Portable programming revisited: 1982-1992

Wallis, P. J. L.

https://doi.org/10.4224/40003381

National Research Council Canada
Conseil national de recherches Canada

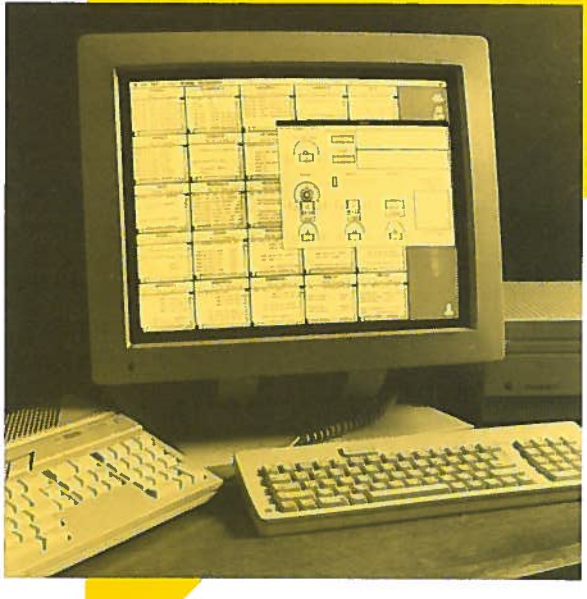
Institute for Information Technology
Institut de technologie de l'information

ERA-382

NRC·CNRC

Portable Programming Revisited: 1982–1992

P.J.L. Wallis
January 1994



Canada

NRC 37094

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

Portable Programming Revisited: 1982—1992[1]

P.J.L. Wallis[2]
January 1994

1 This report was prepared under contract for the Software Engineering Laboratory, Institute for Information Technology, National Research Council of Canada.

2 School of Mathematical Science, University of Bath, Claverton Down, Bath BA2 7AY, United Kingdom.

Contents

Abstract

Developments in portability over the last decade are reviewed and a selected bibliography given.

Résumé

Ce rapport traite des développements en matière de portabilité au cours de la dernière décennie et présente une bibliographie sommaire.

Introduction

The portability of a software product may be loosely described as the ease with which it can be moved between different computing environments. Constructing portable software is clearly a useful way of improving its value and applicability, making studies of how to achieve portability of great practical value. This document, prepared for the National Research Council of Canada, reviews developments in portability in the decade since publication of the author's 1982 book *Portable Programming* [1]. The main source of information used in the preparation of this report was a bibliographic database search for references relating to software portability in the years 1982–1992, which was undertaken by the National Research Council and retrieved over 2500 references.

During the 1960's and 1970's, portability was regarded as mainly involving overcoming hardware differences. Thus the book [1]—a technical overview of the current understanding of issues relevant to portability written in the early 1980's—emphasizes such issues as physical transfer of data, accommodating different characteristics of computer arithmetic, and retargetting of compilers. Other areas covered include the design of portable programs, the maintenance of portable software, and the use of macro processors in portability. The issues identified there are still relevant but now tell only a small part of the story of portability problems.

The 1982 book remains a useful overall guide to portability problems and how to overcome them; changes since the book was written mean that more emphasis must be placed now on software platforms, standards, and ways of ensuring compliance with standards. Portability problems themselves and ways of solving them are in essence barely changed since 1982; then, as now, the basic idea of any approach to portability is to identify abstractions of features used by programs that can be provided similarly on different underlying systems. What has changed in the last 10 years is the relative importance of hardware differences and differences in software platforms as obstacles to portability and the frequency with which software needs to be transferred between different platforms by relatively unsophisticated computer users.

The reasons for these changes in the portability problem in the last decade are ably and amusingly summarized by Gentleman and Feldman [2]. They suggest that vastly increased use of computer systems in the past decade means that many more disparate systems are available and consequent demands for easy portability have greatly increased. The root of most portability problems these days does not involve hardware differences directly but typically involves portability between different software platforms, language dialects, and differences in runtime libraries.

These changes in the portability problem in the past decade have informed the structure and content of the present document. One general improvement since 1982 that is characteristic of computer science as a whole is a greatly improved appreciation of the fact that portability in all its varied forms is fundamentally concerned with the idea of identifying suitable abstractions. In identifying the most important contributions to the portability field since the 1982 book, we found little in the way of improvements on the basic techniques described there, but much that tells the practitioner how to handle differences between different versions of contemporary software platforms. As might be expected, there is also much new material of a "case study" nature, describing experiences gained in particular portability exercises.

Two specific areas covered in the 1982 book are deliberately excluded from the present survey. Legal aspects of software portability are not considered because of lack of expertise; there are, however, journals concerned with issues of computer law such as *Tolley's Computer Law and Practice*, *Software Protection*, and *Jurimetrics Journal*. No attempt is made here either to update the information in the 1982 book on physical distribution issues; this part of the book has been rendered obsolete by hardware developments and any attempt to provide a replacement would be equally volatile, but in any case the area is of far less practical interest than previously because of the widespread use of electronic mail and computer networks.

This report identifies the more important developments in portable programming since 1982, in all areas of interest apart from legal and physical distribution issues. Key references are cited together with a few well-documented case studies. The following sections concern Software Platforms, High-Level Languages, Portable Compilers, Computer Arithmetic, Operating Systems and Multiprocessing, and Parallel Processing. Future developments are anticipated in a brief Conclusion.

Software Platforms

Even in the early 1980's it was becoming apparent that portability involves far more than portable compilers and operating systems. For instance, ref. 3 describes many unexpected problems of detail that may arise in moving a language system from one machine to another. Hanson [4] describes an early design for what would now be called a portable input-output platform. By the mid-1980's demands for the easy transfer of software between different computing environments were making an "open systems" approach imperative.[1]

New technical questions are raised by the emphasis on standard software platforms. These concern the suitability of the interfaces defined for specific application areas such as the development of software tools [6, 7] and, more generally, how to ensure conformance with standards for software platforms. There has been little work as yet on conformance techniques for software platforms; a notable exception is the Posix project, conformance for which is discussed below.

Open Systems Standards

The idea of Open Systems Standards emerged in the 1980's. The objective is to define standard application platforms to which vendors and users of software would comply, enhancing the portability of software products across computing environments. A number of initiatives in this area resulted. The U.S. National Institute of Standards and Technology (NIST) is supporting the development of standards for Open Systems Environments (OSE) to improve the portability of software, data, and skills and the interoperability of different manufacturers' hardware, software, and communications systems. Working in cooperation with users and vendors, NIST has developed an Applications Portability Profile (APP) that provides an initial set of specifications for American government agencies to use in planning for the migration to OSE. See ref. 8 for a recent standard resulting from this effort.

Similar initiatives have been undertaken by major manufacturers such as IBM, DEC, Data General, and Hewlett-Packard. The IBM open systems standard, for example, is their Systems Application Architecture (SAA) dating from 1987. Systems Application Architecture is described in refs. 9–11 and related to IBM's OSI data communications standard in ref. 12. An important effort to develop portable applications platforms independent of the standards of any particular equipment supplier is X/OPEN, a non-profit international and industry-wide organization started in Europe and now supported by many major suppliers of hardware and software. The work of X/OPEN was originally based on the use of the Unix System V operating system and the C programming language, but has since been greatly extended; see refs. 13 and 14 for details.

Software platforms may be used at a number of different logical levels ranging from the low level of operating systems primitives to higher-level features supporting applications programs and user interfaces. Apart from the well-known platforms discussed below there are many others of more limited use such as the Harmony realtime system [15], the Ten15 [16] platform of the UK Royal Signals and Radar Establishment, and the TRON project that is discussed in the section below on portable operating systems. The following brief surveys of well-known current platforms progresses roughly from low- to high-level platforms.

Unix

Unix is a very widely used operating system platform closely connected with the C programming language. Unix and C standards and standardization are reviewed in ref. 17. Portable programming in C is closely connected with the Unix operating system platform; refer to ref. 18 for an overview and to books on portable C and Unix programming such as refs. 19–22 for detailed technical information. Numerous experiences of moving the Unix system between machines have been reported; examples may be found in refs. 23–25. Further detailed information on C and Unix topics may be found in specialist newsletters and conference proceedings concerned with Unix and C programming. The IBM version of Unix is called AIX; its relationship to IBM's SAA is the subject of ref. 26.

Posix

Posix is an interface specification for Unix-based operating systems. The Posix standards are produced by an IEEE working group whose remit is to standardize a Portable Operating System Interface for Computer Environments, colloquially known as Posix. Membership of the working group includes staff from all the major

[1] Refer to ref. 5 for the alternative approach of simulating a number of different platforms within a single computing environment.

American hardware and software suppliers in the Unix market.

The initial Posix work was related to the C language but language-independent standards are now being developed. The current Posix standard is ref. 27 and includes an accompanying rationale, while ref. 28 addresses problems encountered by users of this standard. For the associated C language standard see ref. 29. Significant effort has been devoted to developing standards for Posix conformance, both in general [30] and for applications programs [31].

PCTE

The Portable Common Tools Environment (PCTE) is a European project initiated by a group of six European computer manufacturers with CEC support. It is an evolving standard defining a publicly usable tools interface that has received wide support throughout Europe. Refer to refs. 32 and 33 for technical overviews of PCTE. Experiences of interfacing PCTE to a particular language (the programming language Eiffel) are described in ref. 34. Further discussion of the PCTE project may be found in specialized conference proceedings, such as ref. 35.

X Windows and Graphical Interfaces

The X Window System, originally developed at MIT, is a highly portable window system that provides a basis for higher-level Graphical User Interface (GUI) standards such as the Motif and OPEN LOOK GUIs [36]. See refs. 37–39 for details of various parts of the X Window system and ref. 40 for discussion of the portability of the system. Kimbrough and Oren [41] describe an extension of the X Window System to support interactive Lisp applications. Further information on the X Window System and its portability may be found in the proceedings of specialist conferences devoted to X Windows or to Unix. International standardization of computer graphics platforms has been an active area. The Graphical Kernel System (GKS) is an established standard for 2-D graphics but is too low level to form a satisfactory applications interface; Yaacob [42] discusses the portability of a graphics system written in C and incorporating both applications and GKS layers. A further standard for both 2-D and 3-D graphics is the Programmers' Hierarchical Interactive Graphics System (PHIGS), which supports the construction of application models for hierarchical data structures called structure networks that may be edited; an interactive debugger for PHIGS is the subject of ref. 43. A portable implementation of the PHIGS standard is described in ref. 44, while an implementation of PHIGS based on the X Window System is the subject of ref. 45.

As our discussion of GKS and PHIGS suggests, graphics is an area in which separate low- and high-level abstractions are appropriate. Problems involved in this kind of abstraction are addressed in ref. 46 for user interfaces using GKS and in ref. 47 for 3-D graphics in general.

High-level Languages

As discussed in ref. 1, portable programming in high-level languages requires considerable care. Moreover, it is only to a strictly limited extent that the problems involved are amenable to solutions involving adherence to language standards and the associated use of appropriate software tools—see, for example, ref. 48. The limitations of such solutions tend to be very similar for different languages as explained in ref. 49, which is a comparison of Ada and Fortran portability standards from this point of view.

Portable programming in high-level languages also suffers from differences in software platforms; there have been attempts to minimize these effects using layers of abstraction [50]. Portable programming in C has already been discussed in connection with Unix platforms. Portable programming in Ada and in Fortran is discussed next. There are few references available for portable programming in other high-level languages, but case studies in portability frequently appear in specialist periodicals and conference proceedings associated with particular programming languages.

Ada

Portability was one of the original, well-publicized objectives of the Ada design. However, difficulties with Ada remain, as detailed in the portability section of ref. 51. These difficulties are comparable to those arising with Fortran [49]. Arithmetic portability in Ada has received detailed attention, as have the problems of Ada scientific computation in general [52, 53]. Numerous Ada portability case studies are documented in specialist publications such as ACM *Ada Letters* and the proceedings of Ada conferences.

Fortran

Fortran is widely used for scientific computation and the Fortran standardization process is well established. As explained in ref. 49, remaining portability problems with the language itself are like those of Ada and appear inevitable for any realistic high-level language. For an example of a large-scale Fortran portability case study see ref. 54. Special problems with Fortran portability can arise when considering such problems as realtime programming, message-passing, or multitasking, as in refs. 55 and 56. Other specialized examples of Fortran portability are discussed in specialized publications and conference proceedings relating to parallel processing and multitasking.

Portable Compilers

The principle of portable compiling involving front-end machine independence and back-end machine dependence remains much as it was in 1982, while it is clear that far more is involved in moving a compiler than a simple re-implementation of the back-end—see ref. 3, for example. An Ada "virtual machine" defining a front-end/back-end division for Ada [57] addresses the overall feasibility of this approach, while the Amsterdam Compiler Kit [58, 59] provides a general framework for constructing portable compilers. Recent work on the Amsterdam Compiler Kit [60] stresses the production of small fast portable compilers, a trend also apparent in developments related to portable C compilers [61]. Other recent work on portable compilers also emphasizes techniques for moving more of the work of optimization from the back end to the front end [62–64].

Computer Arithmetic

Difficulties with the variation of properties of integer arithmetic between different machines are usually quite easily handled except in a few specialized cases such as the programming of portable random number generators [65, 66]. Floating point portability remains difficult and controversial [67] despite the widespread use of the IEEE floating-point standard [68], which is so permissive that it might almost be described as a floating-point standards generator! For a comprehensive discussion of floating-point standards and portability problems see chapters 2 and 3 of ref. 52. The work on the detection of the characteristics of hardware arithmetic characteristics reported in 1982 has been extended to include the IEEE Standard [69].

Operating Systems and Multiprocessing

Experiences with portable operating systems continue to be reported. The THOTH portable operating system described in ref. 1 is now comprehensively documented [70], while the papers [71–73] describe the transfer between machines of an IBM standard platform that did not have portability as an original design objective. Unix portability has already been discussed.

Operating systems, like realtime systems, require multiprocessing kernels. For portable multiprocessing using realtime threads see refs. 74 and 75. Descriptions of realtime kernels for particular operating systems, languages, or hardware may be found in specialist conference proceedings devoted to operating systems or to particular languages or hardware; some recent examples are the CHAOS operating system kernels [76] and discussions of realtime kernels for Ada [77] and Fortran [55]. Portable kernels for particular hardware include examples for transputers [78] and for IBM PCs [79]. A wide range of standard interfaces including those for an operating system kernel, communications processing, and multiprocessing are incorporated in the TRON project [80–82].

Parallel Processing

Much recent research activity has been devoted to the problem of portable programming for parallel processing and multiprocessor machines. This is a specialized area of great technical interest because of the wide range of target architectures involved. This wide range combined with limited specialist interest means that this field is relatively immature, many recent publications being case studies, such as refs. 83 and 84. Many further contributions of this kind are to be found in the proceedings of specialist conferences in such fields as super-computing, transputers, or hypercube architectures. The book [85] is a recent collection of papers on software for parallel computers that includes a number of contributions concerned with portability.

Portable parallel programming, like other kinds of portable programming, involves the identification of

suitable abstractions. Some of the proposed abstractions for portable parallel programming involve specific aspects of the problem such as memory models [86] or restricted ranges of target architectures. Abstractions identified may be incorporated into language-independent standard primitives as in the Linda [87] parallel programming environment that is becoming available commercially and can be embedded in conventional programming languages.

Alternatively, special portable parallel programming languages such as P^3L [56] may be advocated. See ref. 88 for a survey of concurrent programming languages that are independent of particular architectures.

Conclusion

The past decade of development in portable programming has been largely concerned with the definition of standard hardware and software platforms. There is now a good understanding both of the limitations of the use of such standard platforms in achieving portability and of the difficulties and costs associated with compliance with them. It may be anticipated that future developments will entail further development of such platforms and of ways of ensuring compliance with them but that underlying difficulties with portability will remain. Thus, portability may be expected to continue to be a desirable attribute of programs whose obvious benefits can rarely be achieved without significant costs for producers. Little can be expected in the way of further technical innovation in portable programming, apart from the presently active areas of ensuring compliance with portability standards for standard platforms, portable compilers, multiprocessing, and parallel processing. However, the careful documentation of portability case studies may confidently be expected to continue.

References

1. P.J.L. Wallis. *Portable Programming*. Macmillan. 1982.

2. W.M. Gentleman and S.I. Feldman. Portability—a no longer solved problem. *Comput. Syst.* 3(2): 359–379; 1990.

3. R.I. Cowderoy and P.J.L. Wallis. The transfer of a BCPL compiler to the Z80 microcomputer. *Software—Pract. Exper.* 12: 235–238; 1982.

4. D.R. Hanson. A portable input/output system. *Software—Pract. Exper.* 13(1): 95–100; 1983.

5. Ce Kuen Shieh and Li Ming Tseng. Extending a stand-alone personal computer to integrate multiple operating systems concurrently. *J. Syst. Software*, 9(1): 41–49; 1989.

6. N.E. Peeling and D.P. Youll. Past and future trends for portable tools interfaces. *Inf. Software Technol.* 31(4): 175–180; 1989.

7. F.W. Long, M.D. Tedd, and S. Heilbrunner. Evaluating tool support interfaces. Ada in Industry. *Proceedings of the Ada-Europe International Conference*, Cambridge, UK. 1988. Cambridge University Press. pp. 245–250.

8. *Application Portability Profile (APP)*. The U.S. Government's Open System Environment profile OSE/1 version 1.0. National Institute of Standards and Technology. 1991.

9. E.F. Wheeler and A.G. Ganek. Introduction to systems application architecture. *IBM Syst. J.* 27(3): 250–263; 1988.

10. D.E. Wolford. Application enabling in SAA. *IBM Syst. J.* 27(3): 301–305; 1988.

11. V. Ahuja. Common communications support in Systems Application Architecture. *IBM Syst. J.* 27(3): 264–280; 1988.

12. S.H. Goldberg and J.A. Mouton, Jr. A base for portable communications software. *IBM Syst. J.* 30(3): 259–279; 1991.

13. *X/OPEN Portability Guide: Data Management*. Prentice Hall, Englewood Cliffs, NJ. 1988.

14. *X/OPEN Portability Guide: Programming Languages*. Prentice Hall, Englewood Cliffs, NJ. 1988.

15. W.M. Gentleman. Using Harmony (industrial control OS). *Advances in CAD/CAM and Robotics: NRC Contributions*. National Research Council of Canada, Ottawa, Ont. May 1987. pp. 253–261.

16. N.E. Peeling. The Ten15 project (software development). *UK IT 1990 Conference* (Conf. Publ. No. 316), London, UK. March 1990. IEE. pp. 306–310.

17. C. Boldyreff. Unix standardisation: an overview. *UNIX: European Challenges. Proceedings of the Spring 1989 EUUG Conference*, Buntingford, UK. 1989. EUUG. pp. 151–156.

18. S.C. Johnson and D.M. Ritchie. Unix time-sharing system: Portability of C programs and the Unix system. *Unix System Readings and Applications*. Vol. 1. Unix Time-Sharing System. Prentice Hall. 1987. pp. 114–141.

19. J.E. Lapin. *Portable C and Unix System Programming*. Prentice Hall. 1987.

20. T. Plum. *C Programming Guidelines*. Prentice Hall. 1984.

21. *C Programmer's Handbook*. Prentice Hall. 1985.

22. M.R.Horton. *Portable C Software*. Prentice-Hall. 1990.

23. D.E. Bodenstab, T.F. Houghton, K.A. Kelleman, G. Ronkin, and E.P. Schan. Unix operating system porting experiences. *AT&T Bell Lab. Tech. J.* 63(8 Part 2): 1769–1790; 1984.

24. A. Filipski, L. Carey, and E. Girard. Experiences in porting Unix sdb to the M68000 processor. *Software Engineering: Practice and Experience. Proceedings of the Second Software Engineering Conference*. 1984. pp. 94–98.

25. P.J. Jalics and T.S. Heines. Transporting a portable operating system: Unix to an IBM minicomputer. *Commun. ACM*, 26(12): 1066–1072; 1983.

26. K. Howells. Portability between AIX and System/370. *Proceedings of SHARE Europe Spring Meeting*, Geneva. 1990. SHARE Europe (SEAS). pp. 321–334.

27. *IEEE Standard 1003.1–1988. IEEE Standard Portable Operating System Interface for Computer Environments*. IEEE, New York, NY. 1988.

28. *IEEE Standards Interpretations for IEEE Standard Portable Operating System Interface for Computer Environments (IEEE Std 1003.1–1988)*. IEEE, New York, NY. July 1992.

29. *Information Technology—Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) (C Language)*. IEEE, New York, NY. 1990.

30. *IEEE Standard for Information Technology-test Methods for Measuring Conformance to POSIX*. IEEE, New York, NY. April 1991.

31. D. Jones. Applications POSIX.1 conformance testing. *Proceedings of the Spring 1992 EurOpen/ USENIX Workshop*, Buntingford, UK. 1992. EurOpen. pp. 33–42.

32. C. De Groote. PCTE—A remarkable platform. *Inf. Software Technol.* 31(3): 136–142; 1989.

33. I. Thomas. PCTE interfaces: supporting tools in software-engineering environments. *IEEE Software*, 6(6): 15–23; 1989.

34. F. Neelamkavil and S. O'Shea. Interfacing Eiffel and PCTE. *Comput. J.* 35(5): A439–A444; 1992.

35. *IEE Colloquium on Standard Interfaces for Software Tools* (Digest no. 97 and 98). IEE, London. 1988.

36. H.A. Barker, M. Chen, P.W. Grant, C.P. Jobling, A. Parkman, and P. Townsend. Tailoring an application to meet two X-based user interface standards. *European X Window System Conference*, Edinburgh, UK. 1990. CEP Consultants. pp. 7–13.

37. *X/OPEN Portability Guide: Window Management*. Prentice Hall, Englewood Cliffs, NJ. 1988.

38. G. Widener. The X11 inter-client communication conventions manual. *Software—Pract. Exper.* 20(S2): 109–118; 1990.

39. S. Angebranndt P. Karlton, R. Drewry, and T. Newman. Writing tailorable software: the X11 sample server. *Software—Pract. Exper.* 20(S2): 69–81; 1990.

40. L. McLoughlin. A simple guide to porting the X Window System. *UNIX: European Challenges. Proceedings of the Spring 1989 EUUG Conference*, Buntingford, UK. 1989. EUUG. pp. 283–291.

41. K. Kimbrough and L. Oren. Clue: a common lisp under interface environment. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, New York, NY. 1988. ACM. pp. 85–94.

References

42. M. Yaacob, Khalid Mohamed Nor, and Khoo Sim Choo. On the portability of a C language Graphical Kernel System. *Int. J. Policy Inf.* 13(2): 165–178; 1989.

43. T.L.J. Howard, W.T. Hewitt, S. Larkin, C.E. Vandoni, and D.A. Duce. An interactive debugger for PHIGS. *Eurographics '90. Proceedings of the European Computer Graphics Conference and Exhibition*, Amsterdam, The Netherlands. September 1990. North-Holland. pp. 177–193.

44. J. Gorog, G. Krammer, A. Vincze, D.A. Duce, and P. Jancene. IXPHIGS: A portable implementation of the international PHIGS standard. Eurographics '88. *Proceedings of the European Computer Graphics Conference and Exhibition*, Amsterdam, The Netherlands. September 1988. North-Holland. pp. 13–19.

45. R.J. Rost. Adding a dimension to X (3D graphics for X windowing system). *Unix Rev.* 6(10): 5–6, 58–9; 1988.

46. F. Neelamkavil and O. Mullarney. A methodology and tool set for supporting the development of graphical user interfaces. *Comput. Graphics Forum*, 10(1): 37–47; 1991.

47. D.L. Brittain. Portability of interactive graphics software. *IEEE Comput. Graphics Appl.* 10(4): 70–75; 1990.

48. S.I. Feldman. Papering over deficiencies in your language for convenience and portability; preprocessors vs. standards. *International Conference on Tools, Methods and Languages for Scientific and Engineering Computation.* 1983.

49. P. J.L. Wallis. The preparation of guidelines for portable programming in high-level languages. *Comput. J.* 25(3): 375–378; 1982.

50. P.D. Gootherts and J.W. Davis. Common programming language ambiguity. *ACM SIGPLAN Notices*, 17(11): 7–10; 1982.

51. J.C.D. Nissen and P.J.L. Wallis (editors). *Portability and Style in Ada.* Cambridge University Press. 1984.

52. P.J.L. Wallis (editor). *Improving Floating-Point Programming.* Wiley. 1990.

53. B. Ford, J.H. Kok, and M.W. Rogers (editors). *Scientific Ada.* Cambridge University Press. 1986.

54. L. Hatton, A. Wright, S. Smith, G. Parkes, P. Bennett, and R. Laws. The seismic kernel system—A large-scale exercise in Fortran 77 portability. *Software—Pract. Exper.* 187(4): 301–329; 1988.

55. S.P. Kumar and I.R. Philips. Portable tools for Fortran parallel programming. *Concurrency: Pract. Exper.* 3(6): 559–572; 1991.

56. M. Danelutto, R. Di Meglio, S. Orlando, S. Pelagatti, and M. Vanneschi. A methodology for the development and the support of massively parallel programs. *Future Generation Comput. Syst.* 8(1–3): 205–220, 1992.

57. L. Ibsen. A portable virtual machine for Ada. *Software—Pract. Exper.* 14(1): 17–29; 1984.

58. A.S. Tanenbaum, H. van Staveren, E.G. Keizer, and J.W. Stevenson. A practical tool kit for making portable compilers. *Commun. ACM*, 26(9): 654–660; 1983.

59. H.E. Bal and A.S. Tanenbaum. Language- and machine-independent global optimization on intermediate code. *Comput. Lang.* 11(2): 105–121; 1986.

60. A.S. Tanenbaum, M. Frans Kaashoek, K.G. Langendoen, and C.J.H. Jacobs. The design of very fast portable compilers. *ACM SIGPLAN Notices*, 24(11): 125–131; 1989.

61. C.W. Fraser and D.R. Hanson. A code generation interface for ANSI C. *Software—Pract. Exper.* 21(9): 963–988; 1991.

62. A. Wilk and W. Silverman. Optima—a portable pcode optimizer. *Software—Pract. Exper.* 13(4): 323–354; 1983.

63. J.W. Davidson and C.W. Fraser. Register allocation and exhaustive peephole optimization. *Software—Pract. Exper.* 14(9): 857–865; 1984.

64. P.J. Hatcher. The equational specification of efficient compiler code generation. *Comput. Lang.* 16(1): 81–95; 1991.

65. P. L'Ecuyer and S. Cote. Implementing a random number package with splitting facilities. *ACM Trans. Math. Software*, 17(1): 98–111; 1991.

66. P. L'Ecuyer. Efficient and portable combined random number generators. *Commun. ACM*, 31(6): 742–749; 1988.

67. W. Kahan. Analysis and refutation of the LCAS. *ACM SIGPLAN Notices*, 27(1): 61–74; 1992.

68. *IEEE Standard for Binary Floating-Point Arithmetic.* ANSI/IEEE Standard P754. 1985.

69. W.J. Cody. MACHAR: A subroutine to dynamically determine machine parameters. *ACM Trans. Math. Software*, 14(4): 303–311; 1988.

70. D.R. Cheriton. *The THOTH System: Multi-process Structuring and Portability.* North-Holland. 1982.

71. G.E. Boehm, A.M. Palmiotti, and D.P. Zingaretti. Porting DPPX from the IBM 8100 to the IBM ES/9370: installation and testing. *IBM Syst. J.* 29(11): 124–140; 1990.

72. R. Abraham and B.F. Goodrich. Porting DPPX from the IBM 8100 to the IBM ES/9370: feasibility an overview. *IBM Syst. J.* 29(11): 90–105; 1990.

73. C. Goodrich and M.B. Loughlin. Porting DPPX from the IBM 8100 to the IBM ES/9370: migration. *IBM Syst. J.* 29(11): 106–123; 1990.

74. K. Schwan, Hongyi Zhou, and A. Gheith. Multiprocessor real-time threads. *Operating Syst. Rev.* 26(1): 54–65; 1992.

75. K. Schwan, Hongyi Zhou, and A. Gheith. Real-time threads. *Operating Syst. Rev.* 25(4): 35–46; 1991.

76. K. Schwan, A. Gheith, and H. Zhou. From chaos/sup base/ to chaos/sup arc/: A family of real-time kernels. *Proceedings, 11th Real-Time Systems Symposium* (Cat. No.90CH2933-0), IEEE Comput. Soc. Press, Los Alamitos, CA. 1990. pp. 82–91.

77. T.E. Griest and M.E. Bender. Limitations on the portability of real time Ada programs. *Proceedings, TRI-Ada '89, ACM.* 1989. pp. 474–479.

78. M. Mandal, P. Vishnubhotla, K. Eswar, C. Gollamudi, and J.A. Board. Alps on a transputer network: kernel support for topology-independent programming. *Transputer Research and Applications 2. NATUG-2 Proceedings of the North American Transputer Users Group*, Amsterdam, The Netherlands. 1990. IOS. pp. 229–251.

79. D. Raja, K. Shivakumar, S. Rao, and K.R.S. Murthy. A portable real-time kernel for 8086/80186/80286/80386 based systems on IBM-PC. *Microprocess. Microprogram.* 28(1–5): 145–150; 1990.

80. TRON. *Proceedings, The Eighth TRON Project Symposium* (Cat. No.91TH0412-7). IEEE Comput. Soc. Press, Los Alamitos, CA. 1991.

81. J.D. Mooney. The CTRON approach to operating system support for software portability. *Oper. Syst. Rev.* 26(4): 90–97; 1992.

82. T. Wasano and Y. Kobayashi. Application of CTRON to communication networks. *Microprocess. Microsyst.* 13(8): 537–547; 1989.

83. C. Lin and L. Snyder. A portable implementation of SIMPLE. *Int. J. Parallel Program.* 20(5): 363–401; 1991.

84. N. Carmichael and M. Norman. Parallel processing: the power and the portability. Experiments with 'reusable toolkits'. *Future Generation Comput. Syst.*, 8(1–3): 3–8; 1992.

85. R.H. Perrott (editor). *Software for Parallel Computers.* Chapman and Hall. 1992.

86. K. Gharachorloo, S.V. Adve, A. Gupta, J.L. Hennessy, and M.D. Hill. Programming for different memory consistency models. *J. Parallel Distributed Comput.* 15(4): 399–407; 1992.

87. M. Schollmeyer. Linda and parallel computing—running efficiently on parallel time. *IEEE Potentials*, 10(3): 43–45; 1991.

88. D.B. Skillicorn. Practical concurrent programming for parallel machines. *Comput. J.* 34: 302–310; 1991.