LECTURE NOTES IN COMPUTER SCIENCE, 57

Portability of Numerical Software

Edited by Wayne Cowell

TP31
U2

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

57

Portability of Numerical Software

Workshop, Oak Brook, Illinois,
June 21–23, 1976

Edited by Wayne Cowell

Springer-Verlag
Berlin Heidelberg New York 1977

Editor
Dr. Wayne Cowell
Applied Mathematics Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439/USA

Library of Congress Cataloging in Publication Data

Workshop on the Portability of Numerical Software, Oak
Brook, Ill., 1976.
Portability of numerical software workshop, Oak Brook,
Illinois, June 21-23, 1976.

(Lecture notes in computer sciences ; 57)
Workshop organized by Argonne National Laboratory.
Bibliography: p.
Includes index.
1. Numerical analysis--Computer programs--Congresses.
2. Software compatibility--Congresses. I. Cowell,
Wayne, 1926- II. United States. Argonne National
Laboratory, Lemont, Ill. III. Title. IV. Series.
QA297.W65 1976 001.6'4 77-13623

AMS Subject Classifications (1970): 00A10, 68A20
CR Subject Classifications (1974): 4.22, 4.6, 5.11, 5.25

ISBN 3-540-08446-0 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-08446-0 Springer-Verlag New York Heidelberg Berlin

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

INTRODUCTION

Those who cannot remember the past are condemned to repeat it - Santayana

The man who doesn't write portable software is condemned to rewrite it -

J. F. Traub (1971)

A computer program represents a body of problem-solving experience, often preceded by painstaking analysis, algorithm selection, and programming. The value of a program is magnified many-fold when it can be moved and effectively used in computing environments other than the one where it originated; but computing environments differ in both obvious and subtle ways and every successful computer program has evolved as an adaptation to a particular environment.

The benefits and difficulties of sharing programs are widely appreciated in the computing community, nowhere more so than among the developers and users of mathematical software, the programs that perform the basic numerical calculations of science and engineering. The finiteness of computers demands mathematical compromises that may be reached in different ways in different programs on different systems. At the same time, numerical calculation is so fundamental that there is a significant movement to create high quality mathematical software for distribution and widespread use in libraries. For some of those involved in this activity the portability problem is primarily a research challenge. Others have a business interest, seeing portability as a way to reach a wider market for their software products. Still others work in computing centers and are continually faced with the task of adapting a program for use in their home environment. Whether its primary motivation is research, business, or services, each group concerned with mathematical software recognizes the breadth of the portability problem and the need to exchange information and collaborate toward improving the current state of affairs.

To facilitate such information exchange, a Workshop on the Portability of Numerical Software was organized by Argonne National Laboratory and held in Oak Brook, Illinois on June 21-23, 1976. About three dozen numerical analysts and software developers from seven countries participated. The papers in this volume are based on oral presentations at the workshop but they were written after the give and take of the discussion that surrounded each presentation; thus we may assume that the impact of that discussion, as perceived by each author, is distilled in his or her paper. The collection is offered to the computing community as a point of reference in the continuing effort to produce portable numerical software.

The authors and the editor are indebted to many individuals and groups who contributed to the workshop and to this volume. Especially deserving of mention are:

- The IFIP Working Group on Numerical Software (WG 2.5), whose members first suggested the workshop and then participated fully;
- The National Science Foundation and the U.S. Energy Research and Development Administration who provided financial support;
- Burton Garbow who made many helpful comments on the manuscripts;
- Patricia Witkowski who skillfully retyped the original manuscripts to achieve uniformity of format throughout the book.

Wayne Cowell
Argonne, Illinois
June, 1977

WORKSHOP PARTICIPANTS

T. J. Aird
International Mathematical and
Statistical Libraries, Inc.
Houston, Texas

James M. Boyle
Argonne National Laboratory
Argonne, Illinois

W. S. Brown
Bell Telephone Laboratories
Murray Hill, New Jersey

George D. Byrne
Univ. of Pittsburgh
Pittsburgh, Pennsylvania

W. J. Cody
Argonne National Laboratory
Argonne, Illinois

Wayne Cowell
Argonne National Laboratory
Argonne, Illinois

Ingemar Dahlstrand
Lund University
Lund, Sweden

Carl de Boor
Mathematics Research Center
Madison, Wisconsin

T. J. Dekker
University of Amsterdam
Amsterdam, The Netherlands

L. M. Delves
University of Liverpool
Liverpool, England

Jack Dongarra
Argonne National Laboratory
Argonne, Illinois

Kenneth W. Dritz
Argonne National Laboratory
Argonne, Illinois

J. J. Du Croz
The NAG Central Office
Oxford, England

Bo Einarsson
National Defense Research Institute
Tumba, Sweden

John Gabriel
Argonne National Laboratory
Argonne, Illinois

B. Ford
The NAG Central Office
Oxford, England

Lloyd D. Fosdick
University of Colorado
Boulder, Colorado

P. A. Fox
Bell Telephone Laboratories
Murray Hill, New Jersey

L. Wayne Fullerton
Los Alamos Scientific Laboratory
Los Alamos, New Mexico

W. Morven Gentleman
University of Waterloo
Waterloo, Ontario

A. D. Hall
Bell Telephone Laboratories
Murray Hill, New Jersey

Dennis Harms
Hewlett-Packard Co.
Cupertino, California

Pieter W. Hemker
Stichting Mathematisch Centrum
Amsterdam, The Netherlands

Frances Holberton
National Bureau of Standards
Gaithersburg, Maryland

T. E. Hull
University of Toronto
Toronto, Ontario

William M. Kahan
University of California
Berkeley, California

P. Kemp
University of Cambridge
Cambridge, England

Fred T. Krogh
Jet Propulsion Laboratory
Pasadena, California

C. L. Lawson
Jet Propulsion Laboratory
Pasadena, California

Dan Lozier
National Bureau of Standards
Gaithersburg, Maryland

Paul Messina
Argonne National Laboratory
Argonne, Illinois

Cleve Moler
University of New Mexico
Albuquerque, New Mexico

James C. T. Pool
Argonne National Laboratory
Argonne, Illinois

J. K. Reid
AERE Harwell
Didcot, England

Christian Reinsch
Leibniz-Rechenzentrum
Munich, Germany

John Rice
Purdue University
Lafayette, Indiana

J. L. Schonfelder
The University of Birmingham
Birmingham, England

Brian T. Smith
Argonne National Laboratory
Argonne, Illinois

Hans J. Stetter
Technische Hochschule Wien
Vienna, Austria

M. Sund
Gesellschaft für Strahlen und
Umweltforschung
Munich, Germany

Christoph Überhuber
Technische Hochschule Wien
Vienna, Austria

N. Victor
University of Giessen
Giessen, Germany

W. M. Waite
University of Colorado
Boulder, Colorado

内 部 交 流

P 69/43

数值软件的可移植性

(英 3-5/7018)

A 00510

1980, 2, 28

TABLE OF CONTENTS

WHAT IS PORTABILITY?

"Then you should say what you mean," the March Hare went on.
"I do," Alice hastily replied; "at least - at least I mean what I say - that's the same thing, you know."

- Lewis Carroll

At first thought one might expect *portability* to be well defined as a concept, whatever the difficulty of its achievement. In fact, however, several concepts emerge when computer scientists are asked to be precise about their use of the term and we find more than one definition in the pages of this volume. The Numerical Algorithms Group (NAG) demands that *portable* software be movable from one environment to another without change. If such a move requires only changes that are capable of mechanical implementation via a preprocessor, the NAG people say that the software is *transportable*. Boyle and Cody explicitly adopt the NAG usage. On the other hand, Brown, Hall, and Fox say that a program is *portable* if the effort to move it to a new environment is much less than the effort to rewrite it for the new environment. Reinsch suggests that software is *portable* if successful running in a new environment is achieved by no more than manual adjustment of machine dependent parameters intentionally provided for that purpose. Then he exposes the folly of trying to be too precise by reminding us that, in the current state of the art, any definition depends on implicit assumptions which are left undefined.

We shall take the position that the meaning of portability is contained in the collective descriptions of what is actually being done to transplant computer programs. Each "definition" serves a group of workers in their local setting. More generally, what we seek is a sense of the relationship between portability investigations and other mathematical software endeavors. In this vein we begin with Christian Reinsch's paper, Some Side Effects of Striving for Portability.

Reinsch provides perspective by discussing seven "side-effects" of our efforts to achieve portability. These are side benefits really -- desirable results of taking portability very seriously in our approach to numerical computation. Reinsch gives special treatment to the implications of a keen awareness of machine arithmetic on the part of programmers intent upon producing portable software. It is both refreshing and sobering to see what might happen if we "mean what we say."

SOME SIDE EFFECTS OF STRIVING FOR PORTABILITY

Christian Reinsch

Portability and *Numerical Software* are concepts whose meaning and contents may vary. Hence, at the beginning of this Workshop, it might be appropriate to work out some delineation of their scopes so that different interpretations and applications of these terms are avoided. This speaker is neither authorized nor otherwise entitled to foist his own and probably one-sided definitions upon the meeting so that a clarification of these points is left open and recommended for subsequent discussion. Here are some possibilities.

NUMERICAL SOFTWARE. From a first point of view, numerical software consists of individual routines, independent of each other and usually of modest length. They are the realizations of numerical algorithms as described in our textbooks and journals. Editors of the latter have a tendency to ask for test examples and numerical confirmation when authors submit papers which propose new algorithms. Therefore, many numerical analysts are inclined to program their methods and to submit for publication these products by which they achieved their results along with the corresponding theory. Among the more conspicuous examples are the program collections and algorithms sections of the journals *Communications of ACM* (since February, 1962, but now in TOMS), *The Computer Journal, Numerische Mathematik, BIT, Computing,* etc. Also, it is now not unfashionable for textbooks of numerical analysis to include some appendix with FORTRAN or ALGOL programs.

This view might seem too narrow or even inappropriate for many. Indeed, one could have the impression that occasionally printed programs are not fully acknowledged as software. And if one accepts as software only what is stated in FORTRAN and punched on cards or written on magnetic tape, then from this point of view, numerical software would essentially be the standard program libraries which come with every computer or the service libraries like *IMSL*, *NAG*, etc.

One step further in the definition of numerical software would include the systematic combination of such algorithms, called "multi-algorithms," which usually consist of a control program and a family of algorithms associated with a given problem. The former acts as a driver, selecting from among the latter the most promising routine. The decision rules, however, might well depend on the machine characteristics and this adds a new flavour to the portability problem. Most of the function approximation routines are indeed multi-algorithms in this sense, say mini-multi-algorithms (e.g., a Chebyshev expansion for one range of arguments and an asymptotic expansion for another range, etc.). Multi-algorithms, in particular, have been tried for numerical quadrature; another example with great reputation is the EISPACK system for algebraic eigenvalue problems.

The most comprehensive interpretation of "numerical software" would have to include all kinds of application packages, e.g., partial differential equation solvers and finite element methods in structure analysis (possibly including some interactive facilities for the geometry definition and element enumeration).

To sum it up, there are different definitions possible of what is to be included in our notion of "numerical software" and the discussion of portability questions might be heavily influenced by the chosen delineation.

PORTABILITY. Intuitively, our notion of portable numerical software is quite clear: the attribute "portable" is to be awarded to those programs which successfully run not only on one computer species but rather a variety of them; at least this should hold true after the manual adjustment of some machine dependent parameters intentionally provided for that purpose. However, the latter can only make up for quantitative differences rather than qualitative ones as provided by the diversity of structures of the more advanced computers. Examples of different computer categories are:

- (i) single processor with uniform random access memory,
- (ii) processors with paging and virtual memory facilities,
- (iii) pipeline or vector computers,
- (iv) parallel computers.

A few words will characterize the last two of these categories. Pipeline processors possess special machine instructions and devices for "vector operations" where one type of floating point arithmetic is executed on many independent pairs of operands which are equidistantly stored in memory, i.e., a hardware analogy of the much discussed basic linear algebra modules. To increase speed, each operation is decomposed into a sequence of more elementary steps such as sign handling, exponent removal, mantissa arithmetic, rounding, etc., which are performed at different stations along an assembly line so that a flow of operand pairs in various states of processing is maintained. In the stationary phase of the instruction one pair of operands is processed per machine cycle. Note that this approach to increased speed is promising not only for the super machines like the CDC STAR-100, the Texas Instruments ASC, etc., but could have a future for the small micro-programmable computers as well.

Parallel computers, on the other hand, have a fixed number of identical though independent processors connected through a network of data buses permitting the interprocessor communication. They need not perform the same instruction at the same time.

In a recent report Don Heller {4} discusses the consequences for software in the linear algebra area. To quote from the abstract: *The existence of parallel and pipeline computers has inspired a new approach to algorithmic analysis. Classical numerical methods are generally unable to exploit multiple processors and powerful vector-oriented hardware. Efficient parallel algorithms can be created by reformulating familiar algorithms or by discovering new ones, and the results are often surprising.*

Here, again, we need a delineation (which is necessarily arbitrary) of the scope of one of our basic terms. Without further discussion and more definite conclusions, the meaning of portability would remain obscure because it depends on implicit assumptions which are then left undefined. Most likely, one would aim at a definition of portability only within the range of a given computer category, and, at the moment, it is probably best to concentrate on just the first target group, viz., the familiar computer type with single processor and a uniform memory.

THE SIDE EFFECTS. Whatever definitions we agree upon, our striving for portability of numerical software has certain consequences, especially since portability has become a major issue for the professional production of numerical software. Some of the noteworthy side effects are as follows.

1. A revived interest in the standardization of programming languages, in particular, FORTRAN. The necessity of having rigorous standards for both syntax and semantics of the underlying programming language is clear.

2. Lots of proposals for FORTRAN extensions. Nearly any FORTRAN compiler will provide some additional features which are not part of the official language. No doubt these individual or local extensions of the possibilities a programmer has at his disposal have greatly relieved the pressure for official extensions. However, if programmers were willing to forgo such non-official extensions for the sake of portability, it is only natural for them to press for some extensions of the standards. Most common is the wish for allocation of working storage though this might cause a conflict with the general FORTRAN philosophy. Not so difficult to achieve and quite beneficial for the portability issue would be a COMMON which is not too common, i.e., the possibility for subroutines to share variables which, in turn, are protected and inaccessible from the rest of the world. Another feature, more useful for the development than for the application of numerical software, would be a data type where the programmer specifies the number of storage units per item and has the responsibility of providing the subroutines for the arithmetic operations and relations.

This would not be the right place to dwell on these features and their potential. The intention is only to demonstrate that each of us could provide his own suggestions and many of us, indeed, have tried it.