HPFF Meeting Notes for the April 22, 1992 Meeting

High Performance Fortran Forum

CRPC-TR93312
May 1993

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

constructs. Proposals for these constructs are now being circulated within the subgroup.

- Two new subgroups were formed: one for discussion of new intrinsic functions, and one for discussion of parallel I/O as part of HPF. Like the other subgroups, these subgroups will have publicly-accessible mailing lists kept at Rice University. Details of joining these groups are in the detailed meeting notes.

In addition, discussion of documentation and distribution requirements lead to proposals for making more HPFF materials available to the public. Details of this are being worked out, but the result should be the electronic availability of all HPFF working group handouts and electronic discussions. In particular, most of the materials handed out at this meeting are now available by anonymous FTP from titan.cs.rice.edu in the HPFF directory. See the README file there for the file names. In addition, an outline for the HPFF language draft is under construction.

Dates and agendas for future HPFF meetings are as follows:

June 8-10	Data Distribution, Entire Non-Character Array Sublanguage
July 23-24	Storage Association, Dynamic Redistribution
September 9-11	FORALL (and Similar) Constructs, Local Subroutines, Intrinsic Functions
October 21-23	Official Fortran 90 Subset, Miscellaneous Features
December 2-4	Approval of Final Draft

All meetings will be in Dallas, except for the July 23-24 meeting in Washington, DC (held in conjunction with ICS). The three-day meetings start with one half-day for subgroup meetings, followed by a day and a half for the full group meeting.

HPFF Documents

The following documents related to HPFF are available by anonymous FTP from titan.cs.rice.edu in the public/HPFF directory. Additions since the March meeting are marked with a plus sign (+).

announce.* - HPFF Announcement and Call For Participation

database - HPFF address database

+ handouts - Directory containing electronic copies of handouts from HPFF meetings.
  + apr - Handouts from the April HPFF meeting
    + ALL - Concatenation of all other files in this sub directory
    + Distribute - Data distribution proposal from distribution subgroup
    + FORALL - Email traffic and proposals from FORALL and local subroutine subgroup
    + Fortran90 - Storage association proposal from Fortran 90 subgroup
    + Pointer - Discussion of distributing Fortran 90 pointers by Barry Keane
    + Subroutine - Email traffic and proposals from subroutine interface subgroup

minutes - Directory containing minutes of past HPFF meetings
  jan-minutes - Summary of the first HPFF meeting, January 27-28, Houston
  jan-proposals - Point-by-point comparison of language proposals at January meeting
  mar-minutes - Summary of HPFF meeting, March 9-10, Dallas
  + apr-minutes - Summary of HPFF meeting, April 23-24, Dallas

papers - Directory containing papers related to HPFF, primarily from presentations at the first meeting.
  convex.* - Slides for Convex presentation at January HPFF meeting
  fd.* - Fortran D language specification
  fd-over.* - Fortran D project overview
  fd-sc91.* - Fortran D compilation paper (presented at Supercomputing '91)
  hpf.ps - DEC HPF language specification from January HPFF meeting
  mpp.ps - Cray MPP Fortran language specification
  tmc - Position paper by Guy Steele from January HPFF meeting
  vf.* - Vienna Fortran language specification

welcome - "Welcome to HPFF" message, including instructions for using anonymous FTP and joining mail lists.

See the README file in the main directory for information on file extensions and compressed files.

Public comment on any of the proposals is welcome. Please address your remarks to the appropriate email list (see the "welcome" file for a list of these groups).

Detailed Meeting Notes

Attendees

Ralph Brickner (Los Alamos National Lab), Alok Choudhary (Syracuse), Don Heller (Shell), Peter Highnam (Schlumberger), Venkata Konda (nCUBE), Maureen Hoffert (Hewlett Packard), Ken Kennedy (Rice), Chuck Koelbel (Rice), Bob Knighten (Intel), John Levesque (Applied Parallel Research), Dave Loveman (DEC), Piyush Mehrotra (ICASE), Andy Meltzer (Cray Research), Rex Page (Amoco), David Presberg (Cornell), J. Ramanujam (Louisiana State), David Reese (Alliant), P. Sadayappan (Ohio State), Rony Sawdayi (APR), Randy Scarborough (IBM), Tin-Fook Ngai (HP), Rob Schreiber (RIACS), Vince Schuster (Portland Group), Marc Snir (IBM), Guy Steele (Thinking Machines), Richard Swift (MasPar), Clemens-August Thole (GMD), Douglas Walls (Sun), Richard Schooler (HP), Rich Shapiro (United Technologies), Joel Williamson (Convex), Min-You Wu (SUNY Buffalo), Mary Zosel (Lawrence Livermore National Lab), Roland Zink (University of Stuttgart)

Memorable self-introductions

Randy Scarborough: "From the now downsizing IBM"
David Loveman: "From the already downsized DEC"
John Levesque: "From the emerging APR"
(There was a long sequence of introductions in a similar vein, which I didn't get copied down in time.)

Fortran 90 subgroup

Mary Zosel presented the Fortran 90 subgroup report. The major points made on her slides are given below.

Fortran 90/77 "Approach"

- Identify issues in both standards.
- From work of *all* groups identify Fortran 90 features which are relevant to solving HPF technical issues.
- Near the end of the process we will identify HPF to include these features and make a specific list of

Fortran 90 requirements and areas of Fortran 90/77 that are restricted.

- We are operating under the KISS (*K*eep *I*t *S*imple, *S*tupid) principle.

Directives

- We recommend directives to declare COMMON blocks and variables to be distributable or sequential.
- We recommend some mechanism to apply a directive to an entire standard program unit (for example, how to interpret an ambiguous COMMON block).
- We observe some (controversial) usefulness for F90 trailing comments. For example,

```
REAL, DIMENSION (10,10) :: A, B !HPF
DISTRIBUTE(BLOCK,BLOCK)
```

- The evidence so far is that the user community wants the Fortran 90 array language. We recommend the *entire* (non-character) array language be included in HPF, including WHERE, sections, interface, array-valued functions.
- Discuss storage association later.

The discussion of the array sublanguage was supported by an informal survey of user opinions, included as a handout. To summarize that survey, when given a choice between the 3 alternatives

1. Bind HPF to F77 and Fortran 90 standards
2. Bind HPF to F77 and minimum other extensions (in particular, *no* array sublanguage)
3. Bind HPF to a subset of Fortran 90, including the array sublanguage

Mostly favored having the array syntax. Ralph Brickner presented a more detailed survey of about 40 Los Alamos users. Some of the results of that survey were

- Users were generally favorable toward data parallelism.
- Several expressed the need for MIMD features (an especially interesting result, given that these were mainly CM-2 users).
- Portability of the same code from workstations to massively parallel processors was favored by a margin of 22 to 0.
- Full Fortran 90 was favored over Fortran 77 as the HPF base language by a margin of 17 to 2 (with 3 not sure).

- When asked about specific Fortran 90 features, users favored keeping array syntax (12 votes for), structured control flow (4 votes), dynamic memory allocation (3), modules (2), and new intrinsics (2). Users would omit modules (3 votes against) and derived types (2). (The phrasing of the question made it clear that adding or omitting features was based on what users thought they could get, not what was ultimately wanted.)
- The CM Fortran LAYOUT/ALIGN model was considered adequate by 7 users, inadequate by 3.
- The CM Fortran ALIGN functionality was judged adequate by 4 users and inadequate by 4 users.
- The CM Fortran :SERIAL, :NEWS, and :SEND functionality was judged adequate by 5 users, inadequate by 2.

Much discussion followed on the topic of how HPF would treat Fortran 90.

John Levesque started the discussion by stating that he was now in favor of including array syntax. He noted that it was still important that extensions can be converted back to Fortran 77.

David Presberg asked what the issues were in the Fortran 77/90 decision. After some discussion, there was some agreement that certain features (such as dynamic memory allocation) were better done directly in a modern language like Fortran 90 than by making compromises and special cases in Fortran 77. Randy Scarborough said that he saw problems of this type appearing in the F90 subgroup the night before. His conclusion was that resolving those issues in Fortran 90 would keep HPF simpler. Mary Zosel mentioned that storage association issues, which appear to be one of the main roadblocks to HPF, occur in both dialects.

Rich Shapiro repeated the argument that having 2 bindings would mean Fortran 77 was the only portable standard. Thus, requiring array syntax gives a better portable standard. Joel Williamson remarked that HPF will have much more than just array notation.

Marc Snir made the strong argument that HPFF cannot subset Fortran 90; it must work with the entire language. Therefore, he advocated a binding to all of Fortran 90. Ken Kennedy and Mary Zosel clarified the proposal by saying that the subset approach would mean defining "required" features, but a binding to the entire F90 standard would still be required. Clemens Thole and Dave Loveman agreed strongly. In answer to a question, Mary said that the Fortran 90 subgroup recommended that F77 plus data distribution should not be considered HPF.

David Presberg played devils advocate by asking whether there was a semantic problem with defining a language with data distribution (and FORALL constructs) but no array operations. Richard Swift and Mary Zosel quickly supplied examples of situations involving passing arrays to subroutines. Defining distributions consistently seems to require explicit array sections.

Ken Kennedy then proposed a survey of alternative strategies. The choices were given as

1. Identify Fortran 90 binding, but do not require the full language in HPF implementations.
2. Identify Fortran 90 binding and require some F90 features (i.e. the subgroup's recommendation).
3. Bind to Fortran 90, no subsets allowed (immediately termed "the Ada model" or "the flexible government approach").
4. Identify separate Fortran 77 and Fortran 90 bindings
5. Identify a Fortran 90 binding and an "official subset"

Rob Schreiber moved to adopt strategy 5, David Loveman seconding. Guy Steele called for a straw poll first, the results of which were

1. 5 yes
2. 9 yes
3. 0 yes
4. 0 yes
5. 18 yes

Another straw vote was then held between strategies 2 and 5, with strategy 5 winning 25 to 4. An official vote was taken to adopt the "official subset" strategy, the results being

- Yes 24
- No 2

(The number of votes was lower because the one-vote per-company rule was enforced.) The group then took a short break before the data distribution discussion.

Before starting on the distribution discussion, Mary Zosel asked for a straw poll on whether the official HPF subset should include the entire non-character array sublanguage. The vote was 22 to 2 in favor, and the Fortran 90 subgroup was directed to start developing the official subset on this basis.

Data Distribution Subgroup

Guy Steele presented the results of the Data Distribution subgroup meeting. The technical details of the proposal were contained in a handout printed the night before after subgroup discussions. (The group was suitably impressed by that quick turnaround!) Guy prefaced his remarks by saying that the proposal was a combination of working group discussion and previous proposals and therefore "shouldn't be construed as official". In fact, some parts had not been discussed at all.

The distribution model consists of several components:

- Data arrays are mapped to conceptual TEMPLATEs by one of two operations: ALIGN (a static declaration) or REALIGN (an executable statement).
- TEMPLATEs are mapped onto logical PROCESSOR arrangements (which are assumed to be in one-to-one correspondence with physical processors) by two operations: DISTRIBUTE (static) and REDISTRIBUTE (dynamic).
- It is expected that vendors will provide a "reasonable" default mapping from logical PROCESSOR grids to physical processors; furthermore, machine-dependent mapping directives will doubtless be defined to allow specialized topology-specific mappings. The distribution group will not address this mapping at this time, however.

TEMPLATE is the a new term equivalent to the Fortran D DECOMPOSITION. The reasons for this change are given below. To unify notation, these minutes will use TEMPLATE exclusively, although most of the discussions actually called it DECOMPOSITION.

Every array is created with a default alignment to some TEMPLATE, and every TEMPLATE has a default DISTRIBUTION to a PROCESSOR grid. The exact nature (e.g. whether the default distribution is BLOCK or CYCLIC) is implementation-dependent. Explicit ALIGN and DISTRIBUTE operations modify these defaults. Either the TEMPLATE or the PROCESSOR can be anonymous if no manipulation of the name is required. In general, an array name can be used wherever a TEMPLATE name can go, and in that context it stands for the array's underlying TEMPLATE.

DISTRIBUTE and REDISTRIBUTE map from TEMPLATE elements to PROCESSOR arrays. Two constructs are given to preserve the clear distinction between Fortran declarations and executable statements.

To mark decompositions that may be redistributed, the REDISTRIBUTABLE attribute is introduced by analogy to the Fortran 90 ALLOCATABLE attribute. Note that since it is the TEMPLATE that is mapped, a REDISTRIBUTE operation always affects all arrays aligned to a TEMPLATE. In particular, using an array in a REDISTRIBUTE operation may affect many other arrays.

ALIGN and REALIGN map data arrays to TEMPLATEs. As above, this is to preserve the distinction between declarations and statements. Also as above, the REALIGNABLE attribute is introduced with the obvious meaning. Both the dummy-variable (Fortran D) notation and the array section (Guy Steele's) notation are supported. Aligning arrays with other arrays is syntactically possible, but as stated above this is merely shorthand for using the underlying TEMPLATE. To avoid various pathologies, if an array is aligned with another array, the "target" array cannot be REALIGNABLE.

A key design decision was to treat most directives (including ALIGN and DISTRIBUTE) as Fortran 90 attributes. This allows us to combine directives in any order, for example

```
!HPF DISTRIBUTE(BLOCK,BKOCK) ONTO P,
DIMENSION (100,100),
& !HPF REDISTRIBUTABLE :: D, E, F
```

(This example assumes continuation comments (from the DEC HPF proposal) for HPF and vendor-specific extensions.)

Predictably, there was a great deal of discussion of this model.

Several people asked, at various times and in various phrasings, whether TEMPLATEs were really necessary. Perhaps the best explanation for this level of abstraction was that it provided an easy way to group related arrays that should always be aligned and distributed identically. Given the syntactic rules in effect, it appeared that explicit TEMPLATE declarations will be rare. (This had caused a heated argument in the subgroup meeting the day before; the response to this comment was "Read Piyush's book!" referring to a future writing project. A similar rallying cry was heard for Clemens Thole's book.) Others made the point that each array had a default TEMPLATE, but nothing was said about the relation between TEMPLATEs for separate arrays. This argued that relying on the default to ensure alignment might be a very bad strategy. Another point was that many of the arguments in favor and against the TEMPLATE construct were based on principles rather than from actual examples. While there was merit to this observation, there was also no way to produce any real-world examples during the

HPFF meeting. Examples that might illuminate this debate would be welcome.

A recommendation was made in the subgroup to rename DECOMPOSITION (the original name for TEMPLATE). There was general agreement that this was a non-optimal name, but no agreement on what to change it to. A straw poll from the data distribution group produced the following suggested names and vote counts (multiple votes were allowed):

DOMAIN: 10 votes in subgroup
LATTICE: 9
GRID: 5
SUPPORT: 4
MODEL: 3
LOGICAL*0: 0 (not a particularly serious suggestion)

After some discussion, a straw poll of the entire group was taken (again, allowing multiple votes). The results were

DECOMPOSITION: 7
*DOMAIN: 22
*LATTICE: 16
*GRID: 9
SUPPORT: 3
MODEL: 1
*TEMPLATE: 25
FRAME: 1
FRAMEWORK: 3
LOGICAL*0: 0
NONE OF ABOVE: 15

A few other names were suggested (notably PEGBOARD), causing Dave Loveman to remark that "LOGICAL*0 is looking better." A final poll was taken allowing only one vote per person, and only considering the most popular choices above (marked with asterisks). The results were

DOMAIN 6
LATTICE 3
GRID 1
TEMPLATE 21

Based on this, the decision was made to publish the minutes using TEMPLATE, with the understanding that the name was still subject to change.

The distribution subgroup had not discussed the treatment of COMMON extensively. Their basic approach was to treat it as a 1-D array whose elements are STORAGE UNITS (as defined in the Fortran standards). A programmer could distribute it if and only if no object in the COMMON block was explicitly ALIGNED. A much more detailed proposal was presented in the afternoon by the Fortran 90 subgroup.

A significant email discussion had examined the definition of BLOCK distribution when the array size N was not evenly divisible by the number of processors P. The current definition (all processors round up the block size) produces quite bad distributions when P<N<2P. Vienna Fortran uses an alternate definition that never leaves processors idle, but that requires a conditional to evaluate subscripts. Guy Steele proposed a new method that attempted to retain the advantages of both definitions, and Joel Williamson questioned whether the inverse of the mapping was efficiently computable. The issue was left unresolved at this meeting.

Mary Zosel pointed out that Fortran 90 pointers can cause serious data redistribution problems, and that the distribution and Fortran 90 groups needed to consider this carefully. Barry Keane produced a write-up of some of the issues involved in this, and basically suggested that pointers would have to be restricted in some way. On that happy note, the group broke for lunch.

After lunch, Rob Schreiber asked if there were any actual examples of the need for REALIGN and REDISTRIBUTE. Several examples, including adaptive and irregular problems and changing distributions based on problem size, were given by Piyush Mehrotra. David Loveman recalled the bashing he took over omitting the dynamic features from the DEC proposal at the first HPFF meeting. Peter Highnam asked whether it was acceptable to force copying to a correctly aligned and distributed array in order to simulate the remapping. Some discussion ensued over whether the compiler could avoid allocating new memory when performing redistribution. Ken Kennedy suggested that HPF should allow programmer intent to be expressed as clearly as possible.

Peter then raised the question of whether (and how often) compilers could override the programmer distributions. Richard Swift believed this could be common for static distributions, but should be rare for dynamic ones. His argument was that if a user inserted an expensive operation, that user was probably quite aware of the consequences. Ken Kennedy stated that compilers should have the latitude to ignore any distribution declarations or statements if they could deduce a better way.

Ralph Schooler stated that the number of mapping levels needed more rationale. In particular, he did not see the need for two types of processor. Ken Kennedy agreed that there should be an aside in the document covering this design decision.

Marc Snir stated that the document should indicate assumptions about architecture. For example, there seems to be an implicit assumption that nearest neighbor communication (in the logical PROCESSOR array)is fast. Guy Steele promised to do so.

At this point there seemed to be general agreement on the broad outlines of the distribution proposal. Ken Kennedy suggested that the data distribution group be given directions to prepare a formal proposal to be approved at the next meeting. David Loveman asked if approving a proposal meant it was set in stone, to which Ken replied, "Yes." There will be a final pass through the draft near the end of the HPFF process to make corrections. Proposals which are approved should stay fixed until that pass, unless irreconcilable conflicts with other issues are found.

Subroutine Interfaces Subgroup

Joel Williamson presented the report of the Subroutine Interfaces Group. The group's charter was to define the behavior of data distributions on parameter arrays, the scope of data distribution directives, and related problems. The report was by far the shortest at this meeting.

Group 3 had met once and unanimously chosen Marina Chen as its chair. (Unfortunately, Marina was unable to attend this meeting because of an emergency.) They then agreed that they couldn't do much, if anything, before the data distribution group finished its work. Furthermore, it was not clear that the distribution group could finish its work without doing most of the subroutine group's work. Joel then opined that the subroutine interfaces group should be absorbed into the data distribution group. There was general agreement from the floor. Guy Steele said he would talk to Marina Chen about the merge. (Subsequent to the meeting, the merge was made official and the hpff-subroutine mailing list merged into hpff-distribute.)

FORALL Subgroup

Chuck Koelbel presented the report from the FORALL group. This group had not been as active as the data distribution group, but had produced 3 proposals available as a handout.

- Guy Steele's "Local Sections" proposal (updated from the first HPFF meeting).
- A taxonomy of FORALL variants by Chuck Koelbel.
- A concrete proposal for semantics of FORALL by Guy Steele.

The presentation focused on Steele's FORALL proposal, as this had been the main topic of the subgroup meeting the night before.

In summary, the proposal had three parts:

+ Single-statement FORALL
  - Only assignments are allowed in the FORALL body.
  - The left-hand sides computed in each instantiation of the FORALL must be independent, and some necessary (though not sufficient) syntactic conditions are imposed to guarantee this.
  - All right-hand sides are computed before any stores are performed.
  - Expressions may not have side effects.
  - (Note: These are the semantics as defined in Fortran 8X and dropped from Fortran 90; they are also the semantics used in CM Fortran.)
+ Block FORALL
  - Multiple assignments allowed in the FORALL body.
  - Equivalent to a series of single-statement FORALLs.
  - Conceptually, a synchronization and/or communication phase after each statement
  - (Note: This is often referred to as SIMD semantics, although it can be implemented on MIMD and sequential machines.)
- Nested IF (that is, IF statements nested within a FORALL)
  - Defined as a series of masked single-statement FORALLs.
  - Values assigned in the THEN branch are available in the ELSE branch.
  - (Note: This is the direct analog of the WHERE semantics in Fortran 90.)

The subgroup recommended accepting the single-statement and block FORALL semantics as stated. No recommendation was given on the nested IF semantics. Single-statement FORALLs were very noncontroversial, given their wide use in CM Fortran. The other variants were not.

The SIMD semantics for block FORALLs was seen as useful for many problems, although there were suggestions that a different syntax would be preferable. In particular, many members were uncomfortable with the notion of values flowing from one instantiation of the FORALL body to another. For example, in

```
FORALL (I = 1:99)
    A(I) = I
    B(I) = A(I+1)
END FORALL
```

some members felt that element B(1) should receive the original value held in A(2), rather than the new value assigned when I = 2. The argument was also made that this semantics did not extend naturally to other nested constructs, particularly DO loops within FORALLs. Still, the group was willing to live with this semantics, although a majority claimed it was not sufficient alone.

The key pathological example of a nested IF statement was

```
FORALL (I = 1:100)
    IF ( PRIME(I) ) THEN
        A(I) = F(I)
    ELSE
        B(I) = A(I+1)
    END IF
END FORALL
```

According to Guy Steele's semantics, B(1) is assigned the *new* value of A(2) (that is, F(2)). Even some people comfortable with the SIMD semantics for block FORALL were troubled by this. Guy later retracted this semantics for the IF statement (as he put it, "I have changed my mind as the result of repeated pounding to the top of my cranium"). He did, however, advocate retaining these semantics for a nested WHERE statement. This seemed less controversial, although some committee members were surprised to discover that this was the semantics of WHERE-ELSEWHERE.

The proposal did not have firm semantics for other statements nested in a FORALL, such as nested DO or WHILE loops. The FORALL group was encouraged to develop them.

In response to a question regarding why a more general FORALL (or similar construct) was needed, Clemens Thole presented 3 examples from his research. All fell into Geoffrey Fox's "loosely synchronous" category.

1. Finite Element Methods - setup of element matrices
   - An embarrassingly parallel problem consisting of generating a fair-sized local matrix at each domain point.
   - Lots of branches and subroutine calls, due to different functionals for different types of elements.

Needs interface blocks and array section passing for effective "command" implementation. Others in the room commented that it also needs something akin to the INDEPENDENT directives.

2. Computational Fluid Dynamics - 3D-O-NET
   Different code used for different sections of the domain (boundaries, interior, etc.).
   Needs functional parallelism, but each function is data-parallel.
3. Computational chemistry - ab initio codes
   Evaluation of integrals for each pair of basic functions.
   For sets of functions, evaluate the integral if and only if the contribution will be significant.
   Different formulations for integrals depending on local curvature and required accuracy of result.

Clemens claimed that executing these computations in SIMD mode was possible, but at a high overhead. Others in the room disagreed to various extents, saying that different formulations were needed for SIMD. (Not all of the objectors were from Thinking Machines and MasPar, it should be noted.) Clemens was encouraged to circulate a requirements lists for supporting this type of application in a language.

A problem of terminology was noted by Randy Scarborough, who objected to the use of the term "parallel loops" to describe the FORALL and similar constructs. His claim was that this name implied sequentialization of control flow. That is, how can you have a loop when the control threads are running in parallel. He raised a similar objection to "iteration" to describe the instantiations of a FORALL body. The point was well-taken, and I have tried to avoid those terms in these minutes. However, the fact remains that there is no really satisfactory term for these concepts. "Parallel indexing construct" was suggested instead of "parallel loop," but this seemed too long; "parallel construct" did not draw a distinction between FORALL and parallel sections as defined by Guy Steele. Suggestions for a better terminology for these constructs would be welcomed.

In addition to the FORALL constructs, Guy Steele proposed a compiler directive called INDEPENDENT which could be applied to FORALLs and DO loops. In either case, the directive asserted that all iterations were independent. In the DO case, the implication was that the loop could be converted into a FORALL. In the FORALL case, this implied that no synchronization or communication was needed between statements. Statements could use values assigned by (lexically) earlier statements in the same iteration, however. After much confusion and a short coffee break, Guy Steele gave a wonderful graphic presentation of the meaning of INDEPENDENT which, unfortunately, is far too complex to reproduce here. In words, the pictures showed the guaranteed ordering of operations for DO and FORALL both with and without INDEPENDENT directives.

- DO without INDEPENDENT showed a loop header that dominated all other operations, the operations within a loop iteration linked in a linear chain, cross-links from the end of each iteration to the beginning of the next, and a loop exit node dominated by all other operations.
- FORALL without INDEPENDENT had the same loop header and exit nodes, but now the second operation in each iteration depended on the first operation in all iterations., not just its predecessor in the same iteration. (Of course, the third operation depended on all second operations, and so forth.) Note that the linear chains from the DO loop formed a subset of these dependences.
- Both DO and FORALL with the INDEPENDENT assertion showed the loop header dominating the linear chain for each iteration, with no cross-links of any kind.

Once this was clear to everyone, there seemed to be general agreement that this was an elegant solution to the directive problem. The question then became whether HPF should consider this type of annotation. Not surprisingly, no consensus was reached. It was pointed out that vendors would provide these directives in any event (although they would probably use different syntax). Ken Kennedy remarked that there was interesting research to be done on what form of directives would be most helpful to the compiler and most understandable to the user.

After a short break, Ken Kennedy took a series of straw polls to give guidance to the FORALL group. Starting from the low-controversy end of the scale, the results were

- Single-stmt FORALL: 30 in favor, 1 against
  It was noted that this was the only language extension (that is, actual new statement rather than compiler directive) proposed so far.
- Block FORALL with SIMD semantics: 25 in favor, 1 against
- Nested IF: no vote taken after Guy retracted his proposal

- INDEPENDENT for DO loops: 20 in favor, 7 against
- Marc Snir suggested postponing the vote on INDEPENDENT for FORALL argument until we have a deterministic parallel indexing construct. Richard Swift suggested examples of all these constructs were needed. Joel Williamson suggested the need for other parallel loops, noting that there is currently no provision for temporary scalars within FORALL. (It was at this point that Guy presented his graphic representation of INDEPENDENT semantics).
- Despite the above objections, a poll was taken on INDEPENDENT applied to FORALL: 21 in favor, 6 against.
- Should HPF consider having more than one parallel FORALL-like construct? 21 yes, 1 no (However, without a firm semantics for the construct the group could not vote on how many constructs or what the semantics should be.)

Suggested names for the FORALL alternative ranged from FREEFORALL (Mary Zosel's suggestion for a completely asynchronous construct) to the PCF DOALL. Some further discussion of compiler directives was also started by Joel Williamson's description of a directive as a pact between the user and the compiler:

- User: I certify that nothing's wrong with this code.
- Compiler: I won't check if something is wrong (i.e. whether you are lying).

As Joel put it, "We aren't calling ourselves Right Answer Fortran, we're High Performance Fortran!" The discussion quickly moved on to the Fortran 90 group's proposal regarding storage association.

Storage Association

Richard Swift and Rob Schreiber led the discussion, which was based on a technical handout. Rob made the bulk of the presentation.

The talk started with definitions of sequence and storage association. To skip the technical details, these are the features of standard Fortran that rely on mapping multi-dimensional arrays onto a linear memory model using column-major ordering. Sequence association occurs as a result of subroutine parameter passing (the old trick of passing a 2-D array to a routine defined to take a 1-D array, or vice versa). Storage association occurs as the result of

EQUIVALENCE statements or inconsistent declaration of COMMON blocks in separate program units. (Readers not familiar with these concepts are referred to the Fortran standards, or to any introductory Fortran text.) Regardless of its source, association with a linear memory model causes implementation difficulties for modern architectures without a linear address space, such as distributed memory machines. It also makes the meaning of data distribution very difficult to define, for example in the case when part of a distributed array is passed to a separately compiled subroutine. For this reason, David Loveman's original proposal disallowed sequence and storage association on distributed arrays.

The goal of the Fortran 90 group was to allow programs to rely on sequence and storage association, but limit distributability on variables and COMMON treated in that way. To this end, memory objects could be sequential or distributable; the basic rule is that programs cannot explicitly DISTRIBUTE or ALIGN sequential objects.

A variable is sequential if

- It is of base type CHARACTER, or
- It is part of a Fortran 90 sequenced type, i.e. a record field or assumed size, or
- It occurs in an EQUIVALENCE statement, or
- It is explicitly declared sequential, or
- It occurs in a sequential COMMON block ("guilt by association").

Otherwise, a variable is distributable and may be given ALIGN and DISTRIBUTE attributes.

A COMMON block is distributable if

- It has no sequential component ,and
- All occurrences of the COMMON have the same number, type, and shape of components in the same order, and
- It is not explicitly declared sequential.

Otherwise, the COMMON is distributable. Note that this does not necessarily mean that the entire COMMON can be distributed as one object; that is a matter for the distribution group to consider. The conditions are sufficient, however, for the components of the COMMON to be distributed individually.

Guy Steele noted that variables and COMMON blocks had different terminology for essentially the same concept. In reference to Fortran 90 specifically, if %loc is used on any variable, then the variable is sequential. Piyush Mehrotra asked if blank COMMON was always sequential (specifically, in regards to some very frequent coding tricks used on it); Richard Swift explained that “We're not trying to preserve the semantics of non-standard-conforming programs.”

Guy Steele remarked “This is great stuff.” It appears to be completely compatible with the results from the data distribution group (except for a minor terminology change).

Rob Schreiber's presentation then moved on to describing rules for using these new classes of variables.

1. Only distributable variables can be used in ALIGN and DISTRIBUTE declarations. Note that this does not conflict with distributing an entire COMMON block.
2. New restrictions must be placed on sequence association. The illegal situations are
   A. Distributed array element associated with sequential array dummy
   B. Assumed-size array associated with a distributable dummy
   C. Sequential array element associated with a distributable dummy
   D. Distributed array element associated with a distributable dummy
3. Some situations need clarification from the subroutine interface group (now the data distribution group).
   A. Sequential array associated with a distributable dummy
   B. Distributable array associated with a distributable dummy
   C. Distributable array associated with a sequential array

Ken Kennedy was concerned that the model might be overly restrictive. In particular, he argued that the compiler could handle the grouping of COMMON variables. Vince Schuster replied that compilers can extend HPF to allow safe cases, in particular COMMON blocks that are part sequential, part distributed. Richard Swift invoked the KISS principle; the language is simpler to explain this way. Piyush Mehrotra noted that Vienna Fortran allows the kind of thing Ken wanted. Richard Swift agreed that there were other possible definitions; the issue is where to draw the line between allowed features and illegal ones. Somebody avoided a long discussion (Rob still hadn't presented the second two points above) by noting that “It looks like we could discuss this for a few minutes.”

Sadayappan noted a problem with the default condition: it appeared that some Fortran 77 programs were illegal. Further discussion did not entirely resolve the issue, but agreed that it deserved further study. Guy Steele noted that this was like DEC proposal, in that arrays were sequential unless otherwise noted.

At this point, the group broke for dinner at a local Japanese restaurant. Discussion after dinner started with group responsibilities, but then moved back to the COMMON proposal.

Andy Meltzer presented his (hastily prepared) proposal for sequential and distributable COMMON blocks.

- Group 1's proposal has a problem: COMMON can only have one distribution (and which distribution is implementation defined).
- But "sequential" (as used there) really means "*M*aintain *S*equence *A*nd *S*hape *A*ssociation" (MSASA). Note: this is not a syntax proposal!
- MSASA says that for correctness the compiler *must* maintain association.
- New proposal: "distributable" is an assertion about what a user wants his data layout to be (i.e. a hint to the compiler, not a binding construct).

His main point was that MSASA and distribution shouldn't be mutually exclusive. Furthermore, the language shouldn't mandate how the machine accomplishes these tasks (e.g. what the distribution is).

As with most proposals, this sparked a spirited discussion. The traditional "but the compiler could mess up this way" argument was dealt with by Joel Williamson's comment, "Users can do stupid things like this now."

Peter Highnam asked what the advantage of this proposal over the previous one was, to which Andy replied that it gave users access to more memory. This was hotly contested by Rob Schreiber and Mary Zosel, who claimed he had misunderstood the proposal. (Andy apparently assumed that shared COMMON would be replicated, which was not the case.)

David Loveman asked if an array could be aligned with COMMON. Andy answered yes, but admitted that he hadn't worked out the details carefully yet. Guy Steele noted that this made COMMON analogous to TEMPLATE, where the units of alignment were storage units (as defined in Fortran standards). Randy Scarborough and David Presberg contended that distributing storage units not a good idea, since it could mess up the compiler on some architectures. Andy replied that he hadn't considered those implications, but noted that there was no problem on the Cray MPP.

Piyush Mehrotra asked whether going to Fortran 90 bypassed these problems, and was shouted down by the group. Storage and sequence association are present in F90 (in fact, according to David Presberg even more attention is paid to it). Piyush responded with a question: since we can see nonlinear memories coming, is it a good idea to keep the linear memory model embedded in Fortran?

Richard Swift asked what the user value of distributing COMMON in this way was. Clemens Thole replied that users need only recode their programs once. The question then became what the advantage over distributing the arrays within COMMON was, to which the reply was avoiding the recoding of memory allocation (i.e. the kind of dividing COMMON done in F77). David Loveman noted that there was no evidence that keeping this allocation strategy would improve performance. Peter Highnam was concerned over additional vendor effort for no gain, to which Andy Meltzer replied that there was no extra effort, since the directives were not binding. Rich Shapiro claimed that the real issue was whether the programmer can always control distribution. Andy Meltzer noted that the user can't say anything under the current proposal.

Ken Kennedy then unveiled his complaint about all the proposals. A simple example showed a situation that he would like to allow, but that was illegal under all current proposals.

```
COMMON /blk/ a(100), b(100), c(100), d(50),
e(50)
REAL f(100)
EQUIVALENCE (d(1),f(1)), (f(51,e(1))
!HPF$ TEMPLATE, DISTRIBUTE dd(100)
!HPF$ ALIGN WITH dd:: a, b, c
```

Randy Scarborough noted that as soon as you allow EQUIVALENCE, users can play games. (In particular, users can play games that defeat compilers.) Ken replied that he was suggesting that programs have grown over years. Variables were added to COMMON for various reasons, and other modifications had happened. His goal was to not force more rewriting than needed. Richard Swift again stated that the hard question is drawing the line between user and compiler responsibilities; his group had drawn it close to the user. Ken stated that what was really needed was a way to align the "super-variables" formed by EQUIVALENCE. Marc Snir stated that correctness under these circumstances kills separate compilation, to which Ken replied that we have to trust the users.

Rob Schreiber objected that the rules for these features were getting complex. David Loveman became nervous because the linker has to figure out all relations to do its job. Marc Snir believed the proposals did allow separate compilation, although Guy Steele thought more HPF stuff in MAIN was needed to ensure consistency.

At this point Richard Swift took a series of straw votes.

1. On the F90 group proposal (key points: good and bad COMMONs, EQUIVALENCE kills distributability)- Are you generally favorable? 14 yes, 1 no, 16 undecided
2. On the Andy Meltzer proposal (key points: distribute sequential COMMON blocks as a whole)- Are you generally favorable? 10 yes, 3 no, 15 undecided
3. On the Ken Kennedy proposal (key points: COMMON can be partially sequential, partially distributable)- Are you generally favorable? 11 yes, 3 no, 14 undecided
4. Should CHARACTER arrays be distributable? 2 yes, 14 no, 11 undecided

Further investigation showed lots of swing voting between the three COMMON proposals. Ken Kennedy said that the group needed to think through this issue and discuss it again. Richard Swift though that email could effectively be used to discuss the issue before the next meeting, but asked what could change the undecided votes. Ken and Mary Zosel replied that finding out tradeoffs for users would be a good start. Ken further suggested going with the simplest proposal (i.e. the original F90 group's) if there was still controversy.

The group then called it a night after a short discussion of the next day's agenda.

Subgroups

(This section is out of chronological order, so that the COMMON discussion above could be presented together.) A few responsibilities for the subgroups were clarified.

1. Fortran 90 subgroup: Define the official subset and interactions with F90.

2. Data Distribution subgroup: Prepare a final proposal of distributions for the next meeting
3. Subroutine Interface subgroup: Absorbed into group 2
4. FORALL and Local Subroutines subgroup: Prepare drafts of the FORALL and other parallel indexing constructs.

   It was noted that new names for these constructs would be welcomed; current candidates include DOALL (from PCF), FREE FORALL, ENDALL, BEALL, and Y'ALL. Some of these are not entirely serious proposals.
5. I/O and Miscellaneous Features subgroup: The group is charged with examining what features for parallel I/O are needed in HPFF.

   This group was activated after being ignored since the last meeting. Bob Knighten is acting as convener. Initial members include Barry Keane, Alok Choudhary, David Loveman, Marc Snir, Peter Highnam, and Rex Page. The mail alias is hpff-io@rice.edu.
6. Intrinsics subgroup: This group was charged with defining new intrinsics useful for parallel computation.

   This is a new subgroup. Rob Schreiber will serve as convener. Initial members are Rich Shapiro, Ralph Brickner, P. Sadayappan, David Loveman, Guy Steele, Peter Highnam, and Rex Page. The email alias is hpff-intrinsics@rice.edu.

Meeting Agendas

The first business item on Friday was planning future meetings. The final schedule was

June 8-10 (note change from June 8-9) - Dallas, TX

Subgroups meet June 8 starting at 1:00. The main group meeting will be June 9-10, all day and evening on the ninth and ending at noon on the tenth.

Subjects to be voted on: data distribution proposal, allowing the entire non-character array sublanguage in the HPF subset. Also to be discussed (but voted on in July): storage association and data redistribution.

July 23-24 - Washington, DC (in conjunction with ICS)

Subgroups meet on the morning of the twenty-third. The main group will meet starting at 1:00 on the twenty-third (probably through the evening) and on the twenty-fourth ending at 5:00.

Subjects to be voted on: storage association and COMMON block proposal, data redistribution. Also to be discussed (voted on in September): FORALL, local subroutines, and intrinsics.

September 9-11 - Dallas

Subgroups meet starting at 1:00 on the ninth, working through the evening. Main group meets on the tenth and eleventh, ending at noon on September 11.

Subjects to be voted on: FORALL and local subroutine proposal, parallel intrinsics. Also to be discussed (but voted on in October): Fortran 90 subset, miscellaneous features.

October 21-23 - Dallas

Subgroups meet starting at 1:00 on the 21st, working through the evening. Main group meets on the 22nd and 23rd, ending at noon on October 23.

Subjects to be voted on: Fortran 90 subset, miscellaneous features. Also, prepare a draft for presentation at Supercomputing '92.

December 2-4 - Dallas

A meeting to finalize the draft, incorporating any public feedback from Supercomputing '92 and other sources. Assuming the schedule holds, this will produce the final, approved draft of High Performance Fortran.

Draft Documents

Much discussion went into deciding how HPF drafts should be presented. David Presberg saw the need for a documentation subgroup or support staff. Ken Kennedy agreed that a strategy was definitely needed, but no grant support was available at this time. Marc Snir remarked on the need for a working draft at each stage of the HPF process.

Guy Steele suggested settling on a format, and nominated LaTeX. After some discussion of whether this was really portable, Chuck Koelbel volunteered to write up a format that could be used portably. This was to include sections, cross-references, figures, BNF for syntax, italics, and an outline for the draft document. Guy promised to provide samples for some of these features. In a leap of optimism, it was hoped that this format could produce high-quality typeset output as well as readable ASCII files.

In a related note, Chuck Koelbel agreed to make as many handouts from the HPFF meetings available electronically as possible. These will reside in the same anonymous FTP directory as the previous HPFF material. A list of the documents available through this mechanism is in a section below. In the future, we will request that anyone making a handout for the HPFF meeting send him an electronic version, preferably in plain ASCII or "vanilla" LaTeX. These documents will be in addition to the normal posting of the HPFF minutes to the net.

Parallel I/O

A short discussion of parallel I/O was started by Vince Schuster's warning to be careful of it sneaking into the language definition process. Rob Schreiber opined that Fortran already has sufficient I/O (in fact, more I/O support than any language except perhaps COBOL). Peter Highnam clarified a remark from the previous meeting, saying that even if HPF only contained standard Fortran I/O, users would take advantage of whatever high performance I/O extensions the vendors provided. Clemens Thole asked what Cray and Thinking Machines were doing on I/O. Andy Meltzer answered that the Cray MPP proposal has much more parallelism, including parallel I/O, but HPF doesn't need it due to the different model.

Clemens Thole believed that the standardization group for message-passing (scheduled to meet after the SHPCC conference in Williamsburg) would attack this problem. He believed that HPFF should accept their results. Ken Kennedy, one of the organizers of that meeting, was less sanguine. He viewed that group as "standardizing assembly language" and thought they would limit themselves to message interfaces.

Mary Zosel returned to the subject of I/O after the group explored a few unrelated tangents. Her basic observation was that Fortran 90 has no asynchronous I/O, which will presumably be quite important in parallel machines. The question was, does HPF need to address this? David Presberg suggested that a subroutine interface specification might solve this problem, while Rich Shapiro claimed that vendor extensions will creep into standards in any case. Rich suggested that HPFF limit its scope due to time constraints. Marc Snir spoke in favor of standardizing foreign interfaces. Robert Babb agreed that only interfaces should be given for such "dark corners" of the language. The I/O subgroup took all these points under advisement.

Public Comment

Ken Kennedy encouraged the HPFF committee to give briefings to any interested groups. A key to the success of HPF is feedback from the users. David Presberg emphasized that people are interested in this, and HPFF should encourage this interest. Widely available documents were one way to do this, public presentations another. There seemed to be general agreement on this point, and several members expressed interest in doing such presentations.