HPFF Meeting Notes for the July 23-24, 1992 Meeting

High Performance Fortran Forum

CRPC-TR93314
May 1993

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

HPFF Meeting Notes

July 23-24, 1992

Washington, DC - Hyatt Crystal City

Notes taken by Chuck Koelbel, with help from Bob Knighten

Executive Summary

This was the fourth meeting of the High Performance Fortran Forum working group. It was held immediately after the International Conference on Supercomputing in Washington, DC, where a Birds-of-a-Feather session introducing the language was also held. The effort is mostly on schedule for presenting a document at the Supercomputing '92 conference.

Some of the more important decisions made at this meeting included:

Adoption of the proposal for REALIGN and REDISTRIBUTE made at the last meeting, except for the sections regarding subroutine linkage, COMMON blocks, and ALLOCATABLE arrays. This proposal contained the basic explicit remapping capabilities within a single subprogram.

Adoption of the proposal for sequence association in the presence of distributed arrays made at the last meeting. The proposal essentially disallows sequence association for distributed actual arguments.

Guy Steele and Rich Shapiro's proposal for managing distributed dummy arguments in subprograms was accepted as a first reading. The major point of the proposal was to disallow the redistribution of an actual argument through a subroutine call, although the argument could be temporarily remapped within the subroutine.

Chuck Koelbel and David Loveman's proposal for a FORALL statement and construct was accepted with minor revisions on first reading. These statements are not intended as a general "parallel loop" or as a general parallel construct; instead, they allow assignments to multiple array elements, in somewhat the same way as array assignments and WHERE constructs in Fortran 90.

Guy Steele's proposal for an INDEPENDENT assertion applied to DO loops was accepted as a first reading. The directive is an assertion providing more information to the compiler; if properly used, it does not change the semantics of the program where it appears. Work is proceeding to apply the assertion to other language constructs.

A large number of new standard library routines and intrinsics was approved as a first reading. These functions perform tasks useful for data-parallel computing, including prefix accumulations, new reduction functions, and system inquiry functions. Discussion continues on whether these functions are sufficiently common to merit their inclusion or simply as standard library functions.

Dates and agendas for future HPFF meetings are as follows:

September 9-11	Storage Association, Subroutine Linkage for Distributions, Pointers and ALLOCATABLE Array Distributions, FORALL, DO INDEPENDENT (all second readings); Local Subroutines, Intrinsic Functions, Fortran 90 Subset, FORALL INDEPENDENT, Parallel I/O (all first readings)
October 21-23	Local Subroutines, Intrinsic Functions, Fortran 90 Subset, FORALL INDEPENDENT, Parallel I/O (all second readings); Tentative approval of draft for Supercomputing '92 Tutorial
December 9-11	Approval of Final Draft

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 June meeting are marked with a plus sign (+).

- announce.* HPFF Announcement and Call For Participation
- archives Directory containing the HPFF mailing group archives. (All files are updated nightly.)
  - + hpff All messages sent to hpff@rice.edu
  - + hpff-core All messages sent to hpff-core@rice.edu
  - + hpff-distribute All messages sent to distribution subgroup
  - + hpff-f90 All messages sent to Fortran 90 subgroup
  - + hpff-forall All messages sent to FORALL subgroup
  - + hpff-format All messages sent to editorial subgroup
  - + hpff-intrinsics All messages sent to intrinsics subgroup
  - + hpff-io All messages sent to I/O subgroup
- database The HPFF mailing list
- handouts Directory containing handouts from the HPFF meetings
  - apr Subdirectory containing April meeting handouts
    - ALL Concatenation of the other files in this subdirectory
    - Distribute Proposal for data distribution model by the distribution subgroup
    - FORALL Proposals and discussion from the FORALL and local subroutines subgroup
    - Fortran90 Storage association proposal by the Fortran 90 subgroup
    - Pointer Discussion of Fortran 90 pointers by the F90 subgroup
    - Subroutine Discussion by the Subroutine interfaces subgroup
  - june Subdirectory containing all June meeting handouts

    - ALL — Concatenation of the other files in this subdirectory
    - Association — Storage and Sequence-Association Proposal
    - COMMON-Kennedy — Distributing COMMON Blocks Proposal (Ken Kennedy)
    - COMMON-Meltzer — Distributing COMMON Blocks Proposal (Andy Meltzer)
    - Distribute.tex — Data Distribution Proposal (Guy Steele)
    - FORALL-Loveman.tex — FORALL construct proposal (David Loveman)
    - FORALL-Wu — FORALL construct proposal (Min-You Wu)
    - Intrinsics — Intrinsics Proposal (Rob Schreiber)
    - Subset — Fortran 90 Subset Proposal
  - + july — Subdirectory containing all July meeting handouts
    - + ALL.shar — Shar file containing all other files
    - + Anti-FORALL.tex — Arguments against the FORALL construct (Alan Karp)
    - + Pro-FORALL — Arguments in favor of the FORALL construct (Rich Shapiro)
    - + Critique.tex — Critique of HPF distributions (Barbara Chapman and Hans Zima)
    - + Distribute.tex — Data distribution proposal (Guy Steele)
    - + FORALL.tex — FORALL proposal (David Loveman and Chuck Koelbel)
    - + Original-Storage — Sequence and Storage Association proposal, beginning of meeting
    - + Final-Storage — Sequence and Storage Association proposal, as finally presented (F90 group)
    - + IO.tex — Parallel I/O proposal (Marc Snir)
    - + Independent.tex — INDEPENDENT proposal (Min-You Wu)
    - + Intrinsics — Intrinsics proposal (Rob Schreiber)
    - + Local-Subr.tex — Local Subroutine proposal (Marc Snir)
    - + MIMD — MIMD support proposal (Clemens-August Thole)
- minutes — Directory containing minutes of past HPFF meetings
  - jan-minutes — Summary of the first HPFF meeting
  - jan-proposals — Point-by-point comparison of language proposals at first HPFF meeting
  - mar-minutes — Summary of HPFF meeting, March 9-10, Dallas
  - apr-minutes — Summary of HPFF meeting, April 23-24, Dallas
  - europe-minutes — Summary of European Workshop on High Level Programming Models for Parallel Architectures
  - june-minutes — Summary of HPFF meeting, June 9-10, Dallas
  - + july-minutes — Summary of HPFF meeting, July 23-24, Washington

papers	Directory containing papers related to HPFF. (Note that these are not intended as formal drafts, but rather as supporting documents.)
convex.*	Slides for Convex presentation
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
+ hpff-europe.ps	"High Performance Fortran: A Perspective" by Brian Wylie, Michael Norman, and Lyndon Clarke (University of Edinburgh)
+ merlin.ps	"Techniques for the Automatic Parallelisation of 'Distributed Fortran 90'" by John Merlin (University of Southhampton)
mpp.ps	Cray MPP Fortran language specification
tmc	Position paper by Thinking Machines on HPFF
vf.*	Vienna Fortran language specification
vf-over.*	Vienna Fortran language overview
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

Robbie Babb (Oregon Graduate Institute), Barbara Chapman (University of Vienna), Marina Chen (Yale University), Alok Choudhary (Syracuse University), James Cownie (Meiko), Rich Fuhler (Lahey), Peter Highnam (Schlumberger), Maureen Hoffert (Hewlett Packard), Hidetoshi Iwashita (Fujitsu Labs), Ken Kennedy (Rice University), Bob Knighten (Intel), Chuck Koelbel (Rice), Dave Loveman (DEC), Piyush Mehrotra (ICASE), Andy Meltzer (Cray Research), Ken Miura (Fujitsu America), Prakash Narayan (SunPro, Sun Microsystems), Tin-Fook Ngai (HP), Edwin Paalvast (Technical University, Delft), Rex Page (Amoco), Jean-Laurent Philippe (Archipel), David Presberg (Cornell), J. Ramanujam (Louisiana State University), P. Sadayappan (Ohio State University), Rony Sawdayi (Applied Parallel Research), Randy Scarborough (IBM), Rich Shapiro (Thinking Machines Corporation), Margaret Simmons (Los Alamos National Laboratory), Marc Snir (IBM), Guy Steele (TMC), Kate Stewart (IBM Canada), Richard Swift (MasPar), Clemens-August Thole (GMD, St. Augustin), Joel Williamson (Convex), Min-You Wu (SUNY Buffalo), Mary Zosel (Lawrence Livermore National Laboratory)

Memorable self-introductions

Margaret Simmons, substituting for the honeymooning Ralph Brickner, signed the attendance sheet "virtual Ralph Brickner".

Dave Loveman identified himself as being "from the newly-CEO-ed DEC".

HPF Europe Report

Clemens-August Thole gave a brief report on developments in Europe related to HPFF. The HPF Europe group held a meeting June 23 in Brussels which included attendees from many major companies and universities on the continent. The topics of discussion included

- Discussion of data distribution at subroutine boundaries: Most of the points made there are covered in the current proposal, thanks to strong European contributions to the mailing list discussion.
- Parallel I/O: Peter Brezany from the University of Vienna presented the results of experiments in Vienna Fortran showing that special treatment of distributed arrays in file I/O could have significant advantages. James Cownie presented a programmer interface approach to the same problem.
- MIMD support and message passing: Clemens-August Thole and Delces Watson presented their ideas on adding this support to HPF. The goal is to allow concurrent execution of everything in a program without resort to local subroutines. Such support would require messages as a part of the language, not restricted to local subroutines. The group plans to propose these features for the second round of HPFF after the core language discussed in the US has been adopted.
- Indirect mapping arrays: Presented by Hans Zima, these were seen as an important feature for supporting irregular computations. They will also be proposed for later versions of HPF, but not for the current draft.

The next European meeting will be held September 21. Interested parties should contact Clemens-August Thole for details.

Another European meeting may be possible after Supercomputing '92 to disseminate information about HPF. This remark started a short discussion of ways that High Performance Fortran could be publicized. Ken Kennedy wanted to disseminate as much information as possible at SC '92, thus getting as much input as possible for the final draft approval. Some felt that SC 92 would be too late for this (a topic that would recur later). It was generally agreed that a full draft would be needed in October, well in advance of Supercomputing ‘92. Ken asked if HPFF could arrange for European feedback. Clemens Thole suggested either a special event or a presentation at a planned Esprit meeting at Brussels; although he did not know of any conferences in the near future, he did promise to check on that as a third possibility. He noted that US participation at such an event would be important for image.

Data Distribution: REALIGN and REDISTRIBUTE

Guy Steele next led a short discussion of the REALIGN and REDISTRIBUTE statements. The proposal had not changed since the last meeting, and there had been little objection to it in the electronic discussions. The major objection raised was that the declaration (without the !HPF$ directive)

```
REALIGNABLE A
```

could also be parsed as

```
REAL IGNABLEA
```

This is not a problem in the current language, because REALIGNABLE is a structured comment; it does, however, conflict with the goal of eventually adding the HPF extensions to Fortran simply by removing the comment headings. Three fixes were proposed:

1. Always take the longest possible keyword when there is an ambiguity.
2. Change the spelling of the HPF attributes to RE_ALIGN and RE_ALIGNABLE.
3. Add significant blanks to Fortran, at least in this case.

The group had a slight preference to the first alternative, and the proposal was amended appropriately. In a vote taken just after dinner, the forum formally accepted the REALIGN and REDISTRIBUTE features, with possible changes in syntax to remove the parsing ambiguity allowed if a new proposal emerged. The final vote was 15 in favor, 3 against, and 3 abstentions.

Discussion of data distribution in subroutines and ALLOCATABLE arrays was deferred until later, to give Guy a chance to make up overhead slides.

Official HPF Subset

(This section is mostly taken from Bob Knighten's notes, as mine were wiped out by an ill-timed battery failure.)

Mary Zosel next took a few straw polls regarding proposed features for the High Performance Fortran subset.

The first question was 'Should procedure-less modules be a part of the HPF official subset?" The full group vote was 15 yes, 9 no, and 13 abstain. Vendors, in a separate count, voted 4 yes, 4 no, and 7 abstain. The intent was to limit the form and content of INTERFACE blocks to provide explicit interfaces and satisfy requirements for explicitly interfaces (useful for handling some distribution scoping issues) but not user-defined operators, overloaded operators or generic interfaces. Some vendors claimed that the most difficult part of modules was the naming features, so restricting them to remove procedures was not an advantage.

The next question was "Should `entity-oriented' (in previous meeting notes, `object-oriented') declarators using attributes in type declaration statements be part of the HPF official subset?" The vote was 19 yes, 8 no, and 10 abstain. Mary noted that this form was already defined in the distribution directives.

Taking the negative, the next question was "Should HPF *not* include free-form source in the official subset?" The vote was 23 yes, 0 no, and 10 Abstain. The group agreed that long variable names and a few other features of the free-form source were needed for HPF, but these had been approved separately already.

Mary noted that the first reading of the subset definition will be at the next meeting, and the only features under consideration for inclusion were those she had just asked for votes on. She asked anybody with any other items from Fortran 90 that should be added to the subset to contact the F90 subgroup soon.

After the first set of polls was taken, a short discussion of the form of the subset started. Rex Page raised the issue of whether the HPF subset should include all new HPF features. The arguments in favor claimed that any subsetting diminished the impact of

HPF. Arguments opposed to this stressed the importance of quick implementations which might be delayed by overly-ambitious HPF features. Mixed into this discussion was a suggestion of an HPF “superset” that might include proposed HPF features that were not adopted; another approach suggested was to have a “Journal of Fortran Development” that could document these suggestions. A very informal poll found the majority in favor of including all of the new features in the subset at this point in the meeting. (See the Intrinsics section for later views on this matter.) Consideration of an HPF superset was deferred until December, when it would be clearer what features were accepted and what (if anything) was not.

Data Distribution: Subroutine Interfaces

After a one-hour break to produce slides and otherwise handle logistics, Guy Steele presented the subroutine interface proposal. This was the first reading for the proposal, and thus it will be formally voted on at the next meeting. The main goal of the document was to define the behavior of distributed arrays when used as actual arguments to subprograms, and to define the meaning of dummy arguments which have explicit data distributions.

The presentation started with a definition and two design decisions:

- Two identically declared processor arrangements are “similar”. (Here, “identically declared” refers only to the arrangements’ shape, not the names used. Some question remains about whether arrangements with differing lower bounds are similar, which will be cleared up in the final draft.)
- Two identically declared and implicitly distributed TEMPLATEs will be similar, and distributed in the same way onto similar processor arrangements.
- Two TEMPLATEs will also be similar if explicitly distributed onto similar processor arrangements in the same way.

The purpose of these definitions was to support later proposal features. The objection was raised that an application may want two same-sized arrays to be dissimilar, to which the response was that the proposal only applied to TEMPLATE constructs. Chuck Koelbel noted that this did not solve the problem, since implicitly declared TEMPLATEs from equal-sized (and not explicitly distributed) arrays would be similar. Rich Shapiro got the proposal out of that corner by suggesting that the distribution restrictions only applies to explicitly declared TEMPLATES. In any case, Guy pointed out that the intent was to make sure identically declared TEMPLATEs are aligned.

Guy advanced to his next slide.

- TEMPLATES declared in different scoping units are distinct (but may be similar).
- Note: TEMPLATES cannot appear in COMMON blocks, so the only way to share a template among several program units is to use Fortran 90 modules.
- Returning from a subprogram causes locally declared TEMPLATES to become undefined. (That is, there is no SAVE for TEMPLATES.)

To clarify the difference between TEMPLATES that are similar and those that are equivalent, Guy used the following analogy: “Any two pennies are similar in the sense that you can spend the in the same way. They are equivalent you put one on railroad track and the other one gets squished.” The intent of these rules is to define a lifetime of

TEMPLATES. Guy pointed out that this was not the only possibility, but was a convenient one. It may, however, have some problems for pointers.

Guy then moved on to his next slide, demonstrating several options for subroutine linkage.

Declarations:

```
SUBROUTINE FOO(A)
TEMPLATE T
REAL A(:)
```

Option 1:

```
!HPF$ ALIGN WITH T :: A
```

The actual argument (or a copy) is forcibly realigned with T on entry to the routine.

Option 2:

```
!HPF$ ALIGN WITH * :: A
```

The dummy argument is aligned with a TEMPLATE which is similar to (a copy of) the actual's.

Option 3:

```
!HPF$ ALIGN WITH *T :: A
```

The dummy is aligned with T; program is nonconforming if this is not true of the actual.

Option 4:

If a dummy has no explicit align attribute the compiler chooses a default of the form

```
ALIGN with T
```

for some T, or

```
ALIGN with *
```

but not

```
ALIGN with *T
```

The full set of rules is laid out in the draft proposal, available by FTP. The limitation on option 4 derives from the fact that the *T form disallows certain actual arguments to be passed to the corresponding parameter. Much discussion followed. Rich Fuller objected to certain forms of T, particularly optional arguments. Guy amended the proposal to disallow both optional and keyword arguments. The key requirement intended was that the resulting mapping (i.e. both alignment and distribution) be expressible in HPF. Clemens-August Thole and Marc Snir argued this was not true for array arguments with inherited mappings (option 2); Chuck Koelbel and Rich Shapiro answered that specific case, and Guy Steele requested that any counterexamples be presented off-line.

The theme of the next slide was that “Nasty aliasing is forbidden”.

```
MODULE FOO
REAL A(10,10)
REALIGNABLE, REDISTRIBUTABLE A
END
```

```
PROGRAM MAIN
<INTERFACE BLOCK FOR SUB>
CALL SUB(A(1:5,3:9))
END

SUBROUTINE SUB(B)
USE FOO
REAL B(:,:)
REDISTRIBUTE A   ! ILLEGAL
REALIGN B        ! ILLEGAL
END
```

In general, remapping is not allowed in HPF whenever assignment is not allowed (i.e. when it reveals aliases).James Cownie claimed that the example breaks this rule because B is redistributed on entry to SUB. Guy Steele responded that explicit and implicit remappings are different, and the above statement only applies to explicit operations. The intent is to steer as far as possible from aliasing problems. A final restriction along these lines was that COMMON arrays may not be REALIGNABLE, since that would allow remapping by aliasing again.

After a short break, Rex Page raised the question “What are we trying to do in HPF? Support linear memory programming styles or evolve Fortran forward?” His point was that many of the points raised in the distribution work, and which would be raised in the storage association proposal, were attempts to retain compatibility with a memory model that does not fit modern machines. Ken Kennedy responded that this point -- compatibility versus flexibility -- keeps recurring, and the forum has generally sided with as much compatibility as possible. Randy Scarborough noted that Fortran 90 MODULES cause the same or similar problems, so we have to address them anyway. Regardless, a straw poll was taken for this case over whether the subcommittee should keep trying to solve the non-modules problem. By a 16-3 margin, the group voted to continue with the work.

Data Distribution: ALLOCATABLE Arrays

After a short break, Guy continued with his presentation. The next part addressed distributing dynamically allocated arrays; as before, this was a first reading for this proposal.

- A POINTER or ALLOCATABLE variable may not have the ALIGN attribute.
- Instead, alignment is specified at ALLOCATE time (This also goes for the DISTRIBUTION).
- Three syntax variants are proposed:

  Syntax 1:

  ```
  allocate( p(100) )
  !HPF$ distribute a(block)                 ! attached declarations
  ```

  Syntax 2:

  ```
  allocate( p(100) )
  !HPF$ redistribute a(block)               ! even if a is not distributable
  ```

  Syntax 3:

```
allocate( p(100) )
!HPF$ allocate( a(100), distribute(BLOCK) )
```

The following discussion started by noting that all three proposals have problems, in that they each break at least one rule followed elsewhere in the language. David Presberg noted that the proposal broadened the distributions possible in any case (because the programmer can use general expressions, not just specification expressions, as arguments to BLOCK and CYCLIC).

Piyush Mehrotra proposed another option: Make the distribution an attribute of the variable, declared when the variable is declared. This disallows allocating arrays with different distributions to the same variable, but that can be fixed by making those arrays REDISTRIBUTABLE or REALIGNABLE.

Rich Fuhler started a tangent discussion by asking “In general, are programmers forced to keep an ALLOCATE statement on one line?” in the current syntax proposals. Guy Steele replied that the directives applied to statements, not proceeding lines, and “Yes, you can write visually confusing things this way.” Rich Fuhler then proposed allocating a POINTER WITH a TEMPLATE, or aligning with another POINTER. Guy pointed out (sorry, no pun intended) that POINTERS as such don't have TEMPLATES. Rich Shapiro noted that of all the United Technologies codes he checked, only 2 or 3 need ALIGNS, but lots need DISTRIBUTE directives. Guy asked if Rich was saying we don't need TEMPLATES? Rich emphatically answered “No”; TEMPLATES are really needed in certain fine grain routines taking 99% of the time in some programs. He argued instead for concise syntax for the most common case, but retaining capabilities for other uses.

Returning to the ALLOCATE syntax, Peter Highnam remarked that syntax 1 and 2 are hacks, but 3 is the future because it brings distribution into the right place. Joel Williamson worried that the maintenance problem is a headache if there are separate comments. James Cownie commented that Piyush Mehrotra wants one declaration for the whole routine, and thought Rich Shapiro was missing this. Joel Williamson remarked that he couldn't see Piyush's proposal, but thought it had the least astonishment value. Clemens-August Thole modified the unseen proposal to make HPF directives additions at the end of a statement, and several other variations were suggested. Richard Swift wondered what happened if ALLOCATE fails, and claimed this was a mark against option 2 and in favor of syntax 3 or Piyush’s proposal. Guy Steele thought it was strange to distribute something that doesn’t' exist yet, but others noted that the system needs the distribution to do ALLOCATE, else it doesn't know the memory requirements. David Presberg concluded that we need to bite the bullet and invent some new keywords; this met with the usual resistance to language additions.

Ken Kennedy called for a point of order “This is the first reading of this proposal, so it is not essential to get the spelling details right today.” The key, he claimed, was whether the group liked the idea of attaching align to allocate, or favored another approach. Maureen Hoffert pointed out that the arguments so far pointed toward adding to the ALLOCATABLE attribute. This effectively supported Piyush Mehrotra’s proposal. Clemens-August Thole proposed leaving the issue for next working group meeting, and was greeted by general applause. Ken, however, wanted to at least take a straw poll to guide the subgroup.

After some further discussion, it was agreed that the basic issue was whether to allow “static” distribution of ALLOCATABLE objects at all, or to always attach the mapping to the ALLOCATE statement. The first question was whether multiple mappings of ALLOCATABLE should be treated as a special case of REALIGN; it passed 17 to 6 with

10 abstains. The next question was whether ALLOCATABLE arrays could be distributed once statically (19 in favor) versus being defined at the point of allocation (7 in favor, with 7 more abstains).

After that series of votes, the group broke for a well-deserved “banquet” of soup and sandwiches.

FORALL and INDEPENDENT

After dinner, Chuck Koelbel presented the FORALL subgroup proposal. This was a first reading; the majority of it will be voted on at the next meeting. The proposal had three parts: the FORALL statement, the INDEPENDENT assertion, and LOCAL subroutines; Chuck presented the first two, and Marc Snir was to present the last one.

The first slide introduced the single-statement FORALL. Chuck prefaced the discussion by clarifying that FORALL is not to be considered as a general-purpose “parallel loop,” but rather as a construct for making simultaneous assignments to array elements. In this respect, FORALL is very similar to Fortran 90 array assignments, rather than (for example) the PCF DOALL. This position was reached in email discussions after it became clear that a deterministic parallel construct was desired, and there was great disagreement on any functionality more elaborate than this. FORALL is a new statement to be added to the language; it cannot be treated as a directive, because it changes the semantics of the statements within it.

- Syntax
  - FORALL (*forall-triplet-spec-list* [, *scalar-mask*]) *forall-assignment*
  - *forall-triplet-spec* as in CM Fortran (i.e. a subscript triplet assigned to an index variable)
  - *forall-assignment* must assign to an array element or section
- Constraints
  - Triplet bounds and strides cannot depend on other indices
  - Mask can depend on the indices
  - Array elements cannot be assigned more than once
  - Side effects in *forall-assignment* cannot affect other instantiations (i.e. computations with other index values)
  - Side effects in left-hand side and right-hand side cannot interfere in the same instantiation

There was relatively little discussion, most of it asking about details of the reasoning about constraints. The side effect constraints are identical in spirit to the constraints on Fortran 90 array assignments; compilers need not check them (as they are undecidable properties, this seems for the best). The independence of triplet bounds was originally in the Fortran 8X proposal; it allows the bounds to be computed in any order, and also makes the task of scalarization somewhat easier. No statements are allowed in the FORALL body except assignments. Andy Meltzer proposed allowing CALL statements as well; this was debated later. The slide on the interpretation of the single-statement FORALL produced more comment

1. Evaluate all bounds and strides in the triplets, in any order. (This produces the *valid set* of index values.)
2. Evaluate the mask for each combination of values in the valid set. (This produces the *active set* of index values.) These evaluations may be done in any order
3. Evaluate the right-hand side for each element of the active set. These evaluations may be done in any order.

4. Assign the computed values to their corresponding right-hand sides.
   These assignments can be performed in any order.

Rich Fuhler questioned whether the definition of active set was consistent with Fortran 90 array expressions, particularly in the case of WHERE masks applied to intrinsic function evaluations. Fortran 90 experts in the audience thought that it was, but asked him to prepare a counterexample off-line if he had doubts. There was some discussion whether the evaluation of functions in right-hand sides following these rules would be generally identical to Fortran 90 array assignments. A consensus eventually developed that there were two appropriate analogies: user functions returning arrays, which are executed once, and intrinsic elemental functions, which are executed once for each element. The semantics given were consistent with elemental functions, satisfying the group. In general, it was agreed by the group that wherever possible the FORALL interpretation should match the array assignment interpretation.

The next two slides proposed the block FORALL (also known as the multi-statement FORALL). This was again proposed as a complex set of assignments rather than a general-purpose parallel loop.

Syntax

- FORALL (*forall-triplet-spec-list* [, *scalar-mask*])
  - *forall-body-list*
- END FORALL
- *forall-body* must be an assignment, array assignment, WHERE, or nested FORALL

Constraints

- Individual assignment statements (always the bottom-level statement) have the same restrictions as in single-statement FORALLs
- Masks, bounds, and stride expressions in nested statements cannot have inter-instantiation side effects
- Inner FORALLs cannot redefine outer FORALL index names

Again, the syntax produced little argument. In general, answers were the same as for the single-statement FORALL. The inter-instantiation side effects on masks, bounds, and strides were meant to refer to interference with the same (lexical) statement, not with other statements. The interpretation slide drew more discussion.

1. Compute outer FORALL bounds and strides. (This produces the *outer valid set.*)
2. Compute the outer FORALL mask. (This produces the *outer active set.*)
3. Execute each body statement in turn for all elements of the outer active set.
   - Assignments and array assignments are interpreted as for single-statement FORALLs
   - WHERE statements compute masks for all outer active set elements, then execute masked array assignments for each assignment nested within them.
   - FORALL generates an *inner valid set* (i.e. bounds and strides) from the outer active set, then generates an *inner active set* (i.e. mask) from the inner valid set. Given this set, apply these rules recursively to evaluate the statement(s) nested in the inner FORALL; that is, evaluate one statement at a time using the (inner) active set.

The intent was to make the interpretation of a block FORALL equivalent to a series of single-statement FORALLs. The major exceptions to this intent were somewhat more general indexing spaces (because inner FORALL bounds can refer to outer indices, triangular and ragged arrays are possible). Evaluation of nested masks is also more fully defined, since Fortran does not have a short-circuit evaluation of logical expressions. All references to nested WHERE statements implicitly include the WHERE-ELSEWHERE construct. WHERE within a FORALL has the same restrictions as in ordinary code, in particular its mask must be an array mask, and the nested assignments must be array assignments rather than ordinary assignments. (Extending WHERE to allow scalar masks and assignments was discussed in the mailing list and rejected as changing the language syntax and semantics.) Several "are these equivalent" examples of block FORALL were given, which didn't necessarily clarify the issues due to possible error conditions in the mask expressions and similar nastiness. Some ambiguities in the draft, particularly in regards to whether function evaluations (and their side effects) were atomic or not, were raised and will be fixed in the next draft.

The final FORALL slide concerned function calls in FORALL statements.

Function calls are allowed in FORALL subject to the following conditions:

- Side effects cannot affect values computed by the same statement in other instantiations
- Side effects cannot affect other subexpressions in the same statement on the same instantiation

John Merlin proposed user-defined ELEMENTAL functions; these were not adopted

John Merlin's ELEMENTAL functions were designed to constrain what was allowed in functions to prevent side effects and avoid random access to distributed memory structures. They were not adopted because there was still intense discussion over particular features allowed in them, not because there was a fundamental resistance to the idea. David Loveman pointed out that these constraints were consistent with both the earlier FORALL constraints and with the Fortran 90 constraints on array assignments. (This had indeed been one of the primary considerations in developing them.) ELEMENTAL functions had been dropped from Fortran 90 for requiring too much support from an already large language. Andy Meltzer took this opportunity to again propose CALL statements within FORALL, arguing that certain subroutines acted in effect as very complex series of assignments. For example, values might be computed for three related quantities simultaneously. Rex Page argued that such subroutines were better handled by returning a structure than using multiple OUT arguments. Others felt that CALL was against the spirit of elemental assignment as FORALL was currently defined. Guy Steele, however, pointed out that MVBITS (a Fortran 90 intrinsic) operated precisely as an assignment, and was extremely important for certain applications run by secret government agencies. He felt that allowing at least that one case in FORALL would be well worth the trouble.

Chuck's final slide concerned the INDEPENDENT directive. This was intended as an assertion to give the compiler more information, not as a new language statement.

Syntax

```
!HPF$ INDEPENDENT
!HPF$ INDEPENDENT(i,j,k)
```

Interpretation

A user assertion that no iteration (of a DO) or combination of
index values (of a FORALL) assigns to a location read or
written by another iteration / combination of values

Does not change the semantics of the construct if the assertion is
true

Not standard conforming if the assertion is false. The compiler can
take any action it deems necessary, including executing the
programmer.

This is a strong assertion, but equivalent to the PCF DOALL (without private variables or explicit synchronization). After some discussion, it was decided to drop INDEPENDENT applied to FORALL from immediate consideration, as there was concern that its meaning was different from the INDEPENDENT DO case. Clemens-August Thole spoke in favor of introducing an interpretation of INDEPENDENT applied to other statement types, and this suggestion was taken under consideration by the subgroup. Limitations of the current INDEPENDENT were pointed out, including that it disallowed reductions and scalars used as temporaries in the loop. PRIVATE variables were suggested, much to the dismay of those who remembered the PCF meetings. The question was raised whether DO INDEPENDENT was a general parallel loop; it was decided that in its present form it was not, because no synchronization or communication was possible.

At this point, Chuck Koelbel was ready to turn the floor over to Marc Snir’s LOCAL subroutine proposal. In the interest of continuity, it was decided to finish discussion of FORALL and INDEPENDENT first. Marc agreed after being promised a chance to present his proposal before the end of the meeting; Ken Kennedy said he would be the next topic that night.

Marina Chen began the discussion by asking, “What about Alan Karp's comments?” This was in reference to a recent -- and very vocal -- discussion on the mailing list, in which Karp claimed that FORALL was unnecessary because it could always be simulated by DO loops and temporary arrays. (His comments, and some of the responses to them, were distributed as handouts, and can be found in the FTP archives.) Rich Shapiro, although he was one of the staunchest defenders of FORALL, tried to present Karp’s views fairly based on an extended discussion they had had. Karp was correct that DO INDEPENDENT and temporary arrays suffice to provide the functionality. It could similarly be argued that DO is unnecessary because GOTO can be used to construct loops, although the analogy is not really fair to the anti-FORALL camp. Karp’s next criticism was that FORALL is dangerous because statements in it have different semantics. For example, the statement

```
A(I) = A(I) + A(I-1)
```

has a markedly different affect inside DO and FORALL loops; one is a prefix sum operation, while the other is a shift and add. Rich admitted this was a problem, particularly in block FORALLs, but was willing to accept it for the other benefits of FORALL. Finally, Karp gave an example of a computation that was much clearer when written without FORALL than with it; Rich countered with a different example that was much longer without FORALL. As Rich put it, “Readability is in the eye of the beholder”. Having both examples suggested to several people that both constructs (FORALL and DO INDEPENDENT) were needed.

Clemens-August Thole argued that FORALL is not intuitive, since the HPFF group itself took so long to agree on its semantics. Rich Shapiro noted that Fortran 90 has other nonintuitive features, such as SUM under a WHERE (see previous discussion of FORALL); he reiterated that programmers will make mistakes, but the benefits when

FORALL is used correctly outweigh the bugs. Dave Loveman opined that the real danger in this discussion was that we knew too much about FORALL; some of the group expected a “parallel loop” and were trying to force FORALL into that mold. Ken Kennedy asked which is the greater oxymoron: “Trust the user” or “Trust the compiler”? Marc Snir emphasized the main point regarding FORALL, that HPF was introducing a new construct not related to the data mapping it was supposed to do, and opposed FORALL on those grounds. Peter Highnam responded that FORALL was correcting brain-dead compilers. Others mentioned that it corrected an omission of Fortran 90. A short digression reviewed of why FORALL had been dropped from F90 (to simplify the language), and noted the usefulness of the Fortran Journal of Development (an F90 appendix for a time) in documenting these features. Marina Chen noted that if FORALL was not included, then implementations would start to diverge, and Richard Swift chimed in that users want this functionality.

Joel Williamson brought up the question of whether FORALL must be part of the HPF subset, or as he put it “Is this decision all or nothing?” Ken Kennedy recommended voting to accept it first, and later move to put it in the subset if the group felt that was appropriate. He referred to the earlier straw poll that restricted the subset to a Fortran 90 subset. There was general agreement to resolve whether to put FORALL in the subset next time.

A series of straw polls was then started.

- Should HPF have at least the single-statement FORALL? Yes 28, No 4, Abstain 4
    - Should HPF leave FORALL out if it could be in a superset? Yes 2
    - Should FORALL be in HPF but not in the official subset? 15
    - Should FORALL be in the official subset? 16
    - Abstain from the subset/superset votes 5
- Should HPF have a block FORALL with at least the following features (with the currently proposed semantics in each case)?
- Only assign & unmasked array assign? 18 yes 7 no 9 abstain
    - WHERE? Yes 16, No 6, Abstain 13
    - FORALL and WHERE? Yes 13, No 8, Abstain 15
    - (Jokes regarding voting for “none of the above” and/or Ross Perot had been building for some time; this vote brought them out in force, too many to record.)
- Should CALL be allowed in FORALL (under the same restrictions as functions regarding side effects)? This vote was delayed pending the next, more fundamental, issue.
- Should function calls in FORALL be required to have no side effects except to return a single value? Yes 15, No 12, Abstain 8
    - That made subroutine calls much less interesting by themselves.
- Should HPF allow CALL in FORALL if side effects are later allowed? Yes 7, No 14, Abstain 12
- Should HPF include the INDEPENDENT assertion for DO loops? Yes 27, No 2, Abstain 5
    - Ken Kennedy’s immediate reaction was “We should run this for President!”

An attempt was made to poll on INDEPENDENT for FORALL, with the intended meaning that no location written for one combination of index values is read or written for another combination. Other meanings, including assertions that synchronization conditions between statements could be relaxed, were also proposed. The group finally postponed discussion of the issue, since it was well after 10:00pm. They agreed to resume at 9:00 the next morning.

In the morning, Min-You Wu presented another example of a possible use of INDEPENDENT in FORALL.

```
a = 0
forall ( i=2:n, a(i-1)=0 )
    a(i) = 1
end forall
```

His question was, “Does INDEPENDENT cover the mask expression of a FORALL?” Using this and examples from a proposal he had e-mailed earlier, he argued for INDEPENDENT blocks to control synchronization operations in FORALL. Piyush Mehrotra and Ken Kennedy pointed out that INDEPENDENT for FORALL was not defined yet, and recommended considering these issues in the FORALL subgroup. Clemens-August Thole agreed, noting again that there is no reason to restrict INDEPENDENT to FORALL and DO statements.

Local Subroutines

Marc Snir then began his presentation of LOCAL subroutines (as Ken Kennedy said, “only 12 hours after starting this discussion.” There were 2 proposals for this functionality in the email list, one from Guy Steele and the other from Marc. Guy’s proposal designed a feature internal to HPF, while Marc’s defined an interface to external routines, or as Marc put it “really an escape to code outside HPF.” Such a feature is needed because HPF needs to interact with non-HPF code (SPMD libraries, message-passing code, etc.). Guy remarked that the two proposals were not in conflict, although they had different origins and goals.

Marc’s first slide presented conventions enforced by the HPF compiler on the called external routine.

- INTERFACE blocks for declarations in HPF, marked by the keyword LOCAL
- Synchronous transfer of control from HPF to the external routine
  - All processors call the routine, execute cooperatively, and later transfer back
  - This proposal does not specify what happens in the routine
- Alignment and/or distribution are performed as required by the INTERFACE block
  - Distributions that the routine expects to see, HPF enforces (as for regular routines)
  - Allow libraries to assume a given distribution (otherwise lots of queries would be necessary)

Relatively little discussion on this interface was necessary, immediately mostly just clarifying points on the original slide. (I smooth out grammar and fix handwriting for these notes.)

The next slide placed some requirements on the LOCAL routine to ensure smooth functioning with HPF, and mechanisms to access HPF data structures from those routines.

• Requirements from the external routine

Keep multiple copies of data in a consistent state (on return from the routine)

•. Mechanism for external routine access to HPF data structures

Array descriptors are passed as arguments along with data

These are explicit arguments in the external routine, not in the HPF call itself

Intrinsics for use of array descriptors

Query functions

Functions to access data using global indices

The format of an array descriptor is not specified in the proposal, just functions for encapsulation of its operations. These can be supplied as a standard module in Fortran 90, or by other mechanisms in other languages.

Marc’s next slide had only one point.

HPF calling conventions and external restrictions make sense for both SIMD and MIMD machines; does the external data access mechanism also?

SIMD vendors should check it (the proposal was written with MIMD in mind)

This opened the floor for discussion. Andy Meltzer asked if SIMD or MIMD style synchronization was assumed in the LOCAL function. Marc replied that it is a black box interface so any paradigm can be used internally. He speculated that EXTERNAL is probably a better name for the feature. Richard Swift suggested “FOREIGN” as a keyword, since LOCAL and EXTERNAL already have meanings in Fortran 90.

Piyush Mehrotra asked if the external could access nonlocal data; Marc replied that FETCH and STORE were defined in the proposal, but the list there was not exhaustive. Piyush Mehrotra followed by asking if a local or global index space was used for HPF arguments within the function, and Marc replied that local was used, with intrinsics for global indexing operations. Clemens-August Thole questioned the need for FETCH and STORE at all, saying that communication could be handled in other (possibly more efficient) ways for a given machine. Marc said he wanted to avoid syntax arguments in this talk. Guy Steele asked about the type declaration for descriptors; Marc admitted that it was an error in his proposal to omit them.

Peter Highnam argued that this defined a foreign interface, so HPF can't say what is on the other side. He agreed that the functionality of descriptors was needed, but claimed that we shouldn't do syntax in this forum. Finally, requiring foreign code to maintain consistency is incorrect since it cannot be checked. Marc replied that a correct external routine must have consistency on return, otherwise the HPF code is in deep trouble. Richard Swift asked if an external routine can redistribute HPF arrays; Marc replied, “Not in this proposal.” Chuck Koelbel and several others pointed out that staying consistent with normal routines in this respect was a very good idea. Mary Zosel commented that she still can't get to the right library directly with Marc’s proposal, but at least it gives a wrapper layer.

David Presberg argued that HPF shouldn't try to define syntax in external routines, just the minimal information flow over interface. His example was using this feature to call a C library, where syntax would be problematic at best. Ken Kennedy took the other side, arguing that it was very useful to specify the FORTRAN intrinsics to avoid divergence in compiler extensions. Randy Scarborough used the example of matrix multiplication to show the usefulness of defining a module whose only responsibility was partitioning arrays. Dave Loveman agreed, but claimed that to do this HPF should give *very* exact Fortran 90 interface.

Clemens-August Thole defined 3 layers of intrinsics for external routines:

1. How is this array distributed?
2. FETCH and STORE operations
3. Pure message passing

He claimed that 1 was clearly needed to program the external routine, and 3 is being standardized by another group (who first met at Williamsburg, VA, in April). The remaining category is 2, which he claimed had too many subtleties to standardize quickly in this group.

Barbara Chapman asked if the proposal was assuming array sections passed as arguments were stored contiguously. The answer was no; the storage scheme would be hidden in the descriptor access functions. Rich Shapiro and Marina Chen argued for keeping the proposal fully general by not specifying storage structures. Marc Snir said he had tried to stay mute on this issue, but the proposal may have to expand the list of intrinsics. Clemens-August Thole pointed out the importance of overlap areas in scientific computations. James Cownie claimed that both overlaps and Marc's proposal do not require contiguous storage, just that the array descriptors exist so that they can be used. Andy Meltzer mentioned that more people want to escape to C than to Fortran 90 due to the availability of C compilers. James assumed that everyone will have Fortran 90 to C links, and that these should generalize to the HPF external linkages. David Presberg noted that external compilers may not adapt to HPF.

Guy Steele reiterated that his and Marc's proposals were not mutually exclusive, and recommended combining them. The group concurred, and scheduled a first reading of the combined proposal for the next meeting.

Storage and Sequence Association

After the LOCAL subroutine discussion, Ken Kennedy called for a break so everybody could read the storage and sequence association proposal (which had been put in final form the night before, and had copies distributed that morning). Before splitting up, Mary Zosel announced a couple of changes to the printed document:

On page 2, near the bottom, a line of italics should be changed to plain text. (Somebody quickly asked, "Is this a printing discussion?")

On page 3 and following pages, EDD was undefined; it should be "HPF alignment or distribution directive." (It was originally "*E*xplicit *D*ata *D*istribution.")

The group agreed that if that was the level of corrections needed, the proposal must be in good shape. A 20 minute break began.

After the break, Ken Kennedy presented the storage association proposal. This was a first reading of that proposal, since it had been changed so heavily from the last meeting. His first pair of slides concerned definitions used in the proposal.

- Aggregate storage sequence: storage in variables that are associated by EQUIVALENCE & COMMON
- Aggregate variable group: a collection of variables whose storage sequences are associated to an aggregate storage sequence
- Aggregate variable group: a collection of variables in an aggregate storage group
- A variable is sequential if and only if any one of the following hold
  a. It is in a sequential COMMON

b. It is in an aggregate variable group
c. It is an assumed size array
d. It is a component of sequenced type
d. It is declared sequential

- Components: variables and aggregate variable groups in COMMON
- A COMMON is nonsequential if and only if all of the following hold
  a. It is not explicitly declared sequential
  b. Every instance of the COMMON has the same number of same-size components
  c. The type, shape, and mapping of any explicitly-mapped variable or variable group are the same in every program unit in which the COMMON occurs

Robbie Babb pointed out that there are lots of possible conflicts in the COMMON definition, and asked if the default was sequential or nonsequential. Ken responded that the issue was dealt with later, but for now assume that nonsequential is the default. The next slide gave some examples to illustrate the definitions.

```
common /foo/ a(100), b(100), c(100), d(100), e(100)
real x(100), y(150)

EQUIVALENCE (a(1),y(1)

aggregate variable groups: (a,b,y), c, d, e, x
sizes: (a,b,y) = 200, others 100
```

Piyush Mehrotra asked about inconsistent declarations of COMMON; Ken replied that they make the COMMON sequential. The subgroup had decided it was more important to support consistently-declared COMMONs than full reshaping. David Presberg noted that implementing full storage association is a heavy burden on vendors. Piyush claimed that the same problems happen here, because of EQUIVALENCE. Ken noted that the effect Piyush wanted could be achieved by just declaring a totally overlapping variable, which would force the aggregate groups in all routines to be equivalent. After a long discussion, Piyush agreed to provide an example of what he was proposing, since everyone appeared confused.

Ken’s next slide gave the storage association rules for HPF. In this context, “mappable” means “can be used in an explicit HPF mapping directive.”

1. A nonsequential array variable is mappable. A variable in an aggregate variable group is mappable if its storage sequence is totally associated (i.e. completely covers) the aggregate storage sequence. Only one variable in an aggregate variable group may be explicitly mapped.
2. The result variable of an array-valued function that is not an intrinsic is a nonsequential array.

Ken noted that the only sequential variables that can be mapped are covering variables; this replaced the capability of distributing the whole COMMON, as had been done in the previous proposal. An example was meant to clarify the rules.

```
common /two/ e(10,10), g(10,100,1000), h(100), p(100)
real cover(200)
EQUIVALENCE (cover(1), h(1))

align e...    OK, since it is in its own aggregate variable group
align cover...   OK, since it covers the whole of its group

can declare /two/ e(10,10), g(10,100,1000), x(200) elsewhere
```

The next topic was the sequence association rules. These had been presented in the same form at the last meeting, and so this was treated as a second reading.

1. When an array element or the name of an assumed-size array is used as an actual argument, the associated dummy must be a scalar or a sequential array.
   An array-element designator of a nonsequential array may not be associated with a dummy array argument
2. A variable of default type CHARACTER (scalar or array) is nonsequential if it conforms to the requirements of definition DE-4.
   The size of a nonsequential, explicit-length, CHARACTER dummy argument must be the same size as the actual argument.

In the discussion, rule 2 was translated as “If you want to map the arrays, the sizes must match.” A short discussion of assumed-length character strings followed, and the subgroup will look at that issue some more.

Returning to the storage association proposal, Piyush Mehrotra presented his examples for extending the current proposal.

```
common /one/ q(100), r(100), s(100)
real u(200)
EQUIVALENCE u(1), q(1)
u & s are mappable: that’s good

common /two/ f(100) g(100) h(100)
real p(100)
EQUIVALENCE p(1) f(50)
p not mappable, (p,f,g) is an aggregate variable group

common /three/ a(100), b(100), c(100)
common /three/ d(150), e(50), c(100)             !new routine
c not mappable, but there is no aggregate variable group in either routine
claim: c should be mappable
```

David Presberg started his response with the claim, “I'm about to put my foot in my mouth.” In COMMON /two/ f and g are clearly aligned, so h can reasonably be aligned as well. In COMMON /three/, there are different parameters early in the sequence, so there is no guarantee that array c can be treated consistently in a separate compilation system. Ken Kennedy defended Piyush, claiming that in any reasonable implementation, if /two/ works, /three/ works as well. Clemens-August Thole and Pres disagreed, noting that without SEQUENTIAL (in Fortran 90), there is no guarantee that no padding is inserted in COMMON. Joel Williamson suggested a goal for compiling HPF: each array in COMMON acts like its own COMMON (in the absence of EQUIVALENCE).

Randy Scarborough claimed that the restrictions placed on COMMON would mean that HPF needs explicit NOSEQUENTIAL declarations, otherwise separate compilation cannot check whether COMMON blocks are declared sequentially. Richard Swift asked what problem we were trying to solve, to which Piyush Mehrotra replied COMMON /three/ should be mappable. After some discussion, Guy Steele asked how far Piyush wanted to go in applying sequential storage models. Piyush claimed that the length of arrays should be the only consideration in defining aggregates. Peter Highnam remarked that this was getting baroque, at which point Ken asked Piyush if he were trying to kill the storage association proposal. Peter continued by asking how to explain this machinery to users, particularly since modules are coming and would avoid much of the need for it.

Ken tried to summarize the discussion so far:

Piyush Mehrotra doesn't like requiring identical declarations of COMMON blocks in order to map the arrays.

Peter Highnam uses this as argument against relaxing original rule, based on the difficulty of explaining the resulting rules to users.

Marc Snir characterized it as “Piyush wants to rely on storage association to decide when to break storage association.” Ken commented that his statement was true, but not a balanced view of the argument. Barbara Chapman stated that Piyush’s ideas are easier to implement than understand, noting that they are very similar to what is already in Vienna Fortran. The compiler should be able to figure out aggregates without explicit EQUIVALENCE statements everywhere. Randy Scarborough asked what the alternative was; the only possibility seemed to be that a compiler sees nonsequential COMMONs, which must be consistent or the program will break on some machines. Robbie Babb made a radical proposal: COMMON blocks can be declared as mappable or not mappable; he was informed that that was the original proposal, and this was an attempt to get around those limitations.

Chuck Koelbel argued against Piyush’s brand of storage association by converting the example to an extreme case:

```
common /three/ a(100), b(100), c(100)
common /three/ d(199), e, c(100)              !new routine
Can you distribute c now?
```

He argued that this would be likely to break many machines. Ken noted that this proposal was just trying to help users porting large codes with lots of distributions. Don Heller believed that the coding rule that would come from this would be to allow only one array per COMMON, to avoid accidental aliasing. Andy Meltzer noted that the future will discourage all of this, but Ken replied that this is all for interim porting problems in the first place. Randy Scarborough proposed requiring COMMON blocks to be declared identically to solve many of these problems.

Richard Swift recalled the history of the proposal: it started as a very simple proposal, then Ken Kennedy extended it to allow some useful cases of association, and now we're talking about extending it again. Ken said he would take it as a friendly amendment that COMMON blocks must be identical everywhere they are declared, including EQUIVALENCE declarations. The discussion shifted somewhat to finding the right terminology for this. Clemens-August Thole asked why distinguish the proposal distinguished components from aggregate variable groups; this was an artifact of how the sections were written. Randy Scarborough pointed out a potential problem if no EQUIVALENCE spans a variable in COMMON, leading to lots of potential technical rewriting. Marc Snir suggested wording the conditions as an identical sequence of aggregates (i.e.. as Ken already intended, but with clarification about what “identical” means). Peter Highnam claimed that identical is too restrictive, using as his example

```
COMMON /foo/ complx(1000)
COMPLEX complx
REAL realx(2000)
EQUIVALENCE (realx(1),complx(1))
```

Some questioned why this was considered doing users a favor. Piyush favored the "identical" restriction, saying that it at least gave some freedom to programmers.

After a short discussion of some minor points, the proposal was brought up for a straw poll as a first reading. The vote to accept storage association (subject to details of defining identical COMMON matching) was 26 yes, 1 no, and 5 abstain.

The discussion then moved quickly to sequence association. The first substantial issue opened was whether assumed-length CHARACTER arguments could be distributed. Ken Kennedy moved to send this question to the distribution group, who should consider it in the context of ALIGN WITH *. Clemens-August Thole asked whether there was a problem with array sections; since these cannot be sequence associated, there was not. Rich Shapiro asked if assumed-size arrays could be distributed, and the issue was again kicked back to Guy Steele's distribution group.

Pending clarification of those issues, an official vote to accept the sequence association proposal was taken. The proposal passed 20 to 0, with 2 abstentions.

Low Performance Fortran

Before moving to lunch Andy Meltzer made his now-traditional presentation of Low Performance Fortran (LPF). As he described it, this is another language model "real similar to HPF." For the attendees, he had the disclaimer "If you recognize something you think you said in this presentation, it probably means that you did say it, and I don't mean any offense by quoting you." After seeing the first few points, Ken Kennedy remarked, "All I ask is that you don't circulate this slide"; unfortunately, it was already too late.

- Low Performance Fortran II
  - LPF varieties of FORALL (to be thought of as a looping construct)
    - FORSOME (*triplet-spec* [,*mask*]) *stmt*
      - The set of iterations executed are implementation defined
    - FORONE (*triplet-spec* [,*mask*]) *stmt*
      - Statement is executed once
    - FREEFORANY
      - Some statement in the program is executed
    - ONCE_AND_FORALL
      - This is the *last time* the statement is executed
    - BASKIN_ROBBINS_FORALL
      - 31 varieties, but you can't see them all at once and their descriptions are so similar that you can't tell what you voted for in the straw poll
    - THERE_EXISTS
      - Asserts that there is one, but since it's an existential statement, it doesn't matter
  - Pointers
    - It is impolite to point
  - !LPF$DEPENDS_ON [line #]
    - Asserts that the following statement depends on line #
  - LPF Superset
    - Contains addition and multiplication (regular LPF makes do with subtraction and division)
  - TEMPLATES

Some templates are “much-alike”. If you want the same thing to happen to much-alike TEMPLATES, run them over with the same train

COMMON

Another name for COMMON is GAUCHE. There is only GAUCHE data and dummy arguments (called parameters)

Actual & dummy arguments

Actuals are copied to a randomly chosen PE's memory

Parameters are hunted for by other PE's & copied back

!LPF$ SOMEONE_ELSES

Will choose some other application on machine and execute it

LPP Low Performance Pascal (sister group)

Have decided to go with Ada

While marginally harder to implement, data hiding deemed worth it

Will retain name to mislead users

Coming Soon - To a Forum Near You!

OPF - Obfuscated Performance Fortran

Intrinsics

After lunch, Rex Page presented the proposal from the intrinsics subgroup. The reference document was from Rob Schreiber (who could not attend the meeting), dated July 22; there was some expansion from the draft at the last meeting, but the spirit was the same.

Rex’s first slide treated the difference between intrinsics and library functions.

- Only some procedures should be intrinsics.
- A Fortran 90 intrinsic is a procedure that is provided as an inherent part of the language processor (compiler).
- There are two types of intrinsics: those allowed in specification expressions and those that are not.
- We recommend: NUMBER_OF_PROCESSORS and PROCESSORS_SHAPE are intrinsics usable in specification expressions, and the rest of the functions defined here are not intrinsics.

The first point brought up discussion was the significance of specification expressions: they can be used in declarations, for example as array bounds. Andy Meltzer pointed out that for reconfigurable machines or operating systems that allowed partitioning of processors (that is, virtually all MIMD machines), the recommendation made NUMBER_OF_PROCESSORS and PROCESSORS_SHAPE load-time constants, which has a major implementation impact on handling symbols. Piyush Mehrotra and Rex clarified that these functions are inquiries about the physical machine, not the HPF PROCESSOR grids. Piyush suggested adding options for distribution inquiry; that was taken under advisement.

Mary Zosel started a long discussion by asking, “Are we assuming a static number of processors in HPF?” Chuck Koelbel came down firmly on both sides of the issue, pointing out that if the number of processors was not static, then the distribution patterns as they stand don't make sense (their definitions require the number of processors). However, there had been some interest from other groups in porting HPF to workstation networks or other situations that allowed (or required) dynamic process creation. By the end of the discussion, both were sorry they had brought the issue up. Alok Choudhary claimed that physical processors are not covered in the rest of the HPF model. Guy Steele noted that this is at bound of machine-independent and dependent features, and HPF may be able to do something with this. Rich Fuhler suggested adding the constraint that the function must evaluate to a constant if used in a specification expression context. Marc Snir noted that there is no way to associate abstract and physical processors within HPF; Guy replied that HPF does have the PROCESSOR size constraint (implementations must allow a processor grid of size NUMBER_OF_PROCESSORS()). Ken Kennedy recalled awful arguments about this issue in PCF and urged steering clear of it. Guy Steele observed that when the NUMBER_OF_PROCESSORS constant is chosen is an implementation issue, not a language issue. Richard Swift said he liked the intent, but questioned whether this was the interface we want.

Finally, a straw poll was taken: those in favor of NUMBER_OF_PROCESSORS() as proposed 23, opposed 1, abstain 7.

The next slide covered the first set of recommended standard library functions.

- Extended MAXLOC/MINLOC
  - Add an optional argument to treat the main argument as an array of vectors
  - Extended function delivers the maximum/minimum value for each vector

The discussion was much briefer than previously. The first question was whether overloading the function name forces the extended functions to be intrinsics; quick reference to the Fortran 90 standard indicated not. Rich Shapiro noted the extensions give the same semantics as MAX and MIN, thus fixing a Fortran 90 shortcoming. Returning briefly to the “intrinsics” question, Marc Snir asked what the difference was between a standard library and intrinsics. Richard Swift noted an interface difference, namely that the programmer must include modules declarations for libraries (particularly when the function names are overloaded, as MAXLOC is). Ken Kennedy claimed the distinction is small, but Marc thought it was important. He argued that added intrinsics are extending Fortran 90 generally, not just in the areas of data distributions and parallel computation as HPFF originally intended. Don Heller suggested that maybe a better design would be to generate APL functionals in a library rather than new intrinsics. Guy Steele had two responses: “That's what I thought I was doing” and “I'm not sure it makes sense to recreate all of APL in HPF.”

Rex then took a vote on whether some form of this (i.e. either library functions or intrinsics) should be in HPF. 13 voted yes, 1 voted no, and 10 abstained.

The next slide covered a new set of functions

- Elemental bit inquiries
  - POPCNT - Number of 1's in the binary representation of an integer
  - PARITY - Parity of the binary representation of an integer
  - LEADZ - Leading 0 bits in the binary representation of an integer
  - ILEN - Number of bits needed to represent an integer

The discussion was very brief. Most of the questions were answered when Rex pointed out that these are elemental functions and work like SIN. Several implementors affirmed that all were needed on many machines (although ILEN can be constructed from LEADZ). A straw poll on whether these functions should be included drew 5 yes votes, 0 no votes, and 16 abstains.

Yet another slide introduced more functions.

- New reductions

AND - corresponding to the IAND operation (bit-wise AND)
OR - corresponding to the IOR operation (bit-wise inclusive OR)
EOR - corresponding to the NEQV operation (bit-wise exclusive OR)
PARITY - corresponding to the parity operation

Again, this just fixed some features (associative and commutative operations) that had been left out of Fortran 90. There was no discussion. The straw poll to include these functions found 13 yes votes, 0 no votes, and 13 abstains.

The next slide defined a very broad class of functions.

- Parallel prefix and suffix operations
  - XXX_PREFIX - for any reduction function XXX (including the new ones defined above)
  - XXX_SUFFIX - for any reduction function XXX
  - Each delivers a series of partial reductions (sums, products, etc.) of the input array
  - The result is defined for all output array elements, even when the (optional) mask is false
  - Reductions are restarted for each segment; segments are portions of ARRAY corresponding to runs of like values in SEGMENT (a LOGICAL array optional argument)
  - Partial reductions start at the identity element for the appropriate operation

The discussion was short, but spirited. Some off by one details of the SEGMENT feature were referred back to the proposal. Dave Loveman noted that this creates lots of functions (particularly when overloading by number of array dimensions is taken into account), and asked if HPF needed all of them. Chuck Koelbel asked which ones he wanted to leave out, and Dave noted that the full set would take valuable implementation time. Ken Kennedy referred to a proposal by Demmel (on the mailing list) regarding prefix operations on 2-by-2 matrix multiplies; Guy Steele wanted more input, particularly on the numerical aspects, before recommending those features. Marc Snir noted that all the functions could be moved to a library by overloading, but Rex Page noted they were hard to write just due to volume. Guy claimed that most of the implementation difficulty is in testing rather than writing. Richard Swift was concerned about the library size; Andy Meltzer concurred.

A straw poll was taken on whether to include the prefix operations in HPF, giving 11 yes votes, 0 no votes, and 19 abstains.

The next slide extended reductions in another way.

- Combining send procedures
  - XXX_SEND(SRC, DEST, IDX1, ...)
  - XXX is any Fortran 90 or HPF reduction
  - CALL SUM_SEND(a,x,v) operates like x(v) = x(v) + a but all elements assigned to a corresponding to duplicate element in v contribute to the final value
  - The number of IDX arguments is the rank of the destination array
  - Source and all IDX arguments are conformable

The intent is to allow parallel accumulations of many irregularly related array elements in a single call; for example, one call can compute a histogram. CALL SUM_SEND(a,x,idx) is equivalent to the following loop:

```
DO i = 1, n
      x(idx(i)) = x(idx(i)) + a(i)
END DO
```

The discussion was lively. Clemens-August Thole suggested avoiding the name SEND because of possible conflicts with message-passing routines; the subgroup agreed to look for a better term. Piyush Mehrotra asked why these were subroutines, and not functions; given current syntax and semantics, they can't be called from FORALL constructs. Guy Steele saw no strong reason against the function form. Rich Shapiro noted that the same operations can be done with FORALLs, but Guy responded that those constructs give the compiler problems in idiom recognition. Mary Zosel smelled something extremely important for unstructured mesh problems in these subroutines, and was greeted with a general “Right!” from the crowd. Richard Swift noted that HPF now required lots and lots of library functions (by his count, over 400 after resolving overloading); David Loveman agreed. Ken Kennedy asked if HPF needed this support for irregular computations, given that it doesn't have irregular distributions. Rich Shapiro replied that these functions are very useful, even without distributions of any kind. Ken was still concerned over the interaction of this feature with others. Richard Swift said his concern was the interface, not the functionality.

Guy Steele made two “meta-remarks” to help clarify the situation. No single one of these functions is vital for HPF (although each has some real application associated with it). But he had promised HPF (particularly the intrinsics group) to give them all the things Thinking Machines had found useful, so the names, order of arguments, and other details could be standardized. Andy Meltzer commented that these libraries take a lot of time to develop, and HPFF should decide in November whether to include them in the HPF subset, superset (if any), or whatever.

With that, a straw poll was taken on whether to include these functions in HPF. There were 8 yes votes, 0 no votes, and 21 abstains. Another vote was taken on whether anybody objected to making these operations functions, finding 0 opposed.

There was only one slide remaining, and little time left before the meeting was scheduled to end.

Sorting functions
- GRADE_UP - Sorts in ascending order
- GRADE_DOWN - Sorts in descending order
- Each delivers a list of subscripts for arranging the array in sorted order (i.e. a permutation array)
- If optional argument DIM is absent - The result has shape [size(shape(ARRAY),product(shape(array))]
- If DIM is present - The result has the same shape as array, and is sorted along the appropriate dimension

Maureen Hoffert asked if sorting was limited to integer and real arrays. Guy Steele responded that any standard type with a “less than” operation defined was allowed. Ken Kennedy suggested sorting could take a comparison function (like UNIX qsort). Robbie Babb asked what sorting had to do with High Performance Fortran. Rich Shapiro noted that it is hard to do parallel sorts (and many other parallel functions); a good philosophy was to let vendor do it instead of the applications programmer. Chuck Koelbel noted that sorting is needed for particle push phases in PIC algorithms, certainly a high-performance application. The straw poll on including the sorting functions found 12 yes votes, 0 no votes, and 16 abstains.

Richard Swift asked for a poll on the intrinsics vs. library question. Rich Shapiro noted that most of the functions proposed were now implemented as calls in the CM Fortran compiler, but it “doesn't follow the rules” on type checking in some cases. If there is strong type checking, the library needs lots more code. Robbie Babb asked if generic functions help the library situation; Rex Page wondered if it was legal to overload intrinsics (the language lawyers believed so) and what the implementation effects were (no answer). Ken Kennedy wanted a description of the pros and cons for intrinsics versus libraries before putting it to a vote, and the group agreed this was wise.

Official HPF Subset

After a short break, the group considered the status of the official HPF subset. The discussion started (rather than ended) with a straw poll: Given all the new features should HPFF allow HPF features to be omitted from the official subset? 22 voted yes, 1 no, and 5 abstained.

Mary Zosel suggested that each subgroup should propose what goes into the official subset from the features they considered. Ken Kennedy recommended that Mary and her group prepare a proposal to the effect that HPF features can be out of subset; assuming that passed, future meetings would allow motions to remove specific features from the subset. The reasoning behind this was that the group needed to know what was in the language before deciding what was in the subset. As Marc Snir pointed out, the argument for removing features from the language (or subset) is size, so we need to know how big it is.

Editorial Matters

With all the proposals, counter-proposals, and other discussions on the email lists, Mary Zosel asked whether somebody could post summaries to the net? Reading all the HPFF mail is too much. Chuck Koelbel muttered “I know” under his breath. After a short discussion, in which it was made clear that such postings would have to be automatic, David Presberg suggested posting just the mail headings regularly and directed Chuck to some software that might do this. He is looking into that possibility.

Dave Loveman summarized the status of the (proposed) High Performance Fortran language document. Chuck Koelbel had proposed two outlines, both of which had the problem that they required too much writing to transform the proposals being made into the right form. He had produced a new outline that would hopefully balance readability and completeness. In essence, each group would work on its own chapter, integrated into a single outline. The subgroup chairs had seen the outline, and most had agreed with the structure. by next meeting he hoped to get a first draft from each group and have a full (albeit rough) document available. Ken Kennedy noted that little effort now will be worth it later. Mary Zosel asked that he send the whole outline to the core group, which Dave agreed to do once some packaging issues had been resolved. Dave and Chuck Koelbel would collaborate on providing FTP access.

Meeting Schedules

The final topic for discussion was the schedule for the next HPFF meeting, to be held September 9-11 in Dallas. The topics to be discussed included

- Second readings
  - Storage association

- Subroutine linkage for distributed arrays
  - Distribution for pointers and allocatable arrays
  - Intrinsics considered at this meeting
  - FORALL
  - INDEPENDENT (for DO only)
- First readings
  - Local subroutines
  - New intrinsic functions
  - Fortran 90 subset
  - INDEPENDENT (for FORALL and other constructs)
  - Parallel I/O

As Ken Kennedy put it, we have lots of work to do. Consensus was that the usual one-and-a-half day meeting was not enough. Piyush Mehrotra was the first to suggest extending the meeting into the afternoon of the last day, and the rest agreed. After a short discussion of flight availability, the group decided to run to about 4:00pm on the last day. At Mary Zosel's suggestion, the subgroup leaders will get together for dinner the night before to set the meeting agenda and other details. Marc Snir suggested distributing the proposal documents for first and second readings at least one week in advance, in order to speed up the discussions. Mary Zosel generalized this to distributing the documents to the hpff-core mailing list, not just the subgroup lists as had been done in the past.

Ken Kennedy ended the meeting with a short report on the HPF Birds-of-a-Feather session held at ICS. About 12 visitors, including Bob Voigt from NSF, had attended that meeting and provided comments on the language. Ken particularly mentioned some helpful comments from John Ried. Most of the questions there had been raised in the discussions, either in the plenary sessions or in the subgroups. The next major opportunity for this kind of feedback appeared to be the workshop at Supercomputing '92, where the near-final draft of the language will be distributed. Ken hoped that this would allow a great deal of feedback before the December HPFF meeting, where the group would have a chance to reconsider language features that had been voted as "frozen" so far. Piyush Mehrotra suggested that HPFF should move the December meeting back, in order to allow more time for feedback. After some discussion, the group decided that December 9-11 had no conflicts from potential attendees; pending checking dates with the Dallas hotel, the meeting will be rescheduled then.