A COMPARATIVE STUDY
OF PARALLEL
PROGRAMMING LANGUAGES:
THE SALISHAN PROBLEMS

Edited by

John T. FEO
Computer Research Group
Lawrence Livermore National Laboratory
Livermore, CA, U.S.A.

1992
NORTH-HOLLAND
AMSTERDAM · LONDON · NEW YORK · TOKYO
Introduction to the Series

Large scale computing is a growing field of research that plays a vital role in the advancement of science, engineering, and modern industrial technology. Computing is fast becoming the most frequently used technique to explore new questions. In just the last few years, the inclusion of computer modeling has produced results that were inconceivable a decade ago. They are an indispensable tool in many areas; from climate studies to chemical dynamics, from automated manufacturing to operating hospital intensive care units. Computer simulations of scientific processes provide, in many cases, substitutions for actual experiments. These simulations are less expensive and can address a wider range of problems. Computer simulations also provide an understanding of physical problems that cannot be obtained from experiments alone.

Research problems of many sorts are now becoming increasingly dependent on computer models, and numerical experiments are taking their place alongside the more traditional methods of research. Along with the theoretical and experimental, there is now a computational aspect of science.

Increasing sophistication in research has led to a need for bigger and faster computers; for Supercomputers. In this quest, supercomputers are themselves stimulating the redevelopment of the methods of computation. Results in one area are quickly adapted for another. The effect is making super-computation a multi-disciplinary adventure. Research scientists in super-computing come from a variety of interests and backgrounds and can be found in all universities, laboratories and industries.

In supercomputing, a large overlap is found between the academic areas of engineering and physical sciences and the academic areas of mathematics, computer science, and computer engineering. The scientific jargons used in each of these areas are different and require translation or understanding before being able to make progress in the supercomputing sciences. Although many advances have made the process easier in many cases, it has not kept pace with the dramatic increase in demands placed on the computer as well as the growing complexity of the computer hardware that has occurred over the past decade.

Special Topics in Supercomputing will take on two directions. First, in recognition of the fact that research in supercomputing is constantly embarking in new directions, a part of the series will be devoted to topics that have just begun to solidify as a well-defined research area. These volumes will contain manuscripts from researchers who are current leaders in the field.
Second, Special Topics in Supercomputing will include as part of its series a collection of monographs and contributed volumes on new- and well-established areas of supercomputing. As certain topics of supercomputing begin to solidify on a firm theoretical foundation, they will be coalesced into monograph or textbook form by author(s) who are experienced in the field. On the other hand, important areas of supercomputing are so new that few manuscripts can be collected to warrant a full-scale book. These topics will then be included in the contributed volumes of the series.

A hope of this enterprise will be to make supercomputing more widely understood, and more accessible in fields where it has not yet penetrated because of insufficient information.

G. Rodrigue
Preface

As execution speeds reach the physical limits of single CPU computers, our only hope of achieving greater computing power is parallel systems. An area of intense research is parallel programming languages. Researchers have proposed countless numbers of new programming languages, extensions to existing languages, and programming tools. Unfortunately, the research is confined primarily to academia with little input from the user community. The differences, similarities, strengths, weaknesses, and appropriate problem domains of the various approaches are subtle and often not well understood. Given this confusion, an informed comparison of parallel languages is difficult.

In this book we use a basis of comparison that both the language and computation communities can understand and appreciate. We compare eight parallel programming languages based on solutions to four problems. Each chapter includes a description of the language's philosophy, semantics and syntax, and a solution to each problem. The chapters are written by recognized experts; in some cases, the author is the language designer. We believe that by discussing solutions rather than language features or theoretical properties, we may bridge the gap between the language specialists and users.

This book is appropriate as a supplementary text for a graduate class in parallel programming languages. Since the book approaches the study of languages from the standpoint of computations, both computer science and computational science graduate students will benefit from the book.

We invite the proponents of other parallel languages to publish their own solutions to the Salishan Problems in other forums. This book includes only a subset of the classes of parallel programming languages. We would like to see solutions in all classes of languages, and
in as many instances of each class as possible. If a real implementation of the language exists, we recommend that the author discuss performance. In the eyes of most application programmers, a language is only as good as its performance. We also encourage scientists in all fields to suggest new problems whose computational models are not already included. Our hope is that the set of problems will become recognized as a standard by which to compare parallel programming languages, just as the Livermore Loops and Linpack are used to evaluate computer systems.

John Feo
1. Introduction

Comparisons of parallel programming languages are more often based on theoretical criteria than practical ones. Traditional studies have compared parallel languages with respect to argument passing, evaluation order, support for concurrency, synchronization and communication methodologies, and support for operations on specific data structures such as arrays, lists, or trees. Typically, the studies use artificial or trivial program examples to illustrate the language features. They rarely discuss how the different approaches affect the development of real applications, leaving it to the reader to infer the effects of the language features on his or her problem.

How many talks have we attended in which the speaker drones on for forty minutes describing the syntax of yet another parallel language and then exhibits the code for factorial, Fibonacci, or the Eight Queens problem? The lecturer leaves the application programmer with little sense of the real advantages and disadvantages of the language, what types of parallel programs the language can and cannot express efficiently, and even whether or not the language can express a particular algorithm. By describing the trees and not the forest, and by using irrelevant examples, the speaker often fails to convince the listener to try the new language.