HPFF Meeting Notes for the September 10-11, 1992 Meeting

High Performance Fortran Forum

CRPC-TR93315
May 1993

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

HPFF Meeting Notes

September 10-11, 1992
Dallas, TX - Bristol Suites Hotel
Notes taken by Chuck Koelbel
With help from Mary Zosel

Executive Summary

This was the fifth meeting of the High Performance Fortran Forum working group. In general, the HPFF effort is on schedule for producing a preliminary draft for public presentation at Supercomputing '92, and a "final" document by December. A follow-on effort in 1933 is also possible, and suggestions for features included then continue to come up at meetings.

Some of the more important outcomes of this meeting were

Documentation: A (very rough) full draft now exists, and may be retrieved electronically. See the "HPFF Documents" section below for more details. A revised draft reflecting decisions made at the September HPFF meeting should be available in early October.

Data distribution: The working group accepted a data distribution proposal including features for subroutine interfaces for distributed arrays, ALLOCATABLE arrays (without pointers), and the basic TEMPLATE/ALIGN/DISTRIBUTE model. Due to strong arguments on the mailing lists, the group agreed to consider eliminating the TEMPLATE construct at the October meeting. The distribution proposals will also be modified to de-emphasize the role of TEMPLATEs (in particular, to emphasize the fact that they are syntactically optional).

Intrinsics and Library Functions: The working group accepted three new intrinsic functions: NUMBER_OF_PROCESSORS, PROCESSORS_SHAPE, and ILEN. Extensions to MAXLOC and MINLOC were also accepted as intrinsics. All other functions proposed as intrinsics previously (including bit manipulation, accumulation, prefix, suffix, scatter, and sorting operations) were accepted as a standard library.

FORALL and INDEPENDENT: The group accepted a proposal defining FORALL, in both single-statement and block forms. Function calls within a FORALL will be limited to procedures without side effects (except for the side effect of returning values, of course); the specification of such PURE functions will be formalized at the next meeting. An INDEPENDENT assertion for loops was also accepted which disallowed all loop-carried dependences, including reduction operations. Other, less restrictive, assertions will be proposed at the next meeting.

Storage and Sequence Association: The group accepted a proposal to allow very limited explicit distributions of equivalenced variables (essentially, if one variable completely covers an equivalence ring then that variable can be distributed, implicitly distributing the other variables in the equivalence class). Some restrictions on the mappings of covering variables may be proposed at the next meeting.

Official subset: The group voted to allow HPF features to be excluded from the official HPF subset. The list of features excluded in this way includes the HPF standard library (previously most of the new intrinsics), the block FORALL statement (although the single-statement FORALL is included in the subset), the dynamic REALIGN and REDISTRIBUTE statements, and the ability to distribute sequential variables (i.e. covering variables as in the storage association proposal)

Parallel I/O: In view of Fortran's extensive formatting capabilities and the difficulty of reconciling parallel I/O operations with existing ordering semantics, the group voted not to include parallel I/O features in the version of HPF.

Many other matters were given first readings, including Marc Snir's LOCAL subroutine proposal and Barbara Chapman's TEMPLATE-less distribution paper.

Dates and agendas for future HPFF meetings are as follows:

October 21-23	LOCAL Subroutines, Distribution Inquiry Intrinsics, PURE Function Specifications, HPF Subset, Miscellaneous Odds and Ends
December 10-11	Additional Distribution and INDEPENDENT Assertions, Execution Model, 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 March meeting are marked with a plus sign (+).

`announce.*` - HPFF Announcement and Call For Participation

`archives` - Directory containing the archives of the HPFF mailing groups. (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

+ `benchmarks` - Directions to the HPF benchmark suite

`database` - The HPFF mailing list

+ `draft` - Directory containing draft specifications of High Performance Fortran (The current draft exists in TeX and Postscript formats; previous ones are in compressed Postscript only, for historical reference.)

+ `draft-v01.ps.Z`	*VERY VERY* Preliminary version 0.1
+ `draft-v02.*`	Version 0.2 - For HPFF meeting 9/10/92

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
  - FORALL-Loveman.tex — FORALL construct proposal (David Loveman)
  - FORALL-Wu — FORALL construct proposal (Min-You Wu)
  - Intrinsics — Intrinsics Proposal
  - Local-Subr.tex — Local Subroutine proposal
  - 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)
  - MIMD — MIMD support proposal (Clemens-August Thole)
- sept - Subdirectory containing all July meeting handouts
  - ALL.shar — Shar file containing all other files
  - Anti-Anti-IO — Argument in favor of I/O statements in HPF (Chapman & Mehrotra)
  - Anti-IO — Argument against including new I/O statements in HPF (Cownie)

Benchmarks.tex List of HPF benchmarks available from Syracuse (Haupt)

INDEPENDENT.tex Proposal for new independence assertions (Ramanujam, Sadayappan & Huang)

Intrinsics Proposed distribution inquiry functions (Shapiro)

No-TEMPLATE.tex Proposal for distribution directives without TEMPLATE (Chapman, Mehrotra & Zima)

ON-Clause Proposal for ON clause (Thole)

PURE-FORALL Proposal for FORALLs limited to calling PURE functions (Koelbel)

Side-Effect-FORALL Proposal for FORALLs with side effects (Koelbel)

Storage Final (!) version of storage & sequence association proposal (Zosel)

Subset Proposed features for inclusion in HPF official subset (Zosel)

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-may Summary of European Workshop on High Level Programming Models for Parallel Architectures

europe-june Summary of European Working Group on High Performance Fortran, 2nd meeting

june-minutes Summary of HPFF meeting, June 9-10, Dallas

BOF-minutes Summary of High Performance Fortran birds-of-a-feather session at ICS '92

july-minutes Summary of HPFF meeting, July 23-24, Washington

X3J3-minutes Notes from ANSI X3J3 committee meeting pertinent to HPF

sept-minutes Summary of HPFF meeting, Sept. 10-11, Dallas

papers - Directory containing papers related to HPFF, primarily from presentations at the first meeting. (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)

fujitsu.comments Comments on Fujitsu foils

fujitsu.over.tar "An Overview of VPP-Fortran Language Specifications" (11 foils in Postscript files)

fujitsu.prog.tar "Parallel Programming and Tuning in VPP-Fortran" (13 foils in Postscript files)

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

Alan Adamson (IBM Canada), Robert Babb (Oregon Graduate Institute), Ralph Brickner (Los Alamos National Lab), Barbara Chapman (University of Vienna), Alok Choudhary (Syracuse), James Cownie (Meiko), Siamak Hassanzadeh (Fujitsu America), Tom Haupt (Syracuse), Don Heller (Shell), Maureen Hoffert (Hewlett Packard), Ken Kennedy (Rice), Bob Knighten (Intel Supercomputer), Chuck Koelbel (Rice), Thomas Lahey (Lahey Computer), David Loveman (Digital Equipment Corporation), Aloke Majumdar (Yale), Piyush Mehrotra (ICASE), Andy Meltzer (Cray Research), Prakash Narayan (SunPro, Sun Microsystems), Tin-Fook Ngai (HP), Rex Page (Amoco), Jean-Laurent Philippe (Archipel), David Presberg (Cornell National Supercomputer Facility), J. Ramanujam (Louisiana State), Rony Sawdayi (Applied Parallel Research), Randy Scarborough (International Business Machines), Robert Schreiber (RIACS), Vince Schuster (Portland Group), Rich Shapiro (Thinking Machines), Henk Sips (Delft), Marc Snir (IBM), Matt Snyder (Lahey Computer), Guy Steele (Thinking Machines), Richard Swift (MasPar), Clemens Thole (GMD), Joel Williamson (Convex), Bernhard Woerner (Stuttgart University), Mary Zosel (Lawrence Livermore National Lab).

Memorable self-introductions

Sorry, I was running errands during the introductions this time.

Data Distribution

Guy Steele made the first presentation regarding distributed data. (See the "DISTRIBUTE Without TEMPLATE," "Distribution Intrinsics," and "PROCESSOR VIEWS" sections for other aspects of data distribution.)

Guy's first slide summarized a relaxation of the rules proposed at the last meeting.

The subcommittee recommends deleting the theory of similarity of templates.

Too many problems - We want to leave freedom to optimize in the future.

Instead, either use modules or rely on similarity of processor arrangements.

The theory had said that two TEMPLATEs of identical shape without explicit DISTRIBUTE declarations must be distributed identically. This rule no longer holds, although the theory of similar PROCESSORs is still active. Marc Snir requested that a formal definition of "identical" PROCESSORs declarations in this context appear in the

draft proposal, and Guy promised it would. David Loveman confirmed that TEMPLATEs are like any other name in a module for scoping purposes.

The next slide described changes for PROCESSOR declarations.

1. Scalar PROCESSOR arrangements are always allowed in HPF. (Previously, only arrangements that matched the result of NUMBER_OF_PROCESSORS() were guaranteed to work.)

```
PROCESSORS H
```

   Guy also added a discussion about host versus non-host architectures to the draft.

2. Allow sections of PROCESSOR arrangements as DISTRIBUTE targets:

```
DISTRIBUTE t(BLOCK) ONTO p(3,1:16:3)
```

   (The new syntax allows any Fortran 90 triplet notation, where the bounds and stride are specification expressions.)

3. A proposal to EQUIVALENCE PROCESSOR arrangements.
   Piyush Mehrotra presented this in detail the next day. See the "PROCESSOR Views" section for more details.

Justification for scalar PROCESSOR arrangements will appear in the next version of the draft. For now, Guy stated that it was a way to allow host programming and "If you want at least one processor, this is the way to do it." Andy Meltzer questioned how difficult the PROCESSOR sections would be to implement, and Piyush Mehrotra noted that those capabilities were needed anyway, since array sections passed to subroutines can end up distributed over a subset of the PROCESSOR arrangement. Tin-Fook Ngai wanted a way to name the section, and asked if this was related to EQUIVALENCE. The question was deferred until Piyush presented his write-up on PROCESSOR VIEWs. Ken Kennedy called for a vote to amend the omnibus distribution proposal to leave out processor sections (for now); 15 voted yes, and 6 no.

The next series of slides examined miscellaneous proposals made for the distribution features.

A. Should we eliminate ALIGN wraparound on TEMPLATEs?

```
REAL a(9)
TEMPLATE t(9)
ALIGN a(I) WITH t(i+5)
PROCESSORS p(2)
DISTRIBUTE t(CYCLIC(2)) ONTO p
```

   Using these declarations, processor 1 gets elements 5, 6, 9, 1,4 while processor 2 gets 7, 8, 2, 3. Other equally messy examples are also possible. The distribution subcommittee generally thought such pathologies should be eliminated.

B. Should we delete BLOCK(n) and just use CYCLIC(n)?
   This was suggested because the two distributions are identical except for an added constraint on CYCLIC. The subcommittee recommends no change, since the extra information helps compilers (avoiding divisions in some subscript translations, for example).

C. Resolve the non-significant-blank ambiguity by using the longest keyword. For example:

```
REALIGNABLER
```

   in fixed source form of hypothetical Fortran 99 means

```
REALIGNABLE R
```

   not

REAL IGNABLER

D. Prohibit sequential variables from appearing in ALIGN, REALIGNABLE, or REALIGN

The subcommittee eventually voted against this prohibition.

The discussion at this point was short. Regarding eliminating wraparound, Ken Kennedy noted that the fix to the example (and many other cases) was to use a big enough TEMPLATE. Henk Sips claimed another fix was to replace the ALIGN by CSHIFT; Chuck Koelbel disagreed, noting that this was not allowed in the current rules. A short discussion revealed that Henk was referring to CSHIFT operations within the code, and there were a number of advantages and disadvantages to writing code this way. Regarding the syntax ambiguities, Ken Kennedy suggested adding "Advice to programmer - don't use names starting with IGNABLE" to HPF programmer's manuals.

The next slide dealt with ALLOCATABLE arrays.

New rule: Static directives may be given for ALLOCATABLE arrays. These are then applied at every ALLOCATE statement. (All variable quantities in specification-expressions are evaluated on routine entry; array size information needed for the distribution (e.g. BLOCK uses the total array size) is plugged in at ALLOCATE time.)

ALLOCATE immediately followed by REALIGN has an obvious optimization, that is implement the alignment at the time of the ALLOCATE. (However, there is no official "theory of attachment" in the HPF language.)

ALLOCATABLE does not imply REALIGNABLE.

Chuck Koelbel asked several questions about how general the redistribution of ALLOCATEd arrays could be (in particular, whether block sizes could be picked dynamically); the new system allows everything the systems discussed at the last meeting did.

Guy's penultimate slide dealt with Fortran 90 pointers interacting with mapping directives.

To avoid aliasing problems, we propose two rules:

1. A pointer may be used in place of an array in REALIGN and REDISTRIBUTE if and only if it is associated with a whole array.
2. If an array is associated with a pointer, it may not be remapped in any way except via that pointer, per rule 1

Consequence: If two pointers are associated with the same array, then you can't REALIGN through either of them (because each disallows mapping through the other under rule 2).

Clemens Thole questioned whether these rules really eliminated the aliasing problem. In particular, he noted that they solved the problem for multiple pointers, but aliasing could still occur under an original name. Rob Schreiber interpreted this as saying that a pointer allows remapping out from under original name, giving the example

```
DIMENSION(:), POINTER :: p
DIMENSION(10), TARGET :: a
REALIGNABLE a
...
p => a
REALIGN p WITH ....

! What is the alignment/distribution of a at the end?
```

(There were some quibbles over whether the pointer needed to be declared REALIGNABLE, but the example got the point across.) Guy Steele clarified that the

purpose of rule 2 was to avoid aliasing between pointers, and this (very similar) problem had not been considered. He suggested adding a rule to disallow this REALIGN as well. Clemens Thole suggested revising the rule so that REALIGN through a pointer is only allowed when that pointer is the only thing associated with the array. (He noted this also eliminates the need for rule 1.) Joel Williamson claimed there was a possibility for astonishment with this rule if there were dangling (non-live) references to an array; under Clemens' proposal, any REALIGN would be illegal. The pointer proposal was eventually sent back to committee for further study; Guy presented a new version the next day.

Guy's last slide was titled "For the Journal of Development" (referring to an idea resurrected from the Fortran 8X development process). These represented features that the committee saw an eventual need for, but which needed much study and/or implementation experience before even attempting to put them in the language.

A. How to ALIGN pointers themselves (as scalars)?
   i. Bury in a derived type
   ii. Use => in ALIGN [HPF2]

```
ALIGN p => h
PROCESSORS h
```

B. Dynamic template computation
   i. Call a subroutine
   ii. ALLOCATABLE templates [Something for HPF2]
C. Remembering the "old" template of a global array to restore later.
   POINTER to template

The group was in general agreement that the Journal of Development was where these belonged.

Ken Kennedy then started a general discussion with the question "Are any of these proposals controversial?" (This was widely acknowledged as a rhetorical question.) Comments from this discussion specific to one of the slides appear in the correct context above; more general issues appear below.

Clemens Thole asked what any vote taken here would mean if TEMPLATEs were removed from HPF, as Barbara Chapman was scheduled to propose the next day. Ken Kennedy said decisions here should be treated as a conditional vote, assuming TEMPLATE would be in the language. Several others agreed this conformed with standard parliamentary procedure.

In answer to exactly what was being voted on, Guy Steele noted that the basic distribution framework had already been accepted. This vote would be to accept the amendments proposed in his presentation. Also, this would confirm the lack of other changes to the subroutine interface proposal from last time. Marc Snir was unconvinced that all possible distribution cases were covered in the proposal. Clemens Thole and Guy Steele asked for specific examples, and Marc could not produce any. He eventually promised to send any that he found later to Guy; anybody else finding a proposal bug was encouraged to do so as well. Rob Schreiber and others claimed that the mappings allowed in HPF formed a closed set under linear ALIGN functions, which was all that were defined.

The group took a preliminary vote to accept all amendments to the data distribution proposal as recommended by the subcommittee. The result was 31 in favor, 0 opposed, and 0 abstain. The total vote count was more than expected, so a fast count of distinct organizations was made; indeed, 31 companies, universities, and national labs were present. A final vote was then taken to accept the data distribution chapter of the draft as a whole (subject to correcting ambiguities and incorporating the discussed amendments). This vote was 25 yes, 1 no, and 4 abstain.

The group then took a coffee break.

Intrinsics and Library Functions

After another short discussion of scheduling for the rest of the day, Rob Schreiber began the intrinsics subgroup proposal. Before beginning, he noted, "My information from the last meeting comes entirely from the notes which were fantastically useful."

Rob's first slide discussed reasons for making the proposed new functions intrinsics rather than a standard library.

- Some are elemental (e.g. ILEN) - Fortran 90 would not allow them to be used elementally as library functions.
- Explicit interfaces would be required for most of them unless they were intrinsic.
- Cumbersome to use the required Fortran 90 interfaces (generic types and shapes are needed)
- Making them intrinsic allows the use of integer-valued functions in specification expressions (e.g. in array declarations)

Based on these arguments, the subcommittee recommended adoption of all the functions as intrinsics. Marc Snir noted that the upcoming FORALL proposal included user-defined elemental functions, which argued against the first and third points. Clemens Thole asked about using Fortran 90 generics, and Rich Shapiro made clear that this would lead to a huge library (without sleazy machine-dependent tricks to handle the linking). Mary Zosel noted that using generic also puts more requirements into the HPF subset. Vince Schuster argued that a consequence of defining intrinsics would be that the compiler writer must handle them directly - this is not equivalent to providing libraries. In answer to some library interface questions, Jim Cownie asked "Is it really so hard to say USE HPFF?" Ken Kennedy moved the meeting along by stating that the amount of discussion implied the group had to take a separate vote on this issue.

Rob's next slide concerned changes to the combining send routines (previously known as XXX_SEND).

```
XXX_SCATTER(SOURCE,DEST,IDX1,..., MASK=mask)
```

- Name change: XXX_SCATTER - This avoids name clashes with message passing routines, and is a better description of the operation anyway.
- Revise the document to remove any guarantee on order of operations - The previous draft (unintentionally) disallowed reorganizing operations using associativity, effectively eliminating tree sums and other useful implementations.
- Adding MASK argument, conformable with SOURCE, to allow conditionals in the operation - This enhances compatibility with reduction intrinsics.

Robbie Babb asked what the numerical folks would say about the defined lack of order. Rob pointed out that the new draft does not even guarantee that the order is necessarily repeatable. Unsupported opinion was that it was fair game to avoid such guarantees in defining routines; if users needed a specific order, they would have to code the operations sequentially.

Rob then presented recommendations from the subgroup regarding the HPF subset. The only new functions to be included in the subset were

- NUMBER_OF_PROCESSORS()
- PROCESSORS_SHAPE
- The improved versions of MINLOC and MAXLOC
- ILEN
- DIM arguments (to new and existing intrinsics) must be constants rather than arbitrary integer functions

The last proposal was based on the observation that allowing nonconstant DIM arguments was very difficult to handle by compiler (in fact, many existing commercial compilers do not handle the feature). Mary Zosel and Ken Kennedy thanked Rob on behalf of the subset group for these recommendations. Tom Lahey asked if the new MINLOC argument was inconsistent with Fortran 90; Rich Shapiro assured him that adding optional arguments doesn't conflict.

No other changes to the previous proposal had been made (or suggested). Ken Kennedy brought the group back to the intrinsics versus library question. Clemens Thole noted that a USE requirement would also standardize the library name. Maureen Hoffert suggested only accepting the recommended subset as intrinsics, and Rex Page proposed this as a vote. David Loveman noted that adding lots of intrinsics would be a big change to Fortran 90, which is not what HPFF claims to be doing. Guy Steele noted that elemental intrinsics of type integer can be used in specification expressions, which would argue slightly in favor of including all the bit manipulation functions as intrinsics as well. This was taken as a friendly amendment. Piyush Mehrotra asked who was going to use POPCNT as an array size, to which Rob Schreiber gave the example of using 2**ILEN(N-1) to round up to a power of 2. Jim Cownie asked to see the whole list of functions before voting; Rob Schreiber responded jokingly, "I won't show you the whole list of 5000." He then listed the list of bit inquiry operations: POPCNT, POPPAR, LEADZ, and ILEN.

About this time people started getting confused about what was being proposed. Ken Kennedy interpreted parliamentary procedure: the top-level vote was the recommendation (by the intrinsics group) that the entire list of new functions be made intrinsics. It had been moved and seconded to amend this to make only NUMBER_OF_PROCESSORS, PROCESSORS_SHAPE, the bit inquiry functions, and the modified MAXLOC and MINLOC intrinsics. New functions that did not become intrinsics would be part of a standard library in the amended version. Several voices were then raised to amend the amendment so that only 3 new intrinsics were added: NUMBER_OF_PROCESSORS, PROCESSORS_SHAPE, and ILEN (MAXLOC and MINLOC already being intrinsics). Ken called for a vote on the last amendment, but was not fast enough to avoid yet another attempted amendment. Fortunately, Guy Steele pointed out that an amendment to an amendment to an amendment was not allowed by Robert's rules of order, "precisely to avoid this sort of recursive thrashing."

After a minimal amount of further discussion, Guy presented a clear phrasing of the questions:

> There is an motion on the floor to adopt a small number of functions as intrinsics, leaving the rest as library functions.
>
> The first question we will vote on is, "Should the short list be 3 functions (NUMBER_OF_PROCESSORS, PROCESSORS_SHAPE, and ILEN) or 6 functions (the above plus POPCNT, POPPAR, and LEADZ)?"
>
> The next question we will vote on is "Should the short list (whichever one we picked) be the only intrinsics, or should all the new functions be intrinsics?"
>
> Finally, we will vote on whether to accept the whole package as part of HPF.

Everybody understood this, and the voting began. On the question of whether the "short list" would be 3 or 6, 17 voted in favor of 3 functions, 10 in favor of 6 functions, and 4 abstained. Given that, the vote was called on whether NUMBER_OF_PROCESSORS, PROCESSORS_SHAPE, and ILEN should be proposed as the only new intrinsics; this vote was 19 in favor, 7 opposed, and 5 abstain.

Before voting on accepting the recommendation as a whole, Randy Scarborough offered a new amendment: accept only the 3 new intrinsics, not the other functions. Rich Shapiro immediately complained that no standard interface for the proposed functions would drive him up a wall. Randy replied, "Good!" Ken Kennedy asked that the record

show that it was not clear if that exchange was in favor of or against the amendment. After the laughter died down, Randy withdrew his amendment, since somebody at his end of the table had pointed out that subsetting could get the effect he was seeking.

The vote to accept the intrinsics proposal as amended (now better called the intrinsics and library proposal) was then taken; 26 voted yes, 0 voted no, and 5 abstained.

Andy Meltzer now called for a vote on what from the intrinsics proposal would go into the official HPF subset. Rob Schreiber (after several friendly amendments) arrived at a proposed subset:

3 new intrinsics, and modified MINLOC/MAXLOC intrinsics
DIM restriction on all intrinsics
None of the standard library

After some discussion of procedures, the proposal was made as a straw vote, since it was out of order (the group had not voted on whether to allow HPF features to be excluded from the subset yet). Maureen Hoffert suggested putting the functions in the Journal of Development, rather than creating layers of subsets. Ken Kennedy argued against this, since the intent is to standardize spelling on these functions, while the JoD is speculation for future and has no official status. After some more discussion, the straw poll was stated as "If new HPF features can be excluded from the official HPF subset, then all of the library as now defined should be out of the subset." The straw poll was taken: 27 yes, 2 no, 7 abstain.

Official HPF Subset

In the short time before lunch, Mary Zosel presented the latest HPF subset proposal. From votes at past meetings, several features were already in the subset (notably the array sublanguage).The new proposal included several other features for consideration. Copies of a handout with the proposed features quickly made their way around the room.

Taking the proposed features out of their order in the handout, the first question was "Should Fortran 90 modules be added to the subset? If so, should they be just vanilla modules or full renaming capabilities, etc.?" Piyush Mehrotra asked compiler vendors to discuss the difficulty of implementation. Joel Williamson replied that renaming was difficult in any case, and Vince Schuster noted that all new features are somewhat difficult. Clemens Thole asked about PRIVATE declarations, and Mary noted that the proposal only included the specification aspect of modules. She then reminded the group of the 2 reasons for modules: global names and/or availability for TEMPLATEs, and implementing the (non-subset) library. David Loveman framed the question as one that HPFF had seen before: more functionality, or faster implementation (indeed, he claimed that the whole subset question revolved around this tradeoff). Andy Meltzer noted that modules were very likely to delay implementation. Guy Steele stated that HPF without modules is still very interesting, and Loveman agreed strongly. The straw poll was phrased as whether to have a limited form of modules in the subset; 7 were in favor, 22 were against, and 7 abstained.

The next sub-proposal was to limit INTERFACE blocks (which had previously been accepted into the subset) to avoid the generic-spec or module-procedure-stmt options. David Loveman noted this was natural since the whole library was not in the subset. The group agreed this was a moot point since generics themselves were not included in the subset.

A few intrinsics were also proposed for inclusion. FORTRAN 77 intrinsics had been accepted, but the Fortran 90 additions had not (yet). RANDOM_SEED and RANDOM_NUMBER were accepted by a straw poll of 26 in favor, 0 against, and 8 abstaining. DATE_AND_TIME and SYSTEM_CLOCK were accepted by an identical margin, despite a tongue-in-cheek proposal by Marc Snir and Joel Williamson to forbid users to time their programs by any means. A short discussion ensued about what guarantees HPF could or should make on the quality and repeatability of

RANDOM_NUMBER outputs. Consensus quickly developed that this was a Fortran 90 issue, not and HPF decision. David Presberg noted that similar repeatability issues appear elsewhere (for example, in mapping inquiry functions); he suggested listing things that can't be used for validation. Ken Kennedy urged the group not to get involved in that morass.

Next, adding the new Fortran 90 declaration form (that is, the syntax with double colons) was considered. Chuck Koelbel asked if nonstandard attributes were allowed as Fortran 90 extensions, and was assured that they were. Joel Williamson made the mild statement that compilers would get out the door faster without such declarations, since they were yet another small thing to do. The straw poll for including the new declarations found 26 in favor, 0 opposed, and 9 abstaining.

Finally, some issues from earlier that morning were considered: "Should the DIM argument in intrinsics be limited to constant expressions?" 8 were in favor, 7 opposed, and 21 abstained. In an effort to get clearer guidance, an amendment was offered limiting DIM arguments to cases where the resulting shape would be known. The straw poll showed 6 in favor, 5 opposed, and 25 abstaining. Jokes about voting for various third-party candidates were tossed around. Ken Kennedy suggested that the subset group come back with a crisply-worded proposal at the next meeting.

The group then adjourned for lunch.

FORALL Subgroup

Most of the notes in this section were taken by Mary Zosel, since I was busy presenting.

After lunch, Chuck Koelbel presented the FORALL and INDEPENDENT assertion proposals.

Chuck started with a presentation of the single statement FORALL. (Still based on the old Fortran 8X syntax - adopted by Thinking Machines, MasPar, and other clients of COMPASS.) The major new addition was that pointer assignments were now allowed (where the pointer is a component of a derived type, stored in an array). Guy Steele had proposed allowing ALLOCATE inside a FORALL as well, but the subgroup had initially decided no, in part because it looked so different from assignment and in part because the interactions with ALLOCATE of distributed arrays were not clear. The single-statement FORALL interpretation had not changed: generate indices, evaluate masks, evaluate the right-hand side for the active indices, and assign to the left sides. A little new explanation had been added to explain what evaluating the right-hand side of a pointer assignment meant. The block FORALL body can be assignment (including array and pointer assignment), WHERE, or nested FORALL. Constraints on an inner FORALL say it cannot redefine indices in the outer FORALL.

Function calls inside FORALLs had changed. The subgroup had prepared two proposals. They recommended the proposal that there be no side effects in any function call inside a forall (by a narrow margin - the vote in subcommittee was 7-6). The other proposal was that side effects are allowed if they don't interfere with each other. The exact types of side effects allowed would be identical to those explained at the last meeting. The side effect proposal would require an interface notifying the compiler of the potential.

Mary Zosel questioned the status of RANDOM_NUMBER calls within FORALL, and got no coherent answer as the potential problems had not come up in the subgroup.

Both proposals define a new HPF feature.

```
!HPF$ PURE
```

identifies functions with no side effects - this is a new attribute of the function. Intrinsics are always PURE (another sign that RANDOM_NUMBER was not considered). A PURE function has

- Only INTENT (IN) parameters
- No SAVE locals
- No assignments to globals
- No I/O statements
- Only calls other PURE procedures
- access to distributed data is allowed for global data, not for locals or dummy argument

To clarify the last statement, a PURE procedure can access data that is already distributed, but can't change or create distributions. You can put the pure assertion in an interface before the function declaration statement. If a dummy argument is declared as PURE function, then its matching actual must also be PURE. These are the kind of functions where you can cache the result and recognize the arguments and reuse (if no global data has changed). In responses to numerous questions, the draft will be modified to allow no PAUSE, STOP, DATA statements.

Several nits were raised about the position of a PURE directive - before or after the function name. Consensus emerged that it should follow the function name, and this will be fixed in the next draft.

The second (non-pure) proposal essentially said that side effects can't interfere with other combinations of FORALL index values, or with the opposite side of the assignment. To the proposal made at the last meeting, the subgroup had added the idea of an IMPURE directive (where anything goes). Functions called from FORALL had to be declared explicitly PURE or IMPURE. This was added because the group felt the Fortran 90 conforming default (impure functions) was not the one expected in a FORALL, and argued that the programmer should be made aware of what he or she was doing.

There was some shooting at the terminology PURE - for example, from one FORALL to another, the value of a global variable read by the function might be changed, unlike a mathematically pure function. Those armed with Microsoft Word in their PowerBooks (one or both of those are trademarks, I'm sure) suggested various synonyms for the idea, including stainless, immaculate, sanitary, spotless, unadulterated, unalloyed, uncontaminated, faultless, genuine, perfect, simple, unmodified, purebred, thoroughbred, complete, sheer, undiluted, unqualified, unmitigated, utter, celibate, innocent, virginal, modest, undefiled, unsullied, virtuous, guileless, sinless, honest, and guiltless. Serious suggestions were referred to the FORALL mailing list.

Despite all the arguing, it became clear that everyone wanted *SOME* functions in a FORALL.

Ken Kennedy took a formal vote for PURE functions, subject to a revised definition. The results were 16 in favor, 12 opposed, and 2 abstaining. The exact definition of pure including the name will be defined more carefully at the next meeting.

The presentation then turned to the INDEPENDENT assertion. In compiler terms, this assertion states that there are no loop carried dependences. The syntax is

```
!HPF$ INDEPENDENT [(I,J,K)]
```

where I,J,K are the indices of to tightly nested DO loops or FORALL constructs. The interpretation is a user assertion that no iteration or combination of index values assigns to a location (in Fortran 90 terminology, any scalar data object) accessed by another iteration or combination of values. This does not change the semantics of the construct if the assertion is true. The program is not standard conforming if the assertion is false. The compiler can take any action it deems necessary (somebody suggested executing the programmer, but that seemed harsh). The INDEPENDENT directive refers to the entire construct.

There was a clarification for Rich Shapiro - this is not exactly the same as the IVDEP declaratives. This declares that the iterations can be done in any order, including concurrently. Thus, privatizing variables is not allowed.

A long discussion ensued about reduction assignments in INDEPENDENT loops. (Unfortunately, all the names have been lost, as well as most of the statements.) By the end, it was clear to all that reductions were not allowed in this form of INDEPENDENT, but were vital for many very interesting applications. Geoffrey Fox and his colleagues at Syracuse were working on a proposal to add this as either a separate assertion or as a modification to the current INDEPENDENT assertion. Similarly, an proposal to add PCF NEW declarations (formerly called PRIVATE variables) was also making the rounds. Chuck promised that these would be presented at the next meeting; as he said, "It's hard to do that now, since the proposals don't exist yet."

A straw poll was taken about having some form of reduction inside of an INDEPENDENT loop? The count was 18 in favor, 7 opposed, and an uncounted number of abstains. The matter was referred back to the subgroup for further study. A random question about how a user would be sure that a loop was INDEPENDENT led to jokes about a Guy Steele icon in the programming environment.

At Mary Zosel's request, Chuck and Guy reproduced Guy's earlier picture clarifying what INDEPENDENT means on a FORALL. The picture is still too difficult to reproduce with ASCII graphics, but Chuck promised to include it in the next LaTeX version of the HPF language draft. In words, the construct

```
FORALL ( i = 1 : 3 )
    lhs1(i) = rhs1(i)
    lhs2(i) = rhs2(i)
END FORALL
```

can be visualized as a begin operation, an end operation, and four rows of three operations.

The first row represents computing rhs1(i) for each value of i.
The second represents the assignments to lhs1(i).
The third row is the computations of rhs2(i).
The last row is assignments to lhs2(i).

Each operation becomes one node in a directed graph. The edges in the graph represent orderings imposed by the FORALL semantics. Considering the edges out from each node, we have

The begin operation has edges to all of the rhs1(i) nodes.
Each rhs1(i) node has edges to all of the lhs1(i) nodes.
Each lhs1(i) node has edges to all of the rhs2(i) nodes.
Each rhs2(i) node has edges to all of the lhs2(i) nodes.
Each lhs2(i) node has an edge to the end node.

So there is a dense connection between each pair of adjacent levels. The corresponding diagram for a DO loop is a long snake going down each column and then connecting to the next column. The INDEPENDENT assertion asserts that the only edges that a compiler must respect are

The edges from the begin operation to the rhs1(i) nodes.
One edge from each rhs1(i) node to the lhs1(i) node directly below it (i.e. the same value of i).
One edge from each lhs1(i) node to the rhs2(i) node directly below it.
One edge from each rhs2(i) node to the lhs2(i) node directly below it.
The edges from each lhs2(i) node to the end node.

Now the graph is just three chains, connected only at their beginning and end points. The corresponding diagram for a DO loop with INDEPENDENT is identical.

Ken Kennedy now started calling for votes on the FORALL proposal. There was a move and second to drop the block FORALL, with the resulting vote being 9 in favor, 14 opposed, and 6 abstains. Next came a move and second to drop the nested FORALL. Rich Shapiro was worried about its implementation; Guy Steele pointed out that there is a

published paper saying how to do it. The formal vote to drop nested forall found 7 in favor, 15 against, and 9 abstaining.

David Presberg started a discussion of WHERE, suspecting a technical flaw; he dropped this suspicion in the following discussion. There was also a question about what INDEPENDENT means with WHERE inside FORALL, and agreement that the problems were orthogonal. Guy Steele gave an example

```
FORALL (I= ..., J=... )
    a(i,j) = 3
    B(i,j) = i*j
WHERE (B(I,J).EQ.i) ....
!    but ARGHHH! can't use where since test is scalar!.
    B(I,J)=43
END WHERE
END FORALL
```

This argued for better conditional control within FORALL. Some discussion followed regarding whether we should extend the F90 definition of WHERE or have some new functionality? Chuck pointed out that these issues (particularly extending WHERE) had been argued on the mailing list before. Piyush Mehrotra pointed out that the same functionality can come from separate FORALL constructs. Joel Williamson asked about how a FORALL behaves inside a WHERE, and was told that the syntax did not allow that. He then asked if there any problems with WHERE inside a FORALL with a mask possibly similar to whyever Fortran 90 threw out nested WHERE. None were known, but the subgroup will stay on the lookout for them. A straw poll was finally taken on extending the conditional capabilities of FORALL constructs, finding 16 in favor, 15 opposed, and 6 abstaining. The committee was requested to continue considering these issues.

In lieu of another proposal, a formal vote was taken to amend the proposal to drop WHERE of any kind inside a FORALL. The count was 4 in favor, 20 against, and 7 abstaining.

Having run out of individual features to remove from the FORALL proposal, a formal vote was taken on the proposal as a whole. The count was 26 in favor, 1 opposed, and 4 abstaining.

Clarifying the vote on the INDEPENDENT assertions, the proposed BEGIN INDEPENDENT - END INDEPENDENT pair was not included. The FORALL committee thought that maybe there was a need for more general INDEPENDENT-like assertions, but decided that the BEGIN/END version was not what they wanted. Another version might be added to a later draft. The group decided to vote on INDEPENDENT applied to DO and FORALL separately, as there seemed to be a difference of opinion. The vote on DO INDEPENDENT (with a possible extension to allow reduction yet to come) found 26 in favor, 1 opposed, 3 abstaining, and 1 self-described "confused". The vote for accepting FORALL INDEPENDENT netted 18 in favor, 5 opposed, and 8 abstaining.

Local Subroutines

After a short break, Marc Snir presented the local subroutines proposal. (Amusingly, the chapter title in the draft was "Foreign Procedures"; Marc preferred the term "Local".)

The proposal was divided into 3 parts.

- An HPF interface to foreign routines (part of HPF language)
- A proposal for a Fortran 90 implementation of local routines (not necessarily part of every HPF implementation)
- Fortran 90 intrinsics to access information regarding HPF array mappings (status as part of HPF to be determined)

There was also a miscellaneous part at the end of the presentation.

Marc next presented the HPF interface, which itself consisted of three parts. First came the requirements on the HPF caller:

- An INTERFACE block for each local routine (with an explicit LOCAL declaration)

```
INTERFACE LOCAL function foo(x)
```

- Scalar arguments are replicated (one copy per processor, all with consistent values)
- The LOCAL function returns one value per processor (declared as an array of size NUMBER_OF_PROCESSORS() with no HPF-defined distribution)
- INTERFACE block may have ALIGN or DISTRIBUTE directives

Piyush Mehrotra checked whether LOCAL was considered a keyword, which it was. Rob Schreiber asked whether it was more natural to return an array of shape PROCESSORS_SHAPE(), but Jim Cownie claimed that would be unportable due to different implementations. Chuck Koelbel inadvertently started a long discussion about defining the mapping of a function return value. The net result of that digression was that HPF can't map the result array of function (now); extensions should do that later. Clemens Thole asked why not use the normal Fortran 90 interface for these functions, and was reminded that LOCAL functions might not be FORTRAN at all.

The next matter was the calling sequence (generated by the HPF compiler). The operations that must be performed are

- Remapping of arguments (before call and after return)
- Updating of replicated variables (before the call)
- Synchronization (before and after the call)
- Transfer of control (on each node)

Closely related to this is the callee contract, which specifies what the LOCAL routine must guarantee in order to be correctly called from HPFF.

- Obey the INTENT IN/OUT restrictions.
- Update replicated data consistently.
- Do not change array mappings.

The intuition is that all processors make their internal states consistent with the HPF semantics, synchronize, and perform the call. Another synchronization is performed at return to guard against processors going off into never-never land. Marc Snir pointed out that his proposal requires the compiler to do remapping before calling routine, not rely on it happening there. It was also suggested that compilers not be allowed to change the distribution at LOCAL calls as they are allowed to do at other points in the program. Guy Steele asked for the wording on what synchronization means and was referred to page 73 of the draft. Informally, the condition is that all actions that logically precede call must be executed before any processor enters the routine.

The callee must satisfy the contract conditions for correctness. Note, however, that the conditions need only be true on return, not necessarily during routine execution. Robbie Babb asked if replication was consistent with the no aliasing conditions of Fortran. David Presberg noted that this replication had a different meaning than on a sequential system, and Guy Steele noted that an implementation may perform checks of replicated consistency, but this proposal doesn't require them. There was some confusion about why scalars are automatically replicated; Guy explained that routines only replicate scalars automatically; the user may replicate arrays explicitly if desired.

Marc then moved on to the information available to a LOCAL routine.

- The local piece of a distributed array passed as an actual argument. (This appears as a contiguous array of the same rank as the HPF array, accessed with local indices.)

- HPF COMMON areas may be accessible. (Details of this are still to be worked out.)
- Full information of the mapping of actual arguments (available via intrinsics in the Fortran 90 local proposal)
- LOCAL can call other internal (LOCAL?) routines. (A precise description of this is needed; it's coming.)

Marc Snir reiterated that users are required to do their own cache consistency; if two locations (presumably on different processors) represent the same array element, then the LOCAL routine is responsible for making them consistent by whatever means are appropriate. Piyush Mehrotra asked how a user would know if something was replicated; Ken Kennedy suggested waiting for the intrinsics presented on a future slide. Robbie Babb asked if Marc was requiring a contiguous storage scheme. Marc explained that the array area is contiguous, but may not be completely used. (For example, some distributions might have a sparse storage scheme; the local piece would be the entire arena for allocating the array elements.) Joel Williamson asked if a routine could be called on a subset of the processors, and got the answer "No." In general, an implementation can't allow asynchrony between HPF and LOCAL routines because it would invalidate the analysis in the HPF parts. David Loveman noted that this meant a LOCAL routine may execute on processors without data, implying a need to check if there is anything to do. Marc affirmed there are inquiries to check this. Joel Williamson sounded a theme that would recur: "Do LOCAL routines calling other LOCALs have to synchronize to maintain replicated consistency?" Guy Steele responded "No"; the calling conventions only guarantee consistent values when returning to HPF.

Leaving the HPF language proper, Marc described the Fortran 90 local intrinsics (or libraries) needed to provide the information described above.

- See "proposal for distribution inquiry intrinsics"
- Plus derived functions
  query_function(array_name, ...)
  Only for HPF arrays (arguments, common?)
- Local-to-global and global-to-local index translations

In general, functions were intended for inquiring about all aspects of an HPF data mapping, and for accessing the section(s) of data stored on the local processor. This led to more discussion about the consistency of replicated values. Joel claimed that the only way to ensure this consistency was to implement synchronous updates in the LOCAL routines, while several others claimed that low-level routines could update asynchronously and combine the results (explicitly) before returning.

Marc then proposed local pointers and global pointers for passing pointers between HPF and LOCAL routines.

```
INTEGER, POINTER :: p, q, r
!HPF$ LOCAL POINTER p, q, r
```

LOCAL code may not dereference global pointers (the default)
  (Perhaps another function is needed?)
HPF code may dereference either pointer class and assign one to another
  However, local = global is only valid when dereferencing the
    global can be done locally.

The idea is to tell the compiler how things are mapped by whether the pointer is local or global. Piyush Mehrotra claimed this needs the distribution group's proposal to tell how well it works. Guy Steele reinterpreted Marc's statement to mean the intent is to let the compiler know that pointer dereference can be done without communication. Rex Page asked what local pointers do for local routines. Marc replied they tell the compiler this is safe. This helpful information is only for HPF-related local routines, of course.

Clemens Thole gave an example of a LOCAL matrix multiplication routine that he would like to write, but claimed it was impossible under Marc's proposal. He needed to

return array patches from each processor rather than a single value stored in an array (and, of course, he needed to control the order those patches were stored). Marc suggested looking at it off-line. Rich Shapiro claimed that the current proposal was now over-defining the interface at other end by specifying the contents of a LOCAL routine. His counterexample was "What if I call assembly language from HPF? I can't use any of this." A long discussion followed. Marc eventually clarified the digression by noting there were 2 layers being discussed: the caller/callee contract (for HPF and any other language), and a Fortran 90 LOCAL interface (specific to that language). Richard Swift noted that if HPF COMMON works in LOCAL routines, it avoids the parameter stuff complicating the proposal. Marc agreed, but said he was designing this to work for scientific subroutine libraries, and thus need parameters. Rex Page asked if the caller would always know the size and shape of the local return. Marc answered that that was the intent, and Clemens noted this information was required in Fortran 90. Clemens then noted that a reasonable target would be shared memory with local and global address generation modes, and proposed calls for describing sections of an array to work on.

Ken Kennedy began wrapping up the discussion (or at least focusing it more) by asking if there were any issues on the HPF caller side. The position of the LOCAL keyword emerged, as it had for PURE functions. Some were still uncomfortable with the replication of scalars and the fact that a function returned only one value per processor. There was a question whether the INTERFACE block may have ALIGN and DISTRIBUTE declarations. Rich Shapiro proposed that a LOCAL function return "whatever is necessary" to fulfill its explicit interface, thus making thus shape explicit. Marc agreed to make this change. Richard Swift questioned the need for specifying the mapping directive in an interface must be honored before the call. Marc explained that this allows writing a routine for a canonical mapping, avoiding the programmer doing the redistribution manually. Richard replied that this is the only instance where such a thing is required, to which Marc replied that it is also the only place where it's semantically meaningful. A straw poll to accept the caller interface as amended passed with 32 in favor, 1 against, and 4 abstaining.

Next to be wrapped up were the called function issues. Rich Shapiro suggested the proposal just needed to specify what information was available. Marc Snir described this as a 2 part question: identifying what must be available (generic), and the actual mechanism (Fortran 90 specific). Clemens Thole suggested that HPF shouldn't specify much at all; Guy Steele amended this to not specifying much except when the callee is written in HPF (or a near relative). Ken Kennedy illustrated the need for local iteration spaces with an example from molecular dynamics. A common algorithm there is to perform a Jacobi iteration between the blocks on each processor, but process each individual block with a Gauss-Seidel sweep which converges faster. He claimed we want to allow these algorithms in HPF. Finally, a straw poll was taken to give advice to Marc. A proposal to define a sublanguage of HPF for use in LOCAL routines and indicate it as a model for other interfaces drew 30 yes votes, 1 no note, and 4 abstentions.

Test Suite

The last presentation of the day was a short description of the HPF test suite being built at Syracuse. Tom Haupt gave the talk.

The suite contains over 30 benchmarks and is available by anonymous FTP from minerva.npac.syr.edu. Unfortunately, the set is currently in very bad shape due to past disorganization. Tom said that a new, better organized release was in the works. He hoped to have the new test suite ready within 1 week (so, in theory, it should be ready by the time you read this).

Tom then put up a chart showing the contents of the test suite on the overhead projector. Unfortunately, the original was in 12-point type so nobody could read it. Joel Williamson subtly pointed this out by simulating reading an eye chart from the back of

the room. Chuck had copies made for the group the next morning; it is also available in the HPFF FTP directory.

The test suite consists of several general groups of applications, most tied together by an applications area. In brief, they can be grouped as

- Validation of Fortran 90 compiler - 5 simple examples
- General applications: Gaussian elimination, integration to compute pi, a Laplace solver, 2D FFT, simplex method
- Purdue set - 15 kernels
- LAPACK subset
- Physics codes
- Financial model - options pricing by Monte Carlo techniques
- Weather - the CAPS program
- Electromagnetic field - EM scattering
- NAS parallel benchmark set
- Stanford parallel applications (SPLASH) suite
- Genesis benchmark
- Molecular dynamics
- Image understanding benchmark

All of the benchmarks would eventually exist in several versions: a sequential version, Fortran 77D and Fortran 90D, CM Fortran, and message-passing dialects. The chart showed a large number of these completed, and estimated completion times for the others ranging from August 1992 through early 1993. Several people noted that August 1992 had already come and gone; most of these programs will be in the new release next week.

Ken Kennedy asked what he thought was an easy question: how hard would it be to convert the Fortran 90D versions to HPF? Tom could only say, "Good question..." Alok Choudhary came to his aid, saying someone would just have to change directives slightly for syntactic reasons. Fortran 90D is "really very close" to full HPF, except for some details of subroutine linkage and other advanced features. David Presberg was confused by the aims of the test suite, so Ken explained the funding situation. Basically, the test suite is part of a grant for the development of Fortran D, and therefore the priorities for the suite are to target that language. Clemens Thole reported that one of his colleagues had used the test suite on his compiler (basically an array-syntax-to-message-passing transformer), and had gotten better performance from the compiled code than the message-passing implementations included in the suite. This was explained by the fact that the message-passing versions were still preliminary and had not been fully tuned yet.

The group then broke for dinner at a Mexican restaurant. In contrast to previous meetings, no evening session was planned; instead, some of the subgroups took the opportunity to meet, and more than one proposal was revised on a laptop computer for the next morning.

Distribution Without TEMPLATE

Friday's first presentation was Barbara Chapman's discussion of "HPF Distributions and Alignments without TEMPLATES." As a preface to the talk, she described the history of the proposal. Some (rather vocal) mailing list subscribers had reservations about use of TEMPLATE in HPF; this was a proposal to fix some of those problems. She was not trying to hide the limitations and other changes of this alternate approach, but to explore its possibilities.

Barbara's first slide was titled "What's Wrong with TEMPLATEs?"

- They are not first class objects and can't be passed to subroutines.
- ...and if they could be passed, what happens to the data objects mapped to them?
- They can't be used with ALLOCATABLE arrays (since TEMPLATEs are static)

She admitted that the last point was somewhat of an overstatement, given the ALLOCATE decisions made the day before; it was meant to refer to explicit TEMPLATEs. Barbara also gave a disclaimer about her general unfamiliarity with the HPF model, since her group had been concentrating on Vienna Fortran.

The next slide introduced the proposed alternative model.

- Processors - as in the current HPF model
- Distribution - map data objects to processors
- Alignment - indirect specification of distribution
- A picture: (many arrays) mapped by ALIGN to (one array) mapped by DISTRIBUTE to (abstract processors) mapped by machine-dependent directives to (physical processors)

The picture was very similar to the one given in the HPF data distribution proposals, reflecting the fact that this proposal has the same intuition as Guy Steele's model.

Barbara referred to the formal language in her proposal to give a complete definition of alignment. This was subtly different from the current HPF proposal, in which an array was always aligned directly to a TEMPLATE (semantically, although not necessarily syntactically). The effect is a one-level structure, with many arrays pointing at one TEMPLATE. In the no-TEMPLATE proposal, chains of alignments could form trees; this allows the same functionality, or full tree functionality. Furthermore, the alignment trees could be treated as static or dynamic structures when arrays were realigned (i.e. a node could be taken out of the tree alone, or it could take its children with it). The Vienna proposal explored the complex (dynamic tree) form. Chuck Koelbel noted that the distribution group had considered alignment trees at one time, and voted them down. Marc Snir supplied the reason for that decision: visibility issues in defining the alignments in subroutines.

The distribute declaration kept the syntax basically the same, but introduced two new orthogonal issues:

- More general block distribution GENERAL_BLOCK
  HPF has limited expressiveness now, GENERAL_BLOCK allows different-sized blocks on each processor. This capability is useful in many circumstances.
- Subsets of processor arrays
  These were described as "very useful and general"; in the hurried presentation, it wasn't clear what the connection was to the processor sections described earlier.

Moving quickly through the draft, Barbara described ALIGN and REALIGN as having the same functionality as before (actually, REALIGN now has more functionality due to the alignment trees). The same was true of REALIGNABLE and REDISTRIBUTABLE, except that they just applied to interfaces (within a routine, it is simple to check whether an array is remapped). In the subroutine interface section, they suggested minor syntax changes to the

```
ALIGN WITH * :: a
```

feature to reflect the model that a distribution was being copied rather than an abstract alignment, but the same functionality was available. ALLOCATABLE arrays could be handled by static distributions as the distribution group now suggested. In general, capabilities in their proposal were "not terribly different from what you have now." Her conclusion slide was simple:

There are *no* inherent problems with ALIGN, REALIGN, DISTRIBUTE, and REDISTRIBUTE without TEMPLATES

Having said that, Barbara's next slide was titled "Problems Brought Up Without TEMPLATES". The basic problem was defining alignment for arrays that do not completely overlap (her slide had a picture of two overlaid rectangles, with one shifted up and right about two units). Several solutions were suggested:

Solution 1 : Extend one array as needed. (The extra space may be negligible.)
Solution 2 : Use templates and modules (in full Fortran 90). (Avoid ALLOCATABLES.)
Solution 3 : Keep templates strictly local.
Solution 4 : Align sections, and extend distributions.
a. Let someone else do it
b. Let the compiler do it
c. Use the GENERAL_BLOCK and processor subsets
d. Find another distribution function

Mary Zosel discounted solution 1, saying it loses conformable array operations (A+B) because array bounds have changed. Explicit triplets are needed everywhere, which is painful. Guy Steele and Rich Shapiro suggested pointers to slices to solve this, and Clemens Thole noted other problems with that approach. Guy Steele suggested another variant of solution 4: use the COMMON solution of distributing only covering variables. Joel Williamson repeated a suggestion by Rich Shapiro to define a data type of size 0, and call that TEMPLATE. Then templates are first class objects.

Chuck Koelbel and Rob Schreiber questioned whether inherited distributions were always describable under the no-TEMPLATE proposal. Their final example was

```
PROCESSORS p(2)
REAL x(100), y(200)
ALIGN x(i) WITH y(2*i)
DISTRIBUTE y(CYCLIC(3)) ONTO p
CALL sub(x(10:100:5))

SUBROUTINE sub( a )
REAL a(:)
ALIGN a WITH *
```

An intermediate level is needed to describe a's distribution. This is possible in the caller, because the array used in the alignment is visible. In sub, however, it is very unclear how the parameter can be aligned, and GENERAL_BLOCK does not seem to handle this case. Barbara admitted that she didn't see a way to describe this, but was initially unconvinced that a description was needed. Various attendees invoked the suggestion by Marc Snir that all distributions be describable, and noted that distribution inquiry intrinsics should be able to return something. Clemens Thole suggested the proposal needed an even more general BLOCK.

Guy Steele piped up with the statement "I can't resist mentioning, this is a great place for zero-sized derived types passed to subroutines." He then demonstrated how such a type could be constructed and used in Fortran 90.

```
TYPE template
    SEQUENCE
    REAL foo(0)
END TYPE template
! occupies zero-sized storage units

TYPE(template) t(100,100)
! t is not SEQUENTIAL in HPF so all is OK

CALL subr(t)
```

```
TYPE(template), ALLOCATABLE :: u(:)
ALLOCATE (u(1000))
REDISTRIBUTE u(CYCLIC)
```

This is legal in straight Fortran 90, makes templates first-class objects, and the SEQUENCE statement forces the compiler to allocate no storage for the type. (There is a fine point regarding copying templates on call rather than passing by reference, but this appears solvable.) Vince Schuster joined the fray with the observation "I used to think this was crazy, but the more I think of this the more nice things fall out of it." Mary Zosel and Maureen Hoffert started checking if allocating a zero-sized object is legal in Fortran 90; Ken Kennedy suggested making this the first thing for X3J3 to consider adding to the next Fortran standard.

Ken sought guidance for the distribution subgroup. Rich Shapiro suggested de-emphasizing the explicit templates in the language draft. Marc Snir noted that templates have two functions: split the mapping process into 2 parts for easier processing, and grouping remappings of arrays. The draft should consider these as different issues. Ken noted that inherited distributions require a structure passed across procedure bounds, and the program needs to describe it; Barbara rephrased this issue as deciding how explicit this mapping is. Marc described alignment as trying to describe only part of a mapping. Vince Schuster said he wanted to see what he was aligning to, and alignment trees started getting hairy after a few levels. Mary Zosel picked up on the TYPE(template) construction and asked "Does this mean derived types need to be added to the subset?" (David Loveman ducked under the table at the very thought of this.) Guy Steele pointed out that other Fortran 90 extensions could provide the same effect (for example, allowing derived types with no components). Andy Meltzer and David Presberg discouraged that suggestion on the grounds that changing Fortran 90 is a bad idea. A somewhat protracted discussion of implementation issues for empty data structures followed. Rex Page noted that asking X3J3 to guarantee no memory would be allocated was not appropriate this was an implementation issue. Most of the group agreed, however, that any implementation that allocated a lot of memory for Guy's example would probably not be much good anyway. Guy Steele had the last word on the whole construction: "This is seductively elegant, and that worries me."

Ken Kennedy noted that the group was going in circles and stopped discussion. The TEMPLATE issue was transferred back to the distribution committee. A straw poll asked if it was a good idea to look at this proposal, netting 29 yes votes, 1 no vote, and 7 abstentions. A second poll was called on whether to direct the committee to investigate ways to get rid of the TEMPLATE directive. Mary Zosel asked if we have a ghost of a chance of getting this done by October, and Piyush Mehrotra noted that templates only appear in Chapter 3 of the draft. The poll directing the group to prepare an alternative eliminating TEMPLATE got 12 yes votes, 15 no votes, and 10 abstentions. This was interpreted as a sign to prepare a proposal, if for no other reason than to clarify the disagreement on the issues. David Presberg suggested incorporating the formalism in Barbara's paper into the HPF draft where it was appropriate; Guy agreed that the draft needed to be rigorous, at any rate.

Pointers to Distributed Arrays Redux

Guy Steele took the floor next to present new proposals for pointers to distributed arrays. He described the proposals as based on Clemens Thole's comments the day before, plus "as a bonus" some alternate suggestions.

The first slide presented "Pointer Proposal #1."

> A pointer may be used in place of an array in a REALIGN or REDISTRIBUTE only if and when it is associated with a whole array or whole allocated target object (as the case may be).

(N.B. Need to extend syntax of REALIGN)

The other proposals placed more restrictions on when pointer remapping was allowed.

"Pointer Proposal 2a" followed.

A named object may not be remapped at a time when any pointer is associated with any part of the object.

An allocated target object may not be remapped at a time when more than one pointer is associated with any part of it.

(This is analogous to parameter aliasing)

This reflects an existing HPF rule: if Fortran 90 forbids writing to an array due to aliasing, HPF disallows remapping for the same reason.

Next came "Pointer Proposal 2b."

One may remap an object via a pointer only if the object was created by ALLOCATE by is not an ALLOCATABLE array.

Remapping an object causes all pointers associated with any portion of the object to have undefined pointer association status, except for the one pointer (if any) used to indicate the array in the remapping directive

(This is by analogy to ALLOCATE and DEALLOCATE.)

Clemens Thole asked if this made some Fortran 90 programs with added HPF directives incorrect; the answer was yes. However, if a program is correct with the HPF directives, then it is also correct without them (as is the case with assertions that may lie). Guy described the undefined pointer status as "a clever dance which I borrowed from DEALLOCATE."

The final slide was "Pointer Proposal 2c."

Pointers always work despite any remapping, doggedly hanging on to their associated objects or portions thereof.

(This is by analogy to "Neither rain, nor snow, nor sleet, nor gloom of night...")

This was essentially a strawman proposal, showing the extreme view of allowing pointer remapping.

Guy personally recommended option 2b, but noted that there had been no group discussion to support that (or any other) recommendation. Ken Kennedy called for a series of fast straw polls deciding between the options. The first poll matched proposal 2a over any less restrictive proposal; it garnered 11 in favor, 14 against, and 10 abstain. The next poll matched proposal 2b against the more dogged proposal; 23 favored proposal 2b, 3 were against, and 8 abstained. A final poll was taken on proposal 1; 27 were in favor, 0 opposed, and 8 abstained.

The group then took a well-deserved break.

PROCESSOR Views

After the break, Piyush Mehrotra was scheduled to discuss extensions to the handling of PROCESSOR arrays. Ken Kennedy reassembled the group with "Let's give Piyush our undecided opinion."

The motivation for this proposal was the observation that now HPF can just distribute arrays over entire processor arrangements. Piyush proposed extending this to allow distribution over array sections of the PROCESSOR array. Fortran 90 triplets were allowed in the processor specification, but not vector-valued subscripts. Chuck Koelbel, Rob Schreiber and Marc Snir pointed out that this allowed many new mappings. A straw poll found 14 in favor of this capability, 7 against, and 16 abstaining. Guy Steele interpreted the result to mean Piyush should "prepare a more refined proposal that looks just like that."

Piyush then moved on to introduce PROCESSOR views.

Syntax:

processor-view-statement is proc-view-list

proc-view is processor-name[(array-int-const)]
array-int-const is array-name or array constructor
constraint: rank of array-int-const same as that of processor-name

Examples:

```
PARAMETER (a, (/ 2,1 /) )
PROCESSORS p(10,10), q(100), r(100)
PROCESSOR_VIEW p, q
PROCESSOR_VIEW p(a), r
PROCESSOR_VIEW p((/2,1/)), r
```

Semantics

For each pi (/d1,d2,...,dn/) form a new processor arrangement pi'
such that the j-th dimension is the dj-th dimension of pi
Unravel pi' to form pi'' using Fortran column-major order
EQUIVALENCE all the pi'' one-dimensional arrays

Clemens Thole complained this was very grid-based. Rob Schreiber replied that it was precisely grid machines that motivate this construct. Clemens further noted that a user can't get some nonlinear mappings (serpentine ring mapping, for example) from this proposal; Piyush yielded on that point. Jim Cownie said, tongue firmly in cheek, "Clemens wants to put HPF INDEPENDENT in front of this."

The next slide considered adding subsections to processor views.

Syntax:

proc-view is processor-name[subsection permutation]

Several possible alternate syntaxes for subsection permutation were offered as well, all clearly preliminary.

After some discussion, a straw poll was taken on whether to put some form of processor views into HPF; 18 were in favor, 6 opposed, and 12 abstained. Another poll asked if permutation capability in processor views should be included; 8 voted yes, 9 voted no, and 15 abstained. A final poll asked about adding subsections to processor views. This gave the new functionality of allowing named subsections. It was also decided that a "yes" vote here counted as a "no" for simple processor subsections since it duplicated the functionality. The count was 7 yes, 7 no, and 19 abstain. With that clear guidance, Piyush promised a refined proposal at the next meeting.

Parallel I/O

Before making the I/O proposals, Bob Knighten asked for a straw poll on the proposition "HPF should not touch I/O in this round." The vote showed 14 in favor, 16 against, and 6 abstaining. Marc Snir then proceeded to present the proposal.

The first slide defined the parallel I/O problem for HPF.

A massively parallel machine needs massively parallel I/O
Efficient programs must avoid sequential bottlenecks from processors to file systems
Fortran specifies a file appears in element storage order; this conflicts with striped files (for example, an array distributed by rows may be written to a file striped by columns).

The proposals generally said that HPF should provide explicit control to avoid this.

The next slide reviewed Fortran 90 file organization.

Sequential files (even direct access files - the records are in sequential order, even though they can be accessed out of order)
Record oriented
Storage and sequence association are in force (when writing and then reading a file, for instance)

No specification of the physical organization; no compatibility with other languages/machines is guaranteed

HPF file system need not be Fortran 90 compatible, but that would be *very* nice

Given this, Marc offered a number of solutions for consideration in HPF.

Solution 0: Define no language extensions. (The compiler does it all)

Metadata can be stored with the file to specify its layout.

Does this assume too much of the compiler and file system?

A (small) majority thinks this is insufficient, since it puts lots of burden on the compiler/operating system.

Solution 1: Define hints (annotations) that do not change file semantics, in the spirit of data distribution. (This gives some information to the compiler.)

Solution 1.1 (Piyush Mehrotra): On write, give a hint about how the data will be read.

```
DISTRIBUTE (CYCLIC) :: a
!HPF$ IO_DISTRIBUTE * :: a

WRITE a, b, c
!HPF$ IO_DISTRIBUTE * :: b
```

When an array is written, it can be easily read back in the given distribution. The annotation can be associated with either the declaration or the write itself; in the first case it applies to all writes of the array, while in the second it only applies to the one statement.

The intent is that metatdata is kept in the file system to record the "right" data layout. The advantages of this proposal include notation and efficiency

Solution 1.2: Give hints about the physical layout (number of spins, record length, striping function, etc.) of the file when it is opened.

This uses the HPF array mapping mechanisms. (A file is a 1-dimensional array of records.) The syntax needs a "name" for the file "template": we suggest FILEMAP. The programmer can align/distribute FILEMAP (on I/O nodes), associate FILEMAP with a file on OPEN, etc. There are again no changes in semantics or file system.

Solution 2: Introduce parallel read/write operations that are not necessarily compatible with sequential ones.

```
PWRITE a
PREAD a
```

Data can be read back only into arrays of the same shape and mapping. Data written by PWRITE must be read by PREAD. This solution does not need metadata in file system or charges in file system.

Mary Zosel questioned why no metatdata was needed in solution 2; Marc replied that it was a "user beware feature." Several other options were also suggested for parallel I/O, including hiding it in a library subroutine (difficult to do well without generics), adding a property like "OUT-OF-CORE" to arrays (creating compiler complexity and possibly terrible performance), or defining a new file type in the spirit of direct access files. A discussion followed on the wisdom of changing semantics. Marc invoked the usual restriction that a program that is correct with the IO_DISTRIBUTE annotations is still

correct without them. Chuck Koelbel and Rich Shapiro pointed out, however, that this is correctness across multiple programs and/or machines, a much nastier thing to define. Marc claimed that these annotations restricted to scratch files still provided good functionality for some applications.

Rob Schreiber asked if using READ/WRITE with HPF assertions (promising the data was transferred with the same shapes and mappings) gave the same performance as solution 2. Joel Williamson claimed this put a directive in front of every file operation, implying this was too cumbersome. Rich Shapiro noted many machines can get fast file systems with metadata always included; reads are faster if the distribution matches the metadata. He advocated making the user explicitly REALIGN, copy, or otherwise move the data to create the right distribution. Clemens Thole remarked on the importance of knowing the file format (HPF file or UNIX file or some other format); Marc pointed out that we can't say this in a language standard. Jim Cownie remarked that everyone wants to use their files everywhere, so vendors can provide nonstandard formats if they want them to be unused. Randy Scarborough recommended standardizing a library interface for reading and writing HPF arrays.

Ken Kennedy called for straw polls to guide the I/O group. He expected a series of such decisions, but the first poll found a count of 16 yes, 10 no, and 8 abstain in favor of solution 0 (doing nothing, as opposed to any active solution). Ken Kennedy recommended that the subgroup come back in another round and provide more functionality then. A recommendation was made and seconded that a rationale for not handling I/O be added to the draft; some of the e-mail discussions appeared suitable for this.

Storage and Sequence Association

Mary Zosel presented the storage and sequence association proposal.
The first slide summarized the changes to storage association from last time.

!HPF$ SEQUENCE :: X (spelling change: was SEQUENTIAL)
!HPF$ SEQUENCE (with no name list means that everything in the current scope is sequential)
For the declarations (in different routines)

```
COMMON /a/ x(512,2)
COMMON /a/ x(32,32)
```

- We now require either /a/ or x to be marked SEQUENCE everywhere.
- Any variable in a COMMON block not declared sequential or part of an aggregate variable group is fair game for compiler-generated mapping

Regarding the first point, Ken Kennedy asked, "Is it your intent to confuse sequence and storage association here?" A hearty "YES!" came from Maureen Hoffert, apparently favoring going back to SEQUENTIAL as the keyword. Guy Steele asked if this was consistent with the use of SEQUENCE in derived types; Mary responded that, by a narrow margin, the vote in the subgroup was "yes" on that question.

Mary then gave a review of the proposal.

COMMON
Sequential
Consists of a single component
1 cover of the component may be mapped
Nothing else related may be mapped
The cover mapping, type, shape must be the same everywhere
All instances of this block are declared sequential everywhere
Storage/sequence association work

Cover mapping is a 1-D template mapped to 1-D processors
(still working the details of this)

Nonsequential

Components are variables or aggregate variable groups (avgs).
Components match in size everywhere.
The COMMON may include a mix of sequential and nonsequential components.
Nonsequential components match type, shape, and mapping everywhere.
All instances are NONSEQUENCE
If a component is an agv, 1 cover variable is the same everywhere.
All nonsequential variables not in an agv may be mapped.
The restrictions on cover mapping apply again.

Joel Williamson claimed he didn't understand the proposal. Ken Kennedy explained it was based on simple principles:

Nonsequential objects can be explicitly mapped.
Sequential objects can only be mapped by the cover mechanism (if one array is totally associated with the object, that array can be mapped).

Clemens Thole asked about COMPLEX covers, which might end up splitting a COMPLEX array element across processors. Richard Swift wanted that feature considered as being out of the subset due to gross inefficiency; this was deferred until later.

The review continued.

Local variables

Sequential if any of the following is true
Member of an aggregate variable group (agv)
Assumed size
Component of a SEQUENCE derived type
Declared SEQUENCE
Storage/sequence association work on these variables; only a cover can be mapped.
Other variables are nonsequential - can be mapped

One remaining issue was whether HPF needs a !HPF$ NOSEQUENCE directive. Andy Meltzer argued this provided another level of compile-time checking, but Rob Schreiber was scared that NOSEQUENCE, which is the default, would require explicit declarations. Richard Swift claimed users don't want this since it is hard to use. Vince Schuster noted it was helpful as a debugging tool. Ken Kennedy said the group should view NOSEQUENCE as an amendment to the proposal, while the others would be straw polls. Jim Cownie asked why it was needed; several people suggested it was to override a SEQUENCE directive applied to an entire scope. In any case, it was not required since it is the default. A vote was finally taken to add NOSEQUENCE to the proposal; the results were 8 in favor, 12 against, and 9 abstaining. Another vote was taken to approve the existing proposal (with possible straw poll modifications); the results were 27 in favor, 0 opposed, and 2 abstaining.

The discussion then turned to the restrictions on the mapping of cover variables. Only 1-D covers are allowed now. A related issue was whether HPF needs to say anything about the mapping of sequential objects. Rich Shapiro suggested that we shouldn't say anything at all, including whether there was a mapping. As it was too technical an issue to handle in the plenary session, the restrictions on mapping of covers were sent back to the committee without a poll. One final poll asked for advice on naming the SEQUENCE directive. "SEQUENCE" got 7 votes, "SEQUENTIAL" got 5 votes, and "Don't care" emerged with a solid majority.

The group then broke for lunch, and to check out of their hotel rooms.

Distribution Inquiry Intrinsics

Rich Shapiro was the first speaker after lunch, presenting a proposal for new distribution inquiry intrinsics.

The idea of the proposal was simple - programmers need the new functionality to check how a variable is mapped, and there are several ways to do this. Rich presented two proposals for a set of new intrinsics along these lines.

Proposal #1: A few complicated things

```
call hpf_inquire_align( array, lb, ub, stride, axis_map,
...)
```

Advantage: Little new syntax, few new functions
Disadvantage: Can't use subroutines in specification expressions

Proposal #2: Lots of simple things

```
hpf_inquire_align_lb(array,dim)
hpf_inquire_align_ub(array,dim),
etc.
```

Advantage: specification expressions
Disadvantage: Lots of syntax, meaningful names are long

Rich and Marc Snir planned to merge this proposal with the intrinsics in the LOCAL functions proposal. For now, however, he asked for guidance: did the group want a few big functions, or lots of little ones?

Clemens Thole was the first to suggest filling in a big structure rather than returning separate parameters. Others suggested using optional and keyword parameters rather than having all parameters required. Joel Williamson suggested that because of the large number, the functions could be part of the library; Rich replied that they couldn't be used in specification expressions then, which was very important. He admitted, however, that it was difficult to use the functions in expressions anyway because of the name lengths. Clemens asked why HPF needed to specify this; in normal code, how can you it use it? Several people suggested algorithms that took distributions into account, particularly in libraries. Joel Williamson suggested reserving a range of distribution types for future HPF standards, not vendor implementations. Rex Page further suggested representing the types as strings. In reference to the "few complicated routines" design, he asked "How many returns will you use when you call it?" Rich replied that almost all the values would be used for an FFT library, didn't expect to use the inquiries very often. A straw poll was taken asking if people were in favor of a few complex things (over many simple things); the count was 16 yes, 9 no, and 10 abstain. Another poll compared the complex interface with many optional arguments (over returning a single structure); that count was 19 yes, 3 no, and 10 abstain.

Official Subset Redux

Mary Zosel next led a series of votes on what HPF features (if any) should be excluded from the official subset. The first order of business was an official vote to allow new HPF directives to be excluded. Jim Cownie noted that if we do not allow exclusion, then an anomaly occurred: redistribution of a global (requiring modules) would be is in full HPF, but the mechanism to do it would not. Ken noted the remark, but wanted an official vote to satisfy parliamentary rules anyway. The count was 25 in favor of excluding some HPF features from subset, 2 against, and 3 abstaining.

Mary quickly moved on to consider categories of features to exclude. All these decisions were straw polls, since this was a first reading of the subset proposal. The HPF library (nee intrinsics) were out of the subset, by the previous poll. Next up was the

FORALL. Rich Shapiro moved that the single statement FORALL be included, and all else removed from the subset. Clemens Thole amended this to be no FORALL in the subset at all. For once, the discussion was short: Clemens noted FORALL was nonstandard Fortran 90, and Rob Schreiber noted that HPF without FORALL was a ridiculous language. The poll on the amended proposal found 7 yes votes, 22 no votes, and 4 abstentions. The vote on including the single-statement FORALL only then received 19 yes votes, 6 no votes, and 8 abstentions. As it is an assertion, INDEPENDENT was assumed to be in the subset without a formal motion. The question of including LOCAL routines was deferred to the next meeting, with a request to the subcommittee to recommend one way or the other.

In the area of mappings, Vince Schuster moved that only static distributions be included in the subset. Clemens Thole noted that the runtime system must redistribute anyway for subroutine linkage, and asked what was gained by Vince's proposal; Rich Shapiro noted that it eliminates a good deal of analysis that would be needed otherwise. Guy Steele and Piyush Mehrotra offered the friendly amendment that the subset not include REALIGN, REDISTRIBUTE, REALIGNABLE, or REDISTRIBUTE. A poll on this phrasing received 16 yes votes, 7 no votes, and 9 abstentions.

The discussion then turned to storage association. Ken Kennedy recommended that storage association be in the subset but not in the language as a whole; although there was wide agreement with this sentiment, it was not taken as a serious suggestion. Mary Zosel and Richard Swift suggested avoiding breaking a DOUBLE PRECISION or COMPLEX across processor boundaries, but didn't have a good wording for the idea. Joel Williamson proposed removing distribution of sequential variables (including covering variables) from the subset. Ken objected that there are lots of innocent things you can do with these variables. Joel replied "When I said that at the January meeting, you and Geoffrey Fox jumped all over me like a cheap suit saying this wasn't about porting codes, it was about writing new ones." Having gotten that off his chest, he left the proposal as it stood. Clemens Thole reminded the group of the "Recode once" goal. A poll was finally taken to remove the ability to distribute any sequential things from the subset; the result was 17 in favor, 8 opposed, and 6 abstaining.

Richard Swift brought up the issue of Fortran 90 I/O in the subset. Rich Shapiro moved that NAMELIST be included in the subset, leading to a poll of 10 in favor, 6 opposed, and 15 abstaining.

Documentation and Editorial Matters

David Loveman was the last formal proposal, presenting a number of issues related to generating the draft High Performance Fortran language specification.

David's first slide was labeled "0. Officialness of text in the current draft."

The fact that a statement appears in the draft at this point means:

1. I have it
2. It runs through LaTeX.
3. Nothing else

Several TeX hackers at the table, however, complemented him on eliminating the "Underfull hbox" messages during the TeX processing.

Dave next presented the objectives of the current drafting process, asking for feedback from the rest of the committee.

- Debug the document distribution method. (Isn't "standard" UNIX wonderful?)
- Debug the document structure.
- Debug and make consistent the LaTeX usage for the document.
- Be able to see the whole elephant.

  Writing style

  Topic factoring into chapter

Correctness

The draft document is available by anonymous FTP from titan.cs.rice.edu in the directory public/HPFF/draft. Both a Postscript version and a UNIX tar file containing the LaTeX source are there. In addition, Dave or Chuck Koelbel will send a uuencoded, compressed, tar file by electronic mail on request. Dave requested anybody who received such a distribution and could not generate the document from that point to contact him; there are a "surprising number" of system limitations that had cropped up while distributing the draft to the committee. Nobody expressed major complaints about the document structure, except that the "examples" section was completely empty.

The next topic was issues for the document in the future.

- Correctness
  - Reflect decisions through this meeting
  - Factor new stuff into the language specification and the Journal of Development
- Writing style
  - Transform informal proposals into formal specifications
- Consistency
  - Unify word usage, use of Fortran 90 concepts, and treatment of HPF concepts

All were deemed to be worthy goals. The subgroup leaders were put in charge of these issues for the chapter (or chapters) they were working on. Several volunteers came forward to look at inter-chapter consistency, once the individual sections were in better shape.

Finally, Dave gave a proposed schedule for generating the draft for the next meeting.

- All updated text (representing decisions at this meeting) back to David Loveman in 2 weeks - i.e. by September 28 - from the group leaders.
- Edit binge to smooth the document - 1 week - Chuck Koelbel & David Loveman.
- Full document proofread - 1 week - Volunteers needed!
- Update the edit pass and distribute the draft to the core group in time for committee to read and proof before the October meeting.
- Then, change it all around in the next meeting.

Stable drafts generated in this process will be available by FTP as well.

A lively discussion of what goes (or should go) in the document started next. Mary Zosel suggested that the first sentence in each section give its status (as accepted on second reading, proposed for first reading, etc.). Dave wanted to put that information into a footnote for easier conversion into a final draft, but otherwise was in general agreement. Piyush Mehrotra asked how alternative proposals should be treated, and Chuck Koelbel suggested identifying them in a separate section. This leaves all the text to be discussed at a meeting in 1 place, which was one of the original goals of having a draft. Ken Kennedy wanted no alternatives in the draft, but Dave and Mary Zosel noted that having two freestanding sections was very useful for comparisons. Guy Steele offered a macro to generate alternatives, shown as two sections with the same number. This approach was adopted, with the group heads responsible for providing good chapters.

Future Plans

Ken Kennedy held a short business meeting to end the technical part of the forum. He noted that HPFF had volunteered to organize a workshop at Supercomputing '92 in Minneapolis. This would be an excellent chance to present the material discussed, preferably by having each subgroup leader present the major feature group he or she had overseen. With the exception of Rob Schreiber, who still needed to make his travel plans, all the group leaders agreed. Clemens Thole mentioned that a similar presentation in Europe would also be welcome.

Ken then drew up the agenda for the next HPFF meeting.

- Second readings
  - LOCAL subroutines
  - HPF subset
  - New intrinsics
  - Distribution odds and ends
  - PURE Functions
  - Distribution of covers
- First readings
  - New FORALL assertions
  - No TEMPLATES
  - Other things from distribution group
  - Execution model
  - Examples

(Several of these entries were filled in during the discussion.) Ken asked whether this schedule required two full days of meetings, or if one and a half was sufficient; Chuck Koelbel quickly recommended two days, as the hotel was already booked in any event.

Low Performance Fortran

Andy Meltzer ended the meeting with his presentation of "Low Performance Fortran III." He claimed, "I thought LPF was done, but" and Ken Kennedy jumped in with "It's back by popular demand!"

FREE_FOR_ANY - (extended from last time)
  Means some arbitrary statement in the program is executed
  Too easy to do - to avoid this, we'll add the requirement that all statements must be labeled with

```
FREE_FOR_THIS_ONE
```

  or

```
NOT_FREE_FOR_THIS_ONE
```

Pointers
  While it's impolite to point, there are impolite programmers
  Pointers can point to anything in the program, but the compiler chooses what
Nested ONCE_AND_FORALL
  Your mother might tell you this
FOREIGN routines (renamed ALIEN)
  To ensure alienness, all known compilers must be proved not to compile ALIEN code. Only then can it be called from LPF
I/O
  No I/O is necessary. users should inspect core dumps
LPF subset -
  Includes all of HPF. But none of HPF is in full LPF.
LPF INDEPENDENT
  Only applied to (now removed) TEMPLATES

Ken thanked Andy for ending the meeting on a high note, and the attendees ran off to catch their planes.

>
>