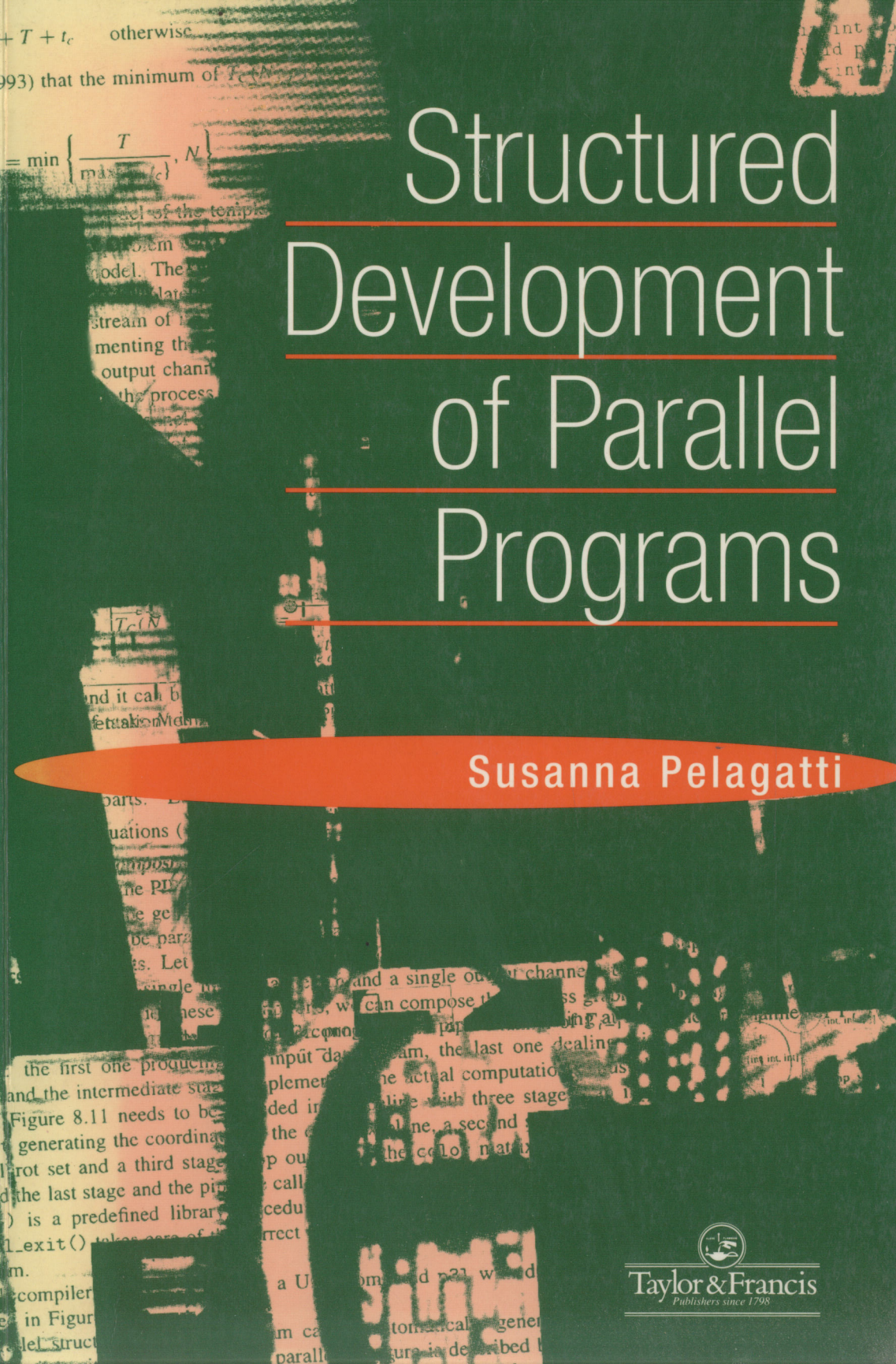

Structured Development of Parallel Programs

Susanna Pelagatti

Taylor & Francis
Publishers since 1798

Structured Development of Parallel Programs

SUSANNA PELAGATTI
University of Pisa, Italy

Taylor & Francis
Publishers since 1798

UK Taylor & Francis Ltd, 1 Gunpowder Square, London EC4A 3DE
USA Taylor & Francis Inc., 1900 Frost Road, Suite 101, Bristol, PA 19007

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library
ISBN 0-7484-0655-7

Library of Congress Cataloging-in-Publication Data are available

Cover design by Youngs Design in Production
Typeset in Times 10/12pt by Focal Image Ltd, London, UK
Printed by T.J. International Ltd, Padstow, UK

Structured Development of Parallel Programs

To Maria Teresa and Pier Luigi

Foreword

Structure and abstraction are the essence of good computer science. Parallel computing has the pursuit of absolute performance as its *raison d'etre*. In what are still, from any sensible perspective, the early days of parallelism, there has been an understandable tendency to ignore issues of higher-level principle, particularly in the area of programming model and language design, in the rush for the improved speed-up curve. This book is representative of a research area of growing interest which seeks to impose structural discipline on the parallel programmer's task, offering in return conceptual abstraction and its concomitant simplicity, portability and predictability, while remaining sensitive to the need for good performance.

The key observation is that real parallel programs are rarely random collections of processes interacting in unpredictable ways, but are in fact well structured in concept (if not in concrete presentation) and adhere to a small collection of more or less regular operational patterns. The challenge is then to embed our understanding of these patterns into the design of programming systems, both at the level of language constructs and in the implementation process.

The book is not simply a research monograph, destined for the library shelves or the desks of a few closely related researchers. Care has been taken to place the ideas within the wider field of parallel computing, making it an appropriate vehicle through which to introduce the subject to aspiring postgraduates and even appropriately focused senior undergraduates. The opening chapters are a properly contextualised manifesto for the structured approach and one might go so far as to suggest that this is the perspective from which parallelism ought to be taught in general. Certainly, there is plenty of food for thought here for anyone who is serious about the development of parallel programming as a well-founded discipline.

It is apt that such a book should emanate from Pisa. Susanna and her colleagues are eminent in the field and their efforts in addressing underlying issues, building concrete implementations and introducing the methodology to industrial partners are praiseworthy.

It is a pleasure to be able to offer these few words of introduction to what should become recognised as one of the foundational texts in the area.

MURRAY COLE
Edinburgh

Preface

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to", said the Cat.
"I don't much care where—" said Alice.
"Then it doesn't matter which way you go," said the Cat.
"—so long as I get somewhere*," Alice added as an explanation.*
"Oh, you're sure to do that," said the Cat, "if you only walk long enough." L. CARROLL

Parallelism has always been considered important, but in the past decade interest in it has grown enormously. The reason for this growth is that parallel and distributed architectures have become readily available as commercial products and parallelism promises to allow the solution of challenging frontier problems. However, from the very beginning the main limit on parallelism has been the ability to write parallel programs at a reasonable cost. Coordinating and managing execution of parallel tasks is too difficult to be left as the responsibility of the programmer. If parallelism is to have an impact in the real world, parallel machines need to be programmed using high-level languages and most of the parallel machine complexity must be implicitly dealt with by compilers.

There is currently a wide variety of ways in which parallel software is developed, but there is no widely accepted software development methodology able to free programmers from parallelism complexity and at the same time able to achieve high degrees of performance on different architectures.

The key problem when developing such a methodology is the intrinsic difficulty of the problems to be solved automatically by the language compiler/support. Data decomposition, process mapping, task scheduling and grain optimization have a tremendous impact on the actual performance achieved on a parallel architecture. Moreover, the performance achieved by different solutions to these problems needs to be predicted to take sensible optimization decisions in the compiling process. Unfortunately, for general parallel program structures the decomposition mapping and scheduling problems are intractable and such that the performance of a given solution cannot be estimated accurately.

The purpose of this book is to show how notoriously difficult problems such as mapping/scheduling can be made tractable by restricting the structure of parallel programs and to propose a methodology for the design of parallel software which is high-level, portable and able to achieve high performance figures.

The methodology uses a small number of parallel forms as building blocks for par-

allel applications and allows the coding of parallel programs in a high-level machine-independent notation which can be automatically reorganized by the compiler. The parallel forms are provided as primitive constructs of a structured parallel language and are the only way in which the application parallel structure can be expressed. The class of applications which can be coded in this way is extremely large, and includes many data parallel and task parallel examples, or applications exploiting a mixture of the two. The restriction imposed by the language allows an innovative organization of the compiler to be adopted, which results in a number of benefits. In particular, machine-dependent problems can be solved effectively and performance prediction can be achieved at all decision levels.

The book is organized as follows. It first analyzes the existing systems for parallel software production, present in the literature, and outlines the intrinsic limitations of these different approaches. Then, it details an innovative methodology for the development of parallel programs which provides the programmer with a small set of parallel forms (skeletons) as building blocks for parallel applications. Finally, it describes P3L, a structured parallel programming language based on skeletons, discusses the organization of the P3L compiler, and gives examples of structured parallel program development.

The book is targeted at researchers, both in academic institutions and in progressive commercial companies, and at aspiring research students in the area of parallel and distributed processing.

A number of people have made important contributions to the development of the ideas, methodology and tools discussed in this book. I owe much to Bruno Bacci, Marco Danelutto, Salvatore Orlando and Marco Vanneschi for their extensive work on the P3L project, first at the Hewlett Packard Pisa Science Center and then at the Department of Computer Science of the University of Pisa. In particular, Bruno Bacci and Salvatore Orlando also developed most of the first prototype compiler of P3L. Marco Danelutto proofread part of the book and provided some of the examples in Chapter 10. Special thanks go to Milon Mackey, who developed the initial front-end and participated enthusiastically in the P3L project in his spare time. Fabio Piazzai, Fabrizio Pasqualetti, Francesco Chiaravalloti, Barbara Cantalupo and Nicola Guerrini contributed to the template development for both the Meiko CS1 and the PVM versions of the compiler. A number of undergraduate students participated in the project to build a set of applications using the P3L compiler: Domenica Barresi, Giacomo Giunti, Stefano Milana, Paolo Pesciullesi, Paola Criscione, Gianni De Giorgi, Antonio Biso, Alessia Conserva, Stefano Bordin, Davide Pasetto and Maria Gabriella Brodi. Roberto Ravazzolo and Alessandro Riaudo implemented the OCR application discussed in Chapter 10. I would like to thank Peter Thanish for the many helpful suggestions on the material and the presentation of Chapter 4. Very special thanks go to Paul Kelly, who is definitely responsible for convincing me to write this book. I am grateful to Murray Cole, whose suggestions greatly contributed to the improvement of the manuscript and to David Skillicorn for his encouragement.

Finally, I am indebted to Bruno, who participated in the book in more ways than I can mention.

SUSANNA PELAGATTI
University of Pisa
Pisa, Italy
susanna@di.unipi.it

Contents

List of Tables

List of Illustrations

CHAPTER ONE

Introduction

The main obstacle to the widespread diffusion of parallel computing is its complexity and the cost of the associated software development process.

What is really needed is a general purpose methodology for the development of parallel programs and their support able to ensure:

- **programmability**, it should be possible to write and modify parallel programs easily, and to prove their correctness against specifications;
- **portability**, the programs written should be portable across a broad range of architectures,
- **performance**, the support should be able to translate a program for different target architectures achieving good performance figures and optimized resource utilization.

In this book, we first analyze the main problems to be solved to achieve these goals, and then propose a methodology that seems able to overcome most of the problems outlined.

1.1 EFFECTIVE SEQUENTIAL PROGRAMMING

A brief analysis of the way in which programmability, portability and performance have been achieved in imperative sequential programming may be useful to illustrate the main issues to be addressed.

In the sequential world, the above-mentioned goals have been achieved by providing

- a high-level language in which a programmer can easily express sequential algorithms, without dealing with machine-dependent features,
- a performance calculus, which can be used to predict the performance of a sequential algorithm (program),
- efficient compiling tools, able to translate and optimize the source code automatically (and effectively) against the target architecture.

Given a problem P to be solved, the programmer first analyzes P in order to find a "good" sequential algorithm $\mathcal{A}_P$ to solve it. The choice of $\mathcal{A}_P$ is guided by a performance calculus allowing the programmer to optimize the algorithm structure *without specifically taking into account the target machine features*. Then, the algorithm is coded in a suitable language and is optimized and translated by the compiler according to the underlying architecture (e.g. superscalar, vector, VLIW). If the underlying architecture changes, it is sufficient to re-compile the program for the new target and all the advantages are maintained.