Reviews

Introduction to Parallel Programming, by Steven Brawer. ISBN 012-128-4700, 1989, $45.00, 422 pp., hardbound. Available from Academic Press, Inc., 1250 Sixth Ave., San Diego, CA 92101 or 21-28 Oval Rd., London NW1 7DX. UK.

DETAILED SUMMARY

Is it useful? This highly practical book is useful for the scientific programmer who wants to learn about parallel programming, primarily for those learning to program in the shared memory model. It is also useful as a reference, with a cookbook style that will help the programmer to parallelize most common classes of loops. By adding to its existing exercises, it would also be usable as a textbook.
Interesting features: All of the software for all programs in the book are available on floppy disk, in C and Fortran, from the author.
Problems: The book's primary weakness is in its production quality. The typesetting of equations is primitive and confusing. The figures are crudely drawn and their graphical design is often confusing. Finally, there are occasional textual mistakes that could have been caught by the editor. While these flaws diminish the quality of the book, they do not severely impact the content.

Another possible problem is the book's bias toward the environment provided by bus-based multiprocessors, such as the Encore. Increasingly, parallel computers do not have the uniform access to memory or the kind of process scheduling provided in this model. For maximum performance on many machines, the user would need to consider issues not covered in this book, such as data distribution.
Conclusion: This book is a useful introduction to parallel programming for the experienced scientific programmer, who has no previous knowledge of parallel programming, and who wants a practical presentation.

In the last 5 years we have seen parallel computers supercede the "traditional" vector computers as the world's fastest computers. The typical scientific programmer, who is more interested in getting scientific results quickly than in playing with new machines, can no longer ignore this new class of machines. Although computer scientists may someday make it possible for "dusty-deck" codes to run faster on parallel computers just by recompiling, this day seems a long way off. The scientific programmer who wants to take advantage of the new supercomputers must learn to write parallel programs.

Brawer's practical book is designed for the scientific programmer who is ready to take the plunge into parallel computing. Brawer carefully leads the experienced scientific programmer into the world of shared memory, locks, and loop dependencies. He assumes only that the reader has programmed in a procedural language like Fortran, C, Pascal, or Ada, and knows nothing about parallel computers, parallel programming, or ad-

Reviewed June 1993

Scientific Programming, Vol. 4, pp. 115–118 (1995)
CCC 1058-9244/95/020115-04

vanced mathematics. All new terms and concepts are carefully defined, in just enough detail for practical understanding, and there are lots of examples.

The book is not specific to any programming language, programming environment, operating system, or architecture. However, the model is clearly that of shared memory architectures, like the Cray, Encore, Sequent, BBN TC2000, or KSR. Although many of the concepts in the book apply to all parallel programming models, the book is less directly applicable for programmers using a message-passing machine (such as Intels or nCUBEs) or using a data-parallel model (such as that used by Maspar or Thinking Machines). In particular, all of the examples used shared memory.

Brawer chose to use a subset of Fortran for the example programs, all of which are available separately (in C or Fortran) on floppy disk. Since they are restricted to an elementary subset of Fortran, the programs could be easily rewritten in another language, such as C. For example, one could easily adapt them for use with the p4 package from Argonne Laboratories, which is a freely available parallel programming library that can run on nearly every parallel computer or workstation network, and is similar to Brawer's model.

The book is well organized, beginning with an introduction to the basic ideas of parallel computer architecture, programming models, processes, shared memory, and synchronization. It then delves deeply into the issues of loop scheduling and data dependencies, familiar issues to the vector computer programmer. Finally, it covers performance issues, and several diverse sample applications, such as discrete event simulation, line fitting, integration, traveling salesman, and Gaussian elimination. Throughout, Brawer works through examples to demonstrate shared and private variables, the importance of synchronization, and so forth.

Brawer devotes four chapters to the issues of loop scheduling and data dependencies, which may be handled by the compiler in some environments. Nonetheless, the rest of the book is still valuable reading for any beginning parallel programmer.

Message-passing compouter architectures have become popular recently. The book focuses on shared memory, although many concepts (such as synchronization and scheduling) apply to message passing as well. Furthermore, despite the prevalence of message-passing architectures (including cluster-based computing) I expect that shared memory will become the predominant programming model for scientific programming, regardless of architecture.

Finally, although compiler support for data-parallel programming is improving (e.g., the high performance Fortran [HPF] project), the book's loop-scheduling information is useful for programmers who cannot wait for that support, and for those who want a better understanding of what the compiler is doing. In addition, the book also contains several examples of irregular problems that would not easily fit the mold of most data-parallel languages.

FIGURE 1 Table of contents.

David Kotz
Dartmouth College
Department of Computer Science
Hanover, NH 03755

e-mail: dfk@cs.dartmouth.edu

FORTRAN 90 Explained, by Michael Metcalf and John Reid. ISBN 0-19-853772-7, 1990 (reprinted 1993), 35.00 Swiss Francs, 306 pp., softbound. Available from Oxford University Press, Inc., 200 Madison Ave., New York, NY 10016.

The nature of scientific programming has changed considerably from the 1950s when the term "formula translation" was, by and large, an accurate description of the task. The last 40 years have witnessed huge improvements in algorithms, hardware speed, and memory capacity. Consequently, scientific programs that manipulate complex data structures, such as nonuniform meshes or multilevel grids, and exploit the potential of multiprocessor architectures are nowadays commonplace.

These enormous changes have been reflected in the evolution of Fortran from the 66 standard through the 77 standard to the more recently introduced 90 standard. Some constructs which were originally significant (such as the EQUIVALENCE statement for partitioning a limited main memory) are now virtually redundant while other constructs have become highly desirable (such as encapsulation mechanisms to organize complex computations). There are many and various differences between the 77 and the 90 standards (although the former is a subset of the latter). Although several of the novel features are of dubious merit there are hugely significant additions:

1. Mechanisms which facilitate the definition of dynamic and user-defined types
2. Data-parallel array operations
3. Powerful encapsulation methods
4. An intrinsic library of scientific operations

The provision of these features should greatly aid the task of constructing scientific software.

The FORTRAN 90 standard [1] acts as the definition of the language. Like many reference manuals it is suitable for investigating particular, very detailed queries rather than providing an overview of the entire language. On the other hand *FORTRAN 90 Explained* both defines the underlying constructs and is arranged in such a way that it can be read from cover to cover. In essence, the content of the book is a distillation of the standard with an approachable presentation. The fact that both authors were involved in the design of the language has helped to make the book both thorough and consistent with the standard.

The early chapters describe respectively, lexicographic tokens and types, expressions and assignments, and control statements. Thereafter, new features are illustrated with appropriate program fragments. Each chapter contains an exercise section and solutions are provided in an appendix. A number of the constructs of FORTRAN 90 have been marked for possible future removal; these are listed in a self-contained appendix. In addition, the authors have selected a number of other features of the language which are considered to be redundant; these are defined in the final chapter of the book. Both the appendix and the final chapter are surprisingly short.

One disadvantage of the organization is that information about a particular topic may be dispersed. For example, pointer variable declarations are described in one chapter, pointer assignment is outlined in another, and allocation / deallocation is discussed in a third.

In general, each linguistic feature is "explained" with (i) a syntactic outline, (ii) an informal description, and (iii) an illustrative example. For the most part the definitions are clear and understandable. When descriptions are difficult to follow the problem often lies with the construct rather than the explanation. For example, it is difficult to see how the declaration

```
INTEGER, PARAMETER :: LONG =
SELECTED_REAL_KIND(9,99)
```

could be made appetizing. It is possible to quibble about some descriptions. For example,

1. Pointers are not described in terms of location to value bindings
2. The first *illustrative* example of a "recursive function" is complex, indirect, and involves a function parameter

Reviewed July 1994

3. The distinction between bounded and unbounded DO iterations is not highlighted
4. An infinite recursion is described with the expression "loop over itself forever."

However, the weaker definitions are relatively rare.

I consider the book to be primarily a reference work. The details of the constructs of Fortran 90 are adequately defined: there is, however, a limited explanation of the more general scheme of things. For example, there is not a convincing justification for, and explanation of, the importance of modules (an abstract data type definition may have the beneficial effect of isolating occurrences of pointers to a single section of program text and, as a result, simplifying the task of reasoning). Be that as it may, I consider *FORTRAN 90 Explained* to be the best current book on this subject matter.

REFERENCE

[1] *International Standard.* ISO/IEC 1539. 1991.

Alan Stewart
The Queen's University of Belfast
Department of Computer Science
Belfast BT 7 1NN
Northern Ireland

e-mail: astewart@cs.qub.ac.uk