High Performance Fortran Comes of Age: Guest Editors' Introduction

ROBERT SCHREIBER[1] AND PIYUSH MEHROTRA[2]

[1]*HP Labs 3L-5, Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, CA 94304-1126; e-mail: schreibr@hpl.hp.com*
[2]*ICASE, MS 403, NASA Langley Research Center, Hampton, VA 23681-0001; e-mail: pm@icase.edu*

The High Performance Fortran (HPF) language was defined in 1992–1993 to address the need for high-level, portable, parallel programming for data parallel algorithms. The language provides directives which allow programmers to control the distribution of data across the processors of the underlying parallel machine. Computation is expressed in a distribution-independent manner using the Fortran 90 array syntax and the language extensions and directives provided by HPF. An HPF programmer expresses the parallel computation in a global index space without any explicit communication. It is the compiler's responsibility to analyze the code and generate the complex, low-level details of communication required for sharing data on the target machine. The specification of HPF version 1.0 was published in 1993, in *Scientific Programming*, volume 2, numbers 1–2. Information on the current effort to develop a revision, version 2.0, is available at `http://www.crpc.rice.edu/HPFF/home.html`.

The first commercial compilers for High Performance Fortran (HPF) are now available, and more are expected shortly. In addition, research groups have developed prototype implementations including advanced optimization and user interface capabilities. It is therefore an opportune time to consider the technologies required to make HPF usable and efficient. This special issue of *Scientific Programming* includes articles by researchers and commercial organizations describing their implementations of compilers and other HPF tools.

The first three papers describe important HPF implementations. In their paper *A Linear Algebra Framework for Static HPF Code Distribution*, C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell describe an elegant approach to some fundamental problems of HPF implementation on a distributed-memory multicomputer: local memory allocation, enumeration of the data to be sent, received, and updated by the participating parallel processors, and message vectorization. These techniques, in which iteration spaces and data references are represented as systems of linear equations and inequalities, which are solved parametrically, are the basis of the HPF compiler developed at the School of Mines of Paris.

The compilation techniques used in the Portland Group HPF compiler, which is one of the more successful and widely available HPF implementations, are described in *PGHPF—An Optimizing HPF Compiler for Distributed Memory Machines* by Z. Bozkus, L. Meadows, S. Nakamoto, V. Schuster, and M. Young. The authors present some of the optimizations performed by their compiler and also its performance on five benchmark codes.

In their paper *Kemari: a Portable HPF System for Distributed Memory Parallel Processors*, T. Kamachi, A. Müller, R. Rühl, Y. Seo, K. Suehiro, and M. Tamura describe a compiler being jointly developed by research groups in Switzerland and Japan and the NEC Corporation. The compiler is part of an overall programming environment which provides support for debugging and for performance monitoring and analysis. They also describe language extensions such as iteration mapping, overlap areas, and generalized distributions that they have implemented in their system, to support both structured and unstructured programs. Some variants of these extensions are being considered for HPF 2.0. They also describe several optimizations and present the compiler's performance on several benchmark codes.

One may already conclude that the sun never sets on efforts to implement HPF well.

© 1997 by John Wiley & Sons, Inc.
Scientific Programming, Vol. 6, pp. 1–2 (1997)
CCC 1058-9244/97/010001-02

Building on a long and distinguished record of research in parallel architectures and software at Carnegie Mellon University, D. R. O'Hallaron, J. Webb, and J. Subhlok have developed a compiler for Fx, a dialect of HPF. Their paper *Performance Issues in HPF Implementations of Sensor-Based Applications* discusses the use of Fx for typical computations in signal and image processing, showing that the implementations of independent loops, reductions, and index permutations play a critical role in obtaining good performance.

While HPF allows programmers to specify the mapping of data to processors in detail, it would be even better if an HPF compiler could assign well-chosen mappings as a program optimization. Indeed, all HPF compilers must do this for compiler-generated temporaries. In *DDT: A Research Tool for Automatic Data Distribution in HPF*, E. Ayguadé, J. Garcia, M. Gironès, M. Luz Grande, and J. Labarta of the Polytechnic University of Catalonia discuss the state of the art of automated mappings, propose some heuristic solution for the interarray axis alignment and the static and dynamic array distribution problems, and illustrate their capabilities on an important application, alternating direction iterative methods.

L. M. Liebrock and K. Kennedy also consider the problem of automatic data mapping. Their paper, *Automatic Data Distribution for Composite Grid Applications*, examines the issues that arise when an application uses a large number of small finite-element meshes, and they are not big enough to allow efficient data parallel execution on one grid at a time, with each grid mapped to all processors. Their solution uses the topology of the interactions between the grids to determine an alignment of the grids to a single template that preserves locality and allows for an automatically determined and efficient implementation.

Programmers who find HPF's multidimensional array mapping capabilities confusing should be very interested in the tool for visualizing these mappings that has been developed at Johannes Keppler University in Linz, and is described in *Visualization of Distributed Data Structures for HPF-Like Languages*, by R. Koppler, S. Grabner, and J. Volkert. The toolkit they have developed allows one to examine data mappings, and to estimate and view load distribution and communication volume.

The special issue concludes with two case studies of HPF programming. In their paper *Scientific Programming with High Performance Fortran*, E. De Sturler and V. Strumpen illustrate the process of porting Fortran 77 codes to HPF, using the xHPF compiler and taking two simple kernels and a small application as examples. The short note entitled *Experiences in Data-Parallel Programming* by T. W. Clark, R. v. Hanxleden, and K. Kennedy describes their experiences in converting a molecular dynamics program into FORTRAN D, a precursor of HPF. In decribing the transformation, they point out several issues that should be kept in mind while developing programs to be parallelized by compilers for languages such as HPF.