OON-SKI: An Introduction

AL VERMEULEN AND MARGARET CHAPMAN
Rogue Wave Software, Corvallis, OR 97339

On April 25–27, 1993, Rogue Wave Software, in cooperation with SIAM, sponsored the first annual Object-Oriented Numerics Conference (OON-SKI '93) at Sunriver, Oregon. The intention was to bring together mathematicians, scientists, engineers, and programmers who are interested in object-oriented numerics: the use of modern object-oriented techniques in the design of software solutions to numerical problems. The conference was very successful: there were over 100 attendees from nine countries and more than 40 articles were presented.

The motivation for using object-oriented techniques in scientific programming is clear: as researchers continue to attempt more and more ambitious models and algorithms, they rapidly run up against the limits that can be handled using traditional procedural languages such as Fortran. Object-oriented programming allows you to effectively code algorithms at an appropriate level of abstraction, and not have to worry about irrelevant details. It does this by allowing you to package related code and data together into objects, and then work with the data through a well-defined interface, rather than working with the data directly. Object-oriented languages have been around for years. Why, then, has it not been until recently that they have been used in scientific codes? The answer is that they lacked the key features needed by the numerics community: runtime efficiency, widespread popularity, and ease of interfacing with existing Fortran and C code. The emergence of C++ has changed all that. Its dominance is clear: over 90% of the articles presented at OON-SKI '93 used C++ as their language of choice, or as the basis for an extension language.

For this issue of *Scientific Programming*, 14 articles were selected* to provide a cross section of what was discussed at OON-SKI. The conference was organized around four themes:

1. Languages and environments
2. Software components
3. Applications
4. Tools for supporting parallelism

LANGUAGES AND ENVIRONMENTS

Some of the most interesting, controversial, and thought-provoking articles at the conference concerned themselves with how to enhance C++ to create a better environment for numerics. Three concerns were addressed: improving the programming environment to make it easier to code, helping the compiler perform optimizations, and adding support for parallelism to the language.

Bob Ballance's article on scientific frameworks presented an approach where an application designer starts with the skeleton of a code already built, and then fleshes it in for his particular requirements. Electronic design aides can assist the user by recognizing elements of his code and reminding and suggesting how other elements should fit in. With such a framework, building complex scientific codes becomes radically simpler than it now is—at least if you can find a framework for your type of problem.

Two of the articles presented here concern themselves with the problem of optimizing expres-

Received September 1993

* One of these articles appears in the previous issue. Please see *Scientific Programming*, Vol. 2, No. 3, pp. 7–22, "Distributed pC++: Basic Ideas for an Object Parallel Language," by F. Bodin, P. Beckman, D. Gannon, S. Narayana, and S. X. Yang.

Scientific Programming, Vol. 2, pp. 109–110 (1993)
CCC 1058-9244/94/040109-02

sions. In C++ an expression such as

`Array A = B + C + D;`

gets compiled into separate calls to operator+(), linked together by temporaries. Here we see that encapsulating abstractions into objects is a double-edged sword: although it is much easier to work with A, B, C, and D as Array objects, the compiler can no longer explicitly see the looping structure implied by the expression, and thus can no longer optimally vectorize the resulting expression. The articles by Ian Angus and Mike Sharp look at different ways of letting the compiler know how to optimize expressions.

SOFTWARE COMPONENTS

One of the promises of object-oriented design is easily reusable code. In this area articles were presented that focused on the design and implementation of libraries of numerical classes intended for use as building blocks in applications. It became clear from conversations at the conference that this is one area where people are having difficulty: finding the correct abstractions to model is not easy.

One abstraction that is clearly important to scientists is arrays of numbers. Tony Willis's article presents a set of C++ classes that represent N-dimensional arrays and allow you to work with them efficiently. The article by Verner et al. describes an array library tailored for finite difference and finite element codes. Kjell Gustafsson presented a library for integrating systems of ordinary differential equations. His approach allows for flexibility and ease of use both for the user of the library, and for someone who is coding new algorithms for ODE integration.

APPLICATIONS

In this section articles were presented on complete applications constructed using an object-oriented methodology. Even though this field is relatively new, large, complex applications have already been constructed, and these are described here. A large number of submissions from the engineering PDE community were received, two of which are reprinted here: Weissman, Grimshaw, and Ferraro's finite element application, coded in the Mentat parallel processing environment, and Crutchfield and Welcome's C++ implementation of a finite difference adaptive gridding algorithm. Donna Calhoun's article gives a lucid description of an application based on classes for automatic differentiation. FastScat, a complex application for computing electromagnetic scattering, is described in the article by Lisa Hamilton et al.

TOOLS FOR SUPPORTING PARALLELISM

Although there was only one session devoted to parallelism, the subject kept coming up again and again in other sessions as well. The reason for this is twofold: One, scientific programming is compute intensive, so there is desire to spread the work among multiple processors. Second, parallel computing is difficult; there is hope that object-oriented techniques can help manage this complexity. Thus, the conference was a natural place to find experts in this field.

Bhatt et al. report on a project aimed at providing support for adaptive, irregular data structures on distributed memory machines. Their algorithms are motivated by the desire to solve large-scale multibody simulation problems.

A set of classes for manipulating arrays distributed across a distributed memory machine is described by Steve Otto in his article. Finally, the article by Dan Lickly describes a hierarchy of C++ classes to support the C* model of data parallelism, but without syntactic language extensions.

In conclusion, it is clear that we are observing the emergence of a subdiscipline: the use of object-oriented techniques in numerics. But what we are really seeing is something even more profound: finally the rejoining of scientific computing with the science of computers. Traditionally, programming has been done by engineers, physicists, and mathematicians with little or no training in computer science. Now, however, we are seeing an infusion of ideas coming from the computer science world into the scientific computing world, bringing along modern ideas on how to structure complex numerical code. Object-oriented techniques is merely one of many such ideas.

For information on next year's Object-Oriented Numerics Conference, contact Margaret Chapman, Program Coordinator, Rogue Wave Software, PO Box 2328. Corvallis OR 97339 - (503) 754-3010 phone, (503) 757-6650 FAX, amc@roguewave.com.