

//ELLPACK

Development

Environment

INTRODUCTION

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 seamlessly 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.

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.

Scenario 1: Using PELLPACK in a Parallel Programming Class

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.

Scenario 2: Using PELLPACK in a Numerical Analysis Class

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.

REFERENCES FOR CHAPTERS 1-4

- [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*

