KfK 4714
Juni 1990

KfK-SUPRENUM-Seminar 19.-20.10.1989

Tagungsbericht

H. Trauboth (Hrsg.)
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE
Institut für Datenverarbeitung in der Technik

KfK 4714

KfK-SUPRENUM-Seminar
19.-20.10.1989
– Tagungsbericht –

H. Trauboth (Hrsg.)

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

KfK-SUPRENUM-Seminar 19.-20.10.1989
– Tagungsbericht –

Zusammenfassung:

Am 19. und 20. Oktober 1989 fand in der Schule für Kerntechnik (SKT) ein Seminar über das SUPRENUM-Projekt statt, welches vom Institut für Datenverarbeitung in der Technik (IDT) des KfK veranstaltet wurde.

Dieser Bericht enthält Zusammenfassungen der wichtigsten Vorträge über Anwendungssoftware.

KfK-SUPRENUM-Seminar 19.-20.10.1989
– Seminar Report –

Abstract:

On October 19 and 20 a seminar on the SUPRENUM project took place at the Schule für Kerntechnik (SKT), organized by the Institut für Datenverarbeitung in der Technik (IDT) of the KfK.

This report contains summaries of the most important papers on application software.

Inhalt:

Vorwort

Die Entwicklung des neuartigen parallelen Superrechners SUPRENUM ging vor rund fünf Jahren vom Institut für Methodische Grundlagen der Gesellschaft für Mathematik und Datenverarbeitung (GMD) bzw. dessen Leiter, Prof. Dr. Ulrich Trottenberg, aus; sie wurde vom BMFT im Rahmen eines Verbundprojekts bis Ende 1989 finanziell gefördert.

Die Hardware des Hochleistungskerns dieses Rechnersystems wurde von Prof. Dr. Wolfgang K. Giloi vom Forschungszentrum für Innovative Rechnersysteme Berlin (FIRST) der GMD entworfen. Die gesamte Hardware und die Systemsoftware (Programmentwicklungswerkzeuge, Simulatoren, usw.) sowie bereits einige wichtige Anwendungsprogramme wurden von den Projektpartnern aus Industrie, Universitäten und Großforschungseinrichtungen in enger Zusammenarbeit unter Federführung der 1986 gegründeten SUPRENUM GmbH, Bonn, fertiggestellt.

Die Projektpartner sind:

- DLR Braunschweig, Göttingen;
- Dornier, Friedrichshafen;
- Prof. Dr. Feilmeier, Junker & Co., München
  (im Auftrag der GMD);
- GMD St. Augustin, Berlin und Karlsruhe;
- KFA Jülich;
- KfK Karlsruhe;
- SIEMENS-KWU, Erlangen;
- Krupp Atlas Elektronik, Bremen;
- Stollmann, Hamburg;
- SUPRENUM GmbH, Bonn;
- TH Darmstadt;
- TU Braunschweig;
- Universitäten Bonn, Düsseldorf und Erlangen-Nürnberg.

Es ist beachtlich, daß ein neuartiges Superrechnersystem vom Konzept bis zur Fabrikation in einer so relativ kurzen Zeit im Verbund von unabhängigen Entwicklungsstellen soweit fertiggestellt werden konnte, daß Ende dieses Jahres der erste SUPRENUM-Rechner ausgeliefert werden kann.

Bei diesem Projekt hat die Wechselwirkung von Algorithmen und Rechnerarchitektur im Vordergrund gestanden. Hardware, System- und Anwendungssoftware sind gleichzeitig entwickelt worden. Ein wichtiges Ziel bei der Entwicklung der Anwendungssoftware besteht darin, moderne numerische Verfahren, wie etwa Multi-Level-Verfahren (z.B. Mehrgitterverfahren), mit den Möglichkeiten paralleler Multiprozessorsysteme zu verbinden. Das KfK leistet hierbei einen Beitrag in der Weiterentwicklung und Parallelisierung eines Particle-in-Cell-Codes (PIC) zur Teilchensimulation in der Plasmaphysik.

Es war nun das Ziel des KfK-SUPRENUM-Seminars, den Stand der Entwicklung, die bisherigen Erfahrungen und die Perspektiven bezüglich des Einsatzes des SUPRENUM-Parallelrechnersystems aus der Sicht des Anwenders von Projektteilnehmern zu erfahren und mit ihnen zu diskutieren. Das wesentlichste Auswahlkriterium für die Vorträge war dabei in erster Linie die Frage, welche der bisher untersuchten oder bereits parallelisierten Aufgabenstellungen für das KfK besonders von Interesse sind und weniger, wie weit diese Arbeiten bereits fortgeschritten sind.

Es sollten solche Mitarbeiter der KfK angesprochen werden, die für die Planung zukünftiger Höchstleistungsrechner verantwortlich sind und solche, die Anwendungssoftware auf den Großrechenanlagen entwickeln. Das Seminar sollte auch im Rahmen der Planung eines Höchstleistungszentrums in der KfK zur Klärung von Fragen zur Softwareentwicklung auf Parallelrechensystemen beitragen.

Dieses Seminar sollte auch die Anwender von Großrechnern anregen, sich stärker mit den angebotenen Superrechner-Architekturen auseinanderzusetzen, um die für ihre spezifischen Bedürfnisse optimale Rechnerarchitektur zu finden (hoher Durchsatz und kurze Antwortzeiten bei minimalen Kosten).

Ich danke den Autoren für ihre spontane Zusage und ihre übersichtlichen Beiträge. Den Herren Dipl.-Math. Manfred Alef und David Seldner gilt mein Dank für die gute Organisation des Seminars.

APPLICATIONS OF SUPERCOMPUTERS IN ENGINEERING:
ALGORITHMS, COMPUTER SYSTEMS AND USER EXPERIENCE
Proceedings of the first International Conference
Southampton, UK, September 1989
Edited by
C.A. Brebbia
A. Peters
ELSEVIER
Co-published with
COMPUTATIONAL MECHANICS PUBLICATIONS

Vector-Supercomputers: Perspectives of Vectorization and Parallelization

F. Hossfeld

Central Institute for Applied Mathematics, Nuclear Research Center (KFA) Jülich, D-5170 Jülich, Fed. Rep. Germany

ABSTRACT

In his Turing Award Lecture 1970, J.H. Wilkinson complained that numerical analysts had failed to influence computer hardware and software in the way that they should, rather than opting out of any responsibility for the design of computing systems. Although science and engineering notoriously have been underestimated as a relevant part of the computer market, in the years since then the situation has changed not only with respect to the colourful spectrum of computers designed for scientific and engineering applications, but also with respect to the pace-making role of supercomputers in the evolution of computer technology as well as at the frontiers of scientific research and development.

Today, the field of high-speed computers and supercomputing applications is dominated by the vector-processor architecture. This paper will give a survey on the architectural principles of vector-supercomputers as well as on the spectrum of real systems available in the market. It will discuss the potentiality and the limitations of vectorization strategies referring to algorithmic, program optimization, and compiler aspects. Recent developments towards multi-vectorcomputer systems give impact to new supercomputing concepts balancing vectorization versus parallel computation by exploiting inherent parallelism through multitasking strategies.

INTRODUCTION

The reason for the continuously increasing need of supercomputers is not primarily the scientific interest in new computer architectures but the fact that physics, technology, chemistry, biology, ecology, and engineering as well as economic and social sciences have been confronted with nonlinear-systems dynamic phenomena which have established barriers, that cannot be overcome just by utilizing today's analytical methods and theoretical knowledge about the behaviour of linear systems (Bishop [1]). The investigation and evaluation of complex nonlinear systems has to rely on the exploitation of the potential provided by numerical simulation and further developments of its methods in order to establish the body of mathematical concepts and theoretical knowledge about complex systems behaviour. The complexity (i.e. the number of degrees of freedom, the dimensionality, in particular the topological and dynamical structure) of many systems in science, technology and society ranges far beyond the available potential of methods and general-purpose computers of today (Berendsen [2]).

Due to this situation a third scientific category has grown during the past decades which may be called "Computational Science" (rather than computational physics) supplementing theory and experiment in a qualitative and methodological manner. Being comparable with an experimental discipline, Computational Science simultaneously extends the analytical techniques provided by theory and mathematics. Computational Science is synonymous with investigating complex systems. Its main instrument is the supercomputer; its primary tool is computer simulation.

SUPERCOMPUTER GENERATIONS

The evolution in the field of supercomputing can be described by generations of supercomputers as is illustrated in (Hwang [3]). While the first generation ranges from the end of the sixties to the middle of the seventies comprising computers like ILLIAC IV, Texas Instruments Advanced Scientific Computer (TI-ASC) and CDC STAR-100 with a peak performance rate of 10 to 60 Mflops (millions of floating-point operations per second, as the generally used 'measure of supercomputer power'), the second generation has been introduced by the delivery of the first CRAY-1 system to Los Alamos National Laboratory in 1976; the peak performance of these second-generation supercomputers, which are primarily represented by CDC CYBER 205, CRAY X-MP, CRAY-2, Fujitsu VP-200/400 and NEC SX-2 in the market, reaches up to almost 2 Gflops (i.e. 2000 Mflops).

In the third generation of supercomputers the two future systems which have been extensively discussed in the 'supercomputer scene' are CRAY-3 with 16 processors which will be based on GaAs technology and, certainly, on the architecture of CRAY-2 and foreseeable extensions yielding 16 Gflops, and ETA-10 which was designed to consist of (up to) 8 processors each similar to the CYBER-205 processor, with new technology and large main memory, targetting for 10 Gflops. The CRAY X-MP series is followed by an innovative series of multiprocessor vector computers, CRAY Y-MP, with promising perspectives, too. Unfortunately, just recently CDC went out of the supercomputer market; thus ETA-10 belongs to the past already. Instead, there are very interesting announcements from Japan: NEC SX-3 and Fujitsu VP-2x00, challenging Cray Research.

All these supercomputers rely on the vector-computing architecture being expanded to the multiprocessor structure in the CRAY X-MP, CRAY-2, CRAY-3, CRAY Y-MP, and ETA-10 as well as in the general-purpose computer series of IBM 3090 which now is available with up to six processors with vector feature for each CPU. Also NEC SX-3 (with 22 Mflops peak performance) and Fujitsu VP-2x00 (with about 4 Gflops peak performance) will be provided with multiple processing units: probably four and six, respectively. The CRAY-3 follow-up CRAY-4 will consist of 64 processors yielding the peak of 128 Gflops and supporting at least 1 Gword of main memory.

Referring to the peak performance rates, one has to be careful in transferring these figures to the real world of supercomputing applications: generally, peak performance rates have the only meaning that 'manufacturers guarantee that you as a user will never get beyond'! This fact is illustrated by data which have been assembled for a broad spectrum of computers by Dongarra with the LINPACK benchmark (Dongarra [4]).

The situation in supercomputing today is dominated by the vector-supercomputers of the market leader, Cray Research (CRI), while the supercomputers of Fujitsu, NEC, and Hitachi come into play as serious competitors and IBM's 3090 with vector features seems to provide more than an appetizer in general-purpose supercomputing on the long term (Lazou [5]).

PIPELINING, CHAINING, UNROLLING

Vector computers owe their power to the technology of high speed circuits, on the one hand, and to the pipeline principle, on the other hand (Hockney [6]). Like in an assembly line, the work of arithmetic operations is divided into subtasks called segments; the successive segments mapped onto hardware represent 'pipelines' or pipelined functional units. Pipelining yields valuable speedup, when many identical operations (e.g. additions) have to be executed with independent data. This prerequisite to feed continuously the pipeline with operands, is usually satisfied with numerical methods in linear algebra and its data structures, the vectors. (Hence, 'vector computers' are synonymous with 'pipeline computers'.) The speedup against a conventional arithmetic processing unit is given, asymptotically, by the number of pipeline segments; the maximum is reached for 'long vectors' when the startup time of the pipelines can be neglected.

A vector processor can consist of parallel pipelined functional units each specialized and dedicated to the processing of a certain arithmetical or logical operation. The alternative provides the supercomputer with one or several parallel multi-purpose pipelines which are functionally identical and which can be switched into different pipeline states to execute different operations.

Whereas the CYBER-205 pipelines communicate directly with the main memory, thus representing a so-called memory-to-memory machine, the CRAY-type vector-supercomputers are provided with fast registers (vector registers) which act as a sort of high-speed memory to feed the functional units and to communicate with the main memory. This vector-computer structure is called 'register-to-register' machine.

In general, data communication turns out to be crucial in order to exploit the supercomputer power. Therefore the structure of the main memory, e.g. the degree of 'interleaving', is quite important. The problem of memory conflicts has been extensively analyzed for instance in (Oed [7], Oed [8]), and strategies have been designed which may be effective, although the problem in general must still be considered a serious one in shared-memory supercomputing.

Many problems cannot use the pipeline principle if the usual sequential algorithm is not restructured (Ortega [9]). The product of a matrix with a vector is a simple example illustrating this. For an (nxn)-matrix A with the elements a_{ik} and a vector x with the elements x_i, the algorithm for the jth element of the resulting vector is

$$c_j = a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n$$

The elements of c are calculated successively. Addition and multiplication alternate, and each product is added to the previous sum. This makes the algorithm unsuitable for the above-mentioned principles of vector computation.

If the algorithm is written for all c_j simultaneously in the following form:

$$\begin{array}{cccccc} c_1 & a_{11}x_2 & a_{12}x_2 & & a_{1n}x_n \\ c_2 & a_{21}x_1 & a_{22}x_2 & & a_{2n}x_n \\ . & = . & + . & + \ldots + & . \\ . & . & . & & . \\ c_n & a_{n1}x_1 & a_{n2}x_2 & & a_{nn}x_n \end{array}$$

then the application of pipelining becomes apparent, if column-wise execution is exploited.

In this way further gain can be achieved, if pipelined functional units can be concatenated to yield a 'superpipeline'; this combination is called 'chaining'. It can be applied if computation consists of 'triads': $z_i = x_i + r_i s_i$, with independent data x_i, r_i, s_i, $i = 1, 2, ..., n$, representing the three vectors x, r, s yielding, finally, z. The result of a multiplication $r_j s_j$ is fed directly from the end of a multiplication pipeline into the first segment of an independent add pipeline and is added to x_j there. The computation therefore has not to wait until all multiplications have been executed before starting the additions; but the additions start as soon as the first product $r_1 s_1$ has left the multiplication pipeline. Hence, the chaining represents an efficient combination of the pipeline principle with the independent operation of different functional units, although the computations to be executed are sequentially dependent with respect to the data.

In (Lubeck [10]) the architectural impact on the performance enhancements as well as the shortcomings and bottlenecks have been extensively investigated for quite a spectrum of vector-supercomputers of today; this report provides broad material which lends itself to gain insight into what vector processing is all about (Schönauer [11]).

Unfortunately, not all computations lend themselves directly to vector processing, although all modern supercomputers are based on the pipeline principle as a major architectural element for achieving high performance; in many cases, therefore, the computation incorporated in a computer program has to be restructured in a way to make it amenable to vectorization. The programming of today's supercomputers almost exclusively relies on FORTRAN (Perrott [12]). Therefore, vectors are rather implicit in loops that manipulate arrays; the vector length is defined by the number of passes through a loop. Strategies for vectorization are necessarily focussing on the analysis and transformation of loop structures. The success of vector processing, however, is strongly limited by the amount of unvectorizable computation within a program; this scalar portion 'contaminates' the optimization of code according to 'Amdahl's Law' which tells us that significant speedup can be obtained only for problems which can be vectorized beyond 90 %. For this reason, the scalar processing power of the vector-supercomputers must not be underestimated in the design and in the assessment of supercomputers.

However, it should be emphasized that in our opinion 'Amdahl's Law' as a very trivial relationship does not cover important aspects of supercomputing evolution; in particular it assumes a fixed problem size, while in practice the problem size is growing with the compute power and the number of processors available, thus yielding potentially a higher degree of vectorization and parallelization. Therefore, 'Amdahl's Law' has to be reconsidered and revised (Gustafson [13]). Although permanent progress has been made with the vectorizing compilers provided by the manufacturers, in many cases the task of program optimization is still up to the programmer. In order to 'squeeze the most out of an algorithm' - at least in FORTRAN - one has to adapt the code to the architecture of the machines in a way which is still far from any software technological paradigm, although very efficient with respect to performance enhancement. For instance, the strategy of unrolling outer FORTRAN DO loops is capable to effectively utilize the vector registers and data-path structure by expanding the scope of the vectorizing process to more than simple vector operations, thus achieving 'supervector performance'.

In (Dongarra [14]) the unrolling strategy has been applied to matrix multiplication, Cholesky decomposition, and LU decomposition. Nowadays, the application of unrolling is a widely used option. A detailed analysis of the state of the art of vector processing comes to the conclusion that the strategies and tools for vectorization, program optimization, and efficient restructuring are getting more and more understood (Detert [15]). Since the community of people using vector-supercomputers is growing rapidly, especially since IBM has added vector features to its recent mainframe system thus broadening the interest in

vectorization strategies to an even bigger user community, however, the need for a coherent body of tools and functions, in particular implemented in intelligent compilers, is urgent.

In the meanwhile, a broad spectrum of numerical methods (and non-numerical, as well) has been published (see, for instance (Ortega [16]); mathematical software can be observed to follow the progress made in vectorization techniques for supercomputers (see, for instance Dongarra [17], Hake [18], and IEEE [19]).

By the trends in vector-computer archtitecture, we are already faced with multiprocessor vector computers well beyond CRAY X-MP/4x and CRAY-2 with four processors. Recent benchmarks on CRAY Y-MP (in comparison with CRAY-2) are quite promising (Steen [20]). Therefore, vectorization versus parallelization is an issue challenging vector processing into parallel (MIMD) processing (Kowalik [21]).

MULTITASKING

CRAY X-MP and CRAY-2 are multiprocessor vector-supercomputers with (up to) four CPUs; CRAY Y-MP is provided with up to 8 CPUs. IBM 3090 can be configured with 6 CPUs at maximum. These supercomputers allow to distribute portions of one program over the different processors and to take advantage of the possibly inherent parallelism in the underlying problem. Therefore, with these computer systems one can exploit parallelism in two ways: by vectorization (referring to each processor involved) and by 'multitasking' of programs.

The Multitasking feature provided by Cray Research for these multi-vectorprocessors can be utilized in three modes:

(1) Macrotasking: Macrotasking exploits coarse-grain parallelism on the subroutine level; different subprograms of one program or program system may be executed on different processors simultaneously and in parallel, if the problem structure allows for partitioning.

(2) Microtasking: With Microtasking which exploits fine-grain parallelism it is possible to execute different parts of one subprogram, e.g. different passes through one DO-loop, in parallel on different processors. This can be achieved by inserting certain preprocessor directives provided by Cray Research; these are translated into assembler code and subroutine calls.

(3) Autotasking: Recently, on CRAY computers running UNICOS automatic parallelization is provided by a third mode of Multitasking (Cray Research [22]).

Experiences with Multitasking strategies and applications on CRAY X-MP are thoroughly discussed in (Hossfeld [23]) where the user interfaces as well as the differences in the concepts and implementations of Macrotasking and Microtasking are analyzed together with parallel linear algebra and graph algorithms and a large application program from hydrodynamic flow simulation. Further references on Multitasking can also be found there.

On the IBM 3090 multiprocessor, Parallel FORTRAN (PF) allows the exploitation of fine-grain as well as coarse-grain parallelism via the definition and parallel execution of FORTRAN tasks; a FORTRAN task is a unit of work, e.g. a chunk of a DO loop, which for execution will be assigned to a processor. The mapping of tasks onto the physical processors is done by the operating system which may be either MVS/XA or VM/XA (Toomey [24]). Parallel FORTRAN involves new language elements, for instance

PARALLEL LOOP for explicit fine-grain parallelization, and supports implicit (e.g. nested) parallelism through the PF compiler by the automatic detection of parallel executable DO loops.

In (Szelenyi [25]), an up-to-date comparison of the parallel processing concepts on CRAY X-MP and IBM 3090 VF multiprocessors is presented. Both concepts permit effective parallelization in the MIMD sense, either for coarse-grain or for fine-grain parallelism. Parallelization turns out to be efficient in many cases applying kernels as well as application routines, although the functionality of the multitasking concepts varies widely over the whole scope of parallelization. Nevertheless, further progress in this field is necessary in order to make parallel processing on multiprocessor vector-supercomputers an instrument which can be easily and efficiently applied on the user level without too much overhead, i.e. effective loss in overall computer power relative to the throughput expectations of the total system.

Multitasking is, in a certain sense, similar or even equivalent to MIMD algorithms; hence, Multitasking can benefit from the developments in this field with respect to algorithmic structures and partitioning concepts.

As in general with parallel processing, the speedup of Multitasking is also said to be controlled by 'Amdahl's Law'. In order to reach worthwhile speedup values, the fraction of parallel work must be very close to 1. This situation is identical to the problem of incomplete vectorization with respect to a single vector processor as discussed earlier. Since there are always portions within a realistic program which cannot be executed in a parallel mode due to unresolvable data dependencies, the parallel fraction will remain remarkably less than unit in many 'real-world' cases; in addition to these causes, the synchronization overhead involved with the Multitasking mode, in particular with Macrotasking, will also decrease the speedup. Therefore the realistically achievable speedups may be quite far from the optimum. Nevertheless, parallel processing on multiprocessor vector-supercomputers is one of the growth paths in high-speed computation of the future.

CONCLUSION

Despite the effectively only slow advances in this field, there is no alternative to parallel processing in order to achieve the further enhancements in computing power which is required for the handling and the solution of the complex application problems which are challenging science and technology today and in the future.

Since, recently, Fujitsu and NEC announced that they will also enter the field of multiprocessor vector-supercomputers with their new products VP-2x00 and SX-3, one might expect that the progress in multitasking concepts and implementation will be stimulated by the strengths of these Japanese manufacturers who certainly will not rely only on the prospective peak performance of the system hardware which is challenging by itself.

In summary, the perspectives of the vector-supercomputers for the next generation - in particular with respect to parallelization while vectorization is getting well understood - can certainly be considered quite promising for the user of high-speed computers. It will be expecially interesting whether the convergence of multiprocessor vector-supercomputer architectures and massively parallel computers will proceed further hence being productively interacting especially on the field of parallelization, or whether massive parallelism will definitely succeed in the vital competition with the vector-supercomputers on the market during the next-generation period. From a user's point of view, one reasonably would like to see both architectures in fruitfully complementing the functions and the power scale required for the manifold of applications - since heterogeneity is

certainly one of the fundamental characteristics of complex problems and one of the fundamental sources of progress as well.

ACKNOWLEDGEMENT

I would like to express my gratitude to Wolfgang Nagel for many discussions and his invaluable contributions to the evaluation of the Multitasking strategies.

REFERENCES

1. Bishop, A., et al., eds. Nonlinear Problems: Present and Future. Mathematics Studies 61, North-Holland, Amsterdam, 1982.

2. Berendsen, H.J.C., et al. Molecular dynamics on CRAY, CYBER and DAP in: J. Kowalik, ed., Proc. NATO Advanced Research Workshop on High-Speed Computation. Lecture Notes in Computer Science, Springer, Berlin, 1984, pp. 425 .

3. Hwang, K., and Briggs, F.A. Computer Architecture and Parallel Processing. McGraw-Hill, New York, 1984.

4. Dongarra, J.J. Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment. ACM SIGARCH Computer Architecture News 16 (1988), No.1, 47.

5. Lazou, Ch. Supercomputers and their Use. Clarendon Press, Oxford, 1986.

6. Hockney, R.W., and Jesshope, C.R. Parallel Computers 2. Adam Hilger, Bristol, 1988.

7. Oed, W. Untersuchungen von Zugriffen auf den Arbeitsspeicher in Vektorrechnersystemen. Dissertation, RWTH Aachen; Report Jül-1994, Kernforschungsanlage Jülich, 1985.

8. Oed, W., and Lange, O. On the effective bandwidth of interleaved memories in vector processor systems. IEEE Trans. Computers 34 (1985), 949.

9. Ortega, J.M. Introduction to Parallel and Vector Solution of Linear Systems. Plenum Press, New York, 1988.

10. Lubeck, O.M. Supercomputer-Performance: The Theory, Practice, and Results. Report LA-11204, Los Alamos National Laboratory, January 1988.

11. Schönauer, W. Scientific Computing on Vector Computers. North-Holland, Amsterdam, 1987.

12. Perrott, R.H. Parallel Programming. Addison-Wesley, Wokingham, 1987.

13. Gustafson, J.L., Montry, G.R., and Benner, R.E. Development of Parallel Methods for a 1024-Processor Hypercube. SIAM J. Sci. Stat. Comput. 9 (1988), No. 4, 609.

14. Dongarra, J.J., and Eisenstat, S.C. Squeezing the most out of an algorithm in CRAY FORTRAN. ACM Trans. Math. Software 10 (1984), 219.

15. Detert, U. Programmiertechniken für die Vektorisierung. Praxis der Informationsverarbeitung und Kommunikation - PIK 10 (1987), 189.

16. Ortega, J.M., and Voigt, R. Solution of Partial Differential Equations on Vector and Parallel Computers. SIAM Review 27 (1985), 149. - Also: A Bibliography on Parallel and Vector Numerical Algorithms. ICASE Report I-3, NASA-Langley Research Center, 1987.

17. Dongarra, J.J., et al. An Extended Set of FORTRAN Basic Linear Algebra Subprograms. ACM Trans. Math. Software 14 (1988), 1.

18. Hake, J.-Fr., and Homberg, W. Linear algebra software on a vector computer. Parallel Computing 10 (1989), No. 1, 65.

19. IEEE, Scientific Supercomputer Subcommittee of the Committee on Communications and Information Policy, United States Activities Board. Software for Supercomputers. IEEE Computer, December 1988, 70.

20. van der Steen, A.J., and van der Pas, R.J. A family portrait: Benchmarktests on a Cray Y-MP and a Cray-2S. Technical Report TR-30, Academic Computing Centre, University of Utrecht, January 1989.

21. Kowalik, J., ed. Parallel MIMD Computation: The HEP Supercomputer and its Applications. MIT Press, Cambridge/Mass., 1985.

22. Cray Research Inc. Autotasking User's Guide. SN-2088, 1988.

23. Hossfeld, F., Knecht, R., and Nagel, W. Multitasking: Experiences with Applications on a CRAY X-MP. Parallel Computing, 1989 (in print).

24. Toomey, L.J., et al. IBM Parallel Fortran. IBM Systems Journal 27 (1988), No. 4, 416.

25. Szelényi, F., and Nagel, W. A Comparison of Parallel Processing on CRAY X-MP and IBM 3090 VF Multiprocessors. Proceedings of the International Conference on Supercomputing ICS '89, 5-9 June 1989, Crete/Greece (preprint).

Computational Fluid Dynamics on MIMD Computers

M. LÖTZERICH, B. WAGNER,

Dornier Luftfahrt GmbH, Friedrichshafen

EINLEITUNG

Supercomputer sind heute schon in vielen Industriezweigen eingeführt und werden in großen Umfang für technische Berechnungen eingesetzt. Weltweit werden 43 % der verfügbaren Supercomputerleistung in der Industrie genützt, 36 % bei staatlichen Institutionen und 21 % im akademischen Bereich. Die industriellen Hauptanwendungsbereiche sind Chemie, Physik, Strukturberechnung, Kerntechnik, Verfahrenstechnik, Strömungsmechanik und CAD (Computer Aided Design). Es besteht ein stetig wachsender Bedarf nach höherer Supercomputerleistung, weil sich einerseits die Anwendungsgebiete ausdehnen und neue Anwendungsbereiche hinzukommen und weil andererseits verfeinerte Modellbildungen einschließlich verfeinerter Diskretisierung für dreidimensionale nichtlineare Probleme benötigt werden. Außerdem kommen ständig neue numerische Methoden aus dem Entwicklungsstadium heraus in den produktiven Einsatz im Rahmen von Projektarbeiten, so daß kurze Antwortzeiten unabdingbar werden. Wettbewerbsfähigkeit und die Durchführbarkeit vieler Projekte können daher von der Verfügbarkeit geeigneter Supercomputer abhängen.

Die nun schon traditionellen Supercomputer sind sogenannte Vektorrechner, die eine wesentliche Leistungssteigerung gegenüber konventionellen, sogenannten Skalarrechnern erreichen, indem durch die sukzessive Anwendung einer Reihe von Rechenoperationen auf die Elemente von indizierten Variablen (Folge der Elemente einer indizierten Variablen = "Vektor") unter geeigneter Verknüpfung der Teiloperationen die Rechenzeit erheblich verkürzt werden kann, d.h. die Rechenarbeit wird sozusagen "am Fließband"

durchgeführt. Nachdem auf diesem Gebiet ein Standard erreicht wurde, der an die technologischen Grenzen heutiger Rechner-Hardware stößt, können drastische Beschleunigungen der Rechengeschwindigkeit kurzfristig hauptsächlich durch Parallelschaltung von Prozessoren (= Rechenwerke) erreicht werden. Die etablierten Großrechnerfirmen haben diesen Weg schon seit einigen Jahren durch Parallelisierung einiger weniger Prozessoren begonnen, wobei in aller Regel an einem zentralen Kernspeicher festgehalten wird, auf den alle Prozessoren zugreifen (shared memory). Daneben gibt es aber auch schon seit mehreren Jahren Rechnerkonzepte, bei denen eine sehr große Anzahl von Prozessoren parallel arbeiten, wobei auch hierbei die einzelnen Prozessoren in vielen Fällen als Vektorrechner ausgelegt sind. Solche Lösungen lassen sich jedoch in der Regel nicht mehr mit einem gemeinsamen zentralen Kernspeicher lösen, sondern erfordern getrennte Kernspeicherteile in Verbindung mit jedem einzelnen Prozessor (distributed memory). Die Hauptschwierigkeit bei einer solchen Lösung liegt dann in der Herstellung einer geeigneten Kommunikationsstruktur zwischen den Prozessoren, die mittels Bussystemen oder durch direkte Vernetzung hergestellt werden kann. Beim SUPRENUM-Projekt, das in Deutschland unter Förderung des BMFT realisiert wird, erfolgt die Kommunikation über zwei hierarchische Ebenen, nämlich einen Clusterbus, der die Kommunikation zwischen bis zu 16 in einem Cluster zusammengefaßten Prozessoren sicherstellt, und einen zweidimensionalen Busnetz, das die Cluster untereinander verbindet.

Im folgenden wird, basierend auf Erfahrungen im Rahmen des SUPRENUM-Projekts über ein Konzept zur Lösung nichtlinearer partieller Differentialgleichungen berichtet und über erste Erfahrungen, die damit unter Verwendung eines Simulationsprogrammsystems gewonnen wurden. Die betrachteten nichtlinearen partiellen Differentialgleichungen sind die Grundgleichungen der Strömungsmechanik, jedoch sind die Überlegungen auch auf andere Systeme partieller Differentialgleichungen entsprechend über-

tragbar. Selbst bei linearen partiellen Differentialgleichungen kann der hier in der Strömungsmechanik beschrittene Weg Vorteile bieten, weil er den Rechenaufwand gegenüber den Randelement-Methoden reduzierem kann, wenn eine sehr feine räumliche Auflösung (viele Unbekannte) erforderlich ist. Diesbezügliche Erfahrungen liegen z.B. für die Maxwell-Gleichungen vor. Natürlich können auch andere technische Probleme, wie z.B. die Lösung von großen Systemen gewöhnlicher Differentialgleichungen effizient mit Parallelrechnern in Angriff genommen werden, jedoch sollen die speziellen Fragen dieser Probleme hier nicht weiter erörtert werden.

Schließlich sei hier noch ausdrücklich betont, daß bei diesen Überlegungen von einer funktionalen Trennung zwischen der Aufbereitung der Eingabedaten (Preprocessing), insbesondere für die Geometriedaten komplexer Konfigurationen, den eigentlichen Rechenläufen zur Lösung des mathematisch-numerischen Problems und der Ergebnisauswertung (Postprocessing) ausgegangen wird. Während für Pre- und Postprocessing die Kapazität moderner Workstations hervorragend geeignet erscheint, dürften für die Lösung der eigentlichen numerischen Aufgaben auch langfristig Hochleistungsrechner der hier betrachteten Klasse benötigt werden.

PARALLELISIERTE LÖSUNGSVERFAHREN FÜR DIE NICHTLINEAREN PARTIELLEN DIFFERENTIALGLEICHUNGEN DER STRÖMUNGSMECHANIK

Schon seit der Einführung der Vektorrechner haben in der Strömungsmechanik numerische Rechenverfahren, die auf regulären Rechennetzen mit Rechteck- (zweidimensional) bzw. Hexaeder-Struktur (dreidimensional) im Indexraum basieren, hohen Stellenwert, weil sie eine effiziente Vektorisierung der Rechenprozedur ermöglichen. Zugleich besteht aus physikalischen Gründen die Forderung nach konturangepaßten Rechengittern, so daß man im technisch üblichen Fall komplexer Geometrie des um- oder durchström-

ten Körpers mit einem derartigen Rechennetz schnell auf Schwierigkeiten stößt, das Rechengebiet vernünftig aufzulösen. Aus dieser Schwierigkeit ergab sich entweder die Hinwendung zu unstrukturierten Netzen, wie sie bei der Strukturberechnung üblich sind, oder die Einführung einer Unterteilung des gesamten Rechengebiets in eine Reihe einzelner, regulärer, in willkürlicher Form miteinander verknüpfter Rechenblöcke.

Der letztere Weg der sogenannten "blockstrukturierten Netze" wurde weitaus häufiger beschritten und liegt den folgenden Überlegungen und Ergebnissen zugrunde. Fig. 1 verdeutlicht diesen Weg am Beispiel einer Flügel-Rumpf-Kombination mit Triebwerksgondeln in der Flugzeugaerodynamik. Die Figur deutet zugleich das Lösungsprinzip mit Hilfe eines Mehrgitterverfahrens an, bei dem jeweils auch in gröberen Rechengittern Lösungsschritte erzeugt werden, wobei jedes gröbere Gitter durch Weglassen jeder zweiten Netzlinie aus dem nächstfeineren Gitter entsteht, also durch Maschenweitenverdoppelung in jeder Indexzählrichtung. Kommunikation zwischen den Netzblöcken ist nur im feinsten Rechengitter vorgesehen, da in den gröberen Gittern nur Korrekturterme zur Dämpfung langwelliger Störungen berechnet werden, die im konvergierten stationären Zustand der Lösung verschwinden müssen. Infolge dessen erweist sich diese Blockeinteilung zunächst als ein entscheidendes Mittel, um die Kernspeicherbegrenzung konventioneller Rechner zu umgehen. In jedem Iterationsschritt des Rechenverfahrens können nämlich die Netzblöcke sukzessiv abgearbeitet werden, so daß der verfügbare Kernspeicher nur die Blockgröße begrenzt, aber nicht die Größe des gesamten Rechennetzes.

Mit dem Aufkommen der Parallelrechner erweist sich diese Vorgehensweise auch als ein sehr geeigneter Weg, durch Zuteilung je eines Rechenblockes zu einem Prozessor eine sehr effiziente Parallelisierung des Rechenverfahrens zu erreichen. Natürlich müssen nach jedem Rechenschritt einige Randdaten der Blöcke an die

angrenzenden Nachbarprozessoren übertragen werden. Bei vorgegebener Leistungsfähigkeit der Prozessoren und des Kommunikationssystems kommt es dann für die Erreichung einer optimalen Effizienz des Gesamtsystems offenbar auf das Verhältnis der Randdatenmenge zur Datenmenge innerhalb eines Blockes an, d.h. auf das Verhältnis der Blockoberfläche zum Blockvolumen. Die lokal einem Prozessor zugeordnete Kernspeichergröße sollte dann hinreichend groß gewählt werden, um einen optimalen Wert dieses Verhältnisses zu ermöglichen. Es ist leicht vorstellbar, daß eine Maschine mit sehr vielen Prozessoren (z.B. SUPRENUM-Prototyp: bis 256 Prozessoren; SUPRENUM-Serienmaschine: bis 1024 Prozessoren) für sehr umfangreiche Rechennetze besonders wirtschaftlich arbeitet. Andererseits eröffnet sich damit aber auch die Chance, durch Wahl einer geeigneten Anzahl von Prozessoren die Rechnergröße den Problemen des Benutzers optimal anzupassen. Solche "maßgeschneiderten" Lösungen stellen eine besondere Marktchance der Parallelrechner dar und könnten in Zukunft die Anschaffung preisgünstiger dedizierter Rechner stimulieren, also eine Trendwende der (zumindest teilweisen) Abwendung von großen Universalrechnern verursachen.

Im Rahmen des durch den Bundesminister für Forschung und Technologie geförderten Vorhabens SUPRENUM werden bei Dornier entwickelte Rechenprogramme zur Lösung der Euler-Gleichungen für reibungsfreie und der Navier-Stokes-Gleichungen für reibungsbehaftete Strömungen parallelisiert. Es handelt sich um Rechenprogramme für kompressible zwei- und dreidimensionale Strömungen mit einem algebraischen Turbulenzmodell für den Fall reibungsbehafteter turbulenter Strömung. Die Lösungsalgorithmen waren bereits an die oben diskutierte Blockstruktur angepaßt, d.h.: sie sind unabhängig von einer speziellen Topologie formuliert. Zur Generierung der entsprechenden Rechengitter sind verschiedene Netzgeneratoren entwickelt worden, inbesondere ein sehr flexibler, interaktiver Netzgenerator, der ausgehend von Geometriedaten, die z.B. aus einer CAD-Darstellung entnommen werden, die

rechnerunterstützte Erstellung und Kontrolle beliebiger Gitter in konstruktiver, effizienter Bildschirmarbeit erlaubt. Neben der Umströmung von Flugzeug- und Raumfährenkonfigurationen, der Schiffshydrodynamik und dem Gebiet der Innenströmungen bildet die Kraftfahrzeugaerodynamik einen Schwerpunkt der Strömungsberechnung bei Dornier, auf den auch das Hauptgewicht im Rahmen des SUPRENUM-Vorhabens gelegt wird.

Die Parallelisierung der Codes erforderte die Erarbeitung eines optimalen Kommunikationskonzeptes für den notwendigen Datenaustausch zwischen den Prozessoren. Da die Hardware bisher noch nicht verfügbar war, würden diese Arbeiten mit Hilfe eines Software-Simulationssystems für MIMD-Rechnerarchitekturen (MIMD = Multiple Instructions/Multiple Data, d.h.: mit der Möglichkeit zur Verteilung weitgehend voneinander unabhängiger Aufgaben auf die Einzelprozessoren) durchgeführt, das im Rahmen des SUPRENUM-Projektes zur Verfügung steht. Damit wurden eine Reihe von Euler- und Navier- Stokes-Simulationen durchgeführt, wobei diese Untersuchungen wegen des beachtlichen Rechenaufwands und des beschränkten Speicherraums im Simulationssystem auf verhältnismäßig einfache zweidimensionale Probleme beschränkt werden mußten. Aus einer Reihe von Euler- und Navier-Stokes-Beispielsimulationen ergab sich eine wechselseitige Abhängigkeit von Kommunikationsstruktur und Konvergenzrate. In Fig. 2 sind für die Umströmung eines Kreiszylinders mit Symmetriebedingung im inkompressiblen Bereich die Isobaren einer Euler-Lösung (Kurven konstanten Druckbeiwerts c_p mit guter Symmetrie, d.h.: keine Strömungsablösung), das Geschwindigkeitsfeld einer Navier-Stokes-Lösung bei hoher Reynoldszahl ($Re = 4 \cdot 10^6$, turbulent; Strömungsablösung nach dem Dickenmaximum mit Totwasserbildung) sowie die zugehörigen Konvergenzraten (quadratischer Mittelwert der Dichteänderungen über alle Rechenzellen) dargestellt, wobei der Mehrgitteralgorithmus zur Konvergenzbeschleunigung eingesetzt wurde. Während sich für die Euler-Lösung infolge des nahezu isotropen Gitters (quadratische Zellen auch in Oberflächennähe)

eine gute und gleichmäßige Konvergenz zeigt, verschlechtert sich die Konvergenzrate der Navier-Stokes-Lösung demgegenüber erheblich, bedingt durch die notwendigerweise anisotrope Struktur in Oberflächennähe (flache, langgestreckte Zellen für eine gute Auflösung der viskosen Wandgrenzschicht). Als Folge dieser Verhältnisse kann, da die Blockgrenzen (angedeutet durch die dicken Linien in den Ergebnisbildern) im allgemeinen nicht im wandnahen Bereich liegen, die Kommunikation für anisotrope Problemstellungen bis zu einem gewissen Grad reduziert werden (z.B. nur Kommunikation im feinsten Gitter im Fall des Mehrgitterverfahrens wie in Fig. 1 skizziert), ohne daß die Gesamtkonvergenz negativ beeinflußt wird, weil die großen Unterschiede in den lokalen Zeitschritten (zur Beschleunigung in der expliziten Zeitschrittmethode verwendet) immer für eine ausreichende Konvergenz an den Blockgrenzen sorgt. Das bedeutet, daß eine reduzierte Kommunikation zwischen den Blöcken für Euler-Lösungen zu einer deutlichen Verschlechterung der Konvergenzrate führt, während sich dies auf Navier-Stokes-Lösungen weniger auswirkt, da in diesen die Konvergenzrate durch Entwicklung in den wandnahen Schichten bestimmt wird. Eine weitere Möglichkeit zur Optimierung des Kommunikationsaufwandes besteht darin, in kritischen Blöcken (Grenzschicht-/ Totwasserbereich) zusätzliche Unterzyklen einzuführen. Kommunikation erfolgt dann nur zwischen den Blöcken, die an dem jeweiligen Unterzyklus beteiligt sind. Auf diese Weise wird sowohl die Kommunikation reduziert als auch eine bessere Konvergenzrate erzielt.

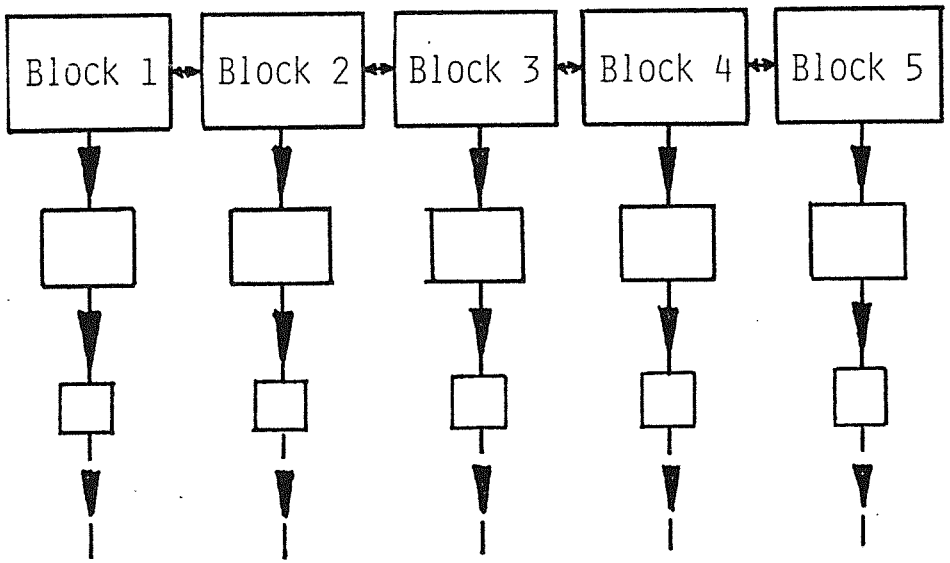
Um eine ausgeglichene Lastverteilung auf die Prozessoren zu erzielen, muß jedoch sowohl die Aufteilung des Gebietes als auch die Verteilung der Blöcke auf die Prozessoren an die Zahl der Unterzyklen angepaßt werden. Die Flexibilität der Blockstruktur, Blocksegmentierung und lokale blockweise Netzverfeinerung, wurde ferner für die Navier-Stokes-Lösung (Fig. 2) in der Weise genutzt, daß entsprechend den zu erwartenden Gradienten im Strömungsfeld, die Blöcke 1, 2 und 3 im Grenzschichtbereich sowie Block 5 im Totwasser eine feinere Diskretisierung als Block 4 aufweisen.

Aufgrund der vorstehenden Diskussion, die auf Simulations-Erfahrungen basiert, ist es plausibel, daß die praktischen Anforderungen an die Hardware erheblich detaillierter sind, als es erste Grundsatzüberlegungen (s.o.) vermuten lassen.

Daher sollten im nächsten Schritt des Suprenum/Genesis-Projektes realistische Problemstelungen (Fig. 3 und 4) auf der Hardware bearbeitet werden, um auf diese Weise weitere Erfahrungen bezüglich des Hardwarekonzeptes zu machen. Prinzipiell stehen auch für dreidimensionale Probleme parallelisierte Programmversionen zur Verfügung, die jedoch wegen der bereits erwähnten Grenzen des Simulationssystems noch nicht erprobt werden konnten.

SCHLUSSFOLGERUNGEN

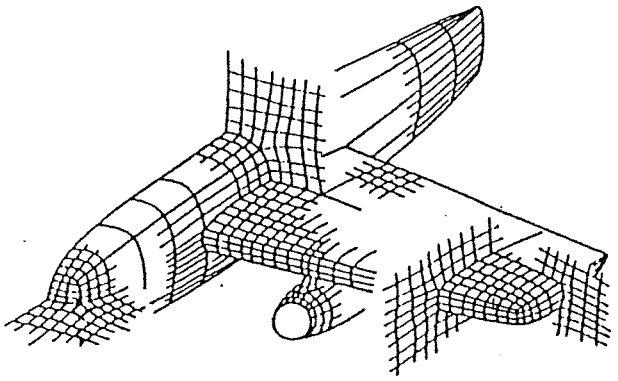
Der Umgang mit modernen Parallel-Rechnern bzw. entsprechende Simulationsergebnisse zeigen, daß die Anforderungen an solche Rechnerarchitekturen nicht allein vorab aus theoretischen Überlegungen hergeleitet werden können. Erst aus der Bearbeitung praktischer technischer Probleme ergibt sich ein realistischer detaillierter Anforderungskatalog bezüglich der Abstimmung von Prozessorleistung, lokaler Speichergröße und Kommunikationsrate. Die im Rahmen des SUPRENUM-Projektes gewonnenen Erfahrungen bezüglich der numerischen Lösungen nichtlinearer partieller Differentialgleichungssysteme verdeutlichen aber auch, daß derartige Parallelrechner für die Probleme der Strömungsmechanik effizient genützt werden können, wenn die Programmstruktur hinreichend auf die Parallelisierung abgestimmt wird. Wegen des voraussichtlich günstigen Preis-Leistungs-Verhältnisses solcher Rechner eröffnet sich für den Industrieeinsatz die Möglichkeit, dedizierte Rechner wirtschaftlich einzusetzen, die durch eine sinnvolle Auswahl der Prozessorzahl für den speziellen Anwender "maßgeschneidert" werden können.



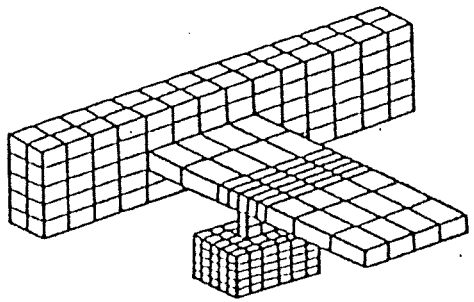
— 1rst GRID LEVEL
(arbitrary block sizes)

— 2nd GRID LEVEL
SIZE = $\frac{1}{8}$ BLOCK 1rst LEVEL

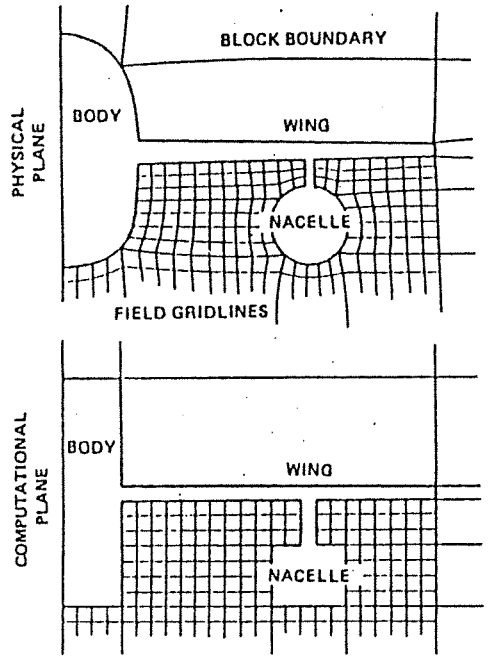
— 3rd GRID LEVEL
SIZE $\frac{1}{64}$ BLOCK 1rst LEVEL



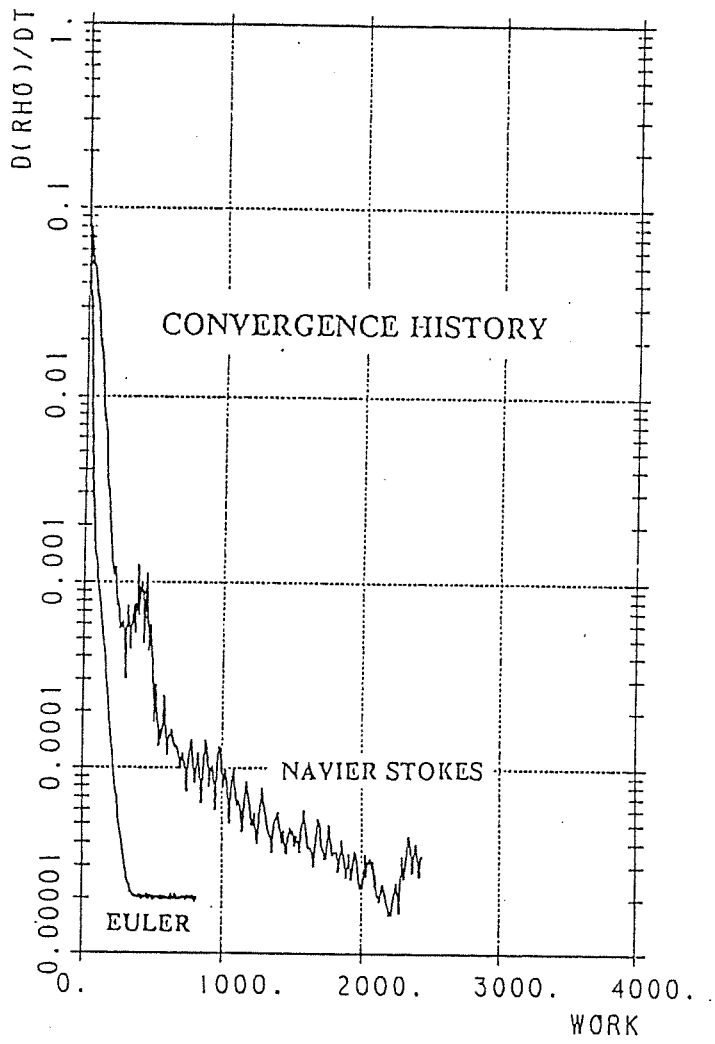
PHYSICAL SPACE

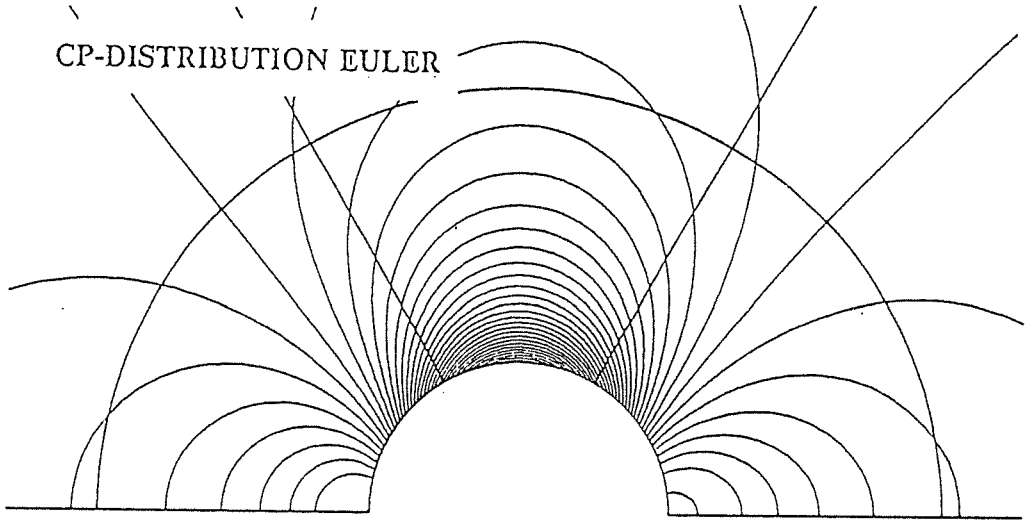


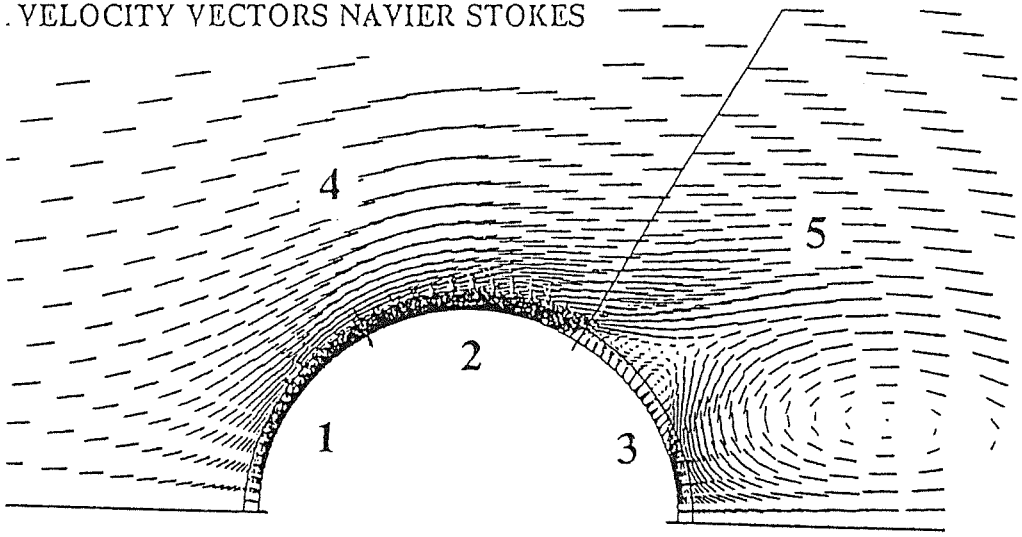
COMPUTATIONAL SPACE

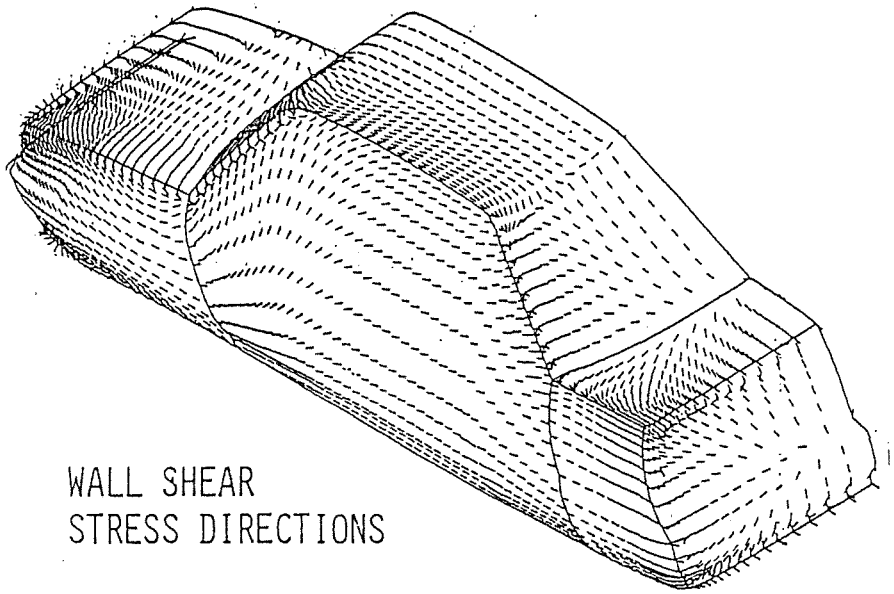


ARBITRARILY BLOCKED MULTIGRID STRATEGY


D(RHO)/DT
1.
0.1
0.01
0.001
0.0001
0.00001
CONVERGENCE HISTORY
NAVIER STOKES
EULER
0.
1000.
2000.
3000.
4000.
WORK

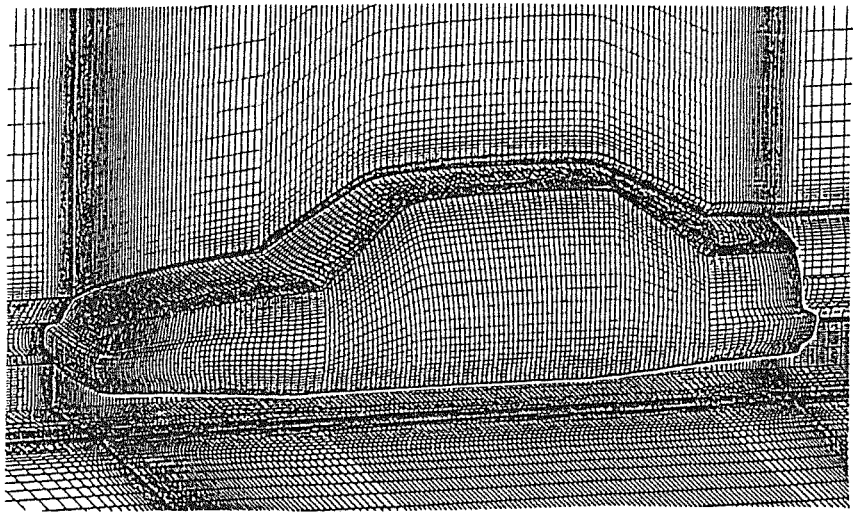

CP-DISTRIBUTION EULER

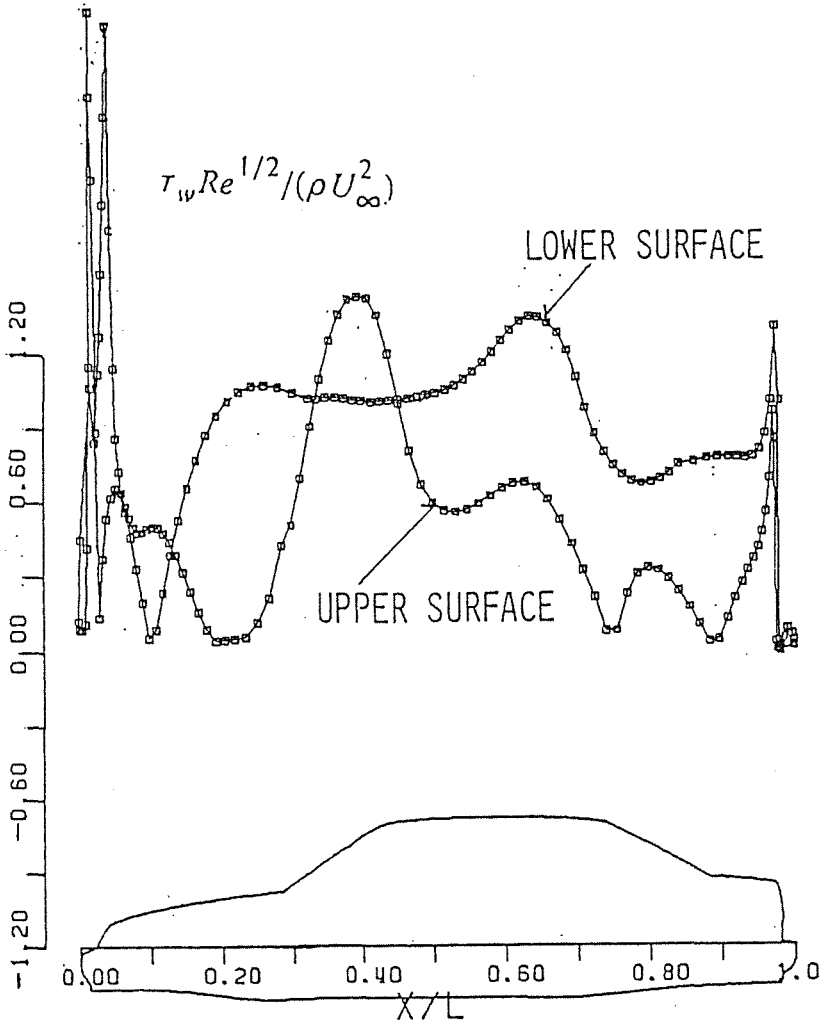

VELOCITY VECTORS NAVIER STOKES
1
2
3
4
5



WALL SHEAR
STRESS DIRECTIONS

SURFACE MESH (INCLUDING SYMM.PLANE & GROUND)





SKIN FRICTION IN SYMMETRY PLANE

HERMES: STRÖMUNGSFELDBERECHNUNG

ZIELE:

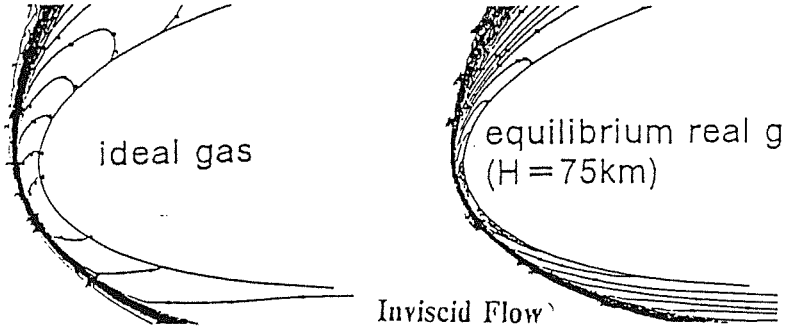
- KONFIGURATIONSANALYSE & -VERBESSERUNG
- STRÖMUNGSFELD VOR DEM HECK
- RUDERWIRKSAMKEIT (RUMPFKLAPPE, ELEVONS, WINGLET-KLAPPEN)

METHODIK:

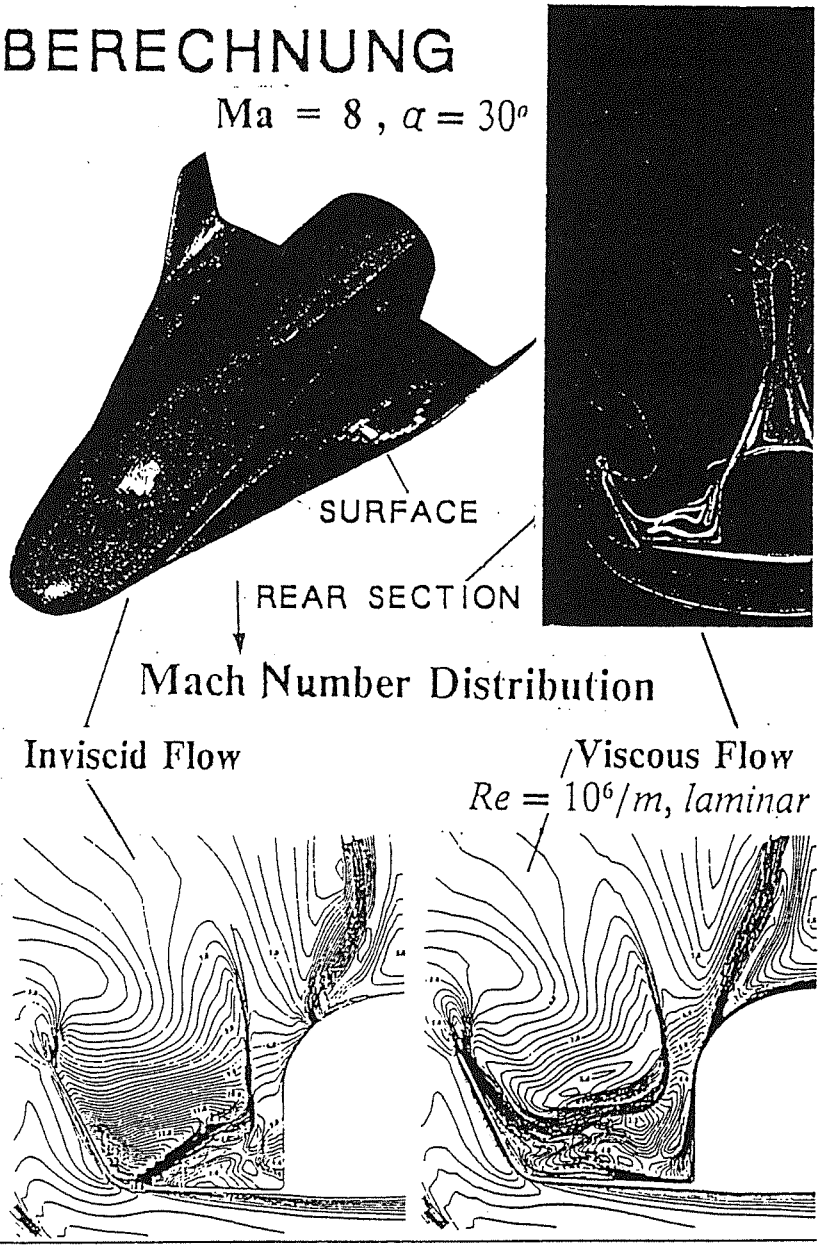
- EULER (REIBUNGSFREI)
- NAVIER-STOKES (VISKOS)
- REALGASEFFEKTE (GLEICHGEW.)
- WÄRMEÜBERGANG, STRAHLUNGSRANDBED.
- TRANSITION? TURBULENZ?

PRINZIPIELLE ERGEBNISSE:

- VISKOSE EINFLÜSSE UND REALGAS WICHTIG
- SENSITIVITÄT DER LÖSUNGEN



Mach Number Distr. at the nose; Ma = 20, $\alpha = 30°$



The L_iSS Package

Johannes Linden
Guy Lonsdale
Anton Niestegge
Hubert Ritzdorf
Anton Schüller
Barbara Steckel
Klaus Stüben

Institut für Methodische Grundlagen
der Gesellschaft für Mathematik
und Datenverarbeitung mbH,
Schloß Birlinghoven
Postfach 1240
D-5205 Sankt Augustin 1
West Germany

1 Overview

The following is a short and preliminary description of the L_iSS package - a package of programs designed primarily for the solution of the laminar, incompressible Navier-Stokes equations on general, two-dimensional domains. The wider applicability of the package as a whole, given user modification of application specific components, or indeed of parts of the package, will be made clear below and, in more detail, in the final L_iSS User's Guide. For example, as the package is constructed in a modular form much of which is independent of the Navier-Stokes application, use of the package for the solution of some other partial differential could be readily achieved by modification of a minimal number of routines.

The underlying numerical methods and approaches used within the package can be summarised as follows. The domain is first sub-divided into blocks, each of which is then ascribed a logically-rectangular, boundary-fitted (and in general non-orthogonal) grid. The use of block-structured grids allows for a greater geometrical flexibility and also provides the basis for parallelization of the Navier-Stokes solution. The composite boundary-fitted grid over the whole domain is generated by solving a system of transformed elliptic partial differential equations using a fast multigrid algorithm. The Navier-Stokes equations are discretized on the given grid using a finite-volume formulation and the resulting nonlinear system of algebraic equations (or systems, when an implicit time-marching technique is used for the time-dependent Navier-Stokes equations) is solved using a further multigrid algorithm.

The package comprises eight programs, the use of which will be discussed in greater detail in Section 2, what follows is only a synopsis of the tasks performed by the various programs.

- Pre-Processing

  - $L_iSS-PRE$ performs pre-processing of user-supplied information regarding the solution domain: its use allows for a relatively simple specification of the geometry, boundary grid spacing and types of boundary condition to be applied.
  - The programs $L_iSS-PIN$ and $L_iSS-BIN$ are both grid generation programs but based on different elliptic generating equations: Poisson's equation and the Biharmonic equation, respectively.
  - A further sub-division of the user defined blocks, to provide the greater number of blocks required for the parallel computation, is carried out by the program $L_iSS-CUT$.

  - The graphics program $L_iSS - PLOT$ can be used during pre-processing in order to examine the results of the above programs.

- **Computation**

  - $L_iSS - NVI$ performs the solution of the Navier-Stokes equations using a discretization formulated on the boundary-fitted grid and methods based on the multi-block structure.

- **Post-Processing**

  - Graphical output of results is carried out by the post-processing programs $L_iSS - PLOT$, $L_iSS - TRACE$.

In addition the program $L_iSS - FORMAT$ is an auxiliary program which can be used to aid file handling.

The above processes are quite independent; inter-communication is via pre-specified interfaces (in the form of input and output files). Any program in the package could be replaced by a user-supplied alternative provided that compatible interfaces are used.

2 Usage

The aim of this section is to describe the use of the whole package in order to solve a complete Navier-Stokes problem; from domain specification through to production of graphical output. As mentioned in Section 1 the components of the package are independent, implying that, given an adherence to the interface structures, the user could use only certain components within some other application. A description of the input files to, and output files from each of the programs will now be given in sequence; i.e. starting with the definition of the domain, through generation of the grid, specification of exactly which problem is to be solved (and perhaps using which method variant) and ending with graphical output. An example of this whole sequence will be given in Section 3.

2.1 Pre-Processor

The purpose of the pre-processor is to define the contour of a general domain and its block structure, the boundary conditions of the associated problem to be solved and the boundary point distribution which is used by the grid generator to create the block-structured grid. Input to the pre-processor

program $L_iSS - PRE$ is via the input file PREDAT, containing a sequence of commands which provide the definitions listed above. Output from $L_iSS - PRE$ is in the form of two new files : the grid control file GRSDAT (required as input to all the remaining programs) and the boundary point distribution file CNTDAT.

2.2 Grid Generation

The package includes a default choice of grid generator $L_iSS - PIN$ and a second, currently as a prototype version, $L_iSS - BIN$. The difference between the two solvers lies in the type of elliptic differential equation which is used to generate the grid, $L_iSS - PIN$ uses a system of transformed Poisson's equations while $L_iSS - BIN$ uses a system of transformed Biharmonic equations. The major disparity between the generated grids is that the use of $L_iSS - BIN$ produces a grid which has, in addition to the given boundary point distribution, 'near' orthogonality of grid lines at boundaries. Currently, the input and output requirements of both grid generators are identical. The required input files are those which can most easily be produced by the preprocessor, namely GRSDAT and CNTDAT. Output from the grid generator is a file containing the coordinates of the grid points : GRXDAT.

2.3 Processing for Parallelization

The number of process blocks required for parallelization is typically much larger than the number of user-defined blocks. The semi-automatic, interactive program $L_iSS - CUT$ operates on the files GRSDAT and GRXDAT to produce a further sub-division of the user-defined blocks. The boundary-fitted grid is unchanged, only the partitioning into blocks is refined in order to obtain full usage of the available nodes.

2.4 Navier-Stokes Solution

Input to the Navier-Stokes solver $L_iSS - NVI$ is in the form of the previously generated block-structured grid, completely described by the files GRSDAT and GRXDAT, a parameter specification file NVIDAT, and user supplied subroutines giving non-standard boundary values and conditions (when required). The specification of exactly which problem is to be solved, e.g. time-dependent or time-independent Navier-Stokes, is achieved via parameter selections given in the NVIDAT file. Output from the code is via screen ouput (for convergence histories etc.) and files GRFDAT, MSKDAT. GRFDAT contains the solution (or solutions in the time-dependent case)

in the form of grid functions, i.e values of the unknowns at all grid points. The file MSKDAT contains information relating to the grid and block structure for use by the graphics routines (required only for debugging purposes). Parallelization of the code is achieved by use of the SUPRENUM Communications Library, for which information is available in separate documentation and will not be repeated here nor in the L_iSS User's Guide.

2.5 Visualization

The graphics program $L_iSS-PLOT$ provides plots of the domain, its block structure and the corresponding grid, contour plots of all grid functions, velocity plots using arrows, and streamline plots. The required input depends upon the desired graphical output. The file GRSDAT together with one of the files containing the boundary point distribution, for instance CNTDAT, are necessary to allow for the display of the contour of the domain, including the assigned boundary condition codes, and its block structure. Input of the files GRSDAT and GRXDAT is required for the display of the domain (as above) together with the generated grid. The files GRSDAT and GRFDAT should be input to allow the full range of graphics options. The separate, interactive program $L_iSS-TRACE$ can be used to perform particle tracing, it requires the files GRSDAT and GRFDAT. For both programs, additional (optional) information for use by the graphics routines can be input using the file POSTDAT.

3 Example

The model problem of steady flow over a plate in a straight channel is employed to illustrate the use of the various package components. While being geometrically very simple, it is sufficient to exemplify the input requirements of the programs in the package. When using the whole package (without special requirements for boundary conditions) the user need only supply the files PREDAT and NVIDAT.

The following is the file PREDAT used as input to the pre-processor $L_iSS-PRE$. The domain is defined in terms of two blocks, each of which is constructed from a sequence of commands defining points, lines, segments and eventually block sides. The associated boundary conditions are given by the command $BCDEF = 0$ in the CONTROL section, and the //10, //20 terms appended to the commands defining sides 1 and 2 in blocks 1 and 2, respectively. These boundary condition specifications assign prescribed boundary condition codes to non-interface segments of the domain boundary;

in this example the BCDEF command sets a default code zero (corresponding to a solid wall) which is superceded only on the two sides given codes 10, 20 (referring to inflow and outflow conditions respectively). Details of the available commands which may be used in the PREDAT file are contained in the *LiSS* User's Guide.

```
$$ ========================================================
$$
$$ EXAMPLE : CHANNEL WITH PLATE AS OBSTRUCTION
$$
$$        *---------0---------*   --   0.5
$$        |         :         |
$$        |         :         |
$$        |         |         |   --   0.5*H
$$        |         |         |   --   0.0
$$        |         |         |   --   -0.5*H
$$        |         :         |
$$        |         :         |
$$        *---------0---------*   --   -0.5
$$
$$        |         |         |
$$        L1        0         L2
$$
$$ ========================================================

&CONTROL
  HEADER= 'EXAMPLE: "PLATE" ON FILE LISS.PRE(PLATE)'
  BCDEF=0; REFINE =4
&ENDCONTROL

&PARAMETER
$$ ---- DOMAIN PARAMETERS ---------------------------------
   L1 = -2.0; L2 = 4.0; H = 1.0/3.0

$$ ---- POINTS -------------------------------------------
  E1=(L1, -0.5); E2=(L2, -0.5);   E3=(L2, 0.5);   E4=(L1, 0.5)

  P1=(0.0,-0.5); P2=(0.0,-0.5*H); P3=(0.0,0.5*H); P4=(0.0,0.5)
```

```
$$ ---- NUMBER OF SUBDIVISIONS -----------------------------

$$ NX = VECTOR WITH NUMBER OF SUBDIVISIONS IN X-DIRECTION
$$      NX (1) = NUMBER OF SUBDIVISION BEFORE PLATE
$$      NX (2) = NUMBER OF SUBDIVISION BEHIND PLATE
$$ NY = VECTOR WITH NUMBER OF SUBDIVISIONS IN Y-DIRECTION
$$      NY (1) = NUMBER OF SUBDIVISION BELOW AND ABOVE PLATE
$$      NY (2) = NUMBER OF SUBDIVISION ON PLATE
$$ NYTOT = TOTAL NUMBER OF Y SUBDIVISIONS

   DIM NX(2),NY(2)
   NX (1:2) =24, 48 ; NY (1:2) = 4, 4
   NYTOT = 2*NY(1) + NY(2)
   HXP = 0.25*L2/NX(2); HYP = 0.5/NYTOT

$$ ------------------ INTERFACE SEGMENTS  -------------------
  SEGL = &SEG (&LINE(P1,P2) // &GM(NY(1),,HYP) )
  SEGU = &SEG (&LINE(P3,P4) // &GM(NY(1),HYP) )

$$ ------------------ PLATE ---------------------------------
  SEGP = &SEG (&LINE(P2,P3) // &GM(NY(2),HYP,HYP) )

&ENDPARAMETER

$$ -------- FIRST BLOCK (USER DEFINED NUMBER = 1) -----------
&BLOCK 1
  INTERFACE SEGL,SEGU
  SIDE1 = &SEG (&LINE(E1,E4) // NYTOT) //10
  SIDE2 = SEGL,SEGP,SEGU
  SIDE3 = &SEG (&LINE(E1,P1) // &GM(NX(1),,HXP) )
  SIDE4 = &SEG (&LINE(E4,P4) // &GM(NX(1),,HXP) )
&ENDBLOCK
$$ -------- SECOND BLOCK (USER DEFINED NUMBER = 2) ----------
&BLOCK 2
  SIDE1 = SEGL,SEGP,SEGU
  SIDE2 = &SEG (&LINE(E2,E3) // NYTOT) //20
  SIDE3 = &SEG (&LINE(P1,E2) // &GM(NX(2),HXP) )
  SIDE4 = &SEG (&LINE(P4,E3) // &GM(NX(2),HXP) )
&ENDBLOCK
$$ ========================================================
```

Using the output from the pre-processor, files GRSDAT and CNTDAT, as input to the graphics program $L_iSS-PLOT$ allows the user to check the specification of the domain and its associated structures; see Figure 1. The files GRSDAT and CNTDAT may now be used as input to one of the grid generators, $L_iSS-PIN$ or $L_iSS-BIN$ to produce the boundary-fitted grid which will be stored in file GRXDAT. Figure 2 shows the grid produced by $L_iSS-PIN$; where only every fourth line has been plotted for clarity.

As stated in Section 2.4, the input requirements to the program $L_iSS-NVI$ are the file GRXDAT and a parameter file NVIDAT. The complete range of selection parameters allowed in file NVIDAT is given in the L_iSS User's Guide. However, for most parameters it suffices to employ default values so that the user may omit their specification; in the extreme case only the Reynold's number needs to be specified. This is indeed the case for this example, the NVIDAT file which follows simply specifies the Reynold's number and provides text which will appear on the screen output.

```
$$ Example input file NVIDAT ====
&NVI
   RE=100.0

&PRINT
**** PLATE ****
  BC CODES : INFLOW - 10
             OUTFLOW - 20
               WALLS - 0
&ENDPRINT
&ENDNVI
```

Input of the files GRFDAT, GRSDAT to programs $L_iSS-PLOT$, $L_iSS-TRACE$ allows for plots of the domain and grid as before but also contour plots of the grid function solutions, streamlines, velocities (using arrows to indicate the strength and direction of the velocities) and particle tracing. Examples of the streamline and contour plotting are shown in Figures 3 and 4.

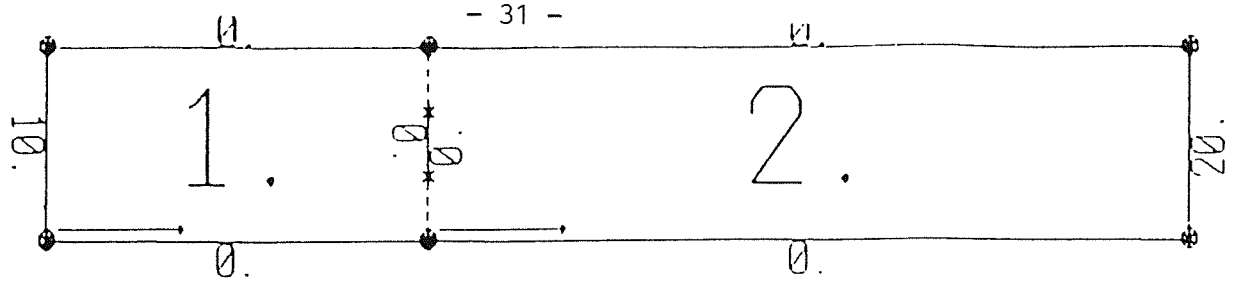


Figure 1. Domain Plot.

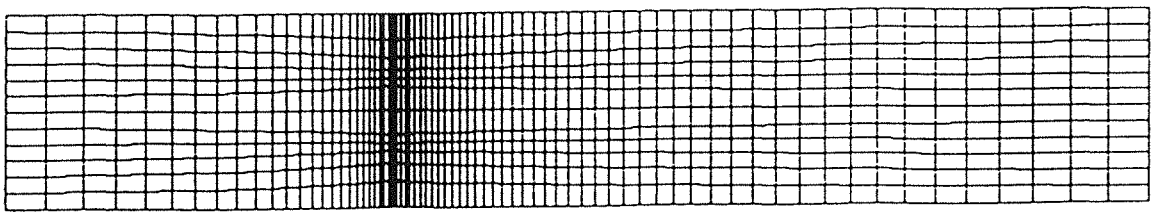


Figure 2. Grid Plot.

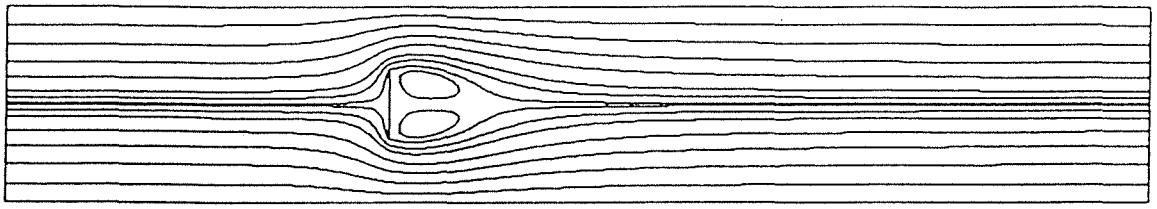
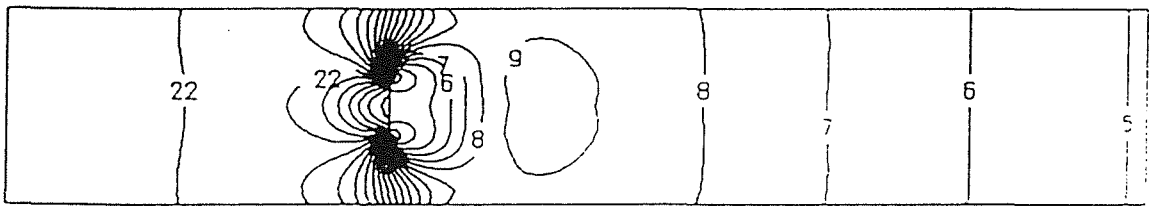


Figure 3. Streamlines.



Label key of P					
1	-0.28320	11	0.44761	21	1.17842
2	-0.21012	12	0.52069	22	1.25150
3	-0.13704	13	0.59377	23	1.32459
4	-0.06396	14	0.66686	24	1.39767
5	0.00913	15	0.73994	25	1.47075
6	0.08221	16	0.81302	26	1.54383
7	0.15529	17	0.88610	27	1.61691
8	0.22837	18	0.95918	28	1.68999
9	0.30145	19	1.03226		
10	0.37453	20	1.10534		

Figure 4. Pressure Contours.

METEOROLOGISCHE STRÖMUNGEN

Perspektiven des Modellierers atmosphärischer Strömungen bei der Nutzung von Parallelrechnersystemen

Thomas Gerz

DLR, Institut für Physik der Atmosphäre
D-8031 Oberpfaffenhofen

In der Erforschung geophysikalischer und auch technischer Strömungsvorgänge spielen neben der Betrachtung rein dynamischer Probleme zunehmend auch das Transportverhalten von Beimengungen (Schadstoffen) im Fluid sowie chemische und phasenumwandelnde Prozesse (Ozonzyklen, Verbrennungen, Wolkenmikrophysik) und deren Beeinflussung durch die Strömung (Turbulenz) eine wichtige Rolle.

Naturgemäß spielen sich solche Prozesse (Reaktionen) bei sehr kleinen räumlichen und oft auch kleinen zeitlichen Skalen ab. Will man solche Vorgänge nun einigermaßen realistisch numerisch simulieren, so steigt der Bedarf an Rechnerkapazität (Größe und Schnelligkeit) einmal durch die erforderliche hohe räumliche und zeitliche Auflösung der Prozesse, zum andern aber auch durch die wachsende Zahl der zu integrierenden Bilanzgleichungen - zu den drei Geschwindigkeitsfeldern und der Temperatur kommen dann je nach Problemstellung noch drei (bei einfachen chemischen Reaktionen), zehn (in mikrophysikalischen Vorgängen) oder 100 Skalare (bei komplexen chemischen Reaktionsketten) hinzu.

1. Grobstruktursimulation (LES) mesoskaliger Wetterabläufe

Anhand von Beispielen aus den numerischen Arbeiten des Instituts für Physik der Atmosphäre (IPA) der DLR in Oberpfaffenhofen sollen die Perspektiven aufgezeigt werden, die sich hier durch den Einsatz von Parallelrechnern wie SUPRENUM ergeben.

Meteorologische Forschungsarbeit wird im IPA vorwiegend im Bereich der Meso- und Mikroskala der Atmosphäre geleistet (Schumann *et al.* 1987). Erstere erstreckt sich über einen Längenbereich von 250 bis 2.5 Kilometer und eine Zeitskala von drei Stunden bis zwei Minuten; die Mikroskala schließt daran an. Die numerische Berechnung mesoskaliger atmosphärischer Vorgänge geschieht mit der Methode der Grobstruktursimulation („large-eddy simulation", LES). Hier werden nur die größten, das Gros der Energie enthaltenden Skalen aufgelöst, während die kleineren Skalen, die sog. Feinstruktur mittels eines Schließungsansatzes parametrisiert werden müssen („subgrid-scale model", SGS). Je nach Anwendungsgebiet kommen dabei unterschiedlich aufwendige Modelle zum Einsatz. Prinzipiell gilt, daß bei Simulationen im Bereich der unteren Mesoskala die verwendeten SGS-Modelle voraussetzen, daß die kleinste noch aufgelöste Skala schon im Trägheitsbereich des Turbulenz-

spektrums liegen muß (siehe z.B. Schmidt & Schumann 1989 als Beispiel für Simulationen **dynamischer** Vorgänge in der konvektiven Grenzschicht). Physikalisch wird also eine statistische Entkopplung der Feinstruktur von der Grobstruktur vorausgesetzt. Dies wiederum ermöglicht den Einsatz von Parallelrechnern, da die (oft diagnostischen) Bilanzgleichungen des SGS-Modells nur lokal (pro Gitterpunkt) gelöst werden müssen und so parallel zur LES gelöst werden können (vgl. auch Solchenbach & Thole 1988).

Ähnliches gilt auch für die Simulation von Bewegungsvorgängen in der **feuchten** Atmosphäre, in der es zu Wolkenbildung und -auflösung mit gegenseitiger Wechselwirkung zwischen Dynamik der Luftbewegung und Thermodynamik der Wolke kommt (Hauf & Höller 1987) und für die Simulation **chemischer** Reaktionen und den Transport der Stoffe durch die Strömung (Schumann 1989). Phasenumwandelnde Prozesse in der Wolke sowie chemische Prozesse laufen bei kleinen Skalen ab verglichen mit den Skalen der Dynamik der Strömung. Diese Mikrophysik/Chemie wird durch etwa zehn oder mehr Bilanzgleichungen beschrieben, die lokal pro Gitter- und Zeitpunkt zu lösen sind. Bei einer Simulation auf Rechnern mit einer Rechnereinheit werden zuerst die dynamischen Gleichungen einschließlich Advektion und Diffusion der mikrophysikalischen und chemischen Stoffe integriert und in einem zweiten Schritt die mikrophysikalischen und chemischen Reaktionen berechnet. Auf Parallelrechnern könnten beide Schritte gleichzeitig ablaufen. Die Nutzung ist aber nur dann effizient, wenn beide Schritte ungefähr gleich lange Bearbeitungszeiten benötigten. Für die meisten meteorologischen Anwendungen ist dies jedoch nicht gegeben.

2. Direkte numerische Simulation (DNS) von Turbulenz

Die bis jetzt vorgestellten Arbeiten meteorologischer Simulationen legen eine Parallelisierung auf einer physikalischen Ebene nahe, die schon mit relativ wenigen Zentralrechnereinheiten zu realisieren ist. Im folgenden soll nun anhand des vom Verfassers bearbeiteten Gebietes der direkten numerischen Simulation von Turbulenz gezeigt werden, wie durch Parallelisierung auf der geometrischen Ebene (Abbildung des Rechengitters auf das Gitter der Parallelrechnerkonfiguration) eine erhebliche Auflösungssteigerung bei erschwinglichen Integrationszeiten erreicht und also numerisch bisher unzugängliche Gebiete der Physik turbulenter Strömungen auch mit dem Rechner erforscht werden können. Auch für die LES-Modelle ist die Konzeption eines Parallelrechners wie SUPRENUM mit sehr vielen Rechnereinheiten nur dann voll ausschöpfbar, wenn eine Parallelisierung des Programmcodes auf der geometrischen Ebene erfolgt.

Das zeitliche und räumliche Verhalten turbulenter Strömungen wird durch die Navier-Stokes-Gleichungen für die Impulsfluktuationen und ggf. durch Bilanzgleichungen für Skalare, z.B. Temperaturfluktuationen, beschrieben. Um in einem numerischen Simulationsmodell nach Wahl geeigneter Rand- und Initialisierungsbedingungen durch zeitliche Integration dieses partiellen Differentialgleichungssystems Erkenntnisse über die Physik turbulenter Strömungen zu gewinnen, müssen alle interessierenden Skalen aufgelöst werden. Dabei macht die große Bandbreite der Skalen turbulenter Strömungen Probleme. Das Modellgebiet und sein Gitternetz müssen so beschaffen sein, daß die räumlichen Ausdehnungen des zu untersuchenden Phäno-

mens gut in ihm Platz haben und auch stark genug aufgelöst werden. Sei L die charakteristische Abmessung der größten Wirbel und Δx die der kleinsten noch interessierenden, dann ist das Verhältnis $L/\Delta x$ eine Abschätzung für die Mindestauflösung M im Modell.

Die Methode der vollen oder *direkten* Simulation (DS) turbulenter Strömungen, die sich zum Ziel setzt, Turbulenz direkt, d.h. über die zeitliche Integration des Differentialgleichungssystems untersuchen zu können, setzt eine Auflösung voraus, die *alle* Skalen vom energiereichsten Wirbel bis zum Wirbel, dessen Energie in Wärme dissipiert wird, umfaßt. Es soll also auch die turbulente Feinstruktur aufgelöst werden, die in den Grobstruktursimulationsmodellen als *subskalige* Turbulenz nur indirekt über Parametrisierungen behandelt werden kann.

Die Behandlung turbulenter Strömungen macht vor allem wegen der großen Bandbreite der an ihnen beteiligten Skalen Probleme. Allen Turbulenztheorien liegt die Annahme zugrunde, daß sich Turbulenz im wesentlichen in großskalige, von Rand- und Anfangsbedingungen beeinflußte Bewegungen und in statistisch davon unabhängige kleinskalige Bewegungen unterteilen läßt. Entsprechend gliedert man das räumliche Energiespektrum $\hat{E}(k)$ einer stark turbulenten Strömung in drei Wellenzahlbereiche (Abb. 1):

1. Großskalige Bewegungen mit integralen Längen ℓ bzw. kleinen Wellenzahlen $k = O(k_p)$, $k_p \approx 1/\ell$ liegen im energieenthaltenden Teil des Spektrums, der stark anisotrop und nicht universell ist.

2. Bei viel kleineren Skalen mit Wellenzahlen, die im Bereich $k_p \ll k \ll k_d = 1/\eta$ liegen (η = Kolmogorofflänge), ist das Energiespektrum der turbulenten Fluktuationen nahezu universell und wird durch das Kolmogoroffspektrum $\hat{E}(k) \propto k^{-5/3}$ beschrieben. Dies ist der Inertial- oder Trägheitsbereich des Spektrums.

3. Die kleinsten Skalen mit typischen Längen η und sehr großen Wellenzahlen $k > O(k_d)$ gehören zum universellen Dissipationsbereich, in dem die Energie exponentiell mit k abnimmt.

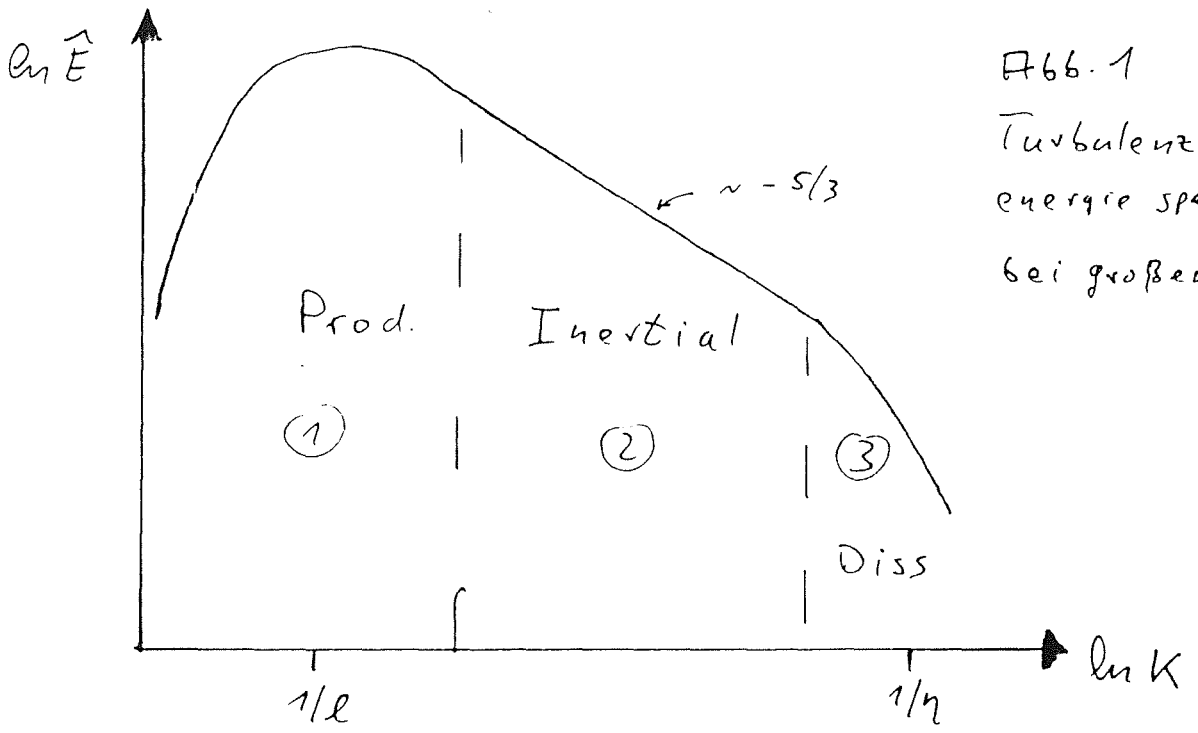


Abb. 1
Turbulenz-
energie spektrum
bei großen Re

Im Bereich 1 liegt die Produktion der turbulenten kinetischen Energie durch äußere Kräfte, wie z.B. Scherung. Typische Maßzahlen sind hier die mittlere Geschwindigkeitsfluktuation v und die integrale Länge ℓ. Die Bereiche 2 und 3 bilden zusammen den universellen Gleichgewichtsbereich des Spektrums, in dem die turbulenten Fluktuationen sich statistisch im Gleichgewicht zwischen dissipativen Kräften am oberen Ende und Trägheitskräften am unteren Ende des Wellenzahlenbereichs befinden. Er wird allein durch die Dissipationsrate ε und die kinematische Viskosität ν beschrieben, woraus für die zugehörige Längenskala die Größenordnungsabschätzung $\eta \approx (\nu^3/\varepsilon)^{1/4}$ folgt. Im Falle eines Gleichgewichts zwischen Produktion und Dissipation kann über die Transfer- oder Umwälzzeit $t = \ell/v$ die Dissipationsrate durch Skalen der Produktion abgeschätzt werden, $\varepsilon \approx v^3/\ell$.

Um in einer turbulenten Strömung einen Dissipationsbereich des Spektrums im Gleichgewicht anzutreffen, müssen die Bereiche der Produktion und Dissipation genügend weit auseinanderliegen, d.h. die Wellenzahlen, bei denen typischerweise Turbulenzenergie produziert wird, müssen deutlich kleiner sein als die Wellenzahlen, bei denen Energie dissipiert wird, $1/\ell \ll 1/\eta$. Daraus folgt für die auf die integralen Maße bezogene Reynoldszahl die Abschätzung

$$Re_\ell^{3/4} \equiv \left(\frac{v\ell}{\nu}\right)^{3/4} \gg 1, \tag{1}$$

d.h. die Strömung muß genügend turbulent sein. Um den Inertialbereich mit typischen Wellenzahlen $k = (\ell\eta)^{-1/2}$ anzutreffen, sind noch einmal um das Quadrat höhere Reynoldszahlen nötig als zur Etablierung des Dissipationsbereichs (Batchelor, 1953).

Da es bei der direkten Simulation *per definitionem* keine subskalige Turbulenz gibt, muß die Viskosität der Strömung so stark bzw. die Reynoldszahl klein genug sein, damit alle nicht aufgelösten Skalen dissipiert werden. Die physikalisch relevanten Längen sind ℓ und η. Wegen der (üblicherweise benutzten) periodischen Ränder und aus Gründen der statistischen Signifikanz muß das Modellgebiet deutlich größer als der größte Wirbel sein. Reynolds (1989) schlägt für isotrope, abklingende Turbulenz die Mindestgröße für die Modellskala $L = 4\,\ell$ vor. Um Wirbel der Skala η aufzulösen, muß die Maschengröße des Gitternetzes $\Delta x = \eta/2$ sein. Hieraus folgt für die benötigte Auflösung M die Abschätzung

$$M = \frac{L}{\Delta x} = f\,\frac{\ell}{\eta} \approx f\,\frac{\ell(v^3/\ell)^{1/4}}{\nu^{3/4}} \approx f\left(\frac{\ell v}{\nu}\right)^{3/4} = f\,Re_\ell^{3/4} \tag{2}$$

resultiert. Der Faktor $f = 8$ gilt für isotrope Turbulenz, bei Scherturbulenz kann $f = 2$ angesetzt werden (Reynolds 1989). Um also den Dissipationsbereich als Teil des universellen Gleichgewichtsbereichs des Gesamtenergiespektrums auflösen zu können, folgt aus (1) und (2), daß die Mindestauflösung pro Raumrichtung

$$M = f\,Re_\ell^{3/4} \gg 1 \tag{3}$$

und, um auch den Inertialbereich zu erfassen, sogar

$$M^{1/2} = f^{1/2}\,Re_\ell^{3/8} \gg 1 \tag{4}$$

sein muß. Mit heute zur Verfügung stehenden Rechnern (CRAY-XMP, CRAY-2) können das Dissipationsspektrum mit $Re_\epsilon \approx 130$ und M = 160 sehr gut aufgelöst und detailierte Studien dieses Bereichs des Spektrums zwecks Feinstrukturmodellierung durchgeführt werden (Gerz *et al.* 1989). Abb. 2 zeigt beispielhaft die räumliche Energieverteilung in einem (x,z)-Schnitt nach Initialisierung der turbulenten Geschwindigkeitsfelder und die dazugehörigen Spektren der Energie und der Dissipationsrate für eine Auflösung mit M = 64 Gitterpunkten und $Re_\lambda = 25$. In Tab. 1 sind für verschiedene Werte der Auflösung der benötigte Gesamtspeicherplatz und die CPU- und I/O-Zeiten für typische Simulationen stabil geschichteter, homogener Scherturbulenz auf verschiedenen CRAY-Rechnern aufgelistet.

M	3D-Feld	13*3D (A)	+ Code	= Summe
64	287.628	3.739.164	400.000	4.139.164
96	941.388	12.238.044	340.000	12.578.044
128	2.197.260	28.564.380	370.000	28.934.380
160	4.251.852	55.274.076	488.000	55.762.076

Tab. 1a. Benötigter Gesamtspeicherplatz in Worten.
3D-Feld = $(M+2)\cdot((M+2)^2+2)$ Worte,
Maximum im Hauptspeicher der CRAY-XMP 2/16 = 15.990.784 Worte;
Maximum im SSD-128 = 131.040 Blöcke oder 67.092.480 Worte;

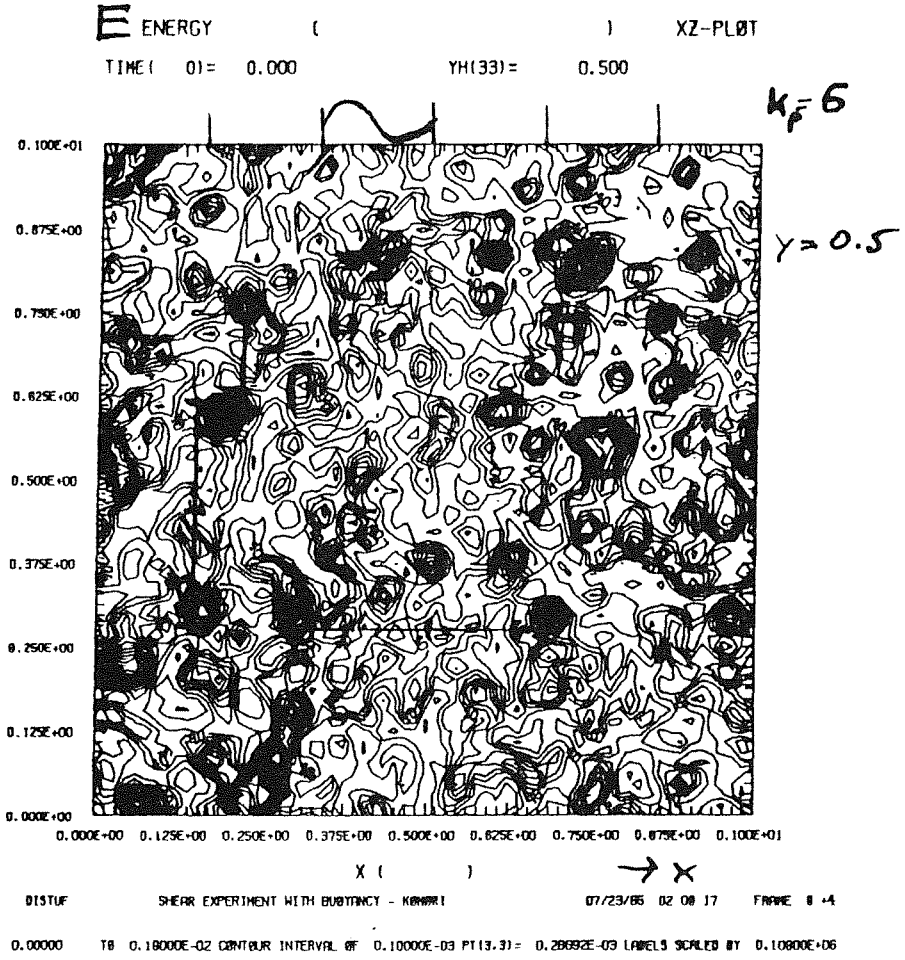
M	Puffer A	n	CPU	CPU/($n\,M^3$)	I-O	I-O/($n\,M^3$)	Q
CRAY 1S							
* 64	0.7 MW	768	49:29	14.75 μs	9:14:33	165.27 μs	11.21
CRAY XMP 2/16							
64	4.3 MW	1280	55:55	10.00 μs	0	0 μs	0
96	14.1 MW	960	2:01:38	8.59 μs	0	0 μs	0
128	11.8+17.4 MW	2560	11:11:16	7.50 μs	7:47	0.09 μs	.01
160	7.0+48.3 MW	3840	22:43:45	5.20 μs	35:05	0.13 μs	.03
CRAY 2							
64	4.0 MW	1280	1:00:04	10.74 μs	0	0 μs	0
128	30.8 MW	768	4:13:51	9.46 μs	0	0 μs	0

Tab. 1b. Statistik typischer CRAY- CPU- und I/O- Zeiten. Typische Werte für den Puffer A , die CPU- und die („waiting for") I-O-Zeiten und deren Verhältnis Q = I-O/CPU bei Simulationen auf CRAY-1S-, CRAY-XMP-2/16- und CRAY-2-Rechenanlagen für verschiedene Auflösungsgrade M. Die Zahl der Zeitpunkte, über die integriert wurde, ist n.

Initialized field of the turbulent Kinetic energy:

(x,z) cross section

$t = 0$



$M = 64$

$Re_\lambda \approx 25$

Spectra of E and ε at $t=0$:

spectral energy density:

$$\hat{E}(k) = \frac{1}{2\pi}\frac{3}{2}v^2\frac{k}{k_p^2}e^{-k/k_p}$$

$$\hat{\varepsilon}(k) = 8\pi^2\nu\int k^2\hat{E}(k)\,dk$$

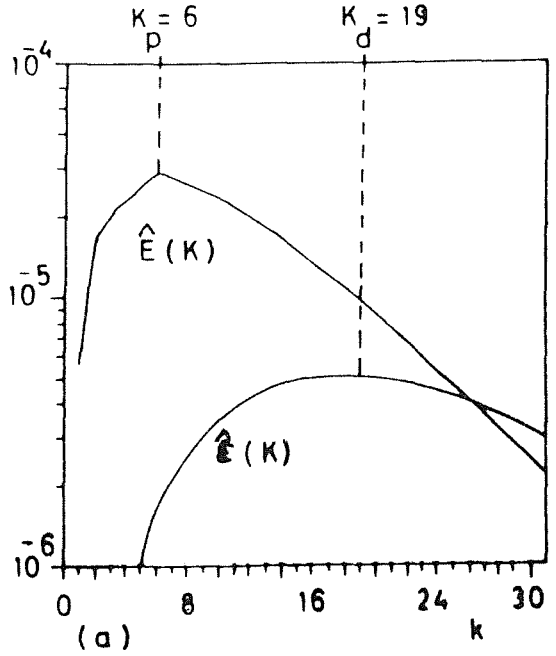


Abb. 2

Heute noch nicht direkt auflösbar ist der von der Kolmogorovschen Theorie vorhergesagte Inertialbereich des Turbulenzenergiespektrums. Das Interesse, dies eines Tages zu schaffen, liegt zum einen in dem Wunsch der Bestätigung des universellen Gleichgewichtsbereichs auch durch Simulationen. Zum anderen aber kann dann gezielt (und ohne Rückgriff auf Schließungsannahmen) die Veränderung des Inertialbereichs durch von außen auf das Fluid wirkende Kräfte wie Scherung und thermische Schichtung und die damit auftretende Anisotropie in diesen Skalen untersucht werden. Schließlich sind dann auch für die Feinstrukturansätzte in den Grobstrukturmodellen allgemeinere Parametrisierungen ableitbar, da hier i.a. bis zum Inertialbereich aufgelöst wird.

Um die Existenz des universellen Gleichgewichtsbereichs der turbulenten Feinstruktur nachzuprüfen, untersuchte Champagne (1978) die Energiespektren aus verschiedenen Laborströmungen und aus Turbulenzmessungen der atmosphärischen Grenzschicht. Er konnte einen Inertialbereich im Gesamtspektrum der turbulenten Strömung dann feststellen, wenn die auf die Taylormikrolänge λ bezogene Reynoldszahl Re_λ bei 200 und darüber lag. Mit $Re_\ell \approx Re_\lambda^2/50$ ergeben sich dann für Re_ℓ Werte in der Größenordnung von 800. Direkte numerische Berechnungen machen also gemäß Beziehung (2) eine Auflösung von ungefähr 300 (Scherturbulenz, f = 2) bis 1200 (Isotropie, f = 8) Gitterpunkten pro Richtung erforderlich, die auch der Abschätzung (4) genügten. Unter Außerachtlassung der Speicherplatzprobleme sind für Simulationen mit 320^3 und 512^3 Maschenpunkten der Platzbedarf und die Rechenzeiten hochgerechnet und in Tab. 2 aufgeführt, die auf heutigen CRAY-XMP- Computern zu erwarten sind. Hier wird deutlich, daß mit den heute verfügbaren Großrechnern nur mäßig turbulente, dreidimensionale Strömungen direkt simuliert werden können, die keinen Inertialbereich aufweisen.

M	Puffer A	n	CPU	CPU/($n M^3$)
CRAY XMP 2/16				
320	426 MW	7680	15.2 d	5 μs
512	1.745 MW	12288	99.3 d	5 μs

Tab. 2. Wie Tab. 1b für geschätzte Werte typischer CRAY-CPU- und I/O- Zeiten.

Mit Rechnern des Typs SUPRENUM wird es nun erstmals möglich sein, homogenturbulente Strömungen mit einer Anzahl von Gitterpunkten in der Größenordnung direkt zu simulieren, die bis zur Auflösung des gesamten Energiespektrums, *einschließlich* des Inertialbereichs vorzudringen vermag. Grundlegende Theorien über homogene und isotrope Turbulenz können damit überprüft, der Einfluß äußerer Kräfte wie Scherung und Auftrieb auf die Ausbildung des Inertialbereichs können dann studiert werden. Tab. 3 gibt die zu erwartenden Rechenzeiten von Simulationsläufen mit dem SUPRENUM-Rechner wieder, wenn die bekannten Zahlen aus Tab. 1 unter Berücksichtigung der angestrebten Parallelstruktur des Programms auf unterster (geometrischer) Ebene gemäß Solchenbach & Thole (1988) hochgerechnet werden. Die Simulationszeiten für die angestrebten Auflösungen rücken mit SUPRENUM demnach deutlich in den Bereich des Durchführbaren.

M	Puffer A	n	CPU
128	29 MW	3072	2.5 h
320	426 MW	7680	1.4 d
512	1.745 MW	12288	9.1 d

Tab. 3. Wie Tab. 2 für Rechnungen auf SUPRENUM.

Ein erster Schritt wäre es, mit $M \approx 300$ hoch auflösende, dreidimensionale, direkte numerische Simulationen inkompressibler, homogener, abklingender Turbulenz mit der Einwirkung von Scherung durchzuführen und dabei die drei Bereiche Produktion, Trägheit und Dissipation im Energiespektrum zu identifizieren. Hieraus gewonnene Turbulenzstatistiken können mit denen aus niedriger auflösenden Simulationen aus der Literatur verglichen werden, um Unterschiede im Verhalten stark turbulenter und nur mäßig turbulenter Strömungen zu erkennen. Ebenso sollte bei gleicher Auflösung isotrop abklingende Turbulenz simuliert werden und die Unterschiede der Spektren zu denen der Scherturbulenz analysiert werden. Mit dem Ausbau der Leistung des SUPRENUM-Rechners sollten dann langfristig Simulationen mit einer Auflösung von bis zu 500 Punkten pro Ortskoordinate und gleicher Physik wie oben durchgeführt werden.

3. Umkehrung der Rechenzeit / Speicherplatz - Kapazitäten: Vom post- zum parallel- processing

Zum Schluß sei noch auf einen Prozess aufmerksam gemacht, der sich von Generation zu Generation der Rechner verstärkt bemerkbar macht und alle Nutzer betrifft, die mit hochauflösenden Modellen den räumlichen und zeitlichen Ablauf physikalischer Phänomene studieren wollen. Es wird immer schwieriger (und teurer), große Datenmengen über den Rechenlauf selbst hinaus aufzuheben und abzuspeichern, um sie später auszuwerten. Auf der anderen Seite werden die Rechner immer schneller, was es erlauben sollte, Läufe mehrmals zu wiederholen und eine gezielte, enge Auswahl von Auswertungen sofort anzuschließen, so daß nur das gewünschte Endprodukt (Film, Dias, Schnitte, Statistiken) nach Integrationsende übrig bleibt. Diese Auswertungen können natürlich parallel zur eigentlichen Integration ablaufen, und das „post-processing" wandelte sich so in ein „parallel-processing".

Literatur

Batchelor, G. K. 1953 The theory of homogeneous turbulence. Cambridge Univ. Press.

Champagne, F. H. 1978: The fine-scale structure of the turbulent velocity field. J. Fluid Mech., **86**, 67-108.

Gerz, T., Schumann, U. & Elghobashi, S. 1989 Direct numerical simulation of stratified homogeneous turbulent shear flows. J. Fluid Mech., **200**, 563-594.

Hauf, T. & Höller, H. 1987: Entropy and potential temperature. J. Atmosph. Sciences, **44**, 2887-2901.

Reynolds, W. C. 1989 The potential and limitations of direct and large eddy simulations; Position Paper zur „Whither Turbulence" - Konferenz, Cornell Universität, 22. -24. März 1989.

Schmidt, H. & Schumann, U. 1989: Coherent structure of the convective boundary layer derived from large eddy simulations. J. Fluid Mech., **200**, 511-562.

Schumann, U., Hauf, T., Höller, H., Schmidt, H. & Volkert, H. 1987: A mesoscale model for the simulation of turbulence, clouds and flow over mountains: Formulation and validation examples. Beitr. Phys. Atmosph., **60**, 413-446.

Schumann, U. 1989: Large-eddy simulation of turbulent diffusion with chemical reactions in the convective boundary layer. Atmospheric Environment, **23**, 1713-1727.

Solchenbach, K. & Thole, C.-A. 1988: The SUPRENUM architecture and its application to computational fluid dynamics; Workshop on Use of Parallel Processors in Meteorology, ECMWF, Reading, 5.-9. Dez. 1988.

Lösung von Gleichungssystemen

Wolfgang Rönsch
Prof.Dr.Feilmeier, Junker & Co.,GmbH
Würmseestr. 35
8000 München 71
Tel. 089 - 7 59 84 - 0
Fax. 089 - 7 55 18 96

1. Einleitung

Das Lösen linearer Gleichungssysteme hat auf innovativen Rechnerarchitekturen schon immer reges Interesse gefunden. Zum einen zeigt dieses recht einfache Problem aus dem Bereich der linearen Algebra den Aufwand, der anfällt, die Lösung dieses Problems (bzw. das ihm zugrunde liegende Programm) optimal an die neue Rechnerarchitektur anzupassen, zum anderen dient es als ein allgemein anerkannter Maßstab für die auf Rechnerarchitekturen tatsächlich erzielbare Spitzenleistung (Benchmarks).

Die Lösung linearer Gleichungssysteme ist Teil des SUPRENUM-Lineare-Algebra-Pakets (SLAP), das gemeinsam von der KFA Jülich und der Prof.Dr.Feilmeier, Junker & Co., GmbH entwickelt wurde. Im folgenden wird sowohl auf den Umfang von SLAP näher eingegangen als auch anhand des Lösens linearer Gleichungssysteme mit vollbesetzter Matrix gezeigt, welche Parallelisierungsstrategie der SUPRENUM-Implementierung zugrunde liegt.

2. Historie der Basisroutinen der linearen Algebra

1979 wurden von Lawson et al. [6] einfache Programmkerne für Anwendungen im Bereich der linearen Algebra definiert, die hauptsächlich Skalar-Vektor-Operationen umfaßten. Diese Kerne, bezeichnet als BLAS (Basic Linear Algebra Subprograms, später als BLAS 1) waren gedacht als Bausteine für komplexere Programme der linearen Algebra und sollten dazu dienen, diese übersichtlich zu gestalten und ihre Portabilität zu vereinfachen. Außerdem war beabsichtigt, durch eine optimale Anpassung dieser Kerne an die entsprechende Rechnerarchitektur mittels maschinennaher Assemblerprogrammierung, die Effizienz der diese Kerne verwendenden FORTRAN-Programme zu steigern.

1979 wurde daraufhin das Lineare-Algebra-Paket LINPACK vorgestellt, das die wichtigsten Lineare-Algebra-Routinen umfaßte und dessen maschinenunabhängige FORTRAN-Implementierungen auf den BLAS 1-Modulen beruhte [3]. Aufgrund der leichten Portierbarkeit und der grundlegenden Bedeutung bei technisch-wissenschaftlichen Anwendungen wurde LINPACK, genauer die Routine zur Zerlegung einer Matrix in eine linke untere und eine rechte obere Dreiecksmatrix (sog. LU-Zerlegung), sehr häufig für Benchmarks herangezogen, so daß sich daraus ein Benchmark-Standard entwickelte.

Wegen der Ende der 70-er Jahre in Erscheinung getretenden Vektorrechner wurden die BLAS 1-Module um die sogenannten Extended-BLAS (heute BLAS 2) ergänzt [5]. Die BLAS 2-Module umfassen eine Reihe von Varianten von Matrix-Vektor-Operationen. Es hatte sich nämlich bei Benchmarks gezeigt, daß die BLAS 1-Module für Vektorrechner Bausteine mit einer zu geringen Operationsanzahl darstellten. Das Verhältnis von Vektorlade- zu Vektorrechen-Operationen war für Vektorrechner bei BLAS 1-Modulen äußerst ungünstig, was zur Folge hatte, daß Vektorrechner beim LINPACK-Benchmark enttäuschend schlecht abschnitten. Sie blieben z.B. bei der LU-Zerlegung weit unter der theoretischen Spitzenleistung. Die BLAS 2-Module, eingebaut in LINPACK, verbesserten die Benchmark-Ergebnisse für Vektorrechner erheblich; jedoch war, wie man sehr schnell erkannte, durch den Übergang von Matrix-Vektor- zu Matrix-Matrix-Operationen, den sogenannren BLAS 3-Modulen (1988), noch höhere Leistung erzielbar [4].

Gleichzeitig entschloß man sich in den USA ([1]) zu einer völligen Umgestaltung und Erneuerung des LINPACK-Paketes. Das neue Paket mit Namen LAPACK (Linear Algebra Package) soll hauptsächlich vier Forderungen erfüllen:

(1) LAPACK soll die neuesten Algorithmenentwicklungen auf dem Gebiet der linearen Algebra berücksichtigen.

(2) LAPACK soll die Routinen aus LINPACK und EISPACK (Paket zur Bestimmung von Eigenwerten und Eigenvektoren von Matrizen) zusammenfassen.

(3) LAPACK soll auf der Grundlage von BLAS-Modulen möglichst hoher Stufe implementiert werden.

(4) LAPACK soll eine überdurchschnittlich hohe Leistung auf Vektor- und Parallelrechnern vorweisen.

Da in den USA die Implementierung von LAPACK zunächst nur auf Parallelrechnern mit gemeinsamem Speicher (shared memory) in Angriff genommen wurde, waren diese Ergebnisse für SUPRENUM (distributed memory) nicht ohne weiteres übernehmbar. Daher wurden im Rahmen des SUPRENUM-Projektes die wichtigsten LINPACK- und EISPACK-Routinen zusammen mit den wichtigsten BLAS-Modulen parallelisiert und für den SUPRENUM-Rechner als SLAP (SUPRENUM-Linear-Algebra-Package) verfügbar gemacht.

3. Der Umfang von SLAP

Das SUPRENUM-Lineare-Algebra-Paket besteht aus

o den folgenden BLAS 2-Modulen:

(Die Namen der entsprechenden seriellen BLAS-Module erhält man durch Streichen des Buchstaben 'P' am Ende des Namens der parallelen BLAS-Module.)

SGEMVP: Y <-- ALPHA * A * X + BETA * Y

SGBMVP: Y <-- ALPHA * B * X + BETA * Y

SSYMVP: Y <-- ALPHA * S * X + BETA * Y

SGERP: A <-- ALPHA * X * Y' + A

SSYRP: S <-- ALPHA * X * X' + S

SSYR2P: S <-- ALPHA * X * Y' + ALPHA * Y * X' + S

STRMVP: X <-- T * X

STBMVP: X <-- TB * X

STRSVP: löst die Vektorgleichung T * X = Y

STBSVP: löst die Vektorgleichung TB * X = Y

wobei

A	eine Rechteckmatrix ist,
B	eine Rechteckmatrix mit Bandstruktur ist,
S	eine symmetrische Matrix ist,
M	M oder M' ist,
M'	die transponierte Matrix von M bezeichnet,
T	eine Dreiecksmatrix bezeichnet,
TB	eine Dreiecksmatrix mit Bandstruktur bezeichnet,
X, Y	Vektoren bezeichnen und
ALPHA, BETA	Skalare sind.

o den folgenden BLAS 3-Modulen

(Die Namen der entsprechenden seriellen BLAS-Module erhält man durch Streichen des Buchstaben 'P' am Ende des Namens der parallelen BLAS-Module.)

SGEMMP: C <-- ALPHA * A * B + BETA * C

SSYMMP: C <-- ALPHA * S * B + BETA * C
oder C <-- ALPHA * B * S + BETA * C

SSYRKP: S <-- ALPHA * A * A' + BETA * S
oder S <-- ALPHA * A' * A + BETA * S

SSYR2KP: S <-- ALPHA * (A * B' + B * A') + BETA * S
oder S <-- ALPHA * (A' * B + B' * A) + BETA * S

STRMMP: B <-- ALPHA * T * B
oder B <-- ALPHA * B * T

STRSMP: löst die Matrixgleichung T * X = ALPHA * B
oder X * T = ALPHA * B

wobei

A, B, C	rechteckige Matrizen sind,
S	eine symmetrische Matrix ist,
M	M oder M' bezeichnet,

M' die transponierte Matrix von M bezeichnet,
T eine Dreiecksmatrix bezeichnet,
ALPHA,
BETA Skalare sind.

- o Datentransfer-Modulen zwischen
  - Host und Knoten (in beiden Richtungen)
  - den Knoten untereinander
- o der parallelen LU-Zerlegung (einschließlich partieller Pivotisierung) mit Lösungsprozeß sowie Determinanten- und Inversenbestimmung
- o der parallelen Cholesky-Zerlegung mit Lösungsprozeß sowie Determinanten- und Inversenbestimmung
- o der parallelen QR-Zerlegung mit Lösungsprozeß
- o einem Löser für lineare Gleichungssysteme mit diagonaldominanter Bandmatrix
- o Routinen zur Reduktion vollbesetzter reeller symmetrischer Matrizen auf Tridiagonalgestalt,
- o Routinen zur Bestimmung der Eigenwerte und optional auch der Eigenvektoren symmetrischer Tridiagonalmatrizen sowie zur Rücktransformation der Eigenvektoren der Tridiagonalmatrizen auf die der Ausgangsmatrix.

4. Entwurfsprinzipien von SLAP

Dem Entwurf des SLAP-Paketes lagen die beiden Forderungen zugrunde [7]:

Zum einen sollen alle SLAP-Routinen hohe Effizienz auf dem SUPRENUM-Rechner zeigen und zum anderen sollen sie dem Benutzer auch hohen Komfort bieten.

Die hohe Effizienz wird erreicht durch

(E1) möglichst gleichmäßige Lastverteilung bei allen SLAP-Routinen.

(E2) möglichst wenig Kommunikation innnerhalb der SLAP-Routinen bzw. durch ein ausgewogenes Verhältnis von Arithmetik und Kommunikation.

(E3) grobe Granularität sowohl von Kommunikation als auch von Arithmetik, d.h. Kommunikation und Arithmetik wird in größtmöglichen Portionen ausgeführt.

(E4) zeitliche Überlappung von Kommunikation und Arithmetik. Diese Technik ist wegen der auf SUPRENUM möglichen asynchronen Kommunikation sinnvoll.

(E5) Verwendung möglichst langer Vektoren. Dadurch ist es möglich, die Leistung der Vektorprozessoren voll auszuschöpfen.

(E6) die Vermeidung von Wartezeiten auf Botschaftsankunft auf den Knoten. Dies wird z.B. dadurch erreicht, daß Botschaften im voraus verschickt werden (sog. look-ahead-Strategie), so daß sie unter Berücksichtigung der Übertragungszeit auf dem Zielknoten schon eingetroffen sind, wenn sie dort benötigt werden.

(E7) die Vermeidung von Zusammenführung von Daten auf dem Host-Prozeß. Alle SLAP-Routinen beginnen ihre Arbeit auf verteilten Daten und hinterlassen ihre berechneten Ergebnisse auch verteilt. Dadurch ist eine SLAP-Routine in der Lage, die Resultate ihres Vorgängers z.B. ohne Einschaltung des Host-Prozesses zu übernehmen und weiterzuverwenden (siehe auch BK6).

Der Benutzerkomfort der SLAP-Routinen drückt sich aus durch

(BK1) weitgehende Aufrufkompatibilität mit bekannter serieller Software, d.h. mit den LINPACK-, EISPACK- oder BLAS-Routinen. Zur Beschreibung der Parallelumgebung, in der die SLAP-Routinen aufgerufen werden können, war es notwendig, die Aufruflisten gegenüber den seriellen Routinen zu ergänzen.

(BK2) hohe Flexibilität bzgl. der Prozeßanzahl. Die Prozeßanzahl unterliegt bei allen SLAP-Routinen nur der Einschränkung, daß sie mindestens eins ist.

(BK3) die Funktionalität wie bei bekannter Software. Bei fast allen SLAP-Routinen ist im Falle einer Prozeßanzahl von eins sichergestellt, daß sie sich funktional wie die entsprechenden LINPACK-, EISPACK oder BLAS-Routinen verhalten.

(BK4) die Möglichkeit der Bearbeitung großer Probleme mit SLAP-Routinen. So ist es z.B. möglich, eine Matrix zu zerlegen, deren Größe die Kapazität eines lokalen Knotenspeichers übersteigt. Dies wird dadurch ermöglicht, daß es in der SLAP-Zerlegungsroutine nicht notwendig ist, die gesamte zu zerlegende Matrix zu irgendeinem Zeitpunkt der Berechnung in einem lokalen Knotenspeicher zu halten; vielmehr werden während der gesamten Bearbeitung nur jeweils Teile der Gesamtmatrix in den einzelnen Knotenspeichern benötigt.

(BK5) die weitgehende lokale Verwendung genormter Arithmetikbausteinen, d.h. der seriellen BLAS 1-, BLAS 2- und BLAS 3-Module.

(BK6) die sogenannte Schnittstellenkonsistenz der SLAP-Routinen untereinander, d.h. die Ausgabeparameter jeder SLAP-Routine sind so auf die parallelen Prozesse aufgeteilt, daß eine nachfolgend aufgerufene SLAP-Routine auf dieser Aufteilung wieder aufsetzen kann. Diese Eigenschaft steht mit (E7) in enger Beziehung.

(BK7) die Unsichtbarkeit der Kommunikation für den Benutzer. Die für die Berechnung der Resultate einer SLAP-Routine notwendige interne Kommunikation bleibt dem Benutzer verborgen, da sie automatisch in der Routine selbst abläuft.

(BK8) leichte Portierbarkeit innnerhalb der Klasse der Rechner mit verteiltem Speicher (distributed memory). Nur durch Änderungen der Kommunikationsbefehle ist es z.B. möglich, SLAP auf einen Rechner mit Hypercube-Architektur zu portieren.

Die wichtigsten gemeinsamen Merkmale aller SLAP-Routinen sind:

- Alle SLAP-Routinen sind lokal vektorisiert bzw. verwenden lokal Vektoralgorithmen.
- Alle SLAP-Routinen sind so entworfen, daß sie auf Knotenebene aufzurufen sind.
- Alle SLAP-Routinen benutzen eine sogenannte Standardaufteilung der Matrizen und Vektoren, um die in (BK6) erwähnte Schnittstellenkonsistenz zu realisieren.
- Alle SLAP-Routinen sind bzgl. der Kommunikation weitgehend optimiert und machen von der Möglichkeit der Überlappung von Kommunikation und Arithmetik Gebrauch.
- Alle SLAP-Routinen sind bzgl. Kommunikation und Arithmetik grob granularisiert.
- Alle SLAP-Routinen verwenden lokal BLAS-Module möglichst hoher Stufe.

Im folgenden Abschnitt soll am Beispiel der parallelen LU-Zerlegung gezeigt werden, welche Vor- und Nachteile verschiedene Datenaufteilungen bei der Parallelisierung des Zerlegungsalgorithmus haben.

5. Lösung linearer Gleichungssysteme mittels LU-Zerlegung

5.1 Das Problem

Um das lineare Gleichungssystem

$$A * x = b$$

mit vollbesetzter quadratischer n x n Matrix A zu lösen, geht man in zwei Schritten vor:

(1) Zerlege A in

$$A = L * U,$$

wobei U eine obere und L eine untere Dreiecksmatrix ist. L ist außerdem mit Einsen in der Hauptdiagonale besetzt.

(2) Löse

$L * y = b$ durch Vorwärtseinsetzen

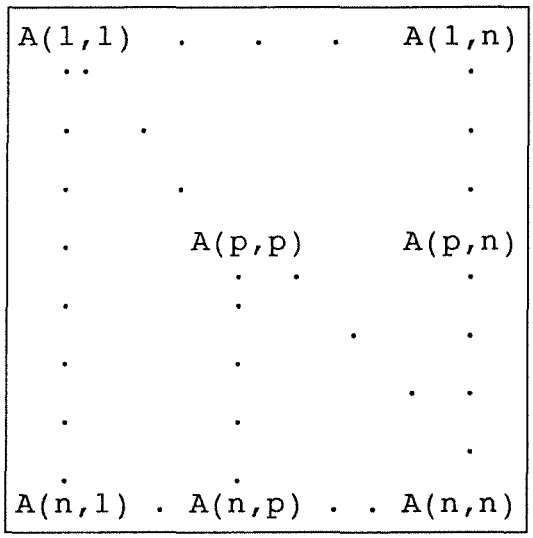
und $U * x = y$ durch Rückwärtseinsetzen.

Die Zerlegung erfolgt durch Berechnung einer Folge von Matrizen beginnend mit A:

$$A = A^{(1)} \dashrightarrow A^{(2)} \dashrightarrow A^{(3)} \dots\dots\dashrightarrow A^{(n)}$$

wobei L als linke untere Dreicksmatrix und U als rechte obere Dreicksmatrix in $A^{(n)}$enthalten ist.

Der Prozeß der Umwandlung von A erfolgt in n Schritten. Er beginnt in der linken oberen Ecke der Matrix A und endet rechts unten. Um die schrittweise Zerlegung der Matrix A mittels der BLAS 3-Module realisieren zu können, stellen wir die Matrix A in Blockgestalt dar (siehe Figur 1), wobei die Diagonalblöcke A(i,i) (i=1,...n) quadratisch sind.



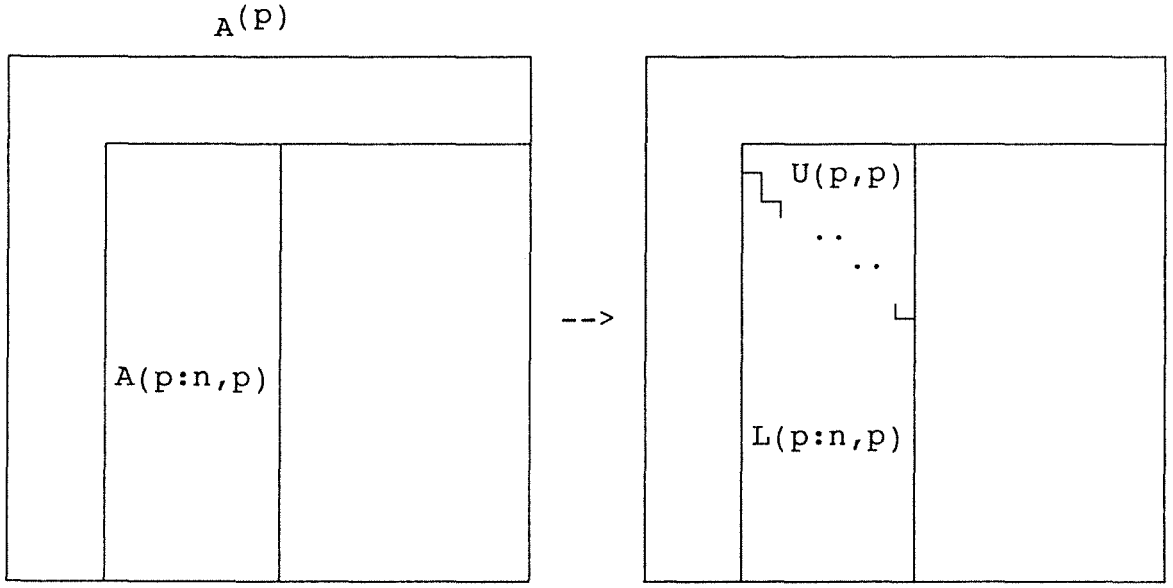
Figur 1: Blockdarstellung der Matrix A

Im weiteren Verlauf dieses Artikels konzentrieren wir uns auf die Möglichkeiten, den Zerlegungsprozeß für SUPRENUM optimal zu parallelisieren. Der Lösungsprozeß wird im folgenden nicht mehr betrachtet.

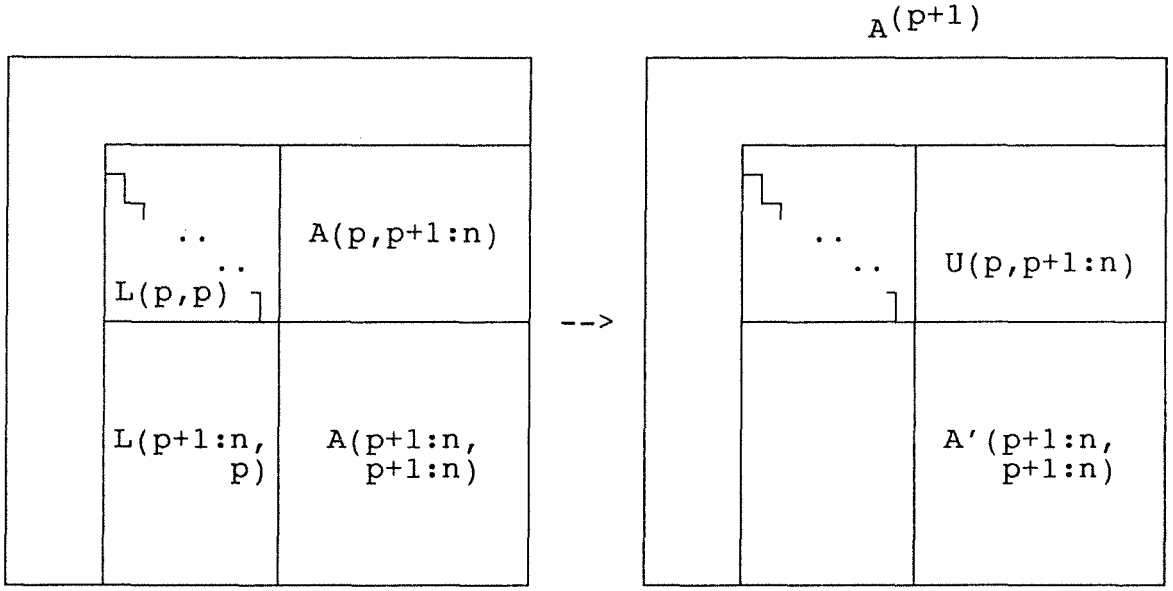
5.2 Die serielle LU-Zerlegung mit BLAS 3-Modulen

Der Übergang von $A^{(p)}$ zu $A^{(p+1)}$ bei der LU-Zerlegung einer

Matrix erfolgt in drei Schritten, die in Figur 2 dargestellt sind.



(1) $A(p:n,p) = L(p:n,p) * U(p,p)$



(2) $U(p,p+1:n) = L^{-1}(p,p) * A(p,p+1:n)$

(3) $A'(p+1:n,p+1:n) = A(p+1:n,p+1:n) - L(p+1:n,p) * U(p,p+1:n)$

Figur 2: LU-Zerlegung einer Matrix mittels BLAS 3-Modulen

Serielle LU-Zerlegung:

(1) LU-Zerlegung eines Spaltenblockes:
Faktorisiere A(p:n,n) mittels einer LU-Zerlegung, so daß gilt:

$$A(p:n,p) = L(p:n,p) * U(p,p)$$

(2) Aktualisieren des Zeilenblockes mittels BLAS 3-Modul STRSM :

$$U(p,p+1:n) = L^{-1}(p,p) * A(p,p+1:n)$$

(3) Aktualisieren der Restmatrix mittels BLAS 3-Modul SGEMM:

$$A'(p+1:n,p+1:n) = A(p+1:n,p+1:n) - L(p+1:n,p) * U(p,p+1:n)$$

Für die LU-Zerlegung des Spaltenblocks in Schritt (1) kann der in Figur 2 beschriebene Algorithmus wiederum verwendet werden, wenn die Schritte (1) bis (3) entsprechend neu interpretiert werden:

Das Wort 'Block' ist durch 'Vektor' zu ersetzen, so daß aus Spaltenblöcken Spaltenvektoren und aus Diagonalblöcken Skalare werden. Schritt (1) wird zu einer Skalar-Vektor-Multiplikation, ebenso Schritt (2). Schritt (3) ist eine Rang-1-Modifikation einer Rechteckmatrix, also durch das BLAS 2-Modul SGER realisierbar.

Es sei hier nur kurz darauf hingewiesen, daß sich in die LU-Zerlegung eines Spaltenblockes eine Spaltenpivotisierung sehr leicht einbauen läßt. Wenn die hieraus resultierenden Zeilenvertauschungen explizit in der gesamten Matrx A ausgeführt werden, können die Schritte (2) und (3) ohne größere Modifikationen beibehalten werden.

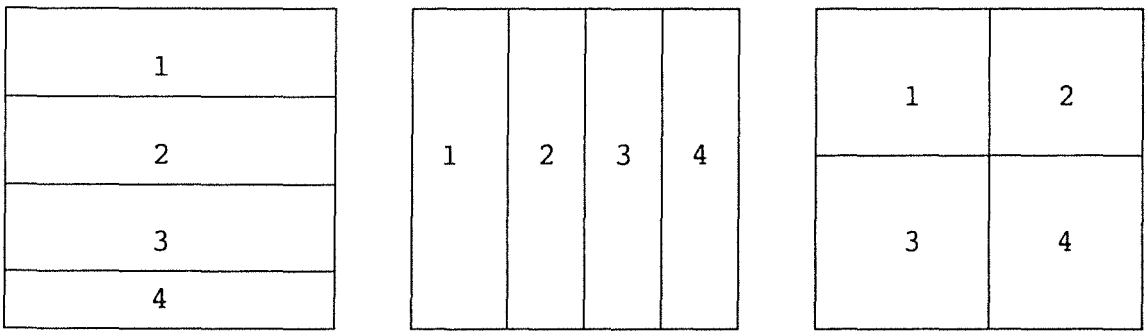
Die entscheidende Rolle für eine gute oder schlechte Parallelisierung der soeben vorgestellten seriellen LU-Zerlegung für einen Rechner mit verteilten Speicher spielt die a-priori gewählte Datenaufteilung der Matrix. Man kann leicht zeigen, daß die Effizienz der parallelen Implementierung sehr stark von der Wahl der Aufteilung abhängt. Sie bestimmt die Lastverteilung der einzelnen Prozesse und den während der Ausführung des Algorithmus notwendigen Kommunikationsaufwand.

Für Rechner mit gemeinsamem Speicher gibt es das Problem der a-priori zu wählenden Datenaufteilung nicht. Hier spielen Zeitverluste durch Prozeßsynchronisation bei der Lösung von auftretenden Zugriffskonflikten auf den gemeinsamen Speicher die entscheidende Rolle.

5.3 Matrixaufteilungen

In diesem Abschnitt stellen wir verschiedene Matrixaufteilungen vor und diskutieren in 5.4 deren Vor- und Nachteile in Bezug auf die in 5.2 beschriebene LU-Zerlegung.

Die naheliegendste Aufteilung einer Matrix ist die Blockaufteilung. Sie kann (a) zeilenweise, (b) spaltenweise oder (c) schachbrettartig (siehe Figur 3) erfolgen.



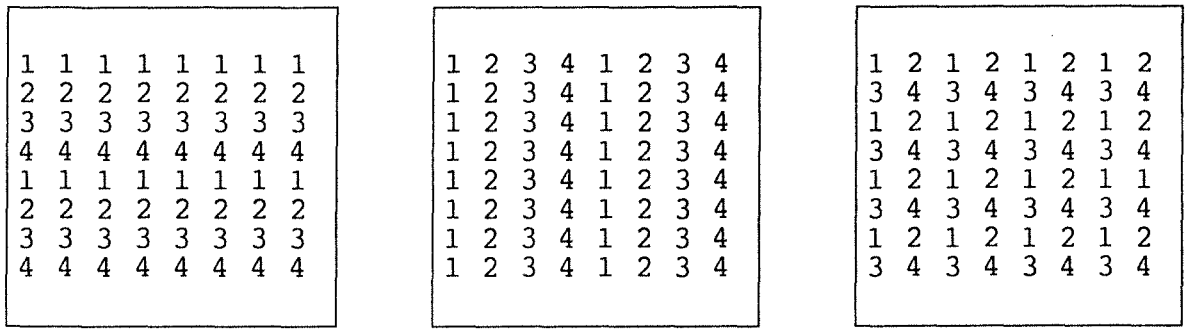
(a) (b) (c)

Figur 3: Blockaufteilungen einer Matrix auf 4 Prozesse: zeilenweise (a), spaltenweise (b) und schachbrettartig (c)

Der Nachteil aller drei Varianten der Blockaufteilung in Bezug auf die LU-Zerlegung besteht darin, daß sie eine ungleichmäßige Lastverteilung der beteiligten Prozesse induzieren:

Setzen wir dazu die Anzahl der Prozesse gleich n (vgl. Figur 1), so daß in n Schritten aus A die zerlegte Matrix gemäß der in 5.2. beschriebenen Zerlegung entsteht, so ist ersichtlich, daß nach p Schritten die Prozesse 1,2,...,p-1 bei zeilen- oder spaltenweiser Blockaufteilung für den Rest der Berechnung untätig werden. Auch bei schachbrettartiger Blockaufteilung tritt der Fall ein, daß nicht alle Prozesse bis zum Ende der Berechnung beschäftigt bleiben. Um eine gute Lastverteilung der Prozesse während der gesamten Berechnung zu gewährleisten, muß die Aufteilung daher anders gestaltet werden. Die Lösung liegt in einer zyklischen Aufteilung der Zeilen-, der Spalten- oder der Zeilen- und Spalten, wie sie Figur 4 zeigt. (Eine zyklische Aufteilung sowohl der Zeilen- als auch der Spalten kann man auch als eine zyklische Schachbrettaufteilung auffassen.)

Statt nur einzelne Spalten- oder Zeilen zyklisch auf die Prozesse zu verteilen, ist es auch möglich, Spalten- oder Zeilenblöcke zyklisch aufzuteilen. (Der j-te Zeilenblock in der Blockdarstellung der Matrix A ist z.B. A(j,1:n), der j-te Spaltenblock dagegen A(1:n,j), siehe Figur 1.) Die Anzahl der Zeilen bzw. Spalten in den betreffenden Blöcken wird als Blockbreite bezeichnet. Derartige Aufteilungen bezeichnen wir im folgenden als block-zyklische Aufteilungen. Diese Aufteilungen sind Verallgemeinerungen der zyklischen Aufteilungen. Nach dem bisher Gesagten ist auch klar, was unter einer block-zyklischen Schachbrettaufteilung zu verstehen ist.



(a) (b) (c)

Figur 4: Zyklische Aufteilung der Zeilen (a), der Spalten (b) und der Zeilen- und Spalten (c) einer Matrix auf 4 Prozesse

Spezielle block-zyklischen Aufteilungen sind diejenigen mit konstanter Blockbreite. Unter einer variablen block-zyklischen Aufteilung versteht man dagegen eine block-zyklische Aufteilung mit variablen Blockbreiten. Diese Aufteilungen sind in dieser Aufzählung die allgemeinsten, da sie alle anderen als Spezialfälle enthalten. Werden bei variabler block-zyklischer Aufteilung außerdem die Blockbreiten vom Algorithmus selbst während der Ausführung gesteuert, so spricht man von einer adaptiven block-zyklischen Aufteilung der Matrix.

5.4 Vor- und Nachteile der verschiedenen Matrixaufteilungen

Blockaufteilungen, die nicht zyklisch-orientiert sind, sind für die LU-Zerlegung ungeeignet, da sie - wie schon oben erwähnt - keine akzeptablen Lastverteilungen der Prozesse ermöglichen. Zyklische Aufteilungen gestatten dagegen nicht den Einsatz der BLAS 3-Module, wenn jeweils nur einzelne Spalten oder Zeilen zyklisch aufgeteilt sind. Es ist daher nur sinnvoll, block-zyklische Aufteilungen zu betrachten.

Block-zyklische Aufteilungen, bei denen Spaltenblöcke zyklisch aufgeteilt sind, bieten eine Reihe von Vorteilen, die block-zyklische Aufteilungen mit Zeilenblöcken oder block-zyklische Schachbrettaufteilungen nicht haben:

- Die klassische partielle Pivotsuche (Pivotsuche entlang einer Spalte mit anschließendem Vertauschen von Zeilen) wird wesentlich vereinfacht, da sie innerhalb eines Prozesses und somit ohne zusätzliche Kommunikation ausgeführt werden kann.

- Das Vertauschen der Pivotzeilen erfordert keine Kommunikation; jeder Prozess vertauscht denjenigen Teil der Pivotzeile, der sich in seinem Privatspeicher befindet.

- Da in FORTRAN traditionsgemäß zwei-dimensionale Felder spaltenweise abgespeichert werden, sollten aus Effizienzgründen Spaltenvektoren für die Vektorverarbeitung verwendet werden. Bei der Ausführung des BLAS 3-Moduls SGEMM in Schritt (3) kann daher mit den jeweils größtmöglichen Vektorlängen gearbeitet werden.

- Die Prozeßstruktur kann als Ring gewählt werden.

- Es gibt keine Einschränkung in der Wahl der Prozeßanzahl.

Die Nachteile der block-zyklischen Schachbrettaufteilung gegenüber der block-zyklischen Aufteilung mit Spaltenblöcken lassen sich folgendermaßen zusammenfassen :

- Es sind für SGEMM keine maximalen Vektorlängen möglich.

- Die Pivotsuche entlang der gesamtem Matrixspalte ist nur mit zusätzlichem Kommunikationsaufwand möglich.

- Das Vertauschen der Pivotzeilen erfordert im allgemeinen Kommunikation.

- Gegenüber einer block-zyklischen Aufteilung mit Spaltenblöcken wird der Kommunikationsaufwand erhöht.

- Eine (echte) block-zyklische Schachbrettaufteilung ist nicht für jede Prozeßanzahl möglich, z.B. wenn die Prozeßanzahl eine Primzahl ist.

- Gegenüber einer block-zyklischen Aufteilung mit Spaltenblöcken resultiert insgesamt eine kompliziertere Kommunikationsstruktur und ein komplizierterer Algorithmusablauf.

Ein Teil dieser aufgeführten Nachteile trifft auch auf die block-zyklische Aufteilung mit Zeilenblöcken zu, so daß die vorteilhafteste Lösung bei der block-zyklischen Aufteilung mit Spaltenblöcken liegt.

5.5 Die parallele LU-Zerlegung in SLAP

Um sich den Ablauf der parallelen LU-Zerlegung auf der Grundlage des in Figur 2 dargestellten seriellen Algorithmus zu veranschaulichen, stelle man sich vor, daß die Matrix in Spaltenblöcke block-zyklisch auf np Prozesse aufgeteilt ist.

Die LU-Zerlegung eines Spaltenblockes wie sie in Schritt (1) der seriellen LU-Zerlegung beschrieben worden ist, erfolgt reihum jeweils auf dem Prozeß k, der den zu zerlegenden Spaltenblock (sog. aktueller Spaltenblock) in seinem Speicher hält. Sodann schickt dieser Prozeß sowohl L(p,p) als auch L(p+1:n,p) an alle anderen Prozesse. Dies geschieht entweder mit Hilfe eines Broadcast-Befehls oder durch (np-1)-maliges Senden an die restlichen Prozesse in der Reihenfolge k+1,k+2,...,np,1,2,..,k-1. (Diese Reihenfolge ist z.B. des-

halb empfehlenswert, damit Prozeß k+1, der als nächster eine (lokale) Zerlegung des aktuellen Spaltenblockes auszuführen hat, möglichst unverzüglich die dafür notwendigen Daten erhält.) Anschließend führt Prozeß k die Schritte (2) und (3) auf denjenigen Teilen von A(p,p+1:n) und A(p+1:n,p+1:n) aus, die sich in seinem Speicher befinden.

Die übrigen Prozesse empfangen L(p,p) und L(p+1:n,p) und bestimmen damit U(p,p+1:n) sowie aktualisieren A(p+1:n,p+1:n) für diejenigen Teile, die in ihrem Speicher liegen. Sodann obliegt Prozeß k+1 die LU-Zerlegung des aktuellen Spaltenblockes; er bestimmt L(p+1,p+1) und L(p+2:n,p+1) und verschickt diese Information an die übrigen Prozesse, die mit diesen Daten ihre Arbeit entsprechend fortsetzen können.

Aus der Beschreibung des Ablaufs ist ersichtlich, daß der Engpaß darin besteht, daß die übrigen Prozesse auf das Eintreffen von L(p+1,p+1) und L(p+2:n,p+1), abgeschickt von Prozeß k+1, warten. Dieser Engpaß kann erheblich entschärft werden, indem Prozeß k+1 die zu verschickenden Daten eher als eben beschrieben aussendet. Prozeß k+1 muß sich nach Empfang von L(p,p) und L(p+1:n,p) nur wie folgt verhalten:

Er bestimmt nur denjenigen Teil von U(p,p+1:n) der seinem aktuellen Spaltenblock angehört, das ist U(p,p+1), und aktualisiert nur den zum aktuellen Spaltenblock gehörenden Anteil von A(p+1:n,p+1:n), d.h. A(p+1:n,p+1), zieht die nun möglich gewordene LU-Zerlegung des aktuellen Spaltenblockes vor, verschickt sodann die erhaltenen Ergebnisse L(p+1,p+1) und L(p+2:n,p+1) und widmet sich dann wieder der Aktualisierung des Restes von U(p,p+1:n) und A(p+1:n,p+1:n), der in seinem lokalen Speicher liegt. Anschließend hat Prozeß k+1 mit dem selbst berechneten L(p+1,p+1) und L(p+2:n,p+1) seine Teile von U(p+1,p+2:n) und von A'(p+2:n,p+2:n) zu aktualisieren.

Das Vorziehen der LU-Zerlegung des aktuellen Spaltenblockes und das Unterbrechen der Aktualisierung der Restmatrix wird als sogenannte Compute-and-Send-ahead-Strategie bezeichnet, d.h. das Senden der von den anderen Prozessen benötigten Daten wird vorgezogen und damit die Wartezeit der anderen Prozesse auf das Eintreffen von notwendigen Informationen möglichst klein gehalten.

Der nachfolgende Pseudocode beschreibt den auf einem Knoten ablaufenden Prozeß, wie er in SLAP für die parallele LU-Zerlegung realisiert worden ist. Prozeß 1 spielt dabei eine Sonderrolle, da er den Zerlegungsprozeß zu starten hat und deshalb zu Beginn keine Daten empfangen muß.

Der parallele LU-Zerlegungsalgorithmus

```
p = 1
n = number_of_blocks
np = number_of_processes
ip = index_of_process

IF ip = 1 THEN
    factorize   A(p:n,p)
    BROADCAST   L(p,p), L(p+1:n,p)
    compute     U(p,p+1:n:np)
    update      A(p+1:n,p+1:n:np)
    p = 2
ENDIF

WHILE p < n DO
    RECEIVE  L(p,p), L(p+1:n,p)
                FROM PROCESS MOD(p-1,np)+1

    IF MOD(p,np)+1 = ip THEN
/ compute-.....          /
        compute    U(p,p+1)                  / step no. p /
        update     A(p+1:n,p+1)              / step no. p /
        factorize  A(p+1:n,p+1)
/ .... and-send-ahead   /
        BROADCAST  L(p+1,p+1), L(p+2:n,p)
        compute    U(p,p+1+np:n:np)          / step no. p /
        update     A(p+1:n,p+1+np:n:np)      / step no. p /
        compute    U(p+1,p+1+np:n:np)        / step no. p+1 /
        update     A(p+2:n,p+1+np:n:np)      / step no. p+1 /
        p = p + 2
    ELSE
        compute    U(p,p+1:n:np)
        update     A(p+1:n,p+1:n:np)
        p = p + 1
    ENDIF

ENDWHILE
```

6. Abschließende Bemerkungen

Das Parallelisierungsprinzip der LU-Zerlegung, wie es in 5.5 im Detail vorgestellt worden ist, ist mit geringen Modifikationen auch auf die Cholesky-Zerlegung (Faktorisierung einer positiv-definiten symmetrischen Matrix) und mit größeren Anpassungen (siehe [2]) auf die QR-Zerlegung übertragbar. In SLAP wurden beide Zerlegungsarten auf der Grundlage block-zyklischer Aufteilungen mit Spaltenblöcken realisiert.

Außerdem wurden die Zerlegungsroutinen von SLAP für den Einsatz automatischer Blockbreitensteuerung vorbereitet, so daß eine adaptive block-zyklische Matrixaufteilung verwendet werden kann. Der Vorteil dieser Technik besteht darin, daß der Algorithmus in die Lage versetzt wird, in Abhängigkeit von Maschinenparametern wie z.B. Kommunikationsgeschwindigkeit und Rechenleistung der Skalar- und Vektorprozessoren

seine Ausführungszeit selbständig zu minimieren, indem er die Blockbreiten jeweils so wählt, daß die Wartezeiten der übrigen Prozesse auf eintreffende Botschaften möglichst gering ist.

7. Literatur

[1] C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, D. Sorensen
LAPACK Working Note #5: Provisional Contents
ANL-88-38, Argonne National Laboratory, September 1988

[2] C. Bischof, C. Van Loan
The WY Representation for Products of Householder Matrices, SIAM Journal on Scientific and Statistical Computing 8,1 (1987), s2-s13

[3] J.J. Dongarra, J.R. Bunch, C.B. Moler, G.W. Stewart
LINPACK User's Guide
SIAM Philadelphia (1979)

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

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

[6] Lawson, R. Hanson, D. Kincaid, F. Krogh
Basic Linear Algebra Subprograms for FORTRAN Usage
ACM Transactions on Math. Software 5 (1979), 308-323

[7] W. Rönsch, H. Strauß
Design Aspects of a Linear Algebra Package for the SUPRENUM Multiprocessor System, in:
J.J. Dongarra, P. Gaffney, I. Duff, S. McKee (Eds.)
Vector and Parallel Computing
Issues in Applied Research and Development
Proceedings of the second International Conference on Vector and Parallel Computing, June 6-11 1988, Tromsö, Norwegen
Ellis Horwood Ltd. Publ. Comp. 1989

Proceedings of the American Nuclear Society Topical Meeting on Advances in Nuclear Engineering Computation and Radiation Shielding, April 9-13, 1989, Santa Fe, New Mexico, USA

SOFTWARE DEVELOPMENT FOR REACTOR SIMULATION ON MULTIPROCESSOR SYSTEMS

R. Müller , R. Böer and H. Finnemann

Siemens AG , KWU Group , U6 514

Hammerbacherstr. 12 + 14

D - 8520 Erlangen

Federal Republic of Germany

SOFTWARE DEVELOPMENT FOR REACTOR SIMULATION ON MULTIPROCESSOR SYSTEMS

R. Müller , R. Böer and H. Finnemann

Siemens AG , KWU Group , U6 514
Hammerbacherstr. 12 + 14
D - 8520 Erlangen
Federal Republic of Germany

ABSTRACT

The present paper gives an account of experiences gained and results obtained in applying two different types of multiprocessor systems to the numerical calculation of nuclear reactors. In a first stage, parallelization of the nodal expansion method to solve the neutron diffusion equation is combined with several multigrid methods. These investigations are performed on the DIRMU distributed shared memory multiprocessor kit consisting of 25 identical processor memory modules. Results are obtained for various domain splitting strategies. With near-perfect load balancing, efficiencies of more than 90 percent are reached for a 24 processors system. The thermal-hydraulic part of the reactor calculation is based on the six equation, two fluid program THERMIT-2. The strategy of parallelization by domain splitting and its implications on organization and control of the iteration procedures are described. Equally high efficiencies as in the neutronics part can be predicted for closed channel models. The ultimate goal is an integrated system formed by coupled modules for neutron kinetics and thermal hydraulics in the reactor core and for coolant loop and plant control simulation. This system is designed to run on the high performance message passing local memory multiprocessor system SUPRENUM which will eventually feature 256 processors aggregated in 16 clusters.

INTRODUCTION

In nuclear reactor safety and performance evaluation, the need arises for refined spatial resolution and improved physical modeling, combined with reduced computing times directed towards real-time simulation. This renders analysis of nuclear reactor phenomena one of the most demanding numerical tasks requiring extremely powerful computer systems. In some scientific applications, the first generations of supercomputers based on the pipelining principle (vector computers and multi-pipeline vector computers with a big shared central memory) are already approaching their limits. This led to the development of multiprocessor architectures with distributed (local) memory units. The performance of such parallel computer systems can in principle be increased without any limit by simply increasing the number of processors (nodes). This paper gives an account of experiences gained and results obtained in applying two different types of multiprocessor systems to the numerical calculation of nuclear reactors.

The ultimate goal is a coupled neutron kinetics - thermal hydraulics program system for the calculation of steady-state and transient conditions in the core and for the modeling of the coolant loops of light water reactors. Making efficient use of the parallel computing architecture requires the implementation of novel, purpose-built software concepts which can be characterized by parallelization and multigrid algorithms. It is shown how algorithmic and multiprocessor speedup are combined by applying the multilevel principle to the solution of the neutron diffusion equation. Similar effects are anticipated for the time-consuming sections of the fluid dynamics computations.

The first part of the job is to develop and test appropriate multigrid algorithms to solve the few-group neutron diffusion equations. Multigrid algorithms are composed of relaxation and grid transfer operators which are all completely parallel and well suited for multiprocessor computation. Their efficiency results from the fact that relaxation quickly eliminates high frequency components of the error. This approach is applied to the nodal expansion method (NEM), a consistent higher-order nodal method. Parallelization of NEM by several multigrid methods is implemented. A multiplicative two-grid method (coarse-mesh rebalancing) is extended to a multiplicative multilevel technique. This approach is then compared with a conventional multigrid method with additive corrections. These investigations are performed on the DIRMU distributed shared memory multiprocessor system.

The thermal-hydraulic part of the reactor calculation is based on the six equation, two fluid program THERMIT-2. It allows 3-dimensional steady-state and transient full-core as well as subchannel analyses to be performed. The natural way of parallelizing THERMIT-2 is by domain splitting, i.e. by distributing the adjacent coolant channels or subchannels to the processors. The capability to calculate crossflow effects is essential to achieve the high degree of spatial resolution and accuracy aimed at in the coupled neutronics - thermal hydraulics part. The modules for reactor core calculation must then be linked with the parallelized version of a coolant loop and plant simulation code. The integrated program system will be implemented on the high performance multiprocessor system SUPRENUM.

MULTIPROCESSOR SYSTEMS EMPLOYED

Parallel supercomputer architectures based on the MIMD (Multiple Instruction stream - Multiple Data stream) principle can be broadly grouped into two categories, according to the way in which memory units are organized. The first category includes global memory machines containing only one large shared memory for all processors. With increasing number of processors and memory access operations, a bottleneck might develop due to access conflicts. The second category comprises local memory machines providing each processor with an individual memory unit. Data exchange occurs solely by message passing via bus links. With growing number of messages to be sent, the corresponding communication (start-up plus transmission) times could hamper overall efficiency. Of the multiprocessor systems considered below, the commercial SUPRENUM design belongs to the second category, whereas the research testbed DIRMU is situated somewhere in between.

DIRMU

The DIRMU distributed shared memory multiprocessor kit / 1 / of the University of Erlangen-Nuremberg consists of up to 25 identical processor memory modules. Memory-coupling with direct memory access between up to 8 neighbouring processors can be realized which allows the programmer to configure a broad spectrum

of multiprocessor structures (ring, array, cube, etc.) . Hence a calculational problem is mapped by setting up a suitable hardware structure.

SUPRENUM

The high performance multiprocessor system SUPRENUM / 2,3 / is modularly structured and will eventually feature 256 processors (nodes) aggregated in 16 clusters. Each node consists of a central unit (Motorola MC 68020), a vector unit (Weitek WT 2264/2265), an 8 MByte private memory and a communication unit linking it to the clusterbus. Apart from the 16 computing nodes, the main components of the cluster are a high performance disk and the connection to the inter-cluster taxibus. For input/output, programming and system control, a front-end computer system is employed. As the peak performance of each of the 256 nodes is expected to be 8 Mflops (16 Mflops with chaining), the theoretical overall peak performance of the SUPRENUM system will be more than 2 (4 with chaining) Gflops.

The SUPRENUM software concept is characterized by message passing and parallel processes. Because the hardware structure is given, configurations to fit a certain problem are user-defined based on system software. This concept leads to some specific features of a corresponding parallel program. The task processes to be performed in parallel are generated by the main (host) program, which also organizes data distribution before and during the calculation run. As no common (shared) memory is available, data needed by any one process must be transferred from the processes where they have been calculated. An example for this message passing via busses is the updating of boundary data belonging to other processes' domains during iterations, which requires process-process communication. Iteration control lies with the host process, which therefore has to collect the individual task processes' results, to form some global parameters (e.g. check of convergence) and to broadcast information back to the task processes. It is evident that both the quality of parallel programming (especially with regard to communication required) and the appropriateness of user-defined system configuration are of crucial importance to the overall efficiency of such parallel applications.

SOFTWARE COMPONENTS AND MODEL EQUATIONS

NEUTRON KINETICS

Consistent nodal methods enjoy widespread computational use because they are easy to program and very accurate even on a coarse mesh. The nodal expansion method / 4 / is a consistent higher-order nodal method which can be derived by converting the P_1-form of the neutron diffusion equation into a set of nodal equations and equivalent one-dimensional diffusion equations for the average flux in each space direction. The nodal equations are obtained by formally integrating the diffusion equation over each volume element (box, node) into which the nuclear assembly is partitioned. In matrix notation, the system consists of the nodal balance equations

$$\frac{1}{v}\,\frac{d\Phi}{dt} + M\Phi = \frac{1}{\lambda}(1-\beta)\,\chi_p\,F^T\Phi + B\,J^{in} + \sum_{i=1}^{I}\lambda_i\,\chi_d\,C_i + G \quad , \qquad (1)$$

the precursor equations

$$\frac{dC_i}{dt} = -\lambda_i C_i + \beta_i \frac{1}{\lambda} F^T \Phi \quad , \tag{2}$$

the outgoing current equations

$$J^{out} = D\Phi + E J^{in} + H \quad , \tag{3}$$

$$J^{in} = A J^{out} \quad , \tag{4}$$

and a set of weighted residual equations for determining higher order polynomial coefficients, with

Φ	flux density vector,	J	partial current vector,
C	precursor density vector,	F	fission vector,
λ_i	decay constants,	M	absorption and scattering matrix,
λ	eigenvalue,	χ_p, χ_d	prompt, delayed emission spectrum vector
A	generalized Albedo matrix which describes coupling to neighbours,		
and	B, D, E, G, H	matrices whose elements are functions of the diffusion coefficient and higher order polynomial coefficients.	

The system of equations may be solved with multigrid methods for neutron fluxes Φ and currents J subject to appropriate boundary and continuity conditions. With a proper ordering for the unknowns, the coefficient matrices of the flux and current equations have a regular sparse structure, and highly vectorizable solution algorithms can be used. Since parallelization decreases vector lengths by a factor given by the number of processors employed, the optimum overall performance is a complex function of computer architecture and solution algorithms used. This is especially true for multigrid methods, where calculations on coarser grids leave an increasing number of processors idle. Care has to be taken lest the resulting efficiency losses outweigh the acceleration by higher-order multigrid algorithms.

THERMAL HYDRAULICS

The thermal-hydraulic module 3D-THERM is a parallelized version of the program THERMIT-2 / 5 / designed to be run on the local memory multiprocessor system SUPRENUM. The additions necessary for parallelization will be described in the next chapter. They include routines for distributing the computational workload to the available processors, for organizing and controlling the parallel computation and for communication between processors. However, all the original THERMIT-2 features have been maintained.

The conservation equations of the fluid are written for each individual phase in time-dependent, three-dimensional form. Rectangular coordinates are assumed for the reactor core. Both pressurized and boiling water reactors can be treated, the applications ranging from full-core to subchannel analyses. Fuel pin temperatures are calculated from the radial heat conduction equation, and the coupling to the coolant determining heat transfer dynamics is realized by appropriate models.

The two-phase fluid conservation equations for mass, momentum and energy are detailed in / 6 / and need not be reproduced here. The resulting system of partial differential equations is discretized in time and space and solved in first-order finite difference form. The calculation procedure for the new-time

fluid quantities consists of Newton iterations which are further simplified to the solution of a pressure-only boundary value problem by a block Gauss-Seidel iteration. Parallelization of this method requires careful study and consideration of its internal details.

The fuel rod model is based on the radial heat conduction equation

$$\rho c_p \frac{\partial T}{\partial t} + \frac{1}{r}\frac{\partial}{\partial r}\left(r k \frac{\partial T}{\partial r}\right) = \dot{q}''' \quad , \tag{5}$$

i.e. axial heat conduction is ignored. One representative fuel rod is assumed for each coolant channel. After dividing the fuel rod into radial mesh cells and integrating between adjacent cell centers, an implicit time differencing yields the difference form of Eq. (5) .

PARALLELIZATION TECHNIQUES AND RESULTS

MULTIGRID ALGORITHMS FOR DIFFUSION EQUATIONS

The neutron diffusion equation discussed in the previous chapter was parallelized and implemented on the DIRMU multiprocessor system. Experiments have shown that a good approach for the parallelization of the solution of large partial differential equations is domain splitting onto the processors. For the discretized equations the fuel rods and the guide tubes are the grid points of the domain.

One possibility for domain splitting is dividing the reactor core into stripes. The demand for load balancing makes it necessary to define stripes with almost the same number of points. Starlike grid operators (Gauss-Seidel, Jacobi, etc.) then require communication with only two neighbouring stripes. Thus the processors are configured as a ring. Because of the fast memory coupling, access to neighbour data causes no loss of efficiency.

To accelerate the solution of the neutron diffusion equation several multigrid methods were implemented / 7,8 / . A multiplicative two-grid method (coarse-mesh rebalancing) was extended to a multiplicative multilevel technique. This approach was then compared with a conventional multigrid method with additive corrections. A coarse grid can also be defined in energy space. To reach better convergence rates the multilevel approaches sometimes have to be extended to higher-order multigrid methods without condensation of the energy groups on the coarse grids. The points of coarse grids are distributed to the processors in the same manner as those of the fine grid. To smooth the errors on the different grid levels synchronous red-black Gauss-Seidel relaxation is carried out.

For the implemented synchronized multigrid method, all processors have to wait until the processor with the highest number of boxes in its stripe finished calculation. Before the next iteration step is initiated some global information has to be formed by all processors. Waiting times cause a loss of efficiency for a distribution of the solution domain into stripes. It is evident that an equal number of boxes for all processors would guarantee a minimum of communication and synchronization overhead. Indeed, direct projection of the fuel assemblies or even smaller subunits to the processors results in a perfect load balancing on the multiprocessor system. Calculations with the 25 available DIRMU processors confirmed the expectations of very good efficiency

for such configurations. The performance curves given in Fig. 1 illustrate the speedup reached for domain splitting into stripes and into single fuel assemblies, with different numbers of processors employed. The corresponding efficiency (defined as the ratio of reached speedup to the number of processors applied) for the stripes approach ranges from 80 to 93 % . In contrast, for direct projection of the fuel assemblies the efficiency losses caused by communication time, by synchronization time and by the sequential part of the algorithm (i.e. computing of some global data) are less than 10 % so that an efficiency of always greater than 90 % is reached (92 - 97 %) . It is worth noting that these results correspond to efficiency values of about 95 % that have been estimated for the calculation of two- and three-dimensional multigrid problems on the 256 processors SUPRENUM prototype / 9 / .

The neutronics code is now also running on the SUPRENUM simulator / 10 / and the SUPRENUM pre-prototype hardware. Further investigations will concern the effect of vector unit performance and the optimization necessary when an increasing number of processors causes decreasing vector lengths. Results will be reported in a later publication.

SOLUTION STRATEGY IN THERMAL HYDRAULICS

For application on the multiprocessor system SUPRENUM, the THERMIT-2 code has been parallelized, resulting in the module 3D-THERM. As mentioned before, the intrinsic SUPRENUM features such as the local memory / message passing and the dynamic processes concept had to be taken into account. However, the basic parallelization strategies are applicable to similar multiprocessor systems as well.

To minimize development and verification effort, all original THERMIT-2 routines used for numerical calculation were to be modified as little as possible. THERMIT-2 uses an indexing of channels as shown in Fig. 2 . An additional fictitious channel is introduced which is assumed to surround the whole domain in radial direction. Once having supplied this fictitious channel with appropriate data, all channels can be calculated in the same manner regardless of whether they are situated at the radial boundary of the treated geometry or not. This allows easy parallelization by domain splitting in radial direction.

In a first step, a coarse grid separated in x- and y-direction can be defined for the mapping of channels to the regions controlled by individual processors. The channel indexing has been solved as illustrated in Fig 3 . For each processor the channels are indexed from 1 to NACP, where NACP is the number of calculated channels for the considered processor P. They are called here "active channels". Then NACP+1 is given as index for the fictitious channel for this processor, whereas NACP+2, ..., NTCP denote the indices for the channels belonging to neighboured processor regions but giving boundary conditions for the considered processor ("boundary channels") . NTCP is the total number of channels which are related to the processor. Similarly, the variables NARODS and NTRODS can be defined as number of active rods (to be calculated) and total number of rods (additionally rods which give boundary conditions) , respectively. Now, in dependence of NTCP and NTRODS, the pointers for the partitioning of the working storage array for each individual processor can be calculated and all relevant arrays can store both data for active and for boundary channels and rods. Finally NACP and NARODS are assigned to the corresponding original THERMIT-2 variables NC and NRODS, and all calculation subroutines can be used without any modifications for calculation on the processors.

The neighbourhood relations which describe the required data exchange between the processors result from both the employed differencing schemes and from the

mode in which the velocities in x- and y-direction are stored in the memory. In THERMIT-2 most quantities are cell-centered (Fig. 4 a), only the x-, y- and z-velocities are defined at the centers of the cell faces as shown in Fig. 4 b for a given channel with discrete coordinates (i,j) in the horizontal plane. For channel position (i,j) the quantities $v^x_{i-1/2,j}$ and $v^y_{i,j-1/2}$ are stored. Thus the velocities $v^x_{i+1/2,j}$ and $v^y_{i,j+1/2}$ are stored for channels at positions (i+1,j) and (i,j+1), respectively. The differencing scheme for the momentum equations introduces the quantities $v^x_{i,j-1/2}$ and $v^y_{i-1/2,j}$ which are defined as averages of surrounding velocities as illustrated in Figs. 4 c,d . Therefore the 3-dimensional thermal-hydraulic calculation of a channel (i,j) generally requires data from up to 6 neighbouring channels. This implies a processor communication with up to 6 neighbours for a considered processor as shown in Fig. 5 a. Of course, if a partition of the radial channel geometry into stripes is selected (special case of a separated grid), only two processor neighbours exist (Fig. 5 b). Moreover, if the thermal-hydraulic calculation is reduced to the 1-dimensional parallel channel model where no crossflows have to be calculated, no connections at all between processors are given.

Hence the main task was the development of routines serving for distribution of initial data from master to the processors, restoring the results of each processor into the master´s working storage array for output purposes, data exchange between processors during and after the Newton iteration procedure, and synchronization tasks. Besides, some modifications of existing control routines were necessary. Synchronization tasks are performed by the master and include checking of convergence status of inner iterations (solution procedure for the pressure problem) and of Newton iterations, calculation of the maximum allowable time step, error handling, and control of requested output.

Fig. 6 shows processor statistics obtained with the SUPRENUM simulator / 10 / for a steady-state 16-subchannel problem. If this problem is calculated with the parallel channel model, processor load factors of up to 95 % result. Processor occupation decreases in the calculation cases including crossflow, due to rising communication needs between the processors. For these cases, the comparatively high load factor of the master stems from the fact that, for reasons of numerical stability, solution of the resulting pressure problem equations during the Newton iteration has been left to the master in this first version of 3D-THERM.

COUPLING REACTOR CORE AND COOLANT LOOP ANALYSIS

In order to determine the nuclear reactor´s behaviour as realistically as possible, two main aspects have to be taken into account. Firstly, it is obvious that the reactor core has to be treated by coupling neutron kinetics and thermal hydraulics, thereby modeling their respective feedback mechanisms. Secondly, appropriate boundary conditions for the core (e.g. mass flow and temperature of coolant at the inlet, boron concentration, pressure) must be specified. To calculate these, it is clearly desirable to include the external coolant circuits in the analysis, especially for longer-term transients.

The final layout of the complete plant simulation system 3D-SIM is shown in Fig. 7 . The shaded area comprises the reactor core itself. The basic equations and the parallelization of the corresponding modules 3D-NEUT (neutron kinetics) and 3D-THERM (thermal hydraulics) were described in the previous chapters. Their coupling is effected by updating cross sections and power density, in dependence of thermal-hydraulic conditions and nuclear fission rate, respectively, during an inner iteration process. In the following, the function of the loop module NLOOP shall be outlined.

In its original form, NLOOP was developed for application in a PWR nuclear plant analyzer. Thus it evaluated all of the main plant characteristics, such as mass and energy flow in primary and secondary circuits, reactions of the safety related instrumentation and control systems, and core neutronics based on a point kinetics model. The plant geometry was defined by input, as well as single perturbations of parameters or random combinations thereof. For application of NLOOP in 3D-SIM, however, all the subroutines dealing with the neutronic and thermal-hydraulic reactor core modeling (hitherto restricted to one dimension) have to be cut out, their functions being taken over by the more sophisticated 3-dimensional modules 3D-NEUT and 3D-THERM. After parallelization on a loop-to-processor-mapping basis, the remaining NLOOP parts will be integrated as a 3D-SIM module as depicted in Fig. 7 . Now NLOOP stands for simulation of only the primary and secondary coolant circuits and the related components. It receives the core outlet conditions calculated by 3D-THERM and in turn provides the core calculation modules with updated coolant inlet conditions, together with the external perturbations (e.g. control rod movements) to be taken into account in the next time step.

Coupling of NLOOP to the core calculation modules will necessitate an optimization of time integration procedures. It remains to be determined to which degree the "inner" iterations between neutronics and thermal hydraulics have to be converged before proceeding to the loop calculations. Besides, the fact that NLOOP requires only a small number of processors might affect overall efficiency. To avoid the other processors being idle during loop calculations, further splitting of NLOOP tasks or concurrent 3D-NEUT/3D-THERM - NLOOP computing prior to synchronization seems feasible, so that overall efficiency will remain high.

CONCLUSIONS

Judging by the results received up to now it can be foreseen that with the described numerical methods and programs efficiencies of 80 - 90 % will be reached when calculating complex 3-dimensional nuclear reactor problems on multiprocessor systems. With the advent of massively parallel supercomputing architectures this experience opens up new perspectives for unprecedented spatial resolution and improved physical modeling, combined with reduced computing times directed towards real-time simulation. This means that problems in benchmarking and high accuracy design applications which hitherto could not be treated now can be solved. Commercially available and competitive parallel machines (mainframes and minicomputers) for the scientific and engineering community will facilitate distributed computing on engineering workstations and incorporation of computer aided engineering (CAE) tools as well as real time applications for training simulators, operator support, and power station control and protection.

REFERENCES

/ 1 / Händler, W., E. Maehle, K. Wirl : The DIRMU Testbed for High Performance Multiprocessor Configurations. Proceedings of the First International Conference on Supercomputing Systems, St. Petersburg, Florida, 1985

/ 2 / Trottenberg, U. : The SUPRENUM Project: Idea and Current State. University of Bern, Speedup, Vol.2 (1), pp. 20-24 (1988)

/ 3 / Solchenbach, K., U. Trottenberg : SUPRENUM - a survey of the system and the applications. Kerntechnik, Vol. 52 (3), pp. 175-180 (1988)

/ 4 / Finnemann, H., H. Raum : Nodal Expansion Method for the Analysis of Space-Time Effects in LWRs. Proceedings of a Specialists' Meeting on Calculation of 3-Dimensional Rating Distributions in Operating Reactors, Paris, November 26-28, 1979

/ 5 / Kelly, J. E., S. P. Kao, M. S. Kazimi : User's Guide for THERMIT-2: A Version of THERMIT for both Core-Wide and Subchannel Analysis of Light Water Reactors. Massachusetts Institute of Technology, Report MIT-EL-81-029 (1981)

/ 6 / Loomis, J., W. H. Reed, A. Schor, H. B. Stewart, L. Wolf : THERMIT: A Computer Program for Three-Dimensional Thermal-Hydraulic Analysis of Light Water Reactor Cores. Electric Power Research Institute, Report EPRI NP-2032 (1981)

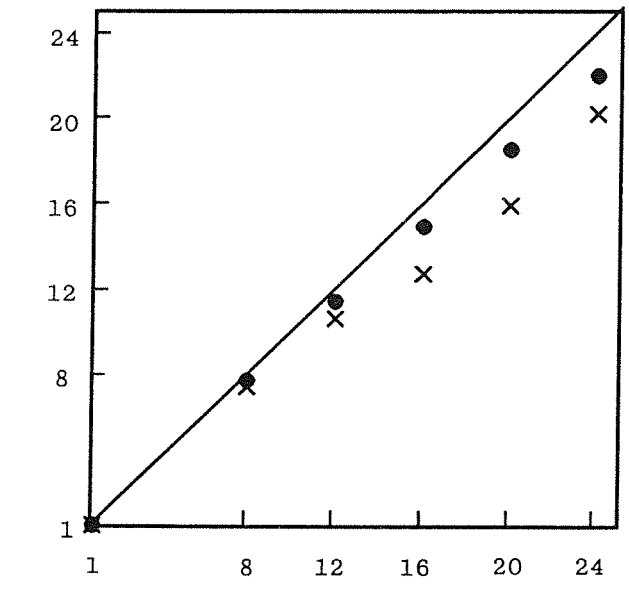
/ 7 / Finnemann, H., J. Volkert : Parallel Multigrid Algorithms Implemented on Memory-Coupled Multiprocessors. Nuclear Science and Engineering, Vol. 100, pp. 226-236 (1988)

/ 8 / Finnemann, H., J. Brehm, E. Michel, J. Volkert : Multigrid solution of diffusion equations on distributed memory multiprocessor systems. Kerntechnik, Vol. 52 (3), pp. 169-174 (1988)

/ 9 / McBryan, O. A. : New architectures: Performance highlights and new algorithms. Parallel Computing, Vol. 7, pp. 477-499 (1988)

/ 10 / Tietz, C. : Das Benutzer-Interface des SUPRENUM-Simulationssystems. Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, FRG, (1988)

Fig. 1 : Performance Curves and Efficiencies for the Solution of the Neutron Diffusion Equation With Domain Splitting Into Stripes (**X**) and With Direct Projection of the Fuel Assemblies (●)



No. of processors	efficiency X	efficiency ●
1	1.00	1.00
8	0.93	0.97
12	0.89	0.96
16	0.80	0.94
20	0.80	0.93
24	0.85	0.92

Fig. 2 : Distribution of a 14-Channel Problem to 4 Processors

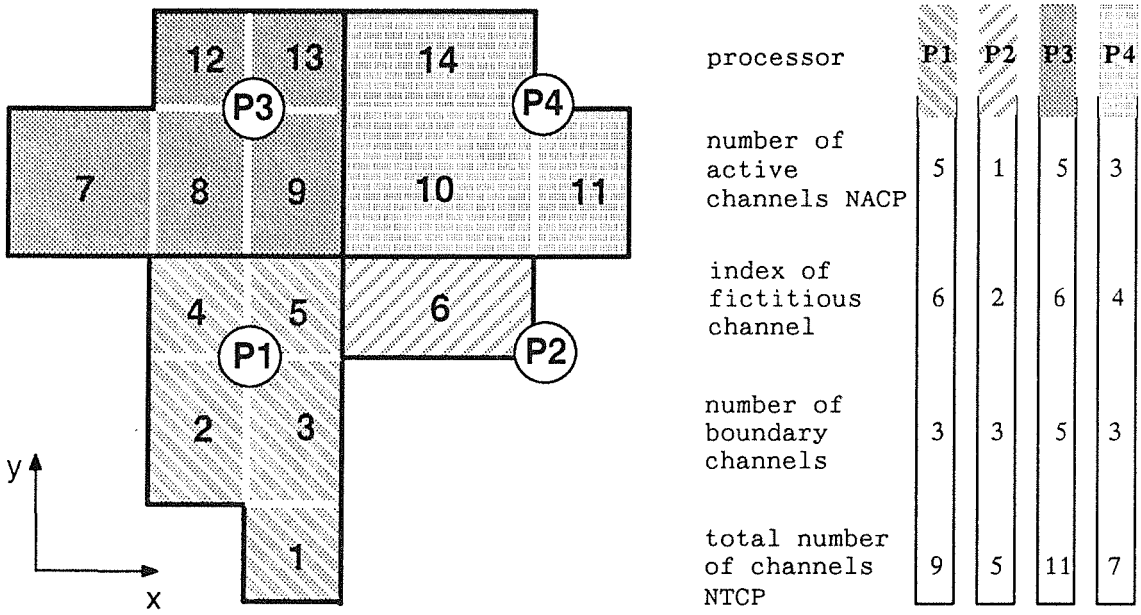


Fig. 3 : 3D-THERM Channel Indexing on the Processors
(Without Indexing of the Fictitious Channel)

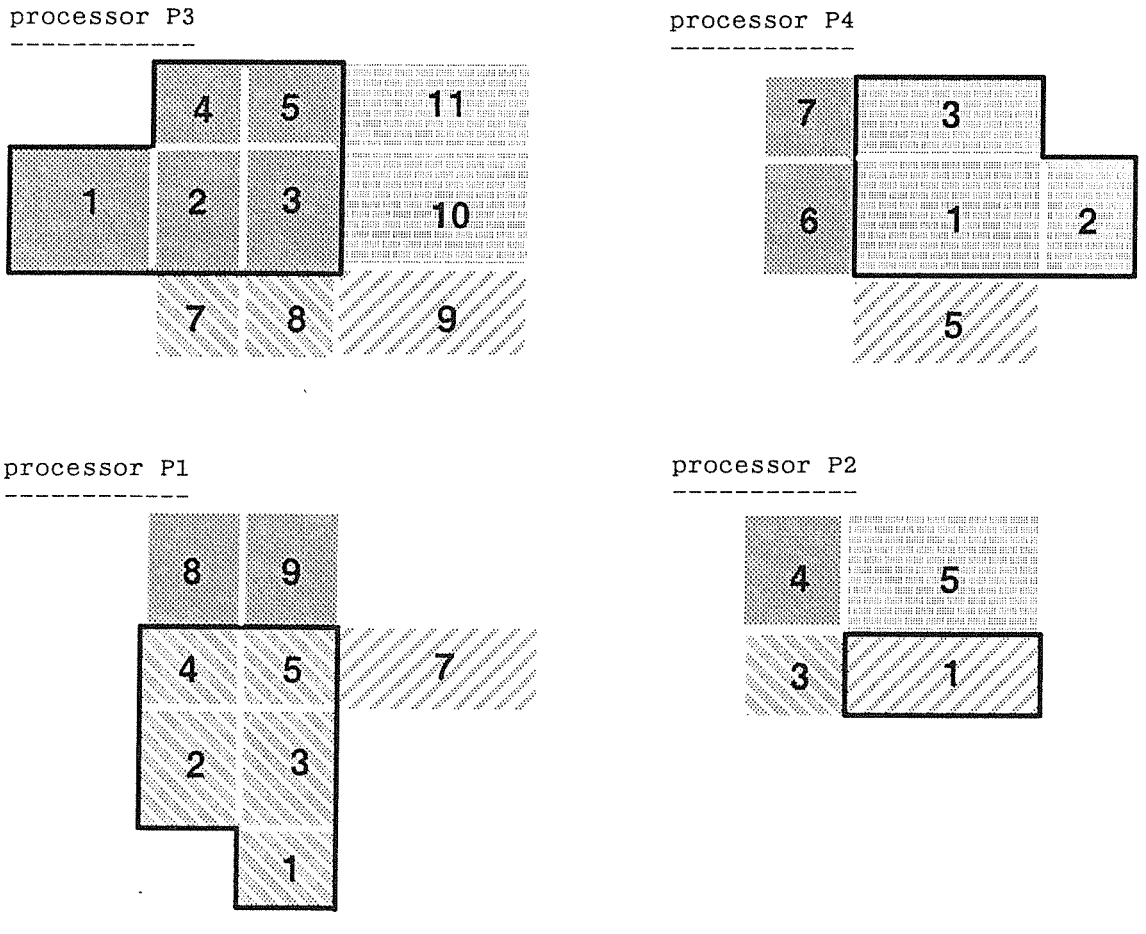
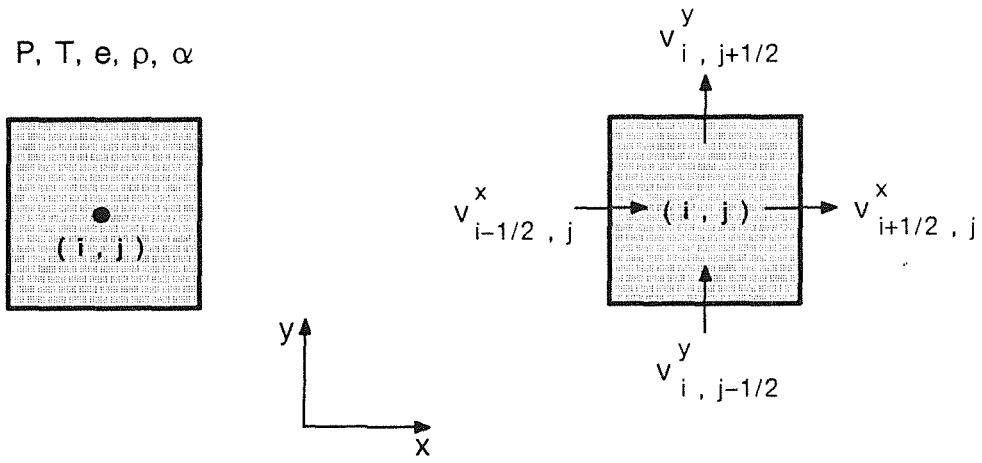


Fig. 4 : Basic Definitions of Thermal-Hydraulic Quantities

a) Cell Centered Quantities b) Definition of Velocities



c) Definition of $v^{x}_{i,\,j-1/2}$ d) Definition of $v^{y}_{i-1/2,\,j}$

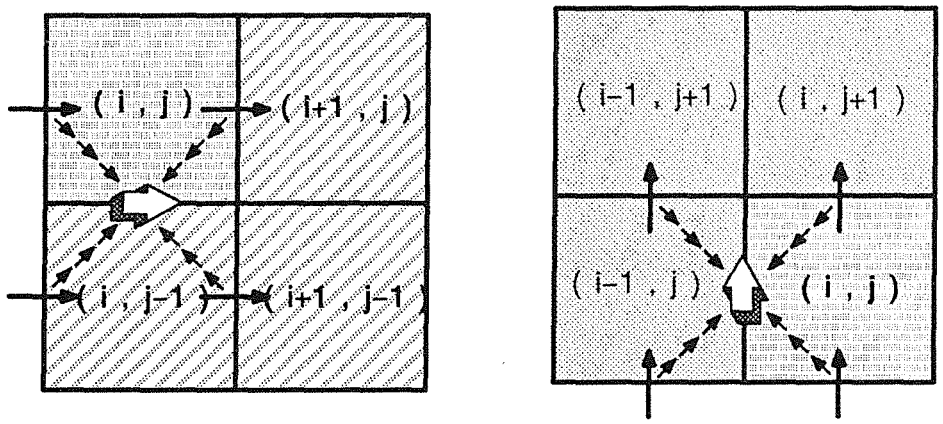


Fig. 5 : 3D-THERM Neighbourhood Relations for a Processor P

a) Domain Splitting by Separated x-y-Grid b) Domain Splitting Into Stripes

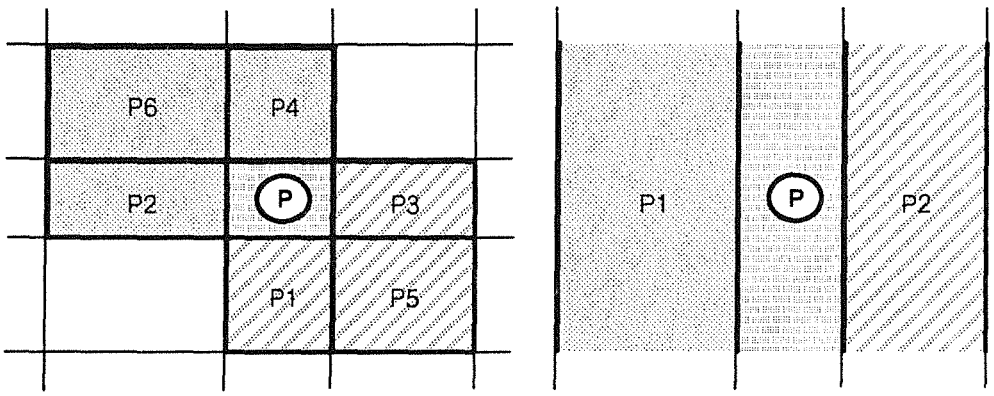


Fig. 6 : 3D-THERM Processor Statistics for a 16-Subchannel Problem
a) With Crossflow Calculation, 2 Processors
b) With Crossflow Calculation, 4 Processors
c) Parallel Channels, 4 Processors

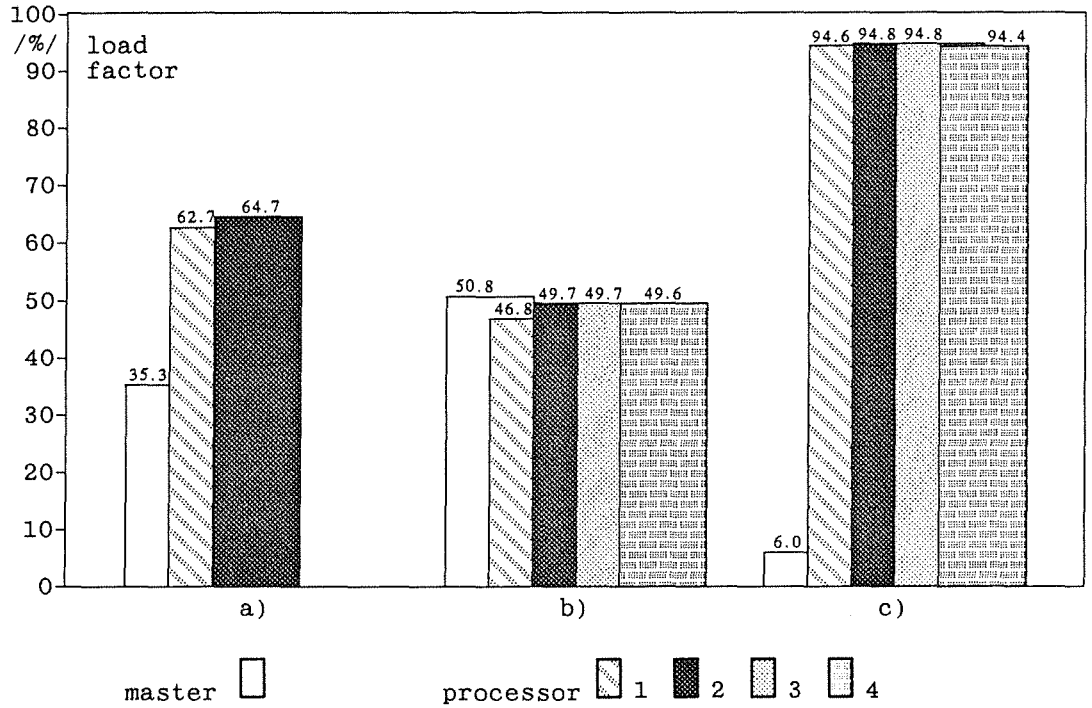
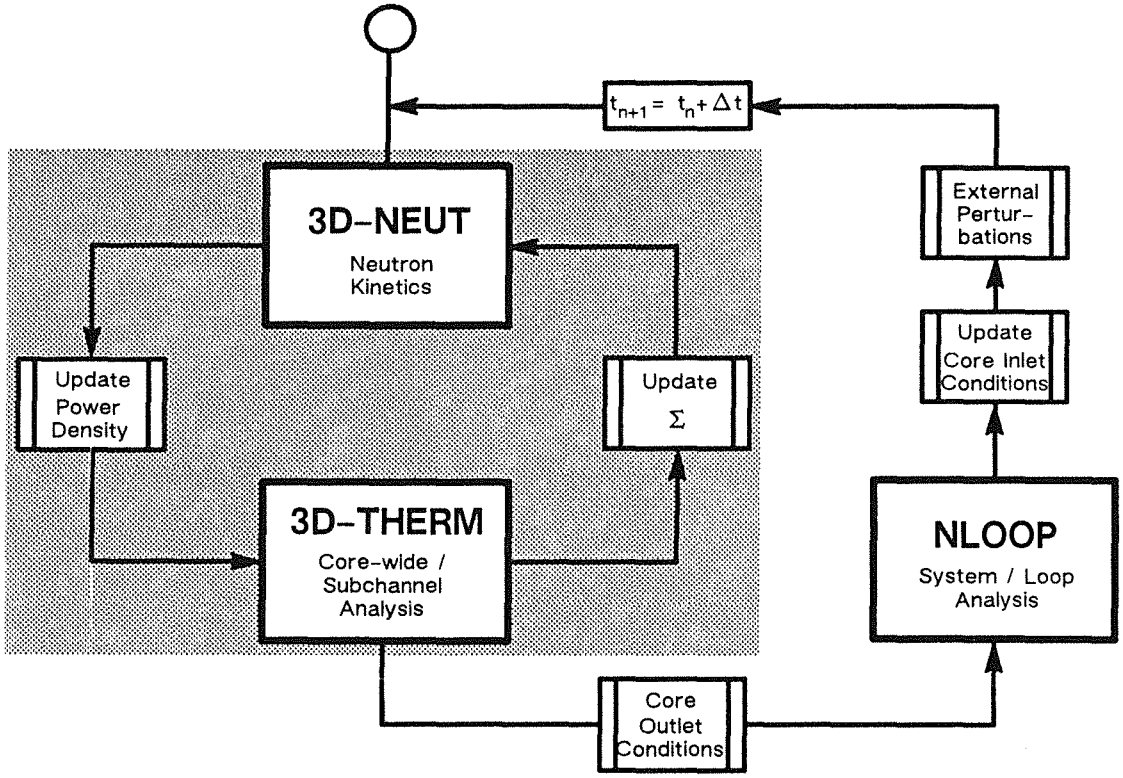


Fig. 7 : Layout of the Complete Plant Simulation System 3D-SIM



Teilchensimulation in der Plasmaphysik

Manfred Alef [1], David Seldner [1], Thomas Westermann [2]

Kernforschungszentrum Karlsruhe GmbH
[1] Institut für Datenverarbeitung in der Technik (IDT)
[2] Abteilung für numerische Physik (HDI-3)
Postfach 3640, D-7500 Karlsruhe 1

Zusammenfassung

Zur numerischen Unterstützung der experimentellen Entwicklung von Hochstrom-Ionendioden wurde im Kernforschungszentrum Karlsruhe das Programmpaket BFCPIC entwickelt. BFCPIC ist ein zweidimensionaler, elektro- und magnetostatischer „Particle-in-Cell"-Code (PIC), dessen Gitterkonzept auf randangepaßten Koordinaten (boundary-fitted coordinates - BFC) basiert.

In diesem Bericht werden die wichtigsten Komponenten dieses Programmpakets kurz skizziert und die Parallelisierungskonzepte für die Implementierung auf SUPRENUM vorgestellt.

Einleitung

Zum Studium von Materie unter extremen Bedingungen werden Apparaturen (Hochstrom-Ionendioden) entwickelt, welche die dem elektromagnetischen System zugeführte Energie in fokussierende Teilchenstrahlenergie umwandeln. Die Fokussierung der Ionen erfolgt dabei primär durch die Formgebung der Anode (ballistische Fokussierung).

Um einen hohen Wirkungsgrad zu erzielen, müssen die durch die Elektronenströme auftretenden Verluste durch Magnetfelder reduziert werden. Diese Magnetfelder beeinflussen wiederum die Fokussierungseigenschaften der Diode. Das Problem bei der Entwicklung von Ionendioden liegt nun darin, die ballistische mit der magnetischen Fokussierung in Einklang zu bringen.

Als Hilfswerkzeug für die Konstruktion dieser Anlagen wurde ein „Particle-in-Cell"-Code zur numerischen Simulation der Dioden entwickelt [1]. Dieser Code ermöglicht es, bei gegebener Geometrie die Bewegung der elektrisch geladenen Teilchen (Elektronen, Ionen) in den zugehörigen elektromagnetischen Feldern zu

Das diesem Bericht zugrundeliegende Vorhaben wurde teilweise mit Mitteln des Bundesministers für Forschung und Technologie unter dem Förderkennzeichen ITR8502K/4 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

simulieren. Die Aufgabe ist es anschließend, den Experimentatoren Hinweise für die Weiterentwicklung der Dioden zu liefern.

Obwohl nur rotationssymmetrische Geometrien betrachtet werden, bei denen eine zweidimensionale Beschreibung ausreichend ist ((r,z)-Zylinderkoordinaten), sind die Simulationen mit einem sehr hohen Rechenaufwand verbunden. Um vertretbare Rechenzeiten zu erhalten, werden einerseits möglichst effiziente numerische Algorithmen entwickelt. Dies gilt insbesondere für die zur Feldberechnung eingesetzten Mehrgittermethoden. Andererseits ist bei zunehmend komplizierteren Aufgabenstellungen der Einsatz von Hochleistungsrechnern (Vektor- und/ oder Parallelrechnern) erforderlich.

Mathematisches Modell

Zur selbstkonsistenten Beschreibung des elektromagnetischen Systems wird die „Particle-in-Cell"-Methode angewendet. Dabei können die Bewegungen elektrisch geladener Teilchen (Elektronen, Ionen) sowohl in von außen vorgegebenen als auch den selbsterzeugten elektromagnetischen Feldern modelliert werden [1, 2].

Um die auf die Teilchen wirkenden Kräfte effizient berechnen zu können, wird das Berechnungsgebiet mit einem geeigneten Gitter versehen (vgl. einführende Bücher über die „Particle-in-Cell"-Methode [3, 4]).

Die Ladung jedes Teilchens wird auf die benachbarten Gitterpunkte verteilt und damit dort die Ladungs- und Stromdichten berechnet. Mittels dieser Dichten werden dann die elektrischen und magnetischen Felder in den Gitterpunkten bestimmt [5]. Die Kraft, die anschließend auf die Teilchen wirkt, erhält man durch Interpolation der Feldstärken in den umliegenden Gitterpunkten auf die Teilchenorte [6, 7].

Durch Lösen der Bewegungsgleichung (Lorentz-Gleichung) werden dann die neuen Phasenraumkoordinaten berechnet [8]. Schließlich werden aus den neuen Teilchenorten und -geschwindigkeiten die Ladungs- und Stromdichten in den Gitterpunkten bestimmt, aus denen im nächsten Zeitschritt wiederum neue Felder berechnet werden.

In Abb. 1 ist diese Zeitschleife schematisch dargestellt.

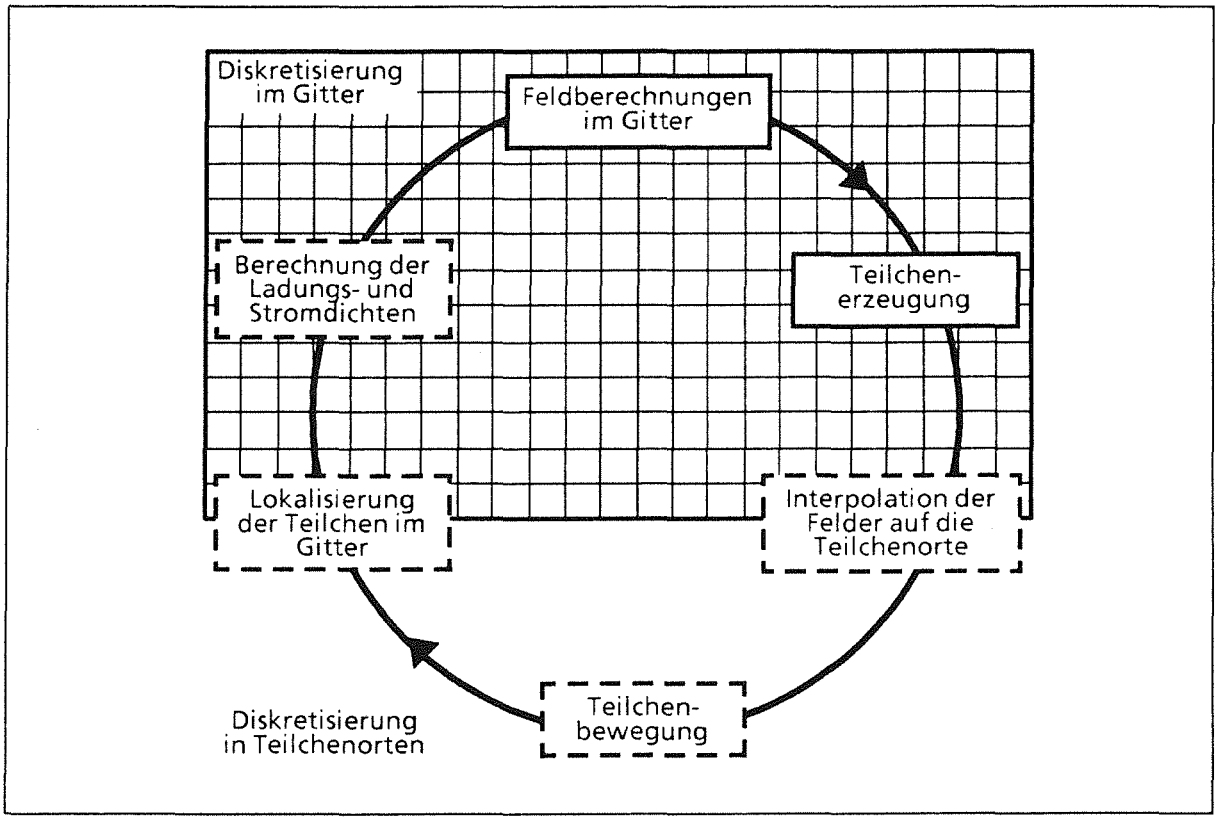


Abb. 1: Zeitschleife und zugrundeliegende Diskretisierung. Ein gestrichelter Rand bedeutet, daß in diesem Modul die Teilchen unabhängig voneinander behandelt werden.

Der Zustand des Systems wird also in jedem Zeitschritt einerseits durch die Ortskoordinaten, Geschwindigkeiten und Ladungen der einzelnen Teilchen, andererseits durch die Ladungs- und Stromdichten und die sich daraus ergebenden elektromagnetischen Felder an den Gitterpunkten beschrieben.

Bisher werden die zeitabhängigen Terme in den Maxwellgleichungen vernachlässigt. Dabei werden zunächst auf einem vorgegebenen Berechnungsgitter das elektrostatische Feld **E** durch Lösen der Poisson-Gleichung

$$\Delta\Phi = -\rho/\varepsilon, \quad \mathbf{E} = -\operatorname{grad}\Phi$$

sowie das magnetostatische Feld **B** durch Anwendung des Ampèreschen Gesetzes

$$\oint_C (\mathbf{B},\mathbf{t})\,ds = \mu \iint_F (\mathbf{j},\mathbf{n})\,df$$

(F: Fläche mit Normale **n**, C: Randkurve von F mit Tangente **t**)
berechnet. Hierbei ist ρ die Ladungs- und **j** die Stromdichte, ε und μ sind materialabhängige Konstanten.

Neue Ladung (Ionen, Elektronen) wird an den Elektrodenzellen gemäß den dort herrschenden Feldern in jedem Zeitschritt emittiert und auf eine vorgegebene Anzahl von Simulationsteilchen aufgeteilt.

Zur Berechnung der Felder werden randangepaßte Berechnungsgitter verwendet [9, 10] (Beispiele s. Abb. 2). Diese haben gegenüber äquidistanten Gittern den Vorteil der großen Flexibilität hinsichtlich der Geometrieerfassung. Dabei können auch Gebiete mit krummlinigen oder geknickten äußeren und inneren Rändern modelliert werden, ohne daß Interpolationen von Randwerten erforderlich werden. Diese Vorgehensweise ist jedoch mit einem Mehraufwand bei der Lokalisierung der Teilchen (Bestimmung der Gitterzelle, in der es sich befindet) sowie bei der Interpolation der Felder auf die Teilchenorte verbunden [6, 7].

Die Differentialgleichungen zur Feldberechnung werden im randangepaßten Berechnungsgitter räumlich diskretisiert. Um eine ausreichend gute räumliche Auflösung des Systems zu erhalten, sind bis zu mehreren Tausend Gitterpunkte erforderlich. Die Lösung der Poisson-Gleichung im Berechnungsgitter erfolgt mittels eines Mehrgitterverfahrens [5].

Die Lorentz-Gleichung wird mit einem expliziten Verfahren („Leapfrog-Algorithmus") zeitlich diskretisiert. Es werden etwa zehn- bis fünfzigtausend Simulationsteilchen verwendet [1, 2].

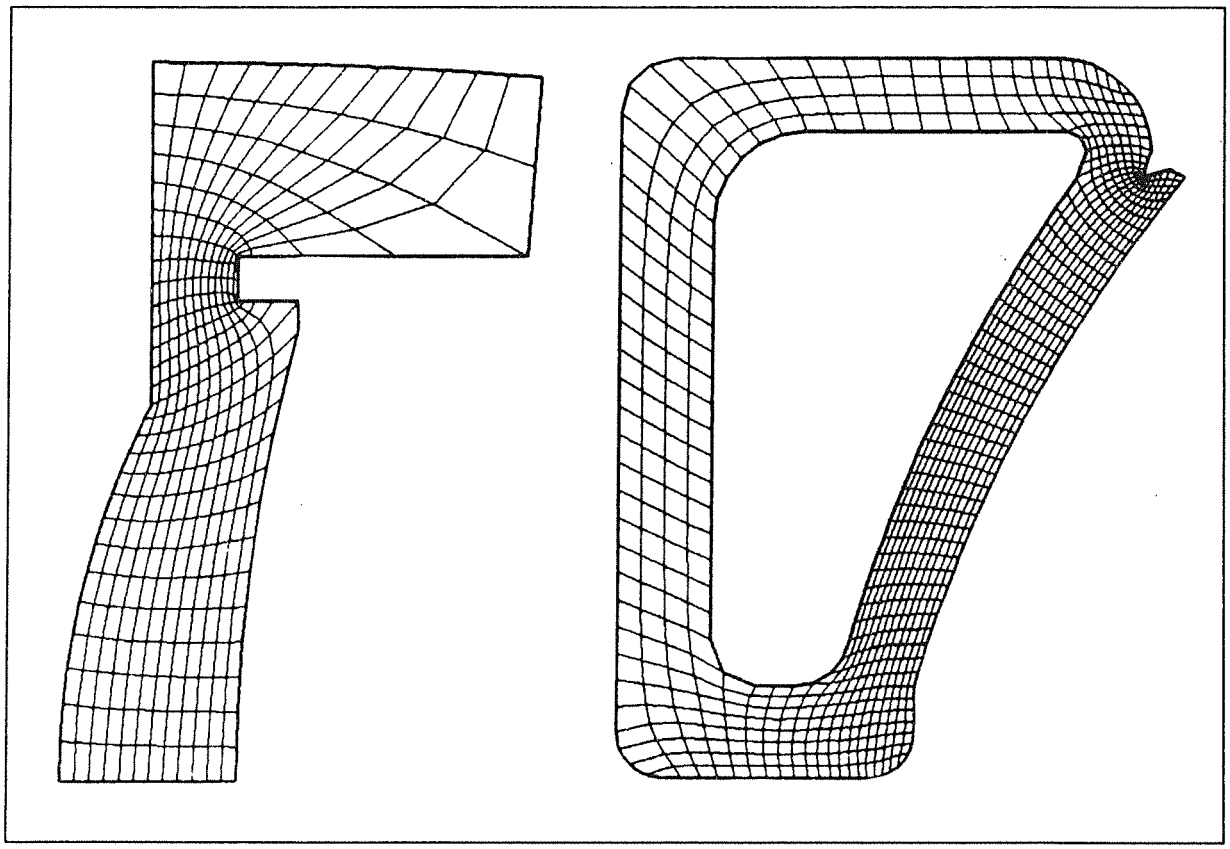


Abb. 2: Beispiele randangepaßter Gitter: links: Pinchdiode, rechts: B_θ-Diode

Parallelisierung auf SUPRENUM

Feldberechnungen im Gitter

Die Parallelisierung der Feldberechnungen basiert auf einer Aufteilung des Rechengebiets in Teilgebiete, die jeweils ungefähr die gleiche Anzahl von Gitterpunkten enthalten [11, 12, 13]. Die günstigste Lösung ist dabei die Aufteilung in nur einer Dimension in „Streifen", die eine feste Anzahl paralleler Gitterlinien umfassen (Abb. 3). Dadurch wird gegenüber einer zweidimensionalen Aufteilung der Kommunikationsaufwand verringert, und wegen der größeren Vektorlängen ergibt sich eine effizientere Vektorisierung. Andererseits wird zwar der mögliche Parallelisierungsgrad eingeschränkt, da bei einer zweidimensionalen Aufteilung kleinere Teilgebiete möglich sind. Eine so feine Parallelisierung wäre jedoch wegen des sehr hohen Kommunikationsanteils ineffizient.

Um die Anzahl der zwischen den parallelen Prozessen auszutauschenden Nachrichten zu minimieren, werden die folgenden Konzepte angewendet [11]:

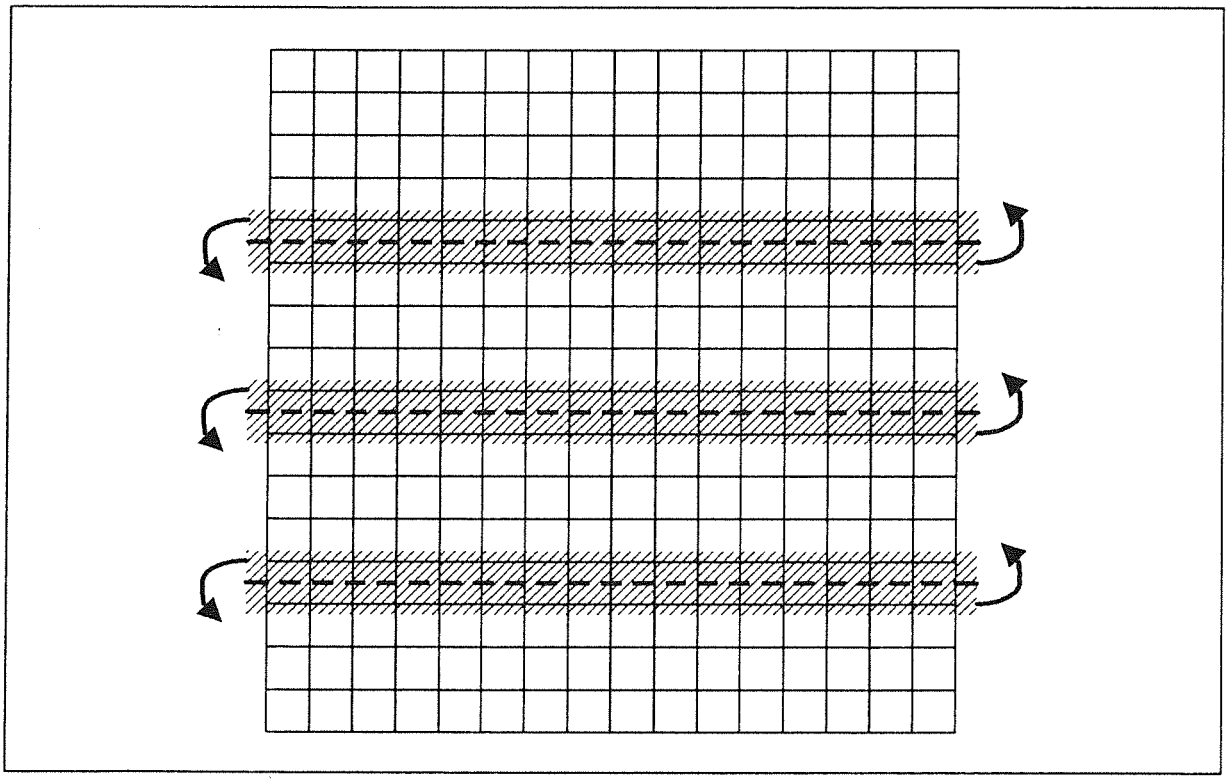


Abb. 3: Prinzip der Gebietsaufteilung in Streifen, die jeweils einem parallelen Prozeß zugewiesen werden.
Bestimmte Zwischenergebnisse aus randnahen Gitterpunkten (s. schraffierte Bereiche) werden von beiden Nachbarprozessen benötigt. Sie müssen nach jeder Aktualisierung vom „zuständigen" Prozeß an den Nachbarn gesendet werden.

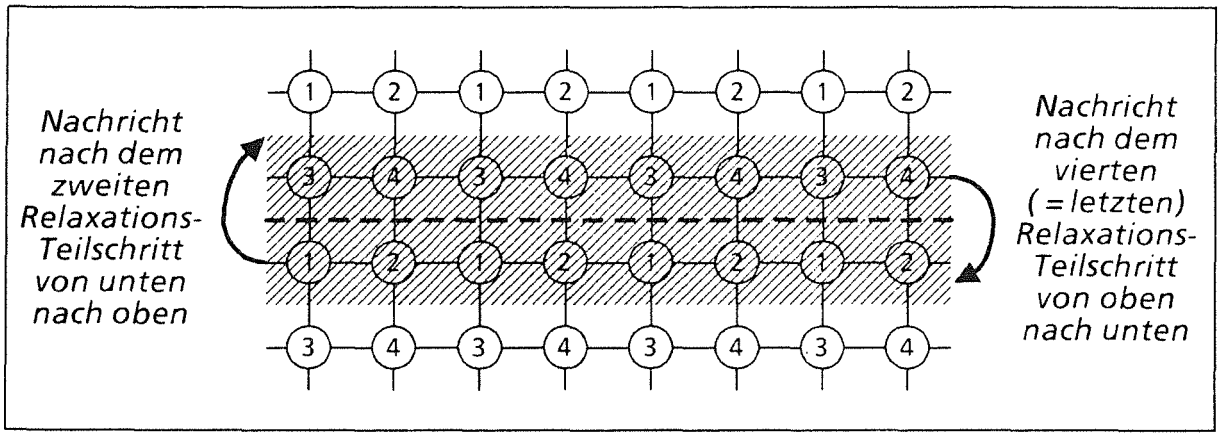


Abb. 4: Vier-Farb-Relaxationsschema mit minimalem Kommunikationsaufwand: Zuerst werden die Punkte „1", dann „2" durchlaufen und die in beiden „Farben" berechneten Zwischenergebnisse an den Nachbarn gesendet (s. Pfeil). Anschließend wird dieses Vorgehen mit den Punkten „3" und „4" wiederholt.

- Zur Fehlerglättung innerhalb des Mehrgitterverfahrens wird eine spezielle Vier-Farben-Relaxation (s. Abb. 4) verwendet, die pro Relaxationsschritt nur eine einzige Nachricht mit jedem Nachbarprozeß erfordert.
- Auf groben Gittern sinkt der Umfang der zwischen zwei Kommunikationsaktivitäten durchzuführenden Berechnungen, d.h. der Gesamtaufwand ist um so mehr durch die Kommunikation bestimmt, je gröber das Gitter ist. Deshalb werden beim Übergang auf sehr grobe Gitter die Teilgebiete jeweils mehrerer Prozesse in nur einem Prozeß vereinigt (Agglomeration).

Teilchenbehandlung

Da die Interpolation, Teilchenfortbewegung und die Lokalisierung der Teilchen im Gitter für jedes Teilchen unabhängig von den anderen erfolgt, ist es sinnvoll, diese Module zusammen mit der Bestimmung der Ladungs- und Stromdichten en bloc zu parallelisieren. Dazu wird jedem Prozeß eine bestimmte, möglichst gleichgroße Anzahl von Teilchen zugewiesen [12, 13, 14].

Jeder Prozeß erhält die Feldstärken in den Gitterpunkten und behandelt dann die ihm zugewiesenen Teilchen. Die aus den neuen Positionen bestimmten Ladungs- und Stromdichten werden von einem Steuerprozeß gesammelt und an die Feldprozesse gesendet.

Gesamte Zeitschleife

Abb. 5 zeigt den parallelen Ablauf der gesamten Zeitschleife. Bei dieser Vorgehensweise wird bei jedem Wechsel zwischen Feld- und Teilchenberechnung ein globaler Informationsaustausch notwendig: Zunächst müssen die Felder an alle Teilchenprozesse versendet werden. Umgekehrt müssen die Ladungs- und Stromdichten über alle Teilchenprozesse aufaddiert und dann auf die Feldprozesse verteilt werden. Dieses Sammeln und Verteilen der Daten (unter Verwendung einer

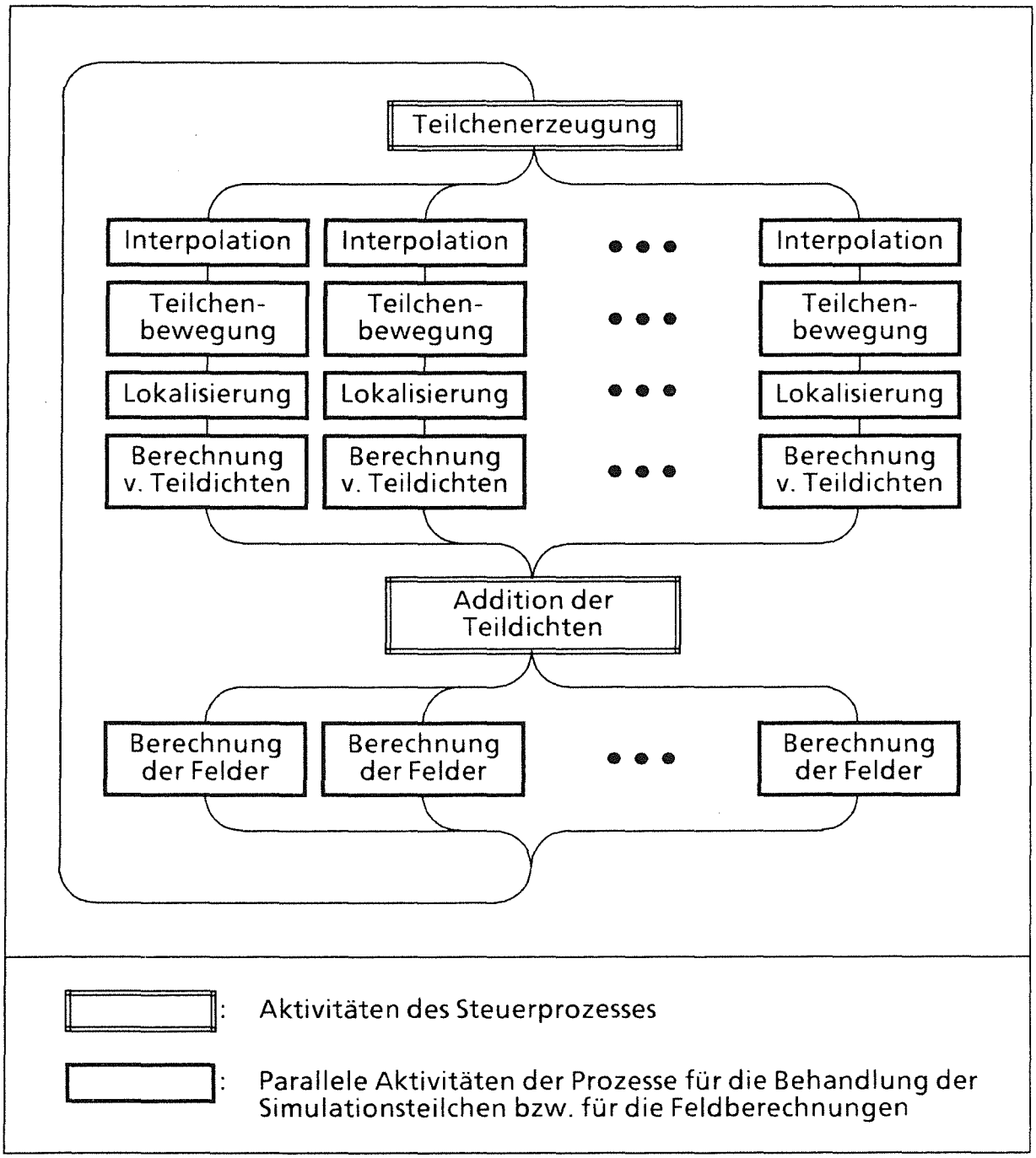


Abb. 5: Prinzip der Parallelisierung der Zeitschleife.

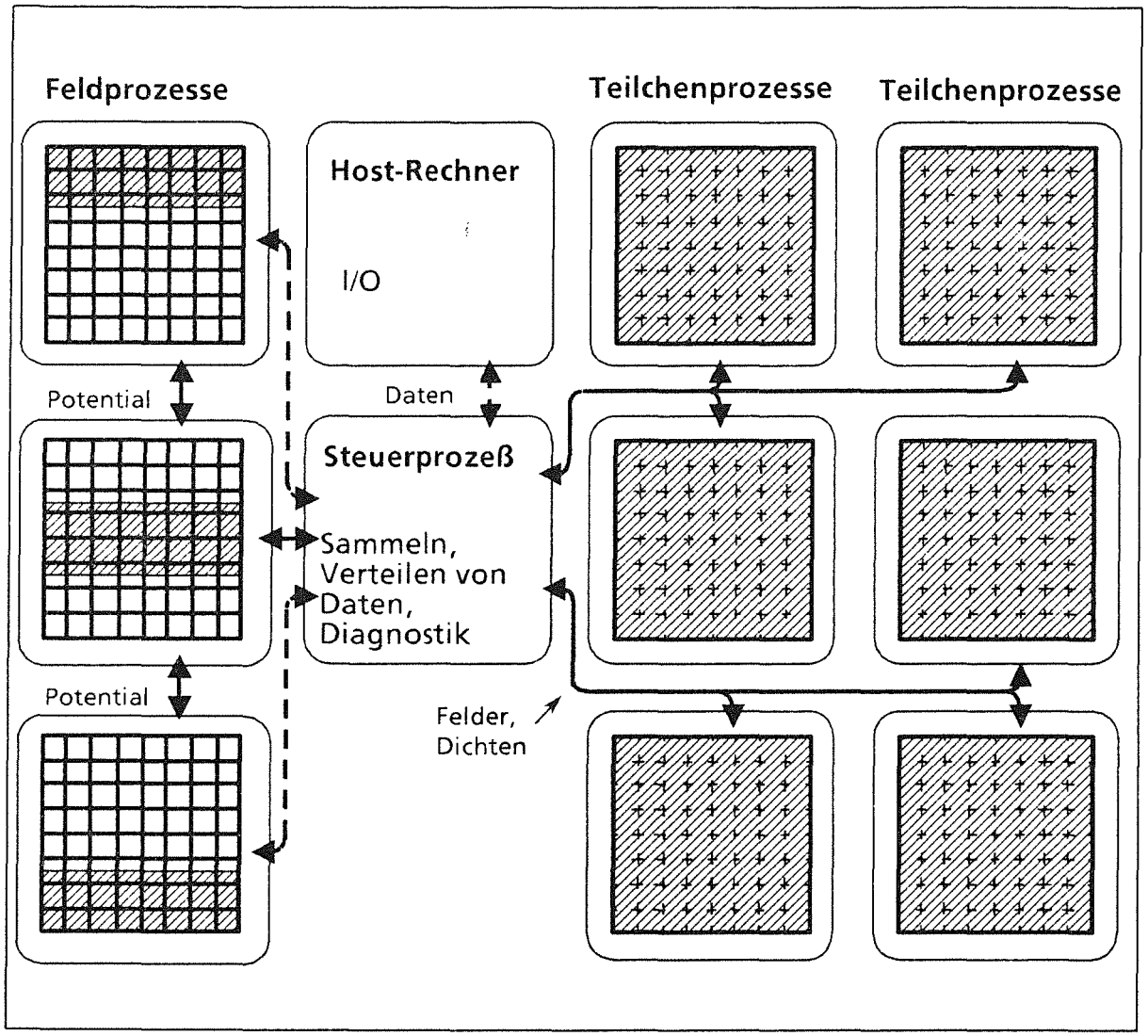


Abb. 6: Kommunikation zwischen den einzelnen Prozessen.

Baumstruktur) geschieht in einem speziellen Steuerprozeß, der die Diagnostik sowie die Kommunikation mit dem auf dem Vorrechner (Host) ablaufenden Hauptprozeß (der für die gesamte Ein-/ Ausgabe zuständig ist) durchführt (vgl. Abb. 6).

Eine alternative Vorgehensweise, bei der weniger Speicherplatz benötigt wird und die Kommunikation weitestgehend lokal abläuft, wird in [14] vorgestellt. Dabei werden die Teilchen gebietsmäßig aufgeteilt gemäß der Streifenaufteilung für die Feldberechnungen.

Erste vorläufige Ergebnisse

Da der SUPRENUM-Rechner noch nicht vollständig fertiggestellt ist, konnten bisher nur Testläufe auf einem Cluster (mit 16 Knotenrechnern) des SUPRENUM-Prototyps gerechnet werden. Dabei war noch keine Vektorverarbeitung möglich.

Die Kommunikation (Versenden der Felder und der Dichten zwischen Steuerprozeß und den parallelen Teilchenprozessen, s. Abb. 5 und 6) war aus Testgründen in jeweils mehrere Einzelnachrichten aufgespalten und zu Kontrollzwecken mit Bestätigungen versehen.

Bei diesen Tests war es noch nicht möglich, sowohl die Feldberechnungen als auch die Teilchenbehandlung parallel auszuführen. Zur Berechnung der Felder wurde daher die skalare Programmversion verwendet. Dies war auch deswegen sinnvoll, weil für die Testläufe nur ein sehr grobes Gitter (11 x 21 Punkte) verwendet werden konnte, das keine effiziente Parallelisierung erlaubt hätte. (Für Abschätzungen der Parallelisierungseffizienz dieses Programms verweisen wir auf [11].)

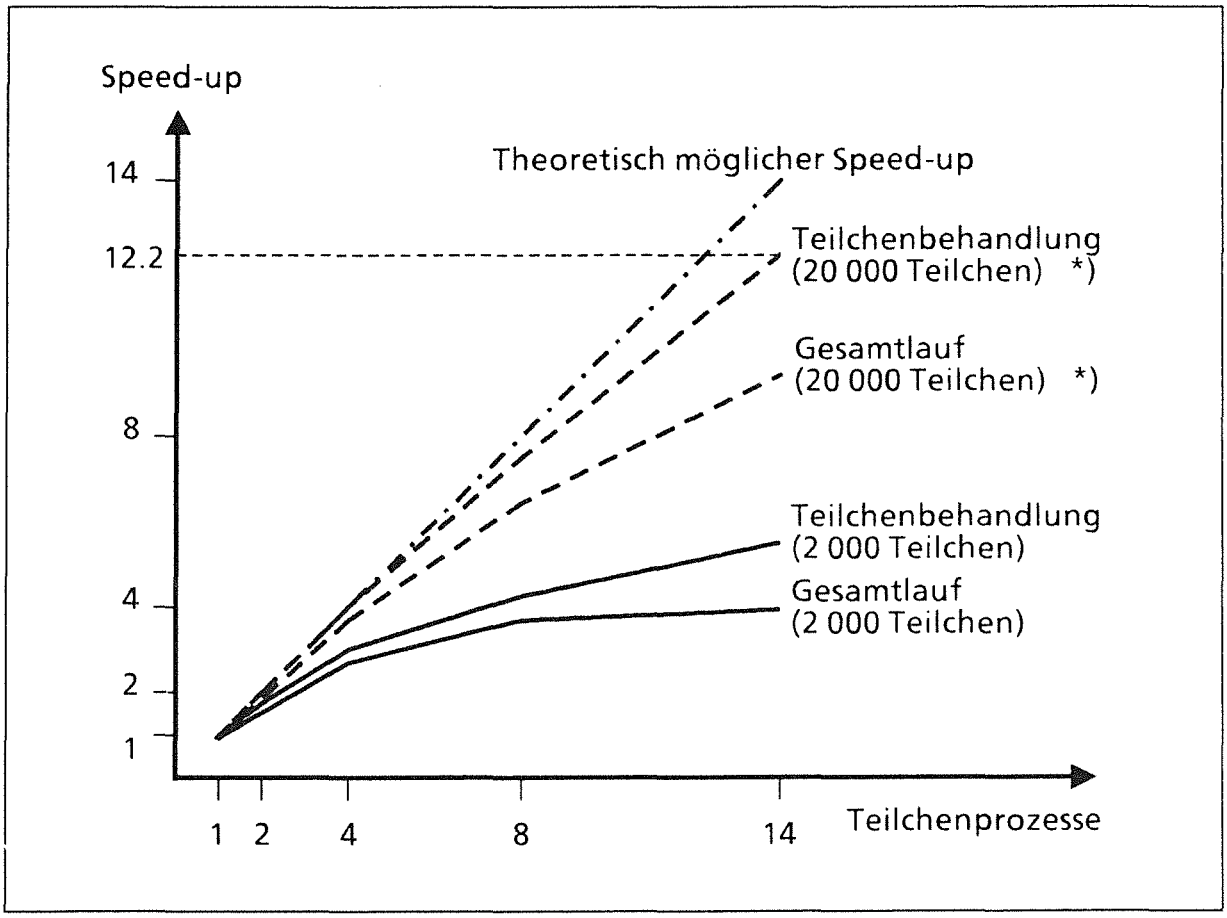


Abb. 7: Speed-up für die Teilchenbehandlung auf 2, 4, 8 und 14 Knotenrechnern.
**) Schätzwerte*

Abb. 7 zeigt den Speed-up, der bei der Durchführung der Teilchenbehandlung auf 2, 4, 8 und 14 parallelen Teilchenprozessen in 1 000 Zeitschritten gemessen wurde. Dazu wurden bis zu insgesamt 2 000 Simulationsteilchen verwendet. Daneben sind auch die entsprechenden Kurven für den Gesamtlauf angegeben, die eine untere Grenze für den für die voll parallelisierte Zeitschleife (d.h. auch die Feldberechnungen erfolgen parallel) zu erwartenden Speed-up bilden.

Bei dieser relativ geringen Teilchenzahl waren die Knotenrechner jedoch noch nicht optimal ausgelastet. Deshalb wurden die Ergebnisse auf eine realistische Anzahl (insgesamt 20 000) von Teilchen extrapoliert, s. die gestrichelten Kurven in Abb. 7. Der bei 14 Prozessen zu erwartende Speed-up liegt dann bei ca. 12.

Aus den eingangs geschilderten Gründen können sich jedoch bei der endgültigen SUPRENUM-Version noch Abweichungen ergeben.

Literatur

[1] T. Westermann:
A Particle-in-Cell Method as a Tool for Diode Simulations.
Nucl. Instr. Meth. A263 (1988), S. 271-279

[2] T. Westermann:
Numerische Simulationen von technisch relevanten Ionen-Dioden mit der Particle-in-Cell Methode.
Kernforschungszentrum Karlsruhe, KfK 4510, Januar 1989

[3] R.W. Hockney, J.W.Eastwood:
Computer Simulation Using Particles.
McGraw-Hill, 1981

[4] C.K. Birdsall, A.B. Langdon:
Plasma Physics via Computer Simulations.
McGraw-Hill, 1985

[5] M. Alef:
Effiziente Berechnung elektrostatischer Potentiale mit Mehrgittermethoden in technischen Geometrien.
Kernforschungszentrum Karlsruhe, KfK 4613, Oktober 1989

[6] D. Seldner, T. Westermann:
Numerische Algorithmen für zweidimensionale Teilchen-Simulationsmodelle in technisch relevanten Geometrien.
Kernforschungszentrum Karlsruhe, KfK 4282, Juni 1987

[7] D. Seldner, T. Westermann:
Algorithms for Interpolation and Localization in Irregular 2D Meshes.
J. Comp. Phys. 79 (1988), S. 1-11

[8] T. Westermann:
Teilchen-Fortbewegung in elektro-magnetischen Feldern.
Kernforschungszentrum Karlsruhe, KfK 4325, Januar 1988

[9] J.F. Thompson, Z.U.A. Warsi, C.W. Mastin:
Boundary-Fitted Coordinate Systems for Numerical Solution of Partial Differential Equations - A Review.
J. Comp. Phys. 47 (1982), S. 1-108

[10] S. Ohring:
Application of the Multigrid Method to Poisson's Equation in Boundary-Fitted Coordinates.
J. Comp. Phys. 50 (1983), S. 307-315

[11] M. Alef:
Concepts for Efficient Implementation of Multigrid Methods on SUPRENUM-like Architectures.
Kernforschungszentrum Karlsruhe, KfK 4614, Oktober 1989

[12] M. Alef, D. Seldner, T. Westermann:
Numerische Algorithmen für elektrodynamische Modelle und ihre Implementierung auf Supercomputern.
Informatik-Fachberichte 150 (J. Halin, Hrsg.), Springer 1987, S. 298-305

[13] D. Seldner, M. Alef, T. Westermann, E. Halter:
Parallel Particle Simulation in High Voltage Diodes
(Algorithms and Concepts for Implementation on SUPRENUM).
Proceedings of the 2nd International SUPRENUM Colloquium "Supercomputing Based on Parallel Computer Architectures" (U. Trottenberg, Hrsg.),
Parallel Computing 7, 1988, S. 445-449

[14] D. Seldner:
Modelle zur Parallelisierung der Teilchenbehandlung in Particle-in-Cell Codes auf MIMD-Rechnern mit lokalem Speicher am Beispiel SUPRENUM.
Kernforschungszentrum Karlsruhe, KfK 4495, Januar 1989

Teilnehmerliste

Dipl.-Math. Manfred Alef
Kernforschungszentrum Karlsruhe GmbH
Institut für Datenverarbeitung in der Technik
Postfach 36 40, D-7500 Karlsruhe 1

Prof. Dr. G. Alefeld
Institut für Angewandte Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. M. Bär
Kernforschungszentrum Karlsruhe GmbH
Institut für Meteorologie und Klimaforschung
Postfach 36 40, D-7500 Karlsruhe 1

Bernd Blankenbach
Geophysikalisches Institut
Universität Karlsruhe
Hertzstraße 16, D-7500 Karlsruhe 1

Dr. Rainer Böhm
Institut für Parallele und Verteilte Höchstleistungsrechner
Universität Stuttgart
Herdweg 51, D-7000 Stuttgart 1

Dr. H. Borgwaldt
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik und Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Maurizio Bottoni
Kernforschungszentrum Karlsruhe GmbH
Institut für Reaktorentwicklung
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Inform. Gabriele Bürle

Dipl.-Phys. E. Diegele
Kernforschungszentrum Karlsruhe GmbH
Institut für Material- und Festkörperforschung IV
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Phys. Michael Dzierzawa
Institut für Theorie der Kondensierten Materie
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Prof. Dr. Eberle
Kernforschungszentrum Karlsruhe GmbH
Projektträgerschaft Wassertechnologie und Schlammbehandlung
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Ehret
Kernforschungszentrum Karlsruhe GmbH
Institut für Nukleare Festkörperphysik
Postfach 36 40, D-7500 Karlsruhe 1

Michael Eiermann
Institut für Praktische Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. Werner Eyrich
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik und Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Jörg Finger
SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Dipl.-Ing. Alfred Geiger
Rechenzentrum Universität Stuttgart
Allmandring 30, D-7000 Stuttgart 80

Dr. Thomas Gerz
Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V.
Institut für Physik der Atmosphäre, D-8031 Oberpfaffenhofen

Prof. Dr. W. Görke
Institut für Rechnerentwurf und Fehlertoleranz
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. Günther Grötzbach
Kernforschungszentrum Karlsruhe GmbH
Institut für Reaktorentwicklung
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Claus Günther
Kernforschungszentrum Karlsruhe GmbH
Institut für Reaktorbauelemente
Postfach 36 40, D-7500 Karlsruhe 1

Rolf Hammer
Institut für Angewandte Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dipl.-Inform. Volker Hatz
Institut für Algorithmen und Kognitive Systeme
Universität Karlsruhe
Haid-und-Neu-Straße 7, D-7500 Karlsruhe 1

Dipl.-Math. Willi Höbel
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik u. Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Math. Marlis Hochbruck
Institut für Praktische Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. W. Hohenhinnebusch
Kernforschungszentrum Karlsruhe GmbH
Vorstand
Postfach 36 40, D-7500 Karlsruhe 1

S. Honcu
Kernforschungszentrum Karlsruhe GmbH
Institut für Meteorologie und Klimaforschung
Postfach 36 40, D-7500 Karlsruhe 1

Prof. Dr. Friedel Hoßfeld
Kernforschungsanlage Jülich GmbH
Zentralinstitut für Angewandte Mathematik
Postfach 19 13, D-5170 Jülich

Dipl.-Math. Claus-Peter Hugelmann
Kernforschungszentrum Karlsruhe GmbH
Abteilung für Numerische Physik (HDI-3)
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Hubert Keller
Kernforschungszentrum Karlsruhe GmbH
Institut für Datenverarbeitung in der Technik
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Inf. (FH) Rolf Kerpe
Kernforschungszentrum Karlsruhe GmbH
Institut für Datenverarbeitung in der Technik
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Inf. Jürgen Kienhöfer
Institut für Telematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. Rudi Klatte
Institut für Angewandte Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Siegfried Kleinheins
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik u. Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Prof. Dr. W. Klose
Kernforschungszentrum Karlsruhe GmbH
Vorstand
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Ing. Andreas Knipschild
SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Dipl.-Inform. Brigitte Knödler-Kagoshima
Institut für Rechnerentwurf und Fehlertoleranz
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. Jörg Krone
SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Dipl.-Math. Klaus Küfner
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik u. Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Dr. M. Lötzerich
Dornier GmbH
Postfach 14 20, D-7990 Friedrichshafen

Dipl.-Math. Dipl.-Ing. Walter Maisel
Institut für Algorithmen und Kognitive Systeme
Universität Karlsruhe
Haid-und-Neu-Straße 7, D-7500 Karlsruhe 1

Prof. Dr. Hans W. Meuer
Rechenzentrum der Universität Mannheim
Postfach 10 34 62, D-6800 Mannheim 1

Dr. Falk Mikosch
Kernforschungszentrum Karlsruhe GmbH
Vorstand
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Roland Müller
SIEMENS AG, UB KWU, U6 514
Postfach 32 40, D-8520 Erlangen

Dr. Dieter Müller-Wichards
IBM Deutschland GmbH
Wissenschaftliches Zentrum
Tiergartenstraße 15, D-6900 Heidelberg

Dipl.-Meteorologe K. Nester
Kernforschungszentrum Karlsruhe GmbH
Institut für Meteorologie und Klimaforschung
Postfach 36 40, D-7500 Karlsruhe 1

Prof. Dr. W. Niethammer
Institut für Praktische Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr.-Ing. Klaus Peinze
Geschäftsführer der SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Dietmar Ratz
Institut für Angewandte Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dipl.-Phys. Thomas Rodach
Kernforschungszentrum Karlsruhe GmbH
Institut für Nukleare Festkörperphysik
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Wolfgang Rönsch
Prof. Feilmeier, Junker & Co.
Institut für Wirtschafts- und Versicherungsmathematik
Würmseestraße 35, D-8000 München 71

Dipl.-Math. Dieter Sanitz
Kernforschungszentrum Karlsruhe GmbH
Hauptabteilung Datenverarbeitung und Instrumentierung
Postfach 36 40, D-7500 Karlsruhe 1

Dr. U. Scharffenberger
IBM Deutschland GmbH
Wissenschaftliches Zentrum
Tiergartenstraße 15, D-6900 Heidelberg

Prof. Dr. Winfried Schmidt
Kernforschungszentrum Karlsruhe GmbH
Abteilung für Numerische Physik (HDI-3)
Postfach 36 40, D-7500 Karlsruhe 1

Frank Schmitz
Kernforschungszentrum Karlsruhe GmbH
Hauptabteilung Datenverarbeitung und Instrumentierung
Postfach 36 40, D-7500 Karlsruhe 1

Prof. Dr.-Ing. Willi Schönauer
Rechenzentrum der Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dr. Günter Schumacher
Institut für Angewandte Mathematik
Universität Karlsruhe
Postfach 69 80, D-7500 Karlsruhe 1

Dipl.-Math. David Seldner
Kernforschungszentrum Karlsruhe GmbH
Institut für Datenverarbeitung in der Technik
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Math. Karl Solchenbach
SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Dipl.-Math. Burghard Stehle
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik u. Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Phys. Eckhard Stein
Kernforschungszentrum Karlsruhe GmbH
Institut für Neutronenphysik und Reaktortechnik
Postfach 36 40, D-7500 Karlsruhe 1

Dipl.-Phys. Hans Stittgen
Kernforschungszentrum Karlsruhe GmbH
Hauptabteilung Datenverarbeitung und Instrumentierung
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Klaus Stüben
Gesellschaft für Mathematik und Datenverarbeitung mbH, F1/T
Postfach 12 40, D-5205 St. Augustin 1

Dipl.-Math. Clemens-August Thole
SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Dr. Georgra Thurner
ipSystems KG Wöst
Steinhäuserstraße 20, D-7500 Karlsruhe 1

Dr. Jens Timm
SIEMENS AG
Zweigniederlassung Mannheim, Regionalvertrieb Vektorprozessoren
Postfach 10 28 62, D-6800 Mannheim 1

Prof. Dr.-Ing. Heinz Trauboth
Kernforschungszentrum Karlsruhe GmbH
Institut für Datenverarbeitung in der Technik
Postfach 36 40, D-7500 Karlsruhe 1

Prof. Dr. Ulrich Trottenberg
Geschäftsführer der SUPRENUM GmbH
Hohe Straße 73, D-5300 Bonn 1

Peter Vogel
Rechenzentrum der Universität Mannheim
Postfach 10 34 62, D-6800 Mannheim 1

Dirk Wenzel
Rechenzentrum der Universität Mannheim
Postfach 10 34 62, D-6800 Mannheim 1

Dr. Heinz Wenzelburger
Kernforschungszentrum Karlsruhe GmbH
Abteilung für Numerische Physik (HDI-3)
Postfach 36 40, D-7500 Karlsruhe 1

Dr. Thomas Westermann
Kernforschungszentrum Karlsruhe GmbH
Abteilung für Numerische Physik (HDI-3)
Postfach 36 40, D-7500 Karlsruhe 1

Dr. G. Wilhelmi
Kernforschungszentrum Karlsruhe GmbH
Hauptabteilung Datenverarbeitung und Instrumentierung
Postfach 36 40, D-7500 Karlsruhe 1

Dr. B. Wolters
Kernforschungszentrum Karlsruhe GmbH
Institut für Nukleare Entsorgungstechnik
Postfach 36 40, D-7500 Karlsruhe 1