1990

The Second International Conference on Expert Systems for Numerical Computing

John R. Rice
Purdue University, jrr@cs.purdue.edu

Robert Vichnevetsky

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

Report Number:
90-963

THE SECOND INTERNATIONAL CONFERENCE ON EXPERT SYSTEMS FOR NUMERICAL COMPUTING*

Co-Chairman: John R. Rice and Robert Vichnevetsky
Conference Coordinator: E.N. Houstis

Computer Sciences Department
Purdue University

Technical Report CSD-TR-963
CAPO Report CER-90-13
March, 1990

Co-sponsored by Computing About Physical Objects Laboratory, IMACS, SIGNUM, NSF and AFOSR.
The conference was partially supported by NSF and AFOSR.

TABLE OF CONTENTS

SESSION 1

EVE: An Object-Centered Knowledge-Based PDE Solver

P. Barras, J. Blum, J.C. Paumier and P. Witomski
IMAG/lab. TIM 3
BP 53 X
38041 Grenoble Cedex, FRANCE

F. Rechenmann
INRIA, IMAG/lab ARTEMIS
BP 53 X
38041 Grenoble Cedex, FRANCE

The EVE system is very representative of the class of knowledge-based systems in scientific computing. It incorporates knowledge on the algorithmic methods involved in the numerical solution of systems of partial differential equations, on the mathematical, and possibly physical, entities manipulated by these methods and on the ways a global problem should be decomposed according to its characteristics. All this knowledge is represented as objects classes: classes describing methods, classes describing entities and classes describing tasks. EVE is written in Lisp and heavily relies on SHIRKA, a frame-based model, to represent the knowledge of exploits. It is the result of the collaboration between a group of artificial intelligence specialists and a group of applied mathematicians. This paper describes the knowledge used by EVE, its global architecture and its user interface.

A class is attached to each algorithmic model needed by EVE. The slots of the class describe the various parameters of the module by their name, their type, their range of admissible values and possibly the ways to compute their value when it is unknown in an instance, i.e. a particular element of the class. The type of a slot can be the name of another class; this means that its values must be instances of that class. It is thus possible to specify that a method can only be applied to some specific type of input parameters and that its output parameters are expected to be of some specific types. One of the slots of a method class contains the name of the executable code.

Classes are also used to describe the entities handled by the system. The classes are organized as a hierarchy in which any subclass inherits the properties of its superclass and possibly adds its own properties. These hierarchies support a classification mechanism which is able to determine the possible classes for a given instance. As this instance is moving down the hierarchy, its associated knowledge is refined and some property values are inferred. in EVE, classes are thus used to describe mathematical entities like equations and their components, variables, parameters or constant values, domains and their frontiers, conditions on frontiers, linear systems and so on.

In EVE, the classification mechanism is used to characterize the set of PDE equations. The set of equations entered by the user through a high level editor is analyzed and translated as a composite object. The components are operators and operands: variables, constant values and parameters. The knowledge base of EVE contains a list of prototypical basic components of PDE to which the basic components of the actual equations are compared through a structural pattern-matching process. When all the components have been identified, it is easy to determine some basic mathematical properties of the system and to rewrite it in a canonical form.

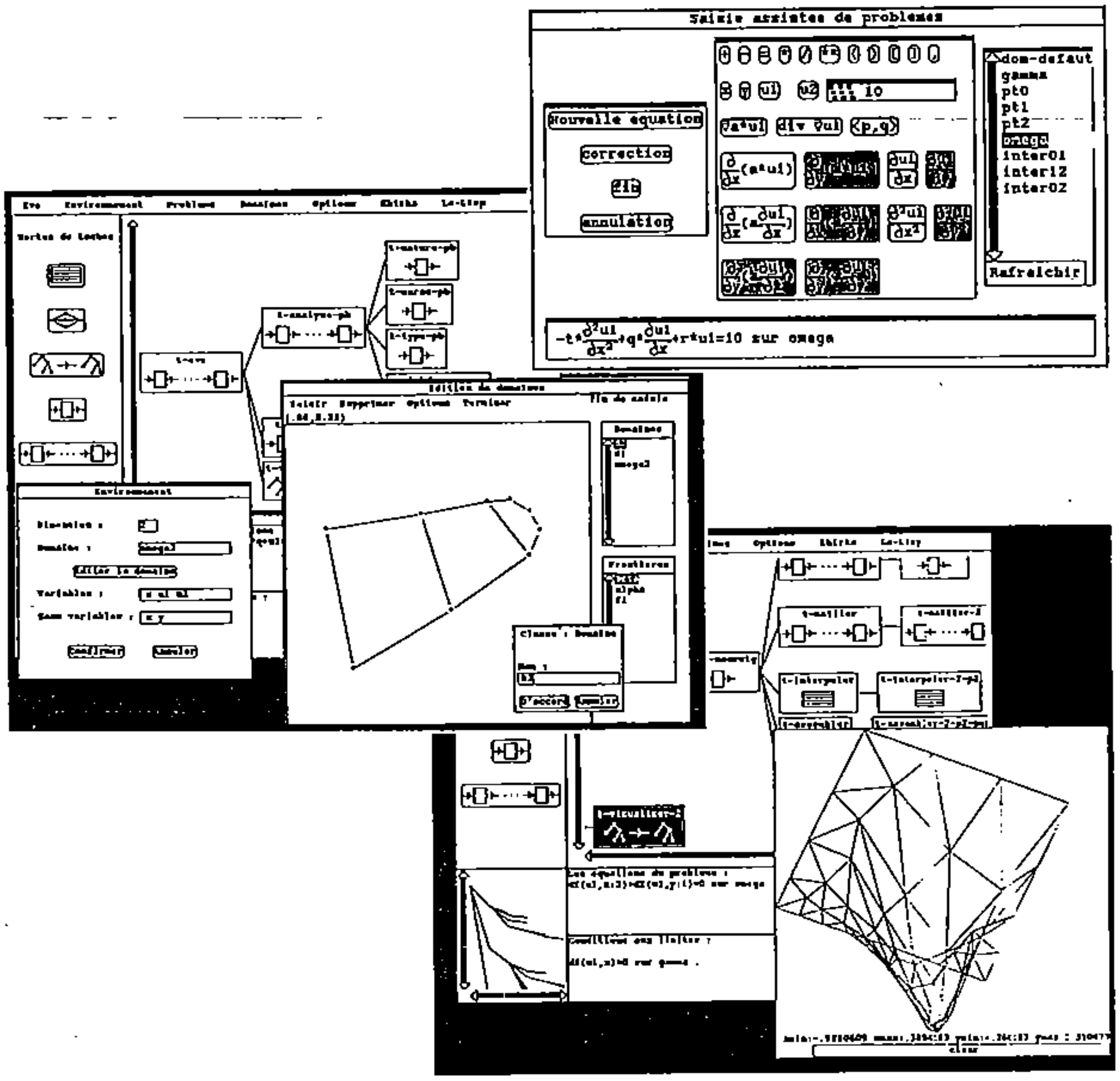
The classification mechanism of SHIRKA is also able to retrieve the adequate methods for some given entity. The entity is first characterized by classification in the hierarchy to which its class belongs. A second classification process is initiated on the hierarchy of methods. The result of this second stage is a list of methods which can be applied to the entity. For instance, this process is used in EVE to determine an adequate method for the solution of the linear system, which is obtained when the finite elements method is applied.

This process has to be repeated each time a precise task must be executed, for which several methods are available. EVE manipulates tasks and incorporates knowledge on the ways a PDE problem should be recursively decomposed in subproblems up to the point it becomes a sequence of calls to algorithmic modules. The decomposition is of course, dependent on the characteristics of the problem. The tasks are themselves described as classes.

EVE is now operational on any workstation supporting Le-Lisp of INRIA and its graphical interface development system. It is presently able to solve linear PDE in one and two dimensions and is well adapted to pedagogical uses. The user enters its system through a specialized equations editor (see figures). Classical operators appear as icons the user selects. These operators can be nested up to the desired level of complexity. In the same way, the user specifies the boundaries conditions. The domain and its frontiers can also be entered via an interactive and graphical way. The user interface offers facilities to link the name of the frontiers in the definition of the boundaries conditions and in the domain definition.

When solving the problem, EVE displays the decomposition of tasks into subtasks it has decided on, and, when requested, provides the user with explanations.

Extensions of the system are planned and some of them are already started. The main extension is to consider time-varying, possibly non-linear equations. This will need a very powerful planning algorithm in order to control the iterations of the solving process. The introduction of some algebraic manipulation modules is also considered.



$-t*\frac{\partial^2 u1}{\partial x^2}+q*\frac{\partial u1}{\partial x}+r*u1=10$ sur omega

Expert Systems For Scientific Computation and The Algorithm Selection Problem

Wayne R. Dyksen and Carl R. Gritter

Department of Computer Science
Purdue University
West Lafayette, IN 47907

In today's world of scientific computing, there exists a large base of well-written, reliable numerical software that solves a host of very complex numerical problems. This base is becoming so large that it has become necessary to investigate the application of expert systems to scientific computation. Thus we are building *Elliptic Expert*, an expert system for elliptic partial differential equations (PDEs). We are in the process of completing a knowledge-base for Elliptic Expert and, using our experience as a springboard, developing a paradigm for expert systems for scientific computing in general.

What the average scientific software user really wants is an expert system which uses artificial intelligence knowledge-base techniques to guide the user towards an accurate solution to his problem. In order to do this, we are using OPS5, a member of the family of production-system languages based on the production system paradigm, to define and control the knowledge-base for our expert system. Our software base is XELLPACK, a large software system that solves PDEs.

There are three main components of such an expert system. The first component focuses on finding the set of algorithms which apply to a given problem. After obtaining such a set, the next step is to select the "best" algorithm or "best" set of algorithms. We call this the *Algorithm Selection Problem.* Once an algorithm is selected, the solution is analyzed using error analysis techniques and the performance of the algorithm is measured. If the user is still unsatisfied, repeated use of the second and third components in conjunction with the knowledge base will provide an excellent guide to a more accurate solution.

This paper concentrates on the second component, the Algorithm Selection Problem. First of all, we focus on the problem of determining what it means to be the "best" algorithm. One of the problems with this is that this concept of "best" may mean entirely different things to different people. Therefore, we develop ways of letting the user, as much as possible, define his own environment by letting him place emphasis on what he thinks are important criteria in determining the best algorithm.

There are two ways of getting information to help find the best algorithm, *symbolically*, and *algorithmically*. This helps build goal strategies and rule strategies for the expert system knowledge base. The symbolic information is based on *priori* knowledge; that is, the information depends only on a symbolic analysis of the elliptic problem, knowledge from the user, theoretical performance knowledge of part or all of an algorithm, and actual performance data of algorithms applied to other problems. The algorithmic information is dynamic and is based on both *a priori* and *a posteriori* knowledge; that is, the information may depend on any of the following: calculations to study the behavior of the coefficients, forcing function and boundary data; trial, low accuracy, cheap solutions; trial solutions using the two or three "most promising" methods; or, additional input from the user after presentation of the initial results.

Once the information has been gathered, we develop techniques to use this information to pick a "best" method by studying what happens in simple cases and slowly building machinery to solve the more complex cases. One idea that is fundamental to this process is the idea of matching the characteristics of problems to the capabilities of algorithms. Finding the "best" algorithm can be viewed as finding the algorithm which is "most" capable of handling the problem. It would be nice if we knew definitely whether every algorithm could handle certain types of problems. Unfortunately, this is not the case and we are left with many gray areas. However, we show how we can overcome some of these difficulties by developing algorithmic and heuristic methods of determining these capabilities.

The Algorithm Selection Problem is not an easy problem and there is no simple, straightforward solution. However, through the machinery available in Elliptic Expert, we can begin to piece together this complex puzzle. One might think that in order to be successful, this system should provide the "optimal" algorithm to handle a particular problem. Because it is unclear what the "best" algorithm is, this can not be guaranteed in general. However, we consider it a success if we can provide a process that provides a better algorithm than the average scientist would obtain with just his own resources.

ODEXPERT: A Knowledge Based System For Automatic Selection of Initial Value ODE System Solvers

Mohamed Kamel

Department of Systems Design
University of Waterloo
Waterloo, Ontario N2L 3G1, CANADA

Wayne H. Enright

Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A4, CANADA

With the availability of a wide range of numerical software for solving initial value ordinary differential equations, users may face some difficulty in selecting an appropriate method to solve their problems. Unless they know their problem's properties and structure this selection will typically be based on factors such as ease and use of availability. Failing to select an appropriate method may lead to results that are not correct or, at best, correct results at high computational costs.

In an effort to automate this selection process, the authors are building a prototype knowledge based system for ODE-solver selection (ODEXPERT). The system gives the user an expert's selection based on automated examination of the problem and user's answers to the expert's questions.

In this paper we describe the design and implementation of ODEXPERT. More specifically, we explain the techniques, their implementation and the testing employed by the selector for representing a problem and automatically identifying its properties and structure (e.g., linear, nonlinear, stiff, sparse, banded, etc.). Based on these properties and structure, the selector, utilizing a knowledge base of decision rules that an expert would normally use, selects the appropriate solver. The intelligent selector is also intended to generate the parameters and any information that may be required by the solver. In case the system is not able to generate the required information from the problem and its properties, it will guide the user to provide such information.

The system is capable of acquiring knowledge about the problem to be solved from the problem specification and is capable of identifying the problem properties and structure and selecting the appropriate solution method. The basis architecture of the system consists of the following components:

1. *User Interface*. This represents the front end of the system. It provides an easy to use representation language which the user can specify the problem in. The input is parsed and information about the problem is represented for further use by other components. Among the features that are included in the parser is the ability to transform high order ODE's into a system of first order equations, the ability to detect the complexity of the problem and the ability to detect an inconsistency in the input.

2. *Structure Identification.* This component has routines for identifying the structure of the problem. For example, it has routines for checking whether a linear system of equation has special structure such as banded, sparse or block structure. Both structure of the problem and the Jacobian are identified. Among the properties that are identified are linearity and stiffness. Linearity of the problem is detected during the parsing of the input using a simple check. Further identification which includes simplification of expressions and identification of implicitness is also possible. For detecting stiffness a test has been developed based on pilot integration of the ODE system backward. The test is adequate for the purpose of the selection and has the advantage of being simple and computationally inexpensive compared to other stiffness detectness tests.
3. *Inference Mechanism.* The information gathered from the input and through the detection components are represented as knowledge which the system has to analyze in order to select a solver. The inference mechanism performs this analysis by correlating the provided information to rules in the knowledge base of the system.
4. *Knowledge Base.* This is the core of the system, it contains the rules and control information about the application domains as well as any acquired knowledge from the user. It is mainly divided into two parts. The first part includes rules that relate to the properties and structure of the ODE system and different possible decisions on recommending appropriate solvers. The second part includes knowledge rules about solution methods and the parameters used in their calling statements. These rules are to be used to generate the appropriate calling sequence to the selected method.
5. *Methods Bank.* This is a bank of the different methods for solving ODE systems.

The paper will discuss the details of the automatic detection of problem's structure and properties, the rules represented in the knowledge base and the user interface. We also discuss the implementation of the system and show some examples to demonstrate its performance.

Implementing Fast Fourier Transforms For Direct Solution of Poisson's Equation

Bert Bradford and Roland A. Sweet

Math Department, Campus Box 170
University of Colorado at Denver
Denver, CO 80204

This presentation will briefly describe compact algorithms used to incorporate the Cooley-Tukey Fast Fourier Transform (FFT) into the solution of finite difference approximations to Poisson's equation. All frequently occurring boundary conditions are considered, including those associated with staggered grids. Emphasis will be given to the complications involved in implementing software for these algorithms, and the potential for the application of automated code generation.

The potential role for automated code generation arises from the large number of specialized codes required for various combinations of boundary conditions, grid sizes, and grid orientations. In each spatial dimension, we must specify boundary conditions at both the left and right endpoint. Boundary conditions we consider include periodic, Dirichlet and Neumann. Furthermore, there is often a need to orient the grid such that one or both of the endpoints of the computational domain are staggered at half of a grid spacing. This leads to staggered Dirichlet and staggered Neumann boundary conditions. We have identified 11 such combinations of boundary conditions. When the Poisson equation is discretized, these boundary conditions are approximated by requiring the real sequence which represents the approximate solution to satisfy discrete analogs. The discretized boundary value problem is solved by the eigenvector expansion method. This method requires finding the eigenvalues and eigenvectors corresponding to the discretized differential operator which also satisfy the appropriate discrete boundary conditions. The discrete solution is expanded in terms of these eigenvectors. The efficiency of this algorithm results from the ability to calculate the coefficients in such eigenvector expansions using an FFT algorithm. Thus, for each of the boundary conditions discussed above, we have developed an FFT algorithm which computes the coefficients in the corresponding eigenvector expansion as efficiently as possible by eliminating all redundant computations which would occur in the full complex FFT, and without pre- or post-processing. The elimination of pre- and post-processing improves performance by reducing the number of data accesses. Such FFT algorithms are referred to as compact symmetric FFTs, and the overall algorithm as fast Poisson solvers.

Flexible grid sizes also contributes to the need for a large number of specialized FFT codes. The FFT algorithms we have developed are all general mixed radix algorithms. In order to achieve efficiency, the grid size must be chosen to be a product of many small primes or small composites. It is possible to implement software which accepts a general factor p as input. However, for typical values of p (2,3,4,5), there will be much loop control overhead which could be eliminated if specialized codes were generated for specific values of p. We have identified a need for at least 5 such specialized sets of code corresponding to $p = 2,3,4,5$ and a general factor p.

Another consideration which leads to the need for additional FFT codes is the ordering of the input and output sequences. All in-place FFT algorithms based on the splitting method require either the input or output sequence to be in bit-reversed order. If additional storage is used, then it is possible for both input and output sequences to be in natural order. This leads to 3 additional options which are independent of those already discussed above. For Fast Fourier Solvers, we prefer in-place algorithms which accept the input sequence in natural order. However, other applications may require the other two options.

If we combine the independent options discussed above, we obtain at least $11 \times 5 \times 3 = 165$ FFT variations. Furthermore, these algorithms have excellent potential for vectorization and/or parallel decomposition. this will lead to additional machine architecture dependent variations. From a review of the code in FFTPAK, we estimate that each FFT variation requires approximately 1,000 lines of FORTRAN code. Thus, we see excellent potential for the application of automated code generation tools. Experience in coding a few of these FFT algorithms indicates that these codes are sufficiently regular to make this feasible.

A Prototyping Environment For Ordinary Differential Equations

Toufic I. Boubez

Department of Biomedical Engineering, and
CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, N.J. 08855-1390

Andy M. Froncioni and Richard L. Peskin

Department of Mechanical and Aerospace Engineering, and
CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, N.J. 08855-1390

1. Introduction

In a general scientific interface environment, an interactive tool to allow scientists the capability of experimentation with differential equations is a desirable feature. The need to enhance scientific interfaces for numerical simulation has been discussed in the literature [1]. More specifically, when dealing with differential equations, there is need for a tool that is capable of handling a reasonable variety of initial value, final value and boundary value problems (including cases where boundary layers and shocks are present). Campbell [2] has built a prototype differential equation tool in Suntools. While this system has a graphical interface, its model flexibility is limited. Russo [3] has an extensive knowledge-based system capable of handling a wide class of problems, but with limited graphical and interactive interface. By constructing a differential equation system in Smalltalk-80 we are able to combine model flexibility and complete graphical interaction capabilities, while taking full advantage of the distributed processing environment available.

In this paper, we present an ordinary differential equation (ODE) tool that is designed to accept user input in the form of a problem statement (string form) of the ODE and its boundary conditions. The output is a graphical display of the solution. User interaction with the input allows rapid change of the equation, its boundary conditions, and any other parameters such as the discretization resolution and the solution domain.

2. Numerical Solution of ODE's

In general, the *n*th order differential equation

$$y^{(n)} = f(x,y,y^{(1)}, \ldots, y^{(n-1)} \tag{1}$$

can be reduced to a set of *n* first order equations

$$y_i' = g_i(x, y_1, \ldots, y_n) \quad i = 1, \ldots, n \tag{2}$$

by using some auxiliary functions.

This set of first order equations is then rewritten as a set of finite difference equations (FDE's) for each of the interior points in the discretized domain. The FDE's are linearized by re-writing them as a set of linear equations in the highest derivatives, taking the non-linear terms from the previous iteration. The equations are then expanded and a matrix equation,

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{3}$$

relating all the interior points in the discretized domain and incorporating the boundary conditions, is constructed. This equation is more specifically written as:

$$\mathbf{A}^{(k-1)} \cdot (\mathbf{x}^{(k)} = \mathbf{b}^{(k-1)}) \tag{4}$$

and solved in an iteration loop, using the solution from the previous iteration $(k-1)$ to construct the matrix and solve for the next iteration (k). The solution is thus relaxed until convergence is reached.

3. Numerical Solutions With the ODE Tool

In standard numerical methods, the equations are usually reduced and prepared beforehand, and the computer is only used in the final solution steps that require number-crunching (matrix solutions and relaxation iterations). The purpose of using our prototyping environment is to automate the initial stages as well as the number-crunching steps, so that an equation is processed from string form to a solution plot, thus allowing user interaction during the solution process.

The most important user-system interface step is the initial one, that of problem formulation. The problem equation has to be entered, parsed and processed. Using Smalltalk syntax rules, a string expression can easily be converted into an equation by using a scanner. The equation $u_{xx} = u$, for example, is entered as: '*uxx* − *u*' and, when scanned, produces a new instance of *EquationList*, a subclass of *Array*, having the value $(uxx - u)$. A set of higher order derivative variables $(a, b, c \ldots)$ are defined such that $a = du/dx$, $b = dx/dx$, etc...

The EquationList instance representing the n-th order equation is recursively scanned to reduce its order, and the above substitutions are applied, producing n first order equations. The FDEs are then produced by performing the following substitutions from another Smalltalk Dictionary instance:

$$\begin{aligned}
ux &\rightarrow (1/\text{delta}X)^*((u \text{ at}: (p+1)) - (u \text{ at}: p)) \\
u &\rightarrow 0.5^*((u \text{ at}: (p+1)) + (u \text{ at}: p)) \\
ax &\rightarrow (1/\text{delta}X)^*((a \text{ at}: (p+1)) - (a \text{ at } p)) \\
a &\rightarrow 0.5^*((a \text{ at}: (p+1)) + (a \text{ at}: p))
\end{aligned}$$

The resulting set of difference equations is linearized and written in the form given by equation 4. To effect this process, the system utilizes an 'expert' algebraic manipulation tool to reduce and discretize an *n*th order ODE to a series of matrix operations. The system builds a back-end MATLAB script and then invokes it via distributed computing on a back-end processor. The resulting solution vector is then graphically displayed in the Smalltalk environment.

An important feature of our system is its ability to let the user "computationally steer" the solution. The user can alter parameters, boundary conditions, and even change the equations during the solution process. Implementation of these features rely, in part, on the use of the Smalltalk incremental compiler.

4. Results

The environment mentioned above was tested on a number of boundary- and initial-value problems with very good results. The use of a relaxation method lessens the problem of Gibbs' phenomena associated with resolving sharp shock-type problems. For example, the following singularly-perturbed problem was tested on the system:

$$\begin{aligned} \textit{equation}:&\quad '0.1*Uxx-(u*ux)+u=0' \\ \textit{BC's}:&\quad u(-1)=1;\ u(1)=-1 \\ \textit{resolution}:&\quad \text{100 points} \end{aligned}$$

The problem was solved in approximately 135 seconds, using 44 nonlinear iterations. As is evident from Figure 1, no Gibbs' oscillation is present in the final solution. In addition, the shock region is properly represented. Similarly good results were obtained for other nonlinear shock- and smooth-type ODE's; the user can be confident of results obtained during prototyping.

5. Conclusion

This paper presents an ODE solution environment for prototyping. This tool has proved to be robust and correct in solving a number of difficult ODE systems, in particular, a singularly-perturbed, nonlinear ordinary differential equation. The tool implements computational steering for ODE's.

Several extensions to the system are being implemented. The Smalltalk algebraic manipulator will be supplemented with a commercial back-end system. This will decrease the manipulation time considerably and allow a greater range of ODE's to be solved. In addition, the system will soon be modified to handle several *n*th order nonlinear ODE's at once. An expert system is being incorporated to implement domain decomposition and high-order initial estimates for singular perturbation problems [4].

References

[1] PESKIN, R.L., WALTHER, S.S. and FRONCIONI, A.M., SMALLTALK – The next generation scientific computing interface?, *Mathematics and Computers in Simulation*, **31**, (1989), pp. 371–381.

[2] CAMPBELL, J.R. and McGAVRAN, L.P., An integrated distributed processing interface for supercomputers and workstations, submitted to *ASE'89, Applications of Supercomputers in Engineering*, Southampton University, UK, (Sept. 1989).

[3] RUSSO, M.F., Automatic generation of parallel programs using nonlinear singular perturbation theory, *Ph.D. Thesis, Rutgers University*, (1989).

[4] RUSSO, M.F. and PESKIN, R.L., Automatically identifying the asymptotic behavior of nonlinear singularly-perturbed boundary-value problems, submitted to *Journal of Automated Reasoning*, (1989).

Figure 1: The ODE Tool

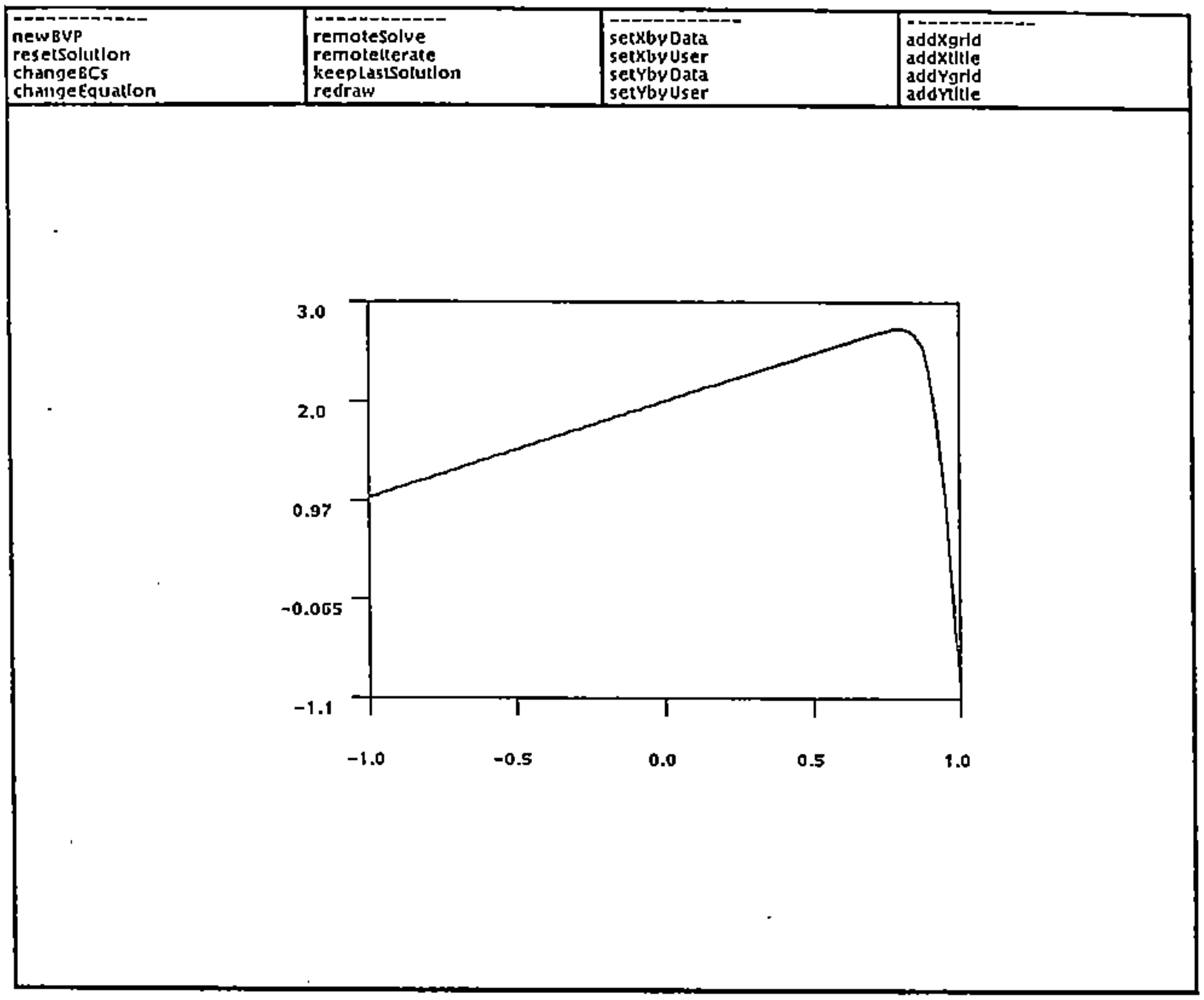


Figure 1. ODE tool display of a sample program

SESSION 2

A User's View of High-Performance Scientific and Engineering Software Systems in the Mid-21st Century

David J. Kuck

Center for Supercomputing Research and Development
University of Illinois
305 Talbot Laboratory
104 S. Wright Street
Urbana, IL 61801

Problem specification systems exist today on sequential machines for several problem domains. If parallel processing is to become commonplace, specification systems will have to exist for future parallel machines. In fact, the development of such systems may make the exploitation of parallelism easier because of the very high-level nature of the problem specification.

The value of problem specification systems will be discussed along several dimensions. These include case of use, target machine portability, ease of implementation, correctness of results, and speed of execution. Specification systems will be compared along each dimension with the languages and programming systems of the 1990's.

It seems clear that a number of 20th century issues will remain, even as computing reaches its centennial in the mid-21st century. For example, no programming system will satisfy all of its users. Thus, notions of reusability and extensibility must be explored in detail to make future generations of specification systems as flexible and useful as possible.

Software Productivity Issues For Scientists

Kenneth G. Wilson

Physics Department
The Ohio State University
174 West 18th Avenue
Columbus, OH 43210

There are three conflicting needs for organization in a large scientific program, namely (1) organization of program execution, (2) organization of data, and (3) organization of the explanation of the program and the design decisions which were made during its development.

The object oriented paradigm has been a notable advance because it elegantly meshes what otherwise would be conflicting organizational needs of program and data, in particular, it allows any data item to be accessible to any function, through the class and object mechanism.

The explanation of a program requires yet another form of organization, essentially a record of how the program was built up from an initial set of goals. At intermediate stages of this build up, both data structures and functions are incomplete, becoming more fleshed out as finer details of the program design are addressed. I will discuss ideas, partly from the Gibbs project, now led by David Gries, for how one might combine explanation oriented and object oriented approaches. I will also note the relationship of these ideas to Knuth's "literate programming".

A Study of Learning, Teaching, Optimization and Approximation

John R. Rice

Department of Computer Science
Purdue University
West Lafayette, IN 47907

A objective of this study is to create a mathematical framework for analyzing the concepts of learning and teaching. This framework is oriented towards enabling computational applications, such as now involved in expert systems, using rule based processors, neural networks, genetic algorithms and similar methodologies. The relation of this framework with the traditional ones from optimization and approximation theory, is analyzed with the view of identifying those aspects of the older, highly developed, analytic framework which can, might be, or cannot be, transferred to the new, more general framework.

SESSION 3

A Development Shell For Knowledge Based Systems in Scientific Computing

F. Rechenmann

INRIA, IMAG/lab. ARTEMIS
BP 53 X, 38041
Grenoble Cedex, FRANCE

B. Rousseau

Cap Sesa Innovation
33 rue du Vieux Chene, ZIRST
38240 Meylan, FRANCE

Introduction

Despite considerable progress in hardware and software, scientists still suffer from the lack of high level tools for numerical computing and programming. End users need more assistance in the choice, practical use and results interpretation of methods. Developers require more facilities for organizing and integrating existing methods. Present artificial intelligence and software engineering based solutions are neither satisfactory, nor generic. A more powerful solution relies on a tight integration, within scientific software, of the knowledge associated to methods and to entities handled by these methods. The user can thus express his goals in terms of tasks to be carried out on the data. Reasoning mechanisms help him in choosing, monitoring and sequencing the algorithmic modules that will enable him to attain his goals, while hiding the low level aspects. This paper describes the various key concepts which have been pointed out during the development of several dedicated knowledge-based systems in scientific computing. The requirements for both the knowledge representation model and the reasoning mechanisms it supports, are presented and discussed. A development shell for knowledge-based systems in scientific computing is being implemented and will be described. It satisfies all these requirements by making extensive use of the object-centered representation model.

Various projects and studies have indeed demonstrated the interest of the object-centered approach for the development of knowledge bases in scientific computing. Among them, the EDORA project aims at developing intelligent tools for dynamic modeling in biology. The SAID project proposes an intelligent system to help signal processing in the field of mechanical diagnostic. The EVE project addresses expert software for solving partial differential equations. Demonstrators of these three projects rely upon SHIRKA, an object-centered knowledge representation developed at INRIA (Institut National de Recherche en Informatique et Automatique). Important concepts have emerged during these projects.

1. Declarative Description of Both Entities and Methods

The basic concept is to embed rough data structures and software modules within object classes describing their properties and semantics. For instance, the *matrix* entity class has slots and constraints related to the size, form, structure, mathematical properties and computational origin of any matrix. The *Gauss-Marquard* optimization method class depicts input and output in terms of entities class (equations, parameters, variables, steps), execution conditions (non stiff linear equations) and additional information about numerical accuracy, precise input formats or memory and time resources needed. The user creates instances of these classes, asks for values of their slots, focusing on the semantics and ignoring the implementation aspects which are left to the system. These classes can be used to construct more abstract entities, which have a stronger meaning for the scientist. For instance, the *population dynamics model* entity class can be build by using the *equation, biological process* and *curve shape* classes and subclasses.

2. Distinction Between Tasks and Methods

In order to allow the incremental enrichment of the knowledge-based systems, one has to describe separately the "what" and "how" of the algorithmic modules they incorporate. The "what" corresponds to the description of the task performed by the module, while the "how" is a description of the method to achieve this task. Both tasks and methods are represented by object classes. A task describes execution conditions and effects in terms of entities. A method covers implementation features such as formats, data structures and resources. For instance, the FO4AEF NAG method class is attached to the *real matrix inversion* task class. Many different methods can be linked to the same task. The user does not need to access the methods directly, he only makes reference to the tasks. Adequate reasoning mechanisms, such as classification, are responsible for the choice of the adapted methods, depending on the context of the task. From the developers point of view, it is easy to add, remove or substitute methods without affecting the rest of the knowledge base.

3. Classification Inference Mechanism

Both entities and tasks classes are structured in specialization hierarchies. An entity is more specialized than another one when it describes a more constrained family of objects. A task is more specialized than another one when it is able to treat more specialized entities. These hierarchies, together with the inheritance mechanism they support, facilitate both the construction and the modifications of the knowledge base. They also give a sound support for the recursive classification inference mechanism, which automates the process of determining the best membership classes for a particular entity or task. The *matrix inversion* generic task gives an example of the development of such a hierarchy. Starting with an instance of the *matrix inversion* task class, and given a particular instance of the *matrix* entity class, the classification traverses this task hierarchy to find out the best specializations of task and thus the best methods. This process requires, of course, some information about the matrix, e.g., whether the matrix is symmetric or not. This information corresponds to the semantics of the slots of the entity class. Their values must be inferred or requested to the user. The classification mechanism appears as a very powerful and versatile mechanism to treat problems of entities characterization and of tasks selection.

4. Multiple Points of View on Entities

Our previous experiences in modeling mathematical objects, such as equations, parameters or models, have shown that, unexpectedly, the related knowledge is difficult to describe with classical object-centered presentations. One of the limitations is indeed the inability to describe an object from different, but complementary points of view. Even a basic example, such as the construction of a taxonomy of types of matrices, shows the point. It seems easy to characterize a matrix as being square or rectangle, banded or triangular, complex or real, symmetric or nonsymmetric, and so on and to propose a hierarchy of classes based on these characteristics. The problem is that a given matrix can be square and banded and real and symmetric. It is then necessary to build the corresponding class, using some multiple inheritance mechanism. The resulting knowledge base is a collection of highly tangled hierarchies. The solution lies in the introduction of points of view on a family of classes. In the example of the matrices, the following points of view could be introduced, structure, algebraic properties, nature of the elements. On each point of view, a strict hierarchy of classes can then be elaborated. An instance is allowed to belong simultaneously to several classes on distinct points of view. The classification process can then take place either on each point of view separately, or on a set of points of view.

5. Integration of Procedural Knowledge

The basic software modules are usually able to perform only a small part of a problem solving process (e.g., matrix computations). In order to solve a global problem, complex tasks describe a problem solving strategy while chaining several tasks together according to a current goal to be attained, through a process which alternates classification and planning phases. The classification algorithm is responsible for the selection of a task among a set of possible tasks, while the planning algorithm decides on the next task to be executed. Three operators are introduced to define a complex task, sequence, selection and recursion. The operands of these operators are either complex tasks, which are further decomposed, or elementary tasks, to which a set of algorithmic methods is directly attached. The selection operator introduces nondeterminism in the exploitation of a complex task description. If an elementary task fails, i.e., the attached methods fail to compute the expected results, the planning process is backtracked to the last decision node and the next application task is retained. The tasks embody the procedural expert knowledge. They allow for modularity through independence of description.

6. Dedicated Development Shell For Scientific Computing

We are currently implementing a dedicated shell, which embodies much of the features discussed above. The first operational version relies on SHIRKA, a frame-based system previously developed, which does not include the facility of multiple points of view, but offers the classification inference mechanism to handle both entities, tasks and methods. The user interface can display the recursive decomposition of the main problem solving tasks into subtasks. The user can point out a task to get more information on it. Other explanations can be obtained once entities, which are parts of the problem, have been characterized by the classification mechanism. The sequence of methods invoked during the problem solving process is recorded and can also be consulted by the user. A new version of this shell is currently under development. It makes use of the multiple points of view facility and is able to reason about the evaluation of the results of the methods. The sequence of tasks is then dependent of these evaluations.

CLAM and CLAMShell: A System For Building User Interfaces

David E. Foulser and William D. Gropp

Department of Computer Science
Yale University
New Haven, CT 06520

1. Summary

We present CLAMShell, a system for the rapid generation of graphical user interfaces and their subordinate numerical computations. Our approach is based on CLAM(R), the Computational Linear Algebra Machine, an interactive language for numerical computation and graphics, and makes use of X Window System graphics.

2. Problem

The essence of an interface is the mediation between the user's high-level visual and cognitive processes and the computer's numerous low-level calculations. The interface becomes more useful as it occupies more of the distance between human and computer.

A useful tool for building interfaces spans the full distance, addressing both the graphical and computational needs in a succinct and efficient manner. The user's side of the interface must make good use of graphical interface concepts, including mouse input, menus, and graphical feedback presented coherently and consistently. The computational side needs a succinct language that executes efficiently, and must be an open system that can tie together existing program fragments from a variety of sources. The tool itself must foster intelligent problem solution through a high-level description of the interface parts, a logical structure that fits the underlying operations, and an overall approach that makes problem solution easier with the tool than without it.

3. Current Approaches

There are many tools for the rapid generation of the graphical interfaces. While these tools include high-level languages for the graphical presentation of a problem solution, they often neglect the computational solution of the problem. For instance, some systems require programming at the level of C or FORTRAN, essentially providing only a high-level front-end. Others provide limited computational languages that are not suitable for medium or large scale scientific computing.

Other systems allow programmers to create special-purpose graphical user interface for existing programs, for instance by using X Window System library calls and the Athena widget set. Although this method allows great flexibility, it requires substantial programming effort and specialized programming knowledge. This approach is unlikely to save time on small projects or when used by novices.

Ideally, building a smart user interface should be simple enough that it saves time, even for short term projects and novice users. For this to be true, both the specification of the interface and the code to perform the actions must be simple to write and small in size. Furthermore, the interface must fit neatly with existing or new programs.

4. New Approach

Build on CLAM and the CLAMShell, we provide a simple user interface specification to construct a graphical user interface and computational programs in the CLAM language. The user interface side has menus, dialog boxes, and help text. It is mouse driven and also accepts command line input. The computational side runs the CLAM interpreter, which executes numerical computations in a matrix-vector style syntax, provides plotting commands, and can integrate C and Fortran routines from existing subroutine libraries. One can specify an entire problem solution using a convenient tree-structured menu system, or simply create a new interface to an existing Fortran or application.

4.1 Details

The CLAMShell is an X Windows client application that serves as a front-end to the CLAM interactive environment. CLAMShell includes panels for command-line input and CLAM text output. It is able to start the CLAM help librarian in another window, allowing the user to scan the help tree while viewing the text output window and executing CLAM computations.

The feature that most distinguishes the CLAMShell from a terminal window is the user definable menu. The contents of this menu and their computational actions are under user control. Any CLAM computation can be initiated from a CLAMShell user menu, which can also create recursive sub-menus, dialog boxes, and help text.

The CLAMShell User Menu consists of four predefined buttons (Exit Menu, Open New Menu, Delete Menu and Toggle Echo, discussed below) and the user-defined buttons. Each user-defined button is the root of a user menu-tree. Clicking a button in a user menu tree may execute some associated CLAM code, initiate queries through dialog boxes, display help text, and/or cause a sub-menu to be displayed.

Each user menu tree is defined in a single flat file. The format is a pre-order traversal of the menu tree, where each new tree node is announced by a single header line with the node's level number (starting at 0 for the root node) and its button title. All text between successive header lines (the end-of-file mark terminates the last node) is associated with the previous header line and comprises the menu node.

Each node of the menu tree is specified as a linear stream of control mixing graphical interface and computational actions. When a node's button is clicked, the initial actions are executed in order, the optional sub-menu is executed, and then the final actions take place. The block-structured node syntax includes the pairs %BEGIN-CLAM/%END-CLAM,%BEGIN-HELP%END-HELP,%BEGIN-QUERY/%END-QUERY, and the markers %SUB-MENU,%POP, and ! (for comments). On exit, the button pops up through a user defined non-negative number of menu levels toward the root of the menu tree.

Once a menu file has been created, there are two steps to incorporate it into the CLAMShell menu tree. First, the menu is compiled by a menu librarian program, converting the text file into a direct access file incorporating the tree structure, for fast traversal of the menu tree. Second, clicking the Open New Menu button on the CLAMShell display adds the compiled menu to the User Menu tree, closing the tree. When the User Menus button is clicked anew, the new user menu is represented by a button with a header line's title. Alternatively, selected user menus can be automatically included when the CLAMShell first starts up. The Delete Menu and Exit Menu buttons are self-explanatory (all menus in a user's tree have an Exit Menu button, so that a menu can be closed without activating any user buttons).

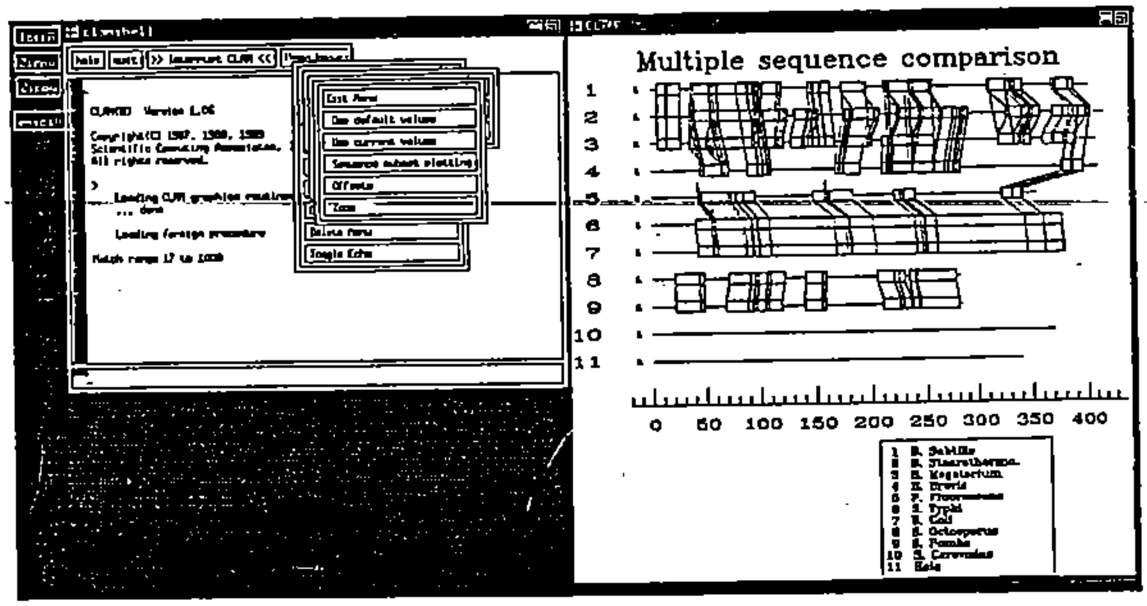
The benefits in graphical interface creation of this system are manifest. Moreover, our novel contribution lies in the fact that the interface integrates the user's side with the computational side. The CLAM program fragments that are embedded in the menu tree control numerical computation.

CLAM is an interactive environment for scientific computation and graphics. It has a high-level syntax built on matrix-vector operations and provides considerable support for linear algebra computations and graphics. CLAM's native data types include dense, banded, and sparse matrices of either real and complex values. It's built-in operations handle all six data types, with efficient algorithms for many sparse and banded operations. CLAM's workstation graphic use the X Window System, and include 2-D and 3-D line drawing, surface and contour plots, color polygon filling, and animation capabilities. CLAM can also call external Fortran and C routines through a convenient "foreign procedure" interface.

CLAM is meant to handle large computations. Its sparse and banded data structures and algorithms provide important computational methods in an interactive setting. Moreover, much of the internal computation in CLAM vectorizes efficiently, resulting in efficient code on problems of significant interest. Through the foreign procedure interface CLAM makes use of existing libraries of software, parts of application programs, and optimized (for vector, parallel, or architecture dependent computations) software. CLAM runs on a variety of computers, from SUN workstations to mini-supercomputers, and is intended to handle problems of substantial interest on the full range of machines.

5. Examples

Even though the CLAMShell and its tool for generating interfaces is still in the final stages of development, it has already been used in a variety of applications. Our early experience shows that the user interfaces are easy to generate and, more importantly, make the underlying computations easier to arrange. We will present a few examples of representative applications. One such example is a graphical interface to an existing package of parallel (biological) DNA sequence comparison routines and graphics output. In this system, most of the computation is done on a parallel computer and the data are filtered and displayed using CLAM graphics. Using CLAMShell, it was straightforward to impose a coherent control structure on the existing functions. The result is a package that is substantially easier for the developer to use, and that can now appeal to novice users. A screen dump from a representative sequence comparison is shown in Figure 1.



```
CLAMED  Version 1.0

Copyright(C) 1987, 1988, 1989
Scientific Computing Associates.
All rights reserved.

>
   Loading CLAM graphics routines
   ... done

   Loading foreign procedure

Match range 17 to 1000
```

Figure 1. CLAMShell with DNA sequence comparison menus

6. Conclusions

We have implemented a system for the rapid generation of complete user interfaces. Based on CLAM and CLAMShell, this system allows the user full control over the graphical user interface, numerical computations in the CLAM language and external libraries, and graphical output under the X Windows System. All parts of the interface can be specified in a single file using a block-structured format. Our initial uses of the system have proven it to be a rapid way of creating interfaces. It is a time-saving tool for numerical and graphical work, because the graphical interface naturally imposes a coherent structure on the underlying computations. Both that structure and the concise CLAM syntax lead to efficient problem statement and solution.

A Software Platform For Relaxation
(Extended Abstract)

Scott McFadden and John Rice*

Department of Computer Science
Purdue University
West Lafayette, IN 47907

1. The Relaxation Paradigm

Almost everyone has used a relaxation method to solve a problem in real life. Consider a person trying to place the thick false-floor tiles found in many computer laboratories. Often, the tiles will not fit into their allotted space exactly and thus cannot be simply dropped into an opening. Instead, neighboring tiles must be slightly raised and the entire area of tiles simultaneously lowered to the floor-frame, squeezing the new tile into place. In effect, the tiling system must be "relaxed" by deforming the position of neighboring tiles before a solution (placement) can be found. The key ingredients to solving this system is that the tiles have compressible strips and the human has a way of deforming the system (the "raise-and-push" strategy) to find a snug fit.

Consider the tiled floor to be a software system. As a software system the tiles are described by "objects" which have "methods" describing their placement. The tile objects also have methods telling how their positions may be deformed to seek a solution to the global system of which they are a part. However, simply providing the human user (the tile placer) with methods for deforming individual objects (tiles) is not enough. This system must also provide one or more global strategems by which tiles may be placed and relaxation may occur. The user may then apply these pre-supplied strategems, possibly simultaneously, to achieve a solution. Note that the tiling problem is very simple. In general, solutions by relaxation may call for iteration and/or other sophisticated strategies.

2. Software Relaxation

Software relaxation systems are object oriented programming systems which operate by successively relaxing interfaces between program objects. The objects in a software relaxation system are software structures which provide methods for deforming their internal state rather than enacting hard changes. Software relaxation is a superset of the mathematical relaxation used in several areas such as the solution of linear systems and PDEs. A programming platform for "software relaxation systems" is currently under active development at Purdue University. The full paper will describe the programming platform and examples of its application.

* This author's research was supported by a fellowship provided by IMSL Inc., Houston, Texas.

3. The System Architecture

Conceptually, our system is an arrangement of two types of entities: software objects and relaxers. The software objects are arranged in a graph, called a *software graph*, or SG for short. The nodes of this graph are the software objects and edges in this graph represent interfaces between these two objects. Figure 1 shows a software graph. The ''Helmholtz'' boxes are software objects which represent numerical partial differential equation problems on various domains. Each of these objects contains several methods for solving the Helmholtz equation $-A\Delta u + \mu u + f$ on the indicated domain. The values of μ and f are internal. The choice of method depends on which boundary conditions are to be shared with the neighboring Helmholtz object. The edges shown connecting the Helmholtz objects represent a shared interface – in this case a common geometric boundary.

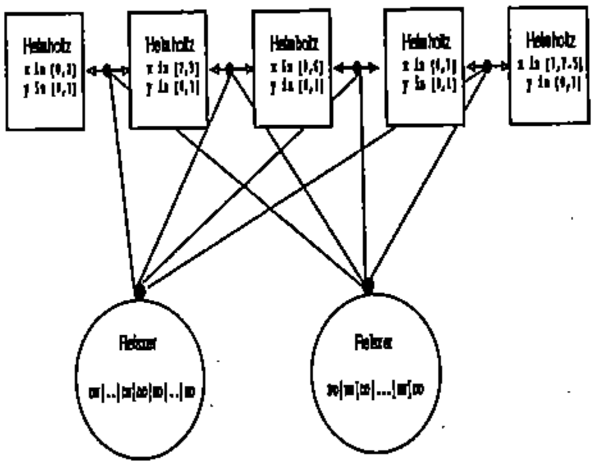


Figure 1. A software graph with attached relaxers

In Figure 1 several relaxes are seen attached to the SG. The relaxers can attach to either nodes (software objects) or edges (interfaces) in the SG. Currently the system provides a graphical editor for directly building and manipulating these graphs. The software objects and relaxers are chosen by selecting their icons from a library. The human user can enact the relazers by clicking on them. Each relaxer employs a different boundary-value relaxation method to achieve convergence. The letters below the ''**Relaxer**'' label indicate a boundary sharing scheme (D = Dirichlet, N = Neumann). The bars represent the geometric boundary. For more details see [1] where these methods are described and analyzed mathematically.

The goal of our system is to separate the functionality of the relaxer from that of the objects being relaxed. The reason for this is that a single relaxer may be applicable to several arrangements of software objects, and that a single arrangement of software objects may be approached with multiple relaxers or multiple instantations of the same relaxer. Also, this separation allows a powerful encapsulation of expertise. Individual internal variations in behavior may be programmed into software objects without regard to the embedded system. The system is not concerned with the inner workings of the software objects - only with their interfaces with each other. Also, different schemes for convergence may be programmed into a complicated arrangement by simply providing different relaxers.

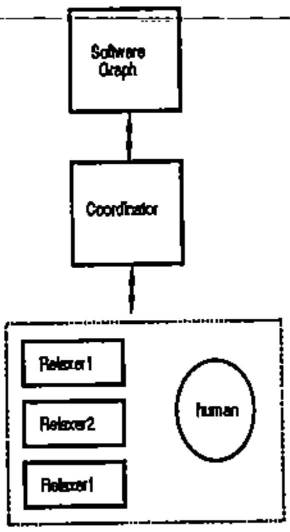


Figure 2. Conceptual Arrangement

Figure 2 is a high level description. Details will appear in the full paper. The human and relaxers work as a team. This structure allows a four-way sharing of expertise among the following parties:

- The person(s) who created the software objects. In the SG illustrated above, this expertise would be manifest in the internal numerical techniques for solving the Helmholtz equations on various domains and under various boundary conditions.
- The person(s) who used the software objects to construct the SG. This may involve specialized knowledge of how the domains fit together, or how the domains apply to some particular problem.
- The person(s) writing the relaxers. This involves a global knowledge of convergence properties.
- The person(s) executing the arrangement, in teamwork with one or more relaxers. The relaxers take high level commands from the human, via the coordinator, and spit back evaluations of their progress. The relaxer may not be able to solve the system all by itself and must rely on the human. The human may not know exactly how the relaxers work, but may be able to spot common sense things the relaxer doesn't.

4. Related Work

This system draws upon work done in several fields. Most fundamentally, we have drawn upon work in *object oriented programming*: See [26], [24], [20], [18], [17]. The graphical editor is basically a *visual programming* system: See [31], [30], [29], [28], [27], [25], [23]. The use of objects as atomic building blocks is reminiscent of the GARDEN "conceptual" programming system: [21], [22]. We draw the notion of relaxation from the field of *numerical computing*, but other fields have employed a notion of relaxation, for example in *constraint-based programming*: See [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [19].

References

[1] DANIELE FUNARO, ALFIO QUARTERONI and PAOLA ZANOLLI, An iterative procedure with interface relaxation for domain decomposition methods, *SIAM J. Numer. Anal.*, Vol. 25, No. 6, (Dec. 1988).

[2] SUTHERLAND, I., SKETCHPAD: A man-machine graphical communication system, *Ph.D. dissertation, Dept. of Electrical Engineering*, Massachusetts Institute of Technology, (1963).

[3] RICHARD E. FIKES, REF-ARF: A system for solving problems states as procedures, *Artificial Intelligence* **1**, (1970), pp. 27–120.

[4] ALAN BORNING, ThingLab – A constraint-oriented simulation laboratory, *Ph.D. Thesis, Univ. of Washington*, (July 1979). Also available as technical report *SSL-79-3*, Xerox Palo Alto Research Center.

[5] ALAN BORNING, The programming language aspects of ThingLab, A constraint-oriented simulation laboratory, *ACM Transactions on Programming Languages and Systems*, Vol. 3, No. 4, (Oct. 1981).

[6] GERALD JAY SUSSMAN and GUY LEWIS STEELE, JR., CONSTRAINTS – A language for expressing almost-hierarchical descriptions, *Artificial Intelligence* **14**, (1980), pp. 1–39.

[7] GUY STEELE, The definition and implementation of a computer programming language based on constraints, *Ph.D. Thesis, MIT*, (Aug. 1980). Also available as *technical report MIT-AI-TR 595*.

[8] ROBERT DUISBERG, Constraint-based animation: The implementation of temporal constraints in the animus system, *Ph.D. Thesis, Univ. of Washington*, (1986). Also available as *Univ. of Washington Computer Science Dept. technical report 86-09-01*.

[9] DAN INGALLS, SCOTT WALLACE, YU-YING CHOW, FRANK LUDOLPH and KEN DOYLE, FABRIK: A visual programming environment, *Proceedings of 1988 Conference on Object-Oriented Programming Systems, Languages and Applications*, (Sept. 1988), pp. 176–190.

[10] DANNY EPSTEIN and WILF R. LaLONDE, A smalltalk window system based on constraints, *Proceedings of 1988 Conference on Object-Oriented Programming Systems, Languages and Applications*, (Sept. 1988), pp. 83–94.

[11] ALAN BORNING, ROBERT DUISBERG, BJORN FREEMAN-BENSON, ALEX KRAMER and MICHAEL WOOLF, Constraint hierarchies, *Proceedings of 1987 Conference on Object-Oriented Programming Systems, Languages and Applications*, (Oct. 1987), pp. 48–60.

[12] JOHN H. MALONEY, ALAN BORNING and BJORN N. FREEMAN-BENSON, Constraint technologies for user-interface construction in ThingLab II, *Proceedings of 1989 Conference on Object-Oriented Programming Systems, Languages and Applications*, (Oct. 1989), pp. 381–388.

[13] BJORN N. FREEMAN-BENSON, A module mechanism for constraints in smalltalk, *Proceedings of 1989 Conference on Object-Oriented Programming Systems, Languages and Applications*, (Oct. 1989), pp. 389–396.

[14] PEDRO A. SZEKELY and BRAD A. MYERS, A user interface toolkit based on graphical objects and constraints, *Proceedings of 1988 Conference on Object-Oriented Programming Systems, Languages and Applications*, (Sept. 1988), pp. 36–45.

[15] GREG NELSON, JUNO: A constraint-based graphics system, *Proceedings of 1985 SIGGRAPH Conference*, Vol. 19, No. 3, (July 1985).

[16] ALAN BORNING and ROBERT DUISBERG, Constraint-based tools for building user interfaces, *ACM Transactions on Graphics*, Vol. 5, No. 4, (Oct. 1985), pp. 345–374.

[17] JOSEPHINE, CALLEF: Encapsulation, reusability and extendibility in object-oriented programming languages, *Journal of Object-Oriented Programming*, (Apr/May 1988), pp. 12–35.

[18] GUL AGHA, An overview of actor languages, *SIGPLAN Notices*, Vol. 21, No. 10, (Oct. 1986), pp. 58–67.

[19] RALPH L. LONDON and ROBERT A. DUISBERG, Animating programs using smalltalk, *IEEE Computer*, **18(8)**, (Aug. 1985), pp. 61–71.

[20] BJARNE STROSTRUP, An overview of C++, *SIGPLAN Notices*, Vol. 21, No. 10, (Oct. 1986), pp. 7–18.

[21] STEVEN P. REISS, A conceptual programming environment, *Proceedings of the 9th International Conference on Software Engineering*, IEEE, (1987), pp. 225–235.

[22] STEVEN P. REISS, An object-oriented framework for graphical programming, *SIGPLAN Notices*, Vol. 21, No. 10, (Oct. 1986), pp. 49–57.

[23] STEVEN P. REISS, PECAN: Programming development systems that support multiple views, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 3, (Mar. 1985), pp. 276–285.

[24] ROB STROM, A comparison of object-oriented and process paradigms, *SIGPLAN Notices*, Vol. 21, No. 10, (Oct. 1986), pp. 88–97.

[25] GRETCHEN P. BROWN, RICHARD T. CARLING, CHRISTOPHER F. HEROT, DAVID A. KRAMLICH and PAUL SOUZA, Program visualization: Graphical support for software development, *IEEE Computer*, **18(8)**, (Aug. 1985), pp. 27–35.

[26] RALPH E. JOHNSON and BRIAN FOOTE, Designing reusable classes, *Journal of Object-Oriented Programming*, (June/July 1988), pp. 22–35.

[27] ROBERT J.K. JACOB, A state transition diagram language for visual programming, *IEEE Computer*, **18(8)**, (Aug. 1985), pp. 51–59.

[28] SHI-KUO CHANG, Visual languages: A tutorial and survey, *IEEE Software*, **4(1)**, (Jan. 1987), pp. 29–39.

[29] GEORG RAEDER, A survey of current graphical programming techniques, *IEEE Computer*, **18(8)**, (Aug. 1985), pp. 11–25.

[30] ROBERT V. RUBIN, ERIC J. GOLIN and STEVEN P. REISS, ThinkPad: A graphical system for programming by demonstration, *IEEE Software*, (Mar. 1985), pp. 73–79.

[31] NAN C. SHU, *Visual Programming*, Van Nostrand Reinhold Company, New York (1987).

Towards an Expert System For Solving Systems of Linear Equations

Stefan Koenig and Charles P. Ullrich

Institut fuer Informatik
Universitaet Basel
Mittlere Strasse 142
CH-4056 Basel, SWITZERLAND

In recent years numerical algorithms have been developed for solving systems of linear equations with high and guaranteed accuracy. Their standard versions are so powerful that even for ill-conditioned problems often more than 10 digits of the results are validated. The effort for calculating a solution by these algorithms turned out to be nearly independent of the condition of the problem. But many users are dealing with well-conditioned problems and expect a better performance of the problem compared, for instance, with Hilbert matrices. In particular, a solution with only three or four correct significant digits asked form should need a fraction of computing time required for a solution with 13 or 14 correct significant digits; otherwise the user will usually not accept the algorithm.

In this paper a first study of an expert system, implemented in PASCAL-SC, is discussed, which applies self-validating methods in a sophisticated manner to a given system of linear equations. Without any help of the user the system calculates the result in a computation time appropriate to the condition of the problem and the number of correct significant digits required by the user. The strongest algorithm available is the Linear System Solver (LSS), which is discussed in [4], whereas the ordinary Gaussian Algorithm using simple interval operations is faster with the factor 2 to 6, but failing in a number of cases. Based on the experience of solving thousands of systems of linear equations the expert system applies a detailed criterion, which algorithm is to be executed first. The decision is depending on the number of unknowns, the required accuracy, the accuracy of the input data (because intervals are allowed too) and a parameter expressing the variation of the components of the matrix. If the Gaussian Algorithm is executed and the required accuracy cannot be achieved in contrast to the expectations, the strong algorithm LSS is started without any use of the results calculated before. Compared with the recent implementations the applied algorithms include some improvements, which modify the control flow depending on intermediate results and lead to an essentially better performance.

References

[1] G. BOHLENDER, Ch.P. ULLRICH, J. WOLFF VON GUDENBERG and L.B. RALL, PASCAL-SC: A computer language for scientific computation, *Academic Press*, Orlando, Florida, (1987).

[2] E. KAUCHER, U.W. KULISCH and Ch.P. ULLRICH (eds.), Computerarithmic: Scientific computation and programming languages, B.G. Teubner, Stuttgart, (1987).

[3] U.W. KULISCH and W.L. MIRANKER (eds.), A new approach to scientific computation, *Academic Press*, New York, (1983).

[4] S.M. RUMP, Solving algebraic problems with high accuracy, in [3], pp. 51–120, (1983).

Numerical and Statistical Knowledge-Based Front-Ends Research and Development at NAG

J. Chelsom, D. Cornali and I. Reid

Numerical Algorithms Group Limited
Oxford OX2 8DR, UK

This paper presents the current research being conducted at the Numerical Algorithms Group (NAG) Limited in numerical and statistical knowledge-based front-ends (KBFEs). NAG has been developing and distributing numerical and statistical software since 1970. Mark 13 of the NAG FORTRAN Library [8], the company's principal product, contains 748 use level routines supporting a wide range of numerical and statistical algorithms on a variety of computing platforms. Additionally, NAG handles a variety of other products, including Ada and graphics libraries, as well as packages (e.g., GLIM[1]).

General purpose numerical software libraries are enormous bodies of information. Ironically, the very generality and breadth of coverage of these libraries can pose formidable problems to the user. These include: choosing the appropriate solver, applying the solver (e.g., writing a calling program) and interpreting the results.

NAG has chosen to develop KBFEs as one way to bridge the gap between the problem (the user's application domain) and the solution (the package/library). KBFEs would serve NAG's present users more effectively, as well as broadening the audience of potential users. This allows the user to focus on solving the problem at hand rather than mastering the software. KBFEs are the key to the long term survival of large software libraries as the trend away from the direct use of traditional programming languages continues. Without KBFEs, those users who are *less conversant* with computing will seek other solutions. In general, other solutions will be inferior, since they will not be able to call on the enormous accumulation of algorithmic expertise encapsulated within a large, tried and tested, numerical subroutine library.

Three KBFE projects currently underway at NAG are now discussed: **KASTLE**, **IRENA** and **FOCUS**.

KASTLE (Knowledge-Assisted Selection Tool for Library Environments [7]) is a UK Government (DTI/SERC Teaching Company Scheme) funded project to develop a KBFE for the NAG FORTRAN Library. NAG and The Royal Military College of Science, Shrivenham, are collaborating on this project to develop a KBFE addressing the problem of routine selection. KASTLE consults library-wide knowledge about routine characteristics to provide advice on routine selection. This will be extended to provide program generation and run-time error analysis capabilities. Now in its third year, KASTLE has delivered two prototype systems.

IRENA (Interface between REduce and NAg) is a project to link the symbolic manipulation capabilities of REDUCE [5] with the numerical solving abilities of the

[1] Generalized Linear Interactive Modeling package [6].

NAG Fortran Library. The system fits into the REDUCE framework and as such provides an interactive front-end to the NAG Fortran Library. However, IRENA also simplifies the use of the Library, by:

- setting many of the *housekeeping* parameters,
- using sensible defaults for the control parameters,
- providing high level data structures,
- enhancing the failure messages, and
- creating Fortran programs, executing them and providing the relevant results.

FOCUS (Front-ends for Open and Closed User Systems [1] is a major knowledge-based system project under the European Community ESPRIT II advanced information technology programme. NAG and the seven other FOCUS partners will develop tools, techniques, and commercial prototypes for KBFEs to *open* (software libraries) and *closed* (application packages) numerical systems. FOCUS will consume over 60 man-years of effort over a four year period, which began in December 1988.

FOCUS has seen significant progress in its first year. Some of the more important results include:

- definition of the overall KBFE architecture,
- first prototypes of key modules (which include the presentation layer and the back-end manager),
- initial extracted tools from existing KBFEs,
- research on user modeling and adaptive interfaces [2], and
- work on user and system developer evaluation methodologies.

The three projects mentioned above form the basis of research and development into KBFE technologies at NAG. However, there are other projects which laid much of the foundation for this work and two of these are described briefly below.

GLIMPSE (GLIM + Prolog + Statistical Expertise [9]) was a UK Government Alvey project carried out in collaboration with the Departments of Mathematics (Statistics Section) and Computing at the Imperial College of Science and Technology, London. The aim of the project was to build a KBFE to the GLIM 3.77 statistics package. GLIMPSE the commercial product was released in February 1989.

NAXPERT (NAg eXPERT [10] was a project carried out in collaboration with Professor Colin Cryer and colleagues at the Westfälisch Wilhelms Universität, Münster. The system was an early attempt to build a KBFE for the NAG Fortran Library. The project proved successful for subsets of the library, but encountered problems when trying to scale up to the full library.

To conclude, NAG is committed to the development of KBFEs to aid the user in the use of its software. Further, NAG through FOCUS is actively involved in the development of tools and techniques for building KBFEs to support all existing numerical and statistical software. The wealth of knowledge contained in the existing software should be be discarded and since users require a more friendly environment in which to solve their problems, NAG will continue research and development in the area of KBFEs for numerical and statistical software.

References

[1] FOCUS Consortium. FOCUS: Internal reports of the FOCUS project. Contact NAG Limited, Oxford, UK (1988/9).

[2] CORNALI, D.J., Adaptive strategies for knowledge-based front-ends, *Master's Thesis, Univ. of Edinburg*, (1989).

[3] FORD, B. and CHATELIN, F. (eds.), Problem solving environments for scientific computing, *Elsevier*, (1987).

[4] FORD, B., HAGUE, S.J. and Iles, R.M.J., Numerical knowledge-based systems, *Mathematics and Computers in Simulation*, 31:395–400, (1989).

[5] HEARN, A.C., *REDUCE User's Manual, Version 3.3*, The RAND Corporation, (July 1987).

[6] NAG Limited, Oxford, UK, *GLIM System Manual, Release 3.77*, (July 1986).

[7] NAG Limited, Oxford, UK, *KASTLE: Internal Project Reports of the Teaching Company Scheme Project*, (1987/8/9).

[8] NAG Limited, Oxford, UK, *NAG Fortran Library Manual, Mark 18*, (Sept. 1988).

[9] O'BRIEN, C.M., The GLIMPSE system, *Technical Report TR 12/88*, NAG Limited, Oxford, UK, (1988).

[10] SCHULZE, K. and CRYER, C., A prototype expert system for numerical software, *SIAM Journal Sci. Stat. Comput.*, **9(3)**, (1988).

SESSION 4

Intelligent FORTRAN Compiler For Parallel Computers

Vasanth Balasundaram and Geoffrey Fox

Mail Code 206-49
California Institute of Technology
Pasadena, CA 91125

For at least the next few years, FORTRAN will be a critical language for scientific computation. One can view it either as a primary user interface or as a portable "machine-language", for which excellent compilers exist and which is a target for more user-friendly systems. Mapping a scientific problem onto a high performance computer can be viewed as a hard optimization problem, where one minimizes some combination of user program development time and production program execution time. We consider how expert systems, neural networks, simulated annealing and related methods can be integrated into a FORTRAN compiler to both give better code and feedback to the user on the appropriateness of a particular problem to particular hardware.

Automatic Parallel Program Instrumentation: Applications in Performance and Error Analysis*
(Extended Abstract)

B. Bliss, M.-C. Brunet and E. Gallopoulos†

Center for Supercomputing Research and Development
University of Illinois at Urbana-Champaign
Urbana, IL 61801

1. Introduction

With the proliferation of supercomputers the need has arisen to develop appropriate parallel programming environments which will facilitate their efficient usage [(1,6,11,13)]. Apart from the well-known need for restructuring compilers, good debuggers, visualization tools, etc., the users are faced with an ever increasing number of tradeoffs that they have to settle for the tuning of their application. For instance, the algorithm selection problem becomes very complicated since the performance of the algorithms used is no longer simply a function of the traditional operation count, but also depends on many other factors such as the amount of parallelism available in the algorithm, the tradeoffs between more parallelism overhead and less favorable access patterns to the memory hierarchy, etc. An area which has received somewhat less attention, for reasons which are well explained by Skeel in [12], is the development of tools for providing information to the user regarding the quality of numerical results from his algorithm. In view of the flurry of research in developing new algorithms which would run effectively on multiprocessors, we feel that such tools would be very valuable, even though unlike most other modules in these programming environments, they require a greater level of sophistication from the user, for a proper interpretation of the returned information.

In the full length paper, we will describe the *design* and the *use* of a system, developed to fulfill some of the aforementioned needs. In this extended abstract, we summarize part of our work.

The system is based on a *program instrumentation* preprocessor. This preprocessor can be used for several, seemingly different goals, such as obtaining a count of floating-point operations, or obtaining information about the error behavior and stability of an algorithm ([2]). In all cases the user is given the tools to select which sections of the code he wants analyzed.

In order to achieve these goals, several libraries were developed for linking with the preprocessor. These are:

* Research supported by the National Science Foundation under Grant NSF CCF-8717942. Additional support provided by AT&T Grant AT&T AFFL67 Sameh and the Department of Energy under Grant DOE DE-FG02-85ER25001.

† Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

- *OP_CT* – designed to provide a report of floating-point operation activity,
- *LGRAPH* – designed to generate a computational graph of a given algorithm,
- *PERTURB* – designed to perform a statistical analysis of algorithm stability.

This system has been integrated (along with other tools) in the Cedar parallel programming environment, which is based on the Cedar Fortran restructuring compiler. Cedar Fortran ([7]), is the language for Cedar ([4)], the hierarchical cluster-based vector multiprocessor developed at the University of Illinois' Center for Supercomputing Research and Development.

2. Preprocessor Design

The preprocessor performs much of the same work as any vectorizing compiler would. Specifically, it must translate the source code into 3-address statements, creating temporary variables to hold intermediate results, and generate code for the calculation of each vector operation's length an memory stride. It differs from a vectorizing compiler in that the target language is a subset of the source language, extended Fortan 8*x*.[1]

If the preprocessor is invoked without any options, the behavior of the target code is equivalent to the original source code. Command line options are specified to replace any operations, type coercions, or intrinsic function calls of a specified data type with user-defined subroutine calls. These routines must be written by the user, or he may include any of several libraries written for different applications. A naming convention allows the author of these subroutines to know the data types of the arguments, and whether the operation is performed upon scalar or array operands. One argument is a slot to which the routine should assign the return value of the operation. This return value and/or any of the arguments may be arrays. In such a case, the array is passed as an argument by reference, followed by the memory stride(s), and the vector length (common to all vector arguments).

For certain applications it may be necessary to pass along additional information with the arguments. With data objects of a specified type, any number of variables of any type may be associated. These *associated variables* follow the original variables in the subroutine argument list.[2]

3. Floating Point Count Activity With OP_CT

When the appropriate options are specified, the preprocessor replaces each floating point operation with a corresponding subroutine call. In addition to performing the appropriate operation, the subroutine stores information regarding the type of operation, and the vector length and stride(s). After the computation has completed, a routine is called[3] which produces a report summarizing the activities involving floating-point

[1] The source language is Cedar Fortran, concurrent Fortran 8x extended to allow access to the hierarchical memory of Cedar.

[2] Therefore it is important that the data types of actual and formal parameters match when using this option.

operands.

This report breaks down each operation, intrinsic function call, or type conversion (explicit or implicit) according to data type, and whether the operation was performed upon scalar or vector operands (and in the case of binary operations or certain intrinsics, whether there were one or two vector operands). For vector operations and intrinsic function calls, the average vector lengths and strides of the arrays accessed are also calculated. The report is generated automatically in a file in the user's current directory, requiring no intervention from the user. Figure 1 provides an example report file from the use of OP_CT on a parallel algorithm for the solution of a block tridiagonal system coming from the discretization of Poisson's equation on a rectangle with a 31 $\times$ 31 grid ([5]).

4. Error Analysis Tools

In order to assess the quality of numerical results, two different methods have been implemented in this instrumentation system. The first method is based on error linearization ([9,8]) and computes the condition number of each output of the program with respect to each intermediate result and input together with a measure of the overall numerical stability. The second is based on a statistical approach (cf. [3] and references therein).

4.1. Deterministic error analysis with LGRAPH.

LGRAPH is used to generate *the computational graph* of the algorithm, that is a graph of the floating point computations performed. The graph is obtained in a form appropriate for direct input into the automatic error analysis tool described in [8]. This is achieved by replacing each floating point operation with a subroutine call, and associating with each floating point variable an integer variable to be passed along as an argument to the subroutine calls. This associate variable is used as a pointer to a structure specifying the operation performed. Pointers to the operands are stored, allowing the generation of an encoded floating point computational graph after processing is complete.

Note that the "minicompiler" described in [10] was also designed to produce a computational graph, but for only a severely restricted subset of Fortran.

4.2. Statistical methods and PERTURB.

For each elementary operation, a function call is generated by the preprocessor that *slightly*[4] perturbs the result. At runtime, the corresponding perturbed final results are automatically saved in a file. After obtaining the results from multiple runs, **PERTURB**, can compute important statistical information (mean, standard deviation, etc.) which can characterize the stability of the algorithm ([3]).

[3] The call is inserted by the preprocessor.

[4] In a relative way, within the corresponding epsilon of the machine.

5. Examples

Examples will be provided from the use of the OP_CT, LGRAPH, and PERTURB libraries. These were also integrated in the *X*-window based parallel programming environment described in [13]. Figure 2 for example, shows the selection of a code fragment for the application of OP_CT, and Figure 3 a report file, both generated under the environment of [13].

6. Further Uses

In addition to the above uses, the system may be augmented to handle a variety of additional applications, although no software is currently developed for them. For example, interval arithmetic could be implemented by associating with each floating point data object a pair of ''interval bounding'' floating point variables to be passed along as arguments to the subroutines replacing operations and intrinsic calls.

One may also use an arbitrary number of these associated variables to hold extra precision for the mantissa, or may use an associated variable to index an area of memory managed as a heap, storing floating point numbers to arbitrary precision, effectively producing an implementation of infinite precision arithmetic similar to that found in many Lisp systems.

Since the creation of any such additional library involves tedious coding, we will discuss the potential for automating the process of library generation.

References

[1] T. BEMMERL, An integrated and portable tool environment for parallel computers, in *Proc. 1988 Int'l. Conf. Parallel Processing*, Vol. II, Software, Univ. Park, Penn., (Aug. 1988), Penn. State Univ. Press, pp. 50–53.

[2] B. BLISS, Instrumentation of fortran programs for automatic error analysis and performance evaluation, *Master's Thesis, Department of Computer Science, Univ. of Illinois at Urbana-Champaign*, (Jan. 1990).

[3] M.-C. BRUNET, Contribution á la fiabilité de logiciels numériques et á l'analyze de leur comportement: une approche statistique, *Ph.D. Thesis, Université Paris IX Dauphine, U.E.R. Mathematiques de la decision*, (Jan. 1989).

[4] P.A. EMRATH, D.A. PADUA and P.C. YEW, Cedar architecture and its software, in *Proc. 22nd Hawaiian Int'l. Conf. System Sciences*, (Jan. 1989). Also CSD TR-796.

[5] E. GALLOPOULOS and Y. SAAD, Parallel block cyclic reduction algorithm for the fast solution of elliptic equations, *Parallel Comput.*, **10**, (Apr. 10), pp. 143–160.

[6] D. GANNON, Programming tools for parallel systems, in *Fourth SIAM Conf. Parallel Proc. Sci. Comput.*, Chicago, (Dec. 1989).

[7] M. GUZZI, *Cedar Fortran Programmer's Manual*, Tech. Rep. 601, Center for Supercomputing Research and Development, (June 1987).

[8] J.L. LARSON, M.E. PASTERNAK and J.A. WISNIEWSKI, Algorithm 594: Software for relative error analysis, *ACM TOMS*, **9**, (Mar. 1983), pp. 125–130.

[9] J.L. LARSON and A.H. SAMEH, Algorithms for roundoff error analysis – a relative error approach, *Computing*, **24**, (1980), pp. 275–297.

[10] W. MILLER and D. SPOONER, Algorithm 532: Software for roundoff analysis, ii, *ACM Trans. Math. Softw.*, **4**, (Dec. 1978), pp. 369–390.

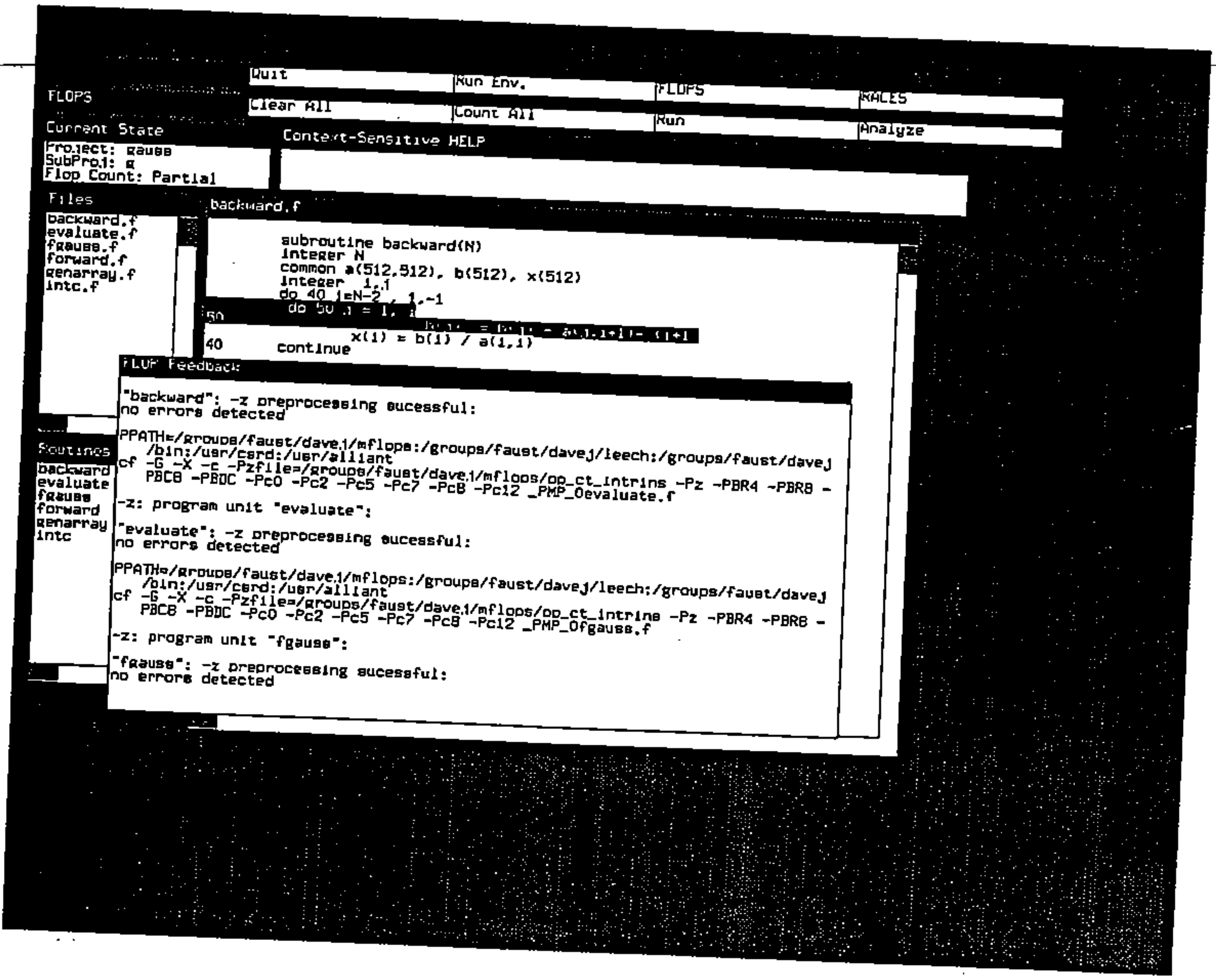
[11] C. POLYCHRONOPOULOS, M. GIRKAR, M. HAGHIGHAT, C.-L. LEE, B. LEUNG and D. SCHOUTEN, Parafrase-2: An environment for parallelizing, synchronizing and scheduling programs on multiprocessors, *Int. J. High Speed Comput.*, **1**, (1989).

[12] R.D. SKEEL, Safety in numbers: The boundless errors of numerical computation, *Tech. Rep. 89-3, Numerical Computing Group, Dept. of Computer Science, Univ. of Illinois, Urbana,* (Apr. 1989). Submitted for publication.

[13] J.V. GUARNA, D. GANNON, D. JABLONOWSKI, A.D. MALONY and Y. GAUR, Faust: An integrated environment for parallel programming, *IEEE Software*, (July 1989), pp. 20–27.

```
REAL * 4     ASSIGNMENTS

assignment   scalar  vec/sca total vector/  av vec  av vec  vec/vec total vector/ av vec  av vec    operation
             count   count    scalar ops    stride  length  count    vector ops   stride  length     total
----------  -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
    =         8967      8           31       8.0     3.9     1689          6040    3.5     3.6          15038
  where          0     42          114       1.0     2.7       42           114    1.0     2.7            228
----------  -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
  totals      8967     50          145       2.5     2.9     1731          6154    3.5     3.6          15266

REAL * 4     OPERATIONS

operation     scalar  vec/sca total vector/  av vec  av vec  vec/vec total vector/  av vec  av vec    operation
              count   count    scalar ops    stride  length  count    vector ops    stride  length      total
----------   -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
comparisons       5      42           114     1.0     2.7       0             0     0.0     0.0           119
     +         1157       0             0     0.0     0.0     543          4247    27.3     7.8          5404
 binary -       310    2599          6954     1.0     2.7    3396         10034     2.9     3.0         17298
  unary -      1109       0             0     0.0     0.0       0             0     0.0     0.0          1109
     *         1374    4759         12669     2.0     2.7    2520          6840     1.0     2.7         20883
     /           56    1339          3534     1.0     2.6       0             0     0.0     0.0          3590
     **          57       0             0     0.0     0.0       0             0     0.0     0.0            57
----------   -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
  totals       4068    8739         23271     1.5     2.7    6459         21121     7.2     3.3         48460

 unary operations are listed
 in vector/scalar columns

REAL * 4     INTRINSIC FUNCTION CALLS

   intrinsic          scalar  vec/sca total vector/  av vec  av vec  vec/vec total vector/  av vec  av vec    operation
   function           count   count    scalar ops    stride  length  count    vector ops    stride  length      total
-----------------   -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
atan ()                 10       0             0     0.0     0.0      0              0     0.0     0.0            10
cos ()                  46       0             0     0.0     0.0      0              0     0.0     0.0            46
dotproduct ()            0    1302          3534     1.0     2.7      0              0     0.0     0.0          3534
sqrt ()                 46       0             0     0.0     0.0      0              0     0.0     0.0            46
-----------------   -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
   totals              102    1302          3534     1.0     2.0  ******   ***********   *****   *****          3636

 totals, non-elemental and unary intrinsics
 are listed in vector/scalar columns

TYPE CONVERSIONS

  type        argument        scalar  vec/scl total vector/  av vec  av vec  vec/vec total vector/  av vec  av vec    operation
conversion      type          count    count   scalar ops    stride  length  count    vector ops    stride  length      total
----------  -------------   -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
real ()     INTEGER * 4        1073       0             0     0.0     0.0      0              0     0.0     0.0          1073
----------  -------------   -------  ------  ------------  ------  ------  ------  ------------  ------  ------  ------------
  totals      totals           1073       0             0     0.0     0.0      0              0     0.0     0.0          1073
 unary type conversions are listed
```

Figure 1: Report from OP_CT

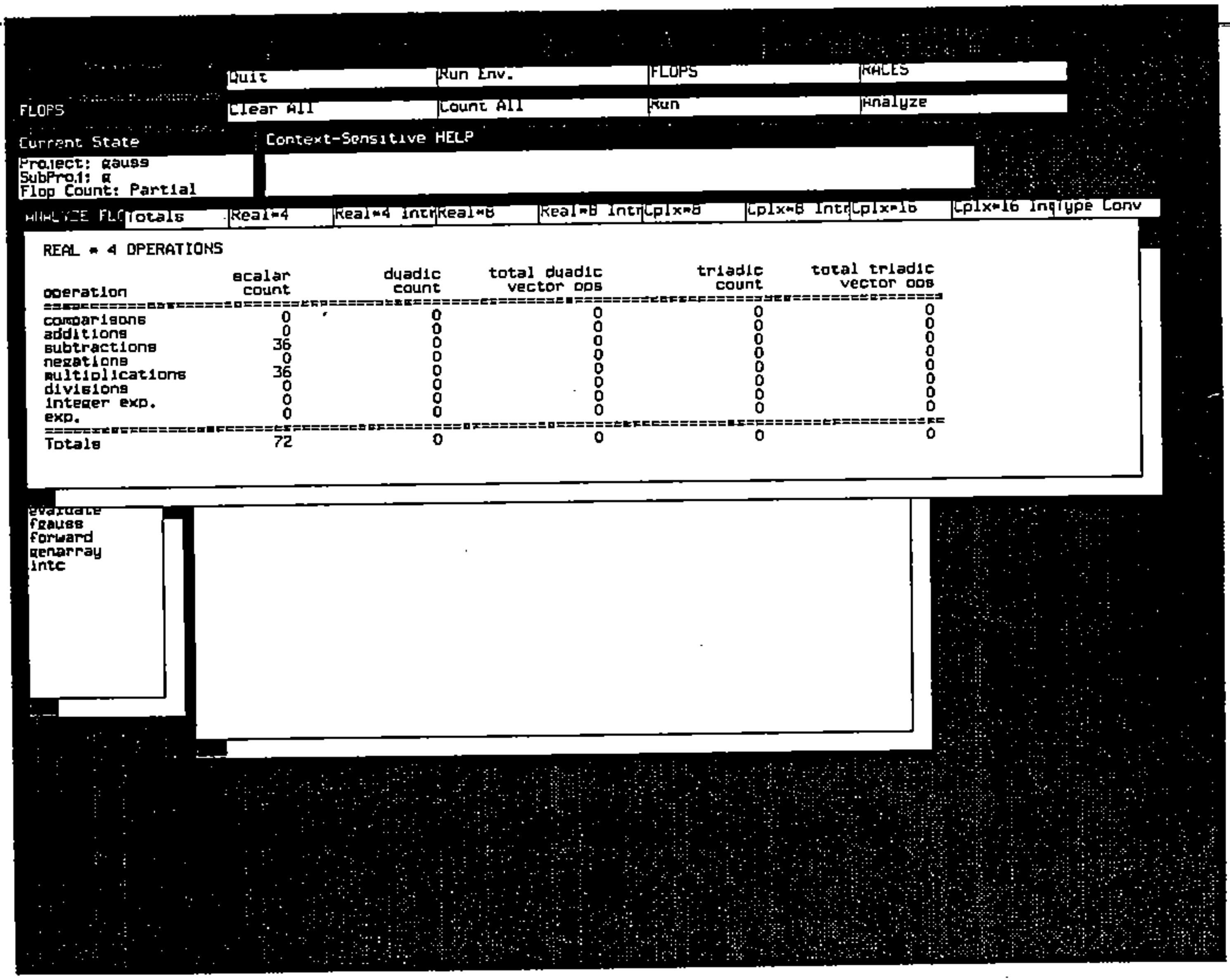


```
Project: gauss
SubProj: g
Flop Count: Partial
```

```
          subroutine backward(N)
          integer N
          common a(512,512), b(512), x(512)
          integer  i,j
          do 40 i=N-2 , 1,-1
50         do 50 j = 1, [illegible]
              [illegible]
40        continue
              x(i) = b(i) / a(i,i)
```

```
"backward": -z preprocessing sucessful:
no errors detected

PPATH=/groups/faust/davej/mflops:/groups/faust/davej/leech:/groups/faust/davej
   /bin:/usr/csrd:/usr/alliant
cf -G -X -c -Pzfile=/groups/faust/davej/mflops/op_ct_intrins -Pz -PBR4 -PBR8 -
   PBC8 -PBDC -Pc0 -Pc2 -Pc5 -Pc7 -Pc8 -Pc12 _PMP_Oevaluate.f

-z: program unit "evaluate":

"evaluate": -z preprocessing sucessful:
no errors detected

PPATH=/groups/faust/davej/mflops:/groups/faust/davej/leech:/groups/faust/davej
   /bin:/usr/csrd:/usr/alliant
cf -G -X -c -Pzfile=/groups/faust/davej/mflops/op_ct_intrins -Pz -PBR4 -PBR8 -
   PBC8 -PBDC -Pc0 -Pc2 -Pc5 -Pc7 -Pc8 -Pc12 _PMP_Ofgauss.f

-z: program unit "fgauss":

"fgauss": -z preprocessing sucessful:
no errors detected
```

Figure 2



Project: gauss
SubProg: g
Flop Count: Partial

```
REAL * 4 OPERATIONS

                      scalar        duadic     total duadic        triadic    total triadic
operation              count         count       vector ops          count      vector ops
=========================================================================================
comparisons              0             0             0                 0               0
additions                0             0             0                 0               0
subtractions            36             0             0                 0               0
negations                0             0             0                 0               0
multiplications         36             0             0                 0               0
divisions                0             0             0                 0               0
integer exp.             0             0             0                 0               0
exp.                     0             0             0                 0               0
=========================================================================================
Totals                  72             0             0                 0               0
```

Figure 3

ATHENA
A Knowledge Base System For //ELLPACK

C.E. Houstis, E.N. Houstis, M. Katzouraki,
T.S. Papatheodorou, J.R. Rice and P. Varodoglou

Computer Sciences Department
Purdue University
West Lafayette, IN 47907

1. Introduction

We consider the design and development of a parallel knowledge base for the parallel (//) ELLPACK [Hous 89] system for solving certain types of partial differential equations (PDEs). Its design objective is to reduce the overhead associated with the parallel processing of these types of computations. Specifically, it will provide, a) facilities for the automatic partitioning and allocation of the PDE computations to a variety of parallel machines and b) expert assistance for selecting "efficient" method/machine pairs. The //ELLPACK system allows many alternative ways to solve elliptic PDEs so the selection of a good way becomes a nontrivial task. The ATHENA expert system is designed to be able to produce expert assistance for the *method/machine selection problem.* Recall that //ELLPACK is designed to run on a multilevel hardware facility consisting of powerful workstations, machines with hundreds of kMIPS processors (MIPs = millions of instructions per second) and machines with tens of BIPS processors (BIPS = billions of instructions per second). Thus the efficiency of the computation will depend critically on the machines. ATHENA's unique design is its use of performance profiles and its ability to generate new performance profiles, and thus better selection capabilities, as it is used. The database facility uses stochastic methods to rank methods and machines using the performance profiles, it also selects the most relevant profiles and evaluates their validity. New data can be incorporated during "training" runs as well as from normal use of //ELLPACK.

Section 2 describes the various performance evaluation data used by ATHENA. The software organization and design goals of ATHENA are described in Section 3. Finally, Section 4 describes the inference mechanism of ATHENA.

2. Performance Evaluation Data

It is clear that the main performance objectives of a user are *accuracy* and *time*. In a PDE computation *accuracy* is controlled through the refinement of the grid and the discretization scheme used, while *execution time* depends on the speed of the targeted machine and the efficiency of the PDE solver. For a given machine, the computation of a solution within a certain accuracy (ε) and time frame (T) requires the selection of appropriate grid, discretization and solution schemes (method) plus an appropriate machine. If the machine is parallel, then the partitioning of the computation into load balanced, optimally parallel subtasks is also required. For the various steps of the computation a number of performance indicators are measured. We present them as *performance profiles*, one for each combination of PDE problem, method and machine. See [Boisvert, Rice and Houstis, 1979], [Boisvert and Rice, 1985, Chapters 8-11 and

Appendix A] for further details. For each such combination we collect, for different grid sizes, data on errors, execution time, linear system size, number of iterations and similar items. Aggregation techniques must be used in retrieving data, for no database can ever have all the data needed. The techniques used here include:

(a) *Machine Equivalences.* If we have execution time data for a VAX 11/780, a VAX 8800 and an Alliant FX80 then we use a conversion factor to estimate the execution time for all cases on any one of these three (or other) machines.

(b) *PDE Problem Associations.* For each PDE problem we collect (or can easily recompute) data on over 60 simple properties of the problem. Sixteen of these can be considered problem features (e.g., rectangular domain, no cross derivative term, Dirichlet boundary conditions) in the usual sense. In addition we have 36 possible features which are more subjective or computationally expensive (e.g., boundary layer present, variable smoothness, nearly singular). Eighteen of these refer to the PDE problem in general, eight to the operator and ten to the solution. These latter 36 features are graded on a scale of 0 to 100.

The machine equivalences allow us to extend the data to many machines in a straightforward manner. The PDE problem features allow us to take a new problem and find "close matches" to problems with existing performance data. Thus the aim of ATHENA is to collect several "close" problems and to estimate a performance profiles based on the existing data. The reliability of the estimate depends on the closeness and, of course, we must be prepared for the case where no relevant data exists, we discuss the action for this case later. In the case of parallel machines, the data points of the performance profiles are assumed to correspond to nearly optimal machine configurations for the corresponding grid sizes. Examples of such performance curves for sequential and parallel machines can be found in [Rice and Boisvert, 1985, Chapter 8-11], [Hous 88] and [Chri 88].

The ELLPACK project has accumulated an extensive database of performance measures for sequential machines, about 15,000 PDE solutions involving over 100 PDE problems, many methods and perhaps 10 combinations of compiler/operating system and machines. The data collected so far for parallel machines is, of course, still quite sparse.

3. ATHENA's Goals and Software Organization

ATHENA is an expert system whose knowledge base consists of performance profiles, which are automatically generated from a database of performance measurements and dynamically updated when the corresponding database is reorganized or enriched. The objective of ATHENA is to select the method (grid, discretization and solver) and machine based on the nature of the PDE problem and user's computational objectives (accuracy, time response). The software infrastructure of ATHENA consists of a performance evaluation facility, a facility for analyzing the data for various classes of problems, a facility for automatically generating performance profiles and an inference facility that provides an expert solution to selection problems.

The overall structure of the ATHENA system is shown in Figure 1. The //ELLPACK system [Houstis et. al., 1989] comprises the three boxes on the left while the ATHENA system is on the right.

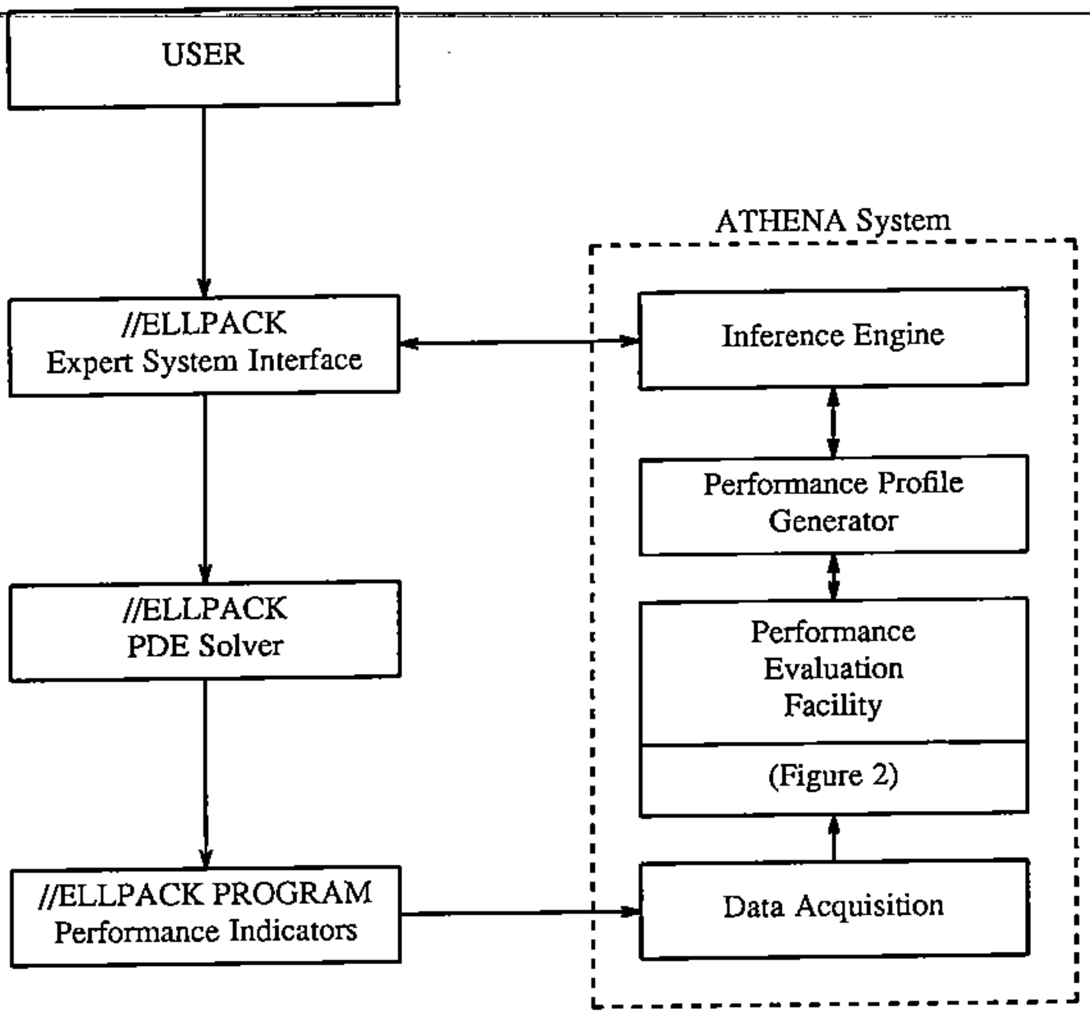

USER
ATHENA System
//ELLPACK Expert System Interface
Inference Engine
Performance Profile Generator
//ELLPACK PDE Solver
Performance Evaluation Facility
(Figure 2)
//ELLPACK PROGRAM Performance Indicators
Data Acquisition

Figure 1. The organization of ATHENA's software structure

Parallel Machines

If the selection process involves a parallel machine, then there is a difficult subproblem which the //ELLPACK system solves, namely selecting the number of processors, decomposing the PDE problem into parts and assigning these parts to the processors. This subproblem logically follows that of selecting a discretization and grid pair (as the accuracy requirement determines these). Once such a pair is identified, one can estimate the time and memory requirements roughly for various parallel machines and make a machine selection. Once this is done we have a method and a machine, but we must specify in detail the mapping of the PDE problem onto the parallel machine. This final step is carried by the //ELLPACK system.

3.1. Performance evaluation and knowledge acquisition facility

Figure 2 shows a block view of the performance database, its data acquisition facility and its data analyzer or performance estimate generator. The primary objective of this facility is to carry out experiments to enrich the performance database. This facility is also used to collect automatically data from the user's //ELLPACK program and dynamically update the database. A knowledge base manager keeps track of these operations. The //ELLPACK system is at the center of this facility as also seen in Figure 1. The performance evaluation generated from the database here is to guide the experimenter rather than to help the user select a method but the computations and software are very similar.

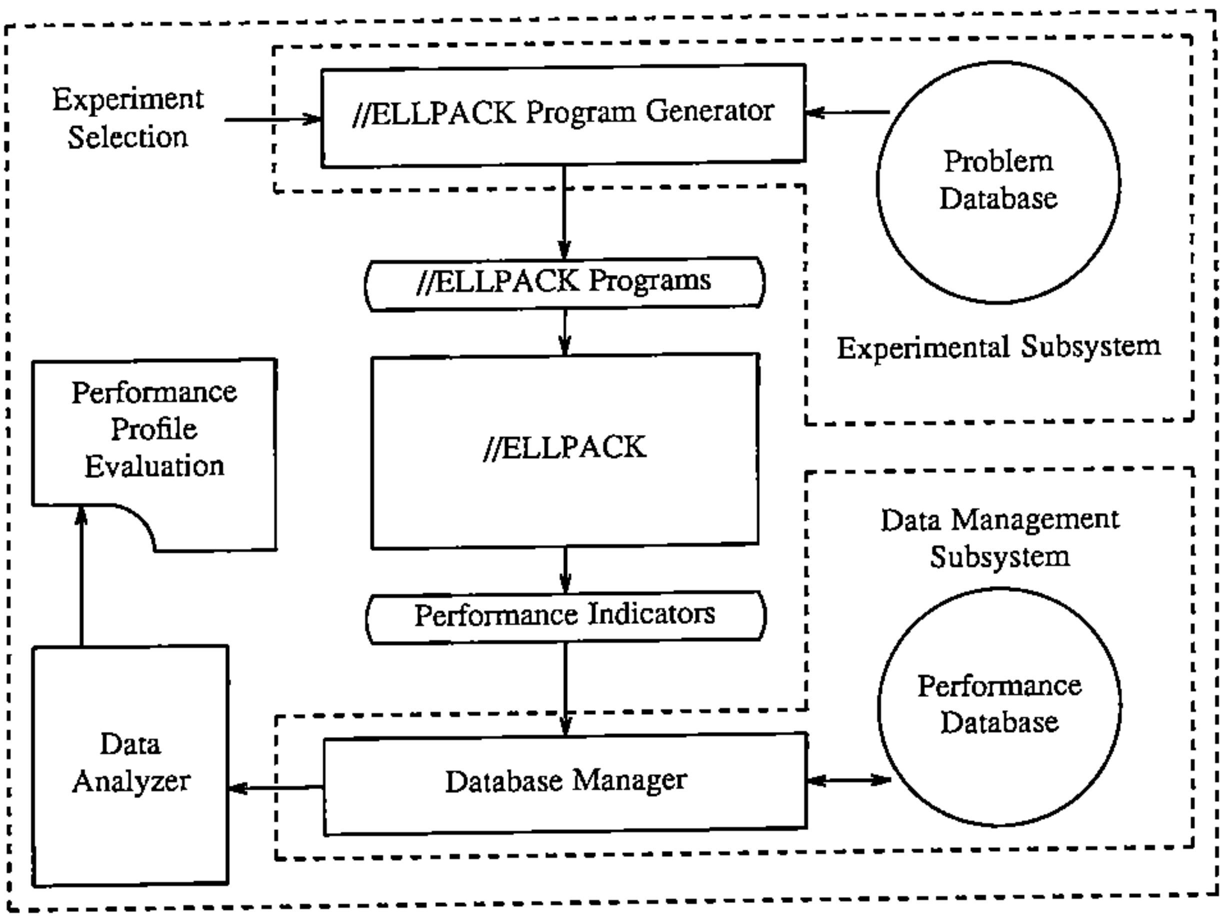


Figure 2. Structure of the performance evaluation and knowledge acquisition facility for ATHENA

The data analyzer is for locating data relevant to the PDE problem at hand. The PDE problem associations are made based on a comparison of the vector of problem properties and features of both the new PDE problem and those in the database. A metric is used to measure the "distances" and one of the key steps in the construction of this facility is to make this measure reliable. The properties and features have both logical and numerical values so the metric has components which are numerical weights associated with these values. The machine equivalences are also applied by this analyzer.

Note that this facility can also be used to tune the inference engine and data analyzer. An ''expert'' may observe the results of the inference engine (playing the role of a user) and then use this facility to review the intermediate steps in the generation of a selection. He may also experiment with changing various weights and values in these processes.

3.2. Knowledge generation

The primary representation of the knowledge are the performance profiles which relate time and accuracy to machines, discretization, solvers, and grids. These profiles depend on the PDE problems and so the larger the set of problems, the better chance of finding profiles relevant to a particular user's PDE. The ATHENA system attempts to synthesize performance profiles from data about problems that are ''close'' to the user's PDE. As the database grows, cluster analysis techniques are used to identify new problems (perhaps completely artificial ones) for which there is sufficient data to generate a reliable set of performance profiles. Once such a situation is identified, the new problem is ''created'' in the database and the synthesized profiles computed and made available for later use. In this way the use and training of //ELLPACK allows the ATHENA system to generate new knowledge about how to select methods and machines.

3.3. Performance estimation

The ATHENA system extracts information about the PDE problem from the user interface where the problem is formulated. It also asks for information, especially about features, from the user. Using this information, PDEs in the knowledge base are located which are close to the given one. The data analyzer then evaluates the relevance and closeness of the problems located and synthesizes performance profiles for this problem on ''standard'' sequential and parallel machines. These standard machines can be equivalenced to any machines actually available for solving the PDE. This process is illustrated in Figure 3.

3.4. ''New'' PDE problems

The ATHENA system must be prepared for PDE problems where little or no relevant information can be located in the knowledge-base. The problem might be one truly different from any seen before or it might be one where most of the features are unknown and not readily computed. The system, of course, asks for guidance from the user when features are missing, but the user may choose not to respond in a helpful way. In such a case, the system requests permission to make exploratory computations. If so permitted, it chooses a very robust, general discretization (say, collocation with Hermite cubics), a coarse mesh (say, 7 by 7), a robust solver (say, Gauss elimination), and a convenient machines (say, the user's workstation) and solves the PDE. The mesh is refined a little and the data collected is examined to see if systematic performance behavior is present. The results are also displayed to the user in an attempt to prod him into giving guidance. However, if no further guidance is given, the system continues along a predetermined path until either the problem is solved or the time response limit is reached.

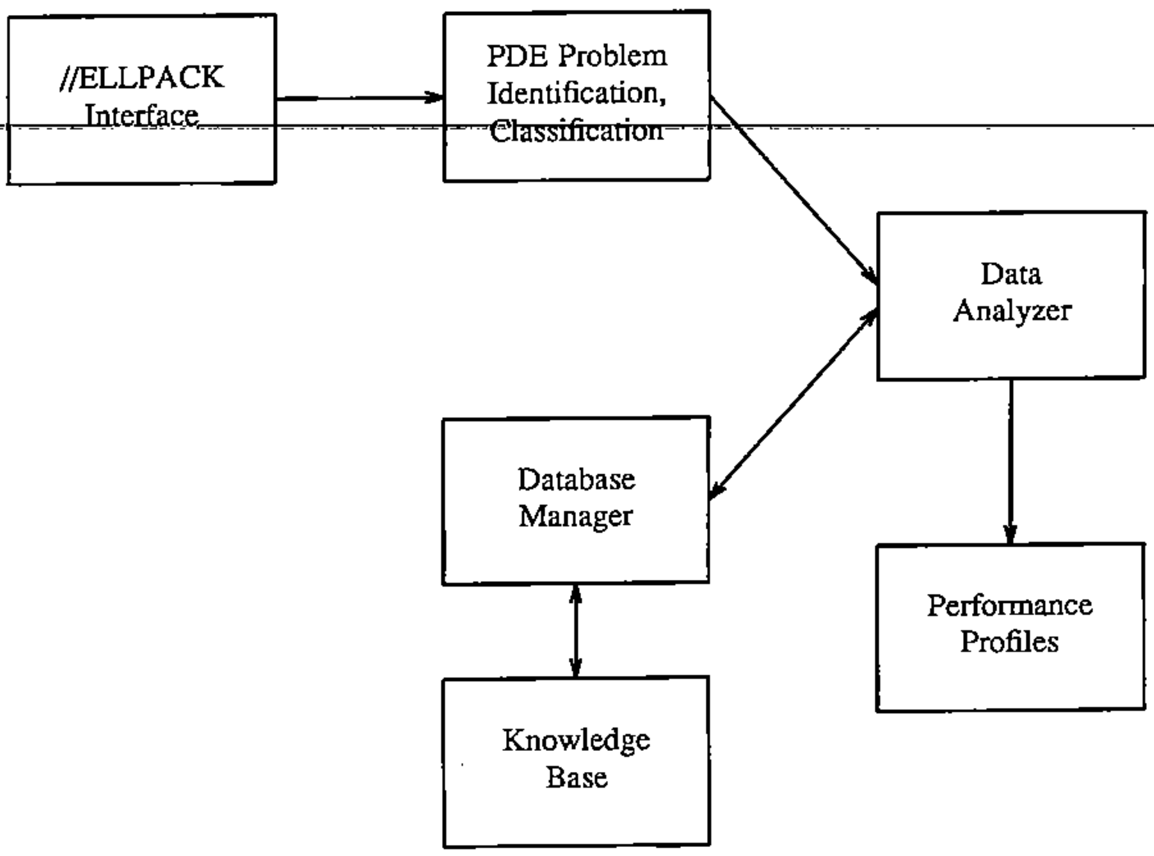


Figure 3. Schematic of ATHENA's use of the knowledge base to synthesize performance profiles upon which performance estimates are based

4. ATHENA's Inference Engine

The purpose of the inference engine is to make selections of method and machines using the performance profiles synthesized for the PDE problem at hand. It also uses rules (in the usual sense of rule based expert systems) that serve to focus the inferences and to resolve uncertainties in the selection. The inference starts with the PDE already classified and a certain number of performance profiles available. The steps in the inference are as follows:

1. Identify all applicable methods.
2. Eliminate methods which.
    a) are generally inferior to other applicable methods
    b) have no performance profiles
3. For each method (discretization) use the performance profiles of accuracy versus grid to estimate grid size required.

4. For each discretization method use the performance profiles to estimate the execution time for solvers (on the "standard" machines). Eliminate grossly inferior solvers.
5. Convert execution time estimates from standard machines to available machines.
6. Query available machines about estimated response for computations that are likely to meet the time requirements.
7. Select the method and machine which meets the requirements and, if possible, meets auxiliary objectives such as: Lowest cost, small impact on network, most confidence in estimates, etc.

These steps are shown in schematic form in Figure 4.

Note that parts of this process is similar to that in *Elliptic Expert* [Dyksen and Gritter, 1989]. The principal enhancements of ATHENA are, a) the heavy reliance on performance profiles, b) the synthesizing of new performance profiles, c) the inclusion of parallel machines in the selection, d) the enhancement of the knowledge base during use of //ELLPACK, and e) the experiment system for "training" ATHENA.

References

[Bois 79] Boisvert, R.F., E.N. Houstis and J.R. Rice, A system for performance evaluation of partial differential equation software", *IEEE Trans. Software Engr.*, **5** (1979), pp. 418–425.

[Dyks 89] Dyksen, W.R. and C.R. Gritter, Elliptic Expert: an expert system for elliptic partial differential equations, *Math. Comp. Simulation*, **31** (1989), pp. 333–342.

[Hous 88] Houstis, E.N., J.R. Rice, C.C. Christara and E.A. Vavalis, Performance of scientific software, *Mathematical Aspects of Scientific Software* (ed., J.R. Rice), IMA Volumes in Mathematics and its Applications **14**, Springer-Verlag, New York (1988), pp. 123–155.

[Hous 89a] Houstis, E.N., J.R. Rice and T.S. Papatheodorou, Parallel (//) ELLPACK: An expert system for the parallel processing of partial differential equations, *Math. Comp. Simulation* **31** (1989), pp. 497–508.

[Hous 89b] Houstis, E.N., M. Katzouraki, T.S. Papatheodorou and V. Sotiropoudou, Logic parallelism in an expert system for solving partial differential equations, *Intelligent Mathematical Software Systems*, (eds., E.N. Houstis, J.R. Rice and B. Vichnevetsky) Elsevier, Amsterdam (1990), to appear.

[Rice 85] Rice, J.R. and R.F. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York (1985).

[Hous 89b] E.N. Houstis, J.R. Rice, C.C. Christara and E.A. Vavalis, Performance of scientific software, *Mathematical Aspects of Scientific Software* (ed., J.R. Rice), IMA publication, Vol. 14, Springer-Verlag 88, pp. 23–155.

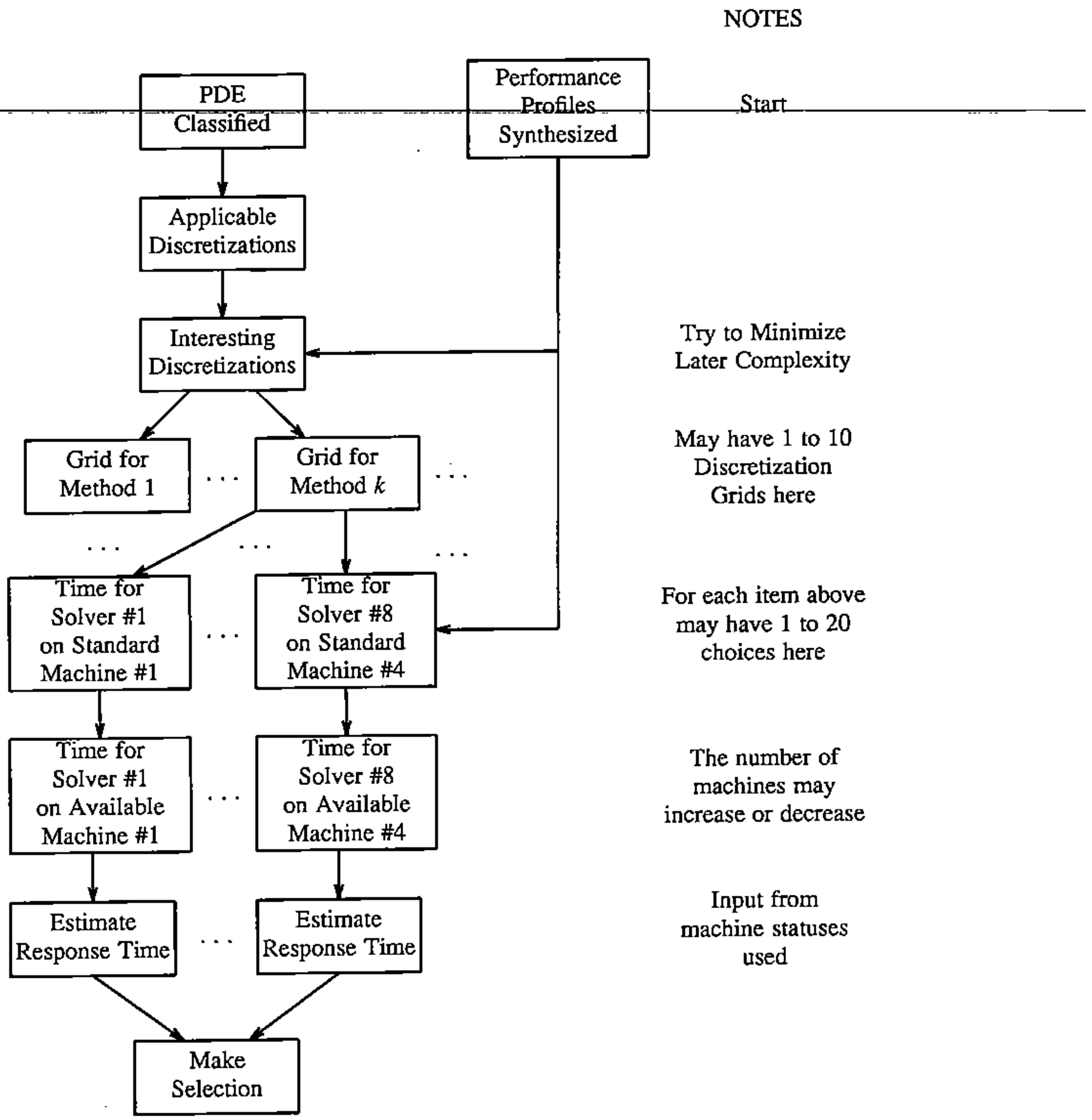

NOTES
PDE Classified
Performance Profiles Synthesized
Start
Applicable Discretizations
Interesting Discretizations
Try to Minimize Later Complexity
Grid for Method 1
Grid for Method k
May have 1 to 10 Discretization Grids here
Time for Solver #1 on Standard Machine #1
Time for Solver #8 on Standard Machine #4
For each item above may have 1 to 20 choices here
Time for Solver #1 on Available Machine #1
Time for Solver #8 on Available Machine #4
The number of machines may increase or decrease
Estimate Response Time
Estimate Response Time
Input from machine statuses used
Make Selection

Figure 4. Schematic diagram of the inference mechanism used to select the combination of method and machine to solve the PDE

SESSION 5

An Expert System as a Support to the Design of Airfoils

L. Ghielmi

Aermacchi S.p.A., Ufficio Progetto Aerodinamico
Via Sanvito Silvestro 80
I-21100 VARESE, ITALY

D. Quagliarella

C.I.R.A. S.p.A., Laboratorio di Intelligenza Artificiale
Via Maiorise
I-81043 CAPUA (CE), ITALY

We describe herein the state of development of an expert system which applies to the quality improvement of the aerodynamic design of airfoils for the wing of high performance airplanes.

A shape which satisfies the design requirements can effectively be obtained resorting to an existing optimization procedure that represents the translation into procedural code of only a part of the semantics involved by the design. With expert systems we want to increase the knowledge owned by the computer.

Our first goal is the elimination of the difficulties a designer finds when using this design procedure.

This expert system can be classified as dedicated for design (in fact it takes actions about it) but, because of the iterative use of the procedure, it must accomplish successfully the tasks of configurating the input file for each modification cycle and those of planning, monitoring and debugging the evolution of the geometry transformation.

The knowledge is acquired through a goal-driven approach, and is implemented resorting to rules of production and inference.

This methodology seems to be useful for the acquisition of other knowledges in the future. Several examples about the implemented knowledge are provided herein.

The Use of an Expert System in the Control of Structural Analysis Idealizations

M.S. Shephard, E.V. Korngold, R. Wentorf, A. Budhiraja and R.R. Collar

Program for Automated Modeling
Rensselaer Design Research Center
Rensselaer Polytechnic Institute
Troy, N.Y. 12180-3590

An engineering analysis employs idealizations to reduce a physical behavior of interest into sets of algebraic equations that can be solved on a computer. Each step of idealization used in this process introduces approximation, and associated approximation error [BABU 86], [BABU 86a], [SHEP 89], [SZAB 88]. The reliability of the modeling process depends on the ability to control the errors introduced by each approximation. To control idealizations, the approximation errors introduced by the idealizations need to be qualified. The methodologies available to control idealizations range from analytically-based adaptive analysis procedures, to codified knowledge based on experience and past practice. This paper will discuss the application of expert system techniques to control structural analysis idealizations within and engineering modeling system that employs all available levels of idealization control.

Over the past several years there has been substantial progress made in the development of idealization control techniques for mesh discretization errors. These efforts have lead to the development of adaptive analysis techniques. An adaptive finite element code employs a posteriori error estimation to determine the magnitude and distribution of the mesh discretization errors, and then selectively enriches the mesh until the desired level of accuracy in the selected norms is obtained. Adaptive analysis techniques to control the discretization errors due to the finite element mesh continue to mature and should begin to become generally available for use by the engineering community. The combination of adaptive analysis techniques with tools such as automatic mesh generators will allow for automated engineering analysis, the application of which will be robust from the aspect of controlling the errors associated with the finite element mesh. However, these tools and techniques will not ensure that the results obtained are robust with respect to the original performance request made, unless the other idealization errors associated with the modeling process are controlled. Therefore, an engineering modeling system must provide techniques to ensure the best level of control over the idealizations.

The most desirable approach to improve the robustness of analysis idealization processes is to develop analytically-based tools which accurately estimate each error contribution. Currently, there is no clear methodology to analytically control the errors for a large number of commonly used idealizations. In fact, a number of critical idealization processes are not likely to be addressed by such methods in the near future. This is particularly true if one considers the engineering design process which employs a series of ever improving idealized models for an artifact which is not completely defined until the design is finished.

The above situation indicates the need to develop engineering modeling systems that can support all possible levels of idealization control. Since the goal of the idealization control capabilities is to support the application of engineering modeling tools during the design process, idealization control must be an integral part of a design modeling system. A key component of such a system is the process navigator [KORN 89], [SHEP-89a] which tracks engineering idealizations and controls the application of the modeling procedures invoked during the design process.

The process navigator consists of three major components. The first is the request interpreter which is responsible for accepting a request to perform a modeling task, determining if the design has progressed to the point where the requested process can be applied in a meaningful manner, and determining if the tools necessary to apply that process are available. The second component, the strategist, is responsible for execution of the modeling task. This includes defining the most appropriate strategies to perform the requested task with the available modeling tools and the current state of the design. For example, given an analysis request, the analysis strategist extracts the model information required for a particular analysis, determines the best way to perform the analysis and idealization control procedures needed for that analysis process. The third component of the process navigator is the process monitor which is responsible for updating the model representation based on the results of a modeling process. It also maintains a log of what procedures were applied and why. The information in the log is used to help control the design process.

The paper will discuss the implementation of engineering modeling and control procedures using a LISP based shell structure. Primary emphasis will be on the implementation of a process navigator, and the use of an expert system to apply structural analysis idealization rules. The shell being used for this system is the frame-based environment KEE by IntelliCorp. In the prototype procedures to be presented KEE is used to support the overall user interface, the functional and attribute models, the request interpreter, the process monitor, the knowledge bases, the inference engine, and the knowledge-based portion of the analysis strategist. The geometric model is stored within the structure of a geometric modeling system and the analytic applications reside in their own software environments.

The paper will describe progress made on the implementation of idealization control procedures for the analysis of airframe structures, and the analysis of two-dimensional plane elasticity problems within the design process. In the case of the airframe analysis, the majority of idealization control will be done through rule sets. In the case of the two-dimensional elasticity analyses, the idealization control procedures will range from adaptive analysis to rule sets.

The process navigator begins the process of controlling analysis requests through the use of an analysis goal graph. An analysis goal graph indicates the various analysis types possible, the inter-relations of those analyses, and the needed status of the design to apply to each analysis. The analysis goal graph for the structural analysis of airframes includes various levels of static and dynamic analyses. The analysis levels range from simplified frame analysis, through structural analyses with all members represented using appropriate idealizations, to detailed fatigue analyses of individual components. The analysis goal graph for the two-dimensional elasticity problems is fairly simple using the same basic analysis techniques for all analysis goals.

The process monitor first examines the functional model to determine if the current state of a given design is sufficient to support the analysis requested. In the case of the airframes, this consists of seeing if the structural components needed for the requested analysis exist and have been defined to the point where the appropriate idealization procedures can be applied in a meaningful manner. For the two-dimensional elasticity applications, the process monitor is concerned with the level of geometric detail defined (cutouts, fillets) and the availability of the analysis attributes.

The analysis strategist combines knowledge of the state of the design, the rules in the rule base, and the results of the analysis process to give the user the most reliable solution results possible. In the case of the airframes, the expert system will invoke various rule sets to perform the dimensional reductions of individual members commonly used in this class of analysis [GABE 81]. The dimensionally reduced members are then discretized using another group of rule sets.

In the two-dimensional elasticity case, a more limited group of rule sets is applied primarily to deal with the geometric simplifications of using sharp fillets and ignoring small cutouts. One rule set uses a priori rules to indicate the acceptance of a geometric simplification. Another rule set uses analysis results in an a posterior manner by applying analytic stress concentration formulas for idealized situations to determine the appropriateness of a geometric simplification used in the analysis [SHEP 86]. The remaining idealization error arises from the mesh discretization errors. These errors are controlled using an automated, adaptive finite element modeling system [BAEH 89].

References

[BABU 86] Babuska, I., Zienkiewicz, O.C., Gago, J. and de A. Oliveria, E.R., *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley and Sons, Chichester, U.K., (1986).

[BABU 86a] Babuska, I., Uncertainities in engineering design: Mathematical theory and numerical experience, *The Optimum Shape: Automated Structural Design*, Bennett, J.E. and Botkin, M.E. (eds.), Plenum Press, N.Y., (1986), pp. 171–197.

[BAEH 89] Baehmann, P.L., Automated finite element modeling and simulation, *Ph.D. Theses, Dept. of Mechanical Engineering*, RDRC TR-89014, RPI, Troy, N.Y., (1989).

[KORN 89] Korngold, E.V., Shephard, M.S., Wentorf, R. and Spooner, D.L., Architectures of a design system for engineering idealizations, *Advances in Design Automation, 1989, Vol. 1, Computer-Aided and Computational Design*, ASME, N.Y., (1989), pp. 259–265.

[GABE 81] Gabel, R., Ricks, R.G. and Magiso, H., Planning, creating and documenting a NASTRAN finite element vibration model of a modern helicopter, *NASA Contractor's Report 165722*, (1981).

[SHEP 86] Shephard, M.S. and Yerry, M.A., Toward automatic finite element modeling for the unification of engineering analysis and design, *Finite Elements in Analysis and Design*, Vol. 2, (1986), pp. 143–160.

[SHEP 89] Shephard, M.S., Korngold, E.V. and Wentorf, R., Design modeling system for engineering idealizations, *Geometric Modeling for Product Engineering, IFIP Proceedings*, North Holland, to appear.

[SHEP 89a] Shephard, M.S., Idealizations in engineering modeling and design, *NSF Engineering Design Research Conference*, Univ. of Mass., (1989), pp. 521–536.

[SZAB 88] Szabo, B.A., Geometric idealizations in finite element computations, *Communications in Applied Numerical Methods*, Vol. 4, No. 3 (May-June 1988), pp. 393–400.

OOPSP: An Object-Oriented Particle Simulation Programming System

Xinming Lin and Walter J. Karplus

Computer Science Department
University of California
Los Angeles, CA 90024

OOPSP is a programming system facilitating particle simulation in an object-oriented programming style. Particle simulation is a generic term representing a class of simulation methods where the stimulated system is described by interacting "particles", which can exist physically or conceptually. The importance of the approach in OOPSP includes

1. The OOPSP approach is natural in particle simulation. Simulations are constructed from carefully designed building blocks like *Models* and *Particles,* instead of from low-level descriptions like numbers and arrays of numbers. It allows the users to concentrate on the high-level physical concepts instead of on the detailed low-level implementation.
2. The OOPSP approach is general in particle simulation. It is applicable to different kinds of particle simulation applications.
3. The OOPSP approach is effort-saving. Frequently-used models, schemes, and algorithms are made as modules that can be used again and again without the necessity of re-implementing them. Particle simulation programs in OOPSP are much shorter and much easier to be composed than programs written in FORTRAN.
4. The OOPSP approach can be efficient. The OOPSP modules can be constructed carefully. Algorithms that implement the modules can be selected carefully. Optimization in the implementation should be attained as much as possible and it has to be done only once. The implementation of OOPSP on a special computer architecture can take the advantage of it, and in that case, the users can take the advantage of the special architecture without knowing it.
5. OOPSP is extendible. Users can replace the pre-defined modules with their own definition. Since OOPSP is hierarchical, new modules can also be *derived* from the existing ones with minimum efforts.

OOPSP is implemented in C++, an object-oriented extension of the C programming language. However, the user does not need to know C++ in order to use it. OOPSP has a C-like syntax, plus some pre-defined data types and functions that support particle simulation.

OOPSP is organized around two main data types or *classes* – *Model* and *Particle.* The Model class is a class of computational methods frequently-used in particle simulation. The Particle class contains various templates to represent the simulated particles.

An instance of the Model class or the Particle class is called an *object* of the class. A combination of the Model objects and the Particle objects through OOPSP functions constitutes a particle simulation. OOPSP functions initialize, control, and visualize a simulation, as well as perform miscellaneous operations.

Examples of the application of OOPSP in particle simulation will be presented. Specifically, the problem of metal dislocation under radiation will be simulated in OOPSP. The effectiveness of the Object-Oriented Particle Simulation Programming system will be demonstrated.

Continuation Expert System - CONVEX

P. Rosendorf, J. Orsag, I. Schreiber and M. Marek

Department of Chemical Engineering
Prague Institute of Chemical Technology
Suhbatarova 5
166 29 Prague 6, CZECHOSLOVAKIA

Numerical methods for analysis of nonlinear dynamical systems including methods based on continuation techniques for obtaining curves of stationary or periodic solutions, bifurcation points and limit points in dependence on parameters have been developed and discussed in several available textbooks [1-4,6,7], and in the proceedings of conferences, c.f., e.g., [5]. Original software for analysis of the dependence of stationary solutions on a parameter [1,4] and of periodic solutions on a parameter [4,7], were also published. Productive application of such a software requires relatively deep knowledge both of specialized numerical methods and of the theory of nonlinear dynamical systems. Also organization of computations and evaluation of computed results can be often quite complicated. Here we describe our attempt to test the idea whether a unification of available numerical algorithms with the means and approaches of logical programming and knowledge engineering could help to increase productivity of analysis of nonlinear dynamical systems.

An expert open modular system – CONVEX (**CONtinuation EXpert system**) has been developed [8-10].

The CONVEX can automatically prepare the actual continuation program in a runable form, control its execution and incorporate results into the database. It works with a library of numerical subroutines; however, only those necessary for the given task are used in the course of generation of the actual program. Minimal hardware requirement is IBM PC/XT compatible computer with a hard disc and 640 KB of RAM. Numerical co-processor and the output graphical device are useful for an effective work. Numerical computations can be executed on a mainframe computer. The CONVEX in principle consists of the knowledge base for analysis, computational knowledge bases (CKB) and data bases of studied problems. In particular, the CKB consists first of libraries necessary to build proper continuation programs (e.g., the library of continuation routines, library of graphics support routines, etc.), and of a set of system routines (e.g., compiler, linker, editor, etc.) and second, of the knowledge base containing information on the use of the above routines and libraries.

The main program for solution of the given problem is generated on the basis of model equations and the chosen type of the continuation. Also other options (e.g., inclusion of graphics, etc.) affect generation of the main program. At the same time, other subroutines (default data initialization, evaluation of right hand sides, evaluation of Jacobi matrices, etc.) are generated. Methods of symbolic differentiation and symbolic simplification of algebraic expressions are used, for example to generate the module for the evaluation of Jacobi matrix. In addition to the above FORTRAN 77 program, also the description of the structure of the input and output data sets and necessary information for evoking the compiler and linker are generated. The description of the input data structure is used by an interactive data editor. The description of the output data set is included into the databases of the studied problem only after successful computation and analysis of the computed data.

There is a possibility to generate full source code of the particular continuation program based on the source code libraries. This feature enables creation of specialized standalone continuation programs. The continuation library in the present form includes software for the following tasks:

(a) *Stationary solutions of ordinary differential equations (ODE).*
  - continuation of nonsingular stationary solutions,
  - continuation of limit points,
  - continuation of Hopf bifurcation points.

(b) *Periodic solutions of difference equations.*
  - continuation of nonsingular periodic solutions,
  - continuation of limit points,
  - continuation of period doubling bifurcation points,
  - continuation of Hopf points.

(c) *Periodic solutions of periodically perturbed ODE's.*
  - continuation of nonsingular periodic solutions,
  - continuation of limit points,
  - continuation of period doubling bifurcation points,
  - continuation of Hopf points.

(d) Periodic solutions of autonomous ODE's.
  - continuation of nonsingular periodic solutions,
  - continuation of limit points,
  - continuation of period doubling bifurcation points,
  - continuation of Hopf points.

(e) *Analysis of the computed continuation curves.*
  - computation of the slope of the continuation curve at the branching point of a stationary solution,
  - computation of the slope of the continuation curve at the branching point of a periodic solution,
  - computation of the slope of the continuation curve at the branching point at the Hopf bifurcation point.

The CKB has been designed so that it permits easy modification for other applications.

References

[1] KUBICEK, M. and MAREK, M., Computational methods in Bifurcation theory and dissipative structures, *Springer-Verlag*, Berlin, (1983).

[2] HOLODNIOK, M., KLIC, A., KUBICEK, M. and MAREK, M. Methods of analysis of nonlinear dynamical models, *Academic*, Prague, (in czech), (1986).

[3] RHEINBOLDT, W.C., Numerical analysis of parametrized nonlinear equations, *J. Wiley and Sons*, New York, (1986).

[4] DOEDEL, E.J. and KERNEVEZ, J.P., AUTO: Software for continuation and bifurcation problems in ordinary differential equations, *Applied Mathematics Technical Report*, California Institute of Technology, (1986).

[5] KUPPER, T., SEYDEL, R. and TROGER, H. (eds.), BIFURCATION: Analysis, algorithms, applications, Birkhauser, Verlag, Basel, (1987).

[6] SEYDEL, R., From equilibrium to chaos, *Elsevier Science Publ.*, New York, (1986).

[7] MAREK, M. and SCHREIBER, I., Chaotic behavior of deterministic dissipative systems, *Cambridge Univ. Press*, Cambridge, *Academia Press*, Prague, (1989).

[8] ORSAG, J., ROSENDORF, P., SCHREIBER, I. and MAREK, M., Development of an expert system for use of continuation techniques in nonlinear dynamics studies, *2nd Workshop on Path Following Methods and Bifurcation Theory*, Leeds, Great Britain, (Jan. 1988).

[9] ORSAG, J., ROSENDORF, P., SCHREIBER, I. and MAREK, M., Expert system for use of continuation techniques in nonlinear dynamics studies, *European Symposium Computer Application in the Chemical Industry*, Erlangen, Germany, (Apr. 1989).

[10] ROSENDORF, P. ORSAG, J., SCHREIBER, I. and MAREK, M., Interactive system for studies in nonlinear dynamics, *NATO Workshop Continuation and Bifurcation: Numerical Technique and Applications*, Leuven, Belgium, (Sept. 1989).

SESSION 6

Synthesis of Mathematical Modeling Programs

Elaine Kant, François Daube, William MacGregor and Joseph Wald

Schlumberger Laboratory for Computer Science
P.O. Box 200015
Austin, TX 78720-0015

We are using Mathematica to build a mathematical program synthesis system, called **Sinapse.** The purpose of the system is to help engineers minimize the time required to design, implement, and evaluate mathematical models. Our initial focus is semi-automatically transforming models based on partial differential equations into efficient numerical codes in Fortran and C for multiple target architectures (including massively parallel). To date, the system has synthesized a modest number of programs, most of which are forward modeling, wave propagation programs that use finite difference techniques.

The system can also synthesize a program that solves a set of equations via Fourier transforms. Domain knowledge in these areas is represented primarily as Mathematica functions with some rules; a simple object hierarchy is used to represent classes of algorithms and domains. Program specifications in the range of 10 to 50 lines generate target code of 50 to 1600 lines in time ranging from 10 seconds to several minutes.

Sinapse helps the user define a set of governing equations and select an appropriate solution algorithm; it then produces an efficient implementation in Fortran or C. The system maintains specifications and design histories to reduce the cost of redesign for altered requirements. The user interface consists of simple menu choices and single-line text entry. Because **Sinapse** has the symbolic manipulation facilities of Mathematica available, an engineer can first work on symbolic formulations of problems and then have the system generate a program embodying a numerical solution.

In a sample session for a wave propagation problem, **Sinapse** helps the user express governing stress-strain equations in Mathematica, then analyzes and discretizes the equations to produce an explicit time-stepping solution code. This includes transformation of partial differential equations into finite difference statements, and the selection of finite difference operators with staggering of the grids. The synthesis methods handle arbitrary dimensionality. Boundary conditions represent a significant part of a modeler's effort. **Sinapse** helps by including domain-specific knowledge for expressing absorbing, reflecting, and free-surface boundary conditions for wave propagation examples.

Such a general purpose modeling tool would be useful to Schlumberger because physical phenomena such as seismic wave propagation, fluid flow, and electromagnetic induction are central to many Schlumberger businesses. Systems for sensing these phenomena, mathematical models of their behavior, and related computer program are frequently revised. Because many models are three dimensional and use large data sets, program execution speed is critical. Unfortunately, customization for performance usually complicates software, obscuring fundamental algorithms, introducing errors, and reducing portability and modifiability. **Sinapse** addresses this problem by enabling modification at the specification level (with domain-specific constructs) and semi-automated implementation and revision. A designer can therefore experiment with alternative implementations, record comparisons and trade-offs for future use, and exploit different types of design knowledge previously recorded by designers with different backgrounds.

Effective Knowledge Representation Schemes For Automatic Numerical Program Generation

(Extended Abstract)

Mark F. Russo

David Sarnoff
Research Center
Princeton, N.J. 08543-5300

A. Daniel Kowalski

CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, N.J. 08855-1390

Richard L. Peskin

Department of Mechanical and Aeroscope Engineering and
CAIP Parallel Computing Laboratory
Rutgers University
Piscataway, N.J. 08855-1390

A critical component of any knowledge-based system designed to solve practical, non-trivial problems is the methodology by which it represents specific domain knowledge. The scheme should be simple enough to avoid adding complexity to already complicated domain knowledge, yet powerful enough to express the intricate relationships which characterize problems of practical concern. Extensive detailed analysis and prototyping of knowledge-based systems for the automatic generation of numerical programs has resulted in a knowledge representation scheme that is both highly expressive and sufficiently flexible to depict the complex relationships in these types of problems [Kowalski 89, Russo 87]. The knowledge representation scheme will be analyzed for the components of an automatic numerical program generation system for solving sample non-trivial mathematical problems. The advantages of applying the scheme to each component will be discussed. In addition, an example will be presented of the application of this technique to the problem of solving the primitive Navier-Stokes equations over a forward facing step using the artificial compressibility formulation and the Marker-And-Cell numerical method. The problem involves several coupled nonlinear partial differential equations, a mixture of boundary conditions, and multiple overlaid rectinlinear grids.

The basis of the representation scheme is an object-oriented, logic programming environment [Kowalski 89, Kowalski 87]. The combination of these two programming paradigms provides a general environment that is highly suited to the problem of numerical program generation. Management of the large quantity of diverse, structured, domain-specific information in this problem requires a sophisticated and efficient organizational strategy as well as a means of reasoning about domain knowledge. The object-oriented programming paradigm provides basic hierarchical organization through its encapsulation, inheritance and message passing characteristics. While these characteristics are essential to the knowledge representation scheme, the object-oriented programming paradigm alone is not sufficient to handle all required representation and reasoning tasks; it must be embedded in an appropriate programming language.

Logic programming provides a powerful, structured, inferential capability which can be applied to solving problems involving heterogeneous information. In addition, the declarative aspect of logic programming provides the ability to express problem details in a natural way within the object-oriented framework. The system we describe is built entirely in Prolog, a sequential form of logic programming. Prolog provides efficient pattern matching and backtracking features which are essential for reasoning about equation solving and numerical methods. In addition, Prolog provides meta-logical programming capabilities which permit the manipulation, alteration, and execution of programs as data. This is especially important to our problem domain since a significant portion of the data represented by the knowledge base are numerical procedures, represented as logic programs, which are combined into complete numerical algorithms.

The flexibility of the development environment has been further extended by implementing a specific form of typed inheritance. Objects in the object-oriented hierarchy can be related to other objects in ways other than the standard behavioral *subclass* or *instance* relationships. The mechanism used to implement these additional relationships between objects is identical to that used to represent standard class/instance relationships except that messages can be directed to use a specific hierarchy. The result is a context-sensitive messaging capability.

The knowledge-based system initially accepts and stores information about the mathematical problem in its natural form via a graphical interface and stores it as instance data in the object-oriented hierarchy. The resulting object-oriented knowledge base describes equations and the domains on which they apply. Typically there is a single domain (Ω) on which a set of partial differential equations is to be solved. An additional set of boundary, and possibly initial conditions, also in the form of equations, are applied respectively to the boundary of the domain ($\partial\Omega$) and to the entire domain (Ω) at some fixed point.

Object-oriented methods symbolically analyze the problem data for particular characteristics and create object instances in one of the appropriate behavioral hierarchies (see Figure 1 and Figure 2). The typed inheritance mechanisms is used to relate each component of the problem back to an instance of a special class of objects called *problem Specification* through the special type *componentOf*. This explicitly relates all components of the problem through the inheritance hierarchy. Thus, one can send a typed message to, say, an instance of the class *nonLinScalPDE*, with the request that the associated method operate on the equation in the context of its function as a component of the mathematical problem hierarchy, rather than as a member of the equation class hierarchy. This provides a great deal of expressive power for representing and reasoning about complex relationships within mathematical problems.

Problem components, like the equations and domains that comprise the mathematical problem, are further decomposed into their subcomponents. This decomposition of single, complex problem component objects permits a "divide-and-conquer" strategy. The equation or domain instance objects create "instance trees" of themselves. Thus, for example, domain instances are decomposed into a special typed hierarchy of instances of subclasses of the class Domain that have equal or less complexity (see Figure 3). This hierarchy is constructed using the *subDomainOf* inheritance type. Similarly, equations are decomposed into instance trees of themselves with the special inheritance type *termOf*. Each term in the hierarchy is also a separate instance of an appropriate equation class. This approach efficiently implements the method by which a complex partial differential equation is automatically discretized.

The representation and symbolic analysis of practical mathematical problems with subsequent automatic construction of numerical solution algorithms requires a system which can not only maintain complex classifiable information, but can also reason about information in the system and represent and manipulate algorithms as data. It is shown that the hybrid object-oriented, logic programming system with typed inheritance and inherent metalogical capabilities, while still a general purpose tool, is especially well suited to this type of complex analysis.

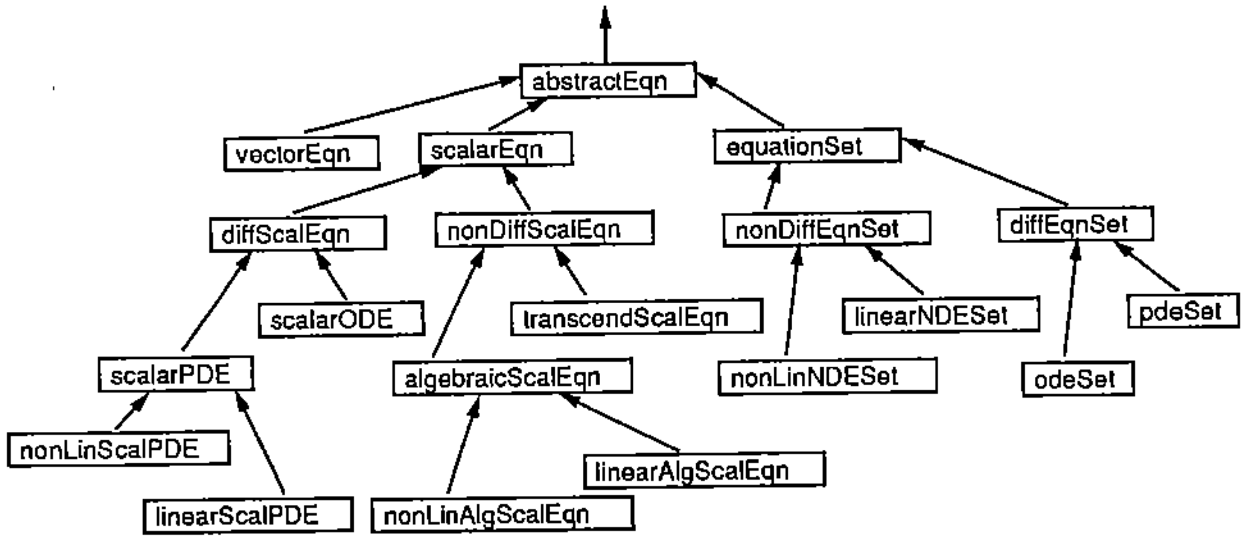


Figure 1. Hierarchy of equation classes

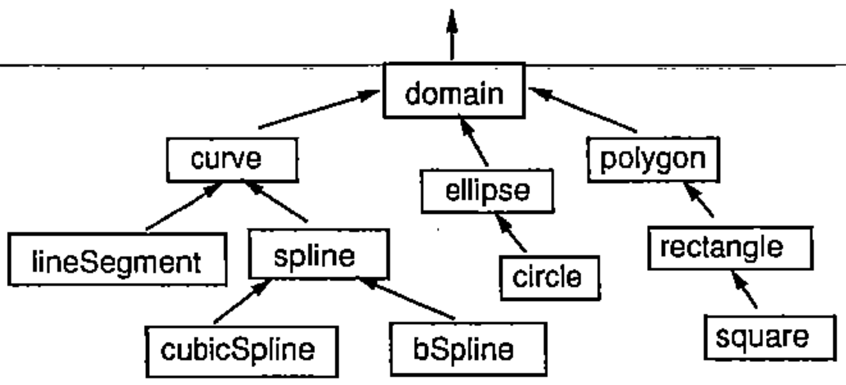


Figure 2. Hierarchy of solution domain classes

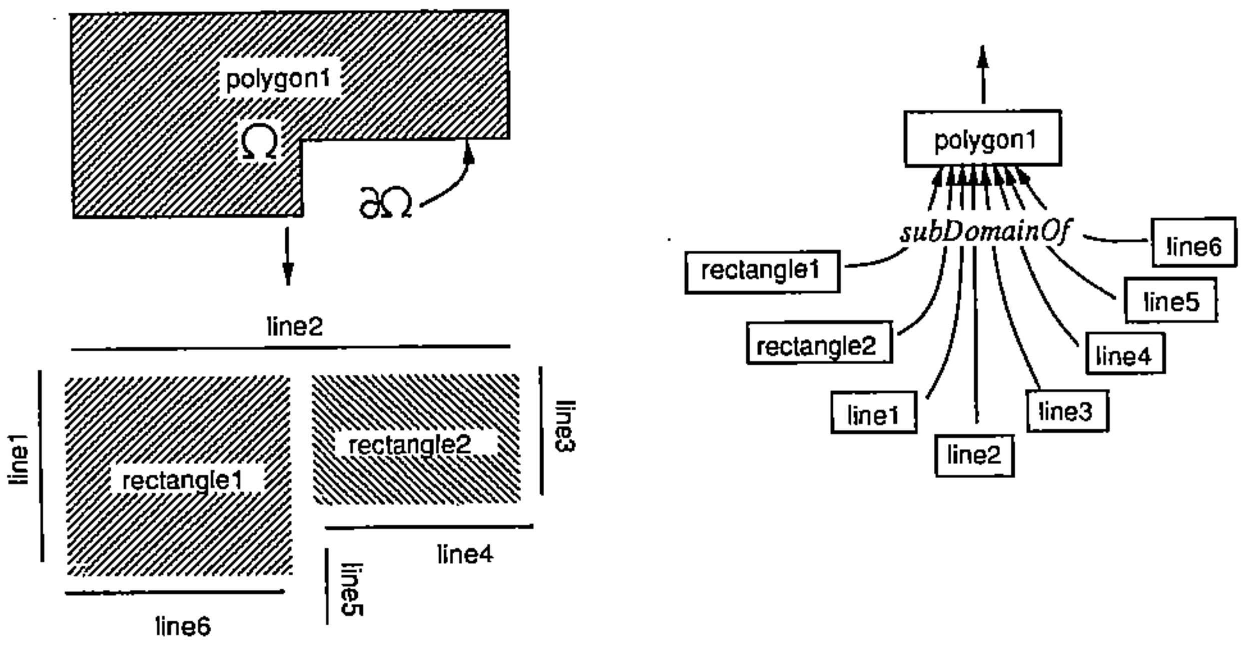


Figure 3. A solution domain decomposed into subdomain components

References

[KOWALSKI 89] Anatomy of AGNES: An automatic generator of numerical equation solutions, A.D. Kowalski, M.F. Russo and R.L. Peskin, *Expert Systems for Numerical Computing*, a special issue of the journal, *Mathematics and Computation in Simulation*, Vol. 31, Nos. 4 and 5, (Oct. 1989).

[RUSSO 87] A prolog based expert system for partial differential equation modeling, M.F. Russo, R.L. Peskin and A.D. Kowalski, in: *Simulation*, a publication of *The Society for Computer Simulation*, San Diego, California, Vol. 49, No. 4, (Oct. 1987).

[KOWALSKI 87] An object-oriented prolog representation of quasilinear partial differential equations, Kowalski, A.D., *Proceedings of the International Symposium on AI Expert Systems and Languages in Modeling and Simulation*, Barcelona, Spain, June 2-4 IMACS, (1987).

Using MACSYMA to Write Finite-Volume Solvers

Stanley Steinberg

Department of Mathematics and Statistics
University of New Mexico
Albuquerque, N.M. 87131

Patrick Roache

Ecodynamics Research Associates, Inc.
P.O. Box 8172
Albuquerque, N.M. 87198

MACSYMA is used to automatically write finite-volume code for solving general symmetric elliptic partial differential equations in general coordinates. Most of the MACSYMA code is written for n-dimensional problems, and then used to write code in one, two, or three dimensions. The three-dimensional code is about 1400 lines long.

It is assumed that a physical problem is presented in a complex geometry, and then transformed (using a general, non-orthogonal transformation) to a unit box in logical space (where the computations are done). The transformation is typically done using numerical transformations (MACSYMA is used to write this code).

In this abstract, finite-volume means that control volumes in logical space are used to derive a finite-difference scheme for approximating the problem. Much of the programming complexity is caused by the fact that different quantities are computed at different points in the grid. The coordinates in physical space are computed at cell corners, the solution of the PDE is computed (as a cell-averaged quantity) at the cell centers, and the fluxes are computed at cell face centers. Thus, MACSYMA must create loops that run over cell corners, cell edge centers, cell face centers and cell centers along with the relevant algebraic formulas.

Both the two-dimensional and three-dimensional codes have been thoroughly tested and will soon be incorporated into production code at Ecodynamics. Amazingly enough, this process produces essentially error-free code. Only one error has been detected in each of the codes.

Formal Specification of Engineering Analysis Programs
(Extended Abstract)

John W. Baugh, Jr.

Department of Civil Engineering
North Carolina State University
Box 7908
Raleigh, N.C. 27695-7908

A precise statement of observable behavior is needed to make engineering and scientific software easier to use. Without such a statement, users are forced to inspect code in an attempt to abstract its relevant properties. The benefits of providing this documentation, or *specification*, not only impact the end-users of programs, but also the researchers who use numerical libraries, and the programmers who develop large-scale engineering software. Unfortunately, specifications are rarely produced during the development of an engineering software system - exceptions may be certain key numerical algorithms, such as equation solvers and numerical integration schemes, which are analyzed for numerical properties such as convergence and roundoff errors. These particular algorithms, however, represent only a small percentage of the program text of large-scale engineering analysis programs. This paper discusses the role of formal specifications in engineering software development, including their effect on program structure, reliability and efficiency.

Consider a typical analysis program or numerical library consisting of tens-of-thousands of lines of FORTRAN statements. Casual inspection of any particular code fragment is enough to reveal that imperative programs do not communicate intent very well - this is the nature of imperative languages, which over-specify details of the computation. In order to be more readily understood, these program fragments must be documented with a higher-level description of their behavior. Increasing numbers of computing researchers advocate formal methods of specification, not only for documentation and communication purposes, but also for early detection of bugs and clarification of the problem statement. Once set in a formal notation, however, such specifications may also be used for verifying program correctness, representing knowledge, and prototyping complex software systems, since some formal notations also have an operational interpretation.

A variety of mathematical notations, such as logic, algebra and functions, may be used to formalize program specifications, and each of these is precise, unambiguous, and amenable to some degree of machine manipulation. In addition to these *definitional* notations are *operational* ones, such as abstract models, in which specifications are constructed by describing a possible implementation. This work focuses on definitional notations, which are generally more concise and less prone to implementation bias.

For example, many-sorted algebras are particularly suitable notations for describing abstract data types in programs, and this approach to specification has received considerable attention in recent years (at least within research communities). However, the role of algebraic specifications in numerical computing has not been addressed, and neither has the effect of data decomposition on program structure, which is used to identify appropriate data types. This work demonstrates that the benefits of data decomposition (e.g., modularity and reusability) are also obtained for numerical applications. In terms of practical implications, this means that numerical programs and libraries become collections of abstract data types, such as matrices, labeled graphs, and coordinate systems, where the representation(s) of each type are hidden. In addition, because each type's specification is cast in a formal framework with an operational semantics, syntactic proofs may be performed (by human or machine) to demonstrate properties such as completeness and consistency. When combined with structural induction as a method of proof, much can be said about a data type at the specification level, i.e., before implementation begins. Because of their underlying formal semantics, data type specifications provide the criteria to establish when an implementation is "correct".

In addition to correctness concerns, formal specifications may also be used to represent knowledge about the algorithms and data types used in AI programs. For example, an "intelligent" implementation of a computationally intense numerical program might accommodate multiple representations and algorithms, with selection based on the problem (input) at hand. This feature could then be used, say, to take advantage of matrix structure in solving linear systems. Note that a precise statement of behavior is required to know when several different representations or algorithms can be used to achieve the same effect.

Alternatively, algebraic specifications can be used for rapid prototyping when restricted to a "functional", and therefore executable, subset. This approach has in fact been used in the development of a linear finite element program. Not only does it improve productivity, but it also gives developers the opportunity to consider the merit of a variety of module interfaces, i.e., the specification. As compilation techniques improve for this class of languages, such "prototypes" may actually become viable implementations. On top of this optimism are the prospects for automated parallelism via dataflow and reduction techniques, which are applicable to functional languages.

Of course, certain consideration must be borne in applying specification technology to numerical programs. Some of these considerations include:

- *Matrix Operations*. The imperative nature of some matrix operations results in what is referred to as the "incremental update" problem in declarative languages. Although the problem is primarily one of efficiency, there is a certain lack of clarity in specifying matrix algorithms with "functional update" operations and recursion. Although various alternatives have been proposed, this paper shows how monolithic array constructors and array comprehensions can be used to improve both clarity and efficiency in many situations.
- *Roundoff Characteristics*. Unlike most non-numerical applications, engineering analysis programs must deal specifically with the effects of roundoff errors, and specifications must account for them. For example, it would probably be wrong to specify the behavior of a routine that performs dot products as $x^T y$. Because of less than infinite precision arithmetic, we might

write instead $fl(x^T y)$, *where* $|fl(x^T y) - x^T y| \le \varepsilon(u,n)$, in which $fl()$ represents the computed quantity and ε is a function of the unit roundoff error and the size of the vectors x and y. Of course, such specifications are not executable. We may opt instead for a more operational specification based on the definition of a dot product.

Among the other considerations in writing specifications for numerical programs are: the specification of truncation error properties, where users typically demonstrate convergence by experimenting with various discretizations; dealing with termination properties (i.e., proving convergence) if and when verification is attempted; handling partial functions, such as equation solvers that are undefined on singular matrices (perhaps by using strict functions and error values, imperative exception handling techniques, etc.); and incorporating facts about the application domain within the specification, e.g., material properties and modeling assumptions.

Although formal specification methodology requires some "adjusting" for numerical applications, the author's preliminary experiences have been both positive and informative. In some situations, the simple act of formalizing a specification or trying to provide enough details to satisfy a proof has led to either increased understanding or a reworking of the interface or both. The intent of this paper and presentation is to demonstrate through discussion and specific examples some of the benefits derived from applying formal specifications to engineering analysis programs.

ALPAL's Matrix Editor For Symbolic Jacobians

Jeffrey F. Painter

L-316, Lawrence Livermore Laboratory
Livermore, CA 94550

The Matrix Editor is a tool for specifying the numerical treatment of Jacobian matrices by an automatically generated code. This Matrix Editor is invoked as a part of ALPAL, a "physics compiler" which generates modules of very large simulation codes. With the Matrix Editor one can describe the sparsity structure of a symbolically defined Jacobian matrix, how to treat it in a numerical code, and how to numerically solve corresponding linear equations.

For writing code to solve complicated physical problems with implicit time integration schemes, one of the most dramatic advantages of using ALPAL over direct Fortran coding is that Alpal can automatically compute expressions for Jacobian matrix elements. For large complex problems, it is nearly impossible to find accurate Jacobians by traditional means; the usual practice is to make rough approximations which sacrifice speed and accuracy. Codes using such implicit methods for large problems often devote most of their resources - time and memory - to solving linear equations. Thus it is important for ALPAL to provide tools to make it easy for its users to solve linear equations with good methods, and experiment if necessary to choose them.

ALPAL's Matrix Editor provides the basic tools needed to generate Fortran code that solves linear equations efficiently. It is mainly an easy-to-use program for specifying how one wants a Jacobian matrix to be stored, and how to call an external linear solver for it. Like the rest of ALPAL, the Matrix Editor follows the philosophy that an intelligent user should make all decisions requiring significant intelligent judgement, and ALPAL should do the more routine, mechanical parts of coding a physics model. Thus the user would be the one to choose between, for example, a direct method on a band matrix and an iterative method on a sparser matrix, because that involves judging trade-offs among speed, storage, accuracy, robustness, etc. But the Matrix Editor would then decide, among many other things, how many arrays to allocate for the Jacobian and how big they will have to be.

Internally, the Matrix Editor works on a detailed representation of the symbolic Jacobian matrix. The most important part of this representation involves descriptions of four ways to index the symbolic matrix elements, and mappings between the different kinds of index. Specifically, a matrix element can be identified by (i) row and column number, (ii) indices referring to the original physical problem, e.g. which partial differential equation and which spatial zone produced a matrix row, (iii) indices identifying the symbolic expression used to compute the matrix element; this typically will arise from a term of a partial differential equation, (iv) where its numeric value will be stored in a Fortran array, as well as an identifier for the array itself. The functions to map between the different indices are computed at runtime, as the necessary information becomes available. Most changes in the representation of the symbolic matrix amount to a computation of new versions of these index mapping functions.

The Matrix Editor is designed to be easy to interact with; that is, to look at the symbolic Jacobian and tell the Matrix Editor what to do with it. Thus the Matrix Editor finds an example of the Jacobian matrix, with the necessary symbolic parameters (e.g., number of zones in a discretization) temporarily bound to particular numbers. Then it draws a picture of that example matrix. By clicking a mouse on menus and the picture, and by typing expressions where necessary, one can change the display and tell the Matrix Editor to compute a new internal representation of the symbolic Jacobian. In particular, one can look at pictures showing how closely the matrix fits the Fortran arrays being used, what expressions appear where in the Jacobian, etc. One can choose standard sparsity structures and linear solver methods (it also is relatively easy to define ones own) and reorder rows and columns in those ways which can be simply defined for symbolic matrices. Once the displayed matrix looks good, ALPAL can go on to generate Fortran code to compute the Jacobian, load it into the appropriate arrays, call a linear solver, and use the results.

Construction of Large-Scale Simulation Codes Using ALPAL

Grant O. Cook, Jr.

L-316, Lawrence Livermore Laboratory
Livermore, CA 94550

Many computational scientists have developed a small simulation code at one time or another. Codes of this scale are characterized by being quickly constructed, uncluttered by operating system-dependent details, and easy to debug. On the other hand, large-scale simulation codes usually take many man-years to construct, remain buggy even after years of effort, have complex internal operating characteristics, and support significant on-line user interaction. For small simulation codes, many feel that Fortran or other high-level languages are adequate programming vehicles. But this feeling is not retained for large-scale simulation codes. While a fair amount of effort has been directed at ameliorating this situation, it is only recently that some partial solutions have been found for this state of affairs. A Livermore Physics Applications Language [1] (ALPAL) is a new tool that fills in more solutions.

Large-scale simulation codes are usually composed of an input generator, numerical modules that solve a particular set of models, and post processors for analyzing the solution data. Sometimes, there are also linkers for transforming results from one code into the form required as input in another code.

ALPAL takes as input a set of integrodifferential equations with associated boundary and initial conditions. The dependent and independent variables must also be specified. For brevity of notation and adherence to journal article presentation style, the input model can be stated in terms of as many auxiliary or subsidiary variables as the user chooses to employ. ALPAL supports general finite difference techniques on logically rectangular grids, and allows the user to either employ the built-in first-order techniques or to supply operators that are more appropriate. Hence, the input must also contain the specification of all finite difference meshes and all added finite difference operators. After input of the model is complete, ALPAL performs the following major steps: discretization, analysis of problem details, Jacobian computation, and code generation. ALPAL-generated code can interface to table lookup systems, as well as already established simulation systems. Furthermore, it can perform "edits" of dependent variables, subsidiary variables, and time-integrated "edit variables" to provide input to a postprocessor for diagnostic and debugging purposes.

While ALPAL focuses on automatically generating the numerical modules in a simulation code, there are significant advances that have been made in the other parts of a large scale simulation code through the use of extensible code systems or "service" languages such as BASIS [2] and PANACEA [3]. These systems are important because they help to make a simulation code more robust and easy to use; i.e., they save labor in the long run. Soon, ALPAL will have the capability to generate code modules for simulations codes that use either BASIS or PANACEA.

For numerical code modules, many approaches have been tried in order to make their development easier and less error prone. Of these, only ALPAL has begun to adequately address the needs of large-scale simulation codes. Further development of

ALPAL will enable it to be used to construct a wide variety of complex simulation codes.

However, ALPAL does more than make it easier to develop numerical modules. It also leverages the scientist's abilities and creativity with a number of important features. First, ALPAL gives the scientist the ability to develop numerical modules that are too hard to derive by hand because of complex discretization and derivative computations. It also accelerates the development of numerical modules that can be tackled by traditional methods.

Second, because large amounts of algebra are associated with both the development of appropriate physical models and the discrete versions of these models on a computer, ALPAL eliminates much of the drudgery and concomitant errors in constructing simulation codes. With fewer errors to worry about, more of the scientist's energies can be focused on improving the models and numerical methods used. Clearly, this would help the computational scientist to be more creative insofar as the modeling issues are concerned.

Third, a more natural way of describing the physical model is possible with such a language. For example, when using the natural language of physics as opposed to Fortran, it is far more transparent how to make modifications to both the physical model and the numerical methods. This results in a savings of time that might be used to examine interesting modifications to the physical model or to the numerical methods.

Fourth, it becomes possible to automate the computation of Jacobian matrices both for linear and nonlinear problems. In many cases, Jacobian computation has not previously been tractable. Hence, implicit numerical methods are possible to use now where they had not been considered viable before.

Fifth, ALPAL produces highly optimized simulation code modules. Nothing is sacrificed in not writing the code by hand.

Sixth, the cost of developing simulation codes is great, especially for new machine architectures. ALPAL generates vectorizable code where the discretization and solution techniques permit, and in principle, capabilities could be added to generate code for parallel computers.

Seventh, ALPAL outputs good high-level documentation. This documentation clearly shows the modular structure of the generated code, and helps the computational scientist to have some confidence that the automatically generated code module corresponds correctly to his input.

In sum, ALPAL presents the user with a rather unusual opportunity. This opportunity is the capability to tackle otherwise infeasible problems, as well as to be more creative in the areas in which the scientist was trained. This is only possible through the removal of mundane work and the associated arcane errors in the process of developing a simulation code.

References

[1] G.O. Cook, Jr., ALPAL: A program to generate physics simulation codes from natural descriptions, *Int. J. Mod. Phys. C*, to appear.

[2] P.F. Dubois, The basis system, *M-225*, Lawrence Livermore National Laboratory, (June 1989).

[3] S.A. Brown, PANACEA Users manual, *M-276*, Lawrence Livermore National Laboratory, (1989).

A Functional Representation of Software Selection Expertise
(Extended Abstract)

*Michael Lucks**
*Ian Gladwell***

*Department of Computer Science and Engineering
**Department of Mathematics
Southern Methodist University
Dallas, TX 75275

Users of numerical subroutine libraries often encounter difficulties in determining the software most appropriate for solving a given problem. For certain important classes of problems (e.g., differential equations, algebraic equations, quadrature and nonlinear optimization), there may be many applicable codes and the best choice depends on a variety of factors, including the properties of the input problem, the requirements of the user and the constraints of the computational environment. In an earlier paper [Gla90], we described some of these difficulties and outlined the design of a rule-based consultation system to aid users in the software selection task. From a knowledge engineering standpoint, production rules are a clear improvement over previous methods for providing such assistance (e.g., the decision tree approach used in [Add86] and [Add89]), particularly with respect to extendibility and understandability of the knowledge base. Subsequent investigation, however, has exposed serious weaknesses in the use of production rules (as well as related methods such as logic programming [Sch88]) to represent software selection expertise. The main limitation of the rule-based approach lies in the coarse, inflexible facilities for interpreting, comparing and aggregating multiple pieces of quantitative information. Production rules require that numerical data be discretized into qualitative abstractions that can be manipulated symbolically. For example, the number of equations in a system of ODEs might be used to confirm or disprove the logical hypothesis "the system is large". Lost in this discretization is the knowledge of exactly "how large", i.e., borderline cases are indistinguishable from extreme cases. In the case of ODEs, similar losses of information occur for other continuous performance criteria such as stiffness, expense of evaluation, number of discontinuities, the desired accuracy of the solution, etc. Since the overall assessment of a code's suitability may depend on all of these factors, the total amount of lost information can be significant. Although the information loss can be reduced by using finer-grained symbolic abstractions, this remedy can result in a severe increase in the size of the knowledge base (and hence a severe decline in the comprehensibility of the encoded knowledge).

The various numerical extensions to production systems (e.g., certainty factors [Sho75], measures a belief [Dud79], belief functions [Sha76], do not provide the appropriate semantics for modeling the quantitative information that is often critical in software selection. Furthermore, we argue that their static, built-in mechanisms for propagating and aggregating numerical data cause additional losses of information.

This work was supported in part by Texas Instruments, Inc.

In the full paper we present an alternative knowledge representation scheme that avoids the above difficulties, but retains the advantages of rule-based systems, e.g., ease of modification and support for explanation. In the new representation, expertise is encoded as real-valued functions that describe certain relationships between the set of available *software modules* $S = \{s_1, s_2, \ldots, s_k\}$ and the set of *performance features* $F = \{f_1, f_2, \ldots, f_n\}$. In addition to characteristics of the input problem (such as stiffness, system size, range of integration, etc.), the feature set also includes user performances (such as requested accuracy) as well as environmental constraints (e.g., memory size). To evaluate the performance of codes from S for an arbitrary problem p from some problem domain P, we construct a *suitability function* H: $P \times S \rightarrow L$, where L is an arbitrarily chosen *evaluation interval* for H. The module s_i such that $H(p,s_i) \geq H(p,s_j)$ for all $i \neq j$ is considered to be the most suitable module for solving the problem. $H(p,s_i)$ is not an estimate of the absolute performance of s_i. Rather, the values of $H(p,s_j)$ for $j = 1, \ldots, k$ are estimates of the *relative* performance of $s_1, \ldots, s_k$ with respect to a user-specified set of performance objectives (e.g., efficiency, ease of use, reliability, etc.).

The knowledge functions provided by the expert represent an attempt to approximate H as a composition of smaller, modular, more understandable functional components. We employ four types of components, each supplied by the domain expert:

1. real-valued *measurement functions*, M_j: $P \rightarrow R$, $j = 1, \ldots, n$, that compute quantitative values describing the presence of each feature f_j in the input problem,
2. *intensity functions* I_j: $R \rightarrow L_I$, $j = 1, \ldots, n$, that normalize the output of the measurement functions into a uniform scale L_I,
3. *compatibility functions* $C_{i,j}$: $L_I \rightarrow L_C$, $i = 1, \ldots, k$, $j = 1, \ldots, n$, that estimate the degree (normalized in L_C) to which the intensity value of each feature f_j is compatible with the behavioral characteristics of each module s_i,
4. *aggregation functions* A_i: $F^n \rightarrow L$, $i = 1, \ldots, k$ that assess the overall suitabilities of the modules $s_1, \ldots, s_k$, given the compatibilities of the modules with respect to the individual features $f_1, \ldots, f_n$.

We describe how the intensity and compatibility functions may be viewed as continuous generalizations of discrete production rules, while the aggregation functions allow the expert to specify precisely how different features contribute to a module's overall suitability. The suitability function H is approximated via functional composition of the smaller components. The control flow for this composition may be visualized as a network, in which the edges represent knowledge functions and the nodes represent the functions' input and output values. The configuration of the network depends on the expert's view of the problem domain. The simplest configuration is shown in Figure 1, which illustrates the computation of the suitability for some software module s_i. The input problem p at the bottom of the diagram is processed upward by the knowledge functions, eventually yielding the suitability value at the top. First, the n measurement functions are applied to the input, yielding n measurement values $m_1, \ldots, m_n$. The intensity functions then map the respective measurement values into intensity values $d_1, \ldots, d_n$, which, in turn are mapped into n compatibility values by the compatibility functions. Finally, the compatibility values are aggregated via A_i into the overall estimate of the module's suitability for the input problem. A separate network (perhaps configured differently) exists for each module and we approximate the suitability function H by the function $\hat{H}: P \times S \rightarrow L$, defined as

$$H(p,s_i) = A_i(C_{i,1}\ o\ M_1(p), C_{i,2}\ o\ I_2\ o\ M_2(p), \ldots, C_{i,n}\ o\ I_n\ o\ M_n(p))$$

where o denotes functional composition.

The configuration shown in Figure 1 requires the expert to assess the direct impact of each feature on the (single) performance criterion. In the paper, we also describe more complicated configurations that allow multiple levels of aggregation via a hierarchy of features and performance criteria. Although such hierarchical arrangements increase the complexity of the knowledge base, they each the process of extracting information from the domain expert.

The knowledge representation scheme and an associated control mechanism comprise a generic framework, similar to an expert system shell, for generating software selection consultation systems. A prototype implementation of this framework has been used to build a knowledge base for the domain of ordinary differential equation initial value problems. Given a description of the input problem, the resulting system requires a list of appropriate ODE software, ranked according to criteria supplied by the user. The prototype framework and the generated knowledge base are described in the full paper.

References

[Add86] C.A. Addison, W.H. Enright, P.W. Gaffney, I. Gladwell and P.M. Hanson, A decision tree for the numerical solution of initial value ordinary differential equations, Chr. Michelsen Institute, CCS 86/3, (1986).

[Add89] C.A. Addison, W.H. Enright, P.W. Gaffney, I. Gladwell and P.M. Hanson, A decision tree for the numerical solution of boundary value ordinary differential equations, *SMU Technical Report 87-7*, (1989).

[Dud79] R.O. Duda, P.E. Hart, P. Barret, J. Gashniq, K. Konolige, R. Reboh and J. Slocum, Development of the prospector consultation system for mineral exploration, *SRI International*, Menlo park, California, (1979).

[Gla90] I. Gladwell and M. Lucks, An automated consultation system for the selection of mathematical software, in: *Fourth Generation Mathematical Software Systems*, (E. Houstis, J. Rice, R. Vichnevetsky, eds.), (1990).

[Sch88] K. Schulze and C. Cryer, NAXPERT: A prototype expert system for numerical software, *SIAM Journal on Scientific and Statistical Computing*, Vol. 9, No. 3, (May 1988), pp. 503–515.

[Sha76] G. Shafer, A mathematical theory of evidence, Princeton University Press, (1976).

[Sho75] E. Shortliffe and B. Buchanan, A model of inexact reasoning in medicine, *Mathematical Biosciences*, Vol. 23, (1975), pp. 351–379.