Technische Universität Chemnitz-Zwickau

DFG-Forschergruppe “SPC” · Fakultät für Mathematik

Matthias Pester

ON_LINE VISUALIZATION IN PARALLEL COMPUTATIONS

Fakultät für Mathematik
TU Chemnitz-Zwickau
D-09107 Chemnitz, FRG
(0371)-531-2656
(0371)-531-2657 (fax)
m.pester@mathematik.tu-chemnitz.de
01.11.94

Preprint-Reihe der Chemnitzer DFG-Forschergruppe
“Scientific Parallel Computing”

SPC 94_23 **November 1994**

Workshop on Visualization
D. Kröner and R. Rautmann (Eds)
1995 VSP/TEV

ON-LINE VISUALIZATION IN PARALLEL COMPUTATIONS

M. Pester

Faculty of Mathematics, Technical University of Chemnitz-Zwickau,
D-09107 Chemnitz, Germany

ABSTRACT

The investigation of new parallel algorithms for MIMD computers requires some postprocessing facilities for quickly evaluating the behavior of those algorithms. We present two kinds of visualization tool implementations for 2D and 3D finite element applications to be used on a parallel computer and a host workstation.

1. INTRODUCTION

In December 1992, at the Technical University of Chemnitz-Zwickau there was established a research group in the field of Scientific Parallel Computing (SPC) named *"Algorithmische Grundlagen der Simulation von ausgewählten Problemen der Kontinuumsmechanik auf massiv parallelen Rechnern"*[1]. The main purpose of this group is to investigate parallel numerical algorithms for the solution of large problems arising from differential equations in solid and fluid mechanics on massively parallel message passing MIMD computers with processor numbers of 100 and more.

For this purpose the major demand of the researchers is to get a *quick and dirty* on-line visualization at any intermediate stage of the parallel algorithms, rather than producing a final high-quality postprocessing presentation of the results.

After some initial remarks on problems of pre- and postprocessing on parallel computers we will present two methods of visualizing data computed on such machines. The first method uses the X11 standard library calls on the parallel machine, the second method is based on a visualization tool for high performance graphics workstations using a socket-based data connection.

[1] supported by the German Research Foundation (DFG)

2. PARALLEL PRE- AND POSTPROCESSING

First, we shall remark that there is a different point of view between sequential and parallel processing with respect to the data interfaces from preprocessing to computation on the one hand, and from computation to postprocessing on the other hand.

One reason for using parallel computers is to get better access to the large amount of data, since it can be computed and stored locally on the processors. As Figure 1 illustrates, the classical tasks of pre- and postprocessing have to be split into two parts each, one of them running on the parallel machine while the other one can be done more practically on a workstation.

On the part of the parallel computer we have to define three data interfaces for preprocessing steps:

Interface I is the external interface to be submitted to the parallel computer including the geometric boundary representation of the domain Ω with an initial coarse grid Ω_0 defined by

- the list of nodes (name, coordinates)
- the list of edges (name, node pointers)
- the list of faces (name, edge pointers), in 2D these are the subdomains.
- the list of subdomains (name, face pointers)
- the list of boundary conditions defined for edges in 2D or faces in 3D (name of the edge/face, descriptors and values)

Interface II is the local interface on each processor after any steps of parallel mesh refinement. Its data structure is similar to Interface I. Boundary conditions can be easily passed down to the new edges or faces.

Interface III is the FEM data interface for generating the system matrices. Boundary conditions have to be evaluated with respect to the nodes.

Thus, the part of preprocessor running on the workstation does not provide the finite element mesh for generating and solving the system, but it supplies the geometric information including boundary conditions and informations about domain decomposition, e. g. preferences for the mapping of the subdomains to the processors. Therefore, the parallel computer starts with a coarse grid having one subdomain (or a few of them) per processor (see Fig. 2). The initial step of the parallel processing is to refine the mesh locally keeping the boundary conditions under consideration for generating the local matrices of the equation system to be solved [2].

The same problem of reducing the amount of data which is required to be transmitted arises in the postprocessing phase.

As an example we consider a system of linear equations with more than

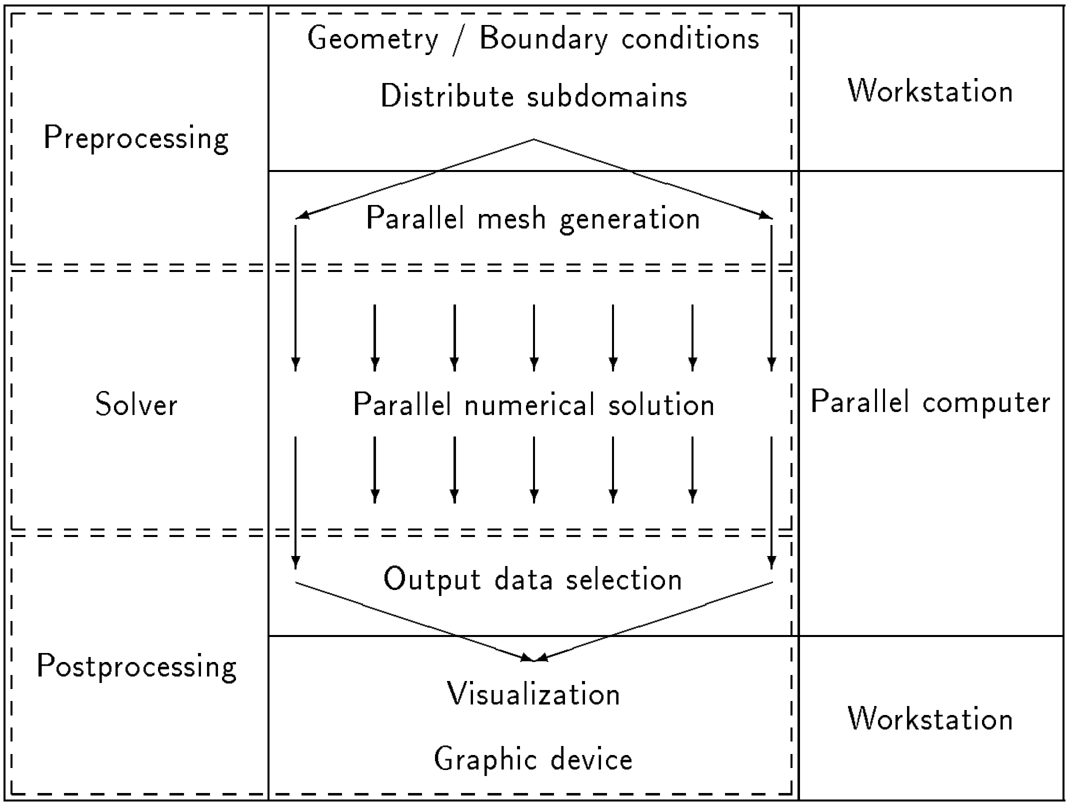

Fig. 1. Pre- and postprocessing interfaces on a parallel computer

4 millions of unknowns (one per node). The solution of this system takes about 34 seconds on a 64-processor system (GC-PowerPlus). Assuming a graphical window of 400×400 pixels each pixel on the screen would have to show 25 nodes (or 8 nodes if we had 3 degrees of freedom per node), i. e. most of the data sent to the display would be useless.

In order to get a sufficiently quick on-line visualization it is necessary to reduce the amount of data before sending it to the displaying workstation. This is a new interface between the computed data being situated in the local memory of the processors and the output device which is, in general, controlled sequentially. A decision is to be made if the large amount of data should be sent to a postprocessor on the host workstation or if the postprocessing should be done on the parallel processors themselves, at least in part.

The internal (hierarchical) data structures of the parallel algorithms we have in mind (see [2]) are well suited for this purpose. Based on a stepwise refinement of the initial coarse grid Ω_0 we get a hierarchical list of meshes Ω_i of level i $(i = 1, 2, \ldots)$ where $\Omega_i \supset \Omega_{i-1}$. With each level of mesh refinement the amount of data increases by a factor of ≈ 4 for 2D or ≈ 8 for 3D. Thus,

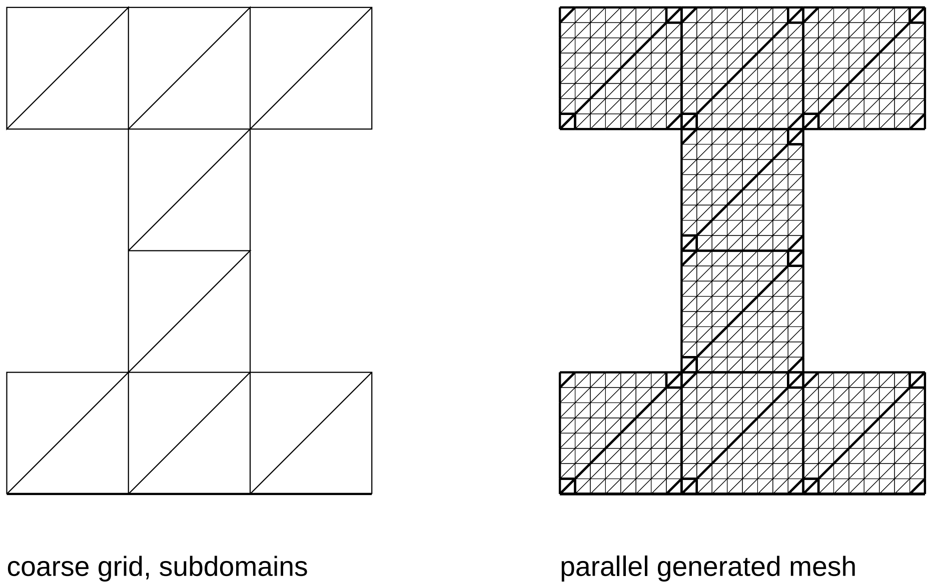

Fig. 2. Coarse grid and parallel mesh refinement (3 levels)

the effort at data handling for the postprocessing can essentially be reduced by selecting a lower level j for visualization than for computation. The higher level brings the computational accuracy. Selecting the lower level for visualization means to extract the corresponding subset of the computed data. This is easy if the refinement algorithm places all new elements (nodes, edges, faces) behind the previous data in each list. However, it is necessary to keep this geometric information of some lower levels in memory instead of throwing them away after the mesh refinement. It is clear that the additional memory requirement for all lower level information together is less than that for the last level.

3. POSTPROCESSING ON THE PARALLEL COMPUTER

As a first example of realizing any on-line visualization for parallel algorithms we consider the case of generating pixel data on the parallel computer, i. e. just on the same processor where the data of interest is produced. The assumptions we make are the following:

- The program runs on a parallel MIMD computer with message passing (distributed memory).

- The communication network within the parallel computer is based on a (virtual) hypercube topology. The processors are numbered from 0 to $2^n - 1$.
- At least one processor (say processor 0) is linked to the host workstation (direct link or connected to a remote host via ethernet).
- The X11 library function calls [3] are available on the parallel computer (at least on processor 0) which may contact the X server on the user's workstation (e. g. via ethernet or FDDI).
- The graphical display appears as a separate window different from the text window where the program was started from. Any interaction with the user that is necessary for the visualization can be done using the mouse in the graphical window or the keyboard in the text window.

The chose of a hypercube topology for message passing was caused by several advantages for the implementation of the inter-processor communications which occur in our applications. Any global communication needs $n = \log P$ time steps only for $P = 2^n$ processors. However, also a sequential transmitting of data from all processors to processor 0 is possible by selecting a subset of hypercube links forming a processor ring [6, 4].

Within the initialization step of the parallel program the processor 0 opens a connection to the X server of the destined workstation where the display should appear. This is realized by X11 library functions such as `XOpenDisplay` [3].

Then the visualization of intermediate or final results can be done by the following algorithm which is executed on each processor of the parallel machine.

Algorithm 1.

1. *What is to do?*
   Verify which data is to be displayed in which manner. For that purpose processor 0 interacts with the user and then broadcasts the user's "message" to the other processors. If the message was *"go on"* then return.
2. *Distributed Postprocessing*
   Compute locally the required data (e. g. isolines, colored areas) for the subdomain(s) of the current processor. Generate a set of polygons to be drawn or filled with specified colors of a given palette. Here, we use device independent coordinates, e. g. for a virtual display of 32000×32000 Pixels.
3. *Verify device parameters*
   Just before starting the graphical output of its local data, processor 0 requests the necessary information about the current size of the output window in order to convert pixel data from the device independent format into correct pixel values.
4. *Internal communication*
   The data of the other processors is forwarded to processor 0 using the em-

bedded ring topology of our hypercube.

5. *Display*
   Each packet of data is displayed by processor 0 just after having received it (sparing memory). At this moment the coordinates are transformed into proper pixel values of the current display.
6. *Printing and Plotting?*
   Optionally, the output can be written at the same time as a postscript file.

Some examples for 2D domains are shown in Figure 3.

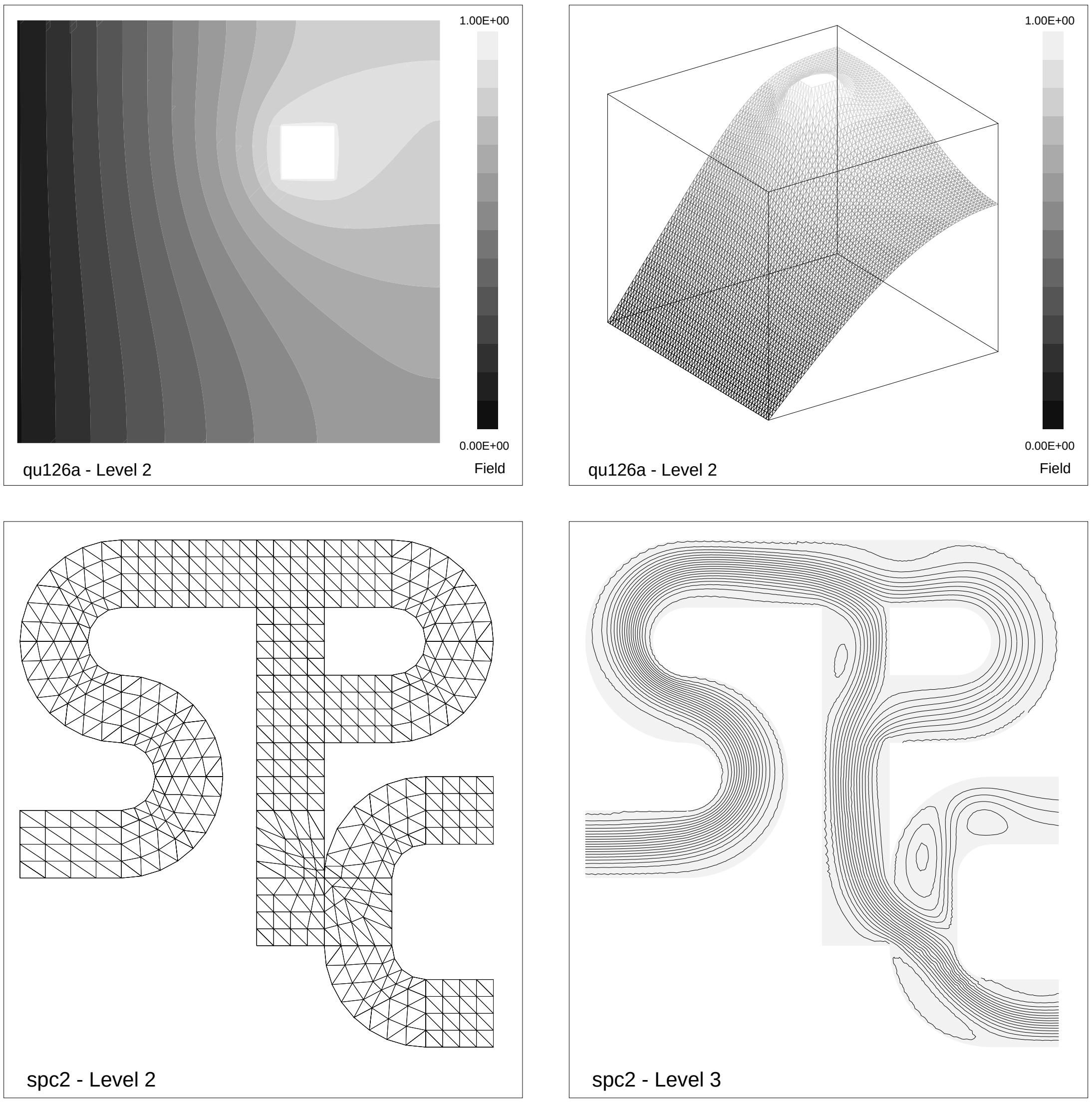

Fig. 3. Examples of parallel visualization for 2D domains

In order to visualize the solution rather quickly than in high quality there is no effort spent on special effects such as shading and lighting. This problem itself is also a well suited subject for parallelization and is already being discussed at other places.

4. POSTPROCESSING ON THE WORKSTATION

Generally, there exist several more comfortable programs for postprocessing running on a workstation. Our second way of visualizing data of the parallel program will make use of such a postprocessing tool for Finite Element data in the following way:

ALGORITHM 2.

1. *Two programs*
   Start both the parallel program on the parallel computer and the postprocessor on the user's workstation.
2. *External communication*
   Open a connection via sockets between both programs: processor 0 of the hypercube and the graphical workstation.
3. *Internal communication*
   All the processors have to route their (generally selected) data to processor 0 which uses a defined interaction protocol with the postprocessing program to send all the data for displaying it.
4. *Display*
   Since the two programs are running separately on different machines the parallel program may be continued after transmitting its data for displaying while the user can interact with the postprocessor either finding a special perspective or creating output files or requesting new data to be received from the parallel computer (e. g. one of the next solutions in fluid dynamics).

One program that fits our purpose is the GRAphical Programming Environment (GRAPE, [8]). Based on its library, we can use a lot of predefined postprocessing functions for 2D and 3D FEM completed by the necessary additional functions for the socket communication with the parallel program. Some examples are displayed in Figure 4.

Unfortunately, most of such programs for postprocessing are (professional) integrated packages whose user interface is unsuited for our purpose to interact with a parallel computer if the internal data structures are not manifested to the user.

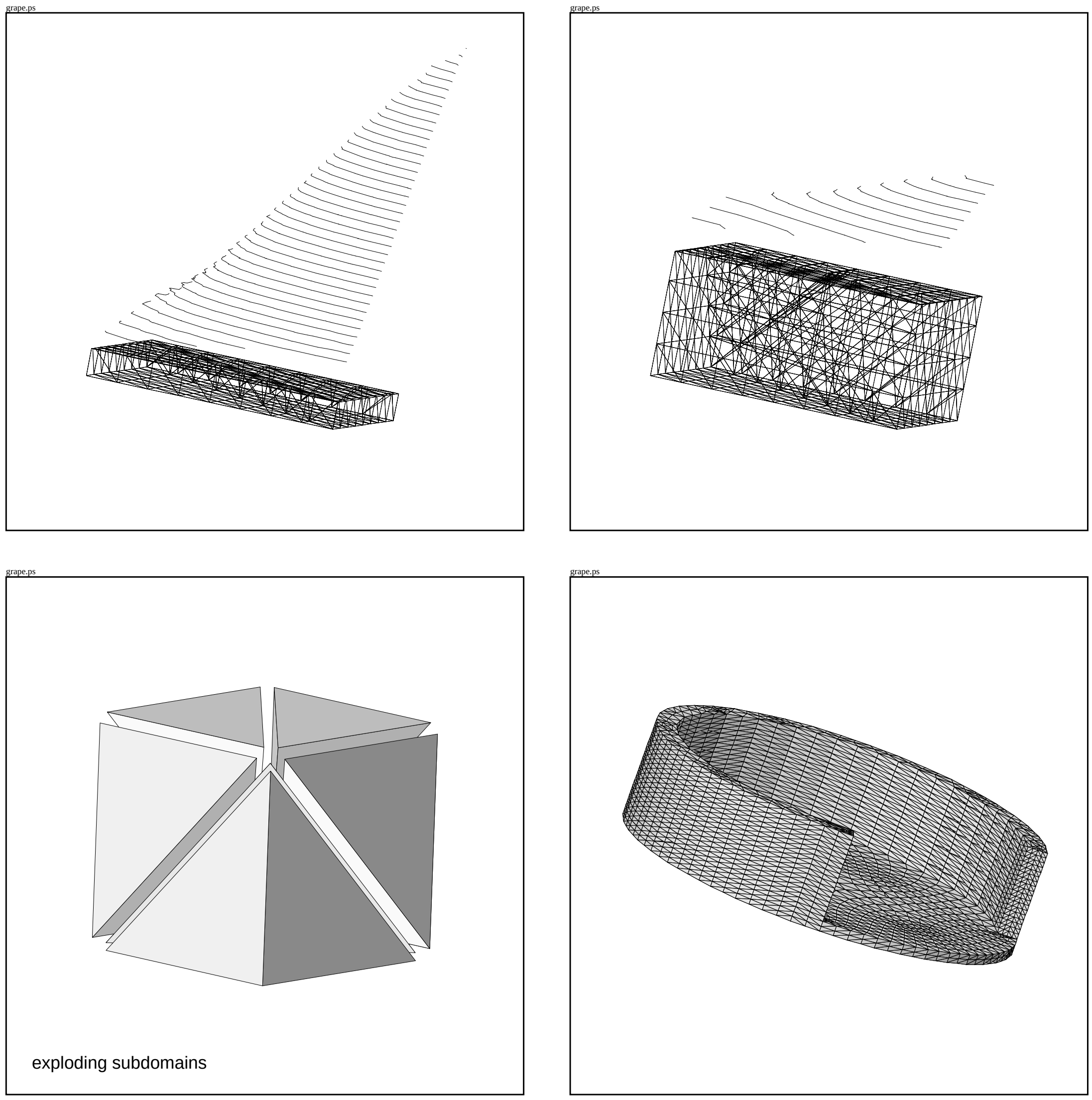

Fig. 4. Visualization of parallel computed data using GRAPE; isolines over a cross-section and patch mode display of solids

5. CONCLUSIONS

Both visualization methods presented above are implemented for a parallel Finite Element program running on several parallel machines. The first method requires the availability of X11 calls on the parallel computer itself which may be a disadvantage in some cases. Another disadvantage may be the idleness of the parallel computer while the user is enjoying the displayed data. The advantage is its "quick and dirty"-concept with an easy to use control. It was tested successfully on transputer systems under PARIX as well as on workstations under UNIX and PVM.

The second method only required some effort to add own functions and data structures to an existing program having the big advantage of using a wide spectrum of existing visualization tools. The "external" part of the postprocessor is running on a high performance graphical workstation corresponding to the high performance parallel computer.

The general problem, however, is to deal with the large amount of data and to find the right subset for displaying. Otherwise the visualization of the result will take much more time than its computation.

REFERENCES

[1] Haase, G., Langer, U. and Meyer, A. (1990). A new approach to the dirichlet domain decomposition method. In: *Fifth Multigrid Seminar, Eberswalde 1990*, (Ed. S. Hengst), Karl-Weierstrass-Institut, Berlin, **R–MATH–09/90**, 1–59.

[2] Meyer, A. (1990). A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, **45**, 217–234.

[3] Nye, A. (1990). *Xlib Programming Manual for Version X11.* O'Reilly & Associates, Inc.

[4] Pester, M. (1991) Implementation und Test paralleler Basisalgorithmen der linearen Algebra. In: *Parallele Datenverarbeitung mit dem Transputer. 2. Transputer–Anwender–Treffen TAT'90. Proceedings X*, (Ed. R. Grebe and C. Ziemann), *Informatik–Fachberichte*, vol. **272**, Springer–Verlag, 111–118.

[5] Rieken, B. and Weiman L. (1992). *Adventures in UNIX Network Applications Programming.* John Wiley & Sons, Inc., New York – Chichester – Brisbane – Toronto – Singapore.

[6] Saad Y. and Schultz M. H. (1989). Data communication in hypercubes. *Journal of parallel and distributed computing*, **6**, 115–135.

[7] Sunderam, V. S. (1992). PVM: A framework for parallel distributed computing. Technical report, Oak Ridge National Laboratory.

[8] Wierse, A. and Rumpf, M. (1992). GRAPE Eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forsch. Entw.*, **7** 145–151.

Other titles in the SPC series:

93_1 G. Haase, T. Hommel, A. Meyer and M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. May 1993.

93_2 M. Pester and S. Rjasanow. A parallel version of the preconditioned conjugate gradient method for boundary element equations. June 1993.

93_3 G. Globisch. PARMESH – a parallel mesh generator. June 1993.

94_1 J. Weickert and T. Steidten. Efficient time step parallelization of full-multigrid techniques. January 1994.

94_2 U. Groh. Lokale Realisierung von Vektoroperationen auf Parallelrechnern. March 1994.

94_3 A. Meyer. Preconditoning the Pseudo-Laplacian for Finite Element simulation of incompressible flow. February 1994.

94_4 M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. (aktualisierte Fassung). March 1994.

94_5 U. Groh, Chr. Israel, St. Meinel and A. Meyer. On the numerical simulation of coupled transient problems on MIMD parallel systems. April 1994.

94_6 G. Globisch. On an automatically parallel generation technique for tetrahedral meshes. April 1994.

94_7 K. Bernert. Tauextrapolation – theoretische Grundlagen, numerische Experimente und Anwendungen auf die Navier-Stokes-Gleichungen. June 1994.

94_8 G. Haase, U. Langer, A. Meyer and S. V. Nepomnyaschikh. Hierarchical extension and local multigrid methods in domain decomposition preconditoners. June 1994.

94_9 G. Kunert. On the choice of the basis transformation for the definition of DD Dirichlet preconditioners. June 1994.

94_10 M. Pester and T. Steidten. Parallel implementation of the Fourier Finite Element Method. June 1994.

94_11 M. Jung and U. Rüde. Implicit Extrapolation Methods for Multilevel Finite Element Computations: Theory and Applications. June 1994.

94_12 A. Meyer and M. Pester. Verarbeitung von Sparse-Matrizen in Kompaktspeicherform (KLZ/KZU). June 1994.

94_13 B. Heinrich and B. Weber. Singularities of the solution of axisymmetric elliptic interface problems. June 1994.

94_14 K. Gürlebeck, A. Hommel and T. Steidten. The method of lumped masses in cylindrical coordinates. July 1994.

94_15 Th. Apel and F. Milde. Realization and comparison of various mesh refinement strategies near edges. August 1994.

94_16 Th. Apel and S. Nicaise. Elliptic problems in domains with edges: anisotropic regularity and anisotropic finite element meshes. August 1994.

94_17 B. Heinrich. The Fourier-finite-element method for Poisson's equation in axisymmetric domains with edges. August 1994.

94_18 M. Pester and S. Rjasanow. A parallel preconditioned iterative realization of the panel method in 3D. September 1994.

94_19 A. Meyer. Preconditoning the Pseudo-Laplacian for Finite Element simulation of incompressible flow. October 1994.

94_20 V. Mehrmann. A step towards a unified treatment of continous and discrete time control problems. October 1994.

94_21 C. He, V. Mehrmann. Stabilization of large linear systems. October 1994.

94_22 B. Heinrich and B. Weber. The Fourier-finite-element method for three-dimensional elliptic problems with axisymmetric interfaces. November 1994.

Some papers can be accessed via anonymous ftp from server **`ftp.tu-chemnitz.de`**, directory **`pub/Local/mathematik/SPC`**. (Note the capital L in Local!)