Introduction

Fortran programming language and Scientific Programming: 50 Years of mutual growth

Boleslaw K. Szymanski
Rensselaer Polytechnic Institute, Troy, NY 12180, USA

This special issue of Scientific Programming is devoted to celebration of fifty years of mutual and remarkable growth of both scientific programming and its primary language, Fortran. In this brief introduction, we will remark on the past, the present and the future of Fortran and summarize the papers included in this issue.

Although the first specification of the Fortran language was released in 1956, IBM delivered its first compiler for its computer, IBM model 704, in 1957, hence this year marks the 50th anniversary of introduction of Fortran to users. The language was designed by John Backus and his colleagues at IBM with the goal to reduce the cost of programming scientific applications by providing an "automatic programming system" to replace assembly language with a notation closer to the scientific programming domain. As the computer technology has been evolving from a single computer, to parallel computers of different kinds, to multi-core processors, clusters and grids, the Fortran language has been evolving as well with the same goal of reducing the cost of programming without sacrificing efficiency. Over the years, the common standard emerged first, followed by a series of revisions. This process continues as the most recent development of a revision, scheduled for 2008, is under way. Yet, care has been taken at each revision to preserve, to the greatest extent possible, compatibility with previous versions to ensure correct recompilation of legacy codes. As a result, each revision includes only a fraction of the proposed new facilities, allowing the ideas for changes to mature before they are accepted. Over half of the century of its existence, the evolving Fortran has been the traditional and major language for scientific programming and it has played a significant role in the research on programming languages and compilers for scientific computing. The vibrant user community, the well-established committee overseeing its evolution and the process for revisions of the language ensure the significant role for Fortran in scientific programming in the immediate future and are likely to continue to secure Fortran's relevance for many years to come.

The first article in this issue, entitled "Scientific programming in Fortran" has been prepared by W. Van Snyder. The author provides a historical perspective on the development of various, progressively more modern, versions of Fortran, starting with the first standard developed by ANSI predecessor, ASA, and introduced in 1966. The article briefly describes the essence of revisions to these standards introduced in 1977, 1990, 1995 and most recently in 2003. Then, the author briefly discusses revisions being prepared for the 2008 standard release and remarks: "Of the thousands of programming languages invented, only a dozen or so have been sufficiently widely used to have had international standards, and only four of those have had revisions that attempted to keep up with language technology while preserving software investments by maintaining compatibility with previous editions: Fortran, Cobol, Ada and C . . . The temperament of the Fortran committees and community, at least at present, is that there will be future developments, which will maintain, and probably enhance, the suitability of Fortran for scientific programming." This clearly shows how unusual the longevity and vitality of Fortran have been.

One of the most significant enhancements of Fortran 2008 are co-arrays. John Reid and Robert Numrich, who pioneered development and use of co-arrays, describe in an article entitled “Co-arrays in the next Fortran standard” the main ideas behind them. The article also enlists many changes that have been made in co-array definitions compared to the initial report on co-arrays that was published in 1998. Co-arrays are interpreted as if the program containing them was replicated and all copies were executed asynchronously, each with a local set of data objects. Each executing copy is called an image. Simple syntax enables the compiler (and the programmer) an easy recognition when the data are accessed in the local image and when an access requires communication with other images. In view of rapidly increasing availability and level of existing parallelism in multi-core hardware, this feature will enable Fortran programmers to write clear and concise codes for such architectures.

In the article entitled “The transition and adoption to modern programming concepts for scientific computing in Fortran,” written by Charles Norton and his collaborators, the authors describe their experiences in their pioneering exploration of object-oriented programming in Fortran90 and Fortran95 for large-scale scientific programs. The paper reviews Fortran90/95 constructs that were used to provide basic features of object-oriented programming style and explains how these features have been employed in modernizing several significant scientific programming applications. The important observation made is that scientific programming uses object composition rather than inheritance, which was difficult to express in Fortran until 2003. The code modernization has been accomplished by separating object-oriented design in the form of skeleton code from the low-level implementation, often existing as a legacy code. This approach facilities rapid design and redesign of such patterns until the desired program is achieved. Finally, the authors discuss how other modern programming techniques, such as Design Patterns, can and have been used in modern Fortran software development.

Finally, Hans Zima in an article entitled “From FORTRAN 77 to locality-aware high productivity languages for peta-scale computing” discusses the impact of the development of High Performance Fortran (HPF) family of languages on the current language developments for peta-scale computing. An example of such a development is the language Chapel that is a modern object-oriented language developed in the High Productivity Computing Systems (HPCS) program sponsored by DARPA. It provides a general framework for the support of user-defined distributions. This is a continuation of the research and development pioneered by the author in Vienna Fortran. The central issue addressed by these developments is providing high-level language support for locality awareness in scientific programming.

During preparations of this special issue, two giants of the Fortran community died this year. John Backus, the Fortran creator, died on March 17, 2007. Ken Kennedy, the pioneer of Fortran compiler optimization and parallelization worked on the contribution to this issue until he died on February 7, 2007. We dedicate this issue to their memory and contributions to scientific programming community.