** **

ARTICLES

** **

An Overview of High Performance Fortran

by **Charles Koelbel**
Executive Director, High Performance Fortran Forum

Introduction

Since its introduction over three decades ago, Fortran has been the language of choice for scientific programming for sequential computers. Exploiting the full capability of modern architectures, however, increasingly requires more information than ordinary Fortran 77 or Fortran 90 programs provide. This information applies to such areas as

- Opportunities for parallel execution
- Type of available parallelism - MIMD, SIMD, or some combination
- Allocation of data among individual processor memories
- Placement of data within a single processor

The High Performance Fortran Forum (HPFF) is a coalition of industrial and academic groups working to suggest a set of standard extensions to Fortran to provide this information. From its beginning, HPFF included most vendors of parallel machines and software, government labs, and many university research groups. Public input was encouraged as much as possible. The intent of HPFF was to develop extensions to Fortran supporting high performance programming on a wide variety of machines, including massively parallel SIMD and MIMD systems and vector processors. The result of this project will be a language standard portable from workstations to massively parallel supercomputers while being able to express the algorithms needed to achieve high performance on specific architectures. Today, the definition is not complete, nor is High Performance Fortran (HPF) under consideration as a formal standard. It is likely, however, that a complete definition will be available by December 1992, and several vendors have promised software tools to compile HPF for release during 1993. By the time this article is published, the High Performance Fortran Forum will have presented a near-final version of the language at Supercomputing '92 in Minneapolis.

The purpose of his paper is to give an overview of the goals and structure of HPF. For detailed questions about the features of HPF, please consult the draft language specification and the meeting minutes; these are available by anonymous FTP, as explained below. Because various features of High Performance Fortran are under active discussion, it is likely that some details in this report will not be accurate by the time it sees print. The author, however, takes full responsibility for any mistakes of commission or omission here.

A Short History of High Performance Fortran

HPF grew out of a number of influences: frustration with the lack of portability for parallel software, advances in compiler technology, corporate strategy decisions for product placement, and others. The following timeline is incomplete, but it captures some of these motivations. The last few entries include events that have been scheduled, but have not occurred at this writing (October, 1992).

Late 1980s: Several research projects (including at least 4 Ph.D. theses) successfully use the idea of using data distribution annotations to drive the compilation process for parallel machines. Published work includes papers by Callahan & Kennedy, Koelbel & Mehrotra, Li & Chen, Rogers & Pingali, and Zima & Gerndt.

November 1990: A group led by Ken Kennedy (Rice University) and Geoffrey Fox (Syracuse University) defines the Fortran D language featuring data distribution directives. Hans Zima's group at the University of Vienna proposes Vienna Fortran shortly thereafter. Both languages inspire voluminous comments.

January 1991: COMPASS, in cooperation with Digital Equipment Corporation, begins researching software issues in parallel computation. Language and compiler technology quickly emerge as key issues, leading to investigations of many research projects, including Fortran D and Vienna Fortran.

September 1991: COMPASS drafts a language specification portable to many parallel machines. It includes the full Fortran 90 standard, Fortran D data distributions, and a *FORALL* construct. They intend this as the centerpiece of a collaborative effort between several vendors. Several other collaborations are brewing at the same time.

November 1991: DEC presents its draft, now named High Performance Fortran, at the Supercomputing '91 conference. Ken Kennedy and Geoffrey Fox propose an informal process, sponsored by the Center for Research on Parallel Computation, to further define and standardize the language. The major vendors, all represented, agree.

January 1992: The first High Performance Fortran Forum meeting convenes in Houston. Approximately 135 people attend. Rice University, the University of Vienna, DEC, Cray Research, Thinking Machines, IBM, and Convex make presentations. The result of the meeting is an agreement to stage a series of (much smaller) meetings to create a firm draft of the language.

March, April, June, July, September, & October 1992: The technical meetings of HPFF are held in Dallas. Extensive discussion is also carried on through electronic mail lists.

November 1992: The HPFF introduces High Performance Fortran at a workshop at Supercomputing '92.

December 1992: HPFF meets to consider public feedback from Supercomputing, and to complete the draft specification.

Goals and Scope of High Performance Fortran

The goals of High Performance Fortran are to define language extensions for Fortran supporting:

1. Data parallel programming (defined as single threaded, global name space, and loosely synchronous parallel computation)
2. Top performance on MIMD and SIMD computers with non-uniform memory access costs (while not impeding performance on other machines)
3. Code tuning for various architectures

The *FORALL* construct and several new intrinsic functions were designed primarily to meet the first goal, while the data distribution features and some other directives are targeted toward the second goal.

HPFF also adopted a number of subsidiary goals:

- Minimal deviation from other standards
- Minimal direct conflicts with Fortran 77 and 90 (if possible, no direct conflicts)
- Maximal simplicity
- Define open interfaces to other languages and programming styles
  - ✓ Provide input to future standards activities for Fortran and C
  - ✓ Encourage input from the high performance computing community through widely distributed language drafts
- Produce validation criteria
- Present final proposal in November 1992, and accept the final draft in January 1993
- Make compiler availability feasible in the near term with demonstrated performance on an HPF test suite
- Leave an evolutionary path for research

These goals were quite aggressive when they were adopted in March 1992, and led to a number of compromises in the final language. In particular, HPFF limited support for explicit MIMD computation, message-passing, and synchronization due to the difficulty of forming a consensus among the participants. We hope that future efforts will address these important issues.

Fortran 90 Binding and the HPF Subset

High Performance Fortran uses Fortran 90 as the language for extension. This was the outcome of a long and serious discussion early in the HPFF process. The working group finally decided that the new array calculation features and dynamic storage allocation make Fortran 90 a natural base for HPF. The HPF language features fall into 3 categories concerning Fortran 90:

- New directives
- New language syntax
- Language restrictions

The directives are structured comments that suggest implementation strategies or assert facts about a program to the compiler. They may affect the efficiency of the computation performed, but do not change the meaning of the program. Where possible, we designed the form of the HPF directives so that a future Fortran standard can include these features as full statements without syntactic change.

HPF introduces a few new language features, namely the *FORALL* statement and certain intrinsics. These are first-class language constructs rather than comments because they can affect the interpretation of a program, for example by returning a value used in an expression. They must therefore be direct extensions to the Fortran 90 syntax and interpretation. HPFF strove to keep this set minimal.

Full support of Fortran sequence and storage association is not compatible with the data distribution features of HPF. Therefore, HPF restricts the use of sequence and storage association in some contexts. These restrictions may in turn require insertion of directives into standard Fortran programs to preserve correct semantics.

An important goal for HPF is early compiler availability. In recognition of the fact that full Fortran 90 compilers may not be available in a timely fashion on all platforms, and that implementation of some of the HPF extensions proposed are more complex than others, a formal HPF subset has been defined. HPF users who are most concerned about multi-machine portability may choose to stay within this subset initially. This subset language includes the Fortran 90 array language, dynamic storage allocation, and long variable names, as well as the MIL-STD-1753 features which are already common in Fortran programs. The subset does not include some features of Fortran 90, such as generic functions and free source form, that are not closely related to high performance on parallel machines. It also excludes some HPF extensions, such as the multi-statement *FORALL*, which may be inessential and particularly difficult to implement.

New Features in High Performance Fortran

High Performance Fortran extends Fortran in several areas. These areas are

- Data Distribution Features
- Parallel Statements
- Extended Intrinsic Functions and HPF Library
- Local Procedures
- Parallel I/O Statements
- Changes in Sequence and Storage Association

The subsections below describe these features.

Data Distribution Features

Modern parallel and sequential architectures attain their fastest speed when the data accessed exhibits locality of reference. Often, the sequential storage order implied by Fortran 77 and Fortran 90 conflicts with the locality demanded by the architecture. To avoid this, High Performance Fortran includes features that describe the partitioning of data among memory regions. Compilers may interpret these annotations to improve storage allocation for data, subject to the constraint that all processors "see" the same value that would be computed sequentially for every array element. In all cases, users should expect the compiler to arrange the computation to minimize communication while

retaining parallelism. The "owner computes" rule (evaluate the statement on the processor owning the left-hand side) is a popular heuristic for achieving this, but is not the only possibility.

The simplest method of data partitioning in HPF is the *DISTRIBUTE* directive, which divides an object (such as an array) among processor memories. In HPF, the programmer can specify a distribution pattern for any dimension of an array. For example, given the following declarations:

```
REAL A(100,100), B(100,100), C(200)
!HPF$ DISTRIBUTE A(BLOCK,*), B(*,CYCLIC), C(BLOCK(5))
```

on a 4-processor machine, the first processor would store the following array sections:

```
A( 1:25, 1:100 )
B( 1:100, 1:97:4 )
C(1:5), C(21:25), C(41:45), C(61:65), C(81:85)
```

It is also possible to distribute multiple dimensions independently, as in the following example:

```
REAL D(8,100,100)
!HPF$ DISTRIBUTE D(*,BLOCK,CYCLIC)
```

which would store

```
D( 1:8, 1:50, 1:99:2 )
```

on a machine configured as a 2 by 2 array. (HPF allows the declaration of coarse-grain virtual processor arrays for this purpose; such arrays have no defined relation to the physical machine topology.)

A programmer can specify partial information about an object's distribution using the *ALIGN* directive. This is useful for ensuring that one array is mapped "together with" an existing array; it also provides some new ways to treat edge conditions on arrays. For example, the directive

```
REAL A(1000), B(0:1001)
!HPF$ ALIGN A(K) WITH B(K)
```

means that when *B* is distributed, *A*'s distribution will be arranged so that *A(K)* and *B(K)* will be on the same processor for all applicable *K*. If the second line were

```
!HPF$ ALIGN A(K) WITH B(K-1)
```

then *A(1)* would be located with *B(0)*, and so on. Alignments can also cause replication of array elements; for example

```
REAL C(100,100), D(100,100)
!HPF$ ALIGN C(I,J) WITH D(I,*)
```

creates a copy of row 2 of *C* on any processor that stores any element of *D(2,1:100)*. HPF limits the forms of alignment expressions to include only constant strides and offsets in each dimension of the target, effectively disallowing skewed distributions such as

```
REAL E(100,100), F(100,100)
!HPF$ ALIGN E(I,J) WITH F(MOD(I+J-1),J) <- ILLEGAL!
```

Along with the static forms described above, HPF offers two forms of dynamic data partitioning. *REDISTRIBUTE* and *REALIGN*, executable forms of the above declarations, allow arrays to change their distribution within a subroutine. Several forms of alignment for dummy arguments allow any of the following to occur in a procedure call:

- The dummy argument inherits the mapping of the actual argument.
- The dummy argument redistributes the actual argument to match an expected distribution.
- The dummy argument is declared with a particular distribution, and it is an error if the actual does not match this distribution.

The compiler must undo any redistribution of arguments before procedure return.

As this is being written, there is some controversy at the HPFF meetings regarding a construct known as the *TEMPLATE*. Essentially, this is a temporary index space used in the alignment phase of mapping. It is syntactically optional, but semantically present at a lower level in the language description. Some members of HPFF claim that this feature should be removed, arguing that it complicates the language; others vehemently disagree. The description

of effects above applies to distribution both with and without *TEMPLATE;* its effects appear in the complete description of the dynamic distribution features.

Parallel Statements

To allow the explicit expression of parallel computation, High Performance Fortran offers a new statement and a new directive. The *FORALL* construct expresses assignments to sections of arrays; it is similar in many ways to the array assignment of Fortran 90. The *INDEPENDENT* directive asserts that the statements in a particular section of code do not exhibit any sequentializing dependences; when properly used it does not change the semantics of the construct, but provides more information to the compiler.

The *FORALL* exists in two forms: the single-statement *FORALL* and the multi-statement *FORALL,* also called *block FORALL.* The single-statement version (which appeared in some Fortran 90 draft proposals prior to 1987) has the following form:

```
FORALL ( I = 2:N-1 ) A(I) = (A(I-1) + A(I+1) + A(I)) / 3
```

The semantics are that the program evaluates the right-hand sides for all index values in the range (in any order), and then assigns the computed values to the corresponding left-hand sides. For instance, the above statement averages each array element with its two nearest neighbors. HPF allows multiple indices, which are combined by Cartesian product, and a mask expression in the *FORALL* header. The block *FORALL* is similar.

```
FORALL ( I = 2:N-1 )
       A(I) = A(I-1) + A(I+1)
       B(INDX1(I),INDX2(I)) = A(I)**2
END FORALL
```

The effect is identical to a sequence of single-statement *FORALL* constructs. HPF allows nested *FORALL*s. Both forms of *FORALL* have restrictions that disallow nondeterminism, such as multiple assignments to the same location.

The *INDEPENDENT* directive is an assertion that no iteration of a *DO* or *FORALL* accesses any data that is written by another iteration. This implies that the *DO* can be executed in parallel, or that the *FORALL* can be executed without copying any array accessed. For example, the statement

```
!HPF$ INDEPENDENT
DO I = 1, 100
       A(INDX(I)) = B(I)
END DO
```

asserts that all elements of *INDX(1:100)* are unique, and therefore it is safe to parallelize the loop. If the assertion is false for some data, then the program is not standard-conforming and the compiler may take any action it deems appropriate.

Several extensions to the *INDEPENDENT* assertion are currently under discussion, including allowing reductions in a parallel construct, private variable declarations, and constructs for functional (MIMD) parallelism. As of this writing, HPFF has not adopted any of these into the language. It is possible that some will be adopted, and that others will become part of a follow-on effort to HPF in the future.

Extended Intrinsic Functions and HPF Library

Experience with massively parallel machines has identified several basic operations that are very valuable in parallel algorithm design. The Fortran 90 array intrinsics anticipated some of these, but not all. High Performance Fortran adds several classes of parallel operations as intrinsics and as standard library functions. In addition, several system inquiry functions useful for controlling parallel execution are now provided in HPF.

The list of all the new functions is too long to give here. The general categories included in HPF are as follows:

- System inquiry functions to return the number of processors and similar information.
- Distribution inquiry functions to test the distribution of an array.
- Extended forms of *MINLOC* and *MAXLOC,* giving them the multi-dimensional capabilities of *MIN* and *MAX.*

- New reduction functions, including bit-wise logical operations.
- Prefix and suffix reduction functions, for forming sets of partial accumulations.
- Combining-scatter intrinsics, which capture the functionality of the following loop:

```
DO I = 1, N
  A(INDX(I)) = A(INDX(I)) + B(I)
END DO
```

- New bit manipulation functions.
- Standard sorting functions.

As of this writing, the distribution inquiry functions are not fully defined.

A point of some debate in the HPFF meetings was whether this (rather large) set of new functions should be full Fortran intrinsics or simply a standard library. The advantages of making them intrinsics were to allow their use in declaration statements, and several implementors believed this would also aid in specifying the library. The argument against was simply that this was too much work, and cluttered the language. In the end, the HPFF committee voted to make the system inquiry functions intrinsics and put the others (except possibly the distribution intrinsics, which are not yet final) in a standard library.

Local Procedures

Because High Performance Fortran is designed as a high-level, machine-independent language, there are certain operations that are difficult or impossible to express directly. For example, many applications benefit from finely-tuned systolic communications on certain machines; HPF's global address space does not express this well. Local procedures define an explicit interface to procedures written in other paradigms, such as explicit message-passing subroutine libraries. HPF will also define a model interface for Fortran-based languages called through local routines. The details of these interfaces are not set as of this writing.

Parallel I/O Statements

By a narrow vote, the HPFF committee declined to include explicitly parallel I/O statements in High Performance Fortran. There were several reasons for this, including the possibility of providing operating system support for parallel files, the lack of a clearly portable paradigm for parallel I/O, and lack of implementation experience (particularly for portable I/O systems). In making this decision, the committee expressed the hope that a follow-on effort would add I/O features later.

Sequence and Storage Association

A goal of High Performance Fortran was to maintain compatibility with Fortran 90. Full support of Fortran sequence and storage association, however, is not compatible with HPF's goal of high performance through data distribution. Some forms of associating subprogram dummy arguments with actual values make assumptions about the sequence of values in physical memory that are incompatible with data distribution. Certain forms of *EQUIVALENCE* statements also require a modified storage association paradigm for compatibility with data distribution. In both cases, HPF provides directives to assert that full sequence and storage association for affected variables must be maintained. Without these inhibiting features, reliance on the properties of association is not allowed. An optimizing compiler may then choose to distribute any variables across processor memories to improve performance. To protect program correctness, a given implementation should provide a mechanism to ensure that all such default optimization decisions are consistent across an entire program.

Participants in the High Performance Fortran Forum

The HPF mailing lists include well over 500 recipients, of whom several dozen are active contributors. The meetings in Dallas include between 35 and 40 attendees, representing about 30 organizations. Although companies are asked to send the same representatives when possible, a few new faces show up at each meeting. The following list represents organizations that have sent a representative to at least one of these meetings:

Amoco Production Research
Applied Parallel Research
CONVEX Computer Corporation
Cornell National Supercomputer Facility
Cray Research, Inc.
DEC

Fujitsu
GMD-I1.I (Germany)
Hewlett Packard
IBM
Institute for Computer Applications in Science and Engineering (ICASE)
Intel Supercomputer Systems Division
ITI-TNO (Delft)
Lahey Computer Systems
Lawrence Livermore National Laboratory
Los Alamos National Laboratory
Louisiana State University
MasPar Computer Corporation
Meiko
nCUBE
Oregon Graduate Institute
Portland Group
Research Institute for Advanced Computer Science (RIACS)
Rice University
Schlumberger Laboratory for Computer Science
Shell Development Company
State University of New York at Buffalo
Sun Microsystems
Syracuse University
The Ohio State University
Thinking Machines Corporation
University of Vienna
Yale University

Many other companies and universities appear on the mailing lists.

The technical development of HPF is done by subgroups whose work is reviewed by the full committee. During the period of development of High Performance Fortran, many people served in positions of responsibility, including the following:

- Ken Kennedy, Convener and Meeting Chair
- Chuck Koelbel, Executive Director and Head of the *FORALL* Subgroup
- Mary Zosel, Head of the Fortran 90 and Storage Association Subgroup
- Guy Steele, Head of the Data Distribution Subgroup
- Rob Schreiber, Head of the Intrinsics Subgroup
- Bob Knighten, Head of the Parallel I/O Subgroup
- Joel Williamson and Marina Chen, Heads of the Subroutine Interface Subgroup
- David Loveman, Editor

Geoffrey Fox convened the first meeting with Ken Kennedy, and is currently leading a group to develop benchmarks for HPF. Preliminary versions of these programs are available by anonymous FTP along with the language specification. In addition, Clemens-August Thole organized a complementary group in Europe and was instrumental in making this an international effort. Space does not permit listing all the volunteers who donated time and resources to the HPFF effort, but we are very grateful for their help.

For More Information

Electronic copies of High Performance Fortran draft proposals, minutes of the working group meetings, and other related documents are available by anonymous FTP from `titan.cs.rice.edu`. The directory containing the documents is `public/HPFF` ; see the `README` file there for the current list of files and their modification dates. Copies of the current HPF language specification are also available by electronic mail from `softlib@cs.rice.edu` (send mail with "`send catalog`" in the body for a complete catalog), or David Loveman (`loveman@mpsg.enet.dec.com`) or Chuck Koelbel (`chk@cs.rice.edu`)

The most recent language specification draft is also available as Technical Report CRPC-TR 92225 from the Center for Research on Parallel Computation at Rice University. Send requests to Theresa Chapman; CITI/CRPC, Box 1892; Rice University; Houston, TX 77251. There is a charge of $4.00 for this report, to cover copying and mailing costs.

A more active way to participate is to join one or more of the HPFF mailing lists. The possible lists are

`hpff`	Main HPFF list (meeting minutes, pointers to language drafts, miscellaneous announcements)

`hpff-f90`	Discussion group for the relationship between HPF and Fortran 90
`hpff-distribute`	Discussion group for models of data and work distribution
`hpff-forall`	Discussion group for *FORALL* statements, local subroutines, and explicit process control
`hpff-io`	Discussion of parallel I/O and other miscellaneous matters
`hpff-intrinsics`	Discussion of new intrinsic functions useful for parallel programming

All mailing lists are kept at `cs.rice.edu`. To add your mail address to one of the HPFF mailing lists, mail a message to `hpff-request@cs.rice.edu` with a subject line containing the word " `add` " and the name of the list. This will add the E-mail address of the message originator to the requested mailing list (if no list name is given, the address is added to " `hpff` "). For example, to receive the Fortran 90 discussions, send a message with " `Subject: add hpff-f90` ".

The High Performance Fortran Forum welcomes comments on any of its work. Send remarks to any of the following:

The mailing list discussing that topic (for example, to report an apparent inconsistency in the *ALIGN* statement, send to `hpff-distribute`).

The author of the appropriate section in the language specification (available by anonymous FTP as explained above; authors are clearly identified).

The core group (`hpff-core`) is usually helpful for general policy questions.

Chuck Koelbel (`chk@cs.rice.edu`) is serving as Executive Director of HPFF; he is usually good at answering mail or fixing problems with the FTP archives.

Bibliography

1. Callahan and K. Kennedy, "Compiling Progams for Distributed-Memory Multiprocessors", *Journal of Supercomputing*, Vol. 2, October, 1988, pp. 151--169.

2. Gerndt and H. Zima, "SUPERB: Experiences and Future Research", in *Languages, Compilers, and Run-Time Environments for Distributed Memory Machines*, J. Saltz and P. Mehrotra, editors, North-Holland, 1992.

3. High Performance Fortran Forum, "High PerformanceFortran Language Specification, version 0.2", Center for Research on Parallel Computation, Rice University, *Technical Report CRPC-TR92225*, Houston, TX September, 1992.

4. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer and C. Tseng, "An Overview of the Fortran D Programming System", in *Languages and Compilers for Parallel Computing, Fourth International Workshop*, U. Banerjee, D. Gelernter, A. Nicolau and D. Padua, editors, Springer Verlag, 1991.

5. Koelbel and P. Mehrotra, "Programming Data Parallel Algorithms On Distributed Memory Machines Using Kali", in *Proceedings of the 1991 ACM International Conference on Supercomputing*, Cologne, Germany June, 1991.

6. Li and M. Chen, "Compiling Communication-Efficient Programs for Massively Parallel Machines, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 3, pp. 361--376, July, 1991.

7. Rogers and K. Pingali, "Process Decomposition Through Locality of Reference", in *Proceedings of the SIGPLAN '89 Conference on Program Language Design and Implementation*, Portland, OR June, 1989.

8. Zima, P. Brezany, B. Chapman, P. Mehrotra and A. Schwald, "Vienna Fortran --- A Language Specification, Version 1.1", *ICASE Interim Report 21*, Hampton, VA March, 1992.