Knowledge-Based Support for Scientific Programming

Elaine Kant
Schlumberger Laboratory for Computer Science
P.O. Box 200015, Austin, Texas 78720-0015 USA
kant@slcs.slb.com

Abstract

Scientific computing traditionally is carried out by mathematical modelers who write their own Fortran programs. As faster machines and new architectures make more complex problems computationally feasible, programming becomes more time consuming and automation becomes more cost effective. Scientific computing comprises a variety of activities including model formulation, coding, and interpretation. This talk considers how knowledge-based tools can support these activities, with a focus on the support of code generation.

1 The Scientific Computing Process

Until recently, scientific computing was carried out by mathematical modelers who wrote their own Fortran programs. However, this programming process is becoming more time consuming as larger, faster machines and parallel architectures make more complex problems computationally feasible. The programming complexity is due both to modelers attempting problems with more detailed mathematical models, and to the difficulties in optimizing codes for the new architectures. The increasing complexity of scientific computing makes automation an attractive prospect.

Today, a broad range of possibilities for support of scientific computing is being explored. We will consider the type of scientific computing that involves the following activities:

- plan experiments to explore aspects of physical phenomena via mathematical models,
- specify computational goals to a desired degree of accuracy and specify constraints on the available target computing environment,
- formulate a mathematical model (including a geometric model) of the physical phenomena,
- solve the model with the help of computer programs,
- validate, visualize, and interpret the results,
- revise the model and carry out additional experiments based on the interpretation,
- document the results for other human beings and for ease of further automation.

2 Automation of Scientific Computing

Possibilities exist for supporting the automation of each of these activities. In particular, knowledge-based support technologies may be helpful for representing and reasoning about many of the activities. However, these technologies are useful only to the extent that they generate solutions more quickly, provide more efficient solutions, or enlarge the scope of what is possible. For example, solving the model with the help of computer programs is one activity that can be profitably automated with knowledge-based software engineering techniques ([3, 4, 12, 9]). Automation today can, in certain domains, generate solutions more quickly and make possible the generation of more complex codes; however, automation does not routinely produce more efficient codes than codes created manually. Similarly, research is taking place in knowledge-based support for most of the scientific computing activities described above. Model formulation is an especially active area ([10, 1, 8, 5, 15]), and there is also work in interfaces to numerical software [14, 11] and data preprocessing [2]. Finally, some work exists on combining support for related activities [13, 6]. For the domains in which these systems have expertise, they can carry out scientific computing steps quickly and reliably. However, the scope of what can be done and the efficiency of the solutions are generally not competitive with manual techniques. Important research remains to be done on codifying the huge vol-
ume of scientific computing techniques and on creating an infrastructure that allows work by different researchers to be complementary and build toward practical systems.

3 Synthesis of Scientific Programs

The SINAPSE program synthesis system [7] illustrates the use of knowledge-based techniques in model solution. SINAPSE generates finite difference programs to simulate partial differential equation models. The specification language approximates the level of description that modelers use in technical papers. The generated codes are optimized for a specified target language or architecture (currently Fortran, Connection Machine Fortran, and C). Comments in the code tie the statements back to the original mathematical and geometric models. SINAPSE has generated on the order of a dozen application programs (primarily for seismic and sonic wave propagation) that Schlumberger modelers have found useful. Although the current focus is on finite difference algorithms, SINAPSE is designed for growth into other algorithms.

SINAPSE is designed to be part of a problem-solving environment for forward modeling that includes several additional scientific programming activities. The problem geometry will be specified by an interactive interface that uses specialized knowledge to create geometries appropriate to the application. The interface will produce a gridded geometry file for data input to the program at run time, or will be queried during code generation to produce a program specialized to the geometry. SINAPSE will generate calls to visualization routines from a library rather than synthesizing visualization code. Some code generation for experiment planning and model validation is anticipated, but no specific plans exist yet for automating feedback and model revision activities.

4 Knowledge for Program Synthesis

A SINAPSE specification, like that of many scientific code generation systems, consists of a basic requirements description and the implementation choices that the modeler wants to enforce. Requirements include the continuous form of the PDEs representing the model and the desired output. These specifications could be generated by some other model formulation system. For formulations known to the system, the specification can be simply the name of the formulation and problem domain (such as StressStrain WavePropagation) and parameters (such as elastic or acoustic) and the names of the spatial dimensions (such as x,y); the system will generate the mathematical model from the parameterized description. Geometry is another part of the specification (some geometric details can be run-time data inputs), as is the physical interpretation of the modeling variables.

Implementation choices involve algorithmic method and numerical approximation decisions. Approximations should be selected based on available resources and knowledge about the performance of the alternatives. SINAPSE can make the choices about data representations (such as special representations for diagonal arrays, compressing arrays with indexing, and storing versus recomputing values in a virtual array) and will apply heuristics and defaults for other choices, but the modeler can override any decisions as part of the specification.

SINAPSE generates code by applying refinements and optimizing transformations, first producing an algorithm description and then an array-based, high-level language program before generating target code. Decisions about the algorithmic method are used to select algorithm schemas in an internal representation. Based on the specific approximation choices, the algorithm schemas are elaborated with constructs in an array-level intermediate language. Patterns on derivatives are replaced by difference operators, with indexes adjusted to compensate for any staggering that was specified. Domain-specific knowledge may be used to determine how to refine some constructs such as absorbing boundaries. Information about the available parallelism is maintained in the intermediate language.

SINAPSE generates implementations targeted toward a specified architecture. One example involves multiplication by a conditional expression with exponentially decaying functions at the edges and ones in the middle. For a massively parallel architecture, a "store" decision implements the conditional expression as one large array that is initialized and then used repeatedly for fast matrix multiplication. For a small-memory, serial architecture, the "recompute" implementation results in a set of loops with small multiplications over only the boundary regions.

At the last stage of code generation, the intermediate-language program description is translated into target code, serializing parallel constructs as necessary or exploiting high-level, target-language constructs. If this intermediate-language could be standardized among different systems, this would be another good interface for sharing back-end optimizers and translators in a common framework.
SINAPSE requires a good deal of knowledge to function effectively. The system consists of over 20,000 lines of Mathematica code. In the current version, the declarative representations of the domain knowledge and problem-solving structure account for about 20% of the system code. This includes design facts in a fill-in-the-blank form, menu-choice design decisions, and program-synthesis tasks. The program-synthesis inference framework is about 15% of the total system. Rule-based knowledge about how to refine domain descriptions to algorithms and coding constructs accounts for about 35% of the system knowledge. Finally, knowledge about code generation is about 30% of SINAPSE (about half is target-language specific). New knowledge is currently added by the system developers with some rudimentary tools to help add and validate new entities.

Research on SINAPSE and other knowledge-based support tools for scientific programming demonstrates the potential effectiveness of the approach. However, further work is needed to integrate different types of tools, to find the most effective representations for shared scientific knowledge, and to support more efficient knowledge acquisition.

References


