Lecture Notes in Computational Science and Engineering

This book presents major advances in high performance computing as well as major advances due to high performance computing. It contains a collection of papers in which results achieved in the collaboration of scientists from computer science, mathematics, physics, and mechanical engineering are presented. From the science problems to the mathematical algorithms and on to the effective implementation of these algorithms on massively parallel and cluster computers we present state-of-the-art methods and technology as well as exemplary results in these fields. This book shows that problems which seem superficially distinct become intimately connected on a computational level.

ISSN 1439-7358
ISBN: 3-540-33539-0
springer.com

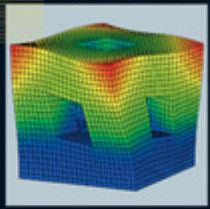
52 LNCSE

K. H. Hoffmann
A. Meyer (Eds.)

Parallel Algorithms and Cluster Computing

Lecture Notes in Computational Science and Engineering

52



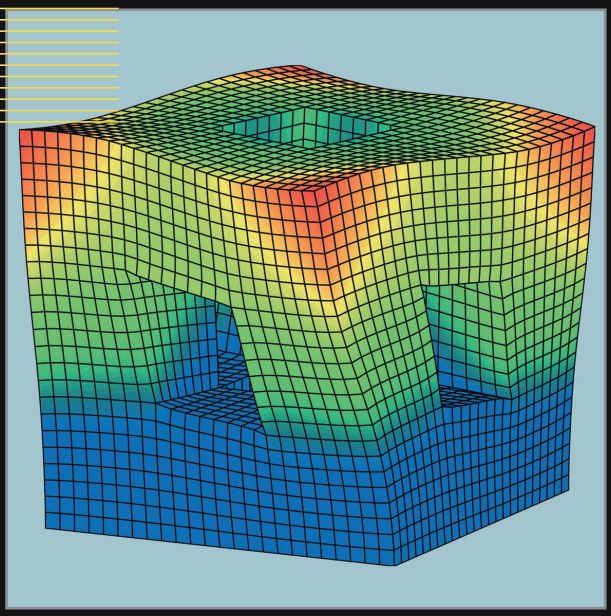
Karl Heinz Hoffmann
Arnd Meyer
Editors

Parallel Algorithms and Cluster Computing

Implementations, Algorithms and Applications

Springer

Lecture Notes in Computational Science and Engineering



Karl Heinz Hoffmann
Arnd Meyer
Editors

Parallel Algorithms and Cluster Computing

Implementations, Algorithms and Applications

Lecture Notes
in Computational Science
and Engineering

Karl Heinz Hoffmann Arnd Meyer (Eds.)

Implementations, Algorithms and Applications

With 187 Figures and 18 Tables

Editors

Karl Heinz Hoffmann
Institute of Physics – Computational Physics
Chemnitz University of Technology
09107 Chemnitz, Germany
email: hoffmann@physik.tu-chemnitz.de

Arnd Meyer
Faculty of Mathematics – Numerical Analysis
Chemnitz University of Technology
09107 Chemnitz, Germany
email: a.meyer@mathematik.tu-chemnitz.de

Library of Congress Control Number: 2006926211

Mathematics Subject Classification: I17001, I21025, I23001, M13003, M1400X, M27004, P19005, S14001

ISBN-10 3-540-33539-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-33539-9 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media
springer.com

Printed in The Netherlands

Typesetting: by the authors and techbooks using a Springer LaTeX macro package
Cover design: *design & production* GmbH, Heidelberg

Printed on acid-free paper SPIN: 11739067 46/techbooks 5 4 3 2 1 0

Acknowledgement

The editors and authors of this book worked together in the SFB 393 "Parallele Numerische Simulation für Physik und Kontinuumsmechanik" over a period of 10 years. They gratefully acknowledge the continued support from the German Science Foundation (DFG) which provided the basis for the intensive collaboration in this group as well as the funding of a large number of young researchers.

Preface

High performance computing has changed the way in which science progresses. During the last 20 years the increase in computing power, the development of effective algorithms, and the application of these tools in the area of physics and engineering has been decisive in the advancement of our technological world. These abilities have allowed to treat problems with a complexity which had been out of reach for analytical approaches. While the increase in performance of single processes has been immense the increase of massive parallel computing as well as the advent of cluster computers has opened up the possibilities to study realistic systems. This book presents major advances in high performance computing as well as major advances due to high performance computing. The progress made during the last decade rests on the achievements in three distinct science areas.

Open and pressing problems in physics and mechanical engineering are the driving force behind the development of new tools and new approaches in these science areas. The treatment of complex physical systems with frustration and disorder, the analysis of the elastic and non-elastic movement of solids as well as the analysis of coupled fluid systems, pose problems which are open to a numerical analysis only with state of the art computing power and algorithms. The desire of scientific accuracy and quantitative precision leads to an enormous demand in computing power. Asking the right questions in these areas lead to new insights which have not been available due to other means like experimental measurements.

The second area which is decisive for effective high performance computing is a realm of effective algorithms. Using the right mathematical approach to the solution of a science problem posed in the form of a mathematical model is as crucial as asking the proper science question. For instance in the area of fluid dynamics or mechanical engineering the appropriate approach by finite element methods has led to new developments like adaptive methods or wavelet techniques for boundary elements.

The third pillar on which high performance computing rests is computer science. Having asked the proper physics question and having developed an

appropriate effective mathematical algorithm for its solution it is the implementation of that algorithm in an effective parallel fashion on appropriate hardware which then leads to the desired solutions. Effective parallel algorithms are the central key to achieving the necessary numerical performance which is needed to deal with the scientific questions asked. The adaptive load balancing which makes optimal use of the available hardware as well as the development of effective data transfer protocols and mechanisms have been developed and optimized.

This book gives a collection of papers in which the results achieved in the collaboration of colleagues from the three fields are presented. The collaboration took place within the Sonderforschungsbereich SFB 393 at the Chemnitz University of Technology. From the science problems to the mathematical algorithms and on to the effective implementation of these algorithms on massively parallel and cluster computers we present state of the art technology. We highlight the connections between the fields and different work packages which let to the results presented in the science papers.

Our presentation starts with the Implementation section. We begin with a view on the implementation characteristics of highly parallelized programs, go on to specifics of FEM and quantum mechanical codes and then turn to some general aspects of postprocessing, which is usually needed to analyse the obtained data further.

The second section is devoted to Algorithms. The main focus is on FEM algorithms, starting with a discussion on efficient preconditioners. Then the focus is on a central aspect of FEM codes, the aspect ratio, and on problems and solutions to non-matching meshes at domain boundaries. The Algorithm section ends with discussing adaptive FEM methods in the context of elastoplastic deformations and a view on wavelet methods for boundary value problems.

The Applications section starts with a focus on disordered systems, discussing phase transitions in classical as well as in quantum systems. We then turn to the realm of atomic organization for amorphous carbons and for heterophase interphases in Titanium-Silicon systems. Methods used in classical as well as in quantum mechanical systems are presented.We finish by a glance on fluid dynamics applications presenting an analysis of Lyapunov instabilities for Lenard-Jones fluids.

While the topics presented cover a wide range the common background is the need for and the progress made in high performance parallel and cluster computing.

Chemnitz
March 2006

Karl Heinz Hoffmann
Arnd Meyer

Contents

Part I Implementions

Parallel Programming Models for Irregular Algorithms
Gudula Rünger 3

Basic Approach to Parallel Finite Element Computations: The DD Data Splitting
Arnd Meyer 25

A Performance Analysis of ABINIT on a Cluster System
Torsten Hoefler, Rebecca Janisch, Wolfgang Rehm 37

Some Aspects of Parallel Postprocessing for Numerical Simulation
Matthias Pester 53

Part II Algorithms

Efficient Preconditioners for Special Situations in Finite Element Computations
Arnd Meyer 67

Nitsche Finite Element Method for Elliptic Problems with Complicated Data
Bernd Heinrich, Kornelia Pönitz 87

Hierarchical Adaptive FEM at Finite Elastoplastic Deformations
Reiner Kreißig, Anke Bucher, Uwe-Jens Görke 105

Wavelet Matrix Compression for Boundary Integral Equations
Helmut Harbrecht, Ulf Kähler, Reinhold Schneider 129

Numerical Solution of Optimal Control Problems for Parabolic Systems
Peter Benner, Sabine Görner, Jens Saak 151

Part III Applications

Parallel Simulations of Phase Transitions in Disordered Many-Particle Systems
Thomas Vojta 173

Localization of Electronic States in Amorphous Materials: Recursive Green's Function Method and the Metal-Insulator Transition at $E \neq 0$
Alexander Croy, Rudolf A. Römer, Michael Schreiber 203

Optimizing Simulated Annealing Schedules for Amorphous Carbons
Peter Blaudeck, Karl Heinz Hoffmann 227

Amorphisation at Heterophase Interfaces
Sibylle Gemming, Andrey Enyashin, Michael Schreiber 235

Energy-Level and Wave-Function Statistics in the Anderson Model of Localization
Bernhard Mehlig, Michael Schreiber 255

Fine Structure of the Integrated Density of States for Bernoulli–Anderson Models
Peter Karmann, Rudolf A. Römer, Michael Schreiber, Peter Stollmann 267

Modelling Aging Experiments in Spin Glasses
Karl Heinz Hoffmann, Andreas Fischer, Sven Schubert, Thomas Streibert 281

Random Walks on Fractals
Astrid Franz, Christian Schulzky, Do Hoang Ngoc Anh, Steffen Seeger, Janett Balg, Karl Heinz Hoffmann 303

Lyapunov Instabilities of Extended Systems
Hong-liu Yang, Günter Radons 315

The Cumulant Method for Gas Dynamics
Steffen Seeger, Karl Heinz Hoffmann, Arnd Meyer 335

Index 361

Part I

Implementions

Parallel Programming Models for Irregular Algorithms

Gudula Rünger

Technische Universität Chemnitz, Fakultät für Informatik
09107 Chemnitz, Germany
ruenger@informatik.tu-chemnitz.de

Applications from science and engineering disciplines make extensive use of computer simulations and the steady increase in size and detail leads to growing computational costs. Computational resources can be provided by modern parallel hardware platforms which nowadays are usually cluster systems. Effective exploitation of cluster systems requires load balancing and locality of reference in order to avoid extensive communication. But new sophisticated modeling techniques lead to application algorithms with varying computational effort in space and time, which may be input dependent or may evolve with the computation itself. Such applications are called irregular. Because of the characteristics of irregular algorithms, efficient parallel implementations are difficult to achieve since the distribution of work and data cannot be determined a priori. However, suitable parallel programming models and libraries for structuring, scheduling, load balancing, coordination, and communication can support the design of efficient and scalable parallel implementations.

1 Challenges for parallel irregular algorithms

Important issues for gaining efficient and scalable parallel programs are load balancing and communication. On parallel platforms with distributed memory and clusters, load balancing means spreading the calculations evenly across processors while minimizing communication. For algorithms with regular computational load known at compile time, load balancing can be achieved by suitable data distributions or mappings of task to processors. For irregular algorithms, static load balancing becomes more difficult because of dynamically changing computation load and data load.

The appropriate load balancing technique for regular and irregular algorithms depends on the specific algorithmic properties concerning the behavior of data and task:

- The algorithmic structure can be data oriented or task oriented. Accordingly, load balancing affects the distribution of data or the distribution of tasks.
- Input data of an algorithm can be regular or more irregular, like sparse matrices. For regular and some irregular input data, a suitable data distribution can be selected statically before runtime.
- Regular as well as irregular data structures can be static or can be dynamically growing and shrinking during runtime. Depending on the knowledge before runtime, suitable data distributions and dynamic redistributions are used to gain load balance.
- The computational effort of an algorithm can be static, input dependent or dynamically varying. For a static or input dependent computational load, the distribution of tasks can be planned in advance. For dynamically varying problems a migration of tasks might be required to achieve load balancing.

The communication behavior of a parallel program depends on the characteristics of the algorithm and the parallel implementation strategy but is also intertwined with the load balancing techniques. An important issue is the locality of data dependencies in an algorithm and the resulting communication pattern due to the distribution of data.

- Locality of data dependencies: In the algorithm, data structures are chosen according to the algorithmic needs. They may have local dependencies, e.g. to neighboring cells in a mesh, or they may have global dependencies to completely different parts of the same or other data structures. Both local and global data dependencies can be static, input dependent or dynamically changing.
- Locality of data references: For the parallel implementation of an algorithm, aggregate data structures, like arrays, meshes, trees or graphs, are usually distributed according to a data distribution which maps different parts of the data structure to different processors. Data dependencies between data on the same processor result in local data references. Data dependencies between data mapped to different processors cause remote data reference which requires communication. The same applies to task oriented algorithms where a distribution of tasks leads to remote references by the tasks to data in remote memory.
- Locality of communication pattern: Depending on the locality of data dependencies and the data distribution, locality of communication pattern occurs. Local data dependencies usually lead either to local data references or to remote data references which can be realized by communication with neighboring processors. This is often called locality of communication. Global data dependencies usually result in more complicated remote access and communication patterns.

Communication is also caused by load balancing when redistributing data or migrating tasks to other processors. Also, the newly created distribution

of data or tasks create a new pattern of local and remote data references and thus cause new communication patterns after a load balancing step. Although the specific communication may change after redistribution, the locality of the communication pattern is often similar.

The static planning of load balance during the coding phase is difficult for irregular applications and there is a need for flexible, robust, and effective programming support. Parallel programming models and environments address the question how to express irregular applications and how to execute the application in parallel. It is also important to know what the best performance can be and how it can be obtained. The requirement of scalability is essential, i.e. the ability to perform efficiently the same code for larger applications on larger cluster systems. Another important aspect is the type of communication. Specific communication needs, like asynchronous or varying communication demands, have to be addressed by a programming environment and correctness as well as efficiency are crucial.

Due to diverse application characteristics not all irregular applications are best treated by the same parallel programming support. In the following, several programming models and environments are presented:

- Task pool programming for hierarchical algorithms,
- Data and communication management for adaptive algorithms,
- Library support for mixed task and data parallel algorithms,
- Communication optimization for structured algorithms.

The programming models range from task to data oriented modes for expressing the algorithm and from self-organizing task pool approaches to more data oriented flexible adaptive modes of execution.

2 Task pool programming for hierarchical algorithms

The programming model of *task pools* supports the parallel implementation of task oriented algorithms and is suitable for hierarchical algorithms with dynamically varying computational work and complex data dependencies.

The main concept is a decomposition of the computational work into tasks and a task pool which stores the tasks ready for execution. Processes or threads are responsible for the execution of tasks. They extract tasks from the task pool for execution and create new tasks which are inserted into the task pool for a later computation, possibly by another process or thread. Complex data dependencies between tasks are allowed and may lead to complex interaction between the tasks, forming a virtual task graph. Usually, task pools are provided as programming library for shared memory platforms. Library routines for the creation, insertion, and extraction of tasks are available. A fixed number of processes or threads is created at program start to execute an arbitrary number of tasks with arbitrary dependence structures.

Load balancing and mapping of tasks is achieved automatically since a process extracts a task whenever processor time is available. There are several possibilities for the internal realization of task pools, which affect load balancing. Often the tasks are kept in task queues, see also Fig. 1:

- Central task pools: All tasks of the algorithm are kept in one task queue from which all threads extract tasks for execution and into which the newly created tasks are inserted. Access conflicts are avoided by a lock mechanism for shared memory programming.
- Decentralized task pools: Each thread has its own task queue from which the thread extracts tasks and into which it inserts newly created tasks. No access conflicts can occur and so there is no need for a lock mechanism. But load imbalances can occur for irregularly growing computational work.
- Decentralized task pools with task stealing: This variant of the decentralized task pool offers a task stealing mechanism. Threads with an empty task queue can steal tasks from other queues. Load imbalance is avoided but task stealing needs a locking mechanism for correct functionality.

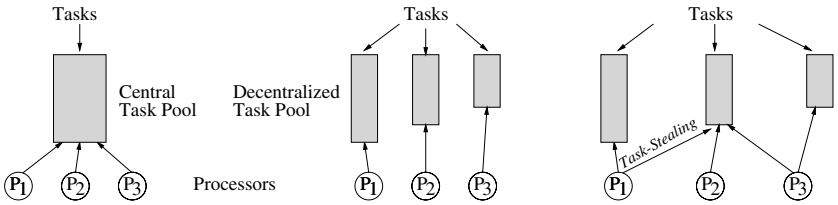


Fig. 1. Different types of task pool variants for shared memory

Due to the additional overhead of task pools it is suggested to use them only when required for highly irregular and dynamic algorithms. Examples are the hierarchical radiosity method from computer graphics and hierarchical n-body algorithms.

The hierarchical radiosity method

The radiosity algorithm is an observer-independent global illumination method from computer graphics to simulate diffuse light in three-dimensional scenes [10]. The method is based on the energy radiation between surfaces of objects and accounts for direct illumination and multiple reflections between surfaces within the environment. The radiosity method decomposes the surface of objects in the scene into small elements A_j, $j = 1, \ldots, n$, with almost constant radiation energy. For each element, the radiation energy is represented by a *radiosity value* B_j (of dimension [Watt/m^2]) describing the radiant energy per unit time and per unit area dA_j of A_j. The radiosity values of the elements