

Lecture Notes in Computer Science

1220

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Peter Brezany

Input/Output Intensive Massively Parallel Computing

Language Support,
Automatic Parallelization,
Advanced Optimization,
and Runtime Systems



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany

Juris Hartmanis, Cornell University, NY, USA

Jan van Leeuwen, Utrecht University, The Netherlands

Author

Peter Brezany

Institute for Software Technology and Parallel Systems

University of Vienna

Liechtensteinstrasse 22, A-1090 Vienna, Austria

E-mail: brezany@par.univie.ac.at

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Brezany, Peter:

Input output intensive massively parallel computing : language support, automatic parallelization, advanced optimization, and runtime systems / Peter Brezany. - Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ; London ; Milan ; Paris ; Santa Clara ; Singapore ; Tokyo : Springer, 1997

(Lecture notes in computer science ; Vol. 1220)

ISBN 3-540-62840-1 kart.

CR Subject Classification (1991): D.3, D.1.3, D.4.2, C.1.2, F.1.2, B.3.2, D.4.4, G.1

ISSN 0302-9743

ISBN 3-540-62840-1 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1997
Printed in Germany

Typesetting: Camera-ready by author
SPIN 10549551 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

*To my wife Jarmila, children Zuzana and Jozef, my
mother Zuzana, and in memory of my father Jozef*

Preface

Massively parallel processing is currently the most promising answer to the quest for increased computer performance. This has resulted in the development of new programming languages, and programming environments and has significantly contributed to the design and production of powerful massively parallel supercomputers that are currently based mostly on the distributed-memory architecture.

Traditionally, developments in high-performance computing have been motivated by applications in which the need for high computational power clearly dominated the requirements put on the input/output performance. However, the most significant forces driving the development of high-performance computing are emerging applications that require a supercomputer to be able to process large amounts of data in sophisticated ways. Hardware vendors responded to these new application requirements by developing highly parallel massive storage systems.

However, after a decade of intensive study, the effective exploitation of massive parallelism in computation and input/output is still a very difficult problem. Most of the difficulties seem to lie in programming existing and emerging complex applications for execution on a parallel machine.

The efficiency of concurrent programs depends critically on the proper utilization of specific architectural features of the underlying hardware, which makes automatic support of the program development process highly desirable. Work in the field of programming environments for supercomputers spans several areas, including the design of new programming languages and the development of runtime systems that support execution of parallel codes and supercompilers that transform codes written in a sequential programming language into equivalent parallel codes that can be efficiently executed on a target machine. The focus of this book is just in these areas; it concentrates on the automatic parallelization of numerical programs for large-scale input/output intensive scientific and engineering applications. The principles and methods that are presented in the book are oriented towards typical distributed-memory architectures and their input/output systems.

The book addresses primarily researchers and system developers working in the field of programming environments for parallel computers. The book will also be of great value to advanced application programmers wishing to gain insight into the state of the art in parallel programming.

Since Fortran plays a dominant role in the world of high-performance programming of scientific and engineering applications, it has been chosen as the basis for the presentation of the material in the text.

For full understanding of the contents of the book it is necessary that the reader has a working knowledge of Fortran or a similar procedural high-level programming language and a basic knowledge of machine architectures and compilers for sequential machines.

The book's development In writing this book, I utilized the results of my work achieved during research and development in the European Strategic Program for Research and Development in Information Technology (ESPRIT), in particular, ESPRIT Projects GENESIS, PPPE, and PREPARE. Most of the methods and techniques presented in the book have been successfully verified by a prototype or product implementation or are being applied in on-going projects. Topics related to parallel input/output have been the basis for the proposals of new research projects that start at the University of Vienna this year.

The material of the book has been covered in courses at the University of Vienna given to students of computer science and in the Advanced Course on Languages, Compilers, and Programming Environments given to advanced developers of parallel software.

Contents of the book Each chapter begins with an overview of the material covered and introduces its main topics with the aim of providing an overview of the subject matter. The concluding section typically points out problems and alternative approaches, discusses related work, and gives references. This scheme is not applied if a chapter includes extensive sections. In this case, each section is concluded by a discussion of related work. Some sections present experimental results from template codes taken from real applications, to demonstrate the efficiency of the techniques presented.

Chapter 1 provides motivation, a brief survey of the state of the art in programming distributed-memory systems, and lists the main topics addressed in the book. Input/Output requirements of the current Great Challenge applications are illustrated in three examples which are both I/O and computational intensive: earthquake ground motion modeling, analysis of data collected by the Magellan spacecraft, and modeling atmosphere and oceans.

Chapter 2 specifies a new parallel machine model that reflects the technology trends underlying current massively parallel architectures. Using this machine model, the chapter further classifies the main models used for programming distributed-memory systems and discusses the programming style associated with each model.

The core of the book consists of chapters 3–7. While the first chapter deals with programming language support, the subsequent three chapters show how programs are actually transformed into parallel form and specify requirements on the runtime system. Chapter 7 develops new concepts for an advanced runtime support for massively parallel I/O operations.

Chapter 3 describes Vienna Fortran 90, a high-performance data-parallel language that provides advanced support both for distributed computing and the operations on files stored in massively parallel storage systems. In this chapter the presentation is mainly focused on the language extensions concerning parallel I/O.

Chapter 4 first describes the principal tasks of automatic parallelization of regular and irregular in-core programs and then addresses several important optimization issues. In-core programs are able to store all their data in main memory.

Chapter 5 deals with basic compilation and optimizations of explicit parallel I/O operations inserted into the program by the application programmer.

Chapter 6 treats the problem of transforming regular and irregular out-of-core Vienna Fortran 90 programs into out-of-core message-passing form. Out-of-core programs operate on significantly more data (large data arrays) that can be held in main memory. Hence, parts of data need to be swapped to disks.

Compilation principles and methods are presented in chapters 5 and 6 in the context of the *VFCS (Vienna Fortran Compilation System)*.

Chapter 7 proposes an advanced runtime system referred to as *VIPIOS (Vienna Parallel Input/Output System)* which is based on concepts developed in data engineering technology.

Chapter 8 (conclusion) presents some ideas about the future development of programming environments for parallel computer systems.

Acknowledgements Many people deserve thanks for their assistance, encouragement, and advice. In particular, I thank Hans Zima for giving me the opportunity to work on the ambitious research and development projects and for his encouragement. I am grateful to Michael Gerndt (KFA Jülich), Erich Schikuta, and Thomas Mück (both from the Department for Data Engineering of the University of Vienna), Joel Saltz (University of Maryland), and Alok Choudhary (Northwestern University, Evanston) for many stimulating discussions and suggestions, and to Viera Sipkova for the implementation of several parallelization methods introduced in this book within the Vienna Fortran Compilation System. Thanks also go to Minh Dang and Ka Heng Wong for carrying out many performance measurements to demonstrate the efficiency of the techniques presented in the book.

It is also a pleasure to thank colleagues from GMD Bonn, GMD FIRST Berlin, GMD Karlsruhe, Parsytec Aachen, TNO Delft, ACE Amsterdam, TU Munich, INRIA Rennes, Steria Paris, University of Liverpool, and University of Southampton with whom I have cooperated in the ESPRIT projects GENESIS, PREPARE, and PPPE.

Table of Contents

Preface	VII
1. Introduction	1
1.1 I/O Requirements of Parallel Scientific Applications	2
1.1.1 Earthquake Ground Motion Modeling	5
1.1.2 Image Analysis of Planetary Data	5
1.1.3 Climate Prediction	6
1.1.4 Summary	6
1.2 State-of-the-Art and Statement of the Problems	6
1.3 The Central Problem Areas Addressed in this Book	11
2. Parallel Programming Models	13
2.1 The Parallel Machine Model	13
2.1.1 IMPA Model	14
2.1.2 Matching the IMPA to Real Architectures	16
2.2 Programming Models for Distributed-Memory Systems	20
2.2.1 Message-Passing Programming Model	21
2.2.2 Data Parallel Programming Model	22
2.2.3 Models Based on Coordination Mechanisms	24
Bibliographical Notes	24
3. Vienna Fortran 90 and Its Extensions for Parallel I/O	25
3.1 Language Model	25
3.1.1 Index Domain and Index Mapping	26
3.1.2 Index Domain of an Array Section	27
3.1.3 Data Distribution Model for Internal Memory	27
3.2 Language Constructs	29
3.2.1 Distribution Annotations	29
3.2.2 Alignment Annotations	32
3.2.3 Dynamic Distributions and the DISTRIBUTE State- ment	32
3.2.4 Indirect Distribution	33
3.2.5 Procedures	35
3.2.6 FORALL Loop	36

3.2.7	Reduction Operators	37
3.2.8	User-Defined Distribution Functions	38
3.2.9	Support for Sparse Matrix Computation	38
3.3	Controlling Parallel I/O Operations	42
3.3.1	The File Processing Model	43
3.3.2	User's Perspective	45
3.3.3	Data Distribution Model for External Memory	46
3.3.4	Opening a Parallel File	49
3.3.5	Write and Read Operations on Parallel Files	58
3.3.6	I/O Alignment	60
3.3.7	Other I/O Statements	60
3.3.8	Intrinsic Procedures	60
3.3.9	Experiments	61
3.4	Out-of-Core Annotation	68
3.4.1	User Controlled Mode	69
3.4.2	Automatic Mode	72
	Bibliographical Notes	73
4.	Compiling In-Core Programs	75
4.1	Parallelizing Compilation Systems	75
4.2	Parallelization Strategies for Regular Codes	79
4.2.1	Parallelization of Loops	80
4.2.2	Adaptation of Array Statements	91
	Bibliographical Notes	98
4.3	Parallelization of Irregular Codes	100
4.3.1	Irregular Code Example: Computation on an Unstruc- tured Mesh	101
4.3.2	Programming Environment for Processing Irregular Codes	103
4.3.3	Working Example	106
4.3.4	Distribution Descriptors	107
4.3.5	Physical Data Redistribution	110
4.3.6	Work Distributor, Inspector and Executor	110
4.3.7	Handling Arrays with Multi-Dimensional Distribu- tions and General Accesses	116
4.3.8	Runtime Support	120
4.3.9	Optimizations	120
4.3.10	Performance Results	123
	Bibliographical Notes	124
4.4	Coupling Parallel Data and Work Partitioners to the Compiler	125
4.4.1	Towards Parallel Data and Work Partitioning	128
4.4.2	Partitioners of the CHAOS Runtime Library	130
4.4.3	High Level Language Interface	131
4.4.4	Interactive Specification of the Partitioning Strategy	133
4.4.5	Implementation Issues	133

4.4.6	Illustration of the Approach	135
4.4.7	Performance Results	140
	Bibliographical Notes	142
4.5	Compile Time Optimizations for Irregular Codes	142
4.5.1	Introduction	142
4.5.2	Partial Redundancy Elimination	144
4.5.3	Specifiers of the Candidates for Placement	147
4.5.4	Influencers of the Candidate for Placement	149
4.5.5	The Form of Specifiers	150
4.5.6	Intraprocedural Optimizations	153
4.5.7	Interprocedural Optimizations	163
	Bibliographical Notes	167
5.	Compiling Parallel I/O Operations	169
5.1	Direct I/O Compilation	170
5.1.1	OPEN Statement	170
5.1.2	WRITE Statement	172
5.1.3	READ Statement	174
5.2	Optimizing I/O Compilation	174
5.2.1	I/O Communication Descriptors	176
5.2.2	Automatic Determination of I/O Distribution Hints ..	177
5.2.3	Overlapping I/O with Computation	178
5.2.4	Running I/O Concurrently with Computation and Other I/O	178
	Bibliographical Notes	181
6.	Compiling Out-of-Core Programs	183
6.1	Models for Out-of-Core Computations	184
6.1.1	Local Placement Model	186
6.1.2	Global Placement Model	187
6.1.3	Partitioned In-core Model	189
6.1.4	Model Based on a Parallel Array Database	190
6.2	Compilation Strategy	191
6.3	Specification of the Application Area	192
6.4	Compiling Out-of-Core Regular Codes	193
6.4.1	Out-of-Core Restructuring Techniques Applied to Loosely Synchronous Computations	193
6.4.2	Out-of-Core Restructuring Techniques Applied to Pipelined Computations	197
6.4.3	Optimizations for Out-of-Core Regular Codes	197
6.5	Parallelization Methods for Out-of-Core Irregular Problems ..	202
6.5.1	Problem Description and Assumptions	203
6.5.2	Data Arrays are In-Core and Indirection/Interaction Arrays Out-of-Core	205
6.5.3	Experiments	214

6.5.4	Generalization of Irregular Out-of-Core Problems	217
6.6	Support for I/O Oriented Software-Controlled Prefetching . . .	226
6.6.1	Compiler-Directed Data Prefetching	227
6.6.2	Data Prefetching Based on Prefetch Adaptivity	229
	Bibliographical Notes	233
7.	Runtime System for Parallel Input-Output	235
7.1	Coupling the Runtime System to the VFCS	235
7.2	Data Mapping Model	237
7.2.1	Mapping Data Arrays to Computing Processors	238
7.2.2	Logical Storage Model	238
7.2.3	Physical Storage Model	239
7.2.4	Example of the Model Interpretation	240
7.3	Design of the VIPIOS	241
7.3.1	Design Objectives	241
7.3.2	Process Model	242
7.4	Data Locality	245
7.4.1	Logical Data Locality	245
7.4.2	Physical Data Locality	245
7.5	Two-Phase Data Administration Process	245
7.5.1	Preparation Phase	246
7.5.2	Administration Phase	246
7.6	Basic Execution Profile	246
7.7	VIPIOS Servers - Structure and Communication Scheme . . .	250
7.7.1	Handling I/O Requests - Control Flow and Data Flow in the VIPIOS	252
7.7.2	Implementation Notes	254
	Bibliographical Notes	254
8.	A New Generation Programming Environment for Parallel Architectures	257
8.1	Overview	257
8.2	Compilation and Runtime Technology	258
8.3	Debugging System	260
8.3.1	Motivation	260
8.3.2	Design of a High-Level Symbolic Debugger	261
	Bibliographical Notes	266
	References	267
	Subject Index	283