

HOW TO WRITE PARALLEL PROGRAMS

A FIRST COURSE

By Nicholas Carriero

and

David Gelernter

Dedications

For D. and J.P. and in memory of
M.R.C. who, having taught, teaches still.

For my parents, the Sibs, Daniel and Jane --
Ein Blick von dir, Ein Wort mehr unterhält
Als alle Weisheit dieser Welt.

Third printing, 1992
©1990 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

Reproduced with the permission of MIT.

Carriero, Nicholas.

How to write parallel programs: a first course / Nicholas Carriero, David Gelernter.

p. cm.

Includes bibliographical references and index.

ISBN 0-262-03171-X

1. Parallel programming (Computer science) I. Gelernter, David Hillel. II. Title.

QA76.642.C37 1990 90-44443

005.2--dc20 CIP

Contents

Preface

Acknowledgments

1 Introduction

1.1 What?

1.2 Why?

1.2.1 Parallelism

1.2.2 Coordination

1.3 How?

2 Basic Paradigms for Coordination

2.1 Introduction

2.2 Paradigms and methods

2.2.1 The paradigms

2.2.2 The programming methods

2.3 Where to use each?

2.4 An example

2.5 A methodology: how to do the three techniques relate?

2.6 Where are the basic techniques supported?

2.7 Exercises

3 Basic Data Structures

3.1 Linda

3.2 The basic distributed data structures

3.3 Structures with identical or indistinguishable elements

3.3.1 Message passing and live data structures

3.4 Exercises

4 Performance Measurement and Debugging

4.1 Introduction

4.2 Debugging

4.2.1 Software tools for debugging

4.2.2 Tuplescope

4.2.3 Logical problems in parallel debugging

4.3 Performance

4.3.1 Modeling the behavior of parallel programs

4.4 Exercises

5 Putting the Details Together

5.1 Result parallelism and live data structures

5.2 Using abstraction to get an efficient version

5.3 Comments on the agenda version

5.4 Specialist parallelism

5.5 Conclusions

5.6 Exercises

6 Databases: Starting with Agenda

6.1 Big Issues

6.2 Problem: database search

6.3 The problem discription

6.4 The sequential starting point

6.5 A first parallel version

6.6 Second version: Adding "Flow Control"

6.7 Third parallel version

6.8 Performance analysis

6.9 Conclusions

6.10 Exercises

- 7 Matrices: Starting with Result
- 7.1 Big issues
- 7.2 Problem: the wavefront computation of a matrix
- 7.3 The result-parallel approach
- 7.4 A result => agenda transformation
 - 7.4.1 Granularity
 - 7.4.2 An agenda-style solution
 - 7.4.3 What should sub-blocks look like?
 - 7.4.4 Task scheduling
- 7.5 Back to the original database problem
- 7.6 And the denouement: hybrid search
- 7.7 Conclusions
- 7.8 Exercises

- 8 Networks: Starting with Specialist**
- 8.1 Big Issues
- 8.2 The data-stream analysis problem
- 8.3 The process trellis architecture
- 8.4 The specialist parallel approach
- 8.5 A specialist => agenda transformation
- 8.6 Static scheduling and realtime
 - 8.6.1 A realtime scheduling algorithm
- 8.7 Conclusions
- 8.8 Exercises

- 9 Coordinated Programming**
- 9.1 And now for something completely different
 - 9.1.1 Two classical problems
 - 9.1.2 A solution
 - 9.1.3 The readers/writers problem
- 9.2 The meeting maker
 - 9.2.1 The problem
 - 9.2.2 The general form of a solution
 - 9.2.3 Implementing the system
 - 9.2.4 Characteristics of the solution: distribution
- 9.3 Exercises

Bibliography

Preface

This book is the raw material for a hands-on, "workshop" type course for undergraduates or graduate students in parallel programming. It can also serve as the core of a more conventional course; and it might profitably be read (we hope and believe) by any professional or researcher who needs an up-to-date synthesis of this fast-growing, fast-changing and fast-maturing field.

By a "workshop" course we mean a course in which student projects play a major part. The exercise sections at the end of each chapter are integral to the text; everyone who consults this book should (at least) read them. Problems in chapters 2 through 5 lay the groundwork; the exercise sections in the last four chapters each center on a detailed investigation of a real and substantial problem. For students who pursue them seriously, these programming problems will require time, care and creativity. In most cases they lack stereotyped solutions. Discussion of student efforts can and ought to play a significant part in class meetings.

The programming examples and exercises use C-Linda (Linda is a registered trademark of Scientific Computing Associates.); C-Linda running on a parallel machine or a network is the ideal lab environment for the workshop course we've described. A C-Linda simulator running on a standard workstation is an adequate environment. Relying on some other parallel language or programming system is perfectly okay as well. The called-for translations between the book and the lab environment might be slightly painful (particularly if the non-Linda parallel language you choose is any variant of the ubiquitous message-passing or remote-procedure-call models), but these translation exercises are always illuminating, and anyway, they build character.

The "more conventional" course we mentioned would deal with parallel systems in general. Parallel *software* is still the heart and soul of such a course, but teachers should add some material to what is covered in this book. We would arrange the syllabus for such a course as follows:

1. *Introduction and basic paradigms.* (The first two chapters of this text.)
2. *Machine architectures for parallelism.* We'd use chapter 21 of Ward and Halstead's *Computation Structures* [WH90], or part 3 of Almasi and Gottlieb's *Highly Parallel Computing* [AG89].
3. *Linda, and parallel programming basics.* (Chapter 3.)
4. *Parallel languages and systems other than Linda; two special-purpose models of parallelism: data-parallelism and systolic systems.* The most comprehensive and up-to-date overview of parallel languages and systems is Bal, Steiner and Tanenbaum's survey paper on "Programming languages for distributed computing systems" [BST89]. (A brief survey appears in section 2.6 of this book.) Hillis and Steele's paper on "Data parallel algorithms" [HS86] is a good introduction to data parallelism; Kung and Leiserson's paper on "Systolic arrays (for VLSI)" [KL79] is the classic presentation of systolic programming.

This section ought to make a point of asking (among other questions) how Linda differs from a horde of competitors. The Bal *et al.* paper discusses Linda in context, as does Ben-Ari's *Principles of Concurrent and Distributed Programming* [BA90] (which is mainly about Ada but has good discussions of occam and Linda as well), the parallel languages chapter of Gelernter and Jagannathan's *Programming Linguistics* [GJ90], and the authors' paper on "Linda in context" [CG89].

5. *Writing parallel programs:* the rest of the text.

Acknowledgments

Many people contributed enormously to the effort that culminated in this book. First, the Linda group at Yale: particularly (in order of appearance) Jerry Leichter, Rob Bjornson, Venki Krishnaswamy, Mike Factor and Susanne Hupfer; many thanks, too, to Paul Bercovitz, Scott Fertig, Jim Philbin and Stephen Zenith. The second author acknowledges with particular satisfaction the help, contributions and advice of two of the same people he mentioned in his thesis seven years ago: Mauricio Arango and Suresh Jagannathan. Both authors are grateful for the help and support of their colleagues and friends at Bell Labs, particularly Sid Ahuja and Tom London.

We will always be grateful to Dick Lau of the Office of Naval Research for funding support that made our work possible. The National Science Foundation has been a friend and indispensable supporter of our research in good times and bad, and a model of seriousness and integrity in all its dealings. Chris Hatchell, our administrative assistant and group manager-coordinator, set new standards. We are much indebted to him for years of brilliant, indispensable helpfulness. And we thank our international visitors, and the research groups around the world who have kept us informed about their work on and with Linda.

But in the end, our acknowledgments come down to one man. If the discipline of computer science had a man of the decade during the 1980's, our candidate would be Martin Schultz. For many of those years Martin was chairman of the department at Yale; for all them, he was a leader of the field. Martin saw clearly that *significant* contributions to computer systems research (*vs.* mere academic wheel-spinning) require hardware *and* systems research *and* real applications. When you put those ingredients together (he thought) the reaction might be spectacular; it might illuminate the field. He was right. He had a vision, he put it into effect, and wonderful things happened. Thanks, Martin—

Nicholas Carriero
David Gelernter