



### Guest View: **Fortran Matters**

Malcolm Cohen is principal technical consultant for Numerical Algorithm Group Ltd.

April 15, 2004 — **Fortran 2003, a bit of a misnomer perhaps, arrives in 2004. However, in a world dominated by Java, Visual Basic, C/C++ and C#, many of which have decent numerical computing capabilities, you might ask whether anyone who doesn't have a huge base of Fortran code should care about the new generation of Fortran. Indeed you should. Here's why.**

The nature of today's computing challenges, and the direction in which many commercial and research areas are moving, is in taking on more complicated problems that involve far more code. When you add to the complexity of problems, the various languages with numeric computing capabilities do not rise to the tasks at hand equally. For many numeric computational problems, Fortran and C are comparable when it comes to computational speed. However, from the numeric point of view, Fortran is inherently safer for certain operations, as compared with C. Fortran's key focus is on highly dependable numerical facilities. If one is equally familiar with computing in any of these languages, and more or less language-neutral in orientation, one commonly concludes that Fortran is the superior alternative.

#### **TIME FOR ANOTHER LOOK**

If your last look at Fortran was in the days of Fortran 77, the choice of Fortran for new code may not be apparent. At that time, old-fashioned Fortran was not differentiated from the other numeric computing languages in this regard. With the advent of Fortran 95, the paths of the languages began to significantly diverge in terms of which were well suited for complex problems. Fortran 2003 is better still.

Indeed, when computing problems involve layers of data abstraction, Fortran 2003 is especially well supported for such challenges, a claim that none of the other languages can make.

Unlike C++, which requires one to do a lot of work with it to learn how to write code well, writing Fortran code is relatively straightforward. Fortran has much better support for detecting errors, and it does not give you as much rope to hang yourself with as other languages do.

Efficient code can be especially elusive in the other numeric computing languages, compared with Fortran. You might be able to get the same efficiency with C or C++, but it will need a lot of work to get there. In particular, they allow aliasing, which makes it harder for a compiler to optimize code. Moreover, whenever problems require array processing, Fortran is clearly better than all the other languages, and especially with the advent of the new Fortran 2003 standard.

In reality, any attempt to make one language the ultimate solution misses the far greater opportunities one can derive from building multilanguage applications that utilize the forte of each language scheme. The superiority of Java front ends for many commercial

applications is widely recognized, in that it is heads above the other languages in bringing nontechnical users seamlessly into a computing engine. However, if you have numerically intensive computational tasks, Java will hinder you with a 3x-to-10x performance hit.

In fact, the trend to build multilanguage operations was very much a front-of-mind concern to the Fortran 2003 standards committee, which put a great emphasis on extending support for other languages into the new standard. For example, the new Fortran has many features that make it easier to join C programs and Fortran programs together, including support for reading and writing files written in C that are read in Fortran as stream files.

When, then, would you include Fortran code in an optimal multilanguage application? Often, when you are handling problems with massive amounts of data, or if you are performing simulation exercises that generate enormous datasets, whatever front ends are in place are best complemented by Fortran as the accompanying numerical workhorse. Handling huge amounts of data has always been a driving factor in the development of the language. If an array processing type approach better matches the computational problem than a pure object-oriented programming (OOP) approach, Fortran would be the better option.

Conversely, if the problem were a better match for advanced OOP techniques, C++ would be a better choice.

#### **LANGUAGE NEUTRALITY**

There are good reasons why C language programming has become predominant. In Unix systems, all the systems calls are written in C and many other software packages with GUIs have a C interface without provisions for other languages. But, frankly, a far greater source of C's popularity is that it is the language that post-baby-boomers learned in college. Educational institutions may have done today's programmers a disservice by not building student expertise in a number of languages such that the pros and cons of each language could be better appreciated.

The developers of the recent Fortran standard were very focused on ensuring that the Fortran 2003 revamp would make it more compatible with an object-oriented approach that did not go overboard in this direction. In fact, C++ was a model of what not to do. For example, Fortran 2003 avoids multiple inheritance, because it adds too much complexity for too little gain in expressive power. Likewise, Fortran's unique ability to handle massive datasets and array processing was kept intact by rejecting the tactic of making everything a polymorphic object, as in other languages.

Indeed, there may be lag in perception of Fortran's role in the software developer community compared with the reality of how widespread that role truly is. Our theoretical projections of why Fortran would grow in importance in future years (i.e., driven by rapidly growing datasets and comparable growth in computing complexity) now must adjust sights to the reality of the nascent trend already taken root. There seems to be far greater demand for Fortran products than the existence of legacy code alone is likely to explain.

Look for other language standards committees to follow suit in years to come in making multilanguage applications far more practical.