** **

Fortran 90 for Physics Students

By John K. Prentice, Quetzal Computational Associates (john@quetzalcoatl.com)

This is a note that I just sent to the chairman of a physics department, where freshmen are required to take a programming class offered by the engineering department. That course has traditionally taught Fortran 77, but there is now a push on from the engineering school to switch to teaching C++. As usual, the arguments being mustered revolve around criticisms of Fortran 77 by people who are too out-of-touch to know much of anything about Fortran 90. A senior member of the physics faculty asked me to contribute my two cents worth about what language they should be teaching. My reply summarizes our experience with C++ and some of our feelings about Fortran 90.

Note Sent To The Chairman Of The Physics Department:

A colleague mentioned to me that the Engineering school is considering changing Eng 120 to teach C++ instead of Fortran and that you were soliciting comments about this with regard to physics students. Even though I am not associated with the university, I wanted to contribute my thoughts, which come from a commercial as well as a research perspective. Perhaps they will be of some value to you.

As you know, Quetzal Computational Associates specializes in computational science. We currently have projects computational physics, earth sciences, and agriculture for clients which include DoE and DoD laboratories as well as commercial clients as diverse as hazardous waste companies and grain companies. During the last year, we have developed numerical methods and codes based on them for modeling contaminant flow in porous media, modeling the structural mechanics of resonant sonic drilling rigs, modeling bistatic ground penetrator radar propagation in partially saturated soils, modeling solid dynamics at high strain rates, modeling the phenological development of corn and soy beans, modeling solar insolation in the photosynthetically active spectrum based on first principle atmospheric physics, and developed neural networks for the detection of dust clouds from satellite imagery.

One of our largest computational physics projects is the development of advanced methods for modeling solid dynamics based on first principle physics. This is a multi-year, multi-million dollar project. This code numerically solves the partial differential equations for continuum solid dynamics using a hybrid finite volume /finite element technique, coupled to advanced equations of state and constitutive models for the solids and fluids in the calculation. This code and others we work with are huge number crunching codes, a modest 3d simulation will take 100 or more hours of Cray C-90 time to complete a single calculation. By any standard, they are amongst the largest computational physics simulations being done anywhere in the world. In addition, we are on the forefront in the application of parallel computing to these problems. We have projects to develop parallel versions of our codes for a diverse collection of computers, including networks of UNIX workstations, the IBM SP-2, the Cray T3D, and the Intel Paragon.

For all of these projects, we employ Fortran 90 as our main language. We do some development work in Fortran 77, C, and C++, but we are moving away from those languages as quickly as possible. There are many reasons for our choice of Fortran 90, but first let me say a bit about why we are not enthusiastic about C++. The biggest strength of C++ is probably the availability of relatively inexpensive and high quality C++ compilers for PCs. But that is a pretty minor consideration in our business and it is outweighed by the enormous liabilities we have observed with C++. First, we regard C++ as the weakest of the object oriented languages. Objective C is a far more solid and well designed OOPS language, C++ is really some OOPS capability slapped on top of C. C++ is consequently extremely inefficient, inconsistent, overly large, and enormously difficult to program in. The experience of our clients mirrors our own, and in fact many DoE and DoD laboratories are finding that their headlong rush to C++ has been a hideously expensive mistake. I know of several C++ scientific coding projects there that consumed millions of dollars and tens of man-years, only to be abandoned because the resulting code was enormously inefficient on both traditional serial computers and on their large parallel supercomputers. Similar horror stories abound throughout the programming community at this point.

Bill Gates claimed that his biggest mistake in designing their new NT operating system was adopting C++ for the graphics coding, the resulting code took years longer to write than it should have and ran

terribly slow. While OOPS is a solid development in the computer science community, I think it is fair to say that C++ is destined to be a passing fad, much like Pascal and Ada before it.

The main reason C++ has attracted the attention it has in the scientific community is because Fortran 77 was a terribly outdated language. The many weaknesses of Fortran 77 were solved with Fortran 90, however. Fortran 90 has every feature in C that is important to scientific programming and most of the features of an object oriented language (it lacks only inheritance, and that is likely going to be added in Fortran 2000). However, unlike C and C++, Fortran 90 is designed to generate executable codes that are highly optimized and thus run extremely fast. An example is pointers. Pointers are integral to C and C++ programming and because the compiler cannot determine whether a pointer is aliased, it is impossible for it to determine interprocedural dependencies. The result is a significant degradation in optimization and extremely slow execution speeds (for most scientific codes, C and C++ generally produce code which is commonly an order of magnitude slower than Fortran 90 codes, based on the benchmarks we and others have done). Fortran 90 pointers are designed to give the functionality of pointers, but with restrictions that eliminate issues such as aliasing. From a programming perspective however, an even more important point is that Fortran 90 has more natural ways of expressing the functionality that C and C++ require pointers to express. Because of this, Fortran 90 is a more natural language to program in and the time required for debugging codes is a fraction of that required by C and C++ (C++ is much worse than C, provided you are really employing an OOPS paradigm, since you find yourself spending alot of debug time going up and down inheritance trees). Another important point is that the time required to learn Fortran 90 is much less than the time to learn either C or C++.

Fortran 90 has another major advantage over C or C++. Modern scientific computing, and computing in general, is moving toward the use of parallel computers. Even PCs and workstations now come with multiple processors, so parallelism is something that everyone from an accountant to a physicist is encountering now. A major problem in programming parallel computers however is the linear memory model that is inherent to all procedural programming languages, with the singular exception of Fortran 90. A linear memory model is one that assumes that consecutive elements of an array are consecutive in memory. This was a reasonable assumption on traditional computers, but it is completely incorrect on a parallel computer. Only Fortran 90 has addressed this problem and providing standardized language support for parallelism. This support includes array syntax and many intrinsics for doing array operations varying from reduction operations such as array sums to matrix operations. With the use of Fortran 90 operator overloading and polymorphism, one can significantly extend the number of operations that avoid any reliance on the linear memory model. The fact that Fortran 90 moved away from a linear memory model is the main reason that it has become the base for so many data parallel languages such as Vienna Fortran, Fortran D, CRAFT, and High Performance Fortran. The availability of data parallel dialects of Fortran 90 is an especially large factor in favor of Fortran 90. Compilers for High Performance Fortran, for example, are now coming on the market for virtually every machine out there (including networks of workstations) and writing parallel codes in this language is straightforward. Of particular importance is that porting a Fortran 90 code to High Performance Fortran is extremely straightforward and codes written in High Performance Fortran can be run unaltered on a Fortran 90 compiler (with the exception of one HPF construct, the forall, which is being put into Fortran 95).

My own opinion is that scientists today need to know more than one language or one computing paradigm. And I think it is entirely reasonable that students learn C++ before they graduate, though even more important is that they learn how to program MATLAB and a computer algebra system such as Maple or Macsyma. But the issue is what freshmen should learn as their first language, and for that I would recommend Fortran 90 hands down. It is a better language for scientific programming and is both easier to learn and use than the alternatives. It is also much more likely to be the language students will be employing in their jobs upon graduation and it is the most promising route currently developing for the programming of parallel computers.

Appendix

Let me offer some documentation to back up my statements about the speed of C++ codes versus Fortran 90 codes. I recommend reading the article "Is C++ Fast Enough for Scientific Computing" by Scott Haney of Lawrence Livermore National Laboratory and published in "Computers in Physics", vol 8, no 6, Nov /Dec 1994, pages 690-694. Haney did three kernel type benchmarks:

1. "codes that use a small subset of C++: a simple `RealMatrix` class that provides storage management and indexing into matrices stored according to Fortran conventions. The test consists of multiplying two `RealMatrix` objects, a and b".
2. a test that "measures the cost of using a `Complex` class and operator overloading to simplify complex-number arithmetic. The test consists of multiplying two complex matrices."
3. a test that "measures the cost of using classes and operator overloading to support arithmetic operatings involving complete arrays. Consider the problem of computing inductances for a series of square coils. In C++, it is possible to calculate the inductances all at once using a `RealVector` class that manages storage and elementwise operations for vectors of arbitrary length."

He coded Fortran 77, C, and C++ versions of these simple benchmarks and ran them on 6 different types of computers ranging from low end UNIX workstations to a Cray C-90. On every test, the C and C++ were slower than the Fortran 77 versions, often by factors of 2 to 3. To quote his conclusions:

"Based on these tests, it appears that a C++ program that does serial calculations with arrays on workstations and vector supercomputers will probably be slower than a similar program written in C or Fortran. Moreover, the speed difference may be uncomfortable large. Other studies support this conclusion [he references A.D. Robinson, computer code OOPACK: a benchmark for comparing OOP vs. C-stype Programming, Shell Development C., TX, 1993 and K. G. Budge, Proceedings of the Second Annual Object-Oriented Numerics Conference, Sunriver, OR, 1994 (unpublished)]. I think it is fair to say that, right now, C++ may *not* be fast enough for some scientific computing applications unless issues other than performance dominate the language decision."

He goes on to say that he is hopeful that continued compiler development will improve the performance of C++ codes and to also note that with the emergence of C++ interfaces into Fortran, one can employ Fortran where you need good performance by calling Fortran routines from C++ (in which case, what does C++ buy you?). His next paragraph confirms my earlier statement about optimization of C++:

"In comparison with Fortran, optimization of C++ is potentially more difficult because of pointer-aliasing issues. In addition, operator overloading can present C++ compilers with constructs that are quite different from those found in C programs."

Haney is hopeful that the performance situation will improve as C++ compiler technology improves. I am more skeptical; I have heard this hope expressed for years now, and yet — despite all the money going into C++ compilers — performance continues to be terrible on scientific applications. If after 20+ years, C compilers still produce slower code than Fortran 77 compilers (to say nothing of Fortran 90 compilers), then what reason is there to believe that a language built on top of C is ever going to run faster?

There are also legitimate questions about whether the OOPS paradigm itself isn't fundamentally impossible to optimize with our current state of the art. C++ adds to the difficulties, it lacks any overall design strategy (there isn't even a grammar defined for C++!) and its implementation of many features is poorly thought through and often times ambiguous and internally inconsistent.

I invite any others with references to well done benchmarks of these languages to post them. Performance isn't the only criterion by which a language should be measured, but in the scientific community, it is extremly important. If C++ offered enormous ease of use, shorter debug times, or less cost to maintain than Fortran 90 (or even C), it would at least have an argument in its favor. But it doesn't offer any of these things.