Report of a Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science held in Washington, D.C., on April 11–12, 1991.

Future Research Directions in Problem Solving Environments for Computational Science

E. Gallopoulos*, E. Houstis†, and J.R. Rice†

October 1992

CSRD Report No. 1259

* Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801
† Department of Computer Sciences
Purdue University
1398 Computer Science Building
West Lafayette, IN 47907-1398

FUTURE RESEARCH DIRECTIONS
IN
PROBLEM SOLVING ENVIRONMENTS
FOR COMPUTATIONAL SCIENCE

Report of a Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science

April 11-12, 1991
Washington, D.C.

E. Gallopoulos
Center for Supercomputing Research and Development,
University of Illinois
Urbana, IL 61801
E. Houstis and **J. R. Rice**
Department of Computer Science
Purdue University
West Lafayette, IN 47907

The preparation of this report was partially supported by Grant CCR-90-24549 from the National Science Foundation. This is a report to the National Science Foundation and other agencies; it is not a report by or of the National Science Foundation or any other agency.

Participants at the Workshop on Research Directions in Integrating Numerical Analysis, Symbolic Computing, Computational Geometry, and Artificial Intelligence for Computational Science

Conference Organizers

E. Gallopoulos, Center for Supercomputing Research and Development, Univ. of Illinois
Elias N. Houstis, Department of Computer Sciences, Purdue University
John R. Rice, Department of Computer Sciences, Purdue University

Participants

Robert Caviness, Computer Science Department, University of Delaware
Grant Cook, Lawrence Livermore National Laboratory
André Deprit, National Institute of Standards and Technology
Joseph Flaherty, Department of Computer Science, Rensselear Polytechnic Institute
Dennis Gannon, Center for Innovative Computer Applications, Indiana University
Keith Geddes, Computer Science Department, University of Waterloo
Luddy Harrison, Center for Supercomputing Research and Development, Univ. of Illinois
Christoph M. Hoffmann, Department of Computer Sciences, Purdue University
Moyyad Hussain, General Electric Research and Development Center
David J. Kuck, Center for Supercomputing Research and Development, Univ. of Illinois
Cleve Moler, The MathWorks, Inc.
David H. Padua, Center for Supercomputing Research and Development, Univ. of Illinois
James L. Phillips, Applied Mathematics and Statistics, Boeing Computer Services
Allan Robinson, Division of Applied Sciences, Harvard University
Jacob T. Schwartz, Computer Science Department, New York University
Siu Shing Tong, General Electric Research and Development Center
Joseph Tribbia, Climate and Global Dynamics Division, Nat'l Center for Atmospheric Research
Paul Wang, Institute for Computational Mathematics, Kent State University

NSF Observers

Kamal Abdali, Computation and Computing Research, National Science Foundation
Charles Brownstein, Directorate of Computer and Information Sciences and Engineering, National Science Foundation

Contents

1 INTRODUCTION

During the early 1960s some were visualizing that computers could provide a powerful problem solving environment ($\mathcal{PSE}$) which would interact with scientists on their own terms. By the mid 1960s there were many attempts underway to create these $\mathcal{PSE}$s, but the early 1970s almost all of these attempts had been abandoned, because the technological infrastructure could not yet support $\mathcal{PSE}$s in computational science. The dream of the 1960s can be the reality of the 1990s; high performance computers combined with better understanding of computing and computational science have put $\mathcal{PSE}$s well within our reach.

1.1 Purpose of this Report

A workshop was held in Washington, D.C., on April 11–12, 1991 to explore future research directions for $\mathcal{PSE}$s. Application areas were represented as well as four of the most relevant areas of computer science: numerical analysis, symbolic computing, computational geometry, and artificial intelligence. The goals of the workshop were:

- to describe the current state of research and development in problems solving environments,
- to indicate future directions for research,
- to assess the role and impact of problem solving environments for computational science, and
- to determine actions needed to advance the field.

This report presents the findings and recommendations of the workshop.

1.2 Background of Recent Reports and Studies

In the past decade there have been a number of reports and studies relevant to computational science that consider various aspects of high performance computing, supercomputers, computational mathematics, and scientific software. The reports listed below provide the background for the present workshop and report.

1. *Report of the Panel on Large Scale Computing in Science and Engineering*. Peter Lax, Chairman. Sponsored by the U.S. Department of Defense and the National Science Foundation, in cooperation with the Department of Energy and National Aeronautics and Space Administration, Washington, D.C., December, 1982.
2. *A National Computing Environment for Academic Research*. Marcel Bardon and Kent Curtis, Editors, National Science Foundation Working Group on Computers for Research. National Science Foundation, Washington, D.C., July, 1983.
3. *A Report of the Panel on Future Directions in Computational Mathematics, Algorithms, and Scientific Software*. Werner C. Rheinboldt, Chairman. SIAM Publications, Philadelphia, 1985.

4. *A National Computing Initiative – The Agenda for Leadership*. Report of the Panel on Research Issues in Large-Scale Computational Science and Engineering. H.J. Raveché, D.H. Lawrie, and A.M. Despain, Editors. SIAM Publications, Philadelphia, 1987.
5. *Research and Development Strategy for High Performance Computing*. Office of Science and Technology Policy, Executive Office of the President, Nov. 20, 1987.
6. *Future Directions for Research in Symbolic Computing*. Report of a Workshop on Symbolic and Algebraic Computation. Anthony Hearn, Chairman. Ann Boyle and B.F. Caviness, Editors. SIAM Publications, Philadelphia, 1990.
7. *Grand Challenges: High Performance Computing and Communications*. Federal Coordinating Council for Science, Enginering, and Technology. National Science Foundation, Washington, D.C., 1991.

2 PROBLEM SOLVING ENVIRONMENTS

The term "problem solving environment" ($\mathcal{PSE}$) means different things to different people because it is relatively immature and development has started only very recently. $\mathcal{PSE}$s of a very simple nature appeared early in computing without being recognized as such, whereas some of the $\mathcal{PSE}$ capabilities we project in Section 4 almost resemble science fiction. It is clear that whatever $\mathcal{PSE}$s eventually turn out to be, they will play a big role in the future of scientific computing and their scientific and economic impact will be enormous.

2.1 Definition of Problem Solving Environments

A problem solving environment is a computer system that provides all necessary computational facilities to solve a target class of problems. These facilities use the terms of the target class of problems and therefore can be used without specialized knowledge of the underlying computer hardware or software system. One might say that a $\mathcal{PSE}$ solves problems by communicating in the user's own terms. Solving power and problem orientation are two essential characteristics of $\mathcal{PSE}$s; other important characteristics include the following:

- The $\mathcal{PSE}$ provides: (a) State-of-the-art solution methods; (b) Automatic and/or semi-automatic selection of solution methods; (c) Facilities for easy incorporation of novel solution methods.
- $\mathcal{PSE}$s use modern computing facilities and methods, for example, interactive color graphics, powerful processors, or networks of specialized services.
- $\mathcal{PSE}$s manage the computing resources for the user, including distributed and/or parallel computing.
- Solving a problem might require long interactions with the user; the $\mathcal{PSE}$ keeps track of the problem solving task and allows the user to review it easily.
- A $\mathcal{PSE}$ is designed to create a framework that is all things to all people, solve simple or complex problems, support rapid prototyping or detailed analysis, and can be used in introductory education or at the frontiers of science.

In summary, a $\mathcal{PSE}$ can almost become a wish list for the capabilities of computers in science fiction. Nevertheless, $\mathcal{PSE}$s having some of the above characteristics have already been built while more powerful ones are being designed. We review the nature and current status of $\mathcal{PSE}$s for computational sciences in Section 3.

Three general measures of $\mathcal{PSE}$s are scope, power, and reliability. By *scope* we mean the extent of the problem set the $\mathcal{PSE}$ addresses. If the scope is small enough, then one can build $\mathcal{PSE}$s easily. For example, one can view Fortran as an early attempt at a $\mathcal{PSE}$ for elementary college algebra (the "modern" computing facilities of the late 1950's were almost fully exploited). It was a great advance compared to machine language to write

```
ANSWER = 3.71*X**(3.2*A)*(1-COS(3*PI*X)*EXP(-Y+X))
```

The *power* of a $\mathcal{PSE}$ refers to its ability to actually solve the problems that can be posed within the $\mathcal{PSE}$. Once the problem class becomes complex, it is almost certain that a knowledgeable user

can pose problems that the $\mathcal{PSE}$ cannot solve. On the other hand, there are examples of $\mathcal{PSE}$s that failed to solve even simple, straightforward problems. An extreme example is a $\mathcal{PSE}$ that purports to converse in natural language but responds "What?" to all input it does not recognize. Then its implementation might recognize only statements of the form A, B, C when A is one of "what", "who", "where"; B is one of "is", "are", and C is one of 500 nouns. Instances of such misrepresentation have occurred.

The *reliability* of a $\mathcal{PSE}$ is a measure of how often it produces correct answers. A $\mathcal{PSE}$ that responds with "unable to solve problem" is much better than one that responds with an incorrect answer. A high level of reliability may be difficult and costly to achieve, for which reasons it is sometimes neglected by $\mathcal{PSE}$ builders.

2.2 Maturation of the Field

Soon after the introduction of high-level programming languages it was realized that computers would make it possible to create powerful problem solving environments. Less than a decade after Fortran was introduced, there were many projects aimed at developing various aspects of $\mathcal{PSE}$s. The proceedings of the 1967 ACM conference, *Interactive Systems for Experimental Applied Mathematics* [121], provides an overview of early work. The title of Culler and Fried's paper, "An On-Line Computing Center for Scientific Problems" [45] indicates the high ambition for $\mathcal{PSE}$s at a time when Fortran and Algol were still novelties.

These early efforts at $\mathcal{PSE}$s failed primarily because of the lack of computing power. It was not until the late 1970's that interactive $\mathcal{PSE}$s reappeared in another context, software for personal computers. In the meantime, there was progress in creating batch processing $\mathcal{PSE}$s. Simple $\mathcal{PSE}$s for statistics (e.g., SPSS and SAS) were created because the bulk of the consumers of statistics could not or would not learn Fortran programming; they demanded a simple way to use statistical methods, and it was provided. Although the statistical systems of the 1970's seem primitive to us now, they were such an improvement over traditional programming that these $\mathcal{PSE}$s "captured" the statistical computing market [163].

The personal computers and workstations of the 1980's finally provided the computing power to realize the hopes of the early 1960's. In 20 years the mass market of computing moved from the research laboratory to the office (spreadsheets, word processors), to the home (tax preparation, education), and to services (airline reservations, banking). That $\mathcal{PSE}$s would thrive in these markets is natural; the solvers are usually simpler and less compute intensive, and the users are less able to do traditional programming. As a result, the scientists, who were the first market for $\mathcal{PSE}$s, might be among the last to enjoy their benefits. Indeed, as of this writing, $\mathcal{PSE}$s are not common in science and engineering except in limited areas, such as computer-aided design (CAD) systems for structural engineering and electronics; see for example [19]. Still, the success of Mathematica [197] and Matlab [138] shows that some other science $\mathcal{PSE}$ markets, such as computational science and engineering education, are large enough to justify the requisite investments. For a view of current developments, see recent conference proceedings [106], [107], [81] as well as Section 3. *It is a*

thesis of this report that the High Performance Computing and Communications program requires a commitment as well to the construction of scientific $\mathcal{PSE}$s.

2.3 Scientific and Economic Impact

Two goals for $\mathcal{PSE}$s are, first, that *they enable more people to solve more problems more rapidly,* and second, *that they enable many people to do things that they could not otherwise do.* Time is money, and since more people would be able to solve problems faster[1], successful $\mathcal{PSE}$s will have an important economic impact. Since people will be able to accomplish tasks more rapidly, it is natural to expect that solving several, otherwise intractable, scientifically important problems, could become feasible. In order for this impact to be realized, engineers and scientists should have $\mathcal{PSE}$s for routine as well as for non-standard parts of their computations.

It is easy to document the enormous impact of computing on science, engineering, and the economy; those who do not exploit this technology fall behind and, eventually, by the wayside. Yet it is harder to define the technology necessary to achieve this impact. Is it high performance computers? Is it the better algorithms and methods for problem solving? Is it the infrastructure of networks, languages, and programming systems? Is it environments that deliver the answers? All of these components are essential, but the importance of the $\mathcal{PSE}$ is less well recognized. For the impact of computational science $\mathcal{PSE}$s on science and engineering, see [81], [107]. For the impact of $\mathcal{PSE}$s on economic industrial activity, see [140], [163], [186].

2.4 Grand Challenges and Petty Challenges

The High Performance Computing and Communications Initiative has popularized the concept of *grand challenges* for computer science [109], and it is natural to relate $\mathcal{PSE}$s for computational science to these challenges. Because $\mathcal{PSE}$s facilitate science in general, they will be expected to contribute to meeting these challenges in many ways. Although it might eventually be desirable to create $\mathcal{PSE}$s specifically in response to the grand challenges, it should not be assumed that this should be done immediately. The nature of most grand challenges is experimental whereas the nature of the science and engineering problems for which $\mathcal{PSE}$s can be developed must be well understood and standardized. One cannot expect a powerful and reliable $\mathcal{PSE}$ in an area where no one yet knows how to solve the principal underlying problems!

Hence $\mathcal{PSE}$s are directed toward *petty challenges* as well as toward grand challenges; toward solving problems that are understood well enough so solutions are possible and are common enough so that it is important to the scientific consumer that this knowledge be codified and made available. It is practical now to create a $\mathcal{PSE}$ that an engineer can use to speed up the design of the crank mechanism for a window, the insulation for a safe, or the electrical controls of a dishwasher. These are bread-and-butter tasks of computational science which require sophisticated, but well-understood, methods. $\mathcal{PSE}$s can deliver problem solving power for most routine problems so that

1. That is, completing everything, from concept to presentation.

time and energy can be devoted primarily to non-routine and innovative aspects of a project. The scientific and economic impact of meeting the many petty challenges is diffuse but of enormous importance.

In summary, *it is a grand challenge for computer science to create $\mathcal{PSE}$s* for the petty challenges of computational science; to increase scientific and engineering productivity to be competitive in technology-driven markets.

3 CURRENT STATUS

3.1 Introduction

It is generally accepted that computational simulation has become an essential component of the scientific process, complementing theory and experiment. To place in perspective any efforts to build $\mathcal{PSE}$s, it is instructive to examine the typical problem solving procedure of a computational scientist, which includes some or all of the following steps:

1. Construction of a mathematical model of the phenomenon under study.
2. Selection of relevant physics and geometry.
3. Manipulation of equations and associated conditions, making simplifications to allow for suitable solution methods to be applied.
4. Specification of a solution method based on analytical and approximate techniques.
5. Construction of [test] problems and data sets.
6. Using appropriate specification and programming languages, specifying and creating (building or evolving from existing material) a program for the solution method. Documentation is an integral part of this step.
7. Application of the program to [test] data.
8. Validation of results.
9. Comparison of the quality of results and performance with alternative solution procedures.
10. Collection and manipulation of output data.
11. Recording of the steps of the experiment.
12. Communication of the results to the scientific community.

Observations

- Not all of these steps need to be applied, and several may be used repetitively. In general, the ordering of the steps is not strict.
- There is constant consultation with knowledge bases such as references, databases, and colleagues to exploit existing material.
- Most steps require monitoring of the quality of results and system performance. The former is critical for establishing confidence in the results, while the latter is desirable where speed is critical.
- Several decisions depend directly on the system platform of the $\mathcal{PSE}$. For example, the architecture(s) of the computer system(s) used for the experiment will influence such factors as the solution strategy and the specifications.
- An essential task is the the development and design of communication protocols, interconnection languages, etc., between the $\mathcal{PSE}$ modules.
- Several steps are currently applied in synergy with human problem solving skills such as pattern recognition and intuition. It is thus desirable to provide tools to facilitate this process.

- The problem solving process described above has an underlying hierarchical structure in that most of the steps could form entry nodes of another problem solving sequence.
- Realistic problems feature solutions that evolve on diverse temporal and spatial scales. An efficient solution method should be able to adapt itself in order to be efficient, reliable, and robust.

The problem solving steps outlined above drive the specifications of $\mathcal{PSE}$ modules.

The simplest $\mathcal{PSE}$s to date are toolkits, which rely on front-end user interfaces issuing calls to a back-end library. As components are added, the system moves closer to being applicable to the problem solving steps outlined earlier. $\mathcal{PSE}$s will be based largely on symbolic, algebraic, and numerical computing tools, artificial intelligence, expert systems, and computational geometry systems. In turn, these components will rely on "backbone" developments in hardware and software technologies. High-speed workstations, parallel architectures and software, windowing environments, graphics, high-level languages, and object-oriented programming are all examples of such critical developments.

By their nature, $\mathcal{PSE}$s are complicated and massive software systems allowing the easy manipulation of high-level objects. Their design, maintenance, and evolution will require the application of software engineering techniques such as modularization, decomposability, and information hiding, captured, for example in the context of object-oriented programming, hierarchical representation, and software reuse [31], [180].

In the next sections we discuss a few examples of $\mathcal{PSE}$s and their infrastructure components.

3.2 Three Scientific Problem Solving Environments

One thesis of this report is that problem solving environments can and will revolutionize many scientific computing activities. In fact, this has already happened in a variety of activities, as indicated in Table 3.1.

These $\mathcal{PSE}$s are not all fully developed in the way we visualize future $\mathcal{PSE}$s, but they have enough of the characteristics of $\mathcal{PSE}$s and have had a major impact on their fields. In particular, the common characteristic of these systems is that they communicate on the users terms, and they enable users to make computations easily that are otherwise either very tedious or beyond their technical capability. We present in some detail three scientific activities where $\mathcal{PSE}$s are being developed.

3.2.1 Matrix Laboratories

Matrix laboratories are good examples of the potential benefits of problem solving environments for numerical scientific computation. Systems such as MATLAB (MATrix LABoratory) [138] and CLAM [92] allow rapid prototyping and testing of new ideas. The success of matrix laboratories is due to a combination of the following characteristics:

1. High-performance computing workstations, sophisticated user interfaces and effective graphics made possible by advances in hardware and software technology.

Table 3.1: $\mathcal{PSE}$s which have revolutionized certain activities.

Activity	$\mathcal{PSE}$	Replaced
Accounting	spreadsheets	desk calculators, paper
Typing	word processors	a) retyping and correcting manuscripts b) TEXand troff
Statistics	SPSS, SAS,	a) desk calculators b) Fortran programs
Architecture and civil engineering	CAD systems	Handbooks, hand calculations and Fortran programs
Publishing	word processing, publishing programs	typesetting, manual page layout
Reservations	reservation systems	telephone/mail, large ledgers

2. A clear and simple programming language, free of several syntactic and format restrictions imposed by prevalent scientific programming languages such as Fortran 77.
3. The maturation of critical areas of numerical mathematics (e.g. linear algebra) and the development of a library of robust mathematical software.
4. Advances in software engineering, integration, packaging and other software engineering aspects of these systems. For example, the systems are open, and provide facilities for importing or exporting data and for interfacing with other systems. The systems are extensible and evolve as users add functionality specialized to their needs.

It can be argued that these characteristics are necessary for a successful $\mathcal{PSE}$ environment, a view that is reinforced by their adoption by symbolic and algebraic computing systems such as Maple and Mathematica.

The creation of matrix laboratories was greatly facilitated by the existence of mature numerical libraries such as LINPACK and EISPACK. It is also common for entire libraries specific to a particular area to be constructed and provided separately, as can be seen from the emergence of MATLAB toolboxes for signal processing, automatic control, simulation, optimization, and splines. For example, automatic control, where the matrix algebra content is well defined and small-scale problems are interesting and realistic [126], is one of the first areas where the tools are used successfully.

Matrix laboratories are expected to gain enormously as their design incorporates tools for one- or two-way interconnection with other systems. For example, MATLAB tools makes it possible (after some work) to call "foreign" Fortran or C modules from inside a laboratory session. The benefits are substantial: first, one has access to an enormous resource of existing Fortran scientific libraries. A recent example is the interfacing of PC-MATLAB with the SLICOT Fortran library (produced by NAG)

for automatic control [161], [190]. Second, it is possible to use mature compiler technology for code optimization of these modules in order to obtain high performance on the underlying computing platform [78]. Two-way interconnection is a recent enhancement which allows MATLAB to be used as a computational engine from a C program. Other tools can be envisaged, for example, that will automatically create Fortran or C output from the $\mathcal{PSE}$ language level. All these tools are of interest because they enable users to program at the levels they find most appropriate.

In order for matrix laboratories to be effective for production size problems, they must be able to handle sparse computations. This is certainly true for areas such as computational fluid dynamics, where resolution requirements demand sparse direct or iterative solvers, and is gradually becoming necessary in areas such as automatic control [127], [170]. Matrix laboratories for sparse data structures have been slow in coming, reflecting the less-developed state of the numerical library technology for sparse computations. Fortunately, as discussed in Section 3.3.2, progress has been made in that area recently to the benefit of matrix laboratories [87]. Matrix laboratories can be used now for rapid prototyping of algorithms and experiments with industrial-strength data, thus better fulfilling their $\mathcal{PSE}$ role. In conjunction with these developments, close attention must be paid to performance issues of matrix laboratories [86]. Indeed, the engineering workstation which is the common platform for the matrix laboratory may be inadequate for large industrial problems and may necessitate running at least portions of the code on very high performance platforms. Performance monitoring, modeling, and evaluation will also be necessary to explain and improve the code behavior. Parallel processing will thus become essential, also leading to scalability concerns; see Section 3.4.1.

3.2.2 $\mathcal{PSE}$s for PDE-Based Systems

Partial differential equations (PDEs) are the fundamental mathematical tools for describing the physical behavior of many application processes in science and engineering. There exists mathematical software to deal with the solution of specific classes of PDEs [26]. A number of software packages exist that are used exclusively to simulate specific applications in structural mechanics, weather prediction, and climate simulation. A partial list of major engineering packages for structural analysis and their capabilities can be found in [79]. These packages implement the finite element method, not on the PDE describing the physical problem but on the physical principles governing it. The software for the other applications mentioned above is very often based on special efficient techniques that cannot be used easily to simulate other applications. Most of these systems are well-defined, documented, and tested libraries of procedures controlled by a well-defined driver. However, a few of them already support some $\mathcal{PSE}$ functionality. In this section we review systems with some $\mathcal{PSE}$ characteristics that have a wide distribution basis or can be found in the public domain but are not tied to some specific application. Table 3.2 juxtaposes, a number of desirable features that $\mathcal{PSE}$s for PDE computations should support, and some of the existing PDE systems with a $\mathcal{PSE}$ type environment.

Table 3.2: $\mathcal{PSE}$ functionality of eight PDE systems and tools. An x means the functionality is present, a * means it is under development.

$\mathcal{PSE}$ functionality	RPI	ELLPACK	XELLPACK	//ELLPACK	DEQSOL	VECFEM	ALPAL	PDE2D
Interactive I/O	x		x	x	x	x	x	
Graphical I/O	x	x	x	x	x	x		
Multimedia I/O								
Interactive Geometry Modeling				x	x	x		
Automatic Geometry Discretization	x	x	x	x	x	x		x
PDE Language	x	x	x	x	x	x	x	x
PDE Model Generator								
PDE Solver Generator		x	x	x	x	x	x	
Advising			*	*	*			
Explaining								
Tutoring								
Navigation								
Decision Making	x		*	*	*			
Parameter Estimation	x		*	*				x
Error Estimation	x							
Interactive Debugging					x			
Interactive Run Time Control								
Symbolic/Numeric Computing	x			x			x	
Sequential Processing	x	x	x	x	x	x	x	x
Programming in the Large	x	x	x	x	x	x	x	
Vector Processing		x			x	x	x	
Parallel Processing				x				
Distributed Processing			x					
Performance Estimation		x	x	x				
Document Generation								
Portability	x	x	x	x	x	x	x	x
Openness		x	x	x		x	x	
Solution Infrastructure	x	x	x	x	x	x		x
Extendability (Generic Architecture)								
Interface to Scientific Instruments								

We now describe each of the systems in Table 3.2. The RPI system [139] is a mathematical software package for the adaptive solution of parabolic PDEs in one- and two-space dimensions by finite element procedures that automatically refine and coarsen computational meshes, vary the degree of the piecewise polynomials basis, and, in one dimension, move the computational mesh. Temporal integration, within a method-of-lines framework, uses either backward difference methods or a variant of the singly implicit Runge-Kutta methods. A high-level user interface facilitates the use of this system.

Another well-known PDE system is ELLPACK [167]. The system was designed to solve second-order elliptic PDEs in two and three dimensions and to evaluate software for such computations. It follows a modular programming paradigm which is supported by a domain-specific PDE language and a variety of elliptic PDE solvers. The PDE language interface allows the user to develop high-level programs that can be used to solve nonlinear and time dependent PDEs. XELLPACK [27] and Parallel (//) ELLPACK [104] are recent extensions of ELLPACK based on the X windowing environment. XELLPACK provides graphical input for constructing grids, pop-up menus for selecting solution techniques, and color graphics output for analyzing solutions. A user can interface with XELLPACK from any X workstation while an XELLPACK client solves an elliptic problem on any machine or machines on the network. //ELLPACK is a parallel version of ELLPACK [108], originally designed for hypercube architectures [105], [108]. It acts as an interface to various libraries of parallel elliptic PDE solvers. It allows the user to specify the PDE problem interactively (e.g. defining the region using a mouse); to use symbolic processing to transform it from nonlinear to linear form; to discretize using finite differences or finite elements. //ELLPACK automatically generates pseudo code for time-dependent PDE solvers, and it determines the mapping of the underlying computation onto parallel machines. This mapping can be displayed and modified interactively. Currently //ELLPACK provides PDE solvers based on the domain decomposition methodology, that generate code for MIMD architectures. All three systems have a facility for collecting, visualizing, and analyzing performance data.

The VECFEM system [93] solves multidimensional elliptic, parabolic, and eigenvalue functional equations on vector machines. The equations are approximated using finite elements in space (thus allowing irregular domains), and finite differences with self-adapted step size and order control in time. Parts of VECFEM are the linear equation solver FEMLIN and the matrix eigenvalue problem solver FEMEPS. Both use iterative methods of the conjugate-gradient type. The current version 1.1 does not offer a user-friendly interface but there is a plan to drive the system through a macro extension of Fortran (PATRAN) and also use the system IDEAS for geometric modeling. The architecture of the system is based on a finite element kernel defined through well-known finite element data structures and interfaces.

The PDE2D system [176] is for the numerical solution of nonlinear elliptic, parabolic, and eigenvalue PDE problems in two dimensions using the Galerkin method with adaptive meshes. The system uses PROTRAN ([9],[113]) to drive the computations and to input/output the PDE data.

DEQSOL supports finite difference and finite element discretizations of time-dependent PDEs

[122]. It includes a very high level specification language, an interactive/visual user interface for PDE problem specification, automatic generation of sequential or vector code, debugging, diagnosis, and visualization of numerical simulation of PDE problems; see also [123], [173], [189].

ALPAL is a $\mathcal{PSE}$ for several PDE-based computations [40]. Given the very high level specification of the equations to be solved and the numerical methods to be used, ALPAL generates sequential or vector code to solve nonlinear integro-differential equations, ALPAL is designed to handle the sort of complicated mathematical models used in very large scientific simulation codes. Other features of ALPAL include an interactive graphical front-end, the ability to compute symbolically exact Jacobians for implicit methods, and a high degree of code optimization.

Table 3.2 indicates that all these systems are either domain or method specific. None of them is easily expandable, and in general they lack many $\mathcal{PSE}$ features. Moreover, none of these systems has been designed to control real or experimental processes used in production or laboratories. Thus, there is need to design and implement software systems for $\mathcal{PSE}$s and their related algorithmic infrastructure for any class of PDE problems and PDE-based applications on hardware platforms. To achieve the above design objectives, one must address the issue of integration of numeric, symbolic, multimedia, and AI processing.

3.2.3 Statistical Systems

Statistics is basic to most experimental sciences, and statistical computations are the principal computations in many disciplines. The field of statistics has two characteristics which strongly motivate the development of high level $\mathcal{PSE}$s:

1. Many statistical quantities are computed by complicated algorithms which must be implemented carefully if accurate results are to be obtained.
2. Even "simple" statistical applications involve assumptions and analyses that are mathematically deep and difficult to understand, even for sophisticated Ph.D.-level statisticians.

Thus statistics was the first area of science to see the widespread use of high-level, user-oriented systems. By the mid 1970s, statistical software suppliers were introducing special languages in an attempt to allow non-statisticians to carry out statistical calculations correctly. Examples of such software came from SPSS, SAS, Minitab, BMD, and Pstat; the languages were initially user-friendly interfaces to a library of Fortran statistical subprograms. This software created considerable controversy in statistical education [163], where it was viewed as allowing students to use statistics without understanding it. By 1991 these systems were evolving into complete $\mathcal{PSE}$s, the use of elaborate graphics was commonplace, and expert systems help (which this community of users needs particularly badly) was being developed and introduced.

3.3 Component Areas

3.3.1 Symbolic and Algebraic Computing

Examples of symbolic and algebraic computing systems (SACs) are MACSYMA [70], [132], REDUCE [97], [98], Maple [37], Scratchpad II [115], DERIVE [181], and Mathematica [1], [197]. Reference

[99] provides several interesting observations on the development of SAC systems in relation to other areas of computer science. A recent important report summarizes a wealth of information about current and future applications of SAC technology [100].

SACs can help in the early problem solving steps of specification and model creation. They can perform analytical manipulations before the application of numerical techniques; these manipulations are useful but also tedious and error prone if done manually. This preprocessing leads to better understanding of the mathematical problem and important simplifications ([84], [174]) and selection of proper solution procedures [61]. SAC systems provide the framework for describing equations and translating them into a suitable format for manipulation in subsequent phases. Systems have been built for automatically writing code to solve elliptic differential equations in general coordinates based on finite-difference/finite-volume approximation; for time dependent problems [64], and for generating finite element code [76], [184], [193]. Another use of symbolic algebra tools for $\mathcal{PSE}$s is in stability investigations of finite-difference approximations to differential equations [65], [83], [185].

An important use of SACs is in the generation and manipulation of Jacobian matrices during the solution of nonlinear systems of equations. Hence SACs as well as recent automatic differentiation techniques can replace error-prone hand manipulation or finite difference oriented schemes contaminated by roundoff[2]; see [41], [139], [191] plus articles and references in [91].

By providing the framework for specifying the mathematical problem in a manner close to the standard scientific notation, SACs approach satisfying an important PSE function, that is, providing communication in the user's own terms. As SACs accumulate mathematical knowledge, they can also replace mathematical handbooks, and even more ambitiously, to become the mathematician's assistant. Significant research questions remain in the development of good SAC systems; see [69]. Several issues must be addressed before SACs can fulfill their role as components of PSEs, namely increased speed of operation by exploiting multiprocessing technology; reliability; and design of the user interface.

Regarding reliability, we note the potential dangers, convincingly demonstrated in [71], with an example in which variables in SAC-generated formulas were substituted with floating-point numbers, producing numerical values less accurate than if one were to use approximate methods from the start.[3] Furthermore there is a need for SAC software libraries. One step in that direction is the NETLIB-type organization and distribution of software for REDUCE.

Another important topic is that of the interface of SACs with users and other systems. Examples include sophisticated graphics, output of Fortran code or LaTeX expressions, and connections with foreign environments [76], [117], [182], [184], [193].

The data structures and computations used by SAC systems are very demanding of the computer system, so that the potential of parallel computation should be exploited. In the same

2. In the words of Painter and Cook in [41], "Before ALPAL, 100% accurate Jacobians were unheard of at LLNL."
3. This is a very worrying result, given the current use of SACs to produce extremely long mathematical expressions, translate them into Fortran, and use them with floating-point arguments.

time, current research tries to address the issues of computational efficiency, expressiveness, and friendliness of SACs [47], [69].

Currently, several companies are relying on SAC systems for complicated industrial tasks ranging from the design of three-dimensional elements to simulate singular behavior in stress fields [110], to studying seismic wave propagation [119], conducting reliability analysis for off-shore oil rigs and nuclear reactors, and complementing finite element methods in the design of wind turbines and next-generation engines.

Overall, we note that by their very nature, SACs come very close to the $\mathcal{PSE}$ idea; see for example [70], [182, Section 8.2.5]. It is thus important to study the design issues raised by SACs as they encapsulate most of the problems that need to be solved for the realization of PSEs.

3.3.2 Numerical Analysis

The primary contributions of numerical analysis consist in the development of efficient and robust algorithms and their realization into numerical libraries whose fundamental role has been advocated since the early days of scientific computing; see articles in [43] and [56], [152], [162], [165], [166].

As problems increase in complexity, the presence of reliable, efficient and easily assembled software parts becomes essential [168]. High-quality numerical libraries allow the user to concentrate on the higher level issues instead of rewriting software [103], [112], [142]. Public depositories (e.g., NETLIB [57]) provide ready access to good quality numerical and other scientific software. Currently, there is intense research and development activity in algorithms and libraries for direct and iterative sparse computations [11], [14], [15], [52], [53], [54], [60], [82], [144], [172], [171], [175]. Some libraries are developed even further into complete environments, (for example, using graphics and interface languages), bringing them closer to $\mathcal{PSE}$s. That these efforts are of importance to the scientific community is evidenced by the success of matrix laboratory tools such as those described in Section 3.2.1; see also [11], [171].

As argued by B. Parlett in [147], numerical analysis research, is not only directed toward solving state-of-the-art problems but also toward re-evaluation of existing solution methods in light of new developments. For example, novel computer systems have triggered research in algorithmic techniques to exploit vector and parallel computational resources, hierarchical and distributed memories, etc. See [82]. Libraries based on such techniques are already under construction and standardization; see for example [13].

It is hoped that $\mathcal{PSE}$s will significantly reduce the present delay in applying and testing novel numerical algorithms in the context of real applications as well as simplifying the design of appropriate test problems. Indeed, the lack of adequate "real" data sets and good reporting procedures has been recognized as a serious impediment to research in many subfields of numerical analysis [22], [114]. Although efforts are being made to construct collections of test data, $\mathcal{PSE}$s offer a natural solution to this problem.

Reliability of results is another important concern. Just as one seldom questions the reliability of results obtained with trigonometric functions, the $\mathcal{PSE}$ user should be able to rely on intermediate

results when using components of the environment; this assumes a high degree of confidence for these routines, demonstrates the necessity for numerical algorithms in a $\mathcal{PSE}$ to be reliable, and indicates the importance of current research in error estimation and control, adaptive algorithms and software for the complex problems to which $\mathcal{PSE}$s will be applied [68], [77], [143].

3.3.3 Artificial Intelligence

Techniques for efficient problem solving constitute an important topic of artificial intelligence (AI) research [12]. Expert systems constitute a major aspect of AI with respect to problem solving tasks.

From early on, polyalgorithms and automatic algorithm selection procedures were recognized as important to the development of efficient and reliable numerical software [164]. With the proliferation of solution methods, it becomes clear that the selection process should be largely automated. See [25] for a review of one such project (GAMS) and references to others (NAXPERT, NEXUS, SLADOC). In areas such as civil engineering, knowledge based systems are combined with CAD tools to improve the overall design process; see [16] and [24]. Expert systems have been developed for differential equations. We note references [18], [118], [131], [149]; Elliptic Expert [62] for the XELLPACK environment [27]; ATHENA for //ELLPACK [105]. Successful use of AI techniques for automatic preparation, execution, and control of numerical experiments has been reported in [2]; other useful references include [32], [38], [44], [95], [187], [81], [106], [107]. It must be noted, however, that the feasibility of constructing systems able to handle general PDEs has still to be demonstrated.

3.3.4 Computational Geometry

Geometry is a critical component for most applications. The almost exclusive use of single rectangular or circular slopes in textbooks clouds the fact that most applications really involve somewhat more complicated shapes. For example, computer-aided design in structural engineering is based on interaction of solid modeling, finite element mesh generation, solution and postprocessing [73]. The structural engineering community has developed a methodology for a wide range of shapes, the "building block" approach (i.e., finite elements or constructive solid geometry), which is quite effective for many applications. On the other hand, it is not as effective when smooth shapes are essential to the applications. More distressing to those trying to build versatile systems, is that most of the geometry manipulation capability is deeply buried within massive software systems. There have been efforts recently to create geometry systems that can interface naturally with various levels of application, for example, the Protosolid system [192]. There are also important efforts to provide "design shells" for large structural analysis systems (e.g., the commercial products IDEAS and Adams), which provide more natural and simple-to-use geometry as well as other benefits.

It is essential that computational geometry be integrated into software environments of computational engineering and science for the 1990's. This task requires efforts on several layers. Beginning at the infrastructure level, geometric modeling systems pose many research problems in the integration of numerical and symbolic computation and in the practical application of theories

from geometry and algebra [102]. On the systems integration level, geometric modeling systems need to be restructured into open systems that give freely access to infrastructure functionalities and provide tools for interfacing with complex interval data structures. On the user-interface level, finally, the traditional geometric design gestures and paradigms need to be rethought from an applications point of view that incorporates into the need for specifying shape, the additional need to specify visually, and conveniently the parameters of the physical problems to be analyzed. Success on this level will require melding different research communities.

Architects and civil engineers have been investigating CAD environments, combining knowledge-based engineering, computer graphics [89], geometry and solid modeling, and design optimization, for some time [72], [159]. It is argued that future architectural $\mathcal{PSE}$s could free CAD from its current restrictions [141] and enable users to explore completely innovative designs [136], [137].

3.3.5 Visualization and Graphics

Visualization is an integral part of $\mathcal{PSE}$s. The case for a visualization initiative was made in [39] so that advances in numerical simulation software/hardware environments can be matched by an improved ability to assimilate the results. For a review of recent activities in the area see [30] and [111]. E. J. Farrell's preface to the latter collection summarizes the importance of visualization in scientific computing. It is noted that graphics, images and presentation of information in image form are essential in the development of science and engineering. For example, they allow viewers to perceive patterns and relationships which may be missed in tables of numbers. It is also noted in the same reference that in addition to forming a three dimensional view of data, a goal is to provide tools and systems which allow the user to extract information from the data. Traditionally, visualization techniques are primarily applied in the pre- or post-processing of the solution process. An important recent advance includes the use of visualization tools to depict sparse matrices in the context of matrix laboratories or other numerical libraries [87], [146], [188]. Often, there is a need to observe and steer the computation during run-time; see for example [28] and [94] and articles in [183]. Furthermore, the introduction of parallel computing and its realization on varied parallel architectures has necessitated the collection of run-time data that show the performance and flow of parallel computations; see for example [183]. Graphical representation of these data is the only way to perceive changes and take appropriate actions. We predict that future $\mathcal{PSE}$s will allow users to visualize their computation.

The symbolic representation of three dimensional post-processing input data is impractical. Already, CAD systems have revolutionized the way we specify such data. The integration of graphics to specify the physical world and support the simulation process is one of the main $\mathcal{PSE}$ design objectives. In summary, the area of visualization should be an important research component for $\mathcal{PSE}$s.

3.4 Supporting Areas

3.4.1 Parallel and Distributed Computation

The capabilities of supercomputers have made possible numerical simulations at a fine level of detail (e.g., using the additional memory and computational power to increase resolution) with corresponding increases to the sophistication of the models. Parallel and distributed computation will affect research in most areas, and with true multiprocessing and large memories, it also becomes possible to attempt the parallelization of symbolic computations [48].

Still, the effective use of parallel computers for several computations is an active research topic. For example, many numerical computations deal with sparse data structures, incorporate adaptive algorithms, etc. The efficient implementation of such algorithms on parallel architectures is a hard problem [23], [139], [196]. Some of the difficulties are similar to those encountered when dealing with the parallelization of symbolic computations. As massively parallel architectures mature, studies of scalability for mathematical and scientific libraries, software tools, and communication and I/O libraries should also intensify [130].

As the user searches for the best algorithm for his particular application, he will be faced with algorithms that tackle the same problem but perform differently, depending on the input data. Adding computer architecture as a parameter opens the field to many new approaches, augmenting the algorithmic choices and constraints. The explosive growth in the set of possible solutions makes expert systems necessary.

3.4.2 Networks

Some $\mathcal{PSE}$ components, (e.g., the knowledge base) could be geographically distributed. High-speed networks and electronic mail would enable users to obtain resources from remote facilities and post inquiries to electronic bulletin boards. The proposed National Research and Education Network (NREN) component of the Federal High Performance Computing and Communications Initiative (HPCC) addresses these areas as it is designed to support the bandwidth required for interactive visualization, file and image transfers, multi-media database access, teleconferencing, and collaboration technology [109], [6]. Some examples relevant to the previous discussion on SAC and numerical systems are the electronic dissemination of information (source code, bibliographies, news) for REDUCE (organized by A. Hearn at RAND), the use of X window technology and Unix tools for ready access to NETLIB [58], and the proliferation of resources accessible via anonymous file transfer over Internet.

3.4.3 User Interface

An area of great importance for the creation of successful and usable $\mathcal{PSE}$s is the field of user interfaces. This is expanding rapidly, an expansion which is largely due to the increasing expectations and demands from the users, the availability of generic software platforms for the development of user interfaces, and the emergence of new input/output technologies [20].

A great many engineers, scientists, and students are familiar with the sophisticated iconic interfaces such as that of the Macintosh and various window systems. These users expect such interfaces to be readily available to engineering and scientific software systems. Unfortunately the cost of providing these interfaces is still high; after the user interface code can be as much as 70% of the total code of a software system.

The range of technologies available for user interfaces is growing rapidly. Apart from today's bit-mapped graphics, other, more exotic interface technologies such as virtual reality and multimedia have been developed. They are already available commercially and shortly they will become inexpensive enough to be readily available. Some interface system requirements for problem expression, automatic programming, visualization, computational steering, and concurrent computing are discussed in [18], [150].

It is expected that the future $\mathcal{PSE}$s will not only assist the modeling and simulation of a particular application but will be used as job simulators or components of process control systems. In any case single-media interfaces have already begun to show serious shortcomings in effective information display. It is likely that these shortcomings can be overcome by spreading information processing across different modal channels. There is, therefore, a hope that multimedia technologies can address the issue of information overload in the user interface of $\mathcal{PSE}$s.

To use these new technologies we must support research and development in the design construction and evaluation of a multi-media tool set which provides facilities for constructing, executing, and emulating multimedia interfaces. There are already some examples of such tool sets [10] especially for process control applications.

3.4.4 Software Infrastructure

$\mathcal{PSE}$s must enable the computational scientist to program in the large. As managers of complexity, $\mathcal{PSE}$s, their component subsystems, and their design targets are also expected to be large and complex. The software infrastructure needs to be developed and complemented in case current techniques are inadequate.

Object-Oriented Design. Object-oriented techniques will be useful for rendering $\mathcal{PSE}$s comprehensible and manageable [31], [7]. $\mathcal{PSE}$ components are expected to cooperate in problem solving and would benefit from research in concurrent object-oriented programming [8]. In particular, object-oriented programming techniques are gradually becoming more common in computational science [35], [129], [138], [145], [149], [160].

Software Interconnection Technologies. The integration of complex numerical and symbolic systems needs appropriate software interconnection technology and module interconnection languages for the efficient description and control of problem solving. When code modules are written to solve PDEs, interfaces must be written to link with general purpose numerical software and application codes [42]. A *software bus* could provide the appropriate connecting infrastructure [157], [156],[158]. Some design goals for the bus are to allow programs to be described and manipulated in terms of minimum specifications and to provide a language for describing module interfaces in

a manner that is independent of the application's implementation language; see [55], [155], [157], [158].

Recent systems have begun to address the interconnection problem. As noted in Section 3.2.1 matrix laboratories are built on top of sophisticated mathematical software libraries. Tremendous power is added as it becomes possible to link the systems with numerical libraries written in Fortran or C. There exist several projects for interconnecting SACs with numerical software libraries (e.g., IRENA and INTERCALL to link REDUCE and Mathematica with the NAG library [33], [50], [51]).

Language and Compiler Technology. The important role of language in the problem solving process is widely recognized [29], [179]. Some of the systems described earlier already provide their own language. New languages are also being proposed, some specifically directed toward scientific computation while object-oriented languages such as C++ gain currency in the scientific computation area [59], [101], [128], [133], [134], [63], [169], [198]. The development of compilers and other tools for these languages and their implementation on target architectures is another important activity; see [120] and articles in [85]. One criticism of symbolic systems is that they are slow. The reasons for this are manifold. Whereas most numerical computations are based on the iterative manipulation of regular data structures, thus allowing loop distribution across processors and regular sequences of memory access patterns, symbolic systems manipulate irregular and dynamic data structures. In addition, SACs are frequently written on Lisp-type languages for which restructuring compiler technology is much less developed. While research for the discovery of better SAC algorithms is continuing, improvements in speed and usability are expected as good compilers for the underlying languages become available [153], [154]; examples are the parallelizing compiler for sequential Scheme [96]; multiprocessing extensions for Lisp [88], [199]; the effort of [75] for constructing a compilation-driven parallel REDUCE system for loosely coupled, distributed architectures. There have also been efforts to provide implementations of Lisp-based systems such as REDUCE in C by building a translator from REDUCE source to C [74]. See also [36] and [195] for additional work on the multiprocessing of SACs.

As most scientific/numeric processing is done with Fortran, much time in symbolic systems is spent in special functions (e.g. GENTRAN) to generate Fortran code. It thus becomes crucial for performance for such functions to produce code which is, in some sense, optimized. This topic has received attention, with some systems performing optimizations over entire code sequences in order to obtain improved scheduling on vector processors.

Future research should examine how to exploit the $\mathcal{PSE}$'s high-level knowledge of the problem to enhance the compilation process and produce better solutions. In addition, recent developments in computer architecture have underlined the difficulties of restructuring dusty-deck codes for full exploitation of parallel machines. There is still a lot to be done in developing parallel software engineering principles for writing codes that demonstrate performance portability. This is an important task, since it can be argued that codes designed on such principles will receive the maximum help from automatic restructurers.

3.5 Domain-Specific Problem Solving Environments

Discussion between users and $\mathcal{PSE}$ developers should be an active component of the $\mathcal{PSE}$ design process. The development of $\mathcal{PSE}$s is envisaged as a collaboration between $\mathcal{PSE}$ developers and applications scientists; otherwise one risks building interesting but "toy" tools. In particular, we note the comment of M. Dertouzos that too much computing has been of a "throw the goods over the fence" type[4]. Thus, it is an important goal of $\mathcal{PSE}$ research funds to support a constructive dialogue between designers and users, with the latter group having easy access to the system and motivation to criticize and use it while it is being designed and developed. Progress can be achieved by building $\mathcal{PSE}$s around selected areas, for example, continuum mechanics or computational electronics, including functionality for as many steps as possible from those described in the introduction to Section 3.

Several environments which incorporate some of the characteristics outlined above were already reviewed in Section 3.2; see also [25] for some comparisons. Some environments are specialized to particular problem domains: for example, in the areas of industrial engineering design [186] and structural mechanics, combining solid modeling, finite element mesh generation, solution, and postprocessing in [73], [148]. $\mathcal{PSE}$s have also been created for pure mathematics (group theory [34]); partial differential equations [17], [139]; general relativity (SHEEP [80]); and numerical analysis and control of precision of arithmetic calculations (ACRITH [125], AQUARELS [67]). EVE [18] is an object-centered knowledge-based PDE solver, constructed around the MODULEF environment. A system, built on top of Mathematica and automating several problem solving steps from specification to code generation, is SINAPSE [119]. The primary application domain of SINAPSE is seismic wave propagation using finite differences and explicit or implicit time stepping. Another important effort, spanning many years of development, is in building P-FINGER, as system for automating finite element analysis using symbolic and numerical techniques and mapping onto shared and distributed memory multiprocessors [177], [178], [194]. Much progress has been made in the area of $\mathcal{PSE}$s for electronic CAD. See for example [19] for a discussion of *CAD Frameworks*, a term which means "... all of the underlying facilities provided to the CAD tool developer, the CAD system integrator, and the end user (IC or system designer) which are necessary to facilitate their tasks".

Specialized $\mathcal{PSE}$s will also have an important impact on education, allowing students to experiment with hard problems and sophisticated solution methods.

We believe that the backbone developments, the equally impressive developments in individual component areas, and most important, the needs of the working scientist, constitute the objective and subjective conditions necessary for the creation of viable $\mathcal{PSE}$s. The conditions are now ripe for the integration of these tools into $\mathcal{PSE}$s and specialized workbenches, in order to create a more productive environment for the scientist.

---

4. It is counterproductive to build a house by having the builder bring truckloads of materials, letting the future occupants decide if they are appropriate; see[49], [90].

3.6 Professional Infrastructure

The educational infrastructure for scientific problem solving environments is not strong. We use the term Computational Engineering and Science (CES) to denote this discipline and area of work. Too often we see highly trained engineers and scientists in CES whose knowledge about computing is that of a college sophomore and highly trained computer scientists whose knowledge about engineering and sciences is at that same level; traditional educational programs in each of these areas stop at the sophomore level (or earlier) in the other area. Thus education in the "other" area tends to be ad hoc, on the job, and self-taught. For computer science, this means that it is hard to find traditionally trained computer scientists who know enough about engineering and science to understand CES applications. Faculty working in CES areas find that their Ph.D. students have often spent a year either learning about application areas or a year passing courses and exams in topics weakly related to CES (e.g., abstract algebra for mathematicians, power systems for electrical engineers, theoretical CS for computer scientists).

A few CES programs have risen out of a desire to remedy this situation. The common thread of these programs is that there is both substantial computer science and engineering/science content. There is a wide variation in the specific nature of the programs because they must be adapted to local faculty interests and university political structures. All involve more than one department, and most involve a computer science department. It is indicative of the situation that at some places one can create a CES program and find no one in the computer science department interested in it. Ideally, one wants the students to learn most of the material from two disciplines. This is an unreasonable load for the students, so there are hard choices about what material to include. Most of the CES programs are at the graduate level, where flexibility in tailoring education programs is common.

Six academic CES programs are described briefly below. These descriptions are adapted from material provided by Robert Funderlic (North Carolina State), Gene Golub (Stanford), Bill Martin (University of Michigan), Gary Rodrigue (University of California, Davis), Ahmed Sameh (University of Illinois), and Danny Sorensen (Rice University). The descriptions illustrate both the diversity of the programs and the common purpose of combining computer science, engineering, science, and applied mathematics in some way.

3.6.1 University of Michigan

The doctoral program in Scientific Computing at the University of Michigan is a joint degree program — students pursue doctoral studies in a home department, typically one of the traditional engineering, science, or mathematics disciplines, and take additional courses in areas such as numerical analysis, scientific computation, applications, or the study of algorithms for advanced computer architectures. This program is based on the recognition that a firm knowledge of the science is an essential ingredient for research in scientific computation — students are expected to complete the normal doctoral requirements for their home departments as well as additional course requirements in scientific computation, numerical analysis, and algorithms for advanced computer architectures. The title of the degree has "and scientific computing" appended to traditional

description, for example, Ph.D. in aerospace engineering and scientific computing.

The Laboratory for Scientific Computation administers the doctoral program in scientific computing, is cooperation with the student's home department. The following list of research topics is representative of this program:

Computational fluid dynamics	Simulation of VLSI circuits
Algorithms for new architectures	Scientific visualization
Computational particle transport	High performance materials
Computational solid mechanics	Molecular dynamics
Simulation of semiconductors	Computational chemistry
Simulation of AIDS transmission	Computer-aided molecular design

3.6.2 North Carolina State University

Strong local institutional support and excellent faculty from several departments have propelled CES programs at North Carolina State University. A plethora of shared memory and message passing parallel computers is available for researchers and graduate students on campus and at the North Carolina Supercomputer Center at Research Triangle Park. The Center for Research in Scientific Computation (joint between computer science and mathematics) acts as the focal point for academic CES programs. The Computer Systems Lab (joint between computer science and computer engineering) provides strong computational infrastructure support.

Various names and emphases describe the academic programs at North Caroline State: *Computational Mathematics* (CMA) within the Mathematics Department, *Scientific Computing* (SC) within Computer Science, and *Computational Engineering and Science* (CES). The latter resembles a well-structured, expanded, split minor in math and computer science and is available in all engineering and physical science graduate programs. Computer science is a vital component of the research and teaching of scientific computing at NC State; for example, of the 23 courses that support the CES program, 18 are computer science, with 8 of these cross listed with mathematics. The SC and CMA programs are very similar and lead to M.S. and Ph.D. degrees in computer science and applied mathematics. With the proper advising, a de facto track in scientific computation is available within the computer science undergraduate program.

North Caroline State's success in establishing CES programs has been strongly influenced by the cooperative efforts of their computer science and applied mathematics faculties even though they are in different colleges.

3.6.3 Rice University

The Mathematical Sciences Department, in conjunction with the Computer Science, Chemical Engineering, and Electrical Engineering Departments, has initiated a new degree program leading to advanced degrees in Computational Science and Engineering (CSE). The program focuses on modern computational techniques and is designed to provide this training throughout Rice University

at the M.S. and Ph.D. levels. The program is governed by a committee of faculty chosen by the Dean of Engineering, with ultimate oversight by the Provost. This Computational Science Committee (CSC) is responsible for assisting the student in designing an appropriate course of study, setting examination requirements, and insuring the integrity of the degree program.

The professional master's degree produces an expert in scientific computing who can work as part of an interdisciplinary research team. A recipient of this degree will be well trained in state-of-the-art numerical methods, high performance computer architectures, software development tools, and in the application of these techniques to at least one scientific or engineering area. The curriculum for this degree consists of a variety of topics from mathematical sciences, computer science, and a selected application area. Requirements include successful completion of 30 semester hours or more of advanced courses. There is no thesis requirement.

The Ph.D. program starts with advancement to doctoral candidacy by the successful completion of a program of approved course work along with satisfactory performance on preliminary and qualifying examinations. The foreign language requirements of the student's department are adhered to, and the student completes an original thesis under the direction of a member of the participating faculty of the CSE program which is acceptable to the Computational Science Committee.

3.6.4 Stanford University

In 1987 Stanford established a degree program in scientific computing and computational mathematics. Its purpose is to train students in the use of modern advanced computer architectures and software tools in various fields of science and engineering. The main thrust is the fusion of ideas from computer science and applied mathematics with a number of application areas. This program resides in the School of Engineering, and students are admitted directly into the program independent of other departments. The program's faculty is made up of faculty from other departments and has three levels of participation. The Core Faculty is responsible for administration; the Associate Faculty consists of people who are heavily involved in computing within their discipline and who offer courses within the program; the Affiliated Faculty are those whose disciplines are peripherally dependent on computing.

The curriculum emphasizes applied mathematics, numerical analysis, and computer science and requires demonstrated expertise in some application area such as fluid mechanics. In addition, there are working relationships with local research organizations such as RIACS, LLNL, and IBM.

3.6.5 University of California at Davis

A program of computational science has been initiated within the departments of applied science and chemistry. Questions of science, computational techniques, computer science, and mathematics are inseparable in addressing the large issues in computational science. A practitioner of computational science must have some skills in each of these areas and be able to interact from each of them. The computational science program at U.C.–Davis was established with this philosophy in mind for the graduate student who is interested in the application of computers to the physical, chemical,

mathematical, and engineering sciences. The program involves course work from the traditional areas of physics, chemistry, computational mathematics, and computer science as well as in the area of the student's specialization. Ph.D. candidates in participating departments declare a designated emphasis in Computational Science, then proceed to take a special set of core courses in the department in which the student is enrolled and also a set of core courses in computational science. For example, in the Department of Applied Science, the core courses are Mathematical Physics, Computational Mathematics, and a course called Computational Science that is designed especially for physical scientists and which covers such topics as computer architecture with emphasis on parallel computers, algorithms, and numerical methods. After passing an examination, the student proceeds to their graduate research by taking electives from a variety of available courses within the department. The degree awarded to the student is: "Doctor of Philosophy in '(department)' with emphasis in Computational Science".

3.6.6 The University of Illinois at Urbana-Champaign

A new graduate program in computational science and engineering (CSE) is under consideration by the College of Engineering at the University of Illinois. Unlike what is now regarded as traditional computer science, a CSE program focuses on the whole computational process. It covers the following topics:

(I) Computers
  (i) architecture for parallel and pipeline processing,
  (ii) simulation from the chip to the system level,
  (iii) hardware to the level of device simulation and packaging,
  (iv) reliability and fault tolerance;

(II) System Software
  (i) compilers, especially restructuring source code and code generation,
  (ii) programming and problem solving environments,
  (iii) operating systems, including interface with compilers, scheduling and dynamic control of systems;

(III) Applications
  (i) design of robust parallel numerical and nonnumerical algorithms,
  (ii) specialization in one application area such as digital circuit simulation, computational fluid dynamics, computational chemistry, etc., and the development of application software that achieves "performance portability" across a wide class of architectures;

(IV) Performance Evaluation
  (i) measuring performance of existing and proposed architectures, compilers, algorithms and whole application codes,
  (ii) analysis and performance improvement suggestions, and validation via measurements.

4 FUTURE RESEARCH DIRECTIONS

It has been predicted that by the beginning of the next century the computer technologies of the 1990s will allow anyone with access to computers to get an answer to any question that has an answer. On the other hand, it has been rightly observed that if someone has only a hammer, then everything looks like a nail to him. The research directions for $\mathcal{PSE}$s should be governed by the desire to make the above prediction a reality and to provide students, scientists, and engineers with problem solving environments and computational power that will make them feel that their only limitation is their imagination.

4.1 Future Problem Solving Environments

The enabling technology for future $\mathcal{PSE}$s is the wide availability of high-performance computers. It is expected that in the 1990s we will see on-chip processing performance in excess of 2000 MIPS and scalable parallel processors containing thousands of such chips. The palmtop (e.g., HP 95LX!) and notebook computers will become as powerful as current workstations. The new generations of workstations will be able to process heterogeneous information at supercomputer speeds, utilizing hundreds of megabytes of main memory, large (greater than 45 inches) flat, high resolution displays, and very large optical and/or magnetic disks. We will see high bandwidth local area networks, wireless communication systems, and laptop computers as routine parts of cellular communication systems. It is widely recognized that the workstations of the 1990s will be able to process multimedia information (i.e., voice, programmed sound, video, photographs, 3D images), which will provide support for the development of new tools that will take advantage of the added value provided by the combination of the "traditional" computer media, existing information systems, and digital video and sound technology. It is clear that we have not yet thought of everything we can do with this technology.

These technological advances are bound to have a significant impact as the way we learn, solve problems, communicate, and interact professionally and personally. "Programming" in such an environment will almost certainly be a different task from what it is today. We anticipate that for most users, programming will be an activity involving high level, interactive, visual-object-oriented languages, supported by multimedia libraries of information and application objects. Traditional algorithmic programming will be more restricted to specialists and system builders. This ability to "program-in-the-large" will be an important feature of the $\mathcal{PSE}$s we envisage.

User interfaces for computational science.

The new technologies will definitely change the way we communicate with electronic media and determine the nature of $\mathcal{PSE}$ interfaces. Most of the interfaces today are tool based and user directed. We need interfaces that support integrated environments capable of organizing the user's computational objectives instead of having the user organize computations piece by piece. Future $\mathcal{PSE}$ interfaces will be connected to more than a trillion objects of useful knowledge. It is unclear whether the current direct human interfaces can handle this workload. Furthermore, future $\mathcal{PSE}$

interfaces will have animation and multimedia processing as a basic capability. The development of $\mathcal{PSE}$s depends very much on interface technologies, and thus research and development of user interfaces for computational science applications is essential.

Software infrastructures for "soft" laboratories.

Computational models have augmented, or even replaced, real experimentation in many areas and now play a significant role in everyday science and engineering. We foresee as a future important research direction the development of a "soft" computational science laboratory where hybrid computational and experimental models interact in a natural way. The goal may be to create a prototype model of some artifact or process that is still in its infancy, or to "surround" an experimental physical device with a simulated physical environment, or simply to exploit the economic advantages of various component types in a process. Although the need for $\mathcal{PSE}$s for such soft laboratories has been identified, no specific architecture has been suggested yet. In this situation one has to face the additional challenge of interfacing software and physical $\mathcal{PSE}$s. Multimedia workstations constitute an initial reasonable step toward the realization of such laboratories which are certain to have a significant positive impact on education and research in science and engineering.

Performance, portability, and extensibility are issues of which the $\mathcal{PSE}$s must be cognizant. It is critical for the $\mathcal{PSE}$ to be be designed so that it matches the users' various levels of ability. The desirability of this feature for future $\mathcal{PSE}$s has been mentioned in the literature; see [124] and [19]. In addition, environments must be able to support "adaptation" to the user's changing needs and resources. Some important problems and a vision for future environments are described in [124].

The design and development of the appropriate software infrastructure to create such "soft" laboratories is an important future research direction.

Expert systems for problem solving.

Applications will no longer be supported by single-minded, deterministic algorithms that require several parameters to be specified by the user. Instead, we will see the development of polyalgorithms, and "smart" algorithms capable of adapting themselves to specific situations. In addition to their computation procedures, these algorithms include knowledge about their applicability and perception of their algorithmic and computational behavior on various hardware platforms. The creation of this new breed of smart or expert systems for problem solving is one of the key research directions.

We believe that the algorithmic/hardware/software advances of the 1990s will be able to support the vision of the 1960s. *The challenge is to create the software to exploit and integrate these technologies*. The goal is to support well-established educational and problem solving processes. This challenge requires that advances be made in infrastructure technologies such as domain specific languages and compilers, interfaces to support integrated environments of multimedia objects, libraries of "smart" objects, transparent use of complex computer architectures, and generic, transportable kernels of capabilities.

4.2 Generic Problem Solving Environments

For many years the brainstorming of scientists and engineers has been supported by a simple generic tool: some combination of a paper notebook, a blackboard, a calculator, a pencil, and chalk. It is now possible to dream of replacing this tool with a single electronic medium capable of supporting small-scale symbolic, numeric, and graphical processing of certain objects, typically mathematical, while large-scale, detailed computations are deferred to a more powerful computing engine. The notebook computer is already a reality and soon will be commercially available. *The research challenge is to scale down the existing tools to fit this hardware platform and interface them to this new environment.*

Another problem solving process is the synthesis of a suite of well-understood operations from well-defined libraries. A recurring dream of the practitioner is a well-organized "workbench" of "smart" software tools capable of assisting in the selection and synthesis process and of hiding most of the non-application specific operations. More is needed than the limited ongoing research in knowledge base front ends for existing well-defined libraries. Experimentation with various software architectures is a task that requires the collaboration of specialists in computer systems, software engineering, human interfaces and computational science. *A research goal is to identify the framework and generic tools appropriate for a broadly applicable scientist's workbench.*

It is clear that there are widely applicable kernels for scientific $\mathcal{PSE}$s. Some of these are easy to identify, for example, facilities for the visualization of data, symbolic processing of problem solving specifications, manipulation of geometric shapes, and tools to create and use libraries. Other less-well-developed kernels include object-oriented knowledge base facilities, language translators geared to scientific and engineering jargon, controllers for complex distributed computational environments, and support facilities for the interface between the computer and the outside world. What is not easy is to identify the right combination of these kernels and the dividing line between generic and application-specific capabilities. *Advances are required in understanding the architecture and properties of these kernel facilities.*

A system that seems particularly attractive would provide access to classical mathematical information. This is a narrow enough area to be feasible to attack now, there are numerous previous "handbooks" to build on, and, most inviting, once completed, the system would become a generic $\mathcal{PSE}$ for many areas of science and engineering.

4.3 Application-Specific Problem Solving Environments

Ken Wilson's dream is that some day engineers will write down a problem on an electronic medium, using a textbook-type language, and a "system" will intelligently respond with a reasonable solution. This dream has been regarded by many as science fiction. We believe that such a goal should be basic for researchers in scientific $\mathcal{PSE}$s, even if its "full" realization does not appear to be possible any time soon. The development of such technology will change completely the way we do science and engineering. It will be a breakthrough with enormous consequences. One can argue that the hardware technology to become available in the 1990s can support this dream. The principal barrier

is the lack of application-specific knowledge bases and an appropriate integrated infrastructure of symbolic, numeric, geometrical, artificial intelligence, and natural languages facilities; it is the lack of these ingredients that puts realization of this dream into the distant future.

The first step towards the realization of this goal should be the demonstration of this idea within small, specific problem domains. This will require the development of

- Expert systems for a few, well-defined application domains. These must be capable of analyzing user specifications, selecting an appropriate problem solving process, generating an appropriate computational model, and producing answers in a rapid, natural form.
- Multimedia user interfaces providing natural forms of communication with languages, graphics, images, and voice specific to these $\mathcal{PSE}$s,
- Access to auxiliary facilities and information such as meta-libraries (libraries of "smart" algorithms), electronic application-books(CD-books), and video instructors (on line, touch screen, audio driven help system).

These future $\mathcal{PSE}$s will be supported by a new breed of information database, a knowledge-base, and science and engineering electronic encyclopedia systems capable of handling or providing heterogeneous multimedia information. New data structures, storage schemes, and manipulation algorithms will be needed for each application domain (i.e., chemists operate on molecular structures, and electrical engineers on circuit diagrams).

An application-specific $\mathcal{PSE}$ that seems particularly attractive would provide access to classical mathematical information. This $\mathcal{PSE}$ covers retrieval and use of the enormous body of information about mathematical formulas, expansions, and associated techniques. It is a narrow enough area to be feasible to attack now, there are numerous previous "handbooks" to build on, and, most inviting, once completed it would become a generic $\mathcal{PSE}$ for many areas of science and engineering. *The goal is to produce an encyclopedia $\mathcal{PSE}$ for applied mathematics.*

4.4 Problem Solving Environments for Education

The use of multimedia technologies can revolutionize the educational process in every field, including computational science and engineering. A "video" instructor can be integrated into a conventional computational platform and can monitor each step of the problem solving process, it can react to pupils' choices and decisions through natural (sight, sound, touch) media. This has the potential to revolutionize every aspect of the instruction and learning process. Most of the $\mathcal{PSE}$ application developments today using this technology are focused on office, publishing, and factory environments, primarily because of their high immediate economic impact, and in some sense, their lower level technical difficulty (or, perhaps, narrower scope). On the other hand, the educational institutions today are faced with the rising cost of teaching, reduced financial resources, and the shortage of qualified instructors in science and mathematics. These institutions are also, in the view of some analysts, less and less effective in meeting their goals. Further, they face a new breed of students who have been exposed almost since infancy to various multimedia technologies and sophisticated computer games, and who have used them as learning devices. *Thus, the development*

of multimedia based problem solving educational technology is critical, yet natural, to the evolution and improvement of the education process.

The $\mathcal{PSE}$ approach is especially attractive in science and engineering for reasons beyond the economic and human resources factors. The long-term goal is to develop $\mathcal{PSE}$s that reflect and mold specific subject areas, just as textbooks and curriculum standards do now in a different way. Thus, ideally, the $\mathcal{PSE}$ learned in elementary school would be completely compatible with that used in high school, university, and later life. More specifically, the notation and displays for simple algebraic equations would remain constant as one moves through the educational system and out into the work world. Each subject area would create its own "tree" of compatible $\mathcal{PSE}$s, starting with the root one for the introductory course at whatever level it might first be offered. *The feasibility of this approach to organizing problem solving capabilities should be demonstrated for some subject area.*

Unfortunately there is a severe shortage of people with backgrounds in both computer science and its applications. There is a need to encourage young people to acquire interdisciplinary training through fellowships and postdoctoral positions, coupled with the fostering of computational science programs. The widespread use of $\mathcal{PSE}$s in education should help to attract students to work in this area.

4.5 Implementation of Problem Solving Environments

Although the architecture, design, and implementation methodology for $\mathcal{PSE}$s are open research issues, it must be recognized that future $\mathcal{PSE}$s will be characterized by immense complexity. They will be capable of interacting naturally with users having different levels of expertise and computational objectives. They will interact with multiple devices, each processing or storing its own gender of information, and together creating a non-homogeneous collection. They will be supported by heterogeneous algorithmic infrastructures, each with its own intelligent front-end. They will execute on a wide variety of machines, over networks, and in complex computing environments. All this must be integrated to achieve a specific computational goal. *The implementation of $\mathcal{PSE}$s presents formidable software engineering challenges.*

The view of a $\mathcal{PSE}$ as a set of collaborating components through some form of a software bus or software kernel fits very well to the object-oriented paradigm which can be used as one of the methodologies for creating $\mathcal{PSE}$s. To support the realization of $\mathcal{PSE}$s, we need to create generic, object-oriented, knowledge-based, $\mathcal{PSE}$ kernels which support programming in the large. These kernels will have some visual script capabilities for developing $\mathcal{PSE}$s for different applications, information storage systems for multimedia and science objects, and domain-specific languages plus their associated compilers. *Discovering and developing appropriate software engineering methodologies for implementing $\mathcal{PSE}$s is one of the critical research challenges for this field.*

5 FINDINGS AND RECOMMENDATIONS

To achieve the potential contributions of computers to science and society we must have a collaborative effort to design and build the problem solving environments that will give the working scientist and engineer routine access to high-performance computers, to the advanced algorithms, to the accumulated know-how of computational science. We must answer the challenge of discovering how to build $\mathcal{PSE}$s wholesale that are powerful yet flexible and not limited to a particular computer, problem solving method, or programming infrastructure.

5.1 Findings

Problem solving environments for computational science is still in its infancy. It is a field of promise, one where the technological infrastructure has advanced enough to allow the dreams of 30 years ago to be realized. It has great diversity, and potentially enormous scientific and economic impact, and immaturity. To discover how to create all the $\mathcal{PSE}$s needed to exploit high-performance computing is one of the grand challenges of computer science. This report offers the following seven findings.

Finding 1. The time is ripe for major efforts to create problem solving environments for computational science.

Just a vision in the 1960s, harnessing computers to interact with people on their own terms is now a possibility. The required high performance computing power is here and much more power is on the way. A massive amount of computational problem solving expertise has accumulated, and many of the best methods are so new or so complex that few can implement them well. It is time that our scientists and engineers receive the adequate software support for their tasks.

Finding 2. Problem solving environments for computational science will have an enormous positive impact on the productivity of scientists and engineers.

The time required for many design, analysis, development, and, eventually, production tasks will be shortened by one or two orders of magnitude. Further, the results produced will, in many cases, be better and more reliable. There are many areas of computational science that are well understood (by some at least) and, in some sense, routine. Yet in these areas the implementation of a new design analysis or on a new computing environment seems to take as long the tenth time as the first time. The challenge for $\mathcal{PSE}$s is to encapsulate this problem solving know-how into an easily used, flexible system. Thus the promise of $\mathcal{PSE}$s is to capture what is well known or routine and *not* to provide magic bullets for problems at or beyond the frontiers of computational science. Even so, by raising the level of analysis $\mathcal{PSE}$s will increase the range of individual expertise and speed up projects to such an extent that, in fact, the frontiers will be expanded greatly.

Finding 3. Problem solving environments require the expertise of many subdisciplines of computer science as well as that of the application areas involved.

The workshop title lists numerical analysis, symbolic computing, computational geometry, and artificial intelligence as relevant subdisciplines, but it is clear that others are heavily involved in creating $\mathcal{PSE}$s. Language processing systems are involved from the level of optimizing code execution locally through automatic parallelization of computations up to natural language issues. Larger $\mathcal{PSE}$s in computational science will be self-contained systems with all the responsibilities of operating systems. And, of course, modern high performance architectures and networks will greatly influence $\mathcal{PSE}$ design and operation. Thus it is obvious that $\mathcal{PSE}$ creation requires a collaborative effort of several computer scientists as well as experts from application areas.

Finding 4. The lack of appropriately trained people is an impediment to progress in computational science $\mathcal{PSE}$s, and yet these same $\mathcal{PSE}$s will, in turn, greatly alleviate the shortage of computational scientists.

The designers and architects of $\mathcal{PSE}$s need to have a deep understanding of several subdisciplines of computer science and to be sophisticated about science and engineering practice. Such people are rare. Current educational programs tend to produce computer scientists whose understanding of other sciences is at the sophomore level and to produce scientists whose understanding of computer science is at the same level. To require students to learn both a science or engineering discipline *and* computer science is a heavy burden that only a few will accept. These few are badly needed but we work toward the time when $\mathcal{PSE}$s will give scientists and engineers access to sophisticated computing and modern problem solving methodology without becoming either computer scientists or hackers. The small numbers of trained computational scientists have the responsibility of bridging the gap between computing and applications and of building the $\mathcal{PSE}$s for the massive volumes of scientific computing.

Finding 5. There is a lack of models of computational science problem solving environments with all the ingredients desired; also lacking are certain generic building blocks for $\mathcal{PSE}$s.

The $\mathcal{PSE}$s that exist today mostly have their roots in one area and are much less developed in other areas. This is not unexpected as a large, production-quality $\mathcal{PSE}$ that would serve as a good model would require many dozens of man-years of effort if built with today's methodology. Thus we must visualize the next generation of $\mathcal{PSE}$s much as the blind men visualize the elephant, we see certain features well, but we do not yet appreciate the nature of the whole beast. There are certain $\mathcal{PSE}$ components that are clearly generic, which have been implemented well several times, and yet which are not available for another $\mathcal{PSE}$ without a very large reimplementation effort. Examples here include (a) symbolic manipulation for basic mathematical expressions, (b) visualization packages for two, three or four dimensional phenomena, (c) geometric modelers for a broad class of shapes.

Finding 6. There is an ample number of basic research issues associated with building problem solving environments.

As with any complex grand challenge, there are many sub-challenges where more research is needed. Examples of important research issues for $\mathcal{PSE}$s are:

- *Architecture*: What is an appropriate structure for a $\mathcal{PSE}$? How are its components best organized? How does one allow for growth and evolution?
- *Kernel*: We believe that there is a basic kernel of facilities that can be used for many $\mathcal{PSE}$s. Which components belong to this kernel? How can very large generic facilities be included in a kernel? Must the kernel allow for easy pruning or expansions?
- *Interface Technology*: $\mathcal{PSE}$s will involve very large subsystems that are independently constructed as stand-alone systems. What are the data structures and protocols that such components should use to interact with the $\mathcal{PSE}$? What are the limits on such interfaces?
- *Scientific Interface*: What are the best ways for a user to communicate with a $\mathcal{PSE}$? Should computers provide additional capabilities beyond traditional equations, text, and pictures?

Finding 7. New engineering design and science are hard; it is implausible that successful $\mathcal{PSE}$s for these purposes can avoid extensive user interaction.

There are many instances in $\mathcal{PSE}$s where expert systems and other artificial intelligence techniques will be useful in guiding the user or selecting among certain alternatives. However, the $\mathcal{PSE}$ concept includes the "well understood" attribute, so that novel tasks will require human direction for the foreseeable future. Care must be taken in $\mathcal{PSE}$ design to support the user during a long term interaction.

5.2 Recommendations

Broadly speaking, our recommendations for action concern support for research in $\mathcal{PSE}$ design for the long term, current "targets of opportunity" in $\mathcal{PSE}$ construction, and education in computational science. As with the workshop findings, these recommendations are presented as seven statements along with some explanations.

Recommendation 1. Provide support for research into the architecture, design, and methodology of problem solving environments.

The critical aspects of the architecture appear to be: (a) How does one represent methods in a way that they can be compiled into a programming language as machines and systems change? How does one transform such representations as methods are interfaced? How does one modify algorithmic constituents of methods as better algorithms are discovered? (b) What is an appropriate kernel for $\mathcal{PSE}$ construction? (c) How are large complex components of radically different designs incorporated into a single $\mathcal{PSE}$? (d) Can the $\mathcal{PSE}$ kernel and major components remain computationally efficient if high modularity is enforced? (e) Can a good "$\mathcal{PSE}$ generator" be constructed?

Recommendation 2. Prototypes of complete problem solving environments for computational science should be constructed.

A selection of prototype $\mathcal{PSE}$s should be built using "targets of opportunity" involving good groups of collaborators. Each $\mathcal{PSE}$ built should be complex enough to exhibit the principal features and difficulties of the process but not be so complex that an enormous investment of time and money is required. Budgets of $250,000 – $500,000/year for three or four years are suggested. More ambitious projects could be undertaken if the scientific or economic pay off is high enough and the probability of success is adequate.

Recommendation 3. Key major components of problem solving environments for computational science should be constructed.

In addition to complete $\mathcal{PSE}$s, a few particularly important and difficult generic components should be constructed. There are software subsystems that have been implemented several (or many) times but are not yet easily incorporated into a larger, unrelated system. Examples include: (1) visualization of functions of two, three, and four variables, (2) symbolic manipulation of common mathematical expressions, (3) geometric modeler in two and three dimensions. Ideally, projects here would take an existing component and rework it or perhaps put a shell (external interface) around it.

Recommendation 4. The development of interdisciplinary teams should be encouraged throughout computational science research.

It is obvious that several disciplines are almost surely needed to build a complex $\mathcal{PSE}$. Since there is a severe shortage of people well versed in both computer science and its applications to science and engineering, this recommendation generally applied can increase this important pool of talent. It can be carried out for younger people by requiring that research assistants and postdoctoral fellows receive interdisciplinary training.

Recommendation 5. Create "Encyclopedias of Computational Science" which provide quick access to the accumulated formulas, algorithms, and knowhow.

Computational science has a long tradition of handbooks that collect important formulas and results for convenient use. Computers have made tables of function values nearly obsolete, but there still remains an enormous body of knowledge that is hard to access. For example, the *Handbook of Mathematical Functions* [3] and the three volumes *Higher Transcendental Functions* [66] are monumental works of this type whose content should be reorganized and made easily available to the computational science community. Computational science, like all other sciences, should have on-line search facilities for the published literature. These encyclopedia should have algorithms more general than mathematical formulas. How to represent these algorithms is an important research issue.

Recommendation 6. Foster educational programs in interdisciplinary computational science at the advanced level and use $\mathcal{PSE}$s to teach computational science at all levels.

There is a shortage of people who have the broad range of computer science knowledge and application discipline knowledge needed to do computational science. This holds both for specific applications and for building $\mathcal{PSE}$s. The few computational science educational programs recently established show that it is practical to design viable programs in this area. More of these are needed.

The ideal $\mathcal{PSE}$ will accommodate a wide range of sophistication in its users, from young students to advanced researchers. Until we can realize the ideal, however, some $\mathcal{PSE}$s for science should be targeted specifically to science education at the junior high, senior high, and undergraduate college levels. These $\mathcal{PSE}$s will introduce students to the future paradigm of science and engineering work and expose them to scientific activities and phenomena that are not commonly found in simple laboratories.

Recommendation 7. Provide support for research on effective techniques for interaction between computers and computational scientists.

Many important $\mathcal{PSE}$ applications will involve sustained interaction with the user as a problem solution or a design is developed. Computer graphics opens up new mechanisms for representing and visualizing the scientific phenomena involved. These mechanisms must be perfected, and novel approaches explored. Keyboard entry of information will soon be replaced by voice input and handwritten formulas, which will surely change the nature of the user interfaces for $\mathcal{PSE}$s. Further, the $\mathcal{PSE}$ should maintain a "laboratory notebook" of the problem solving process, both as a means to verify how a solution was obtained and to back up the solution process when a dead-end path has been followed.

REFERENCES

[1] P. C. Abbott. Problem solving using Mathematica. In Gaffney and Houstis [81], pages 33–43. 1992.

[2] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G. J. Sussman, J. Wisdom, and K. Yip. Intelligence in scientific computing. *Comm. Assoc. Comput. Machin.*, 32(5):546–562, May 1989.

[3] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*, volume 55 of *Applied Mathematics Series*. National Bureau of Standards, Washington, D.C., 1964.

[4] ACM. *Proc. International Symposium on Symbolic and Algebraic Computing (ISSAC'89)*, ACM Press, New York, 1989.

[5] ACM. *Proc. International Symposium on Symbolic and Algebraic Computing (ISSAC'90)*, ACM Press, New York, 1990.

[6] *Comm. ACM*, 34(12), Dec. 1991. Special Issue: Collaborative Computing.

[7] *Comm. ACM*, 33(9), Sept. 1990. Special Issue: Object-Oriented Design.

[8] G. Agha. Concurrent object-oriented programming. In *Comm. ACM* [7], pages 125–141. Special Issue: Object-Oriented Design.

[9] T. J. Aird and J. R. Rice. PROTRAN: Problem solving software. *Adv. Engin. Software*, 5:202–206, 1983.

[10] J. L. Alty, C. D. McCartney, and M. Zalloco. A multimedia interface support tool for process control interface design. Technical Report Tech. Report ESPRIT P2397, University of Loughborough, November 1991.

[11] F. L. Alvarado. The Sparse Matrix Manipulation System. Technical Report ECE-89-1, Dept. of Elec. and Comp. Eng. , Univ. of Wisc., Madison, WI, 1989.

[12] S. Amarel. Problems of representation in heuristic problem solving. In Jernigan et al. [116], pages 11–32.

[13] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.

[14] S. F. Ashby and M. K. Seager. A proposed standard for interative linear solvers. Version 1.0. Technical Report UCRL-102860, Lawrence Livermore National Laboratory, Livermore, Jan. 1990.

[15] C. C. Ashcraft, R. G. Grimes, J. G. Lewis, B. W. Peyton, and H. D. Simon. Progress in sparse matrix methods for large linear systems on vector supercomputers. *Int'l. J. Supercomput. Appl.*, 1(4):10–30, Dec. 1987.

[16] N. C. Baker and S. J. Fenves. Towards a grammar of structural design. In T. O. Barnwell, Jr., editor, *Computing in Civil Engineering*, pages 178–185. ASCE, New York, 1989.

[17] D. Balaban, J. Garbarini, W. Greiman, and M. Durst. Knowledge representation for the automatic generation of numerical simulators for PDEs. *Math. Comput. Simul.*, 31:383–393, 1989.

[18] P. Baras, J. Blum, J. C. Paumier, P. Witomski, and F. Rechenmann. EVE: An object-centered knowledge based PDE solver. In Houstis et al. [107], pages 1–18.

[19] T. J. Barnes, D. Harrison, A. R. Newton, and R. L. Spickelmier. *Electronic CAD Frameworks.* Kluwer Academic Pub., Boston, 1992.

[20] L. Bass and J. Coutaz. *Developing Software for the User Interface.* Addison-Wesley, 1991.

[21] H. H. Bau, T. Herbert, and M. M. Yovanovich, editors. *Symbolic Computation in Fluid Mechanics and Heat Transfer*, The American Society of Mechanical Engineers, New York, 1988.

[22] A. Bellen. ODE test problems, Oct. 1991. NA Digest 91(42), C. Moler ed.

[23] M. Benantar, R. Biswas, J. E. Flaherty, and M. S. Shephard. Parallel computation with adaptive methods for elliptic and hyperbolic systems. *Comput. Meth. Appl. Mech. Engin.*, 82:73–93, 1990.

[24] A. Bijl. *AI in architectural CAD.* Kogan Page, London, 1986.

[25] R. F. Boisvert. The guide to available mathematical software advisory systems. *Math. Comput. Simul.*, 31:453–463, 1989.

[26] R. F. Boisvert and D. K. Kahaner. DEQSOL and ELLPACK: Problem solving environments for partial differential equations. *Office of Naval Research Asian Office Scientific Information Bulletin (NAVSO P-3580)*, 16(1):7–19, 1991.

[27] J. Bonomo and W. R. Dyksen. XELLPACK: An interactive problem-solving environment for elliptic partial differential equations. In Houstis et al. [106], pages 331–341.

[28] T. I. Boubez, A. M. Froncioni, and R. L. Peskin. A prototyping environment for differential equations. *ACM Trans. Math. Softw.*, 18(1):1–10, March 1992.

[29] J. C. Boudreaux. Problem solving and the evolution of programming languages. In Jernigan et al. [116], pages 103–126.

[30] K.W. Brodlie, M. Berzins, P.M. Dew, A. Poon, and H. Wright. Visualization and its use in scientific computations. In Gaffney and Houstis [81], pages 293–304.

[31] F. P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Comput.*, 20:10–19, Apr. 1987.

[32] K. Broughan. SENAC: Lisp as a platform for constructing a problem solving environment. In Gaffney and Houstis [81], pages 351–360.

[33] K. A. Broughan, G. Keady, T. D. Robb, M. G. Richardson, and M. C. Dewar. Some symbolic computing links to the NAG numeric library. *SIGSAM Bulletin*, 25(3):28–37, July 1991.

[34] G. Butler and J. Cannon. The design of Cayley - a language for modern algebra. In Miola [135], pages 10–19.

[35] G. Carey, J. Schmidt, V. Singh, and D. Yelton. A scalable, object-oriented finite element solver for partial differential equations on multicomputers. In *Proc. 6th ACM International Conference on Supercomputing*, pages 387–396, ACM Press, New York, 1992.

[36] B. W. Char. Progress report on a system for general-purpose symbolic algebraic computation. In *Proc. ISSAC'90* [5], pages 96–103.

[37] B. W. Char, K. O Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt. *MAPLE Reference Manual*. Waterloo, 1988.

[38] M. Clarkson. Expert systems as an intelligent user interface for symbolic algebra. In Gaffney and Houstis [81], pages 205–213.

[39] *Comput. Graph.*, 21(6), Nov. 1987. Special issue on Visualization in Scientific Computing, edited by B. H. McCormick, T. A. DeFanti and M. D. Brown.

[40] G. O. Cook, Jr. ALPAL: A program to generate physics simulation codes from natural descriptions. *Int'l. J. Modern Phys.*, 1(1):1–51, 1990.

[41] G. O. Cook, Jr. and J. F. Painter. ALPAL: A tool to generate simulation codes from natural descriptions. In Houstis et al. [107], pages 401–419.

[42] G. O. Cook, Jr., J. F. Painter, and S. A. Brown. How symbolic computation boosts productivity in the simulation of partial differential equations. Technical Report UCRL-JC-106442, Lawrence Livermore National Laboratory, Livermore, Feb. 1991.

[43] W. R. Cowell, editor. *Sources and Development of Mathematical Software*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

[44] C.W. Cryer. The ESPRIT project FOCUS. In Gaffney and Houstis [81], pages 371–380.

[45] G.J. Culler and B.D. Fried. An on-line computing center for scientific problems. In *Proc. 1963 Pacific Computer Conf.*, pages 221–242. IEEE, 1963.

[46] J. H. Davenport, editor. *EUROCAL'87: European Conference on Computer Algebra, Leipzig, GDR, June 1987*, number 378 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1989.

[47] J. H. Davenport. Current problems in computer algebra systems design. In Miola [135], pages 1–9.

[48] J. Della Dora and J. Fitch, editors. *Computer Algebra and Parallelism*. Academic Press, London, 1989.

[49] P. J. Denning. Massive parallelism in the future of science. *American Scientist*, 77(1):16–18, Jan./Feb. 1989.

[50] M. C. Dewar. IRENA - an integrated symbolic and numerical computation environment. In *Proc. ISSAC'89* [4], pages 171–179.

[51] M. C. Dewar and M. G. Richardson. Reconciling symbolic and numeric computation in a practical setting. In Miola [135], pages 195–204.

[52] D. S. Dodson, R. G. Grimes, and J. G. Lewis. Sparse extensions to the Fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 17(2):253–263, 1991.

[53] D. S. Dodson and J. G. Lewis. Issues related to extension of the basic linear algebra subprograms. *ACM SIGNUM Newsletter*, 20(1):19–22, 1985.

[54] D. S. Dodson and J. G. Lewis. Proposed sparse extensions to the basic linear algebra subprograms. *ACM SIGNUM Newsletter*, 20(1):22–25, 1985.

[55] Y. Doleh and P. S. Wang. SUI: A system independent user interface for an integrated scientific computing environment. In *Proc. ISSAC'90* [5], pages 88–95.

[56] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide.* SIAM Pub., Philadelphia, PA, 1978.

[57] J. J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Comm. ACM*, 30:403–407, 1987.

[58] J. J. Dongarra and T. Rowan. Test version of Xnetlib available, Dec. 1991. NA Digest 91(49), C. Moler ed.

[59] Y. Dubois-Pélerin, T. Zimmermann, and P. Bomme. Object-oriented finite element programming: II. A prototype program in Smalltalk. *Comput. Meth. Appl. Mech. Engrg.*, 98(3):361–397, 1992.

[60] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1989.

[61] D. Duval and F. Jung. Examples of problem solving using computer algebra. In Gaffney and Houstis [81], pages 133–141.

[62] W. R. Dyksen and C. R. Gritter. Scientific computing and the algorithm selection problem. In Houstis et al. [107], pages 19–32.

[63] R. Eigenmann, J. Hoeflinger, G. Jaxon, and D. Padua. Cedar Fortran and Its Compiler. In *Proc. of the Joint Conf. on Vector and Parallel Processing, Zürich, Switzerland*, volume 457 of *Lecture Notes in Computer Science*, pages 288–300. Springer-Verlag, Berlin, 1990.

[64] B. Engquist and T. Smedsaas. Automatic computer code generation for hyperbolic and parabolic differential equations. *SIAM J. Sci. Stat. Comput.*, 1:249–259, 1980.

[65] B. Engquist and T. Smedsaas. Automatic analysis in PDE software. In B. Engquist and T. Smedsaas, editors, *PDE Software: Modules, Interfaces and Systems*, pages 399–409. North-Holland, Amsterdam, 1984.

[66] A. Erdélyi, W. Magnus, F. Oberhettinger, and F. Tricomi. *Higher Transcendental Functions*, volumes 1-3. McGraw Hill, New York, 1953-55.

[67] J. Erhel and B. Philippe. Aquarels: A problem-solving environment for numerical quality. In R. Vichnevetsky and J.J.H. Miller, editors, *Proc. IMACS Conf.*, pages 45–46, Dublin, July 1991.

[68] R. E. Ewing. A posteriori error estimation. *Comput. Meth. Appl. Mech. Engin.*, 82:59–72, 1990.

[69] R. J. Fateman. Advances and trends in the design and construction of algebraic manipulation systems. In *Proc. ISSAC'90* [5], pages 60–67.

[70] R. J. Fateman. A review of Macsyma. *IEEE Trans. Knowledge and Data Eng.*, 1(1):133–145, March 1989.

[71] R. J. Fateman and W. Kahan. Improving exact integrals from symbolic algebra systems. Unpublished note, Aug. 1987.

[72] S. J. Fenves, M. L. Maher, and D. Sriram. Expert systems: C.E. potential. *Civil Engineering*, 54:44–48, Oct. 1984.

[73] D. A. Field. From solid modeling to finite element analysis. In T. L. Kunii, editor, *Application Development Systems*, pages 220–249. Springer-Verlag, Tokyo, 1986.

[74] J. Fitch. A delivery system for REDUCE. In *Proc. ISSAC'90* [5], pages 76–81.

[75] J. P. Fitch. Can REDUCE be run in parallel? In *Proc. ISSAC '89* [4], pages 155–162.

[76] J. P. Fitch and R. G. Hall. Symbolic computation and the finite element method. In Davenport [46], pages 95–96.

[77] J. E. Flaherty, P. Paslow, M. S. Shephard, and J. D. Vasilakis, editors. *Adaptive Methods for Partial Differential Equations*. SIAM, Philadelphia, 1989.

[78] D. E. Foulser and W. D. Gropp. CLAM and CLAMShell: A system for building user interfaces. In E. N. Houstis, J. R. Rice, and R. Vichnevetsky, editors, *Pre-proceedings Second Int'l. Conf. Expert Systems Numer. Comput.*, pages 22–25, West Lafayette, March 1990. Extended abstract.

[79] B. Frederickson and J. Mackerle. Partial list of major finite element programs and description of some of their capabilities. In A.K. Noor and W.D. Pikley, editors, *State of the Art Surveys on Finite Element Technology*, pages 363–403. The American Society of Mechanical Engineers, 1983.

[80] I. Frick. SHEEP and classification in general relativity. In B. Buchberger and B. F. Caviness, editors, *Proc. EUROCAL'85, vol. 2*, number 204 in Lecture Notes in Computer Science, pages 161–162, Springer-Verlag, Berlin, 1985.

[81] P. Gaffney and E. N. Houstis, editors. *Programming Environments for High-Level Scientific Problem Solving*. North-Holland, Amsterdam, 1992.

[82] K. A. Gallivan, M. T. Heath, E. Ng, J. M. Ortega, B. W. Peyton, R. J. Plemmons, C. H. Romine, A. H. Sameh, and R. G. Voigt. *Parallel Algorithms for Matrix Computations*. SIAM, Philadelphia, 1990.

[83] V. Ganzha and R. Liska. Application of the REDUCE computer algebra system to stability analysis of difference schemes. In E. Kaltofen and S. M. Watt, editors, *Proc. Computers and Mathematics '89*, pages 119–129. Springer-Verlag, New York, 1989.

[84] M. Garbey, H. G. Kaper, and M. K. Kwong. Symbolic manipulation software and the study of differential equations. In H. G. Kaper and M. Garbey, editors, *Asymptotic Analysis and the Numerical Solution of Partial Differential Equations*, volume 130 of *Lecture Notes in Pure and Applied Mathematics*, pages 241–265. Marcel Dekker, Inc., New York, 1991.

[85] D. Gelernter, A. Nicolau, and D. Padua, editors. *Languages and Compilers for Parallel Computing*. MIT Press, Cambridge, MA, 1990.

[86] A. Genz, Z. Lin, C. Jones, D. Luo, and T. Prenzel. Fast Givens goes slow in MATLAB. *ACM SIGNUM Newsletter*, 26(2):11–16, Apr. 1991.

[87] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, 1992.

[88] R. Goldman and R. P. Gabriel. Qlisp: Parallel processing in Lisp. *IEEE Software*, 6(4):51–59, July 1989.

[89] D. P. Greenberg. Computers and architecture. *Scientific American*, pages 104–109, Feb. 1991.

[90] D. Gries, T. Walker, and P. Young. 1988 Snowbird report: A discipline matures. *IEEE Comput.*, 22(2):72–75, 1989.

[91] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation and Application*. SIAM, Philadelphia, 1991.

[92] W. D. Gropp, D. E. Foulser, and S. Chang. *CLAM User's Guide.* Scientific Computing Associates, Inc., New Haven, 1989.

[93] L. Gross, P. Sternercker, and W. Schönauer. The finite element tool package, VECFEM (version 1.1). Technical Report 45/91, University of Karlsruhe, 1991.

[94] R. Haber, B. Bliss, D. Jablonowski, and C. Jog. A Distributed Environment for Run-Time Visualization and Application Steering in Computational Mechanics. *Computer Systems Engineering Journal*, (to appear).

[95] S.J. Hague. Using FOCUS technology to build front ends. In Gaffney and Houstis [81], pages 383–392.

[96] W. L. Harrison III and D. A. Padua. PARCEL: Project for the automatic restructuring and concurrent evaluation of Lisp. In *Proc. 1988 Int'l. Conf. Supercomput.*, pages 527–538, ACM Press, New York, 1988.

[97] A. C. Hearn. Reduce 2: A system and language for algebraic manipulation. In Petrick [151], pages 128–133.

[98] A. C. Hearn. *REDUCE User's Manual.* The Rand Corporation, Los Angeles, 1987.

[99] A. C. Hearn. Algebraic computation: The quiet revolution. . In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation. Papers in Honor of John McCarthy*, pages 177–186. Academic Press, San Diego, 1991.

[100] A. C. Hearn, A. Boyle, and B. F Caviness. *Symbolic computation: Directions for future research. report of a worshop on symbolic and algebraic computation.* SIAM, Philadelphia, 1989.

[101] P. N. Hilfinger and P. Collela. FIDIL: A language for scientific programming. In R. Grossman, editor, *Symbolic Computation: Applications to Scientific Computing*, pages 97–138. SIAM, Philadelphia, 1989.

[102] C. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, CA, 1989.

[103] M. J. Hopper. *Harwell subroutine library. A catalogue of subroutines*. HMSO, London, 1978.

[104] E. N. Houstis, T. S. Papatheodorou, and J. R. Rice. Parallel ELLPACK: An expert system for the parallel processing of partial differential equations. In Houstis et al. [106], pages 63–73.

[105] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis, K.-Y. Wang, and S. Weerawana. //Ellpack: A numerical simulation programming environment for parallel MIMD machines. In *Proc. 1990 Int'l Conf. Supercomput.*, pages 96–107, Amsterdam, ACM Press, New York, 1990.

[106] E. N. Houstis, J. R. Rice, and R. Vichnevetsky, editors. *Intelligent Mathematical Software Systems*. North-Holland, Amsterdam, 1990.

[107] E. N. Houstis, J. R. Rice, and R. Vichnevetsky, editors. *Expert Systems for Scientific Computing*. North-Holland, Amsterdam, 1992.

[108] E. N. Houstis and J. R. Rice. Parallel ELLPACK, a development environment and problem solving environment for high performance computing machines. In Gaffney and Houstis [81], pages 229–241.

[109] *Grand Challenges: High Performance Computing and Communications. A report by the committee on Physical, Mathematical, and Engineering Sciences*. Office of Science and Technology Policy, Washington, D.C., 1991.

[110] M. A. Hussain, L. F. Coffin, and K. A. Zaleski. Three-dimensional singular element. Technical report, Corporate Research and Development, General Electric, Schenectady, NY, Dec. 1980.

[111] *IBM J. Res. Dev.*, 35(1/2), Jan./Mar. 1991. Special Issue on Visual Interpretation of Complex Data.

[112] IMSL, Houston, TX. *Math/Library, Stat/Library,* and *SFUN/Library*, 1987.

[113] IMSL, Houston, TX. *User's Manual: PDE/PROTRAN*, 1989.

[114] R. H. F. Jackson, P. T. Boggs, S. G. Nash, and S. Powell. Guidelines for reporting results of computational experiments. Report of the ad hoc committee. *Math. Progr.*, 49:413–425, 1991.

[115] R. D. Jenks, R. S. Sutor, and S. M. Watt. Skratchpad II: An abstract datatype system for mathematical computation. In J. R. Rice, editor, *Mathematical Aspects of Scientific Software*, volume 14 of *The IMA Volumes in Mathematics and its Applications*, pages 157–182. Springer-Verlag, New York, 1988.

[116] R. Jernigan, B. W. Hamill, and D. W. Weintraub, editors. *The Role of Language in Problem Solving*, volume I. North-Holland, Amsterdam, 1985.

[117] N. Kajler. Building graphic user interfaces for computer algebra systems. In Miola [135], pages 235–244.

[118] M. Kamel and W. H. Enright. ODEXPERT: A knowledge based system for automatic selection of initial value ODE system solvers. In Houstis et al. [107], pages 33–54.

[119] E. Kant, F. Daube, W. MacGregor, and J. Wald. Automated synthesis of finite difference programs. In Noor et al. [140], pages 45–61.

[120] K. Kennedy, K. S. McKinley, and C.-W. Tseng. Analysis and transformation in the ParaScope editor. In *Proc. ACM Int'l. Conf. Supercomput.* held in Cologne, Germany, June 1991, pages 433–447. ACM Press, New York, 1991.

[121] M. Klerer and J. Reinfelds. *Interactive Systems for Experimental Applied Mathematics*. Academic Press, New York, 1968.

[122] C. Konno, Y. Umetani, M. Igai, and T. Ohta. Interactive/visual DEQSOL: Interactive creation, debugging, diagnosis, and visualization of numerical simulation. In Houstis et al. [106], pages 301–317.

[123] C. Konno, M. Yamabe, M. Saji, N. Sagawa, Y. Umetani, H. Hirayama, and T. Ohta. Automatic code generation method of DEQSOL. *J. Inform. Proc.*, 11(1):15–21, 1987.

[124] D. J. Kuck. A user's view of high-performance scientific and engineering software systems in the mid-21st century. In Houstis et al. [107], pages 69–87.

[125] U.W. Kulisch. *Computer Arithmetic in Theory and Practice*. Academic Press, New York, 1981.

[126] A. Laub. Numerical linear algebra aspects of control design computations. *IEEE Trans. Automat. Control*, AC-30:97–108, 1985.

[127] A. Laub. Invariant subspace methods for the numerical solution of Riccati equations. In S. Bittanti, A. Laub, and J. Willems, editors, *The Riccati equation*, pages 163–196. Springer-Verlag, Berlin, 1991.

[128] B. Leasure, editor. *PCF Fortran: Language Definition, version 3.1.* The Parallel Computing Forum, Champaign, IL, Aug. 1990.

[129] J. K. Lee and D. Gannon. Object-oriented parallel programming experiments and results. *Proc. Supercomputing'91*, pages 273–282. IEEE Computer Soc. Press, Los Alamitos, Calif. 1991

[130] M. R. Leuze, editor. Scalable parallel libraries workshop report. Preproceedings of a workshop conducted at Oak Ridge National Laboratory, September 1990.

[131] M. Lucks and I. Gladwell. Automated selection of mathematical software. *ACM Trans. Math. Softw.*, 18(1):11–34, March 1992.

[132] W. A. Martin and R. J. Fateman. The MACSYMA system. In Petrick [151], pages 59–75.

[133] P. Mehrotra and J. van Rosendale. The BLAZE language: A parallel language for scientific programming. *Parallel Comput.*, 5:339–361, 1987.

[134] P. Mehrotra and J. van Rosendale. Programming distributed memory architectures using Kali. Technical Report 90-69, ICASE, Oct. 1990.

[135] A. Miola, editor. *DISCO'90: Design and Implementation of Symbolic Computation Systems*. Number 429 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1990.

[136] W. J. Mitchell. Afterword: The design studio of the future. In M. McCullough, W. J. Mitchell, and P. Purcell, editors, *The Electronic Design Studio*, pages 479–494. MIT Press, Cambridge, MA, 1990.

[137] W. J. Mitchell and M. McCullough. *Digital Design Media. A Handbook for Architects & Design Professionals*. Van Nostrand Reinhold, New York, 1991.

[138] C. Moler, J. Little, and S. Bangert. *PRO-MATLAB for Sun Workstations: User's Guide*. The MathWorks, Inc., Sherborn, MA, 1990.

[139] P. K. Moore, C. Ozturan, and J. E. Flaherty. Towards the automatic numerical solution of partial differential equations. *Math. Comput. Simul.*, 31:325–332, 1989.

[140] A. Noor, I. Elishakoff, and G. Hulbert, editors. *Symbolic Computations and their Impact on Mechanics*, volume PVP-205. The American Society of Mechanical Engineers, New York, 1990.

[141] B. J. Novitski. CADD holdouts. *Architecture*, 80(8):97–99, Aug. 1991.

[142] Numerical Algorithms Group, Oxford, England. *NAG Library Manual*, 1988.

[143] J. T. Oden. Smart algorithms and adaptive methods for compressible and incompressible flow: Optimization of the computational process. In J. P. Mesirov, editor, *Very Large Scale Computation in the 21st Century*, pages 87–99. SIAM, Philadelphia, 1991.

[144] T. C. Oppe, W. D. Joubert, and D. R. Kincaid. An overview of NSPCG: A nonsymmetric preconditioned conjugate gradient package. *Comput. Phys. Commun.*, 53:283–293, 1989.

[145] E. M. Paalvast, A. J. van Gemund, and H. J. Sips. A method for parallel program generation with an application to the Booster language. In *Proc. 1990 Int'l. Conf. Supercomput.*, pages 457–469, ACM Press, New York, 1990.

[146] G. V. Paolini and P. Santangelo. An interactive graphic tool to plot the structure of large sparse matrices. In *IBM J. Res. Dev.* [111], pages 231–237. Special Issue on Visual Interpretation of Complex Data.

[147] B. N. Parlett. Progress in numerical analysis. *SIAM Rev.*, 20(3):443–455, July 1978.

[148] A. Perronnet. MODULEF: A library of subroutines for finite element analysis. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering, 1977, 1*, volume 704 of *Lecture Notes in Mathematics*, pages 127–153. Springer-Verlag, Berlin, 1979.

[149] R. L. Peskin. Symbolic manipulation in engineering user interface systems. In Noor et al. [140], pages 97–111.

[150] R. L. Peskin, S. S. Walther, and A. M. Froncioni. SMALLTALK – The next generation scientific computing interface? In Houstis et al. [106], pages 257–267.

[151] S. R. Petrick, editor. *Proc. Second Symposium on Symbolic and Algebraic Manipulation. Los Angeles, California*, ACM SIGSAM, ACM Press, New York, 1971.

[152] R. Piessens, E. de Doncker-Kapenga, C. W. Uberhuber, and D. K. Kahaner. *Quadpack, A subroutine package for automatic integration*. Springer-Verlag, Berlin, 1983.

[153] C. G. Ponder. *Evaluation of "Performance Enhancements" in algebraic manipulation systems*. PhD thesis, University of California, Berkeley, August 1988. Also Tech. Rep. UCB 88/438.

[154] C. G. Ponder. Parallel processors and systems for algebraic manipulation: Current work. *ACM-SIGSAM*, 22(3):21, 1988.

[155] J. M. Purtilo. Dynamic software reconfiguration supports scientific problem solving activites. In Gaffney and Houstis [81], pages 245–254.

[156] J. M. Purtilo. A software interconnection technology to support specification of computational environments. Technical Report R-86-1269, Department of Computer Science, University of Illinois at Urbana-Champaign, Sept. 1986.

[157] J. M. Purtilo. The Polylith software bus. *ACM Trans. Progr. Lang. Syst.*, to appear.

[158] J. M. Purtilo, D. A. Reed, and D. C. Grunwald. Environments for prototyping parallel algorithms. *J. Paral. Dist. Comput.*, 5:421–437, 1988.

[159] A. Radford and G. Stevens. *CADD Made Easy: A Comprehensive Guide for Architects & Designers.* McGraw-Hill Book Co., New York, 1988.

[160] F. Rechenmann and B. Rousseau. A development shell for knowledge based systems in scientific computing. In Houstis et al. [107], pages 157–173.

[161] W. Renes, M. Vanbegin, P. Van Dooren, and J. Beckers. The MATLAB gateway compiler. A tool for automatic linking of Fortran routines to MATLAB. In *IFAC Symp. on CADCS*, pages 95–100, Swansea, UK, July 1991.

[162] J. R. Rice. *Mathematical Software.* Academic Press, New York, 1971.

[163] J. R. Rice. Statistical computing: The vanguard of the revolution in education. In D.C. Hoaglin and R.E. Welsh, editors, *Ninth Interface Symposium on Computer Science and Statistics*, pages 1–4. Prindle, Weber & Schmidt, Boston, 1976.

[164] J. R. Rice. Mathematical aspects of scientific software. In J. R. Rice, editor, *Mathematical Aspects of Scientific Software*, volume 14 of *The IMA Volumes in Mathematics and its Applications*, pages 1–40. Springer-Verlag, New York, 1988.

[165] J. R. Rice. Mathematical software and ACM publications. In S. G. Nash, editor, *A History of Scientific Computing*, pages 217–227. ACM Press, Addison Wesley, Reading, Mass., 1990.

[166] J. R. Rice. *Numerical Methods, Software and Analysis*. McGraw-Hill, New York, second edition, 1992.

[167] J. R. Rice and R. F. Boisvert. *Solving Elliptic Problems using ELLPACK*. Springer-Verlag, New York, 1985.

[168] J. R. Rice and H. D. Schwetman. Interface issues in a software parts technology. In *ITT Proc. Workshop on Reusability in Programming*, pages 129–137, 1983. Reprinted in *Software Reusability*, (P. Freeman, ed.), IEEE Tutorial, Computer Soc. Press, 1987, pp. 96-104. Revised version in *Software Reusability*, (T. Biggerstaff and A. J. Perlis, eds.), ACM Press, 1989, pp. 125-139.

[169] M. Rosing and R. Schnabel. An overview of DINO - a new language for numerical computation on distributed memory multiprocessors. In G. Rodrigue, editor, *Proc. Third SIAM Conf. Parallel Processing for Sci. Comput.*, pages 312–316, SIAM, Philadelphia, 1987.

[170] Y. Saad. An overview of Krylov subspace methods with applications to control problems. In M. A. Kaashoek, J. H. van Schuppen, and A. C. Ran, editors, *Signal Processing, Scatter-*

ing, Operator Theory, and Numerical Methods. Proceedings of the international symposium MTNS-89, vol III, pages 401–410, Boston, 1990. Birkhauser.

[171] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computation. Technical Report 1029, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, August 1990.

[172] Y. Saad. Krylov subspace methods for supercomputers. *SIAM J. Sci. Stat. Comput.*, 10(6):1200–1232, Nov. 1989.

[173] N. Sagawa, D.P. Rinn, and N.J. Hurley. An integrated problem solving environment for numerical simulation of engineering problems. In Gaffney and Houstis [81], pages 191–199.

[174] F. Schwarz. Symmetries of differential equations: From Sophus Lie to Computer Algebra. *SIAM Review*, 30(3):450–481, Sept. 1988.

[175] M. K. Seager. A SLAP for the masses. In G. F. Carey, editor, *Parallel Supercomputing: Methods, Algorithms and Applications*, pages 135–155. John Wiley & Sons, Chichester, 1989.

[176] G. Sewell. *Analysis of Finite Element Method-PDE/PROTRAN*. Springer-Verlag, 1985.

[177] N. Sharma. Generating finite element programs for Warp machine. In Bau et al. [21], pages 93–102.

[178] N. Sharma and P. S. Wang. Generating finite element programs for shared memory multiprocessors. In Noor et al. [140], pages 63–79.

[179] B. Shneiderman. Overcoming limitations imposed by current programming languages. In Jernigan et al. [116], pages 253–275.

[180] H. A. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, Mass., second edition, 1981.

[181] Soft Waterhouse, Inc., Honolulu. *DERIVE User Manual*, 3rd edition, 1989.

[182] N. M. Soiffer. *The design of a user interface for computer algebra systems*. PhD thesis, University of California, Berkeley, CA, 1991.

[183] F. Szelényi and V. Zecca. Visualizing parallel execution of FORTRAN programs. In *IBM J. Res. Dev.* [111], pages 270–282. Special Issue on Visual Interpretation of Complex Data.

[184] H. Tan. Symbolic derivation of material property matrices in finite element analysis. In Bau et al. [21], pages 111–116.

[185] M. Thuné. Automatic GKS stability analysis. *SIAM J. Sci. Stat. Comput.*, 7(3):959–977, July 1986.

[186] S.-S. Tong. Coupling symbolic manipulation and numerical simulation for complex engineering designs. *Math. Comput. Simul.*, 31:419–430, 1989.

[187] S.-S. Tong. Integration of symbolic and numerical methods for optimizing complex engineering systems. In Gaffney and Houstis [81], pages 3–18.

[188] A. Tuchman and M. Berry. Matrix Visualization in the Design of Numerical Algorithms. *ORSA Journal of Computing*, 2(1):84–92, 1990.

[189] Y. Umetani, C. Konno, and T. Ohta. Visual PDEQSOL: A visual and interactive environment for numerical simulation. In Gaffney and Houstis [81], pages 259–267.

[190] A. van den Boom, A. Brown, F. Dumortier, A. Geurts, S. Hammarling, R. Kool, M. Vanbegin, P. Van Dooren, and S. Van Huffel. SLICOT, a subroutine library for control and system theory. In *Preprints IFAC Symp. on CADCS*, pages 89–94, Swansea, UK, July 1991.

[191] P. van den Heuvel, J. A. van Hulzen, and V. V. Goldman. Automatic generation of FORTRAN-coded Jacobians and Hessians. In Davenport [46], pages 120–131.

[192] G. Vanecek. Protosolid: An inside look. Technical Report CAPO-89-26, Dept. Comput. Sci., Purdue University, Nov. 1989.

[193] P. Wang. Finger: A symbolic system for automatic generation of numerical programs in finite element analysis. *Journal of Symbolic Computation*, 2:305–316, 1986.

[194] P. S. Wang. Applying advanced computing techniques in finite element analysis. In Noor et al. [140], pages 45–61.

[195] S. M. Watt. *Bounded parallelism and computer algebra*. PhD thesis, Univ. Waterloo, 1986. Avail. Univ. Waterloo CS dept. tech. rep. CS-86-12.

[196] H. A. G. Wijshoff. Implementing sparse BLAS primitives on concurrent/vector processors: A case study. Technical Report 843, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Jan. 1989.

[197] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, Boston, 1988.

[198] T. Zimmermann, Y. Dubois-Pélerin, and P. Bomme. Object-oriented finite element programming: I. Governing principles. *Comput. Meth. Appl. Mech. Engrg.*, 98:291–303, 1992.

[199] B. Zorn, K. Ho, H. Larus, L. Semenzato, and P. Hilfinger. Multiprocessing extensions in Spur Lisp. *IEEE Software*, 6(4):41–49, July 1989.