Preliminary Report on HPF2

L.M. Delves
N.A. Software Ltd

Deliverable:	D3.2a
Date:	December 1996
Status:	draft
Confidentiality:	public
Responsible:	NA Software
Revised:	January/7/1997

The **HPF+** Consortium:
AVL GesmbH - Austria
European Centre for Medium-Range Weather Forecasts (ECMWF) - England
Engineering System International SA (ESI) - France
NA Software Ltd (NAS) - England
University of Pavia - Italy
University of Vienna - Austria

1 Introduction

HPF2 is an amended and extended version of HPFV1.1. At its meeting in early December, the language was finalised for public comment. This preliminary Report relates to the December version of the language, although few changes are currently expected to emerge from the commentary period.

In this document we provide an overview of HPF2 and an outline comparison with the initial version of the HPF+ language (Deliverable D2.1a). A final comparison will be given later (D3.2b: PM24). The comparison is based on a study of the current HPF2 draft standard; on discussions with HPFF members carried out during attendance at HPFF meetings in September (LMD) and December (DW); and on discussions within the HPF+ (Siegfried Benckner VCPC) and Pharos (Thomas Brandes GMD) projects.

2 Overview of HPF2

HPF2 provides a reasonably graceful upgrade to HPF1.1:

- Nearly all of the language content of 1.1 has been retained.
- Some facilities have been changed slightly (so that HPF2 is *not* a superset of HPF1.1). But the changes are restricted to those features of 1.1 which have not in practice been implemented by vendors; few if any existing users will notice the changes.

  The removed features are:

  - Sequential arrays may no longer be explicitly mapped.
  - An explicit interface is always required for any call needing remapping of one or more arguments.
  - `INHERIT` and `DISTRIBUTE` may not both be specified for a dummy argument.

  These restrictions were already imposed by the N.A. Software HPFPlus compiler.
- HPF2 is described as based on Fortran95 rather than Fortran90. So, `FORALL, PURE`, and extensions to `MINLOC, MAXLOC` are no longer described within HPF2 since they have been adopted within Fortran95. To the user of course, this is irrelevant: the features are still available.
- Subset HPF is no longer recognised. Instead, the language has been split differently:

  1. Base HPF2 contains all of Fortran95, plus those HPF1.1 features which have been most commonly implemented.
  2. An Approved Extensions Annexe describes many more features which are *not* part of Base HPF2, but which provide a standard syntax and semantics for those vendors wishing to implement them.

The HPFF hopes that all vendors will implement Base HPF2 as quickly as possible. They expect that different vendors will implement different selections from the Approved Extensions (there are so many that it is *most* unlikely that all will be implemented in the foreseeable future).

2.1 Base HPF2

Compared with HPF1.1, Base HPF2 has additions and subtractions:

Additions : A `REDUCTION` clause has been added to the `INDEPENDENT, NEW` feature to help parallelise existing loops without major rewriting.

Minor extensions to the intrinsic library have been made.

The syntax for extrinsic routines has been extended.

Subtractions : Dynamic mappings (`REDISTRIBUTE, REALIGN`) are not part of Base HPF2 but are in the Approved Extensions. We strongly approve of this move.

2.2 Approved Extensions

These include:

Extensions for Data Mapping :

- Mappings to processor subsets to assist MIMD description and load balancing
- Mapped Derived Type components (arrays of derived types may already be mapped within Base HPF2; mapped arrays may not have mapped components).
- Irregular distributions: `GEN_BLOCK, INDIRECT`.
- `REALIGN, REDISTRIBUTE`
- A `RANGE` directive to reduce combinatorial explosion with `INHERIT`.
- A `SHADOW` directive to specify overlap regions

Intrinsic and Library procedures : have been extended

Asynchronous I/O : but no parallel I/O

Extrinsic Interfaces : Extensions to these.

ON directive : to specify where to compute

RESIDENT directive : An assertion that no communication outside a specified set of processors, will be required.

Tasking Directive : to specify a simple form of MIMD parallelism with separate tasks being carried out on disjoint processor subsets (but with semantics such that no multitasking is ever needed on a given virtual processor).

3 HPF2 Language: Selected Examples

In this section we illustrate some of the parts of HPF2 which we expect to be the most important in practice; but start with an important simplification over HPF1.1.

3.1 Remapping at Procedure Boundaries

In HPF1.1, `descriptive` and `prescriptive` mappings are defined. A descriptive mapping provided a promise that the actual arguments would match the specification in the procedure so that no remapping would be needed. An implicit interface was then allowable under some circumstances. Now, HPF2 removes the distinction between these two classes of procedure argument (though both are kept syntactically for backwards compatibility).

The HPFPlus compiler already implements the HPF2 rules (as does that of DEC) by insisting on an explicit interface whenever any argument is mapped.

3.1.1 Example

```
Program EX1
interface
  subroutine cyclic_map(a,b)
  real,dimension(:)::a,b
  !hpf$ distribute(cyclic)::a,b
  end subroutine
end interface
real, dimension(1000):: x,y
!hpf$ distribute(block)::x,y
....
call cyclic_map(x,y)
...
end program
```

In this example, the explicit interface is mandatory because the actual arguments must be remapped at the call.

3.2 EXTRINSIC program units

The HPF2 extended syntax is:

```
EXTRINSIC(LANGUAGE=.., &
          MODEL=.., &
          EXTERNAL_NAME=..) [TYPE] FUNCTION/SUBROUTINE(...)
```

with three models being supported: `MODEL = 'GLOBAL', 'LOCAL', 'SERIAL'` and languages `LANGUAGE = 'HPF', 'F77', 'FORTRAN', 'C'`.

There are abbreviations for backwards compatibility:

Abbreviation	LANGUAGE	MODEL
HPF	HPF	GLOBAL
HPF_LOCAL	HPF	LOCAL
HPF_SERIAL	HPF	SERIAL
F77_LOCAL	F77	LOCAL
FORTRAN_LOCAL	FORTRAN	LOCAL
C_LOCAL	C	LOCAL

HPFPlus currently implements `HPF_SERIAL` (and `HPF`). Note that the syntax used is *not* that of a directive but of a first class language feature. However, it seems possible that Fortran will adopt the syntax to formalise the calling of C and other alien language routines.

3.3 ON Directive

This is an Approved Extension, with syntax exemplified by:

```
!hpf$ ON HOME(a(2:4))
 statement
```

or in block form:

```
!hpf$ ON HOME(a(2:4)) BEGIN
   block
!hpf$ END [ON]
```

where a block will typically be a loop. The semantics are as follows:

- Restricts the set of active processors, which is specified by the HOME clause. This can mention a program variable, a template, or a processor arrangement.
- The ON clause does *not* assert anything about data residency; so communication with other processors may be needed.
- ON clauses may be nested provided the nesting makes sense.
- ON can be applied to subprogram invocations:

```
!hpf$ processors p(10)
!hpf$ distribute x(block) onto p
!hpf$ on (p(4:7))
   call collect(x)
```

  will cause `X` to be mapped to P(4:7) before the call; any local variables used in `collect` will also be stored on P(4:7).

3.4 RESIDENT

This directive asserts that the data accessed by the computations in the scope of the directive, is local to the set of processors specified. The compiler can use this information to avoid generating communication tests, or to simplify address calculations.

The syntax is exemplified by:

```
!hpf$ on home (z(i)), resident (x,y, record(i))
x(i) = y(i+1) + record(i)%field1 + record(i+1)%field2
```

This directive applies to one statement. It asserts that `x(i)`, `y(i+1)`, and `record(i)` all lie on the same processor as `z(i)`. But `record(i+1)` may be on a different processor.

More commonly, the assertion will be used to label a loop:

```
real, dimension(1000):: x,y
integer,dimension(1000)::ix1,ix2
!hpf$ processors p(np)
!hpf$ distribute(block) onto p:: x,y,ix,iy
....
!hpf$ independent
do i = 1,n
  !hpf$ on home(x(i)), resident(y(ix(i)))
  x(i) = y(ix(i)) - y(iy(i))
end do
```

3.5 TASK_REGION directive

This provides a simple form of heterogeneous task parallelism which we expect to be of substantial importance in porting applications. Here is a code fragment which illustrates it:

```
!hpf$ processors procs(8)
....
done1 = .false.
do while(.true.)
! First task on first four processors:
!hpf$ ON (procs(1:4)) BEGIN, RESIDENT
  read(unit=iu, end=100)a1
   call rowffts(a1)
   goto 101
   100 done1 = .true.
   101 continue
!hpf$ END ON
   IF(done1) EXIT
   a2 = a1
```

```
! Second task on processors 5-8
!hpf$ ON(procs(5:8)) BEGIN, RESIDENT
   call colffts(a2)
   write(unit=ou)a2
!hpf$ END ON
end do
!hpf$ END TASK_REGION
```

4 Comparison with HPF+

Apart from the inclusion of full Fortran95 (eg: derived types, pointers) Base HPF2 is quite close to Subset HPF1.1. The HPF+ language includes extensions aimed at improving the efficiency of description and (hopefully) compilation of Fortran codes using highly irregular data distributions. Comparison should therefore be made with Base HPF2 together with the Approved Extensions Annexe. This comparison ignores the likelihood of commercial compilers implementing specific features; we include here such comments as we can for the NAS HPFPlus compiler. The comparison is with HPF+ Release 1; the baseline is HPF V1.1.

4.1 Processor Array Subsets

HPF+ and HPF2 each address this, but HPF+ provides greater functionality.

4.2 Data Mappings

Both provide `GEN_BLOCK, INDIRECT` mappings; mapping of derived type components; mapping of pointers; and `RANGE`.

4.3 Independent Loops

Both provide reduction clauses and `ON`; HPF+ provides also `PUREST`.

4.4 Mapping of Dummy Arrays

Both provide `INHERIT, RANGE` though HPF+ differs slightly from HPF2.

4.5 SHADOW

HPF+ does not currently include the `SHADOW` directive, but substantial thought has been given to a similar construct (haloes). NAS would urge caution here: certainly we feel the `SHADOW` directive to be awkward to use, and would hope that compilers can manage without.

4.6 TASK_REGION

This is not addressed in HPF+ Version 1, but some thought has been given to the topic. We believe that:

1. Multitasking is a natural complement to the ability to address processor subsets; and is potentially a very valuable language addition;

2. The HPF2 `TASK_REGION` construct is unnecessarily restricted in providing only wholly independent tasks with no mutual synchronisation points; a richer construct would be preferable.

We hope that HPF+ will address this area in the future.

Overall, HPF+ V1 and HPF2 are quite close on both content and syntax. This is not too surprising: the developments in HPF+ have been influential in steering the HPFF development of HPF2, and in particular `GEN_BLOCK, INDIRECT` are directly attributable to University of Vienna input to HPF2.

However, the HPF2 Approved Extensions contain features (eg: tasking) not addressed in HPF+.

The definition of HPF+ V1 appeared quite early in the HPF+ project. Development of the language since then has been steered by experience with the project benchmark codes, and this has suggested further constructs; for example for assisting schedule re-use. The emphasis in the project has naturally been to pursue these rather than to follow every line explored by HPFF. We believe that this experience gained with industrial codes has been very beneficial; and look forward to extending the current comparison in due course to include Release 2 of HPF+.

4.7 Effort at N.A. Software

We are moving as rapidly as possible towards implementing full Base HPF2. Our current release, V1.2, lacks support for pointers and derived types as the major omissions from this; these holes will be filled as soon as possible. Beyond this, we see `GEN_BLOCK` and `INDEPENDENT, NEW, REDUCTION` as priorities; and would like to address a range of other features too. It is however very difficult to forecast the rate of progress in including new language features: the largely invisible but very important inclusion of additional compilation optimisations, is also of high priority.