HPFF Meeting Notes for the January 27-28, 1992 Meeting

High Performance Fortran Forum

CRPC-TR93310
May 1993

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Notes from the First High Performance Fortran Meeting

Houston Plaza Hilton, Houston, TX
January 27-28, 1992

Notes taken by Charles Koelbel
With contributions from Geoffrey Fox, Guy Steele

Overview

The High Performance Fortran Forum is a working group convened by Ken Kennedy and Geoffrey Fox to create an informal standard for Fortran extensions aimed at data parallel computation. The meeting lasted for one and a half days. The first day was characterized by Kennedy as "fact finding"; companies and university research groups made presentations of several proposals for possible sets of extensions. The second day was primarily concerned with planning future directions for the HPFF, in particular creating a smaller working group to create a unified proposal. Some 10-minute research presentations were also made on the second day. This report gives an overview of the first day's talks, particularly audience reactions, and reports on the second day's business meeting. (Relatively little technical detail is included because PostScript versions of most of the full proposals is available via anonymous FTP; that source should be used for detailed comments. Another summary will offer point-by-point comparisons of the proposals.) Remarks on the short presentations on the second day are not included because I was not present; if somebody wants to write a short summary of them, I will be happy to incorporate it in a future edition of the minutes.

HPF Proposal (presented by David Loveman, Digital Equipment Corporation)

Although this proposal lent its name to the meeting, it was not presented as having any special standing. Rather, it was primarily an overview of what a new set of Fortran extensions should do, and the

tradeoffs involved in choosing them. The major features of the DEC HPF language were presented after that discussion.

In the credit-where-credit-is-due department, the HPF draft was originally written by employees of COMPASS, working under contract to DEC to define a technology for parallel programming. The draft has been revised significantly since the demise of COMPASS.

The major objectives that Loveman put forward were

1. Hardware independence
2. A comprehensive language (meaning that the language should be complete for the type of problems it was meant to solve)
3. Support for standards (meaning that existing formal and de facto standards should be fully supported)
4. Phased implementation (meaning that incremental adding of features would be encouraged, rather than forcing an all-or-nothing compiler effort)

DEC's assumptions in making tradeoffs (mainly between language completeness, implementation constraints, and other costs) included

1. "Dusty programmers, not dusty decks" (phrasing attributed to Bert Halstead - meaning that programs would still have to be rewritten to take advantage of new machine architectures and/or language extensions, but the underlying language would still be Fortran)
2. Array-based parallelism (and possibly also support for Geoffrey Fox's "embarrassingly parallel" problem class, in which no communication between tasks is needed)
3. Environment support (debuggers, performance estimators, annotated listings, etc. to help users make sense of what their programs are doing)

This analysis led to a set of design issues which were presented as orthogonal:

1. Base language for the extensions (DEC proposed full Fortran 90 - it is notable that they were the only group making this a requirement)
2. I/O features (DEC proposed staying away from parallel I/O since there is not widespread consensus yet)
3. Intrinsics (DEC proposed a new category of intrinsics -system inquiry functions - initially containing one member NUMBER_OF_PROCESSORS)
4. Directives (DEC proposed all extensions be in the form of structured comments which would not change the program result if assertions mad by the directives

were true. Such comments could then be ignored by non-HPF compilers.

Vendor-specific extensions were also allowed in the same style.)

The full HPF draft was then presented. The briefest possible description of it would be "Full Fortran 90 + static Fortran D data distributions + single-statement FORALLs" (see the actual proposal for more information). Some comments from this part of the presentation (Q=audience question, A=Loveman answer, X=audience comment without answer):

Q: Can NUMBER_OF_PROCESSORS() be used as a constant?

A: It is a constant if the module is compiled with a fixed number of processors, and a specification expression (see Fortran 90 standard) otherwise.

X: (in various phrasings, from several people) The programmer should specify the processor layout and make the compiler map that onto the physical layout, rather than the other way around (as HPF now does)

Q: Are replicated values (allowed via ALIGN statements) consistent? (If yes, does this mean there is no private storage per processor?)

A: Yes, but the compiler may violate this consistency if it can prove it is safe to do so. There is no concept of private storage, although compiler should be smart enough to notice some cases (like loop induction variables).

Q: Are the data distribution directives absolute rules, or can the compiler override them?

A: In general, the compiler chooses distributions as it sees fit, and the directives are "hints". So we lean toward no override (but the compiler should provide feedback when it sees a better way).

MPP Proposal (presented by Tom MacDonald, Cray Research Inc.)

The proposal from Cray was presented as the programming model offered on that company's Massively Parallel Processor machine. Although it was not presented explicitly as a industry-wide proposal, it was clearly an indication of the way that a major vendor was headed.

The programming model for the MPP has two major aspects: data sharing and code sharing. These are treated essentially orthogonally; it is perfectly possible to distribute an array in contiguous blocks, and spread the iterations of a loop accessing that array cyclically

among processors. The Cray philosophy was announced as "small extensions", but the programming model was certainly further removed from sequential Fortran than DEC's proposal. As in the DEC proposal, directives were represented as comments, although some comments did change the meaning of the program. The MPP programming model uses CFT77 as its base programming language.

In the area of data sharing, the situation can be described as follows:

1. Arrays are private to each processor by default (and the copies of the arrays on each processor may have different values).
2. COMMON blocks and local arrays can be marked simply as SHARED, giving them a canonical distribution (essentially, cache lines distributed cyclically among processors). Sequence and storage association are fully supported for these objects.
3. Dimensional distributions (BLOCK and BLOCK(N)) can be applied to each array dimension, providing the same functionality as the DEC proposal (except there is no alignment). Consistent declarations are needed if these arrays are passed (COMMONs redeclared) in other scopes.
4. Subprograms can either declare the distribution of their arguments or mark them as UNKNOWN, which will compile into more general but slower code.
5. Limited forms of redistribution are allowed.

As to work sharing, the situation is more complex:

1. Serial and parallel regions (ala the PCF and X3H5 constructs) are supported. The program begins execution in a serial region.
2. Within parallel regions, loop nests can be marked as shared or private. Processors divide the iterations in a shared nest between themselves (by one of several mechanisms), while each processor executes all iterations of a private loop.
3. A rich set of synchronization primitives is supported, including barriers, locks, events, and critical sections.
4. There is also a rich set of new intrinsics for working with local array sections within parallel sections (MY$HIIDX for the upper bound of the local section, etc.)
5. Intrinsic functions are provided for reductions, prefix operations, and segmented scan operations. Complex

semantics are needed for the various combinations of shared and private array parameters to and results from these.

Parallel I/O was mentioned only in passing in the talk, although there is a section on it in the MPP manual. In essence, I/O can be shared (i.e. all processors read the same file) or private (every processor has its own file). Values read are automatically distributed to the correct processor(s) by the underlying system.

Some of the audience comments included:

X: Rounding block sizes to powers of 2 (done to facilitate subscript generation) is a bad move.

Q: (at several points, for various reasons) Isn't that nondeterministic?

A: (usually) Yes. Many of these cases can be checked in the compiler or on-the-fly by inserted code; we'll develop an environment to do that.

X: This may not be a good thing to have the language lean on so heavily.

X: (Guy Steele) Note that "nondeterministic" in theory circles means "the automaton always makes the right choice"; what these features offer is "indeterminate" execution. A semantic difference worth preserving.

Q: Can the program check what the actual distribution of an array parameter is?

A: Yes, but only indirectly.

Fortran D (presented by Chuck Koelbel, Rice University)

This talk presented only the outlines of the Fortran D language. Most of the emphasis was placed on making the final HPF language extensible (for language researchers), rather than pushing advanced features as required parts of the language. Either Fortran 77 or Fortran 90 can be used as the base language.

Some of the points raised were

1. The DECOMPOSITION represents the finest-grain level of parallelism of a program, which is mostly machine-independent. Having this level explicit for the programmer was thought to promote portable code.
2. The current ALIGN syntax admits certain pathological cases that the Fortran D group did not intend to plague implementors (Guy Steele gave the example of ALIGN A(I) WITH D(INT(LOG(FLOAT(I))))). On the other hand, there are real problems that need highly irregular alignments, such as finite element codes.

Therefore, we want whatever syntax is accepted to be extensible to these cases (although we cannot recommend that the full syntax be implemented everywhere).

3. DISTRIBUTE represents the machine-dependent level of parallelism. This is perhaps more naturally expressed by an explicit processor array like several other proposals have.
4. Irregular DISTRIBUTIONs are useful for some problems, and provide a different conceptual functionality from irregular ALIGNs. (ALIGN concerns connectivity patterns between different arrays, while DISTRIBUTE handles reference patterns within the same array.) Again, an extensible syntax is recommended.
5. Fortran D has a determinant multi-statement FORALL loop, and we think both those qualities are desirable in a parallel loop. If users really want indeterminate execution, we suggest providing another construct in addition.
6. Array distributions should be propagated automatically to procedures, and the compilation system be responsible for sorting out the actual inputs and generating correct code. This may be overly ambitious, since Rice has more interprocedural analysis experience than many other places.

Vienna Fortran (presented by Hans Zima, University of Vienna)

Vienna Fortran was presented as a complete proposal, although more emphasis was placed on the parts of the language which differed significantly from the preceding talks. Like Fortran D, Vienna Fortran will eventually have Fortran 77 and Fortran 90-based versions.

The key issues in Vienna Fortran (as opposed to other proposals) were

1. Explicit physical processors with user-definable topology. A default (linear) processor array is always defined, but the programmer can also define other structures.
2. No virtual processor mechanism (ala Fortran D DECOMPOSITION) is provided. It was not considered necessary.

3. Alignment is provided between arrays directly, rather than between arrays and (physical or real) processors. Element-wise alignment is equivalent to the Fortran D ALIGN, and a different concept of structural alignment is also provided (useful between different-sized arrays).
4. Both static and dynamic distributions are provided, but different syntax is used for them. Either static or dynamic mappings can use regular or INDIRECT (user-defined) distributions. Dynamically distributed arrays can also be annotated with their set of possible patterns; this may simplify compiler analysis.
5. A variety of parameter passing mechanisms are provided, by which the programmer can control fairly precisely whether an array is redistributed on entry to the subprogram or not.
6. A number of distribution inquiry functions are also provided, which can be checked at run-time (particularly useful for parameters).

Data Parallel Fortran (presented by Jorge Sanz, IBM Almaden)

This talk was a much more theoretical offering than the other presentations. Sanz presented some recent work at IBM Almaden aimed at evaluating the expressiveness of parallel languages. As he said in his disclaimer "I'm going to work on anything I say shortly, BUT this doesn't mean that IBM will."

Data Parallel Fortran (DPF) considers the expressiveness of parallel languages. In particular, it attempts to answer the question "Is there a safe, portable way to program parallel machines?" The starting point for this research is looking at sequential and parallel languages as two endpoints of a spectrum. Most proposals for new languages are movements from the sequential side towards parallelism; DPF moves from parallelism towards sequential languages. The hope is that this approach will show limits of expressiveness more explicitly than other languages.

The complete model is too complex to fully describe here. (Interested readers are referred to the author, who is working on a complete specification.) Its major components can be summarized as

1. The underlying model is an unbounded number of recursively nested virtual processors. Each processor has a local memory which can be initialized (or can

later write to) a global store. Such global accesses are presumed to be expensive, and must be made explicitly.

2. MAP functions are provided to move (distribute) global data to the local memories. Once a MAP has been performed, the processor may only access its own data.
3. FORALL statements specify parallel execution. Since no off-processor accesses are allowed within a FORALL, execution is determinant. Communication and synchronization are performed at FORALL entrance and exit.

Comments on the HPF Draft (presented by Guy Steele, Thinking Machines Corporation)

This presentation gave constructive criticism of the HPF draft (actually, a somewhat older draft than was presented earlier) in four areas: Fortran 90 compatibility, data alignment and distribution, SPMD programming support, and miscellaneous features. The talk was generally very well received.

In the area of Fortran 90 compliance, Thinking Machines believes that requiring full Fortran 90 as part of HPF will hinder (short-term) acceptance. They propose only requiring a subset at first, and moving to full Fortran 90 at a later date. The subset they propose is essentially the new array operators and intrinsics, allocatable arrays, and the WHERE statement.

In the area of data alignment, Guy characterized Fortran D's ALIGN as too general and HPF's restrictions as too limited. He proposed a new syntax that would allow dynamic alignments, but only a simple subset of HPF's ALIGNment patterns (essentially affine translations of arrays). The syntax had the key advantage that alignment functions were guaranteed to be invertible (required for efficient implementation), but the disadvantage that it was not extensible (even transpose could not be supported). Potential users in the audience seemed satisfied with this tradeoff, although language researchers seemed less enthusiastic. A PROCESSORS declaration was also proposed, leading to an interesting (and unresolved) debate on how many levels of mappings were sufficient and necessary.

In the area of SPMD programming support, Guy suggested LOCAL code sections containing code to be executed on each processor. In contrast to the Cray parallel sections, no off-processor references could be made in a LOCAL section (except through a complex syntax).

Arrays would be referenced relative to their local sections, rather than by global indices. This proposal generated a great deal of comment from the audience, mostly at least generally favorable on the grounds that LOCAL sections provided low level control. On the down side, however, it was observed that LOCAL sections changed the semantics of programs (for example, array indexing changed), and that the feasibility of LOCAL sections was tied closely to the generality allowed in ALIGN statements.

Because of time constraints, the discussion of miscellaneous features was cut short. The features proposed (parallel prefix operations, for example) did not appear controversial, however.

Views on the HPF Drafts (presented by Joel Williamson, Convex Computer)

This talk was in much the same spirit as the Thinking Machines talk (i.e. constructive criticism), but did not propose any new features. Joel did, however, have one of the more noteworthy quotes of the day, regarding shooting holes in proposals: "I'm well aware that the ducks in this gallery are armed with automatic weapons."

Some of the comments included

1. The Thinking Machines proposal was generally seen as the best of the lot, although there was still room for improvement (particularly for explicit synchronization).
2. Real-world criteria were needed for validating the proposal, in particular rewriting significant scientific codes (such as the PERFECT club benchmarks). This was followed by a lively debate on the difficulty of choosing ALIGN and DISTRIBUTE statements for a shallow water model benchmark, Joel defending the "It's hard" viewpoint.
3. Requiring full Fortran 90 was not a popular idea because of that language's complexity.
4. Extending the HPFF extensions from Fortran to C is something that Convex feels must be done (by someone).

Business Meeting (chaired by Ken Kennedy)

The business meeting started with a short discussion by Ken Kennedy summarizing the proposals made the day before. He then presented a list of suggested issues, principles, and procedures for future development of a High Performance Fortran standard. There

was then a lively discussion, which resulted in a consensus for going ahead with a standard drafting process. Below are Ken's lists, a very brief list of important points made in the discussion, and a summary of the future of HPFF.

Proposed features:

From DEC's HPF proposal:

- All of Fortran 90 (plus some other standards)
- Static data distributions
- Single-statement FORALL

From others:

- Block FORALL statements
- Parallel loops (ala PCF)
- Dynamic redistributions
- Local procedures
- Irregular data distributions

Issues for discussion:

- Minimum vs. Maximum Standards (Time to implement vs. comprehensiveness)
- Data Distribution (Dynamic? Is ALIGN necessary?)
- Parallel Loops (Block FORALL? PCF DOALL? Synchronization between iterations?)
- Subroutines (Redistribution on input and/or output?)
- Porting vs. Programming (Will users rewrite?)
- Directive Strategy (Can non-parallel machines really ignore them?)
- Test Suite
- Compatibility with Advanced Features (of Fortran 90 and other dialects)

Principles for HPFF:

- Maximum Compatibility (with existing practice and standards)
- Minimum Requirements (for first prototypes)
- Machine Independence
- Leave out features that give major advantages/disadvantages to specific machine types
- Consider near-term first
- Leave an evolutionary path (for research)

Procedures for HPFF:

- One-year project (finished draft by Christmas '92)
- People committed to multiple meetings (to ensure continuity)
- One person per organization (plus a named alternate)
- Meeting costs

Materials fee for copying (could be avoided by use of anonymous FTP)

Meeting fee for actual attendees

Test suite built separately (but coordinated activity)

Major discussion topics

(Speakers are identified, but remarks are paraphrased. Not all comments are recorded, but I hope I've gotten the most important ones.):

Q: What exactly are the goals?

Ken Kennedy: Develop a portable language that can be compiled for high performance on a variety of parallel machines

Rich Shapiro: Users need to know the features NOW, for planning future codes

Q: How do we make this standard work (unlike certain other languages that were mentioned in negative lights)?

Ken Kennedy: Get the vendors to agree, thus forming a de facto standard

Clemens-August Thole: Lots of groups are already working, try to unify them

Dave Presberg: Get coordinated fast, hold that momentum

Peter Belmont: Draft will get done faster with a strong leader and willing helpers, rather than a true democracy (Ada used as an example)

Bob Metzger: Build the test suite, let that serve as the standard

An informal survey drew about 20 vendors (and other parties) who were willing to commit to the process outlined above. This was deemed a good size for a working group (i.e. small enough that real work could be accomplished in reasonable time, unlike the conditions at the first meeting). Chuck Koelbel collected names to coordinate the next meeting, which would focus on producing a clear statement of goals and principles. With that statement in hand, it was expected that work on language design could be divided more rationally. Current plans are to have the next meeting in Dallas toward the end of February. Mail lists for meeting attendees and for other interested parties will be set up in the very near future. It is hoped that future language drafts will be available by anonymous FTP (in raw ASCII form, to reduce difficulties in printing). Geoffrey Fox is coordinating the collection of the test suite.

HPF Benchmark Suite (presented by Geoffrey Fox)

Geoffrey Fox proposed the establishment of a HPFF application suite which would be "experimental data" with which to test and evaluate language design. Eventually it could evolve into a test suite to certify the compiler. He contrasted this with the existing Fortran 77, 90D test suite where we are developing at least 6 versions of each code:

1. Fortran 77 Original
2. Fortran 77 + Message Passing (Express or Intel)
3. CM FORTRAN
4. Fortran 77 written so it can be parallelized (possible language and/or algorithm changes)
5. Fortran 77D
6. Fortran 90D

The HPFF application suite would record for each application in it:

1. Source of and reason for application being present
2. Fortran 77 OR Fortran 90 original
3. At least one good parallel version e.g. Fortran 77+MP and/or CM Fortran and/or MasPar Fortran ...
4. If necessary the parallelizeable Fortran 77 or Fortran 90 version (as in item 4 of Fortran 77,90D benchmark suite)
5. One or more HPFF versions written in different proposed dialects.

There seemed to be a good deal of enthusiasm at the meeting for this project, particularly in view of earlier remarks regarding having real examples. Please email gcf@npac.syr.edu if you are interested in creating or using this facility. This HPFF application suite would be available by netlib or anonymous FTP.