Conf-9302135--1

UCRL-JC-112573
PREPRINTRECEIVED
JUL 19 1993
OSTI

High Performance Fortran: An Overview

Mary E. Zosel

This paper was prepared for submittal to
IEEE COMPCON Spring '93
San Francisco, CA
February 22-26, 1993

December 23, 1992

Lawrence
Livermore
National
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

High Performance Fortran: An Overview

Mary E. Zosel

Lawrence Livermore National Laboratory

Abstract

The purpose of this paper is to give an overview of the work of the High Performance Fortran Forum (HPFF). This group of industry, academic, and user representatives has been meeting to define a set of extensions for Fortran dedicated to the special problems posed by very high performance computers, especially the new generation of parallel computers. The paper describes the HPFF effort and its goals and gives a brief description of the functionality of High Performance Fortran (HPF).

1: Background

From the first description of non-uniform memory access multiprocessors, it was clear that there was a new problem facing compiler optimization and operating system process scheduling. The relationship between the location of data arrays in memory and the processor needing access to that data would be a key element in program efficiency. The same problem is even more obvious in distributed memory parallel processors where message passing is required when data is not local. When the application designer modifies code to address explicit memory optimization, experience shows that the complexity of the code increases dramatically.[1] The problem for the designer is compounded because different optimizations are needed for each different parallel architecture.

Ken Kennedy's description of this portability problem at a panel session of the Supercomputing 90 conference and the subsequent description of the Rice Fortran D project[2] struck an immediate response around the country. Fortran D was not the only research effort addressing this problem, but was one of the most visible. By the time of the SC91 conference in November 91, there was already pressure to standardize the language associated with this research project. At a January 92 meeting sponsored by Rice University, David Loveman, Digital Equipment Corporation, presented a proposal for "High Performance Fortran" (HPF). This was followed by related presentations from other vendors and universities. The level of interest was apparent from the attendance of well

over 100 people from about 60 organizations. The decision was made to convene a small working group to defining a language specification for HPF.

2: The creation of HPF

An important part of the HPF development is related to the process used to produce the draft for the first version. Language design by committee has enjoyed only limited success, and the formal standards process has the reputation of being very slow. The very first meeting of the HPFF working group covered important ground in agreeing on the goals for the language, the process, and the base language framework. Each of these are described here.

2.1: HPF language goals

A specific list of goals for HPF was created. The full list is found in the first chapter of the HPF document.[3] Key items from the list were:

- support *data parallel* programming (single thread, global name space, loosely synchronous parallel computation);
- address top performance on both MIMD and SIMD non-uniform memory access machines;
- aim for "rewrite once" for users;
- support code tuning on various architectures;
- minimize conflicts with Fortran standards;
- plan for near term compiler availability;
- leave an evolutionary path for research; and
- finish the process in a year.

2.2: HPFF process

Meetings of the working group varied between 30 and 40 people, representing up to 30 organizations and were held approximately every six weeks. Subgroups were defined to address the specific issues needed for the language specification. Major language features were proposed in two stages: a first reading with straw polls taken for direction; and a second reading with formal

organization votes for formal inclusion in the document. Substantial work was done over email between meetings. The central motivation that kept the process moving smoothly was the goal of having a specification by the end of the year.

2.3: Language specification basis

The third area discussed in detail and settled at the first meeting was the language basis for the specification. At the January meeting, specific user interest in using Fortran 90 was expressed. After discussion, it was agreed that HPF would be based on full Fortran 90[4] with a subset defined to aid rapid availability. Further debate settled the issue of including some elements of Fortran 90, especially the array sublanguage in the HPF subset.

The other important issue related to the base language came to consensus quickly: where possible, the HPF extensions would be defined as comment-directives, to leave a functional Fortran program when the directives were ignored. It was further agreed that the directives should be considered advice to the compiler, rather than a binding order about implementation.

3: Language overview

In addition to the parallelism of the Fortran 90 array language, HPFF identified four areas of concentration to enhance application performance: explicit data distribution, a FORALL construct, new intrinsics and subroutines, and an interface for localized computation. An important area discussed, but *not* included in the first version of HPF, was language extensions to support I/O on parallel machines. This was omitted because of a combination of architectural variation and lack of a clear understanding of what applications specifically needed from a language rather than the operating system.

3.1 Data mapping features

The primary focus of HPF is on the new directives to control data layout across the memory components of a parallel system. Three kinds of data mapping functionality are recognized: (1) the relation of data objects with respect to each other (alignment), (2) the layout of the data on a virtual processor grid (distribution), and (3) the ability to describe the "shape" of the actual processor grid (processor view). The term *mapping* is used to describe the combination of these features. Use of the mapping directives is optional. The user need not spell out every detail of a mapping, but any information provided to the compiler potentially helps the performance of the generated code.

The intent of the alignment directive ALIGN is to provide a mechanism for indicating that arrays are commonly accessed together with a certain relationship between the indexes. This is used to ensure that these index patterns are assigned to the same processors in order to reduce communication overhead. The various options of the alignment directive allow the user to indicate that indexes be mapped:

- identically (e.g. A[I] with B[I]),
- with an offset (e.g. A[I] with B[I-1]),
- with a stride (e.g. A[I] with B[2*I]), or
- with a dimension embedded (e.g. A[I] with B[I,*]).

The strides and offsets must be expressed as a linear multiple of the index variable with a constant offset. In the case the a group of arrays are commonly used together, a shorthand is provided to express the relationship. A template may be defined which is an abstract (no storage allocated) set of index positions. The purpose of a template is to describe an overall size and shape of a group of related arrays, when no one array exactly matches the entire index space. The individual arrays are then aligned to the template which can, in turn, be mapped to the processors. Alignment and the use of a template should not be confused with the Fortran concept of equivalence. They indicate how arrays are mapped to common processors, but not to common memory locations.

Examples:

```
!Layout A and B on same processors
!HPF$ ALIGN A WITH B
!Each row of X aligned with Y
!HPF$ ALIGN X(I,*) WITH Y(I)
!Use of a template
!HPF$ TEMPLATE T(1000,1000)
!HPF$ ALIGN WITH T(*,I) :: Q(I), R(I-1)
```

The DISTRIBUTE directive is used to specify how arrays or templates are to be distributed over the processors. A BLOCK distribution in a given dimension divides the index range into equal chunks across the processors (with allowance for the end case). CYCLIC distributes the elements in a round-robin fashion, one at a time. Both BLOCK(n) and CYCLE(n) distribute with a specific chunk size where the resulting distributions are the same as long as there are more processors than the dimension of the array. When DISTRIBUTE BLOCK is used with a template, arrays which are smaller than the total template dimension may possibly be spread over only a subset of the available processors.

Examples:

```
!Map Z cyclic by rows, columns together
!HPF$ DISTRIBUTE Z (*, CYCLIC)
```

```
!A single distribute for multiple
!arrays aligned to a template.
!HPF$ TEMPLATE T1(10000)
!HPF$ ALIGN WITH T1:: A,B,C
!HPF$ DISTRIBUTE (BLOCK)::T1
```

So far, no mention has been made of how the processors are allocated. For a rank-one array there is no issue, but consider the case of two dimensional arrays using a BLOCK-BLOCK distribution: how many processors should be assigned to each dimension? A square grid of processors may not be a natural fit for a rectangular array. A PROCESSORS directive is provided to allow the user to describe a processor arrangement that best fits the problem. Vendor directives may further bind the processors in architecture-dependent ways. An additional directive called VIEW provides a mechanism to define names for different shapes or subsets of PROCESSORS.

Example:

```
!HPF$ PROCESSORS P(4,16,)
!HPF$ DISTRIBUTE X(BLOCK, BLOCK) ONTO P
```

The ALIGN and DISTRIBUTE directives define an initial mapping for data objects, but for some applications it may be the case that the ideal layout changes during the course of the execution as different phases of a calculation are encountered. For this purpose, HPF defines a DYNAMIC directive to specify that a data object may change mapping attributes. The REALIGN and REDISTRIBUTE directives are provided to specify when the change should take place. Code to handle variables declared with such dynamic mappings may be complicated unless a compiler can prove that only a single mapping can reach a given statement. There is also the cost of moving the entire array to a new mapping. These costs must be weighed against the penalty of a less-than-optimal mapping to decide when to use this extra functionality. Arrays that are defined in COMMON blocks may not be declared dynamic, so this feature applies only to local arrays and to variables that are declared in a Fortran 90 MODULE.

There is one more important additional case to consider where a data object may require a runtime mapping change: the subroutine interface. HPF recognizes four cases the user may wish to express:

- the dummy argument should inherit the mapping from the actual argument;
- the dummy argument should inherit the mapping of the actual argument and that mapping is a specific mapping;

- the dummy argument should be coerced to a specific mapping; or
- nothing is said about the mapping.

There are additional distinctions about referring to the processor arrangement of the actual argument and a distinction when an actual argument is a slice of an array instead of a full array. The details of these options are still being finalized and a full description is complicated. The goal is that the functionality the user requires for different cases will exist with a relatively simple notation, without requiring knowledge of all of the options. HPF will take advantage of Fortran 90 procedure interface descriptions, either from MODULE definitions or INTERFACE blocks to allow a user to specify to a calling routine details about the mapping expectations of the dummy arguments so that the best code can be generated.

Before leaving the subject of data mapping directives, a note about the syntax of the directives is in order. Similar to Fortran 90 declarations, two forms of directive syntax are defined. They can be given in statement (FORTRAN 77) style, with a single attribute followed by a list of names, or they may use the combined style of Fortran 90 where multiple attributes are specified on a single line with the variable names following a double-colon. Examples above use both forms.

3.2 Other HPF features

HPF defines the addition of a FORALL statement to the base language. The syntax for the statement is taken from early versions proposed for Fortran 90, but not adopted in the final standard. The assignment statement associated with the FORALL is executed for all combinations of a set of index ranges given. In simple cases, it is similar to writing multiple nested DO statements on a single line. The FORALL construct extends the functionality to operate over a block of assignment statements. HPF strictly limits the kind of subprograms that can be called from within a FORALL to avoid side effects which would raise potential for differing answers under parallel execution.

An INDEPENDENT directive is provided for optimization of loop constructs. It applies both to standard DO loops and to FORALL constructs, asserting independence of the iterations.

HPF defines a set of library and intrinsic procedures. The library procedures are modeled very closely on the intrinsics provided in Thinking Machine's CM Fortran. [5] They include support for scatter/gather operations, broadcasts, location of minimum and maximum elements, etc. The HPF intrinsics are defined to allow query of the size and shape of the processor array assigned to an

execution. They can be useful in the declaration of array sizes.

Finally, an interface to EXTRINSIC procedures is specified in HPF to allow use of individual processors in a local context rather than the data parallel model. As part of the interface, an additional set of intrinsics is defined to query the layout attributes of an array.

It should be noted that the HPF data mapping features in the previous section all are based on comment directives and can be completely ignored in a serial execution. However, most of the features in this section appear directly in the syntax and require language or compiler support.

3.3 The HPF subset

A subset of HPF has been defined to encourage quick release of compilers for user experimentation. The subset includes only the Fortran 90 features directly needed for HPF and does not require some of the more complex features of HPF extensions. Primarily, the subset includes the mil-spec extensions to FORTRAN 77 as common practice and adds the array language, allocatable arrays, and intrinsics. It does not contain provision for HPF dynamic mapping (e.g., the REDISTRIBUTE and REALIGN directives), limits the form of FORALL and extrinsics that support the a more general MIMD style parallelism. The HPF library is also not required in the subset simply because of the large number of routines involved.

4: Relation to standards

There are several issues relating the Fortran standards and HPF. Some of the important questions are addressed here.

Can standard Fortran code be compiled by an HPF compiler? The answer to this important portability issue is both yes and no. A design goal for HPF was to allow the compiler as much leeway as possible for optimization when no user directives about distribution are given, such as might happen when a standard Fortran code is compiled. No mapping directives are required by the language. It is the case, however, that many, if not most, Fortran codes depend on properties of storage and sequence association that apply to variables in COMMON and EQUIVALENCE, as well as to actual procedure arguments that differ in exact size and shape from associated dummy arguments. HPF data mapping is not defined for variables that depend on these properties. An HPF directive (SEQUENCE) to declare the *sequential* property is required for such variables or COMMON blocks. There are three general rules of thumb:

- COMMON blocks require SEQUENCE directives unless they are "the same" everywhere;
- equivalenced variables can only be explicitly mapped if there is a rank-one variable which completely covers the equivalence group;
- procedure arguments which do not match exactly in size and shape require SEQUENCE directives.

To facilitate porting old libraries there is a directive form to declare that all variables in a procedure should be considered sequential.

The second standards-related question is: can HPF code execute correctly under a standard Fortran compiler? This is an important issue for applications that need to run on workstations which do not support HPF. This depends on the application, but portability should be straightforward when required. Obviously, the application cannot use the FORALL syntax and cannot call HPF library routines, unless there is a version on the workstations. The application must further limit itself to use of the standard Fortran features supported on that workstation, e.g., if there is no Fortran 90 compiler, the application must avoid array syntax.

Finally, HPF is already often called a "standard". This is true only in the informal, *defacto* standard sense of the word. It is not a formal standard. The working group has had an open exchange of information with ANSI X3J3 and has sought to make its work upwardly compatible with the Fortran 90 standard. It is the hope of HPFF that X3J3 will consider the successful features of HPF as candidates for direct inclusion in a new release of a formal Fortran standard.

5: What is next for HPFF?

The most important step for HPF at this time is implementation and experience. Since several vendors plan to release initial versions during 93, HPFF has decided to postpone consideration of any new features for HPF until 1994. Support for irregular meshes, parallel I/O and nested WHERE statements are examples of functionality that might be considered for a later version of HPF. The appendix of the HPF document includes descriptions of some of these features.

6: Summary

It is too early to evaluate HPF implementations in order to gauge success and impact of the language extensions. The story, at the current time, is in the process of the language definition. The HPFF effort was characterized by cooperation, hard work, and agreement. There was consensus about many issues, and where votes were close, there was a willingness to abide by majority

decision. The resulting specification document shows the commitment of many organizations.

7: Acknowledgments

Too many people participated in the HPF design process to name them all. The people who contributed text to the document included: A. Choudhary, G. Fox, T. Haupt, M. Hoffert, K. Kennedy, R. Knighten, C. Koelbel, D. Loveman, P. Mehrotra, J. Merlin, T. Ngai, R. Page, S. Ranka, R. Schreiber, R. Shapiro, M. Snir, M. Snyder, G. Steele, R. Swift, M. Wu, and M. Zosel. In addition the list would not be complete without special reference to Dave Loveman as the document editor, Chuck Koelbel as the HPFF executive director, and Ken Kennedy for his leadership of HPFF.

The author's involvement in HPFF and preparation of this paper were performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

1: Silvio Picano, Eugene Brooks, and Joseph Hoag. *Programming Costs of Explicitly Memory Localization on a Large Scale Shared Memory Multiprocessor*, In: The 1991 MPCI Yearly Report: The Attack of the Killer Micros, UCRL-ID-107022, Lawrence Livermore National Laboratory, Livermore, California, March 1991.

2: Geoffrey Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, and Min-You Wu. *Fortran D Language Specification*. Report COMP TR90-141 (Rice) and SCCS-42x (Syracuse), Department of Computer Science, Rice University, Houston, Texas, and Syracuse Center for Computational Science, Syracuse University, Syracuse, New York, April 1991

3: HPFF, Draft High Performance Fortran Language Specification, CRPC Technical Report #92225 (Rice), Department of Computer Science, Rice University, Houston, Texas, Nov. 1992.

4: ISO. *Fortran 90*, May 1991. [ISO/IEC 1539: 1991 (E) and now ANSI X3.198-1992].

5 Thinking Machines Corporation, Cambridge, Massachusetts. *CM Fortran Reference Manual*, July 1991.

END

**DATE
FILMED**

9/17/93