EPCC-TN92-05.04

High Performance Fortran: A Perspective

Brian J N Wylie
Michael G Norman
Lyndon J Clarke

May 1992

Abstract

High Performance Fortran (HPF) promises an attractive abstraction, with high-performance potential, for Fortran programs on parallel (and massively-parallel) computers. This report is intended to provide background information on the HPF initiative, the technical aspects thereof, as well as our impressions of why this activity is progressing with the apparent enthusiasm of a number of parallel computer vendors and other interested parties.

It is contended that the array-parallel operations of Fortran-90, coupled with additional directives providing information about appropriate decompositions and distributions, can enable compilers with sophisticated dependence analysis and knowledge of the MIMD parallelisation techniques to produce native message-passing code for efficient and effective utilisation of distributed-memory parallel computers.

In addition to providing a perspective of the developing HPF situation in the United States, European activities and involvement are also addressed. It seems clear that the considerable momentum which HPF possesses, coupled with its attractiveness to customers of parallel computing, will make it an increasing force to be reckoned with in the numerical and scientific computing market.

Early versions of HPF are, however, likely to have only limited application areas, since they will have very restricted MIMD support, and neither unstructured data objects nor sophisticated load-balancing support, although such developments are likely in the future. Fortran-90 has been chosen as the base for HPF development, and Fortran-77 will not be supported. This is despite the fact that the US DoD requires Fortran-77 on its contracts, and recommends against the use of Fortran-90. There are also worries that no consideration has been made of providing HPF semantics in C.

Contents

1 What's the HPFF?

1.1 Announcement

The information in this section is largely extracted from the HPFF 'Net announcement of mid-March '92 on the comp.parallel *newsgroup. No formal definition of the project goals is currently known about, although the HPFF plan to produce preliminary documentation soon (May '92).*

Since its introduction over three decades ago, Fortran has been the language of choice for scientific programming for sequential computers. Exploiting the full capability of modern architectures, however, increasingly requires more information than ordinary Fortran-77 or Fortran-90 programs provide. This information applies to such areas as

1. Opportunities for parallel execution
2. Type of available parallelism — MIMD, SIMD, or some combination
3. Allocation of data among individual processor memories
4. Placement of data within a single processor

The High Performance Fortran Forum is a coalition of industrial and academic groups working to identify the basic issues and suggest a set of standard extensions to Fortran to provide the necessary information. To date, the group includes most vendors currently delivering parallel machines, government labs, industry and many university research groups. The intent is to develop extensions to Fortran which provide support for high performance programming on a wide variety of machines, including massively parallel SIMD and MIMD systems and vector processors. Some of the topics considered will be

1. Data distribution and alignment
2. Explicit parallelism, particularly for data-parallel problems
3. Computation distribution
4. I/O support

The result of this project is intended to be an industry-wide standard language portable from workstations to massively parallel supercomputers. Furthermore, it is hoped that the language will be able to express the algorithms needed to achieve high performance on specific architectures over this range.

Parties with an interest in using Fortran on modern architectures are welcome to participate in HPFF. In particular, comments are welcomed from

1. Fortran users writing new applications for modern architectures
2. Fortran users porting programs to new architectures
3. Fortran compiler and programming environment implementors
4. Researchers interested in extending Fortran

1.2 Mechanics of the Forum

A partial list of parties (which should be considered representative, rather than comprehensive) currently involved in HPFF is as follows:

Laboratories *ICASE* (Piyush Mehrotra, Joel Saltz), *LANL* (Ralph Brickner), *LLNL* (Mary Zosel), *RIACS* (Rob Schreiber), *ITI-TNO/TU Delft* (Henk Sips), *GMD* (Clemens-August Thole),

Universities *Rice U* (Ken Kennedy, Chuck Koelbel), *U Vienna* (Hans Zima, Barbara Chapman), *Syracuse U* (Geoffrey Fox, Alok Choudhary), *Yale U* (Marina Chen), *Auburn U* (Cherri Pancake), *Cornell U* (David Presberg), *SUNY Buffalo* (Min-You Wu), *Louisiana SU* (J Ramanujam), *Ohio SU* (P Sadayappan), *U Stuttgart* (Roland Zink),

Parallel Computer Vendors *TMC* (Bob Milstein, Guy Steele, Richard Shapiro), *Convex* (Bob Metzger, Joel Williamson), *IBM* (Jorge Sanz, Randy Scarborough, Marc Snir), *DEC* (David Loveman, Bert Halstead), *CRI* (Tom McDonald, Andy Meltzer), *Intel SSD* (Bob Knighten), *nCUBE* (Barry Keane, Venkata Konda), *Alliant* (David Reese), *MasPar* (Richard Swift),

Miscellaneous *Hewlett Packard* (Maureen Hoffert, Tin-Fook Ngai, Richard Schooler), *Applied Parallel Research* (John Levesque, Rony Sawdayi), *Amoco Production* (Rex Page), *Portland Group* (Vince Schuster), *Schlumberger* (Peter Highnam), *Shell* (Don Heller), *Sun* (Douglas Walls), *United Technologies* (Rich Shapiro).

Attendance at meetings is open to all interested parties. Each organisation (company, school, etc.) is limited to two representatives, however, in order to keep the meeting size manageable. Every organisation may have one voting representative. The organisation should name a single individual for this task; if that person cannot attend, however, another can cast votes instead. To be eligible to vote, a representative must have attended 2 of the last 3 meetings, counting the current meeting. (That is, miss two consecutive meetings and you can't cast a vote.) Eligibility is decided based on organisation.

Meetings have evolved to be day-and-a-half gatherings, and although originally planned to alternate between different locations within the Unites States, to simplify administration the majority of meetings will end up being in Dallas.

The current HPFF meeting schedule is as follows:

Date	Location	Agenda
27–28 Jan	Houston	inaugural meeting
09–10 Mar	Dallas	
23–24 Apr	Dallas	
08–10 Jun	Dallas	data distribution, array sub-language
23–24 Jul	Washington	storage association, dynamic redistribution
09–11 Sep	Dallas	`FORALL` construct, local subroutines, intrinsic functions
21–23 Oct	Dallas	official Fortran-90 subset, miscellaneous features
02–04 Dec	Dallas	approval of final draft

A draft proposal is to be presented at *Supercomputing '92*, with the final draft due for January 1993. The first HPF compilers should also become available in early 1993.

Several working groups have been established, to discuss (largely via E-mail) specific parts of HPF:

1. Relationship between HPF and Fortran-90 (Mary Zosel, hpff-f90)
2. Models of data and work distribution (Guy Steele, hpff-distribute)
3. Subroutine interfaces & automatic redistribution (Joel Williamson, hpff-subroutines)
4. FORALL statements, local subroutines & explicit process control (Chuck Koelbel, hpff-forall)
5. Concurrent I/O and miscellaneous features (Bob Knighten,hpff-io)
6. New intrinsic functions for parallel programming (Rob Schreiber, hpff-intrinsics)

All working group mailing lists are kept at cs.rice.edu, along with the main hpff mailing list for meeting minutes and miscellaneous announcements. To date, almost all of the activity has been on the 'distribute' group, and the 'subroutines' group has been (temporarily?) merged with it.

1.3 Goals

The following goals have been identified by the Forum:

- Language extensions and feature selection for Fortran, supporting:
  - Data parallel programming (defined as single thread, global name space, and loosely synchronous)
  - Top performance on MIMD and SIMD computers with non-uniform memory access costs (but should not impede performance on other machines)
  - Code tuning for various architectures
- Minimal deviation from other standards
- Minimal direct conflicts with Fortran-77 and -90 (if possible, with no direct conflicts)
- Open interfaces to other languages & programming styles
- Compiler availability possible in near term (1993) with demonstrated performance on HPF test suite
- Produce validation criteria
- Provide input to future standards activities for Fortran and C
- Maximal simplicity!
- Involvement (input) of high performance computing community; widely distributed language drafts
- Present final proposal in November '92; final draft in January '93
- Leave an evolutionary path for research

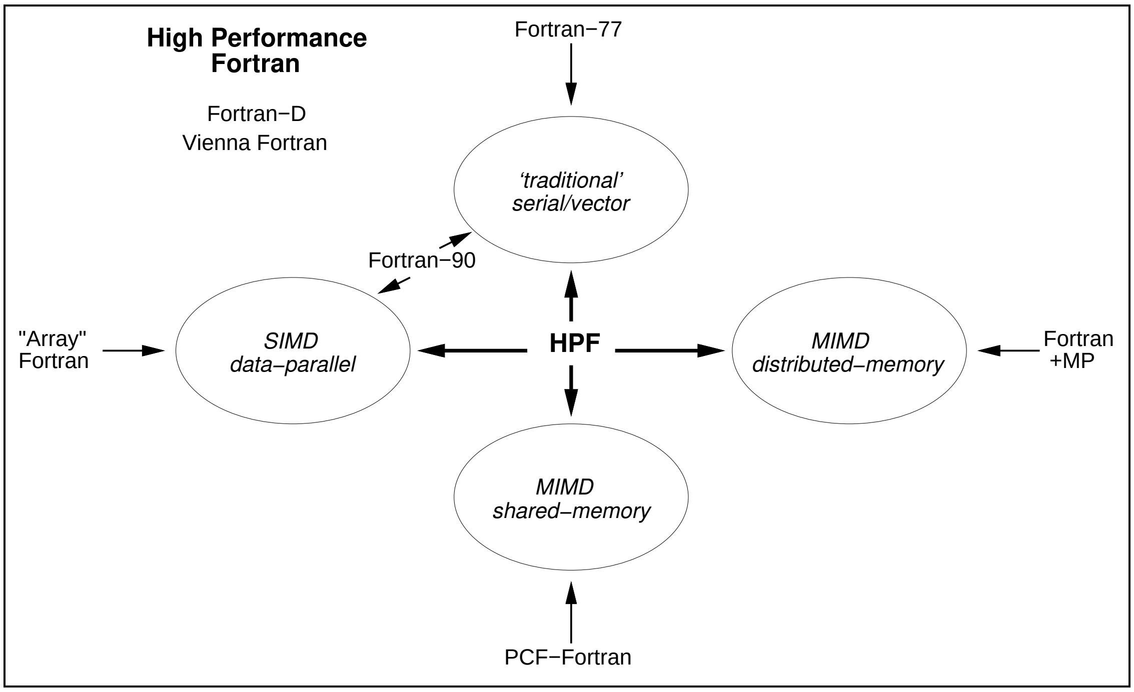

Figure 1: The rôle of High Performance Fortran

2 What's currently available?

The HPF developments are seen as a natural and necessary culmination of a number of existing Fortran language development schemes: in particular, it addresses the limitations of Fortran-77 and Fortran-90 for parallel and massively-parallel computing. Programmers who currently use Fortran (in any of its existing varieties) will undoubtedly still be using a language called Fortran in the next century (regardless of whether it looks more like C++ or occam than Fortran-77) and HPF is part of that evolution. There has been no consideration of providing HPF semantics in C.

While 'traditional' Fortran-77 proved capable of exploiting performance vector architectures through the introduction of a few directives and the rewriting of small code sections, this was not the case for computers with specialist parallel architectures. A number of different Fortran variants were developed, each appropriate to only its targetted architecture (see Figure 1).

PCF-Fortran offered additional primitives for shared-memory systems, 'array'-Fortran variants (such as CM-Fortran and DAP-Fortran) were tailored for SIMD machines and data-parallel operation, and distributed-memory required the introduction of explicit message-passing calls within Fortran programs. Fortran-90 went a long way towards incorporating the widely useful 'array'-parallel operations within the framework of its expanded language definition, but did little to facilitate the use of distributed memory computers. Although HPF, and its immediate predecessors (Fortran-D and Vienna Fortran), do not provide a unification of all the various Fortran variants for the diverse parallel architectures, they do provide a portability platform, enabling a single code to run, and be tuned for near-optimal performance, across the available architectures.

Some of the key language developments en route to HPF are outlined in the rest of this section. Fortran-90 incorporates a considerable number of language extensions to provide the Fortran programmer with constructs and abilities available to programmers in other modern high-level languages. CM-Fortran covers that subset of these extensions applicable to data-parallel computers such as the Connection Machine. Fortran-D and Vienna Fortran offer compiler directives which provide additional data decomposition and distribution information, in a similar manner to that commonly used in compilers for vector computers, e.g., Cray Fortran vectorising directives.

2.1 Fortran-90

As the long-debated and long-awaited successor to Fortran-77, Fortran-90 promises a great deal to the programmer. The standard represents a major development in the Fortran language, providing new programming constructs and a concise ability to express natural problem data-parallelism. A number of the out-dated and idiosyncratic features of the language have been deprecated — although none have been removed from the language at this stage, since Fortran-77 is a complete subset of Fortran-90. Details are summarised in [16] and extensively documented in [10].

Fortran-90 therefore provides an effective high-level language for numerical and scientific problems, which should be natural for such programmers. In the embodiment of a powerful and much-needed standard, it also provides a mechanism for the unification of programming practises which have evolved in Fortran-77 programs: one particular traditional problem area has been dynamic memory allocation, which has been so oft-encountered that the number of diverse solutions to it may well of itself preclude the further use of Fortran-77.

One cloud on the horizon is the general belief in the Fortran community that full Fortran-90 isn't very interesting, and largely a distraction from the 'real' business of Fortran programming. Accordingly, the Center for Standards of the US Department of Defense announced in November 1991 the rejection of Fortran-90 for DoD contract work, with the recommendation to stick with Fortran-77 (with various extensions) and Ada. (As reported on USENET and in [22])

2.2 CM-Fortran

The part of the Fortran-90 language which provides a concise and coherent embodiment of array processing operations, which would otherwise have required 'DO-loops' (cycling over the array indices), is often referred to as the "array subcomponent" or "extended array section features" of Fortran-90. This is naturally exploited in data-parallel programs on SIMD architectures, such as the CM-2 by Thinking Machines Corporation (TMC). [19]

The CM-Fortran language developed by TMC [20] essentially only caters for the array subcomponent of the Fortran-90 language, and additionally provides a few extensions to it which are natural for SIMD architectures, e.g., the (single-statement) `FORALL`, and a few compiler directives:

```
      DIMENSION array (M1,M2,M3), vector (M1)
CMF$  LAYOUT array ( :SERIAL, :NEWS, :SEND)
CMF$  ALIGN vector (I) WITH array (I,1,1)
CMF$  COMMON FEONLY /comblk/
```

TMC plan at some unspecified stage in the future to upgrade CM-Fortran to include Fortran-90 as a full subset, but this is not seen as central to the use of CM-Fortran in practise. While SIMD architectures can effectively use the array subcomponent of Fortran-90 for parallel performance, other architectures have only benefited from the clarity and conciseness it provides.

2.3 Fortran-D

Fortran-D is primarily a result of on-going work by Ken Kennedy and his group at Rice University (Dallas, Texas, USA). The 'D' in Fortran-D is not specified, but is said to stand for any and all of: directives, data, decomposition and distribution. [6, 7]

It is designed to support what are, in the views of its authors, two fundamental stages of writing a data-parallel program: *problem mapping*, that is indicating the co-location of array elements (which are commonly accessed together in program statements) so as to minimise the requirements for communication, and *machine mapping* through a rich set of data distribution functions. Fortran-D is intended to provide a simple machine-independent programming model for most numerical computations, although it appears most naturally appropriate for 'embarassingly parallel' programs.

Compiler algorithms which automatically derive efficient message-passing programs based on data decompositions are used. Optimisations are used to minimise load imbalance and communication costs for both loosely synchronous and pipelined loops. User direction is seen as greatly extending the exploitable parallelism.

2.4 Vienna Fortran

In much the same style as Fortran-D, the work of Hans Zima and his group at the University of Vienna (Austria) has produced a similar scheme through language extensions, although in a syntactically cleaner and more robust way.

Vienna Fortran [1] is a language extension of Fortran which provides a wide range of facilities for mapping of data structures across distributed-memory processors. Programs in Vienna Fortran are written using global data references, providing the advantages of a shared memory programming paradigm while explicitly controlling the placement of data.

An effective collaboration between the Vienna and Rice groups seems to be, at least in part, one of the driving forces behind HPF developments.

2.5 PCF-Fortran

An attempt to standardise Fortran in a shared-memory context appears to be on-going by a consortium known as the Parallel Computing Forum (PCF, ANSI committee

X3H5). The features of shared memory parallelism under discussion were access to locks, events and barriers, and `FORALL`-type constructs to explicitly specify parallel executability.

The politics of the likely failure of this activity are unclear, although there appear to have been some organisational difficulties, and a technical sticking point over the inability to remove non-determinism from the language — or more particularly the impossibility of showing whether a non-deterministic program is correct, and the inability to recognise such a program in the first place.

3 What's possible and being proposed?

3.1 HPF and the other Fortran dialects

HPF is basically a fusion between Vienna Fortran and Fortran-D, with Fortran-90's array operations and dynamic array allocation facilities. It is unlikely there will ever be a Fortran-77 version because of the problems of distributing the arrays that are allocated in the various hacks used to simulate dynamic memory allocation. Use of a modern language, like Fortran-90, provides a standard method for dealing with features (such as dynamic memory allocation) which would otherwise require a number of compromises and special cases in Fortran-77. In addition to the full Fortran-90 binding to HPF, there will also be a specified "official subset" of Fortran-90 (including the entire non-character array sublanguage) which will be supported.

It is widely considered that Vienna Fortran is more elegant than Fortran-D. As the two developing groups have an effective collaboration there appears to be no barrier to features of Vienna Fortran being included in HPF in preference to those of Fortran-D. Accordingly, Vienna Fortran is described in later sections. Much of this elegance may not remain in the final HPF, however, as preliminary examples of proposed HPF directives have appeared rather cumbersome and 'noisy.'

HPF compilers may initially be modified Fortran-D compilers, although both Fortran-D and Vienna Fortran have advanced implementations. Apparently, Zima is not playing a direct rôle in driving HPF developments, and the majority of the running is being done by the Americans. It is important to note that these 'compilers' are source-code to source-code translators. The target language is a form of Fortran with added message passing constructs. The message passing and associated code is referred to as the Run Time Executive (RTE). It is likely that in due course HPF compilers will target hardware features directly.

3.2 HPF capabilities

One of the main desires is for unmodified Fortran programs to execute effectively on diverse high-performance (i.e., parallel) architectures. This leads to the abstraction whereby code with a single thread of control is written in Fortran, containing directives specifying the desired/required data mappings. An HPF compiler/translator undertakes the necessary code transformations and inserts the system-specific message-passing, etc, which will enable the code to run after being subsequently compiled by the native compiler for the parallel system.

HPF can be seen as a fusion of two different existing technologies: the "hypercube experience" in developing the tools and techniques used to write explicit message passing programs for data parallel applications [5], and the compiler work in dependency analysis used in optimising compilers for superscalar architectures and in shared memory parallelising compilers [13, 4]. The hypercube experience brings with it an understanding of the tricks that are required to get a message passing program to execute efficiently. The compiler work allows the definition of (a subset of) the legal transformations that may be made to the program. Together they allow the automatic transformation of a subset of data parallel sequential programs into efficient message passing programs.

There are two problems that HPF has to solve: *mapping of computation to processors* and the *formulation of the communication* implied in so doing. Optimisation of mapping is of course NP-Complete in most formulations, and also incredibly difficult to formulate (see [12]). The approach in HPF is that the transformations are guided by statements made by the user in the form of compiler directives (which are often executable statements).

3.3 Language overview — Vienna Fortran

Mapping of computation to processors in Vienna Fortran is performed implicitly by the mapping of multidimensional data arrays. The computation follows according to the "owner computes" rule: the processor to which a data point has been mapped is responsible for assigning values to it. Mapping functions and inverses are required for all data arrays. Scalars and non-decomposed arrays are replicated.

The Vienna Fortran language extensions (as described in [2, 23]) provide the following features:

- The **processors** which execute the program may be explicitly specified and referred to. It is possible to impose one or more structures to them.
- The **distributions** of arrays can be specified using annotations. These annotations may use processor structures introduced by the user.
  - Intrinsic functions are provided to specify the most common distributions.
  - Distributions may be defined indirectly via a map array.
  - Data may be replicated to all or a subset of processors.
  - The user may define new distribution functions.
- An array may be **aligned** with another array, providing an implicit distribution. Alignment functions may also be defined by the user.
- The distribution of arrays may be changed **dynamically**. However, a clear distinction is made between arrays which are statically distributed and those whose distribution may be changed at run-time.

- In **procedures**, dummy array arguments may
  - inherit the distribution of the actual argument, or
  - be explicitly distributed, possibly causing some data motion.
- A **forall** loop permits explicit parallel loops to be written. Intrinsic reduction operations are provided, and others may be defined by the user. Loop iterations may be executed
  - on a specified processor,
  - where a particular data object is stored, or
  - as determined by the compiler.
- **Assertions** about relationships between objects of the program may be inserted.

Vienna Fortran does not introduce a large number of new constructs, but those it does have are supplemented by a number of options and intrinsic functions, each of which serves a specific purpose. They enable additional control to be exerted over the manner in which data is mapped or moved, or the code is executed. HPF proposals appear to incorporate the majority of this Vienna Fortran functionality, with the likely exception of user-defined distribution functions.

3.4 Language details — preliminary HPF

Abstracted from Guy Steele's April 21st proposal to HPF [17] and the minutes of the April 22nd HPFF meeting [8]:

Data (i.e., array) distribution in HPF will incorporate a three-level mapping which combines features of Fortran-D and Vienna Fortran. In outline, (hardware-independent) *data arrays* are mapped to conceptual entities called *templates* (which were previously called decompositions) which are in turn mapped to (virtual) *processor grids*. Processor grids are in turn mapped onto (hardware-dependent) *physical processors*. A diagram of this model is shown in Figure 2.

The general idea is that every array is created with an alignment to some template, and every template is created with some distribution. If the specification statements contain explicit specification directives specifying the alignment of the array, and perhaps also the distribution to which the array is aligned, then that information is used when the array is created. Once an array has been created, it can be moved about by realigning it or redistributing the template to which it is aligned; there's then work involved, however.

There's a clear separation between specification statements and executable statements. Specification statements are carried out on entry to the program unit, and only then are executable statements carried out.

A directive has the form of a comment, introduced with the characters "`!HPF$`", and may be continued on following lines, if necessary. Various proposals to extend this syntax to include "`CHPF$`" in fixed-source form, or restrict it to only apply to complete comment lines (rather than end-of-line comments) are currently being considered.

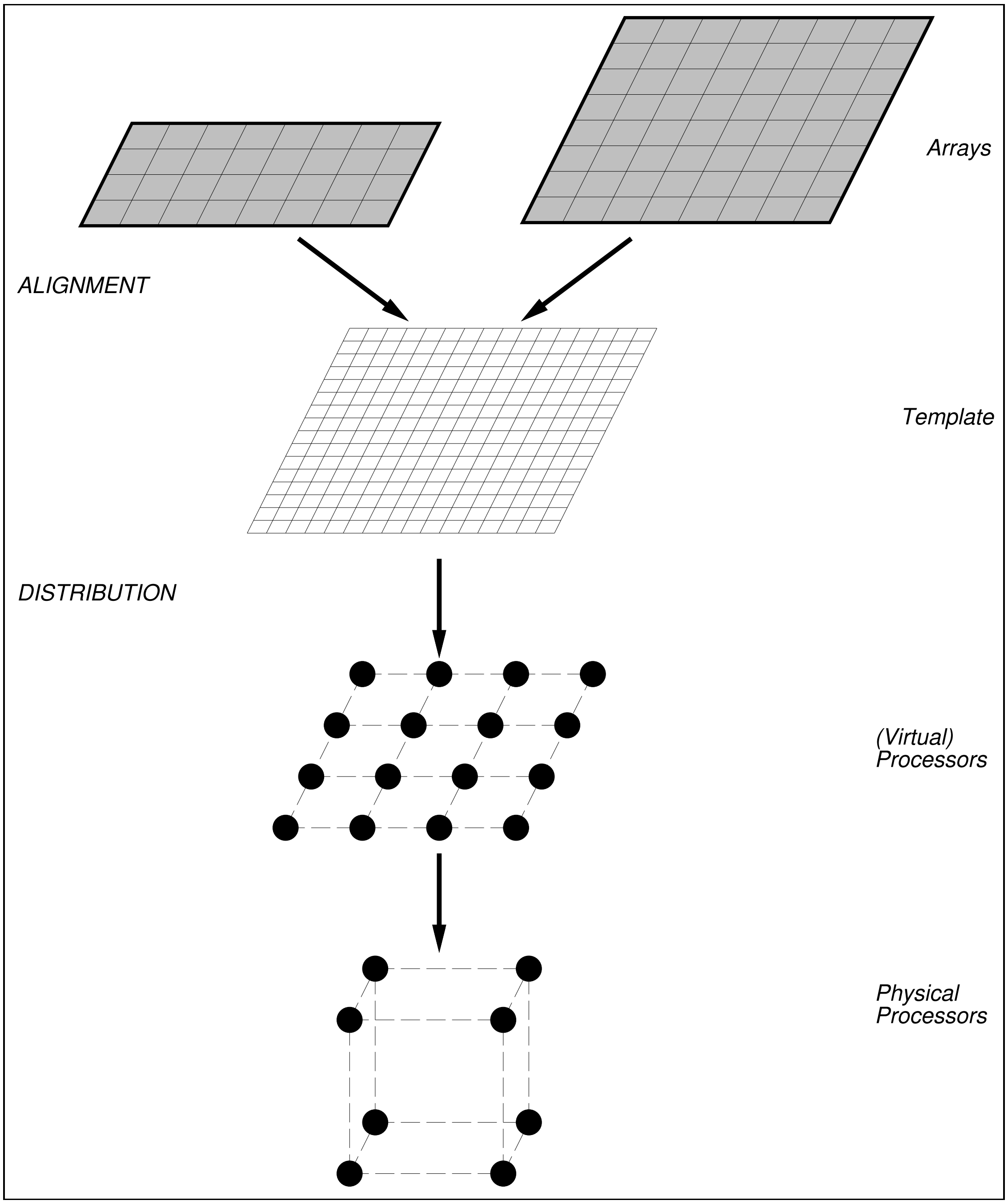

Figure 2: HPF data alignment and distribution model

Directives may also optionally be combined. A flavour of the proposed HPF directive and language syntax follows.

3.4.1 Directive syntax

The TEMPLATE directive (equivalent to the Fortran-D DECOMPOSITION) declares an abstract space of indexed positions:

```
!HPF$ TEMPLATE grid (2*N+1,2*N+1)
```

The PROCESSORS directive declares virtual processor arrangements, with the run-time constant N$PROC refering to the total number of processors used to execute the program:

```
      PARAMETER (N=SQRT(N$PROC), M=N$PROC/N)
!HPF$ PROCESSORS P (N,M)
```

Different views of the same physical processor arrangement may be specified by different specifications in separate program units. There is also a need for an implementation-dependent directive or environment variable to specify at compile-time (or more specifically, at link-time) the number of processors to be used. This would allow straightforward selection of appropriate numbers of processors when compiling/running the same code on (fully-populated) Cray Y-MPs or Connection Machines.

The ALIGN directive is used to map data objects onto elements of a template, such that operations on aligned data objects are likely to be more efficient:

```
      REAL X(:,:), Y(:,:), Z(:,:), W(:,:)
!HPF$ ALIGN X WITH grid
!HPF$ ALIGN X (I,J) WITH grid (2*J+1,2*I+1)
!HPF$ ALIGN X (:,*) WITH grid (*,:)
!HPF$ ALIGN WITH X:: Y, Z
```

Arrays which have been previously declared to be REALIGNABLE may dynamically be realigned by an executable REALIGN statement:

```
!HPF$ REALIGNABLE Z
!HPF$ REALIGN Z WITH W
```

Realignment at subroutine boundaries is therefore possible.

The DISTRIBUTE directive specifies a mapping of abstract template elements to processors in a processor arrangement.

```
!HPF$ DISTRIBUTE grid (BLOCK,*,BLOCK) ONTO P
!HPF$ DISTRIBUTE ONTO Q:: /cause/
```

Distributions may either be BLOCK or CYCLIC, with an optional stride.

A template marked as REDISTRIBUTABLE may dynamically be redistributed at any time by an executable REDISTRIBUTE statement:

```
!HPF$ REDISTRIBUTABLE:: A
!HPF$ REDISTRIBUTE A ONTO P
```

More than one attribute may be combined in a single directive, in the style of Fortran-90 (or Vienna Fortran) attribute specification statements:

```
      REAL A (100,100)
!HPF$ DISTRIBUTE (BLOCK,BLOCK) ONTO P, REDISTRIBUTABLE :: A
```

```
      INTEGER work (100,43,100)
!HPF$ ALIGN work (:,*,:) WITH A
```

Several parallel indexing constructs (confusingly also referred to as “parallel loops”, which incorrectly imply a sequentialisation of control flow) are likely to be included in HPF. The single-statement `FORALL` contains the semantics defined in Fortran-8X and used in CM-Fortran, but dropped from Fortran-90. Only assignments (without ‘unexpected’ side-effects) are allowed in the `FORALL` body.

Proposals also exist for a shorthand form of multiple single-statement `FORALL`, known as block `FORALL`, which conceptually requires a synchronisation and/or communication phase after each statement (SIMD semantics), and handling `IF` statements within a `FORALL`, defined as a series of masked single-statement `FORALL`s (and having a direct analogue of the `WHERE` semantics of Fortran-90). In addition to the `FORALL` constructs, the `INDEPENDENT` directive for `FORALL`s and `DO`-loops would assert that all iterations were independent.

3.4.2 Proposal comments

There is some doubt about the benefit of the level of abstraction, but it has supposedly been provided to allow an easy way to group related arrays that should always be aligned and distributed identically. Zima has said that programmers naturally think in terms of arrays, and that therefore the intermediate template stage in HPF is both unnecessary and unnatural. Explicit template declarations are thought to be rarely necessary. Use of the name ‘template’ itself is still subject to confirmation.

The `FORALL` statement is currently the only proposed language extension (i.e., an actual new statement, rather than a compiler directive). Many people are uncomfortable with the SIMD semantics for the block `FORALL`s, even though they are seen as useful for many problems. No firm proposals for other statements nested in a `FORALL`, such as `DO` or `WHILE`, have been made yet.

`COMMON` blocks are provisionally to be treated as a one-dimensional array whose elements are storage units. A programmer could distribute it if and only if no object in the `COMMON` block was explicitly aligned.

HPF will be unable to allow Fortran-90 programs to rely on sequence and storage association (i.e., the mapping of multi-dimensional arrays onto a linear memory model using column-major ordering, and `EQUIVALENCE` statements) since there is no linear address space in distributed-memory machines. In fact, association will be supported, but variables and `COMMON` blocks treated in that way will only have limited distributability.

Fortran-90 pointers can cause serious data redistribution problems, and it has been suggested that these will be restricted in their use.

Compilers may well be given the latitude to ignore any distribution declarations or statements if they can deduce a better way. It has, however, been left to the programmer to ensure that specifications are correct, with there being no possibility of ‘High-Performance Fortran’ becoming ‘Right-Answer Fortran’.

There is uncertainty over whether parallel I/O is something which HPF should address, with opinions to the effect that Fortran already supports more I/O than any other

```
      PARAMETER (N=   , M=   )
      PARAMETER (NP=   , MP=   )

!HPF$ PROCESSORS P (NP,MP)
!HPF$ TEMPLATE   GRID (2*N+1,2*M+1)
!HPF$ DISTRIBUTE GRID (BLOCK,BLOCK) ONTO P

      DOUBLE PRECISION  U (0:N,1:M),    V(1:N, 0:M), P(1:N,1:M), &
                        DU (1:N-1,1:M), DV(1:N,1:M-1)

!HPF$ ALIGN  U (I,J) WITH GRID (2*I+1,2*J  )
!HPF$ ALIGN  V (I,J) WITH GRID (2*I  ,2*J+1)
!HPF$ ALIGN  P (I,J) WITH GRID (2*I  ,2*J  )
!HPF$ ALIGN DU (I,J) WITH GRID (2*I+1,2*J  )
!HPF$ ALIGN DV (I,J) WITH GRID (2*I  ,2*J+1)

!HPF$ ON GRID (2:2*N:2, 2:2*M:2)
      P (1:N,1:M) = U (1:N,1:M) - U (0:N-1,1:M) &
                  + V (1:N,1:M) - V (1:N,0:M-1)

!HPF$ ON GRID (3:2*N-1:2, 2:2*M:2)
      DU (1:N-1,1:M) = P (2:N,1:N) - P (1:N-1,1:M)

!HPF$ ON GRID (2:2*N:2, 3:2*M-1:2)
      DV (1:N,1:M-1) = P (1:N,2:M) - P (1:N,1:M-1)
```

Figure 3: Example of Fortran-90 code fragment with HPF directives

language except perhaps COBOL, and that it certainly has no rôle in the language definition process.

Further MIMD support is unclear, although there is likely to be an `ON` clause which would specify which processors should execute particular statements.

An example of a code fragment containing preliminary HPF directives, presented by Clemens-August Thole in [21], is shown in Figure 3.

3.5 Implementation details

3.5.1 Communication between processors

Communication occurs when operands are on a different processor from the data-point to which an operator is assigning. The philosophy is not that data-points are fetched (as in the PRAM model), rather the compiler creates the send and receive calls that are required in order to get the data to the assignment. Blocks of decomposed arrays are allocated with overlap areas into which their edges are received. Unfortunately the send and receive calls naturally end up in the middle of the loop, and correspond to sending, waiting for and receiving lots of small messages. In practice, code transformations can be made which optimise the performance. Vienna Fortran has a unified framework within which all the optimisations are described. This is based on sets, and is a little obscure. It is equivalent to the Fortran-D transformations which are as follows:

message vectorisation sending the edge of a block of a distribution as a single communication not a series of individual data-points,

message coalescence sending only once data points that are read more than once between being assigned,

message aggregation sending the edges of blocks of more than one array in the same packet,

latency hiding by doing the bit in the middle of the block while waiting for communications, and

non-blocking sends and receives.

3.5.2 Additional and future functionality in the RTE

As implied in the description of rôle of the HPF compiler, there is actually a run time executive (RTE) which is inserted by the compiler in order for the code to work on the parallel computer. Both Fortran-D and Vienna Fortran provide basic RTEs for this purpose.

Where distribution statements are executable statements, decomposition is not known until execution time so the compiler does not insert code which specifies message destinations; rather it specifies a function which will be used to compute where to send messages (given data put in place by the decomposition calls). Static distribution of arrays obviates this requirement and makes for more efficient code.

Vienna Fortran does not efficiently handle run-time dependencies (i.e., indirections in the middle of a loop which cannot be pre-analysed by the compiler), but it can handle loops with run-time defined data access patterns. It works by computing at run-time the access patterns within the loop, and then generating code to perform the necessary message transfers. Data access analysis and code generation are known as the *inspector* phase, and it's subsequent actual execution is known as the *executor* phase. It is hoped that the number of inspections can be minimised by the recognition of loop invariance. This is not expected to be implemented in the initial release of HPF.

Saltz and Fox are working on extensions to HPF which allow more irregular applications, but require a more sophisticated RTE. The PARTI environment (Parallel Automated Run-Time Toolkit at ICASE, developed by Joel Saltz and his group) offers a lot of this functionality. [3, 15] For example, PARTI operates on loops within which data access patterns imply dependencies which cannot be pre-analysed by the compiler. It again makes use of inspector and executor phases. This time, the inspector phase computes the dependencies implicit in the access patterns within the loop and, whilst respecting the dependencies, optimises each processor's scheduling of computation and transfers of messages. Only an approximate solution to the optimisation is possible, and even then with considerable overheads. It is therefore even more important to minimise the number of inspections.

It should be emphasised that early HPF compilers will only have a very basic functionality, and it may be several years before the majority of these sophisticated features appear in production compilers.

3.6 Support environment

It seems likely that the abstraction will dramatically break when it comes to attempt to debug the HPF program. (This view seems to be shared by Ken Kennedy.) Debugging will often have to be done on the transformed code — i.e., the code with the restructured loops, the message-passing calls, etc — since this is the *only* executable code. Since the developer of the HPF program will not have encountered this code before, never mind not having generated it in the first place, there are considerable difficulties presented in having to 'maintain' this code.

HPF debuggers will most certainly exist, and be provided by various manufacturers, but their effectiveness is seriously in question. Possible developments in parallel debuggers, like some of those considered by TMC, may well alleviate some of this situation, by providing concise 'debugging pyramids' for the native code in a window alongside the HPF code.

Not only will it be necessary to debug message passing code it will also be necessary to write it, if the program contains sections which are not data-parallel. A good example is inserting long chain polymers into Cellular Automaton Lattice Gases. It would be natural to use data-parallel code to handle the update of the cellular automaton lattice-gas, but operations on the polymer do not naturally fit the data-parallel model. The two substances interact so it will be necessary to get access to the mapping function for the data-parallel arrays. HPF will let you do this, but it is conceptually rather difficult, the more so because the mapping is hidden by the data-parallel abstraction.

EPCC has had experience with CAPE in this area. [11] has a more complete description of these two problems and concludes that the data-parallel abstraction is ineffective.

4 What are the vendor attitudes?

Thinking Machines appear to be very satisfied at the moment, since they have a "high-performance Fortran" compiler (called `cmf`) for their CM-2, in which the HPF directives are essentially redundant, but they're watching with interest and taking an active rôle in developments for the future.

Intel SSD appear to have been wrong-footed by their Mach developments, and have not been able to adequately exploit Fortran-90 developments which have proven to be an exceptional wind-fall for manufacturers of competing data-parallel computers. Like other vendors of MIMD computers, Fortran-90 has left them side-lined, therefore they are very keen to produce something which will allow them to at least bring them back into the game, and with the future potential for further gains. Since they have no compiler group of their own, they have entered into a collaboration with DEC, to jointly produce a HPF compiler.

Cray (CRI) are committed to producing a version of HPF for their MPP product, which will access the hardware directly for efficiency. They are providing PVM as a message passing layer, although this is only really supported for porting existing applications.

Meiko are developing an HPF compiler as part of the Esprit PPPE (P^3E) project (discussed further later). Initial development will compile HPF down to PARMACS message-passing macros. They may well also be including PARTI or PARTI-style extensions to HPF at an early date. It is possible that in due course, their HPF compiler will be made to run at a level very close to their new hardware.

All vendors believe that it will be essential for them to support HPF on their systems if they are to be competitive, since customers will simply require it as a key component of any computer system. Furthermore, the port of existing programs to parallel architectures is likely to become much easier.

A number of vendor proposals have been made and are being considered by the HPFF [9, 14, 18].

5 Where's Europe?

While it is clear that Hans Zima's group in Vienna have earned the right to play an important rôle in the formulation of a universal High-Performance Fortran, the financial out-lay involved in participation is such that European involvement is likely to be minimal. Clemens-August Thole (GMD) has individually been participating to date, and may well be unable to continue to do so, although Henk Sips (TU Delft) is likely to.

A European Workshop on "High-Level Programming Models for Parallel Architectures" was used to discuss in a European context, HPFF developments in the US. The meeting, in Brussels on May 19th, followed and reported on April's HPFF meeting [21].

Coordination of the newly-established "European Working Group on HPF" (aka HPF-Europe) is being temporarily undertaken by Clemens-August Thole (as chairman) and Arthur Veen (as vice-chairman). This is intended to meet at another Brussels workshop on June 22nd will also follow the next scheduled HPFF meeting. There is also an E-mail list distributed from GMD.

5.1 European workshop participants

A partial list of parties (which should be considered representative, rather than comprehensive) currently involved in the European HPF Workshop is as follows:

Laboratories *GMD* (Clemens-August Thole, Hans-Christian Hoppe), *ITI-TNO/TU Delft* (Henk Sips), *IPVR Stuttgart* (Uwe Geuder), *KFA Jülich* (Michael Gerndt), *Ld'I Paris* (Nicolas Paris),

Universities *U Vienna* (Hans Zima, Barbara Chapman), *U Liverpool* (Steve Morgan), *U Southampton* (John Merlin),

Vendors *Meiko* (James Cownie), *ACRI* (Karen Spackman), *Cray* (Reiner Vogelsang), *IBM* (Michael Roethlisberger), *Parsytec* (Jan Petersen),

Miscellaneous *CERN* (F Carminati, Mike Metcalf), *Parallel Computing* (Arthur Veen), *NA Software* (Mike Delves), *GLOSSA* (Tom Lake), *ACE* (Willem Wakker), *ACSET* (Jean-Eric Waroquier), *NAG* (Sven Hammarling).

Lyndon Clarke also attended, representing EPCC.

5.2 The European rôle

It is not seen as useful for European effort to duplicate that already under way in the United States. Any modifications which might be desirable to HPF should be presented to the HPFF after prior consideration at a European level, which is seen as providing additional weight to such proposals. In practice, the tight timetable which HPFF are working to is likely to mean that little already decided may be reversed, and European opportunities to incorporate improvements — using expertise which it is recognised that Europeans could usefully contribute to the American proposals — may well be very limited.

Deficiencies in the existing HPF proposal are identified in the areas of *MIMD support* and *integration of message-passing*. In the areas of *parallel I/O* and the *treatment of subroutine boundaries* the Vienna Fortran route is accepted as a reasonable starting point for consideration.

Existing European ESPRIT projects, targeting HPF implementations in a different context, are seen as important for the European parallel computer industry. These are PPPE (set up around Meiko) and PREPARE (around ACE/Parallel Computing, and involving Parsytec): PPPE (Portable Parallel Programming Environment) will provide an extensive HPF interface (including a profiler and debugger) producing PARMACS calls for message-passing on a range of parallel platforms, while the similar PREPARE project also is intended to produce a parallelising compiler (rather than a translator), specifically targetted at the T9000.

References

[1] Siegfried Benkner, Barbara M Chapman, and Hans P Zima. 'Vienna Fortran-90'. In Robert Voigt and Joel Saltz, editors, *Proceedings of the Scalable High-Performance Computing Conference*, pages 51–59. IEEE Computer Society Press, April 1992. (Williamsburg, VA, April 26–29).

[2] Barbara M Chapman, Piyush Mehrotra, and Hans P Zima. 'Programming in Vienna Fortran'. ICASE Report 92–9, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23665, USA, March 1992. (Also available from Vienna as ACPC/TR 92-3).

[3] Raja Das, Joel Saltz, and Harry Berryman. 'A Manual for PARTI Runtime Primitives'. ICASE Interim Report 17, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23665, USA, May 1991.

[4] J Ferrante, K J Ottenstein, and J D Warren. 'The Program Dependence Graph and its Use in Optimisation'. *ACM Transactions on Programming Languages and Systems*, 9(3), 1987.

[5] G C Fox, M A Johnson, G A Lyzenga, S W Otto, J K Salmon, and D Walker. *Solving Scientific Problems on Concurrent Processors*. Prentice Hall, New Jersey, 1988.

[6] Geoffrey Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, and Min-You Wu. 'Fortran D Language Specification'. Technical report, Center for Research on Parallel Computations, Rice University, Houston, TX-77251, USA, January 1992. Earlier version previously available as CRPC-TR90079. [Document source available via anonymous FTP from titan.rice.edu:public/HPFF].

[7] Ken Kennedy. 'Compiler Support for Programming Regular Problems in Fortran-D'. Material for 'Compilers for Scalable Architectures' tutorial presented at the 'Scalable High-Performance Computing Conference' (Williamsburg, VA, April 1992), April 1992.

[8] Chuck Koelbel. 'HPFF April 22nd (Dallas) Meeting Notes'. Distributed electronically on USENET, May 1992. [Document source obtained via anonymous FTP from titan.rice.edu:public/HPFF].

[9] David B Loveman. 'Digital Equipment Corporation High Performance Fortran Language Specification Proposals'. Presented to HPFF and distributed electronically, January 1992. [Document source obtained via anonymous FTP from titan.rice.edu:public/HPFF].

[10] Michael Metcalf and John Reid. *Fortran-90 Explained*. Oxford Science Publications, 1990.

[11] M G Norman, J R Henderson, I G Main, and D J Wallace. 'The Use of the CAPE Environment in the Simulation of Rock Fracturing'. *Concurrency Practice and Experience*, 3(6):687–698, 1991.

[12] Michael G Norman and Peter Thanisch. 'Models of Machines and Modules for Mapping so as to Minimise Makespan in Multicomputers'. Technical Report EPCC-TR-91-14, Edinburgh Parallel Computing Centre, 1991.

[13] D A Padua and M J Wolfe. 'Advanced Compiler Optimisations for Supercomputers'. *Comm. ACM*, 29(12), 1986.

[14] Douglas M Pase, Tom MacDonald, and Andrew Meltzer. 'MPP Fortran Programming Model'. Distributed electronically, March 1992. [Document source obtained via anonymous FTP from titan.rice.edu:public/HPFF].

[15] Joel Saltz. 'Runtime Compilation'. Material for 'Compilers for Scalable Architectures' tutorial presented at the 'Scalable High-Performance Computing Conference' (Williamsburg, VA, April 1992), April 1992.

[16] Mark Sawyer. 'A Summary of Fortran-90'. Technical Report EPCC-TN92-04, EPCC NSG, May 1992. Version 1.2.

[17] Guy L Steele Jr. 'Proposal for alignment and distribution directives in HPF'. Made available electronically for distribution via anonymous FTP from titan.rice.edu:public/HPFF, April 1992.

[18] Guy L Steele Jr. 'Thinking Machines Corporation Proposals for Amending High Performance Fortran'. Distributed as electronic text, January 1992. [Document source obtained via anonymous FTP from titan.rice.edu:public/HPFF].

[19] Thinking Machines Corporation, Cambridge, MA, USA. *CM-5 Technical Summary*, October 1991.

[20] Thinking Machines Corporation, Cambridge, MA, USA. *CM Fortran Reference Manual*, July 1991. Version 1.1.

[21] Clemens-August Thole. 'High Performance Fortran Forum: Aims, Organisation and Current Status of the Proposal'. Presented at the European Workshop on High-Level Programming Models for Parallel Architectures (Brussels, May 19th, 1992), May 1992. (copies of overheads).

[22] Joel Williamson. 'Convex Views on Fortran-D and HPF'. Presented to HPFF and distributed electronically., January 1992. (copy of overhead slides).

[23] Hans Zima, Peter Brezany, Barbara Chapman, Piyush Mehrotra, and Andreas Schwald. 'Vienna Fortran – A Language Specification: Version 1.1'. Technical Report ACPC/TR 92-4, Austrian Center for Parallel Computation, March 1992.