PELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms

**E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin,
P. Papachiou, K.-Y. Wang and M. Gaitatzes**

ABSTRACT

This paper presents the software architecture and implementation of the problem solving environment (PSE) PELLPACK for modeling physical objects described by partial differential equations (PDEs). The scope of this PSE is broad as PELLPACK incorporates many PDE solving systems and some of these, in turn, include several specific PDE solving methods. Its coverage for 1-D, 2-D and 3-D elliptic or parabolic problems is quite broad, and it handles some hyperbolic problems. Since a PSE should provide complete support for the problem solving process, PELLPACK also contains a large amount of code to support graphical user interfaces, analytic tools, user help, domain or mesh partitioning, machine and data selection, visualization, and various other tasks. Its total size is well over 1 million lines of code. Its open-ended software architecture consists of several software layers. The top layer is an interactive graphical interface for specifying the PDE model and its solution framework. This interface saves the results of the user specification in the form of a very high level PDE language which is an alternative interface to the PELLPACK system. This language also allows a user to specify the PDE problem and its solution framework textually in a natural form. The PELLPACK language preprocessor generates a Fortran control program with the interfaces, calls to specified components and libraries of the PDE solution framework, and functions defining the PDE problem. The PELLPACK program execution is supported by a high level tool where the virtual parallel system is defined, where the execution mode, file system, and hardware resources are selected, and where the compilation, loading, and execution are controlled. Finally, the PELLPACK PSE integrates several PDE libraries and PDE systems available in the public domain. The system employs several parallel reuse methodologies based on the decomposition of discrete geometric data to map sparse PDE computations to parallel machines. An instance of the system is available as a *Web server* (***WebPELLPACK***) for public use at the *http://pellpack.cs.purdue.edu*.

keywords: domain decomposition, expert systems, framework, knowledge bases, parallel reuse methodologies, parallel solvers, problem solving environments, programming-in-the-large, programming frameworks, software bus.

1. INTRODUCTION

The concept of a mathematical software library was introduced in the early 70s [41] to support the reuse of high quality software. In addition, special journals, conferences, public domain software repositories (e.g., ACM, Netlib), and commercial libraries (i.e., IMSL, NAG) have been established to support this concept. Similar efforts can be found in engineering software, particularly in the areas of structural and fluid mechanics. The increasing number, size, and complexity of mathematical software libraries necessitated the development of a classification and indexing of existing and future software modules. This software is currently organized in terms of the mathematical models involved. A significant effort in this direction is the GAMS on-line catalog and advisory system [5] which has become a standard framework for indexing mathematical software. Information about engineering software can be found in several handbooks which usually describe the applicability and functionality of existing packages. The advances in desktop software/hardware, workstation clustering and distributed computing technologies, and the ease of access to supercomputing facilities have made computational prototyping a new, cost effective alternative for the design of new products and for the study of science and engineering phenomena in general. Although the software library provides some form of abstraction and a facility of reusing software parts, it still requires a level of computing expertise above the background and skills of the average scientist and engineer who usually design manufactured products. This recognition has lead to the new concept of software reuse referred throughout as Problem Solving Environment (PSE). The current PSEs consist of small sets of modules, usually taken from existing libraries, integrated (packaged) to handle a predefined class of mathematical models. In these PSEs the specification of the mathematical model, the problem solving process, and the required pre-processing or post-processing phases are done with a high level user interface. This interface usually consists of a very high level language and graphical interface that allows the user to specify the problem and visualize the solution in some “natural” form. Early examples of PSEs are Macsyma, Mathematica, Maple, ELLPACK, MatLab, and several engineering software systems. Similar software evolution can be observed in the pre-processing (CAD, mesh generation) and post-processing (data visualization) tools. These PSEs and the associated pre- and post-processing tools have greatly increased the abstraction of computational prototyping for some applications. As a result users with minimum computational background can be engaged in the prototyping of complex artifacts. PSEs are distinguished with respect to the domain of problems or applications they can handle.

An important distinction between a PSE and a monolithic software system is that PSE's have a flexible and extensible architecture that is easy for a user to tailor or a builder to enhance. The software architecture of PSEs is characterized by the integration model used to connect the software parts involved and the underlying execution model assumed.The common shortcoming of current PSEs is that the knowledge associated with the library, the applicability, compatibility, and performance (i.e. complexity) of library modules, the selection of the computational parameters, error estimation, etc. is not part of the PSE but is part of the responsibility of the user. One can argue that the ideal PSE should make decisions to help the user by consulting a knowledge base about the user, the problem domain, and past solutions of similar problems. This leads us to the following formal definition of a PSE:

PSE = User interface + libraries + knowledge base + software bus.

In this paper we describe the architecture and functionality of a PSE called PELLPACK for solving certain classes of partial differential equations (PDEs) on sequential and multicomputer platforms. It is a descendent of ELLPACK [40] which allows users to solve PDEs for linear and nonlinear field and flow problems. Figure 1 depicts a user’s view of the PELLPACK system in terms of the tools and libraries needed to specify and solve a PDE problem on a target computational platform and to visualize the solution. Figure 1 is further illustrated by a PDE solving scenario in section 2.4.2. PELLPACK provides an interactive graphical user interface for specifying the PDE model, its solution method and post-processing, supported by the Maxima symbolic system and well known libraries. In addition, it provides an intermediate high level facility for composing new algorithms from existing parts and it supports a programming-in-the large environment with a language which is an extension of the ELLPACK language [40]. The user interface and programming environment of PELLPACK is independent of the target machine architecture and its native programming environment. PELLPACK is supported by a library of parallel PDE modules for the numerical solution of stationary and time dependent single equation PDE models on two and three dimensional regions. A number of well known “foreign” PDE systems have been integrated into PELLPACK which are listed in Table 1. PELLPACK can simulate structural mechanics, semi-conductors, heat transfer, flow, electromagnetic, microelectronics, and many other scientific and engineering phenomena. Five different implementation languages have been used to build the system. The current size of PELLPACK is 1,900,000 lines of code. The parallel codes of PELLPACK currently use the PICL, PARMACS 5.1, MPI, PVM, NX and Vertex communication libraries. The size of the parallel library is 128,000 lines of Fortran code for each implementation and consists of finite element and difference modules for discretizating elliptic PDEs, a parallelization of the ITPACK library [28], [30], [32] and the MP-PCG (parallel preconditioning conjugate gradient) package [44]. The parallel library is based on the discrete domain decomposition approach and it is implemented in both the host-node and hostless programming paradigms. A number of tools and libraries exist to support the domain decomposition methodology and estimate (specify) its parameters. For the reuse of existing “legacy” sequential PDE software we have implemented two domain decomposition based reuse methodologies described in [33].

The paper is organized in nine sections. Section 2 describes the exact applicability of the system in terms of the existing PDE libraries and pre-defined frameworks. We list several standard solution frameworks for various PDE models, and we describe the frameworks needed to use one of the integrated “foreign” systems. In addition we describe parallel re-use frameworks for steady-state PDE software. The multi-level PELLPACK architecture is discussed in Section 3, and Section 4 describes the three level programming environment. The PELLPACK PSE allows the user to execute programs in a variety of physical and virtual parallel architectures. Section 5 describes a visual scripting execution environment that allows the user to select the computers and to direct the running of computations and the visualizing of results. Section 6 describes an expert system methodology that can be used to implement the adaptability of the system to user's expertise and computational objectives. This methodology and its software has been implemented and tested in the context of the ELLPACK library [25] whose highlights are presented in Section 6. Section 7 presents two scenarios that demonstrate the PELLPACK design objective of reuse of high quality mathematical software, the facility for development of new PDE software, and the integration of "foreign" software. The future scenario for usage and maintenance of high quality mathematical software calls for remote "net-centered" servers and networked software that will allows users to compute over the "network" as they compute in the world of front-end workstation to an intranet computer system. We have created a Web server for PELLPACK that allow users to experiment with the system and get answers, instead of downloading software and addressing issues of local installation, maintenance, and licensing. This server and its accessibility is described in Section 8 and its Web location is http://pellpack.cs.purdue.edu.

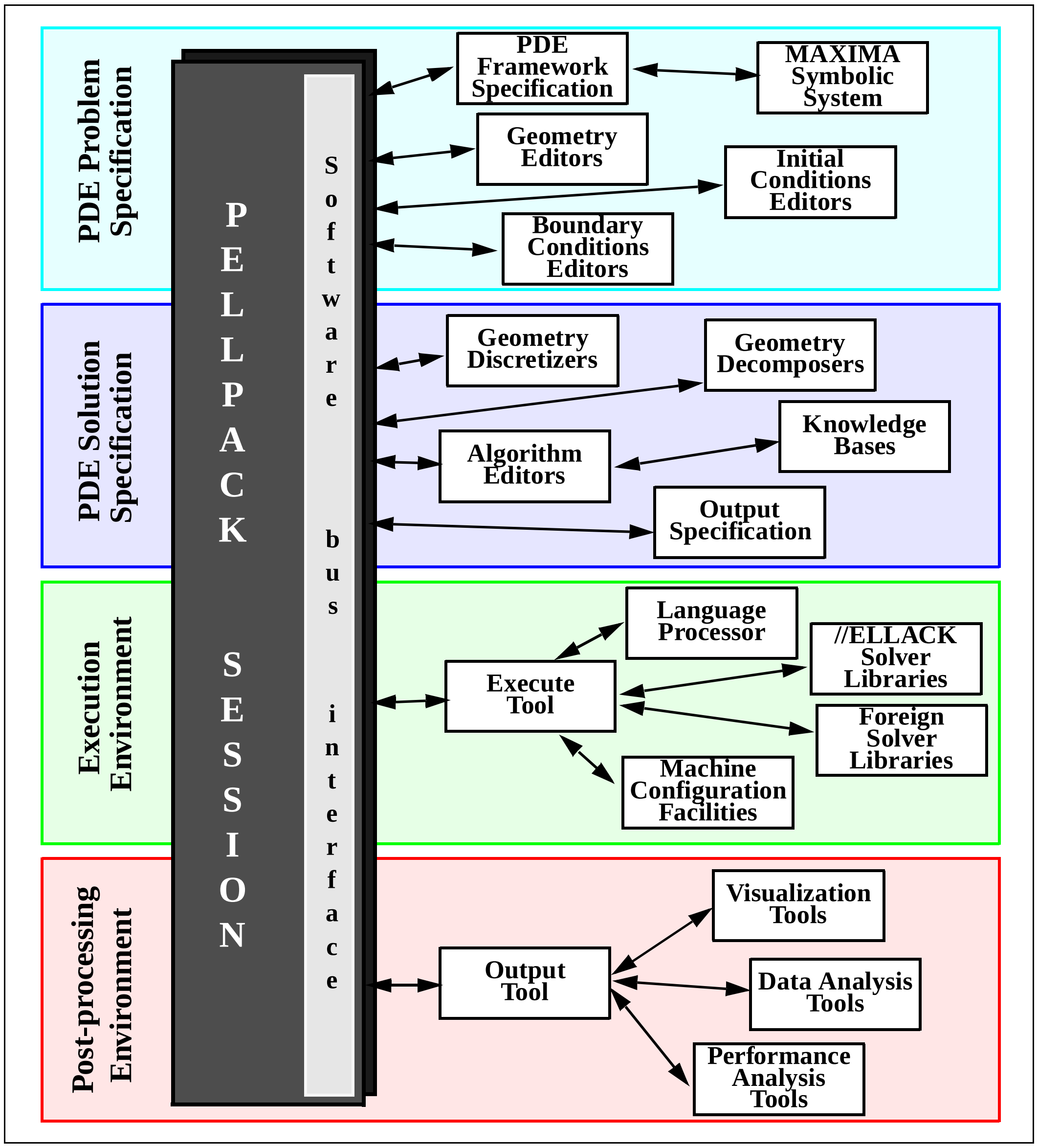


FIGURE 1. A user's view of the PELLPACK system depicting the tools and libraries supported. The diagram is organized in terms of the four solution phases involved in PDE computing: problem specification, solution specification, problem execution, and solution post-processing.

This work is the result of a significantly large group of people and the support of many government and industrial organizations listed in alphabetical order in Section 9.

2. DOMAIN OF APPLICABILITY

The applicability of the PELLPACK system is defined in terms of the types of PDE software libraries integrated into the system, and the pre-defined algorithm skeletons and frameworks directly supported at the PELLPACK very high language and graphical user interface levels. An algorithm skeleton is a "solution driver", i.e., a specification of the methods which are to be used in the solution of a PDE problem. A PELLPACK *framework* is a customized solution driver, requiring a specialized form of PDE problem and solution specification. The form of this specification is determined by the user-selected PDE software library to be used in the solution process. The framework includes the solver system selection, the mathematical representation of the PDE model (which often depends upon the selected solver), and the interfaces between the solver library and the PELLPACK runtime system. Most frameworks in PELLACK handle general (systems of) PDEs. A PELLPACK *template* is a framework for a specific PDE model, such as the Navier-Stokes equations. The PDE specification in this case is a set of parameter values.

2.1 PDE SOFTWARE LIBRARIES

The PDE libraries currently integrated in PELLPACK are listed in Table 1. They allow the numerical solution of *field* and *flow* PDE problems in various geometric regions. The integration of these simulation libraries is done at the PDE language, graphical interface, and data interface levels. The PELLPACK programming environment allows *differential*, *variational*, and *template* forms for specifying the PDE and auxiliary operators. The PELLPACK PDE problem specification and its "derivatives" (i.e., Jacobian, linearization transformations, forcing functions) are computed and converted symbolically to the pre-defined Fortran interface format assumed by the selected PDE library. The 3-D PDE domain geometry can be specified *only* in terms of files in well established geometry data formats (e.g., polyfile) that PELLPACK recognizes. The system provides a 2-D geometry specification tool.

TABLE 1. PDE systems integrated in PELLPACK, their applicability, and major characteristics

Solver Name	PDE Model Type	Mathematical Representation and Mesh Restrictions	Dimensionality and Geometry	References
ELLPACK	single elliptic equation	Differential e.g. $uxx + uyy = f$	2-D general, 3-D box geometry	[40]
PELLPACK	single elliptic equation	Differential	2-D and 3-D general geometry	[21], [22], [23], [29], [57]
VECFEM	non-linear, elliptic, parabolic systems, eigenvalue problems	Variational e.g. $\int_{\Omega}(u_x v_x + u_y v_y)d\omega = \int_{\Omega} fv\,d\omega$	1-D, 2-D, 3-D general geometry	[17]
FIDSOL	nonlinear, elliptic, parabolic systems	Differential	2-D and 3-D box geometry	[43]
CADSOL	nonlinear, elliptic, parabolic systems	Differential	2-D general geometry	[42]
PDECOL	nonlinear, parabolic systems	Differential	1-D interval	[31]

TABLE 1. PDE systems integrated in PELLPACK, their applicability, and major characteristics

ITGFS	2-D Navier-Stokes	Template, structured meshes e.g. *transonic turbulence flow parameter values*	2-D general geometry	[57]
NSC2KE	2-D Navier-Stokes	Template, structured meshes	2-D general geometry	[3]
NPARC3-D	3-D Navier-Stokes	Template, multi-block structured meshes	3-D general geometry	[10]
PDEONE	nonlinear, parabolic systems	Differential	1-D interval	[19]

2.2 FRAMEWORKS FOR PELLPACK PDE SOLVERS

The design of the PELLPACK programming environment (i.e., a very high level PDE language and interactive editing tools) has been influenced by the requirements of its current solving capabilities and the structure of the solution skeletons (i.e., drivers) that the user is allowed to specify and run. Other solution frameworks, can be easily created in the PELLPACK system by utilizing the pre-defined *fixed interfaces* among the PDE solution phases, existing or new PDE software parts, and Fortran code. For example, the parallel time-stepping methodology described in [53] has been implemented in PELLPACK utilizing a variety of PELLPACK iterative solvers and its performance was measured on a variety of platforms [48]. In this section we describe the various pre-defined solution frameworks that PELLPACK currently supports.

2.2.1 ELLIPTIC AND PARABOLIC PDE SOLUTION FRAMEWORKS

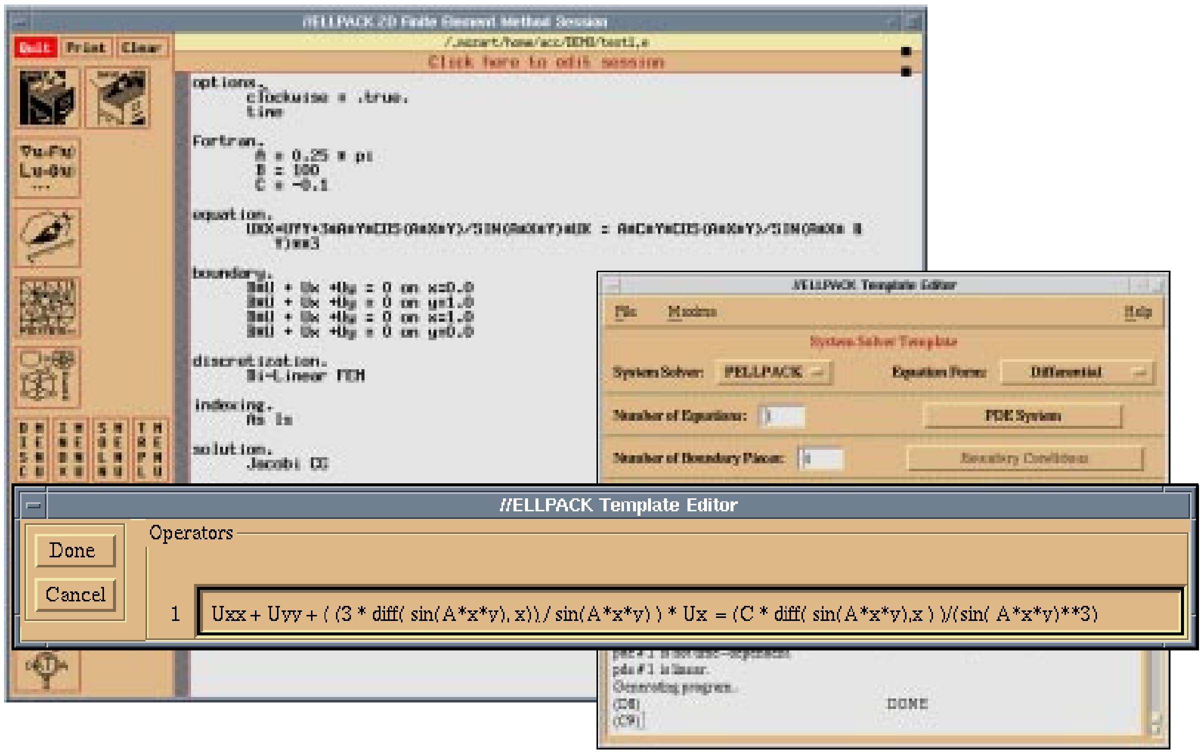
PELLPACK allows the solution of single linear and non-linear elliptic and parabolic PDE equations defined on 2-D and 3-D domains. In this framework, the user can specify a solution method by naming (referencing) selected library modules (`discretization, indexing, solution`) corresponding to the phases of the PDE solution process [40] (see Figure 2 for an example). In the case of coupled or single-phase solvers the name of the `triple` module is specified. Framework 1 below lists the segments of this framework. The parallel elliptic framework currently supported in PELLPACK is based on geometric partitioning of the grid or mesh data. Thus, the user is required to specify the decomposition data in the form of a file with appropriate format and parameters. This segment can be generated by an interactive editor which allows the visualization and editing of mesh/grid decomposition data and uses a library of semi-optimal partitioning algorithms for their automatic generation [7], [9], [54]. In the case of parallel elliptic solvers, the parallel versions of the library modules specified have been implemented using several virtual (e.g., PVM, MPI) and machine native (e.g., Vertex, NX) communication libraries [28],[29],[32].

FRAMEWORK 1. Module based linear elliptic solution

Segment	**Description**	**Options**
Declarations, Options	Space for saving solution, parallel machine configuration and model	sequential, parallel
Equation, BCs	PDE problem definition	differential
Grid/Mesh	Domain discretization	sequential, parallel
Decomposition	Grid/Mesh partitioning file needed for the parallel solution	sequential, parallel
Multi-phase PDE solver		
Discretization	PDE problem discretization	sequential, parallel

Indexing	Discrete equations ordering scheme	sequential, parallel
Solution	Linear solver	sequential, parallel
Single-phase PDE solver		
Triple	Integrated discretization, indexing, solution phases	sequential
Output	Format for solution output	

For non-linear elliptic PDEs, a linearization procedure is applied at the continuous PDE problem level which is described in [51]. This framework is generated symbolically using the Maxima-based PDE framework specification editor of the PELLPACK graphical interface, which is described in Section 4.2. Framework 2 describes the segments of this framework.


//ELLPACK 2D Finite Element Method Session
Quit
Print
Clear
/.smart/home/acc/DEMO/test1.e
Click here to edit session
//ELLPACK Template Editor
File
Maxima
Help
System Solver Template
System Solver:
PELLPACK
Equation Form:
Differential
Number of Equations:
PDE System
Number of Boundary Pieces:
Boundary Conditions
//ELLPACK Template Editor
Done
Cancel
Operators

```
options.
      clockwise = .true.
      time

Fortran.
       A = 0.25 * pi
       B = 100
       C = -0.1

equation.
      UXX+UYY+3*A*Y*COS(A*X*Y)/SIN(A*X*Y)*UX = A*C*Y*COS(A*X*Y)/SIN(A*X* B
         Y)**3

boundary.
      B*U + Ux +Uy = 0 on x=0.0
      B*U + Ux +Uy = 0 on y=1.0
      B*U + Ux +Uy = 0 on x=1.0
      B*U + Ux +Uy = 0 on y=0.0

discretization.
      Bi-Linear FEM

indexing.
      As Is

solution.
      Jacobi CG
```

```
1   Uxx + Uyy + ( (3 * diff( sin(A*x*y), x)) / sin(A*x*y) ) * Ux = (C * diff( sin(A*x*y),x ) )/(sin( A*x*y)**3)
```

```
pde # 1 is not disc-dependent.
pde # 1 is linear.
Generating program...
(C8)                                DONE
(C9)
```

FIGURE 2. An instance of PELLPACK user interface for an elliptic framework

FRAMEWORK 2. Nonlinear sequential elliptic PDE solution

Segment	**Description**
Declarations, Options	Space for saving solution(s)
Equation, BCs	PDE problem definition
Grid/Mesh	Domain discretization
Triple	Initial guess
Fortran	Newton loop start
Linearized Elliptic Solver	Elliptic problem discretization, indexing, solution

FRAMEWORK 2. Nonlinear sequential elliptic PDE solution

Output	Format for solution output
Fortran	Convergence test
Fortran	Newton loop end
Subprograms	Initial guess, Jacobian and other support functions

Similarly, there is a framework for implementing semi-discrete parabolic PDE solvers which utilizes the available PELLPACK elliptic PDE solvers. In this case users can select pre-defined time discretization schemes or specify their own and reduce the parabolic PDE problem to a set of elliptic PDEs defined at each time-step. The framework for these solvers is described in Framework 3 and [51].

FRAMEWORK 3. Parabolic sequential PDE solution

Segment	**Description**
Declarations, Options	Space for saving solution(s)
Equation, BCs	PDE problem definition
Grid/Mesh	Domain discretization
Triple	Initial condition
Fortran	Time stepping loop start
Elliptic PDE solver	Elliptic problem discretization, indexing, solution
Output	Format for solution output
Fortran	Convergence test
Fortran	Time stepping loop end
Subprograms	Initial condition and other support functions

2.2.2 MPLUS (MATRIX PARTITIONING) STEADY-STATE SOLUTION FRAMEWORK

This framework is applicable to any non-time dependent PDE computation and is designed to re-use existing sequential PDE discretization software in a parallel solution scheme. It assumes that the discrete equations are generated *sequentially* with any of the existing libraries. It uses mesh/grid decomposition data or user defined partitions for the algebraic data structures associated with the selected PDE solver. The partitioned discrete PDE (i.e., algebraic) equations are loaded into the targeted multicomputer platform and solved in parallel by the available parallel solvers. Framework 4 displays the skeleton of this framework. The methodology and its performance evaluation described in [33].

FRAMEWORK 4. Parallel matrix solution

Segment	**Description**
Sequential solution framework	The PDE problem, its discretization, and sequential solver
Partition	Discrete geometric or user defined algebraic data partitioning strategy
Load	Loads partitioned algebraic system

Display	Display the structure of partitioning system
Solve	Apply a parallel solver
Output	Format for solution output

2.2.3 DPLUS (DOMAIN PARTITIONING) STEADY-STATE SOLUTION FRAMEWORK

This framework is currently applicable to steady-state PDE models and their derivatives (i.e., implicit parabolic solvers) defined on 2-D and 3-D domains. It is also based on a methodology to reuse sequential PDE discretization software in a parallel computation [33]. It involves a decomposition of the model based on a balanced partitioning of the PDE domain with appropriate artificial interface conditions that allow the uncoupled generation of the discrete equations in each subdomain. The decomposition of the domain is obtained via the partitioning of a relative course grid or mesh [7]. Unlike MPlus, DPlus runs the sequential discretization code in parallel (i.e., each processor runs sequential code on its assigned subdomain). Framework 5 lists the segments of this framework.

FRAMEWORK 5. Parallel stationary PDE solution

Segment	**Description**
Declarations, Options	Space for saving solution, parallel machine configuration and model
Equation, BCs	PDE problem definition
Mesh generation and decomposition	Parallel multiphase mesh generation and decomposition
Interior interface conditions	Interior interface BCs definition so that the generation of global discrete equations among sub-domains is decoupled
PDE discretization	Local PDE problem discretization in parallel
Solve	Parallel solution of distributed discrete PDE equations
Output	Format for solution output

2.3 FRAMEWORKS FOR “FOREIGN” PDE SYSTEMS

Most general PDE solving systems require users to define PDE problems by writing Fortran functions with fixed argument lists and data structures for the PDE equation, boundary, and initial conditions. Users write driver programs to allocate space, initialize variables and call the solver routines with appropriate parameters and control variables. Often, Jacobians or other symbolic computations are also required, and the results of these computations must be written as additional Fortran functions. The functions and driver are compiled and linked with the solver library to produce the program. PELLPACK generates these functions and drivers symbolically for the PDE solving systems of Table 1 and the frameworks presented in the previous sections. This is the result of the integration at several PELLPACK levels.

A “foreign” PDE system can be integrated in PELLPACK at the *PDE language level*, the *graphical interface level*, and the *data level*. Each level of integration provides a further level of abstraction by placing an additional software interface layer between the user and the foreign system, thus simplifying the input required for defining a PDE problem. To support the *language level integration*, a specialized interface library is developed for each system. The interface code defines the required data structures, allocates space, initializes variables, and calls the appropriate system solver routines with appropriate values for the parameters. Users still specify the PDE problem and symbolic computations via Fortran functions that are similar (or identical) to those required by the original system, and these functions are placed in the `subprograms` segment of the PELLPACK problem definition. Users name the solver in a high level way and identify various high level problem characteristics such as number of equations, non-linearity, and time-dependence. The language integration supplies default parameter values when needed. Interface routines for all systems generate PELLPACK format output which is used for visualization and animation of solution data with PELLPACK's output tool (see Section 4.3). The PELLPACK execution environment identifies the selected system solver so that it can link automatically with the correct library to generate the program. The language interface simplifies the specification of the PDE problem and sets the foundation for integration at the graphical level.

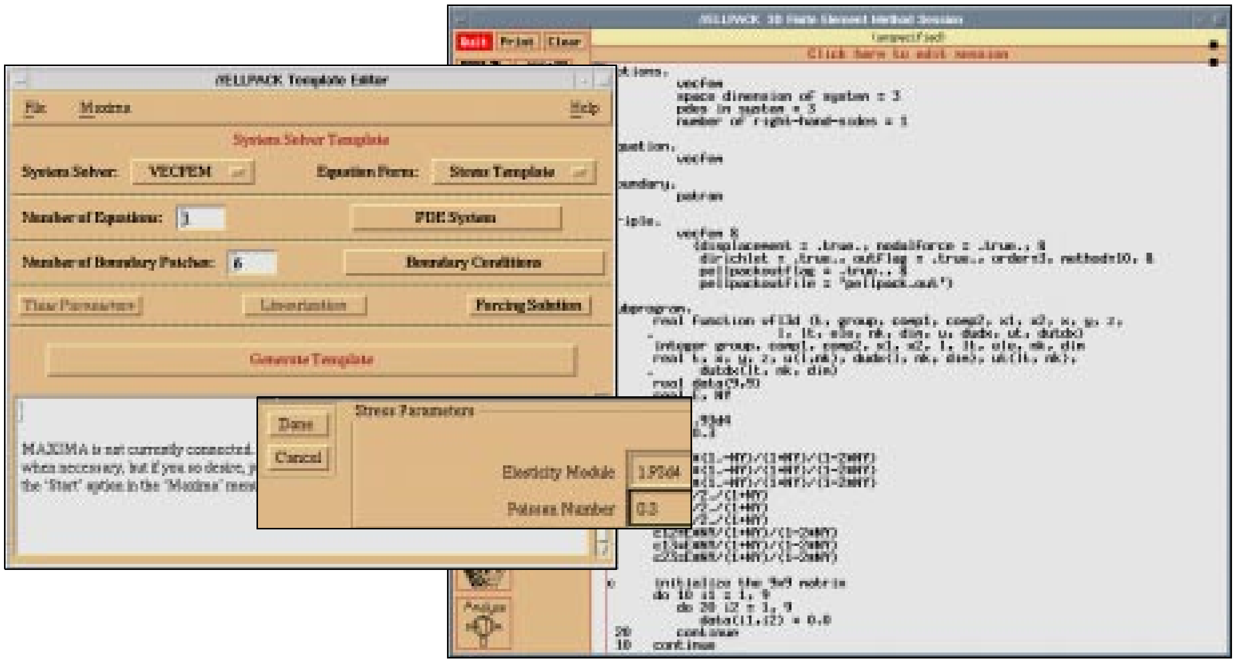
At the *graphical interface level*, users can define PDE problems using a graphical editor. To simplify the process of specifying the PDE system, the interfaces are tailored to the representation of the equation(s) used in the selected system. After a user enters the equations, the editor determines what symbolic manipulations are needed for defining the problem with the selected framework, and accesses the Maxima symbolic system to perform the computations. The editor generates the Fortran functions in the format required by the solver, and places them in the `subprograms` segment. High level problem characteristics are identified symbolically, and the editor assigns appropriate values to solver parameters. Users can later view and modify these parameters via a graphical algorithm editor. At this level of integration, users must still be familiar with the applicability and functionality of the PDE solving system, but the intrinsic details of problem specification are completely hidden from them.

The *native data structures* of the "foreign" PDE system are integrated at the Fortran level using appropriate subroutines specified at the PDE language interface.

We now describe the frameworks of the integrated "foreign" PDE systems at the PELLPACK PDE language level and depict instances of their graphical user interface.

2.3.1 VECFEM FRAMEWORK

VECFEM [17] solves non-linear, time-dependent 1-D, 2-D, and 3-D systems of equations on general domains using mixed finite element methods. Framework 6 lists the segments of the VECFEM framework in the PELLPACK system. Some of the PDE problem input data for VECFEM are generated by the PDE framework specification editor (see Section 4.2.1). For VECFEM elliptic problems, this editor supports a variational template for specifying the coefficients of bi-linear and linear forms and a functional template for entering the PDE in differential form. For the stress analysis of isotropic materials, a stress template is available for entering only the elasticity modulus and Poisson's number of the material. The differential form of the PDE equations is symbolically transformed to a variational form. Figure 3 displays an instance of the PELLPACK graphical interface for VECFEM.



System Solver: VECFEM

Equation Form: Stress Template

Number of Equations: 3

Number of Boundary Patches: 6

Elasticity Module 1.93e4

Poisson Number 0.3

MAXIMA is not currently connected. [illegible]
when necessary, but if you so desire, [illegible]
the "Start" option in the "Maxima" menu [illegible]

```
tions,
        vecfem
        space dimension of system = 3
        pdes in system = 3
        number of right-hand-sides = 1

uation,
        vecfem

undary,
        patran

iple,
        vecfem &
          (displacement = .true., nodalforce = .true., &
           dirichlet = .true., outflag = .true., order=3, method=10, &
           pellpackoutflag = .true., &
           pellpackoutfile = 'pellpack.out')

bprogram,
      real function of13d (t, group, comp1, comp2, x1, x2, x, y, z,
     .                   l, lt, nlo, nk, dim, u, dudx, ut, dutdx)
      integer group, comp1, comp2, x1, x2, l, lt, nlo, nk, dim
      real t, x, y, z, u(l,nk), dudx(l, nk, dim), ut(lt, nk),
     .       dutdx(lt, nk, dim)
      real data(9,9)
      real E, NY

         93e4
         0.3

        *(1.-NY)/(1+NY)/(1-2*NY)
        *(1.-NY)/(1+NY)/(1-2*NY)
        *(1.-NY)/(1+NY)/(1-2*NY)
        /2./(1+NY)
        /2./(1+NY)
        /2./(1+NY)
      c12=E*NY/(1+NY)/(1-2*NY)
      c13=E*NY/(1+NY)/(1-2*NY)
      c23=E*NY/(1+NY)/(1-2*NY)

c     initialize the 9x9 matrix
      do 10 i1 = 1, 9
         do 20 i2 = 1, 9
            data(i1,i2) = 0.0
20       continue
10    continue
```

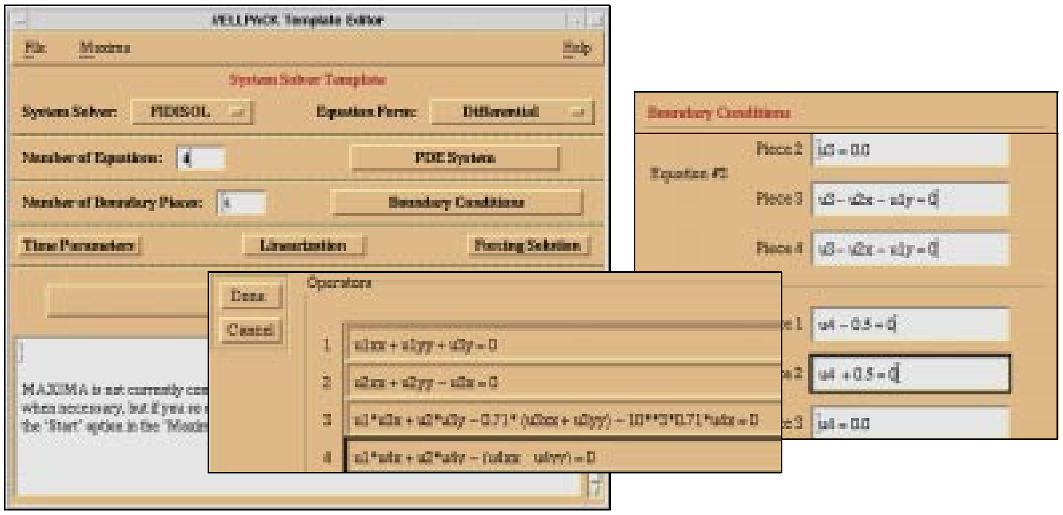
FIGURE 3. An instance of the PELLPACK interface for the VECFEM structural analysis framework

FRAMEWORK 6. VECFEM

Segment	Description of language interface
Options	VECFEM id, tags indicating the type of PDE (i.e., non-linear, parabolic), number of PDE equations in the system
Equation, BCs, IC	VECFEM tag for all equations indicating that the equations are defined by Fortran subroutines in the subprogram segment
Mesh	a triangular or tetrahedral mesh file generated by PELLPACK's native mesh generators, or a neutral file generated by a "foreign" mesh generator
Triple	VECFEM solver and associated parameters, output specification parameters
Subprograms	Fortran functions describing the PDE equations, boundary conditions, and initial conditions. These functions are interfaces to the functions used by VECFEM to describe the equations.

2.3.2 FIDISOL FRAMEWORK

FIDISOL [43] solves non-linear, time-dependent 2-D and 3-D PDE systems on rectangular domains using finite difference methods. Framework 7 describes the framework for this library. Jacobians are required for the nonlinear equations and boundary conditions; these are computed symbolically by the PDE framework specification editor. Figure 4 displays an instance of the PELLPACK interface for FIDISOL.



System Solver: FIDISOL
Equation Form: Differential
Number of Equations: 4
Number of Boundary Pieces: 4

```
1  u1xx + u1yy + u3y = 0
2  u2xx + u2yy - u3x = 0
3  u1*u3x + u2*u3y - 0.71* (u3xx + u3yy) - 10**3*0.71*u4x = 0
4  u1*u4x + u2*u4y - (u4xx  u4yy) = 0
```

Equation #3
Piece 2 u3 = 0.0
Piece 3 u3 - u2x - u1y = 0
Piece 4 u3 - u2x - u1y = 0

1 u4 - 0.5 = 0
2 u4 + 0.5 = 0
3 u4 = 0.0

MAXIMA is not currently co
when necessary, but if you w
the 'Start' option in the 'Maxi

FIGURE 4. An instance of the PELLPACK interface for the FIDISOL framework

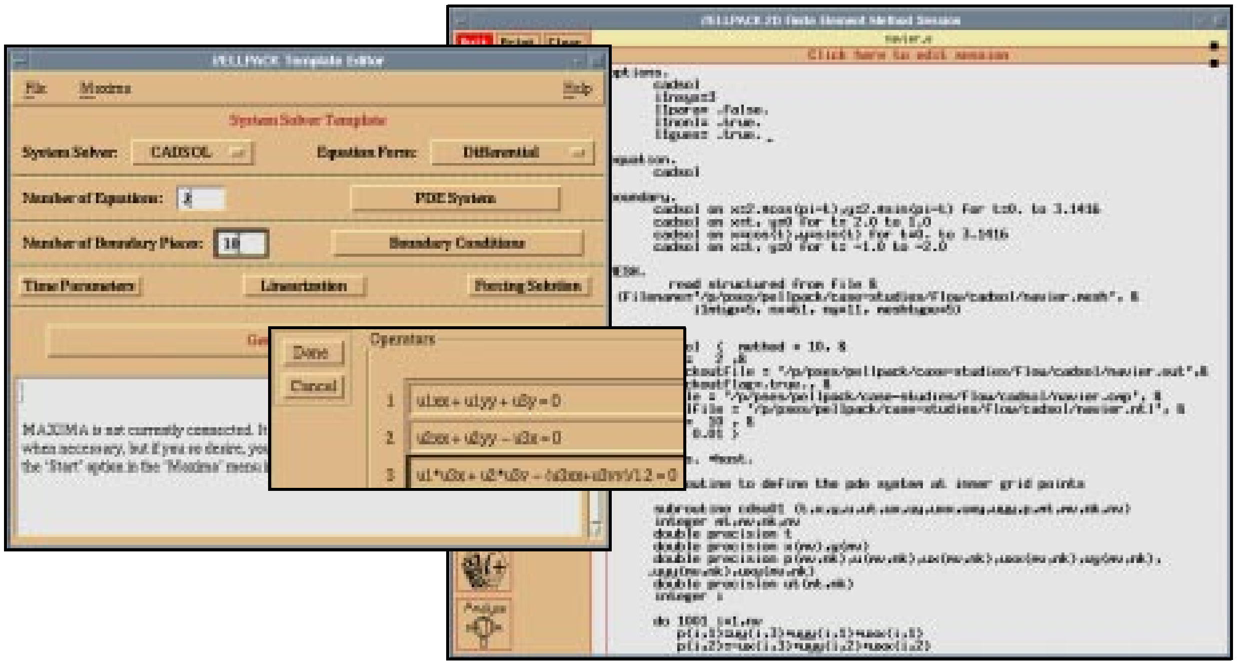
FRAMEWORK 7. FIDISOL

Segment	Description of language interface
Options	FIDISOL id, tags indicating the type of PDE (i.e., non-linear, parabolic), number of equations in the system
Equation, BCs, IC	FIDISOL tag for all equations indicating that the equations are defined by Fortran subroutines in the subprograms segment
Boundary	2-D, 3-D box geometry
Grid	Domain discretization (uniform, non-uniform grid)

Triple	FIDISOL solver and associated parameter, output specification parameters
Subprograms	Fortran functions describing the PDE equations, boundary conditions, initial conditions. These functions are identical to the functions used by FIDISOL to describe the equations. Functions describing the Jacobians for the PDE equations and boundary conditions are also placed here.

2.3.3 CADSOL FRAMEWORK

CADSOL [42] solves non-linear, time-dependent 2-D systems of equations on general domains using finite difference methods. Framework 8 describes the framework for CADSOL. The required Jacobians are computed by the PDE framework specification editor. Figure 5 displays an instance of the PELLPACK interface for CADSOL.



System Solver: CADSOL
Equation Form: Differential
Number of Equations: 3
Number of Boundary Pieces: 10

1 u1xx + u1yy + u2y = 0
2 u2xx + u2yy − u3x = 0
3 u1*u3x + u2*u3y − (u3xx+u3yy)/1.2 = 0

MAXIMA is not currently connected. It [illegible] when necessary, but if you so desire, you [illegible] the 'Start' option in the 'Maxima' menu [illegible]

```
options.
        cadsol
        iiruga=3
        llporpe .false.
        lltranls .true.
        llguess .true.

equation.
        cadsol

boundary.
        cadsol on x=2.*cos(pi-t),y=2.*sin(pi-t) for t=0. to 3.1416
        cadsol on x=t, y=0 for t= 2.0 to 1.0
        cadsol on x=cos(t),y=sin(t) for t=0. to 3.1416
        cadsol on x=t, y=0 for t= -1.0 to -2.0

mesh.
        read structured from file &
  (Filename='/p/pses/pellpack/case-studies/Flow/cadsol/navier.mesh', &
           ilhdgs=5, nx=61, ny=11, meshtype=5)

[illegible]l  (  method = 10, &
[illegible]   2 ,&
[illegible]outfile = '/p/pses/pellpack/case-studies/Flow/cadsol/navier.out',&
[illegible]outflag=.true., &
[illegible]le = '/p/pses/pellpack/case-studies/Flow/cadsol/navier.cmp', &
[illegible]File = '/p/pses/pellpack/case-studies/Flow/cadsol/navier.nl1', &
[illegible] 30 , &
[illegible] 0.01 )

[illegible]. *end.

[illegible]outine to define the pde system at inner grid points

        subroutine cdsu01 (t,x,y,u,ut,ux,uy,uxx,uxy,uyy,p,nt,nv,nk,nv)
        integer nt,nv,nk,nv
        double precision t
        double precision x(nv),y(nv)
        double precision p(nv,nk),u(nv,nk),ux(nv,nk),uxx(nv,nk),uy(nv,nk),
       ,uyy(nv,nk),uxy(nv,nk)
        double precision ut(nk,nk)
        integer i

        do 1001 i=1,nv
           p(i,1)=uyy(i,3)+uyy(i,1)+uxx(i,1)
           p(i,2)=-ux(i,3)+uyy(i,2)+uxx(i,2)
```

FIGURE 5. An instance of the PELLPACK interface for the CADSOL framework

FRAMEWORK 8. CADSOL

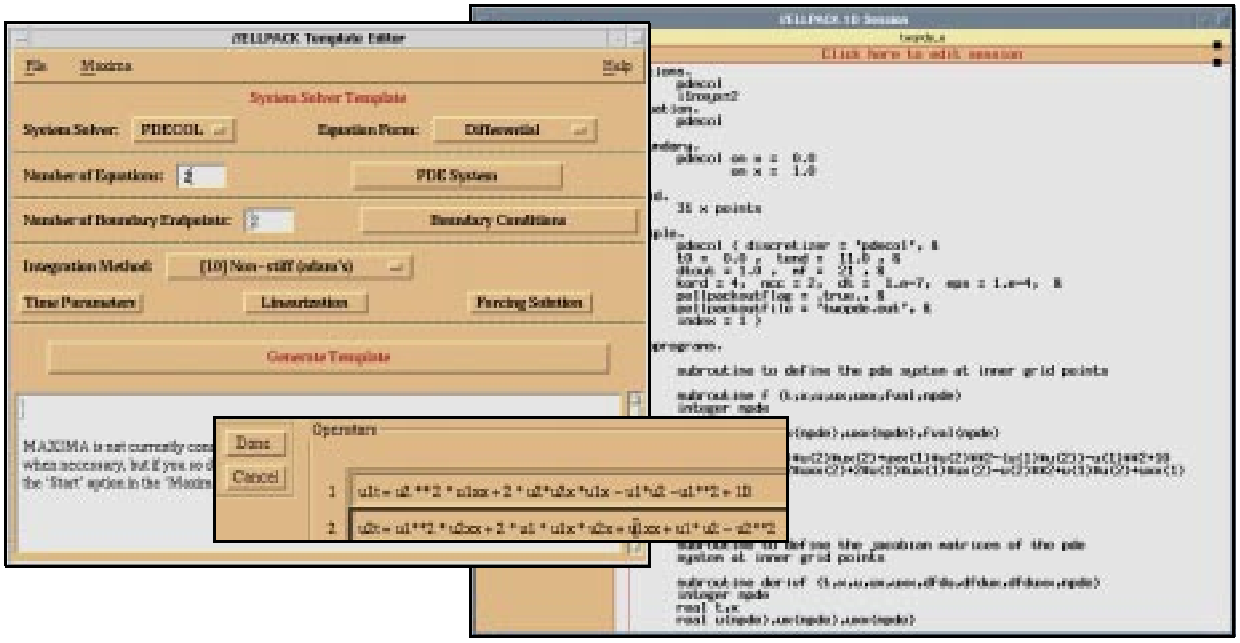
Segment	**Description of language interface**
Options	CADSOL id, tags indicating the type of PDE (non-linear, parabolic), number of equations in the system
Equation, BCs, IC	CADSOL tag for all equations indicating that the equations are defined by Fortran subroutines in the subprograms segment
Boundary	domain definition (can be specified by the PELLPACK domain editor)
Mesh or Grid	specify a body-oriented grid (can be generated by PELLPACK's structured mesh generator) or a uniform or non-uniform grid and include a user-written routine that generates the body-oriented grid in the subprogram segment.
Triple	CADSOL solver and associated parameter, output specification parameters
Subprograms	Fortran functions describing the PDE equations, boundary conditions, initial conditions. These functions are identical to the functions used by CADSOL to describe the equations. Functions describing the Jacobians for the PDE equations and boundary conditions are also placed here.

2.3.4 PDECOL Framework

PDECOL [31] solves time-dependent coupled systems of 1-D non-linear equations using the method of lines. For the space discretization a spline collocation scheme is employed. The user can select the time discretization scheme and integration method from several options. Jacobians are symbolically generated by the PDE framework specification editor when they are required for the problem definition.

FRAMEWORK 9. PDECOL

Segment	Description of language interface
Options	PDECOL id, tags indicating the type of PDE (linear, non-linear), number of equations in the system
Equation, BCs, IC	PDECOL tag for all equations indicating that the equations are defined by Fortran subroutines in the SUBPROGRAMS segment
Domain	interval endpoints defined in the PELLPACK domain editor
Grid	points in the interval are specified with the 1-D grid editor
Triple	PDECOL solver and parameter specification, output specification parameters
Subprograms	Fortran functions describing the PDE equations, boundary conditions, initial conditions. These functions are identical to the functions used by PDECOL to describe the equations. Functions describing the Jacobians for the PDE equations and boundary conditions are also placed here.



System Solver: PDECOL

Equation Form: Differential

Number of Equations: 2

Number of Boundary Endpoints: 2

Integration Method: [10] Non-stiff (adams's)

MAXIMA is not currently con[illegible]
when necessary, but if you so [illegible]
the "Start" option in the "Maxima[illegible]

1 u1t = u2**2 * u1xx + 2 * u2*u2x *u1x - u1*u2 - u1**2 + 10

2 u2t = u1**2 * u2xx + 2 * u1 * u1x * u2x + u1xx + u1*u2 - u2**2

```
ions.
    pdecol
    [illegible]
ation.
    pdecol
ndary.
    pdecol  on x =  0.0
            on x =  1.0
d.
    31 x points
ple.
    pdecol ( discretizer = 'pdecol', &
    t0 =  0.0 ,  tend =  11.0 , &
    dtout = 1.0 ,  nf =  21 , &
    kord = 4,  ncc = 2,  dt =  1.e-7,  eps = 1.e-4,  &
    pellpackoutflag = .true., &
    pellpackoutfile = 'twopde.out', &
    index = 1 )
programs.
    subroutine to define the pde system at inner grid points
    subroutine f (t,x,u,ux,uxx,fval,npde)
    integer npde
[illegible](npde),uxx(npde),fval(npde)
[illegible]
[illegible]
    subroutine to define the jacobian matrices of the pde
    system at inner grid points
    subroutine derivf (t,x,u,ux,uxx,dfdu,dfdux,dfduxx,npde)
    integer npde
    real t,x
    real u(npde),ux(npde),uxx(npde)
```

FIGURE 6. An instance of the PELLPACK interface for the PDECOL framework

2.4 Templates for "Foreign" PDE Systems

There are PDE systems whose mathematical model and numerical solver is specified through a set of physical and numerical parameters (usually numerical data). These systems are usually associated with flow problems. In these cases the PELLPACK interface consists of a hierarchical set of templates corresponding to various models the "foreign" system supports. In general, these solvers do not require symbolic processing or Fortran code generation. Three such solvers (NPARC3-D, ITGFS, NSC2KE) have been integrated into PELLPACK. NPARC3-D is a general purpose

CFD simulator for three dimensional fluid problems. ITGFS and NSC2KE are two turbulence solvers for 2-D problems. ITGFS is only applicable for internal flows, however it is expected to be more efficient than the others.

2.4.1 NPARC3-D TEMPLATE

NPARC3-D [10] is a general purpose CFD simulator, which can be used for most gas flow computations, such as 2-D axisymmetric, or 3-D for states of inviscid, laminar, or turbulent, and steady or transient with complex geometry flow.

The original NPARC system requires the fluid problems to be defined through the NPARC standard input text-file and the initial solution file. This case can involve very tedious work, especially for complex geometries. NPARC provides some utility tools that assist the user in the pre-processing phase. In addition, the original solver must be recompiled when the mesh sizes changes. We have created PELLPACK templates for the NPARC system that support a graphical user interface to allow direct access to the NPARC utilities for redefinition of global parameters, including memory allocation options. The memory space for the solver is automatically allocated without recompiling the NPARC library. Further work is necessary for this solver to fully utilize the pre- and post-processing capability of the PELLPACK environment. Template 1 depicts the items of the NPARC template.

TEMPLATE 1. NPARC3-D

Segment	**Description of language interface**
Options	NPARC id
Equation	NPARC tag indicate model specific equations
Domain, BC	NPARC tag indicates model specific boundary conditions
Mesh	uses blocked structured meshes specified in PLOT3D or GRIDGEN format [10], and an initial NPARC solution file in binary format
Triple	NPARC solver and associated parameter, output specification parameters

2.4.2 ITGFS TEMPLATE

The internal turbulence gas-flow solver ITGFS [57] is designed for the simulations of transonic turbulence flow in an internal flow field. The equations governing the flow consist of two-dimensional, compressible, time-dependent, Reynolds averaged Navier-Stokes equations, supplemented by an equation of state together with the constant total temperature assumption. Template 2 describes the items of this template.

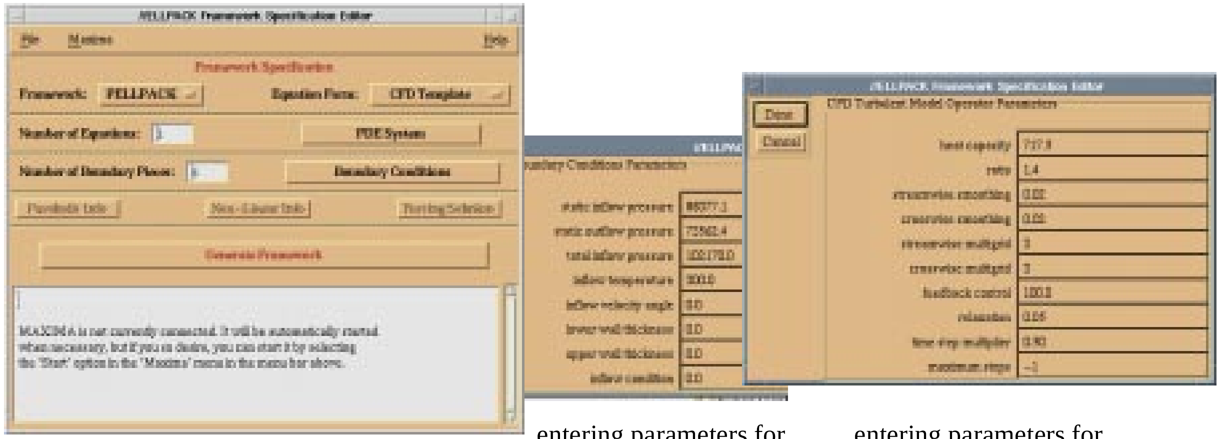
TEMPLATE 2. ITGFS

Segment	**Description of language interface**
Options	ITGFS id
Equation	ITGFS tag identifies model specific equations
Domain, BC	specified graphically by the 2-D domain editor or textually by boundary parametrization; boundary conditions are model-specific tags: inflow, outflow, wall
Mesh	generated by PELLPACK’s structured mesh generator
Triple	ITGFS-turbulent solver and associated parameters, output specification parameters

We now use the PELLPACK problem solving environment to solve a separated, transonic diffuser flow problem. We will illustrate how each PELLPACK subsystem is used in the solution process, and indicate how the components of Figure 1 are used.

The user scenario within the *PDE Problem Specification Subsystem* is depicted in Figure 7 using snapshots from the PELLPACK system along with a brief commentary for each of the editors.

The PDE Framework Specification Editor



Framework: PELLPACK
Equation Form: CFD Template
Number of Equations: 1
Number of Boundary Pieces: 6

MAXIMA is not currently connected. It will be automatically started when necessary, but if you so desire, you can start it by selecting the 'Start' option in the 'Maxima' menu in the menu bar above.

Boundary Conditions Parameters

static inflow pressure	86377.1
static outflow pressure	73562.4
total inflow pressure	102170.0
inflow temperature	300.0
inflow velocity angle	0.0
lower wall thickness	0.0
upper wall thickness	0.0
inflow condition	0.0

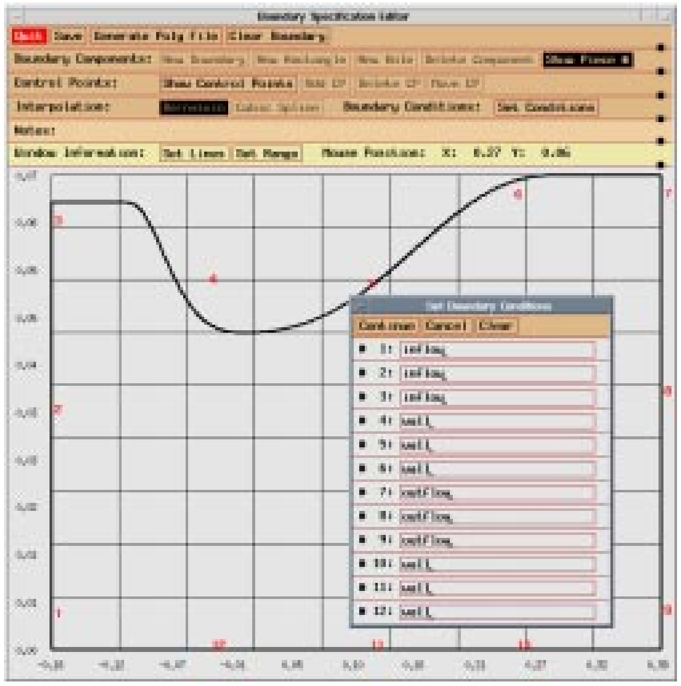
CFD Turbulent Model Operator Parameters

heat capacity	717.8
ratio	1.4
streamwise smoothing	0.02
crosswise smoothing	0.01
streamwise multigrid	3
crosswise multigrid	3
feedback control	100.0
relaxation	0.05
time step multiplier	0.90
maximum steps	-1

selecting the CFD Template for ITGFS

entering parameters for the boundary conditions

entering parameters for governing equations



1:	inflow
2:	inflow
3:	inflow
4:	wall
5:	wall
6:	wall
7:	outflow
8:	outflow
9:	outflow
10:	wall
11:	wall
12:	wall

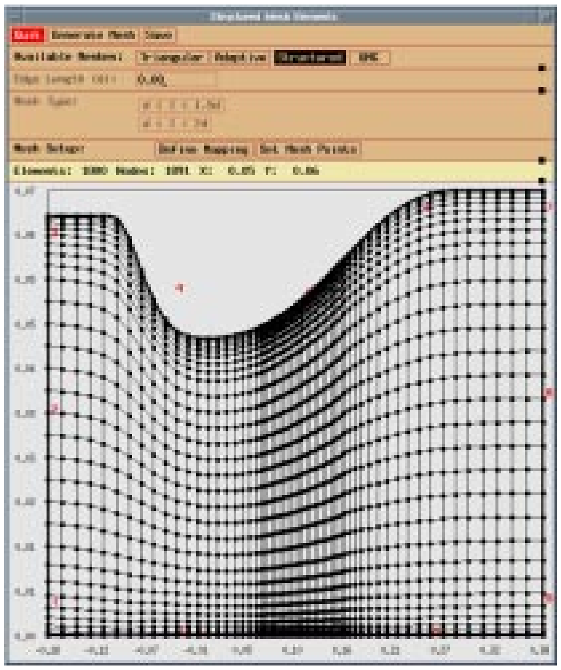
The 2D Geometry Editor and the Boundary Conditions Editor

The domain can be drawn with the Geometry Editor, or the boundary can be parameterized by the user and dynamically loaded into the editor. Note that the upper and lower wall of the boundary have been divided into 3 pieces. This allows the specification of a varying grid density across the domain.

Specialized boundary conditions such as inflow, outflow, wall, and slipping are recognized within this framework, and can be assigned to the boundary pieces.

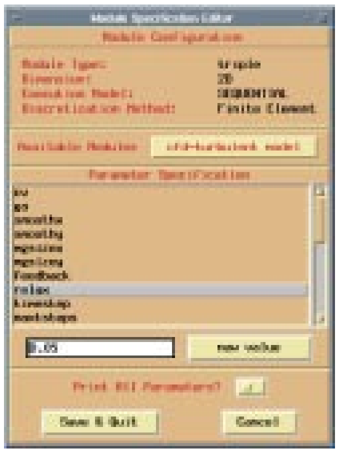
FIGURE 7. PDE Problem Specification

The user scenario within the *PDE Solution Specification Subsystem* is illustrated in Figure 8. Since we have already specified the PDE solver library via the framework selection, we need only to generate the appropriate domain discretization and specify the solver parameters.



The 2D Structured Grid Generator

The ITGFS solver requires a structured grid. We can define our own mapping of a general boundary to a rectangle or let the system determine one. We can also specify the number of grid points to use at each boundary piece so that the solution is more accurately computed.

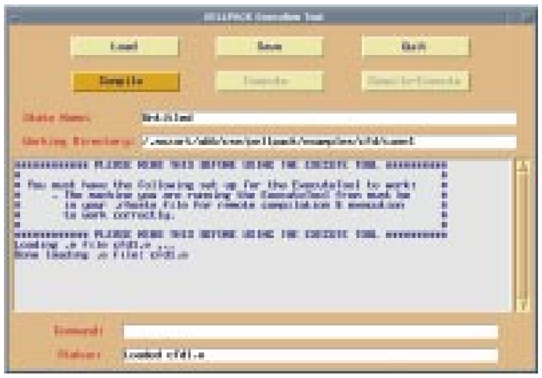


The Algorithm Editor

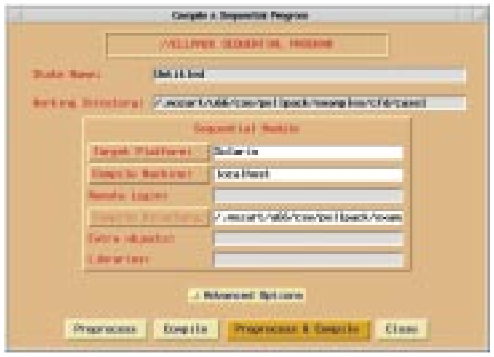
The parameters of the ITGFS solver can be displayed or modified via this editor.

FIGURE 8. PDE Solution Specification

A PELLPACK language description of this PDE problem (.e file) is generated by the editors and written to the PELLPACK session. The language processor within the *Execution Environment Subsystem* converts the ".e file" to a Fortran driver program. The driver is linked with the PELLPACK CFD libraries, and then executed. Below are snapshots from the Execution Environment.



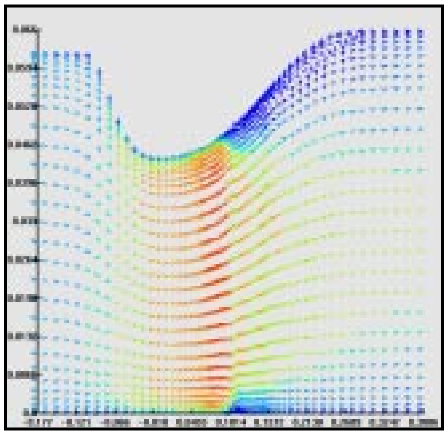
ExecuteTool accesses
Language Processor and PDE Libraries



Compile window
for Target Platform selection

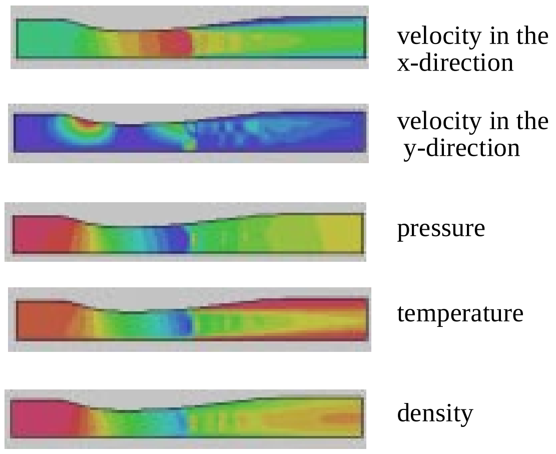
FIGURE 9. Execution Environment

PELLPACK format output is generated during execution, and can be loaded into the OutputTool within the *Post-processing Subsystem* for solution visualization. Figure 10 contains snapshots from several visualizers available from the OutputTool.



2-D Flow Plot

Vector plot of velocities in the x and y directions. Turbulence occurs in the mid upper and lower walls.



2-D Contour Plots

FIGURE 10. Post-processing Environment

2.4.3 NSC2KE TEMPLATE

NSC2KE [3] is a 2-D axisymmetric fluid flow solver applied on unstructured meshes. It solves the Euler equations using a Roe, Osher, and a Kinetic solvers and the Navier-Stokes equations using a *k-epsilon* method with two approaches of wall-laws and a two-layer model of the near wall turbulence. Template 3 describes the items of this template.

TEMPLATE 3. NSC2KE

Segment	Description of language interface
Options	NSC2KE id
Equation	NSC2KE tag identifies model specific equations
Domain, BC	specified graphically by the 2-D domain editor or textually by boundary parametrization; boundary conditions are model-specific tags: inflow, outflow, wall
Mesh	generated by PELLPACK’s structured mesh generator
Triple	NSC2KE solver and associated parameters, output specification parameters

3. SOFTWARE ARCHITECTURE

In this section, we present the architecture of PELLPACK in terms of (i) the level of programming supported, (ii) the software subsystems involved, and (iii) the software layers used to implement PELLPACK.

3.1 THE PROGRAMMING VIEW

In order to realize the PELLPACK computational environment, we have adopted three levels of programming with standardized data structures and interfaces among the various PDE objects involved in the solution process. At the highest level, the graphical user interface provides application users with knowledge-based, object-oriented editors to define problem components, specify the solution process and perform various post-processing analyses. The problem and solution specifications are expressed in terms of a high level PDE language, which is used to represent the PDE objects produced by the graphical editors. At the second level, the PELLPACK language processor compiles this high level problem and solution specification into a procedural driver program. In the third level, the driver program invokes various library modules to realize the user's solution process. These three programming levels are illustrated in Figure 11.

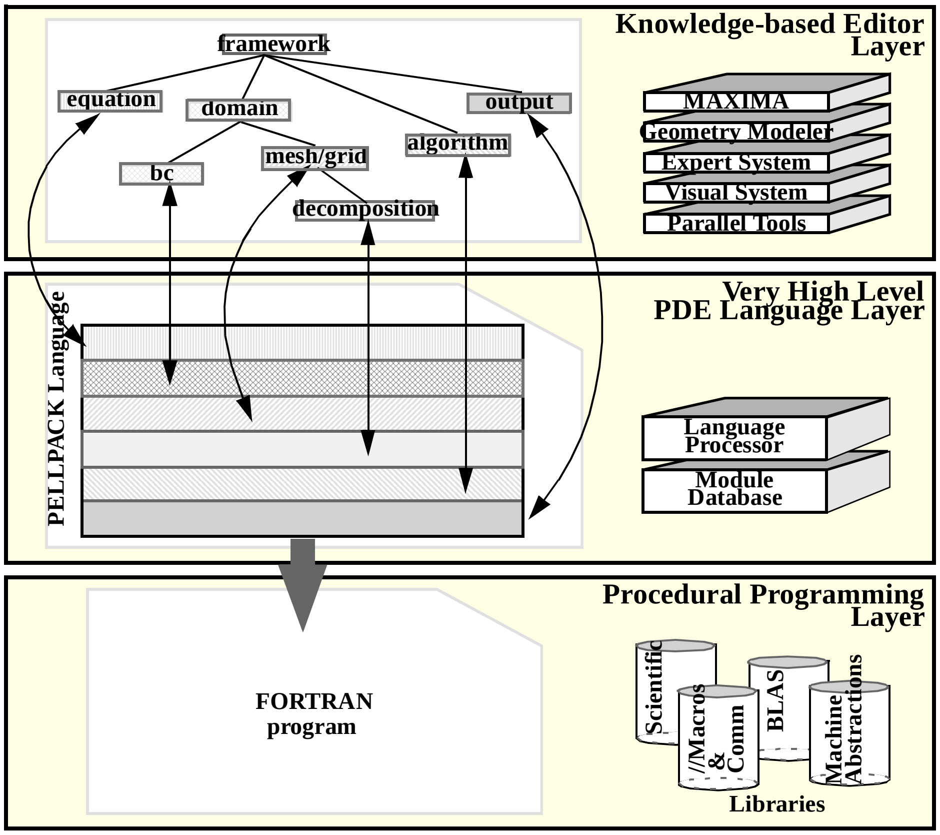


FIGURE 11. Three level programming view of PELLPACK

TABLE 2. PELLPACK Subsystems

Subsystems	Components
PDE problem specification	Editors, foreign templates, PDE language and embedded fortran
PDE solution specification	Editors, foreign templates, PDE language and embedded Fortran
Execution environment	Language processor: Solver module database, program templates and preprocessor
	PDE libraries: ELLPACK, PELLPACK, foreign solvers
	ExecuteTool: Target platform properties database, libraries and editor
Post-processing environment	Visualization tools, performance analyzers and editor

3.2 THE SUBSYSTEM VIEW

The functionality of PELLPACK is organized into four subsystems. These subsystems represent the solution process that application users follow. The *PDE Problem Specification Subsystem*, *PDE Solution Specification Subsystem* and *Post-processing Environment Subsystem* provide users with graphical editors, "foreign" system templates, the PELL-PACK language and a facility for embedding Fortran code. The *Execution Environment Subsystem* provides a framework for processing, compiling, and executing PELLPACK programs. It consists of a language processor, PDE libraries, and the ExecuteTool. The *language processor* uses the high level PDE language specification produced by the graphical editors of the problem and solution specification subsystems to generate a driver program. It can also be used to integrate new PDE solver components to the PELLPACK system.The *PDE libraries* implement sequential and parallel solver components that are available to users via the solution specification subsystem. They include the ELLPACK solver library, the PELLPACK solver library and "foreign" solver libraries such as FIDISOL, VECFEM, PDECOL and PDEONE. The *ExecuteTool* helps users compile and execute programs on all the hardware and software platforms that PELLPACK supports by managing the complexities associated with sequential and multi-platform parallel execution Table 2 summarizes the subsystems. This subsystem view of PELLPACK is illustrated by the **vertical layers** of Figure 12. Contained in each vertical layer are the PELLPACK programs and libraries that support the subsystem represented by that layer.

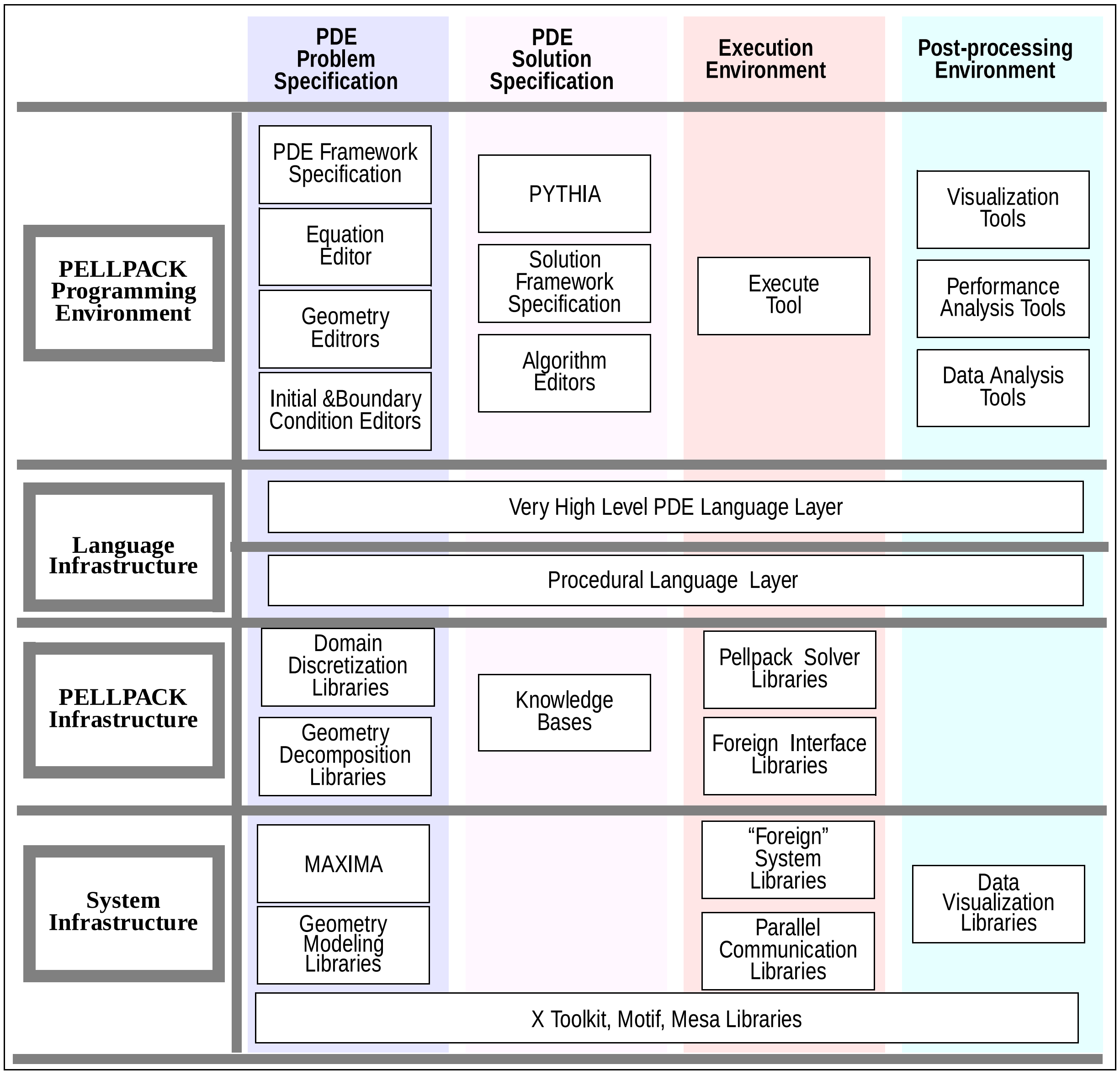

PDE Problem Specification
PDE Solution Specification
Execution Environment
Post-processing Environment
PELLPACK Programming Environment
PDE Framework Specification
Equation Editor
Geometry Editrors
Initial &Boundary Condition Editors
PYTHIA
Solution Framework Specification
Algorithm Editors
Execute Tool
Visualization Tools
Performance Analysis Tools
Data Analysis Tools
Language Infrastructure
Very High Level PDE Language Layer
Procedural Language Layer
PELLPACK Infrastructure
Domain Discretization Libraries
Geometry Decomposition Libraries
Knowledge Bases
Pellpack Solver Libraries
Foreign Interface Libraries
System Infrastructure
MAXIMA
Geometry Modeling Libraries
"Foreign" System Libraries
Parallel Communication Libraries
Data Visualization Libraries
X Toolkit, Motif, Mesa Libraries

FIGURE 12. The subsystem (vertical) view and the software layered (horizontal) view of PELLPACK

3.3 THE SOFTWARE LAYERED VIEW

The software is implemented in five layers: the *Programming Environment* layer, the PELLPACK *Very High Level Language* (VHLL) layer, the *Procedural Language* (Fortran) layer, the *PELLPACK Infrastructure* layer and the *System Infrastructure* layer. This view of the PELLPACK architecture and the specific programs and libraries contained in each layer are illustrated in Figure 12. Notice that the Language Infrastructure layer consists of two software layers supporting the VHLL layer and the procedural (code generation) language layer. Figure 12 shows the software view as **horizontal layers** which span the (vertical) subsystem layers. This figure illustrates how the PELLPACK architecture can be viewed from the standpoint of functionality and from the standpoint of system design.

The implementation language and code size for each software layer are listed in Table 3. Table entries for the System Infrastructure layer do not include generic system utilities such as X, Motif, etc.

TABLE 3. PELLPACK software layers, implementation languages, and lines of code

Layer	Implementation language	Lines of Code
Programming environment (Graphical user interface)	C, C++, Tcl/Tk, Perl, lisp, mac, flex, bison	172,000
Language infrastructure: Very high level language interface	Fortran, custom parser generator (tp, pg)	80,000
Language infrastructure: Procedural language (Fortran) interface		
PELLPACK infrastructure: PELLPACK and "foreign" system interface libraries	Fortran, C	175,000
System infrastructure: MAXIMA, "foreign" PDE libraries, parallel communication libraries, visualization libraries/tools.	Fortran, C,C++, lisp, mac	1,500,000

The next four subsections discuss the architecture of the top four software layers in more detail.

3.3.1 PROGRAMMING ENVIRONMENT (GRAPHICAL USER INTERFACE)

The GUI of PELLPACK serves two main purposes: PELLPACK program building and solution/performance visualization/analysis. The GUI supports multiple problem *sessions* within the same process. Each session represents a single problem to be solved. The tools that are made available to the user within a session are dependent on the type of session: 1-D, 2-D, 3-D and finite difference / finite element. Different tools support a different part of the problem specification or the solution specification. As the problem and solution are being defined, the session editor reflects the current status by displaying the specification in the PELLPACK language. The user may choose to edit the language directly as well, but in order to maintain consistency the user must not be running any of the graphical tools at the same time. For solution and performance visualization and analysis, the user specifies where to save the appropriate data at problem specification time and the visualization environment loads this data at postprocessing time to visualize the results.

While the graphical tools are active, the current PELLPACK program is internally represented in a set of data structures in a parsed form. In addition, it is textually represented within the session editor for the user's benefit and comfort. Each tool manipulates one or more pieces of this data structure and is responsible for leaving them in a consistent state. In some cases, a tool is actually a separate process. Then, the appropriate data structures are communicated to the other process via inter-process communication and made consistent when the changes are "committed." The tools also have a dependency relationship; for example, the mesh tool cannot be invoked until a domain has been specified. This is supported by having the tools themselves be aware of their position in the chain of operation and having them do the appropriate tests to ensure that the proper order is maintained.

3.3.2 VERY HIGH LEVEL LANGUAGE INTERFACE

The PELLPACK language interface gives full flexibility to the user to specify their PDE problems and solution framework using a convenient, high level PDE-specific language. The language uses *segments* based on the natural components of the PDE problem and the solution process. The user may write a program in this language directly or use the graphical user interface to automatically generate the program.

The language processor translates the PELLPACK program into a FORTRAN *control program* that invokes the appropriate library components to solve the problem according to the user's specifications. Each problem component is transformed into the PELLPACK standard representation for it. Each solution step is converted to the call(s) to the appropriate solver library using the standard interfaces described earlier. The resulting control program is then linked with the appropriate libraries to build the program for solving the problem. If the problem is to be solved in parallel, then there may be more than one control program based on the model of execution selected (see Section 5).

In order for the language processor to be able to generate the control program, information about the top-level calls for each library module must be given to the system at the time a library is integrated into the system. In addition, memory requirements of the module must be explicitly stated here so that the control program can allocate memory before calling the module. This information is kept in a module database and looked up at language translation time.

3.3.3 PROCEDURAL LANGUAGE (FORTRAN) INTERFACE

The Fortran interface of PELLPACK is defined based on a decomposition of the PDE problem and the solution framework into their constituent parts: domain, interior equation, boundary conditions and initial conditions for the problem, domain discretization, domain decomposition, operator discretization and algebraic system solution for the solution framework. Each problem part is represented at run-time using a set of standard data structures and/or functions. Each solution framework part (e.g., an operator discretizer) uses a set of well-defined data structures (interfaces) and/or functions for input and output. In addition, each such part may use and/or set certain global conditions which imply some properties about that part of the problem at hand. For example, if an operator discretizer notices that the resulting matrix is symmetric, it may set the "matrix is symmetric" flag and then use a more efficient data structure for storing the matrix. Solvers are expected to first check the symmetricity flag and then select the appropriate data structures. These definitions extend those of the ELLPACK system [40].

3.4 PELLPACK INFRASTRUCTURE

From a run-time view of the architecture the PELLPACK infrastructure, consisting mainly of PDE system libraries, is below the Fortran interface (Figure 12), but the libraries themselves are integrated to the PELLPACK system based on their compliance with the component interfaces. The entire PELLPACK collection of solvers is composed in this manner, i.e., there is no intrinsic or built-in set of libraries. Some libraries (most notably, the ELLPACK libraries of sequential solvers and the PELLPACK libraries of parallel solvers) natively support the PELLPACK component interface standards and hence can be "plugged-in" to the system immediately. However, many other libraries (for example, VECFEM, FIDISOL, PDECOL and MGGHAT) use their own interfaces and representations internally. To integrate such libraries, one must develop an interface library that transforms the PELLPACK representations produced by higher levels of the system to the internal representations assumed by the solver library. It is important to note that due to the structured nature of PDE components and PDE solution frameworks, this is a feasible task; we have so far not encountered any solver library that could not be integrated in PELLPACK in this manner.

The result of this integrated framework is that components from different libraries can easily be mixed-and-matched to form interesting and powerful PDE solvers. There is no doubt a performance cost with having a layer of software that allows this flexibility, but it should be clear that the advantages of having standard interfaces to widely differing software packages easily outweighs the cost.

4. THE PELLPACK PROGRAMMING ENVIRONMENT

The implementation of PDE *frameworks* in PELLPACK provides a three level programming environment depicted in Figure 11. In this section we give a brief description of PELLPACK programming-in-the-large environment starting with the PDE language.

4.1 VERY HIGH LEVEL PDE LANGUAGE

In the PELLPACK problem solving environment, a PDE problem is defined in terms of the PDE objects involved: PDE equations, domain of definition, boundary and initial conditions, solution strategy, output requirements, and option parameters. The textual representation of the PDE objects and its syntax comprise the PELLPACK language, which is a significant extension of the ELLPACK language defined in [40]. This language layer is the foundation of the PELLPACK environment and underlies all levels and components. It defines the intrinsic objects which are needed to specify a PDE problem and its solution strategy. It is parsed and generated by special editors, and it is loaded by the execution environment and processed by the language translator into Fortran control program(s) which are compiled and executed. All PELLPACK system functionality is represented in some way by the PDE language.

In the ELLPACK language, the PDE objects are defined by language segments which either specify the PDE problem (`equation`, `boundary` and associated boundary and initial conditions) or name the module to be used in the solution process (`grid`, `discretization`, `indexing`, `solution`, `triple`, `output`). To support the insertion of arbitrary Fortran code for control and assignment statements, the ELLPACK language uses the `declarations`, `global`, `procedure`, and `Fortran` segments. The number and types of segments and modules which have been added to the original ELLPACK have greatly increased the types of problems that can be solved and the methods for solving them. The extensions to the ELLPACK language which were defined by PELLPACK follow.

PELLPACK introduced a `mesh` segment to support solution schemes using finite element methods. The integration of "foreign" solvers required the introduction of tags and specialized identifiers in the `option` segment for relaying information about the system solver and its interface requirements to the language processor. The `triple` segment is the standard which was adopted to specify the numerical solver associated with a foreign system. The `save` segment allows persistent storage of solutions, linear system matrices, and performance data for visualization and/or animation. Finally, the ELLPACK language and system was extended to support a domain decomposition strategy [21] to solve PDE problems in parallel on multicomputer platforms. Specifically, the `decomposition` and `parallelsolution` segments define the geometry partitioning of the discrete domain and handle the assembly of the partitioned solutions from the parallel processors. The existence of several parallel execution models necessitates the use of tags in the `option` segment (i.e., hostless, Mplus) to identify the parallel model selection.

The language definition of existing segments, modules and module parameters was amplified to contain information related to the graphical environment. In this way, the language, the graphical interface, and the execution layers work smoothly together to provide a unified PDE problem solving environment.

4.2 PDE OBJECT BASED GRAPHICAL USER INTERFACE

The process of specifying, solving, and analyzing a PDE problem occurs within a PELLPACK session editor This editor consists of a text window and an attached toolbox of editors. Figure 13 displays an instance of this editor. The toolbox editors are used to create or modify the PDE objects which specify the PDE problem and describe how to solve it. Each toolbox editor is a graphical, interactive tool that generates a textual representation of the object and the associated PELLPACK language segment in the main session editor window. Editors in the toolbox are able to reload a PDE object by reading its PELLPACK language representation, and then displaying the object for viewing or modification. PDE objects are communicated between editors or between an editor and the PELLPACK session editor. Moreover, these editors may transform objects when needed. For example, domain objects are transported to mesh editors, where any generator requiring a piecewise-linearization of the boundary will transform the domain object appropriately. Table 4 lists the editors and their design objective in PELLPACK.

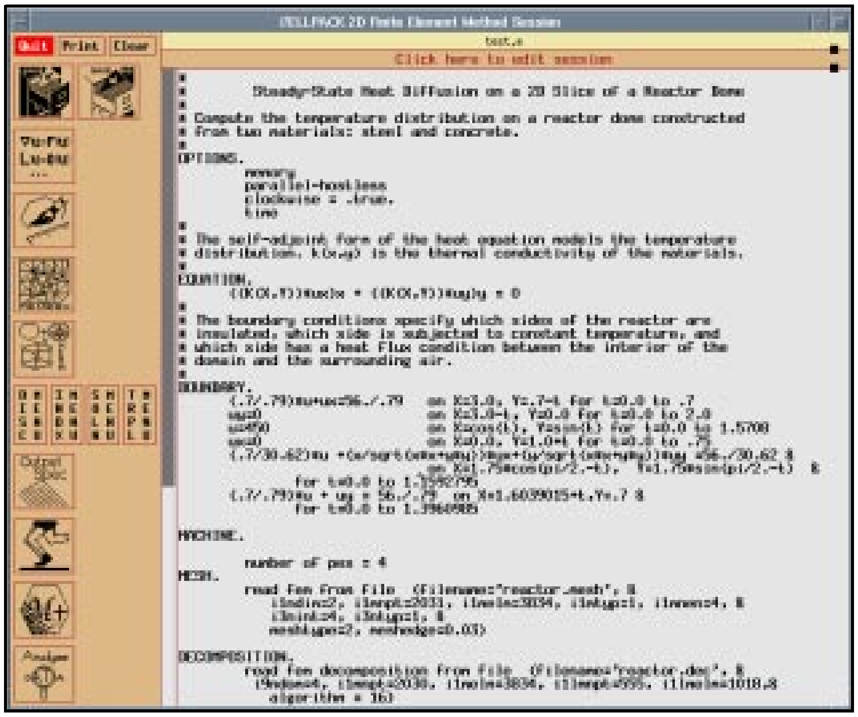
TABLE 4. The PDE object based editors in PELLPACK

Editor	**Design objective**
PELLPACK session	editable textual representation of the PELLPACK problem and solution specification
PDE framework specification	symbolic PDE operators definition, input functions transformation in Fortran, linearization, Jacobian, and default framework generation for each PDE library
Domain and boundary conditions	CAD tools for 1-D, 2-D, 3-D domain boundary specification and auxiliary conditions, or file in some standard format
Mesh generators	2-D, 3-D mesh generators using PELLPACK domain (or other standard format) as input.
Grid generators	1-D, 2-D, 3-D uniform/non-uniform grid generators
Domain decomposers	2-D, 3-D geometry decomposition using a library of partitioning heuristics
Discretizers Linear system solvers Triples / Foreign system solvers	algorithm specification, where choices for the solution scheme are controlled by a knowledge base to provide numerical method modules from the data base (using dimension, domain discretization, sequential vs. parallel, etc.)
Output specification	solution or performance data output format specification
Output visualization	visualization and animation of all possible output data produced by solution (solutions, error analysis, performance data), including necessary data conversion when accessing “integrated” visualizers

4.2.1 PDE FRAMEWORK SPECIFICATION EDITOR

This editor is used to specify the PDE equations and generate the program framework used for solving the PDE problem. The format of the framework generated depends upon the PDE-solving system selected by the user. For each of these systems, certain forms of the equation are valid. For example, PELLPACK solvers allow differential and self-adjoint forms of the equation; VECFEM allows differential and variational forms. PDE equations are specified via a graphical interface which has been tailored to the representation of the chosen form of the equation. The editor then performs the specialized symbolic processing and code generation required for the definition of PDE problems in the format required by the selected system solver. It generates by default a boundary segment for a rectangular region with Dirichlet boundary conditions and zero initial condition. Toolbox editors are used to define the actual values of these PDE objects.

To implement the PELLPACK framework generation, the PDE system is sent in string form to the Maxima computer algebra system. Depending on the framework, different symbolic transformations are performed. If non-linear equations are entered for the PELLPACK system solver, these equations are linearized by computing their Frechet derivatives. If FIDISOL or CADSOL is the selected system solver, Jacobians are computed for the specified system of equations and boundary conditions. A symbolic representation of the PELLPACK template is then developed inside Maxima. This representation is communicated to the PDE framework specification editor which converts it to a PELLPACK template using the GENCRAY system [49]. Finally, this template is written in the PELLPACK session window. All symbolic operations of this editor are provided by Maxima [12].


PELLPACK 2D Finite Element Method Session
Quit
Print
Clear
test.e
Click here to edit session

```
*
*        Steady-State Heat Diffusion on a 2D Slice of a Reactor Dome
*
* Compute the temperature distribution on a reactor dome constructed
* from two materials: steel and concrete.
*
OPTIONS.
        memory
        parallel-hostless
        clockwise = .true.
        time
*
* The self-adjoint form of the heat equation models the temperature
* distribution. k(x,y) is the thermal conductivity of the materials.
*
EQUATION.
      ((K(X,Y))*ux)x + ((K(X,Y))*uy)y = 0
*
* The boundary conditions specify which sides of the reactor are
* insulated, which side is subjected to constant temperature, and
* which side has a heat flux condition between the interior of the
* domain and the surrounding air.
*
BOUNDARY.
      (.7/.79)*u+ux=56./.79    on X=3.0, Y=.7-t for t=0.0 to .7
      uy=0                      on X=3.0-t, Y=0.0 for t=0.0 to 2.0
      u=450                     on X=cos(t), Y=sin(t) for t=0.0 to 1.5708
      ux=0                      on X=0.0, Y=1.0+t for t=0.0 to .75
      (.7/30.62)*u +(x/sqrt(x*x+y*y))*ux+(y/sqrt(x*x+y*y))*uy =56./30.62 &
                                on X=1.75*cos(pi/2.-t),  Y=1.75*sin(pi/2.-t)  &
           for t=0.0 to 1.3592796
      (.7/.79)*u + uy = 56./.79   on X=1.6039015+t,Y=.7 &
           for t=0.0 to 1.3960985

MACHINE.

         number of pes = 4
MESH.
         read fem from file  (filename='reactor.mesh', &
            i1ndim=2, i1enpt=2031, i1nelm=3834, i1ntyp=1, i1nnen=4, &
            i3nind=4, i3ntyp=1, &
           meshtype=2, meshedge=0.03)

DECOMPOSITION.
         read fem decomposition from file  (filename='reactor.dec', &
          i9ndom=4, i1enpt=2030, i1nelm=3834, i1lenpt=595, i1lnelm=1018,&
           algorithm = 16)
```

FIGURE 13. An instance of the PELLPACK session editor

In addition to the general PDE frameworks generated by this editor, there are several model specific templates that are supported. In these cases, users enter the crucial pieces of information that define the problem parameters. These templates are implemented without support from a computer algebra system. User input is inserted in the appropriate locations in the template for the selected model, and the result is written in the PELLPACK session window.

4.2.2 DOMAIN AND BOUNDARY CONDITIONS EDITORS

These editors are used to generate the boundary segment and define the PDE domain and boundary conditions. For 1-D and 2-D domains, PELLPACK provides its own geometry editors. For 3-D cases, PELLPACK supports well established geometry interfaces and the XXoX CAD editor [55] for the geometry modeling library XoX Shapes [46].

With the 1-D domain editor, users can define the interval endpoints and assign a boundary condition to each endpoint. Boundary conditions for 1-D problems which are solved by PDECOL use a foreign system tag to identify the boundary equations in the boundary segment, since the equations are defined in the subprograms segment by Fortran routines as described in previous sections.

For 2-D domains, PELLPACK provides a 2-D drawing tool where users can draw the boundary piece by piece, and then assign boundary conditions to each piece. Users may instead define any 2-D boundary parametrically, using the session editor and following the PELLPACK language syntax. This includes using complicated Fortran functions to describes boundary pieces, holes and slits. These parameterized functions are then dynamically loaded into the domain editor so that the domain can be displayed and boundary conditions assigned. Any of these domain definitions can be used as input to the PELLPACK grid and mesh generators. Boundary conditions for foreign systems either follow the ELLPACK language syntax, or they are tags to foreign system or model-specific identifiers indicating specialized conditions such as inflow, outflow, wall, etc. All identifiers are handled by the language processor and the PELLPACK foreign system interface so that the appropriate boundary conditions are applied.

In the 3-D case, box geometries with associated boundary conditions per face can be specified using a 3-D domain editor. More complicated domains are defined using constructive solid geometry in the XXoX geometry editor.

XXoX generates surface triangulations which can be used to define the geometry for PELLPACK's 3-D mesh generators. Boundary conditions are generally applied discretely on groups of surface nodes (called patches) resulting from the mesh generating process. PELLPACK provides a 3-D boundary conditions editor which allows users to apply boundary conditions on surface patches of nodes. Tags or model-specific identifiers may be used to specify the boundary conditions for foreign system solvers.

For many solvers integrated into PELLPACK, the domain and boundary conditions may be defined outside of PELLPACK and saved in files which are then accessed by PELLPACK during the solution process. Packages such as TrueGrid [56] and Patran [2] can be used to define the domain (or subsequent mesh) and boundary conditions.

4.2.3 GRID GENERATION EDITORS

PELLPACK supports both uniform and non-uniform grid generation for 1-D, 2-D, and 3-D domains. For uniform grids, the number of grid lines in any direction can be specified. Non-uniform grids are specified by point-and-click (to add, move or delete grid lines) or by listing coordinates. Grids can be uniform in one direction and non-uniform in another. For collocation methods based on tensor product spline basis functions, the 2-D grid editor and the corresponding overlay grid to a domain can be used to generate and display the collocation meshes and points. In addition, PELLPACK supports a body-oriented grid generator. It allows users to define the mapping of an arbitrary domain to a four-sided domain, and allows the specification of an arbitrary number of grid lines per piece of the original domain definition. The body-oriented grid generator supports systems such as CADSOL, which require a body-oriented grid for the solution method.

4.2.4 MESH GENERATION EDITOR

This editor is the driver and graphical interface to the finite element mesh generators integrated in PELLPACK. The available mesh generators are listed in Table 5. For 2-D mesh generators, boundary conditions which have been defined on the original domain boundary pieces are maintained throughout the mesh generation process. Thus, the element edges on the domain boundary inherit the conditions of the original boundary piece. These meshes can be graphically modified by moving appropriate nodes. In the 3-D case, boundary conditions defined on the surface triangulations are maintained throughout the mesh generation process, so that additional faces on the surface inherit the appropriate boundary condition. A 3-D mesh editor is also available to display or modify 3-D meshes and boundary condition assignments.

TABLE 5. PELLPACK supported mesh generators and their applicability

Mesh generator	Domain definition	Description
2-D triangular	PELLPACK domain editor	for given edge length, generates a uniform, triangular mesh and outputs a PELLPACK mesh format file
2-D adaptive	piece-wise linear approximation of domain from PELLPACK domain editor	uses quadtree method to generate a first-cut mesh which users refine by point-and-click, outputs a PELLPACK mesh format file
2-D structured	arbitrary domain from PELLPACK domain editor is mapped to 4-sided figure	user specifies any number of "points" per side on original domain then structured mesh is generated, outputs a PELLPACK mesh format file
2-D QMG [37]	piece-wise linear approximation of domain from PELLPACK domain editor	user specifies maximum edge length which is used to generate a mesh using the quadtree algorithm and the mesh is refined by subsequent applications of algorithm, outputs a neutral [2] format file.

3-D Geompack [27]	surface triangulation from 3-D domain editor (e.g. XXoX)	users specify a set of parameters controlling edges, angles, etc., and a tetrahedral mesh is generated, outputs a neutral format file.
3-D QMG [36]	surface triangulation from 3-D domain editor (e.g. XXoX)	generates a tetrahedral mesh and outputs a neutral format file

4.2.5 DOMAIN DECOMPOSITION EDITOR

The decomposition editor supports the decomposition of meshes/grids into "balanced" subdomains. These data are used to parallelize the underlying PDE computations. A library of partitioning algorithms is provided to automatically generate the decomposition. These algorithms produce decompositions that balance the load among processors and minimize communication between processors. Users may choose from many automatic partioning heuristics, such as Inertia Axis, Neighborhood Search, Recursive Spectral Bisection, and others. These algorithms allow users to specify numerous input parameters which control the partitioning process. In addition, decompositions can be modified manually. The decomposition data is written to file(s) used uniformly across all supported target parallel platforms, communication libraries, and execution models (hosted, hostless, Mplus, Dplus). Extensive parallel processing performance data has been collected using the PELLPACK environment, comparing and analyzing algorithms, platforms, communication libraries, and execution models [29],[32],[33].

4.2.6 ALGORITHM AND OUTPUT SPECIFICATION EDITORS

These editors help the user to specify the solution and output segments by visualizing in a menu form the options that currently exist in various PDE libraries. The ELLPACK and PELLPACK modules which are available for the solution process depend on the problem description in the session. The problem dimension and selected method (finite difference or finite element) are used by internal filters to identify the applicable modules displayed in the discretization and triple menus of the algorithm editor. If the problem language specification has parallel information, only the parallel modules are listed in the menus. Any controlling parameters are accessible through the algorithm editor, where they can be viewed, modified and saved.

For a solution process which uses foreign system solvers, users must specify the appropriate triple module identified by the framework they selected when defining the problem (e.g., VECFEM, NSC2KE). As in the case of a PELLPACK triple, the foreign system triple module represents the entire numerical solution process. Selecting the triple module and specifying the values of the required parameters is done within the algorithm editor.

To specify output requirements, the output editor may be used for any ELLPACK or PELLPACK solution process. Foreign system output requirements are identified directly in the triple module as one of the parameters.

4.3 POST-PROCESSING TOOLS

This software layer includes the output tool which is an interactive environment used to analyze and visualize scientific data generated by PELLPACK solvers. It consists of customized and public domain visualization tools used to visualize and/or animate 1-D, 2-D, and 3-D PDE solution data. Every solver available in PELLPACK including the integrated foreign systems supports the PELLPACK output file format. In addition, some solvers generate "solution component" data files, which together with a mesh file describe the problem solution. Any of these file formats can be loaded into the output tool. Once the data is loaded, all tools that can load the data (or a transformation thereof) are made available for selection. Tool applicability is based on problem dimension, domain discretization type (grid vs. mesh), and the possibility for animation (time-dependent solution). When a visualization tool is selected, the output tool handles all conversions and data transformation required by the visualization tool.

In addition, it contains performance tools that use timing and trace files generated by sequential or parallel PELLPACK programs to evaluate the performance of pre-processing and solution modules (the algorithms) and the performance of execution models.

TABLE 6. Output tool applications and recognized input

Visualization tools	Applicability	Solutions generated by solvers
XGraph	1-D solutions 1-D time-dependent solutions	PDECOL
Time1-D	1-D time-dependent solutions	PDECOL
Visual2-D	2-D grid or mesh solutions 2-D grid or mesh time-dependent solutions (animation)	solutions generated by any 2-D solver, including PELLPACK, VECFEM, FIDISOL, CADSOL, ITGFS, NSC2KE.
Flow2-D	2-D mesh solutions 2-D body-oriented grid solutions (Vector plot visualization)	solutions generated by any 2-D mesh or body-oriented grid solver, including PELLPACK, VECFEM, CADSOL, ITGFS, NSC2KE.
Contour2-D	2-D mesh solutions 2-D body-oriented grid solutions (Contour plot visualization)	solutions generated by any 2-D mesh or body-oriented grid solver, including PELLPACK, VECFEM, CADSOL, ITGFS, NSC2KE.
MeshTV [6]	2-D and 3-D mesh solutions 2-D and 3-D body-oriented grid solutions	solutions generated by any 2-D or 3-D mesh or body-oriented grid solver, including PELLPACK, VECFEM, CADSOL, ITGFS, NSC2KE.
Visual3-D	2-D and 3-D mesh solutions 2-D body-oriented grid solutions 3-D solutions on a box geometry	solutions generated by any 2-D or 3-D mesh solver, including PELLPACK, VECFEM, ITGFS, NSC2KE.
PATRAN	2-D and 3-D mesh solutions 2-D and 3-D body-oriented grid solutions 3-D grid solutions on a box geometry	solutions generated by any 2-D or 3-D mesh solver, including PELLPACK, VECFEM, ITGFS, NSC2KE.
XDS	2-D and 3-D grid solutions on a box geometry	2-D or 3-D PELLPACK FDM solvers

All PELLPACK solvers generate timing information which identifies the elapsed CPU time used by each step of the numerical solution process. When programs are executed in parallel, timing information is generated for each processor. The timing information can be loaded into the output tool and analyzed via PELLPACK's performance visualization tool, ViPerform. Timing and trace data can also be analyzed by Pablo [38] and PATool [39]. Timing data is transformed by the output tool into the required format, and users can select any of the built-in configurations required by these performance analysis tools. ParaGraph [12] is available for parallel execution analysis when certain parallel communication libraries are used.

5. EXECUTION ENVIRONMENT

The design objective of the execution environment is to assist users in compiling, linking, and running programs produced by the different frameworks discussed earlier. In addition, the environment is responsible for locating compilers, allocating machines, running execution scripts, and scattering/gathering data to/from distributed machines. This environment is realized by the execute tool which provides support for remote login, file transfer, and platform-dependent execution management. In this section we describe the functionality and architecture of the PELLPACK execute tool.

5.1 ExecuteTool Functionality

The main task of the execute tool is to execute a PDE solving program specified in the PELLPACK language. The PELLPACK language file is first translated to one or more Fortran programs, and then compiled to binary format using the native Fortran compiler. Different types and numbers of programs are generated based on the execution model selected by the user.

5.1.1 Framework and Execution Model Determination

Special identifiers in the `options` segment of the PELLPACK source file specify the type of framework (e.g., PELLPACK, CADSOL) and the execution model [33] to be used. When the execute tool first loads the PELLPACK program, it uses this information to configure its operation appropriately. The framework determines which system solver library will be used in the linking stage. The execution model identifies whether the execution is sequential or parallel, and when execution is parallel, it specifies the type of parallel model. For sequential execution, a single Fortran program corresponding to the PELLPACK problem specification is generated. This program solves the problem and generates the global solution in a single output file.

5.1.2 Parallel Execution Models

For parallel execution, the parallel model tells the execution environment the number and types of Fortran files to be generated. In the parallel case, a partitioning of the PDE mesh/grid is assumed, and all computations are done on a per-subdomain (local) basis. Computations for each subdomain are mapped to the processors of a (virtual) parallel machine, where multiple processors compute on different parts of the domain. Communication between subdomains occurs on the subdomain interfaces, which are specified in the decomposition data. Processors are thus able to compute a local solution. To generate the global solution, the system needs to collect the local solutions and compute the global one. Different control programs are generated for each of the parallel execution models, each performing specific phases of the numerical solution process. The parallel execution models and the corresponding execution tasks supported by the PELLPACK execute tool are described in Table 7.

5.1.3 Compilation and Execution Parameters Determination

Following the principle of late binding, the user does not select the architecture on which to execute the program until after loading it into the execution environment. That is, PELLPACK allows the user to completely specify the PDE problem as well as how to solve it without having to select the specific type of hardware to be used for solving the problem. Selecting the target platform requires the selection of the hardware (e.g., Sun Sparc) as well as the version of the operating system (e.g., SunOS 4.1, Solaris 2.5). The user also selects the communication library to be used at run-time. After this information is specified, the execute tool determines the possible machines that can be used for compilation and execution based on the local configuration. If these machines need to be accessed with a different user name and/or login name, those must be specified as well. Finally, the compilation and linking steps are performed to produce the executable file(s).

TABLE 7. Parallel execution models

Execution model	**DISC**$^{(1)}$	**LSS**$^{(1)}$	**Control programs and their execution tasks**
Hosted	P	P	<u>Host program</u>: Reads in decomposition file and sends appropriate data to each of the node programs. Receives local solution data and generates global solution. <u>Node program</u>: Each node discretizes and solves the linear system on its subdomain, collaborating with neighboring subdomains. Local solutions are sent back to the host.

Hostless[2]	P	P	Node program: Each node reads its own decomposition file. It discretizes and solves the linear system on its subdomain, collaborating with neighboring subdomains. It generates a local solution file. Post-processing program: Collects local solutions and generates a global solution.
MPlus	S	P	Discretization program: Generates linear system on the entire domain. MPlus environment: Partitions linear system using decomposition file and global linear system. Node program: Each node reads its local linear system file. It solves the linear system on its subdomain, collaborating with neighboring subdomains. It generates a local solution file. Post-processing program: Collects local solutions and generates a global solution.
DPlus[2]	S	P	Node program: Each node reads its own decomposition file. It discretizes and solves the linear system on its subdomain, collaborating with neighboring subdomains. It generates a local solution file. Post-processing program: Collects local solutions and generates a global solution.

(1) DISC = Discretization phase, LSS = Linear System Solver phase.
S = the code is sequential, P = the code is parallel.

(2) The node program for the Hostless and DPlus models carry out identical tasks. Note, however, that all numerical code in the Hostless model is parallel. DPlus, on the other hand, uses sequential discretization code; each node performs a sequential discretization on its partition of the domain. In this way, available sequential discretization codes may be used for the discretization phase, while parallel codes can still be used for the more time-consuming linear system solver phase.

5.1.4 FILE AVAILABILITY

For compilation and execution on remote machines, the execute tool addresses the issue of file availability. Since the user can choose to compile and execute the program on different machines, these files might not necessarily be available on the target machines. The files considered here include the PELLPACK source file, any generated object/binary files, the files specified inside the PELLPACK program (such as mesh and decomposition files), and the output file. These files are handled differently by the execute tool. The location of the source and generated object files is known to the environment since these files are generated during the compilation phase. The location of the PELLPACK program specification files (i.e., file system paths), are specified within the PELLPACK program file. These files are needed during the execution and post-processing phases. We identify three different possibilities for each of these files: i) the file is available on the execution machine with the same pathname, ii) the file is only available on the current machine, and iii) the file is available on the remote machine with a different path name. In the first case, nothing special need be done to gain access to the input file. For the second case, a copy of the input file is temporarily generated on the remote machine. In the last case, the execute tool must map the path of the input files appropriately so that the correct name is used on each machine.

5.1.5 COMPILATION AND EXECUTION CONFIGURATION

Unlike the sequential case, the process for compiling a Fortran file into binary form for the parallel case may not be straight forward. In certain cases, cross compilers can be used that are available on certain machines. In some other cases the compilers require certain environment variables to be set before running them. The execute tool allows the local site specialist to configure this information at the time of PELLPACK installation, so that this information is known to the environment at run time and is available for user selection of platform and communication parameters.

The information needed to configure the execute tool for a user’s site consists of:

- information about the hardware platforms and communication packages available at this site, including the list of machines available to the user, the availability of compilers and cross-compilers, the environment settings needed to compile a program on a specific architecture, and the process of executing on the selected parallel architectures
- information about the availability of the foreign solver libraries that are available at the site.

This information is specified at installation time and loaded in at runtime, so that the end-user is presented with a convenient and knowledgeable graphical interface that automatically determines the site-specific configuration based on the target platform selected by the user. The diagram in Figure 14 lists the platforms and communication libraries currently supported by the PELLPACK system. Figure 15 shows how the platform configuration information is presented to the user during PELLPACK execution in a convenient and easy-to-use format.

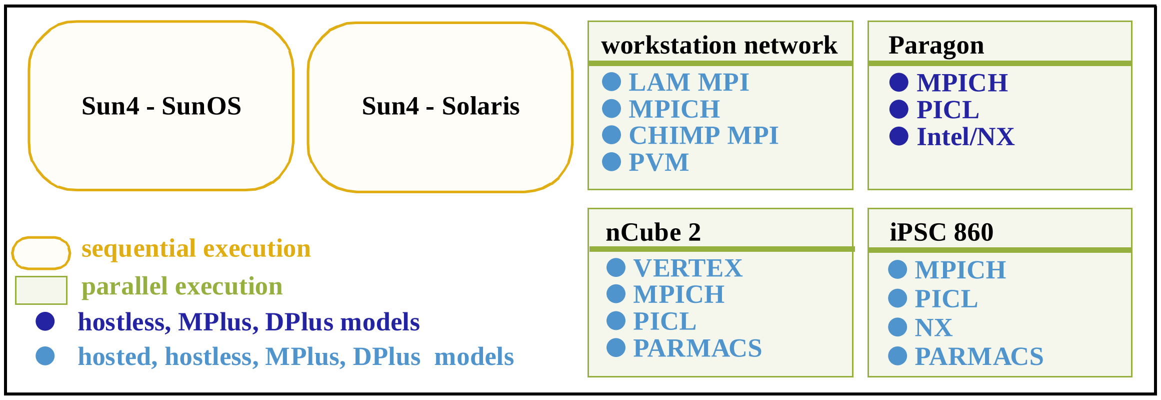
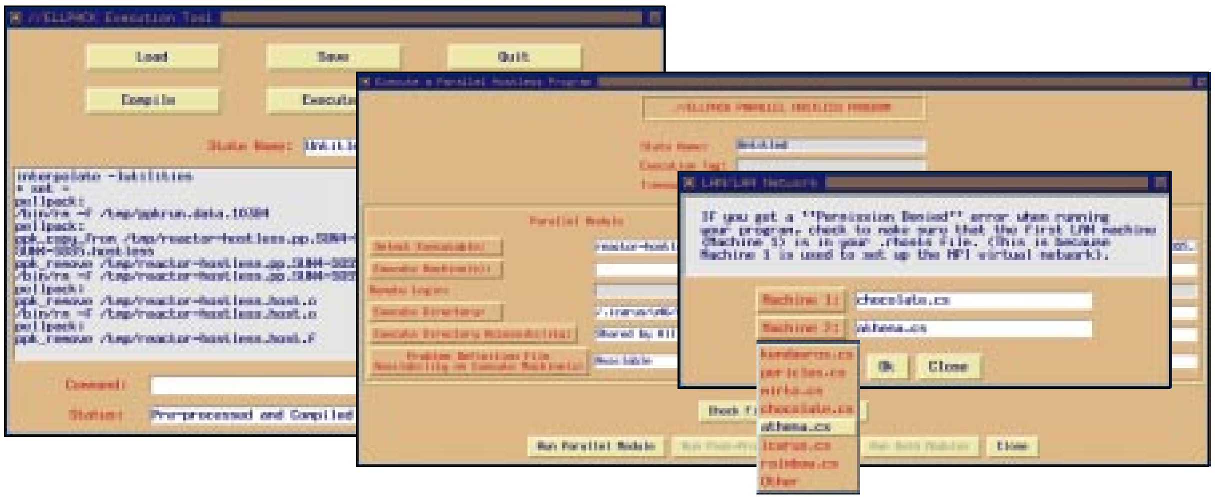


FIGURE 14. Available platforms and communication libraries

5.1.6 Execution State

Since users may wish to re-start the execution process at any point in the execution sequence, a state file is introduced to maintain state data about the actions that have been performed on PELLPACK source files. For example, if a source file is compiled for the Intel Paragon using the MPICH communication library, a record in the state file contains this information along with the names of the generated object/binary files and other pertinent information. The user may enter the execute tool at a later time with this state file, which identifies that compilation has already taken place.



State Name: Untitled
Status: Pre-processed and Compiled

```
interpolate -lutilities
+ set -
pellpack:
/bin/rm -f /tmp/ppkrun.data.10384
pellpack:
ppk_copy_from /tmp/reactor-hostless.pp.SUN4-
SUN4-SOS5.hostless
ppk_remove /tmp/reactor-hostless.pp.SUN4-SOS
/bin/rm -f /tmp/reactor-hostless.pp.SUN4-SOS
pellpack:
ppk_remove /tmp/reactor-hostless.host.o
/bin/rm -f /tmp/reactor-hostless.host.o
pellpack:
ppk_remove /tmp/reactor-hostless.host.f
```

If you get a ''Permission Denied'' error when running your program, check to make sure that the first LAM machine (Machine 1) is in your .rhosts file. (This is because Machine 1 is used to set up the MPI virtual network).

Machine 1: chocolate.cs
Machine 2: athena.cs

kanaberos.cs
pericles.cs
mirto.cs
chocolate.cs
athena.cs
icarus.cs
rainbow.cs
Other

FIGURE 15. Executing a parallel hostless program on a network of 2 Sun4-Sos4 machines

5.2 Architecture

The PELLPACK execute tool consists of a Tcl driver, a Tcl/Tk graphical user interface, and the site specific configuration database. Upon invocation, the execute tool loads all local configuration required to perform the supported operations. Figure 16 depicts the software components of this tool.

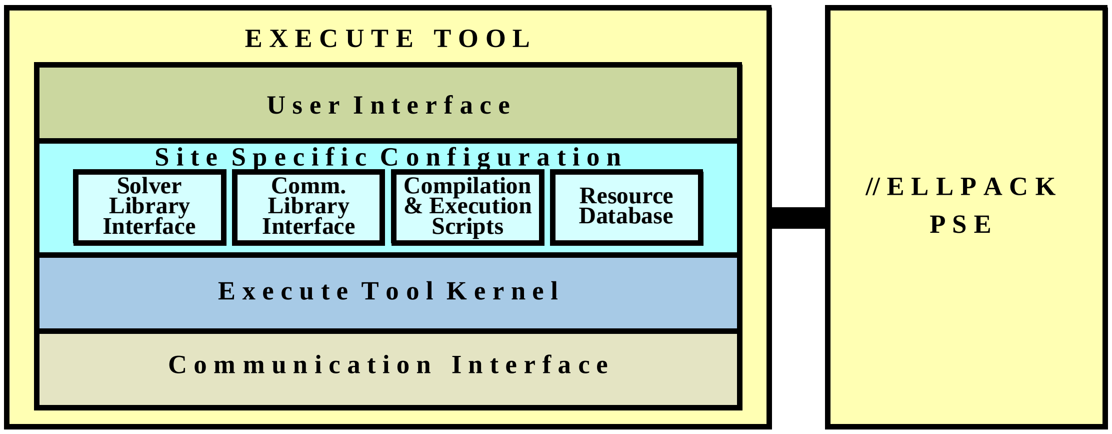


FIGURE 16. The software architecture of the execute tool

6. The PELLPACK Expert System Support

Given the large number of solution frameworks available in PELLPACK and the number of possible implementations of a framework segment, it is clear that most users will not be able to select the most efficient set of options needed to best solve the problem at hand. The PYTHIA knowledge based system component of PELLPACK addresses this algorithm selection problem by automatically selecting a solution scheme to use to solve a given problem within user specified performance objectives [25]. The PYTHIA system is not a part of PELLPACK, but it operates on performance data produced by PELLPACK.

The approach that PYTHIA takes to solve this problem is to select the best solution scheme based on the measured performance of various solvers on “similar” problems. Problem similarity is measured by comparing characteristics of the problem at hand with the characteristics of problems that have been solved before. Problem characteristics and performance information about the effectiveness of various solution schemes on these problems are assumed to be available from the PYTHIA knowledge base. Clearly, for better performance it is important to have a very large and growing database that continues to accumulate knowledge about the PDE problems that are being solved. Hence it is important to develop techniques for efficiently locating “similar” problems so that the algorithm selection can be done quickly. This is achieved by grouping sets of problems into classes based on some set of characteristics and then restricting the search to problems belonging to similar classes of problems.

The PYTHIA system utilizes the ELLPACK Performance Evaluation System [4] (as modified to support the PELLPACK solvers) to generate the performance data that provides initial information to the knowledge bases. The raw performance information is automatically transformed into rules and facts and stored in the knowledge base. The information in the knowledge base includes individual problem characteristic vectors, problem class characteristic vectors, and problem and class performance rules. These are used by the inferencing logic to determine the best solution scheme and parameters for a given problem.

PYTHIA inferencing logic includes traditional case-based and clustering-type techniques as well as neural network techniques to help determine the class(es) to which a problem belongs to. Once a problem’s class is determined, then the problem is compared against all the exemplars of that class to determine the best match. Then, the performance of various solution methods on that problem is used as the basis upon which to select the solution scheme for the given problem.

After the solution method is selected, its parameters must be determined. The basic parameter to determine is the number of degrees of freedom that should be present in the discretized PDE to achieve the user specified performance objectives. PYTHIA balances between conflicting user constraints to give the best possible choice for the solution algorithm and its parameters.

7. The PELLPACK Development Environment

The following two scenarios show how the PELLPACK development environment can be used for educational purposes. It is important to note that both scenarios are possible without the slightest modification of the PELLPACK system itself. Instead, we rely on the power and flexibility inherent in the design of the open architecture which was described earlier.

A graduate class in parallel programming is assigned to write the code for several domain decomposition algorithms. After generating their decomposition, the student must write the data to a file in the PELLPACK format. This file can immediately be brought into PELLPACK's graphical environment by inserting its filename into a `decomposition` segment, thus allowing the decomposition editor to load and display the new decomposition. Moreover, these decomposition files can even be used to execute a PELLPACK problem in parallel, using any of the available parallel solution schemes. The class executes the program on all available parallel platforms, and collects timing data for several different decompositions by varying the number of subdomains generated by their algorithms. The data collected by the students describes the performance of their decompositions for different numbers of subdomains, so they can compute and graph the speed-up. They can also compare the performance of their decompositions to those generated by the algorithms already available within PELLPACK. This use of the development environment requires no programming on the part of the students other than writing the decomposition to a file in the pre-defined (PELLPACK) format.

A class in numerical methods is to write a collocation discretizer. Testing the correctness of the code and evaluating its efficiency for a test suite of PDE equations is done within the PELLPACK environment. The discretizer is written using the PELLPACK data structures for the input and output arrays and variables. Workspace and other variable space allocation is defined through PELLPACK language constructs. The discretization code is inserted as a Fortran segment immediately before the solution segment of the problem definition. PELLPACK's language processor embeds this code in the resulting program at the appropriate location, and the execute tool handles the linking of the additional user specified compiled objects. Students can very easily test their discretizers on many different PDE problems, using PELLPACK's test suite. The performance of the discretizer is captured as timing data which is output at each execution. The development environment has been used in this way for testing linear system solvers, mesh generators, and many other kinds of user-written sequential and parallel code.

The components of PELLPACK that allow it to function as a development environment are the following: open architecture, standard interfaces for the PDE problem and solution process specifications, published file formats and data structures for all input and output, an extensible database defining the solver modules, the Fortran language segment, the language processing tools, and the configurable facilities of the execute tool. The table below describes how these components work together to allow developers to add their own PDE solver components and have them inter-operate seemlessly with the existing components. Development tasks that require PELLPACK source code are marked with (*).

TABLE 8. PELLPACK development tasks

Development task	**Description of integration process**	**Components of the PELLPACK development environment used**
Use off-line code to generate mesh, decomposition, etc.	use PELLPACK file format to save data and insert filename in appropriate language segment.	published input file formats, language processing tools

Write new code for discretizer or linear system solver	define routines using PELLPACK data structures for input/output, insert call to top-level routine in Fortran segment, compile routines on target platforms	standard interfaces for PDE solution process specifications, Fortran language segment, language processing tools, configurable execution facilities
Test existing code for discretization or linear system solver	write interface routines transforming existing data structures to PELLPACK data structures, insert call to conversion routines and call to top-level solver routine in Fortran segment, compile routines on target platforms	standard interfaces for PDE solution process specifications, Fortran language segment, language processing tools, configurable execution facilities
Integrate permanent module code into PELLPACK (*)	define interface routines using PELLPACK standard interfaces, add module definition to extensible module data base, add permanent new library to Execution configuration	standard interfaces for PDE solution process specifications, extensible module data base, configurable Execution facilities
Integrate visualizer, mesh generator, geometry decomposer or other new tool to PELLPACK GUI (*)	write routine to convert PELLPACK format data structure or file to new tool format. Add following items to graphical environment: button to invoke tool, callback to call converter, call to start up tool	published file formats and data structures for all input and output

Adding a permanent module to PELLPACK requires modification of the module data base. This can only be done when source code for the language processor is available, since the changes must be compiled into the runtime system. To add a permanent module to PELLPACK, a developer must put the module definition into the data base. This information includes: (1) the “type” of module (identified by the language segment where it will appear in the problem definition), (2) the name of the module, the list of module parameters, and their default values which can be modified by the user, (3) the Fortran call to the top-level routine of the module, (4) the list of data structures needed by the new code, and (5) the memory requirements for existing PELLPACK data structures and any new data structures. After the modified language processor is installed, the new module is available as a standard part of the PELLPACK system.

8. WEB PELLPACK

Web PELLPACK [52] is an instance of the PELLPACK system that has been made available for public use via the World Wide Web at the URL http://pellpack.cs.purdue.edu/. The goal of the Web PELLPACK service is to allow remote users to access and use the PELLPACK system in a safe, secure and effective manner. The design was guided by the following principles: 1) outside users should not have direct access to server machine(s) for obvious security reasons, 2) control access to the software for accountability purposes, and 3) users must have privacy; i.e., one user should not be able to freely browse other users’ files.

To satisfy these constraints, an account-oriented model where users “log in” to their “account” and then access the software was developed. These “accounts” are created within the data space of a custom web server. Access to files within such an account is controlled using standard web security constraints. To maintain security from users’s “breaking in” to the server machines, several levels of Unix security are used.

To run Web PELLPACK (as a demonstration or otherwise), users need to have the X window system operational on their machines. In order for the machine providing the Web PELLPACK service to display X windows on the user’s display, users must instruct their own machine to permit this action. The command for doing so is conveniently provided to the user in a set up page in both demonstration and actual runs. Once the appropriate permissions are set up, the demonstration is started by pressing the “Run” button. The demonstration PELLPACK system then runs on the web service machine and displays its windows on the user’s display. The web browser is blocked until the execution is complete. Once the operation is complete, the user is presented with a page that allows them to send comments to the PELLPACK developers.

For normal operation, the user must first request access to use the Web PELLPACK service by filling out a (Web) form and submitting it to the service administrators. Processing the request requires the administrator to provide the user with an initial login identifier and an initial password which the user may use to create an account for themselves. Then, this login is added to the WWW server access control file that controls who has access to the account creation page. Once users receive the initial login and password, they visit the account creation page using these tokens and sets up an account for themselves. The account creation request is processed automatically by creating a "home" directory for the user within the WWW server's data space and by creating an access control file there that restricts access only to this user.

Once the account has been set up, users may log in any time and use the PELLPACK system. The home directory works like a normal home directory; i.e., users may use it as a persistent working area and may save programs and results there. Files may be off-loaded from this directory using a web browser, but we currently do not support uploading files to this directory. File uploading will be supported later using the FILE input type in HTML3 [26] forms.

The primary concern of anyone providing any Internet-wide service is that of security. The security concerns in the Web PELLPACK service include ensuring that users do not get access to files outside of the service boundaries, that they have restricted access outside of their home directory, that they cannot compromise the system in any way and that they cannot access and compromise the local network in any way. Details of how all these constraints are maintained are included in [52].

9. ACKNOWLEDGMENTS

Many people have contributed to the design and implementation of PELLPACK in a period of about 6 years. The following is an incomplete list of the contributors to PELLPACK (in alphabetical order): A.C. Catlin, Y. Chen, N.P. Chrisochoides, C.C. Chui, C.L. Crabill, M.G. Gaitzatzes, S.G. Gaitatzes, P.N. Galani, H. Gu, A. Hadjidimos, E.N. Houstis, N.E. Houstis, B. Jackson, A. Joshi, H.C. Karanthanasis, S.B. Kim, S. Kim, T. Ku, Y.L. Lai, D. Maharry, A.L. Ng, W.L, Ng, S. Markus, J.R. McCombs, A. Ocken, K. Pantazopoulos, P.N. Papachiou, N. Ramakrishnan, J.R. Rice, M.K. Samartzis, M.L. Shyu, C.H. Song, E.A. Vavalis, V. Verykios, R.L. Walker, S. Weerawarana, P. Wu, S. Zhang, and J.L. Zhen.

This project was partially funded by the National Science Foundation (CCR-8704826, CDA-9123502, CCR-9202536, ASC-9404859, CDA-9422038, CCR-8819501, CCR-8922537, CCR-9311486), National Aeronautics and Space Administration (NGT-50708), Air Force Office of Scientific Research (AFOSR-88-0243, F49620-92-J-0069), the Army Research Office (DAAH04-94-G-0010, DAAH04-G-0010), ESPRIT grants 2702, 6643, and support from the AT&T Foundation, INTEL Corporation, FIRST Info Inc, and the Purdue Research Foundation.

10. REFERENCES

[1] Baldwin, B. and Lomax, H. 1978. Thin-layer approximation and algebraic model for separated turbulent flows. *AIAA-78-257.*

[2] Baldwin, K. 1990. *Patran Plus User Manual, Release 2.5, Vols I and II*. PDA Engineering, PATRAN Division.

[3] Bijan, M. 1978. Fluid Dynamics Computation with NSC2KE, A User Guide, Release 1.0. *No RT-0164, Mai 1994, Institut National de Recherche en Informatique et en Automatique 257*.

[4] Boisvert, R. F., Houstis, E. N., and Rice, J. R. 1979. A system for performance evaluation of partial differential equations software. *IEEE Trans. Software Engineering. SE-5, 4*, 418-425

[5] Boisvert, R. F., Howe, S. E., and Kahaner, D. K. 1985. GAMS -A framework for the management of scientific software. *ACM Trans. Math. Softw. 11*, 313 -355.

[6] Brugger, E. S., Leibee, A., and Long, J. W. 1994. *MeshTV User's Manual*. Lawrence Livermore National Laboratory.

[7] Chrisochoides, N. P., Houstis, E. N., and Rice, J. R. 1994. Mapping algorithms and software environments for data parallel pde iterative solvers. *Journal of Distributed and Parallel Computing, 21*, 75-95.

[8] Chrisochoides, N.P., Houstis, C.E., Kortesis, S.K., Houstis, E. N., Papachiou, P.N., and Rice, J. R. 1991. Domain Decomposer: A software tool for mapping PDE computations to parallel machines. R. Glowinski, et al. (Eds.). *Domain Decomposition Methods for Partial Diferential Equations IV,* 341-357. SIAM Publications.

[9] Chrisochoides, N.P., Houstis, C.E., Kortesis, S.K., Houstis, E.N., and Rice, J. R. 1989. Automatic load balanced partitioning strategies for PDE computations. E.N. Houstis and D. Gannon, editors, *Proceedings of International Conference on Supercomputing,* 99-107. ACM Press.

[10] Cooper, G.K., Jones, R.R., Power, G.D., Sirbaugh, J. R., Smith, C.F., and Towne, C. E. 1994. *A User's Guide to NPARC, Version 2.0.* NASA Lewis Research Center and Arnold Engineering Development Center.

[11] Denton, J. D. 1982. An improved time marching method for turbo-machinery flow calculation. *ASME 82-GT-239*.

[12] Energy Science & Technology Software Center. 1995. *The Maxima system*. Oak Ridge, TN.

[13] Gallopoulos, E., Houstis, E.N., and Rice, J. R. 1994. Computer as thinker/doer: Problem solving environments for computational science. *IEEE Comp. Sci. Engr., 1,* 11–23

[14] Gallopoulos, E., Houstis, E.N., and Rice, J. R. 1995. Workshop on problem-solving environments: Findings and reommendations. *Computing Surveys, 27,* 277-279.

[15] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. 1993. PVM 3 User's Guide and Reference Manual. *Technical Report TM-12187*. Oak Ridge National Laboratory.

[16] Gropp, W., Lusk, E., and Skjellum, A. 1994. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press.

[17] Gross, L., Roll, C., and Schoenauer, W. 1993. VECFEM for mixed finite elements. *Technical Report Interner Bericht Nr. 50/9.* Rechenzentrum der Universitat Karlsruhe.

[18] Heath, M. T. and Finger, J. E. 1991. Visualizing the performance of parallel programs. *IEEE Software,* 8, 29-39.

[19] Hindmarsh, A.C. 1983. Odepack, A systematized collection of ODE solvers. *Scientific Computing*. R.S. Stepleman, et al. (Eds). 55-64. North-Holland, Amsterdam.

[20] Houstis, E. N., Mitchell, W. F., Rice, J. R. 1983. Collocation software for second order elliptic partial differential equations. *CSD-TR 466,* Department of Computer Science, Purdue University.

[21] Houstis, .E. N., Papatheodorou, T. S., and Rice, J. R. 1990. Parallel ELLPACK: An expert system for the parallel processing of partial differential equations. *Intelligent Mathematical Software Systems.* 63-73. North-Holland, Amsterdam.

[22] Houstis, E. N., Rice, J. R., Chrisochoides, N. P., Karathanasis, H. C., Papachiou, P. N., Samartzis, M. K., Vavalis, E. A. , Wang, K. Y., and Weerawarana, S. 1990. Ellpack: A numerical simulation programming environment for parallel MIMD machines. In D. Marinescu and R. Frost (Eds.). *International Conference on Supercomputing,* 96-107. ACM Press.

[23] Houstis, E. N., and Rice, J. R. 1992. Parallel Ellpack: A development and problem solving environment for high performance computing machines. In P. W. Gaffney and E. N. Houstis (Eds.). *Programming Environments for High-Level Scientific Problem Solving,* 229-241. North-Holland.

[24] Houstis, E.N., Kim, S.B., Markus, S., Wu, .P., Houstis, N.E., Catlin,, A.C., Weerawarana, S., and Papatheodorou, T.S. 1995. Parallel ELLPACK PDE solvers. *Second Annual Intel SuperComputer User's Group Conference*.Also: CSD-TR 95-042. Department of Computer Science, Purdue University

[25] Houstis, E.N., Weerawarana, S., Joshi, A., and Rice, J. R. to appear. PYTHIA: A knowledge based system to select scientific algorithms. *ACM Trans. Math. Software.*

[26] HyperText Markup Language (HTML). 1996. Working and Background Materials, http://www.w3.org/pub/WWW/MarkUp/.

[27] Joe, B. 1991. GEOMPACK-A software package for the generation of meshes using geometric algorithms. *Adv. Eng. Software, 13*, 325-331

[28] Kim, S. B. 1993. *Parallel Numerical Methods for Partial Differential Equations.* Ph.D. Thesis. CSD-TR-94-000. Department of Computer Science, Purdue University.

[29] Kim, S. B., Houstis, E. N., and Rice, J. R. 1994. Parallel stationary iterative methods and their performance. Marinescu, D. and Frost, R. (Eds.), *INTEL Supercomputer Users Group Conference*.

[30] Kinkaid, D., Respess, J., and Grimes, J. 1982. Algorithm 586: Itpack 2c: A Fortran package for solving large linear systems by adaptive accelerated iterative methods. *ACM Trans. Math. Software., 8*, 302-322.

[31] Madsen, N.K. and Sincovec, R.F. 1979. Algorithm 540: PDECOL, general collocation software for partial differential equations, *ACM Trans. Math. Software, 5*, 326-351.

[32] Markus, S., Kim, S. B., Pantazopoulos, K., Ocken, A. L., Houstis, E. M., Wu, P., Weerawarana, S., and Maharry D. 1996. , Performance evaluation of MPI implementations and MPI based Parallel ELLPACK solvers, *Proc. 2nd MPI Developer's Conference*. 162-169. IEEE Computer Society Press.

[33] Markus, S. and Houstis, E.N. 1996. Parallel Reuse Methodologies for Elliptic Boundary Value Problems. *CSD-TR 96-056*. Department of Computer Science, Purdue University.

[34] Melgaard, D. K. and Sincovec, R. F. 1981. General software for two-dimensional nonlinear partial differential equations. *ACM Trans. Math.Software, 7*, 106-125.

[35] Mitchell, W. F. 1991. Adaptive refinement for arbitrary finite element spaces with hierarchical bases. *J. Computational and Applied Math., 36*, 65-78.

[36] Mitchell, S.A. and Vavasis, S.A. 1992. Quality mesh generation in three dimensions. *Proc. ACM Computational Geometry Conference*, 212-221. ACM Press.

[37] Mitchell, S.A. and Vavasis, S.A. to appear. An aspect ratio bound for triangulating a mesh cut by an affine set.

[38] Reed, D. A., Aydt, R.A., Noe, R., Phillip, J., Roth, C., Shields, K. A., Schwartz, B., and Tavera, L.F. 1993. Scalable performance analysis: The Pablo performance analysis environment. Anthony Skjellum (Ed.) *Proceedings of the Scalable Parallel Libraries Conference,* 104-113. IEEE Computer Society.

[39] Reed D.A. 1994. Experimental performance analysis of parallel systems: techniques and open problems. *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 25-51. IFIP.

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

[41] Rice, J.R. 1989. Libraries, software parts and problem solving systems. In Cai, Fosdick and Huang (Eds.) *Symposium on Scientific Software,* 191-203. Tsinghua Univ.Press.

[42] Schmauder, M., Weiss, R., and Schoenauer, W. 1992. The CADSOL program package. *Technical Report Interner Bericht Nr. 46/9.,* Rechenzentrum der Universitat Karlsruhe.

[43] Schoenauer, W., Schnepf, E., and Mueller, H. 1985. The FIDISOL program packag*e. Technical Report Interner Bericht Nr. 27/85*. Rechenzentrum der Universitat Karlsruhe.

[44] Scientific Computing Associates, Inc. 1995. *PCGPACK2: A library of Fortran 77 subroutines for the solution of large sparse linear systems*.

[45] Shadid, J.N. and Tuminaro, R.S. 1992. Coarse iterative algorithm software for large scale MIMD machines: an initial discussion and implementation. *Concurrency:Practice and Experience*, *4*, 481-497.

[46] SHAPES Geometric Computing System. 1992. *Geometry Library Reference Manual (C Edition)*. XoX Corporation.

[47] Sincovec, R. F. and Madsen, N. K. 1975. *S*oftware for nonlinear partial differential equations. *ACM Trans. Math. Software, 1*, 232-260.

[48] Walker, R. 1996. The performance of a parallel time-stepping methodology in the Parallel (//) ELLPACK programming environment. *Proceedings of the 1996 Simulation Multiconference.* 206-213.

[49] Weerawarana, S. and Wang, P. S. 1992. A Portable code generator for Cray Fortran. *Trans. Math. Software, 18*, 241-255.

[50] Weerawarana, S., Houstis, E.N., Catlin, A.C. and Rice J.R. 1995. *PELLPACK*: A system for simulating partial differential equations. C.E. deSilva and M.H.Hanzu (Eds). *Modeling and Simulation*. 122-126. IASTED-ACTA Press Anaheim, Ca.

[51] Weerawarana, S., Houstis, E.N., and Rice, J.R. 1992. An interactive symbolic-numeric interface to parallel ELLPACK for building general PDE solvers. Donald, Kapur and Mundy (Eds.) *Symbolic and Numerical Computation for Artificial Intellligence,* 303–321. Academic Press.

[52] Weerawarana, S., Houstis, E.N., Rice, J. R., Gaitatzes, M.G., Markus, S., and Joshi, A. 1996. Web PELLPACK: A Networked Computing Service on the World Wide Web. *CSD-TR-95-011*. Department of Computer Science, Purdue University.

[53] Womble, D.E. 1990. A time-stepping algorithm for parallel computers. *SIAM J. Sci. Stat Comput, 11,* 824-837.

[54] Wu, P. and Houstis, E. N. 1994. Parallel mesh generation and decomposition*. CSD-TR-93-075*. Department of Computer Science, Purdue University.

[55] Wu, P. and Houstis, E.N. 1993. An interactive X-windows based user interface for the XoX solid modelling library. *CSD-TR-93-015, CAPO Report CAPO-93-08.* Department of Computer Science, Purdue University.

[56] XYZ Scientific Applications, Inc. 1993. *TrueGrid Manual*.

[57] Zhang, S. 1995. *Molecular-mixing measurements and turbulent-structure visualizations in a round jet with tabs*. Ph.D. Thesis. School of Aeronautics and Astronautics. Purdue University.