hier objektorientierte Methoden. Anhand eines parallelen konjugierten Gradientenverfahrens wird die Verwendung von C++ als Implementationssprache demonstriert.

Preprint-Reihe des Chemnitzer SFB 393

SFB393/98-10 **April 1998**

Inhaltsverzeichnis

Author's addresses:

Thomas Ermer
TU Chemnitz
Fakultät für Informatik
D-09107 Chemnitz

`therminformatik.tu-chemnitz.de`

1 Algorithmische Grundlagen

Zunächst werden die theoretischen Grundlagen – beschränkt auf die Besonderheiten eines parallelen konjugierten Gradientenverfahrens sowie der Vorkonditionierungsproblematik für Systeme mit verteiltem Speicher – kurz skizziert. Für eine detailliertere Beschreibung, insbesondere zur Finite–Elemente–Substrukturtechnik, sei der Leser auf [3] verwiesen.

Ausgangspunkt sei ein System partieller Differentialgleichungen 2. Ordnung in einem d–dimensionalen beschränkten Gebiet Ω. Dieses Gebiet wird in p sich nicht überlappende Teilgebiete $\Omega_i (i = 1, ..., p)$ unterteilt. Für jedes dieser Teilgebiete wird desweiteren eine Unterteilung in finite Elemente durchgeführt.

1.1 Ein paralleles CG–Verfahren (PPCGM) für Systeme mit verteiltem Speicher

Analog zu den p Teilgebieten $\overline{\Omega}_i, (i = 1, 2, ..., p)$ seien die für die Berechnung benötigten Matrizen und Vektoren über die p Prozessoren verteilt. Zwei Arten von Vektoren werden nach ihrer Verteilung unterschieden, die *überlappenden* (Typ I) und die *addierenden* (Typ II) Vektoren:

$$
\begin{array}{rl}
\text{Typ I}: & \underline{u}, \underline{w}, \underline{s} \text{ werden in Prozessor } P_i (\equiv \overline{\Omega}_i) \text{ als} \\
& \underline{u}_i = A_i \underline{u}, \underline{w}_i = A_i \underline{w}, \underline{s}_i = A_i \underline{s} \text{ gespeichert} \\
\text{Typ II}: & \underline{r}, \underline{v}, \underline{f} \text{ werden in } P_i \text{ als } \underline{r}_i, \underline{v}_i, \underline{f}_i \text{ gespeichert, wobei} \\
& \underline{r} = \displaystyle\sum_{i=1}^{p} A_i^T \underline{r}_i \text{ gilt } (\underline{r}_i, \underline{v}_i \text{ analog})
\end{array}
$$

Die $(N_i \times N)$–Matrix A_i ist hierbei die Boolesche Superelementzusammenhangsmatrix des Teilgebietes Ω_i. Mit diesen Matrizen kann die Steifigkeitsmtarix K in eine andere Form umgeschrieben werden:

$$
K = \sum_{i=1}^{p} A_i^T K_i A_i \tag{1}
$$

, wobei K_i die zu Ω_i gehörende Superelementsteifigkeitsmatrix ist.

Mit dieser Datenverteilung und obiger Notation kann folgender paralleler CG–Algorithmus angegeben werden:

	Berechne parallel auf allen Prozessoren P_i:
0.	<u>Start</u>
	Wähle einen Anfangswert für $\underline{u}$, z.B. $\underline{u} = \emptyset$
	$\underline{r}_i = \underline{f} - K_i \underline{u}_i$
	$\underline{w} = C^{-1} \sum A_i^T \underline{r}_i$
	$\underline{s}_i = \underline{w}_i \quad (\underline{w}_i = A_i \underline{w})$
	$\sigma_i = \underline{w}_i^T \underline{r}_i$
	$\sigma = \sigma^0 = \sum \sigma_i$
	Iteration
1.	$\underline{\tilde{v}}_i = K_i \underline{\tilde{s}}_i$
	$\delta_i = \underline{\tilde{v}}_i^T \underline{\tilde{s}}_i$
	$\delta = \sum \delta_i$
	$\alpha = \tilde{\sigma} / \delta$
2.	$\underline{u}_i = \underline{\tilde{u}}_i + \alpha \underline{\tilde{s}}_i$
	$\underline{r}_i = \underline{\tilde{r}}_i - \alpha \underline{\tilde{v}}_i$
3.	$\underline{w} = C^{-1} \sum A_i^T \underline{r}_i$
4.	$\sigma_i = \underline{w}_i^T \underline{r}_i$
	$\sigma = \sum \sigma_i$
	$\beta = \sigma / \tilde{\sigma}$
5.	$\underline{s}_i = \underline{w}_i + \beta \underline{\tilde{s}}_i$
6.	$\sigma \leq \epsilon^2 \cdot \sigma^0$? @ > *nein* >> gehe zu 1.
	$\downarrow$ ja
	Stop

Im obigen Algorithmus markiert das Tilde–Symbol "˜" die vorangegangene Iteration. Eine Summation ($\sum$) beinhaltet Summenbildung und Kommunikation. Im Fall $C = I$ (also ohne Vorkonditionierung) ist ein Datenaustausch in der Größenordnung $O(h^{-(d-1)})$ für die Koppelranddaten notwendig, um Schritt 3 zu realisieren.
Durch den Vorkonditionierungsoperator C sollte sich also der Kommunikationsaufwand nicht wesentlich erhöhen. Desweiteren zeigt sich für zahlreiche Fälle, daß die notwendige Kommunikation auf eine Typumwandlung Typ II $\longrightarrow$ Typ I eines gewissen Vektors zurückzuführen ist. Ein Vorkonditionierungsoperator, der obige Bedingung erfüllt, ist im Abschnitt 1.2 dargestellt.

1.2 Vorkonditionierung im Distributed–Memory–Fall

Es sei $C = \hat{V}\hat{V}^T$ der Vorkonditionierungsoperator in obigem Algorithmus. Mit

$$\underline{r} = \sum_{i=1}^{p} A_i \underline{r}_i$$

läßt sich $\underline{w}$ wie folgt bestimmen:

$$\underline{y} = \hat{V}^T \underline{r} \qquad \text{und} \qquad \underline{w} = \hat{V}^T \underline{y}$$

Die lokale Berechnung von $\underline{y}_i = \hat{V}^T \underline{r}_i$ verändert nicht die Typ II – Eigenschaften von $\underline{y}$. Ebenso erhält die Lösung von $\underline{w}_i = \hat{V}^T \underline{y}_i$ die Typ I –Eigenschaften des Vektors $\underline{w}$.
Demzufolge benötigt ein Schritt des CG–Verfahrens mit hierarchischer Vorkonditionierung denselben Kommunikationsumfang wie ein Schritt ohne Vorkonditionierung.

2 Objektorientierte Realisierung der PPCGM

2.1 Struktur der Fortran–Implementierung

Zum Vergleich sei hier noch einmal der prinzipielle Ablauf der originalen Fortran–Implementierung angegeben:

Parallele CGM mit paralleler Vorkonditionierung:

1. *Vorbereitende Schritte zur Vorkonditionierung* (Routine PRE-VOR), Cholesky–Zerlegung, Initialisieren der Grobgittermatrix, Erzeugen der Matrizen für die Jacobi–Modifikation

2. Iteration

   (a) *Anwendung der Vorkonditionierung* (Routine PRLOES)

      i. *tree_up_dod* Aufsummieren der Crosspoints, das Ergebnis entsteht nur auf Prozessor 0

      ii. *Grobgitterlösung* auf Prozessor 0 (falls erwünscht)

      iii. *tree_down* Ergebnis an alle Prozessoren verteilen

      iv. *KettAkk* Behandlung der Randketten[a]

   (b) Bestimmung der neuen Iterierten
   Ist Abbruchkriterium erfüllt ⇒ fertig, sonst neue Iteration.

[a]Dieser Schritt kann auch vor der Verarbeitung der Crosspoints erfolgen.

Der entsprechende Quelltextabschnitt sieht wie folgt aus (Routine PPCGM):

```
      CALL PREVOR(NK,NC,N,A,LA,C,LCC,CC,Kette,Iglob,V,IER)

      BE  = 0D0
      ABBR = 0D0
      ITMAX=IT
      IT=0
      CALL AXMKLZ('0',N,A,LA,X,S)
      CALL VDminus(N,R,1,S,1,B,1)

C   ***       STARTRESIDUUM AUF R

    1 IT=IT+1

C ****** Start : Anwendung der Vorkonditionierung

      CALL PRLOES(NK,NC,N,A,LA,C,LC,CC,LCC,Kette,Iglob,W,R,V)

C ****** Ende der Vorkond. : W fertig

      RW = DSCAPR(N,W,R)

      CALL Cube_DoD(1,RW,RW,H,VDplus)

      IF ( IT .GT. 1 )  BE = RW / RWV
      IF ( RW .LE. ABBR )   GOTO 10
      RWV = RW
      IF ( RW .EQ. 0D0 ) GOTO  10
      IF ( IT .EQ. 1 ) ABBR = EPS*EPS * RW

      CALL VDaxpy(N,S,W,BE,S)
      CALL AXMKLZ('0',N,A,LA,S,W)
      SAS = DSCAPR(N,W,S)
      CALL Cube_DoD(1,SAS,SAS,H,VDplus)
      AL  = -RW / SAS
      CALL VDaxpy(N,X ,X ,AL,S)
      CALL VDaxpy(N,R ,R ,AL,W)
      IF ( IT .GE. ITMAX )   RETURN
      GOTO 1
10    ...
```

Die Anwendung der Vorkonditionierung und die (eventuelle) Lösung des Grobgittersystems erfolgt in der Routine PRLOES (original, Ausschnitt):

```
      SUBROUTINE PRLOES(NKu,NCu,N,A,LA,C,LC,CC,LCC,Kette,
     +                  Iglob,W,R,V)

C  ****** Initialisierungen ...
C  ...
C  ****** Crosspoint-Vektor fuellen:

      CALL VDcopy(NCrossG,V,1,0D0,0)
      DO  1  I=1,NCrossL
         Ito =Iglob(I)
1     V(Ito) = W(I)

C  ****** Crosspoints zur 0:

      CALL TreeUp_DoD(NCrossG,V,V,V(NCrossG+1),VDplus)

C  ****** Grobgitterloesung auf 0:

      IF (Ivar .NE.0 .AND. Ich.EQ.0 ) THEN
          CALL RueVBZ(NCrossG,NCrossG,1,CC,LCC,V,V)
          CALL VorVBZ(NCrossG,NCrossG,1,CC,LCC,V,V)
      ENDIF

C  ****** Verarbeitung der Randketten :

      CALL KettAKK(1,W,Kette,V(NCrossG+1))

C  ****** Crosspoints an alle:

      ll=2*NCrossG
      CALL Tree_Down(ll,V)

C  ****** Crosspoints-Vektor zurueckschreiben:

      DO 2 I=1,NCrossL
         Ito =Iglob(I)
2     W(I)=V(Ito)
C  ...

      RETURN
      END
```

Unter Verwendung von MPI–Kommunikatoren ändert sich die Verarbeitung der Randketten (Routine KettAkk wird ersetzt):

```
      SUBROUTINE PRLOES(NKu,NCu,N,A,LA,C,LC,CC,LCC,Kette,
    +                 Iglob,W,R,V,COMM,TYPES,NCOMM)

C  ****** Initialisierungen ...
C  ...
C  ****** Verarbeitung der Randketten:

      do 125 i=1,ncomm
                call mpi_pack_size(1, types(i), comm(i),
    +                  isize, ierror)
                ipos=0
                call mpi_pack(w, 1, types(i), v, isize,
    +                        ipos, comm(i), ierror)
                isize=ipos
                ih = ipos/8
                call mpi_allreduce(v, v(ih+1),ih,
    +                  mpi_double_precision, mpi_sum,
    +                  comm(i), ierror)
                ipos=0
                call mpi_unpack(v(ih+1), isize, ipos, w, 1,
    +                  types(i), comm(i), ierror)
125   continue

C  ****** Crosspoint-Kommunikation und Grobgitterloesung wie oben
C  ...

      RETURN
      END
```

Trotz gestiegener Effizienz bei der Verarbeitung der Randketten mittels MPI–Mechanismen greift diese Realisierung doch auf spezifische Bibliotheksfunktionalitäten zurück und verhindert die Portierung auf Hardware, für die MPI nicht verfügbar ist (siehe auch [2, 1]).

2.2 Struktur der objektorientierten Realisierung

2.2.1 Utility–Klassen

Basis der folgenden Klassenhierarchie ist die GNU C++–Bibliothek **AVec**. Ausgehend von den von der Bibliothek zur Verfügung gestellten Vektoren werden die für das GC–Verfahren benötigten Vektoren und Matrizen abgeleitet. Ziel ist es, die formale Algorithmenbeschreibung annähernd in den Programmtext umzusetzen und damit ein Unabhängigkeit von der konkreten Kommunikationsbibliothek sowie der zugrunde liegenden Hardware zu erzielen.

- *Vektoren:* Die Basisklasse *Vec* der GNU C++–Library stellt einen einfachen Vektor mit den für alle abgeleiteten Vektoren benötigten Basisoperationen (Zuweisung, Vergleich, Sortierung, usw.) bereit. Bei dem Vektor handelt es sich um einen sogenannten generischen Datentyp, d.h. der konkrete Typ wird erst durch die Initialisierung bestimmt, ebenso arbeiten alle Operationen mit diesen generischen Datentypen.

  AVec (arithmetischer Vektor) erbt die Eigenschaften von *Vec* und definiert zusätzlich folgende Operationen (Auswahl):

  – Vektor $\times$ Skalar $\rightarrow$ Vektor: $+, -, *, /$

  – Vektor $\times$ Vektor $\rightarrow$ Vektor: $+, -,$ Produkt, Quotient

  – Vektor $\rightarrow$ Skalar: $*$ (Skalarprodukt), Summe der Komponenten

  AVec_TYPEI bzw. *AVec_TYPEII* abstrahieren die in Abschnitt 1.1 eingeführten Vektortypen. Mittels objektorientierter Methoden lassen sich auch für diese Vektorklassen die Zuweisung und Skalarproduktbildung definieren:

  – Die Zuweisung

  $$a = b, \quad a \text{ ist Typ I}, b \text{ ist Typ II}$$

  beinhaltet Typumwandlung. Die Summation der Ketten und Crospoints ist Teil dieser, d.h. die hierfür notwendige Kommunikation ist innerhalb des Zuweisungsoperators implementiert.

  – Die Skalarproduktbildung

  $$a * b, \quad a \text{ ist Typ I}, b \text{ ist Typ II}$$

  beinhaltet die Summation der lokalen Teilskalarprodukte.

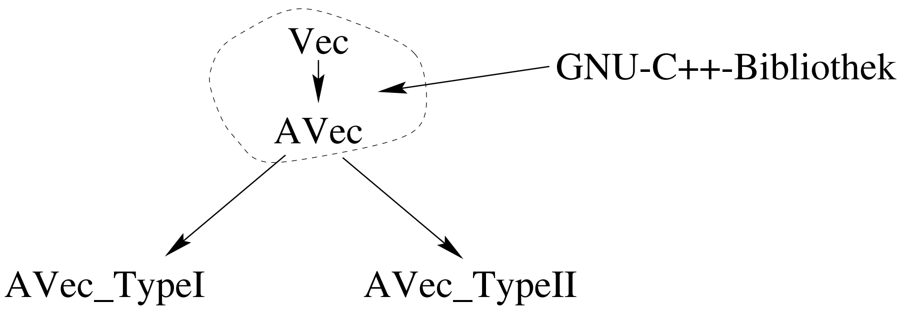

Abbildung 1: Klassenhierarchie der Vektoren

Abbildung 1 veranschaulicht die Klassenhierarchie.

- *Matrizen:* Die Basisklasse *Matrix* stellt eine einfache quadratische Matrix zur Verfügung, woraus die Klassen der KLZ–Matrizen (Kompaktliste zeilenweise) bzw. VBZ–Matrizen (variable Bandweite zeilenweise) abgeleitet werden. Folgende Methoden werden bereitgestellt:

  – Operationen zur Multiplikation mit Vektoren der AVec–Klasse

  – Operationen für Cholesky–Zerlegung, Dreieckslöser, usw...

  Abbildung 2 zeigt die Zusammenhänge.

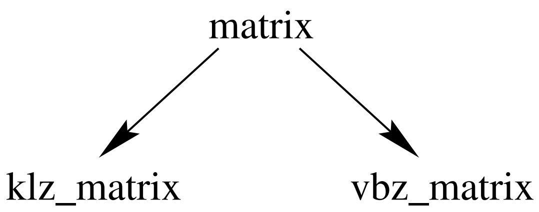

Abbildung 2: Klassenhierarchie der Matrizen

- *Löser:* Drei Methoden werden innerhalb der Basisklasse für Löser definiert:

  – Init_solver: Initialisierung von MaxIT (maximale Iterationsanzahl) und EPS (Maschinengenauigkeit)

  – convergence: Berechnung des Abbruchkriteriums

  – get_eps: Liefert die Maschinengenauigkeit (EPS)

  Konkrete Löser, wie PPCGM, werden von obiger Klasse abgeleitet.

- *Vorkonditionierer:* Zwei Methoden sind (vorerst) vorgesehen:

  – prevor: Vorbereitung der Vorkonditionierung

  – prloes: Vorkonditionierung

Insbesondere die Definition der Klassen für die Typ I – bzw. Typ II–Vektoren ist wesentlich für die Formulierung des parallelen Algorithmus.

2.2.2 Der CG–Algorithmus

Die im CG–Verfahren erforderliche Typumwandlung der Vektoren (Typ II ⇒ Typ I) kann mit den in Abschnitt 2.2.1 eingeführten Vektorklassen sehr einfach formuliert werden (Vergleiche 2.1):

$$\left.\begin{array}{r}\text{Tree\_up\_dod}\\ \text{KettAkk}\\ \text{Tree\_down}\end{array}\right\} \hat{=} \text{Type II} \Rightarrow \text{Typ I}$$

Problematisch bei obiger Vorgehensweise ist die Grobgitterlösung. Bei Berechnung dieser in bisheriger Vorgehensweise, d.h. nur Prozessor 0 besitzt die zugehörige Matrix und löst das System, würde die zusätzliche Ausführung der Typumwandlung einen unnötigen Kommunikationsschritt bedeuten. Der Crosspoint–Vektor müßte im Anschluß erneut an alle Prozessoren verteilt werden.

Ein Ausweg ist die Generierung des Grobgittersystems auf allen Prozessoren und damit eine lokale Berechnung des Systems. Der rechnerische Mehraufwand bedingt keine zusätzliche Zeit, da während der Lösung des Grobgittersystems auf Prozessor 0 die anderen ohnehin untätig auf dessen Ergebnisse warten. Mit der Definition von Skalarproduktoperationen Typ I $*$ Typ II kann schließlich der Algorithmus insbesondere unabhängig vom Kommunikationsmechanismus formuliert werden. Damit ist vor allem die Formulierung unabhängig davon, ob die ursprüngliche Variante (KettAkk) oder die MPI–Variante genutzt wird, welche in der Fortran–Implementierung zu umfangreichen Quelltextänderungen, auch struktureller Art, führte.

Die veränderte Realisierung sieht wie folgt aus:

Cube_dod
KettAkk
lokale Grobgitterlösung

bzw. als C++–Programmfragment:

```
v = w;                          // Zuweisung (Typumwandlung)
if (/* Grobgitterloesung /*)
{
   vc = cg.get_crosspoints ( v );  // Extrahieren der Crosspoints
   cg.tsolve ( vc );               // Grobgitterloesung
};
```

Die Iteration im Löser ist dann von folgender Struktur:

```
// ...

do
{
   iteration++;
   vk->prloes(cc, w, r);

   rw = *w * *r;

   if (it > 1) be = rw / rwv;
   if (rw >= abbr)
   {
      rwv = rw;
      if (it == 1) abbr = get_eps() * get_eps() * rw;
      *s = *s * be + *w;
```

```
        *w = *a * *s;
        sas = *w * *s;

        al = -rw / sas;
        *x += *s * al;
        *r += *w * al;
    }
} while (!convergence(rw, abbr, iteration));

// ...
```

3 Zusammenfassung

Aufbauend auf verfügbare C++–Bibliotheken (als konkrete Implementierungsbasis diente die GNU C++–Bibliothek) lassen sich die für numerische Methoden benötigten Datenstrukturen und Algorithmen auf hinreichendem Abstraktionsniveau formulieren. System–und kommunikationsspezifische Details werden in den Basisklassen implementiert, so daß die Struktur des Programmes der algorithmischen Beschreibung weitestgehend entspricht.
Noch gerät C++ gegenüber Fortran in Sachen Effizienz ins Hintertreffen – z.B. ist durch häufige Verwendung temporärer Objekte dieser Nachteil vorprogrammiert – , jedoch sind mit dem erst kürzlich verfügbar gewordenem Standard (existiert als Vorschlag, mit baldiger Ratifizierung ist zu rechnen) nun auch leistungsfähigere Compiler zu erwarten, die in der Lage sein sollten, optimierten Programmcode zu generieren. Derzeit existieren mehrere hochoptimierte C++–Numerikbibliotheken, die bei der vorliegenden Implementierung zwar noch nicht berücksichtigt werden konnten, aber eine weitere Beschleunigung des Lösungsvorganges versprechen.

Literatur

[1] L. Grabowsky. MPI-basierte Koppelrandkommunikation und Einfluß der Partitionierung im 3D-Fall. Preprint SFB393/97-17, TU Chemnitz, August 1997.

[2] L. Grabowsky, T. Ermer, and J. Werner. Nutzung von MPI für parallele FEM-Systeme. Preprint SFB393/97-08, TU Chemnitz-Zwickau, März 1997.

[3] G. Haase, U. Langer, and A. Meyer. Parallelisierung und Vorkonditionierung des CG-Verfahrens durch Gebietszerlegung. In *Proceedings of the GAMM-Seminar "Numerische Algorithmen auf Transputersystemen" held at Heidelberg, 1991.* Teubner Verlag, Stuttgart, 1991.

Other titles in the SFB393 series:

96-01 V. Mehrmann, H. Xu. Chosing poles so that the single-input pole placement problem is well-conditioned. Januar 1996.

96-02 T. Penzl. Numerical solution of generalized Lyapunov equations. January 1996.

96-03 M. Scherzer, A. Meyer. Zur Berechnung von Spannungs- und Deformationsfeldern an Interface-Ecken im nichtlinearen Deformationsbereich auf Parallelrechnern. March 1996.

96-04 Th. Frank, E. Wassen. Parallel solution algorithms for Lagrangian simulation of disperse multiphase flows. Proc. of 2nd Int. Symposium on Numerical Methods for Multiphase Flows, ASME Fluids Engineering Division Summer Meeting, July 7-11, 1996, San Diego, CA, USA. June 1996.

96-05 P. Benner, V. Mehrmann, H. Xu. A numerically stable, structure preserving method for computing the eigenvalues of real Hamiltonian or symplectic pencils. April 1996.

96-06 P. Benner, R. Byers, E. Barth. HAMEV and SQRED: Fortran 77 Subroutines for Computing the Eigenvalues of Hamiltonian Matrices Using Van Loans's Square Reduced Method. May 1996.

96-07 W. Rehm (Ed.). Portierbare numerische Simulation auf parallelen Architekturen. April 1996.

96-08 J. Weickert. Navier-Stokes equations as a differential-algebraic system. August 1996.

96-09 R. Byers, C. He, V. Mehrmann. Where is the nearest non-regular pencil? August 1996.

96-10 Th. Apel. A note on anisotropic interpolation error estimates for isoparametric quadrilateral finite elements. November 1996.

96-11 Th. Apel, G. Lube. Anisotropic mesh refinement for singularly perturbed reaction diffusion problems. November 1996.

96-12 B. Heise, M. Jung. Scalability, efficiency, and robustness of parallel multilevel solvers for nonlinear equations. September 1996.

96-13 F. Milde, R. A. Römer, M. Schreiber. Multifractal analysis of the metal-insulator transition in anisotropic systems. October 1996.

96-14 R. Schneider, P. L. Levin, M. Spasojević. Multiscale compression of BEM equations for electrostatic systems. October 1996.

96-15 M. Spasojević, R. Schneider, P. L. Levin. On the creation of sparse Boundary Element matrices for two dimensional electrostatics problems using the orthogonal Haar wavelet. October 1996.

96-16 S. Dahlke, W. Dahmen, R. Hochmuth, R. Schneider. Stable multiscale bases and local error estimation for elliptic problems. October 1996.

96-17 B. H. Kleemann, A. Rathsfeld, R. Schneider. Multiscale methods for Boundary Integral Equations and their application to boundary value problems in scattering theory and geodesy. October 1996.

96-18 U. Reichel. Partitionierung von Finite-Elemente-Netzen. November 1996.

96-19 W. Dahmen, R. Schneider. Composite wavelet bases for operator equations. November 1996.

96-20 R. A. Römer, M. Schreiber. No enhancement of the localization length for two interacting particles in a random potential. December 1996. to appear in: Phys. Rev. Lett., March 1997

96-21 G. Windisch. Two-point boundary value problems with piecewise constant coefficients: weak solution and exact discretization. December 1996.

96-22 M. Jung, S. V. Nepomnyaschikh. Variable preconditioning procedures for elliptic problems. December 1996.

97-01 P. Benner, V. Mehrmann, H. Xu. A new method for computing the stable invariant subspace of a real Hamiltonian matrix or Breaking Van Loan's curse? January 1997.

97-02 B. Benhammouda. Rank-revealing 'top-down' ULV factorizations. January 1997.

97-03 U. Schrader. Convergence of Asynchronous Jacobi-Newton-Iterations. January 1997.

97-04 U.-J. Görke, R. Kreißig. Einflußfaktoren bei der Identifikation von Materialparametern elastisch-plastischer Deformationsgesetze aus inhomogenen Verschiebungsfeldern. March 1997.

97-05 U. Groh. FEM auf irregulären hierarchischen Dreiecksnetzen. March 1997.

97-06 Th. Apel. Interpolation of non-smooth functions on anisotropic finite element meshes. March 1997

97-07 Th. Apel, S. Nicaise. The finite element method with anisotropic mesh grading for elliptic problems in domains with corners and edges.

97-08 L. Grabowsky, Th. Ermer, J. Werner. Nutzung von MPI für parallele FEM-Systeme. March 1997.

97-09 T. Wappler, Th. Vojta, M. Schreiber. Monte-Carlo simulations of the dynamical behavior of the Coulomb glass. March 1997.

97-10 M. Pester. Behandlung gekrümmter Oberflächen in einem 3D-FEM-Programm für Parallelrechner. April 1997.

97-11 G. Globisch, S. V. Nepomnyaschikh. The hierarchical preconditioning having unstructured grids. April 1997.

97-12 R. V. Pai, A. Punnoose, R. A. Römer. The Mott-Anderson transition in the disordered one-dimensional Hubbard model. April 1997.

97-13 M. Thess. Parallel Multilevel Preconditioners for Problems of Thin Smooth Shells. May 1997.

97-14 A. Eilmes, R. A. Römer, M. Schreiber. The two-dimensional Anderson model of localization with random hopping. June 1997.

97-15 M. Jung, J. F. Maitre. Some remarks on the constant in the strengthened C.B.S. inequality: Application to h- and p-hierarchical finite element discretizations of elasticity problems. July 1997.

97-16 G. Kunert. Error estimation for anisotropic tetrahedral and triangular finite element meshes. August 1997.

97-17 L. Grabowsky. MPI-basierte Koppelrandkommunikation und Einfluß der Partitionierung im 3D-Fall. August 1997.

97-18 R. A. Römer, M. Schreiber. Weak delocalization dueto long-range interaction for two electrons in a random potential chain. August 1997.

97-19 A. Eilmes, R. A. Römer, M. Schreiber. Critical behavior in the two-dimensional Anderson model of localization with random hopping. August 1997.

97-20 M. Meisel, A. Meyer. Hierarchically preconditioned parallel CG-solvers with and without coarse-matrix-solvers inside FEAP. September 1997.

97-21 J. X. Zhong, U. Grimm, R. A. Römer, M. Schreiber. Level-Spacing Distributions of Planar Quasiperiodic Tight-Binding Models. October 1997.

97-22 W. Rehm (Ed.). Ausgewählte Beiträge zum 1. Workshop Cluster-Computing. TU Chemnitz, 6./7. November 1997.

97-23 P. Benner, Enrique S. Quintana-Ortí. Solving stable generalized Lyapunov equations with the matrix sign function. October 1997

97-24 T. Penzl. A Multi-Grid Method for Generalized Lyapunow Equations. October 1997

97-25 G. Globisch. The hierarchical preconditioning having unstructured threedimensional grids. December 1997

97-26 G. Ammar, C. Mehl, V. Mehrmann. Schur-like forms for matrix Lie groups, Lie algebras and Jordan algebras. November 1997

97-27 U. Elsner. Graph partitioning - a survey. December 1997.

97-28 W. Dahmen, R. Schneider. Composite Wavelet Bases for Operator Equations. December 1997.

97-29 P. L. Levin, M. Spasojević, R. Schneider. Creation of Sparse Boundary Element Matrices for 2-D and Axi-symmetric Electrostatics Problems Using the Bi-orthogonal Haar Wavelet. December 1997.

97-30 W. Dahmen, R. Schneider. Wavelets on Manifolds I: Construction and Domain Decomposition. December 1997.

97-31 U. Elsner, V. Mehrmann, F. Milde, R. A. Römer, M. Schreiber. The Anderson Model of Localization: A Challenge for Modern Eigenvalue Methods. December 1997.

98-01 B. Heinrich, S. Nicaise, B. Weber. Elliptic interface problems in axisymmetric domains. Part II: The Fourier-finite-element approximation of non-tensorial singularities. January 1998.

98-02 T. Vojta, R. A. Römer, M. Schreiber. Two interfacing particles in a random potential: The random model revisited. February 1998.

98-03 B. Mehlig, K. Müller. Non-universal properties of a complex quantum spectrum. February 1998.

98-04 B. Mehlig, K. Müller, B. Eckhardt. Phase-space localization and matrix element distributions in systems with mixed classical phase space. February 1998.

98-05 M. Bollhöfer, V. Mehrmann. Nested divide and conquer concepts for the solution of large sparse linear systems. to app.: April 1998.

98-06 T. Penzl. A cyclic low rank Smith method for large, sparse Lyapunov equations with applications in model reduction and optimal control. March 1998.

98-07 V. Mehrmann, H. Xu. Canonical forms for Hamiltonian and symplectic matrices and pencils. March 1998.

98-08 C. Mehl. Condensed forms for skew-Hamiltonian/Hamiltonian pencils. March 1998.

The complete list of current and former preprints is available via
`http://www.tu-chemnitz.de/sfb393/preprints.html`.