High Performance Fortran (HPF)

John Merlin and Bryan Carpenter
University of Southampton.

Workshop on Data-parallel Languages and
Compilers,
University of Lille,
20 April 1994.

Contents

(1) Overview of HPF

Introduction

- **H P F HPF**: an informal standard
  for extensions to Fortran 90 to assist its implementation
  on parallel architectures (particularly distributed
  memory), especially for data-parallel computations.

- *N* ! Instead, specify

  HPF compiler automatically partitions data and
  inserts message-passing.

- HPF model:

  global name space;

  compiles to data parallel SPMD program;

  single execution thread functions

in `FORALL`, `INDEPENDENT` loops, 'extrinsic' procedures.

- Many extensions, including those for data distribution, are (i.e. structured comments).

  Thus HPF should be portable across the whole spectrum of architectures (SM, DM, SIMD, uniprocessor).

- Developed by the HPFF (HPF Forum), Jan 92 – May 93.
  Over 40 organisations participated. Significant European input.

  Currently HPF 1.1 development is underway.

- HPF Spec published in *S P*, 2 (1), Sept. 93.
  Available by ftp from and elsewhere.

Alignment

`TEMPLATE` declares abstract array, e.g.:

```
CHPF$ TEMPLATE  T (10),  T2 (10,10)
```

`ALIGN` declares mapping of array → template or another array, e.g.:

```
CHPF$ ALIGN  A(I)  WITH  T(I)
CHPF$ ALIGN  B(I)  WITH  T(2*I+2)
CHPF$ ALIGN  A2(I,J)  WITH  T2(J,I) ! transpose dimensions
CHPF$ ALIGN  A2(I,*)  WITH  T(I)    ! 2nd. dim 'collapsed'
CHPF$ ALIGN  A(I)  WITH  T2(I,*)    ! 'replicated'
CHPF$ ALIGN  A(I)  WITH  T2(I,2)    ! 'embedded'
```

			B_1		B_2		B_3		B_4

Also an form (`REALIGN`)—i.e. can perform re-alignment.

Distribution

`PROCESSORS` declares (abstract) processor arrays, e.g.:

```
CHPF$ PROCESSORS P (4),  P2 (4,4)
```

`DISTRIBUTE` declares mapping of template or array
→ processors, e.g.:

```
CHPF$ DISTRIBUTE  T (BLOCK) ONTO P ! distrib elems in blocks
CHPF$ DISTRIBUTE  T (CYCLIC)       ! distrib elems cyclically
CHPF$ DISTRIBUTE  T (CYCLIC (2))   ! distrib blocks of 2
                                   ! elements cyclically
CHPF$ DISTRIBUTE  T (*)            ! not distributed
CHPF$ DISTRIBUTE  T2 (BLOCK, CYCLIC) ONTO P2
```

Also an form (`REDISTRIBUTE`)—i.e. can perform
re-distribution.

Concurrent execution features

`FORALL` a data parallel 'loop', e.g.:

```
  FORALL (i = 2:9) A(i) = 0.5*(A(i-1) + A(i+1))
```

`PURE` Defined to be
side-effect free, so can be executed concurrently, e.g.
in `FORALL`.

Hence can get 'MIMD parallelism' over array elements, e.g.:

```
  PURE INTEGER FUNCTION itns (c)
    COMPLEX, INTENT (IN) :: c;   COMPLEX :: z = c
    itns = 0
    DO WHILE (ABS (z) < 2.0  .AND.  itns < 100)
      z = z*z + c
      itns = itns + 1
    ENDDO
    IF (ABS (z) < 2.0) itns = -1
  END FUNCTION itns
  ...
```

```
FORALL (i=1:m,j=1:n) mandel(i,j) = itns (CMPLX(.1*i, .1*j))
```

`INDEPENDENT` asserts that iterations of a `DO`-loop or `FORALL` are data independent, and so can be executed concurrently. E.g.:

```
CHPF$ INDEPENDENT, NEW (s)
      DO i = 1, 100
        s = b(i-1) + b(i+1)
        CALL sub (a(i), s)
      ENDDO
```

HPF intrinsics

- system inquiry functions: `NUMBER_OF_PROCESSORS ([DIM])` and `PROCESSORS_SHAPE ()`. These query the physical (not abstract) machine, and can be used in declarations.
- computational intrinsic: `ILEN (I)`;
- extensions to Fortran 90 intrinsics `MAXLOC` and `MINLOC`.

HPF standard library

Obtained from a module: `USE HPF_LIBRARY`.

- combining-scatter functions; parallel prefix/suffix functions; sorting functions; bit operation functions.
- mapping inquiry subroutines:

```
HPF_ALIGNMENT (ARRAY, [LB, UB, STRIDE, AXIS_MAP, ...])
HPF_TEMPLATE  (ARRAY, [TEMPLATE_RANK, LB, UB, ...])
HPF_DISTRIBUTION (ARRAY, [AXIS_TYPE, AXIS_INFO, ...])
```

Extrinsic procedures

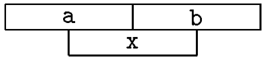
Can call non-HPF procedures ( )
from an HPF program, e.g. for message-passing, assembly or other non-Fortran language programming, etc.

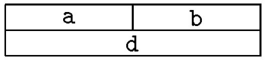
Generally not portable.

Restrictions on sequence and storage association

S : equivalencing arrays via `EQUIVALENCE` & `COMMON`:

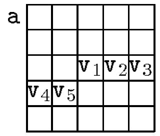
```
SUBROUTINE s1
  REAL x(100)
  COMMON /com/ a(100), b(100)
  EQUIVALENCE (x(1), a(50))
  ...
SUBROUTINE s2
  COMMON /com/ d (200)
```





S : sectioning / reshaping arrays via proc arguments:

```
  REAL a(5,5)
  CALL s ( a(2,2) )
  ...
SUBROUTINE s (v)
  REAL v(5)
```



Difficult to support this for distributed arrays in HPF,
so:

- `SEQUENCE` directive must declare arrays subject to these assocs;

- there are restrictions on distribution.

Subset HPF

An official subset of HPF has been defined for fast implementation. It comprises:

F which is (roughly) all of Fortran 77 plus ...

- non-character array syntax;
- all new Fortran 90 intrinsics;
- procedure interface blocks;
- 'cleaner' syntax features (MIL 1753), e.g. `DO WHILE`, `ENDDO`;
- lexical improvements & new declaration forms.

HPF (roughly speaking) ...

- dynamic realignment & redistribution;
- `PURE` procedures;
- `FORALL` constructs (but not statements!);
- the `HPF_LIBRARY` module;
- the `EXTRINSIC` mechanism.

(2) ‘Efficient translation of HPF for distributed memory systems’

Introduction

2 year SERC-funded project, starting 1 April 1994, in collaboration with NAG Ltd.

Objectives:

- To develop optimisation techniques for HPF, in particular:

  efficient placement of calculations;

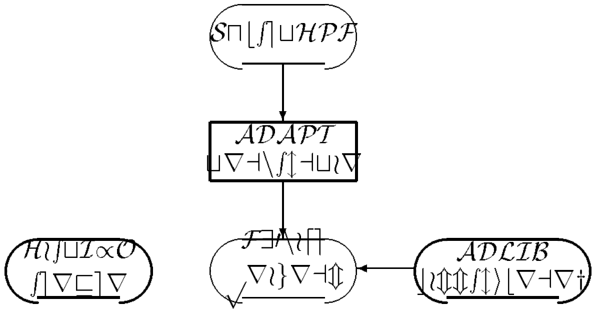
  efficient organisation of communications.

- To implement this research in a Subset HPF compilation system (i.e. translator + communications library).

Also, JISC-funded project (Jan–Dec 94) to develop runtime library for the system.

Overview of Subset HPF compilation system

Based on further development of the 'ADAPT' system, developed at Southampton over the last few years.



'ADLIB' will contain high-level, array-based comms routines (e.g. data remapping operations, gather-scatter, . . .).

Message-passing implemented in PVM (and later MPI?).

Plan initial release end of 1994; optimised version at end of project.

Efficient calculation placement

HPF defines the distribution of , but not of
.

The heuristic is widely used. However, it is often inefficient in comms, temporary storage and processor use. E.g.:

```
CHPF$ ALIGN WITH A(I) :: B(I), C(I), D(I)
      A(1) = B(9) + C(9) + D(9)      ! each RHS element
                                     ! copied to home (A(1))

CHPF$ DISTRIBUTE D1 (BLOCK), D2 (BLOCK), R (*)
      R(:) = D1(:) + D2(:)           ! D1 & D2 copied to
                                     ! all processors.
```

Also, it gives no guidance if expression is not RHS of an assignment, e.g.: output item, actual argument, `IF` condition, . . .

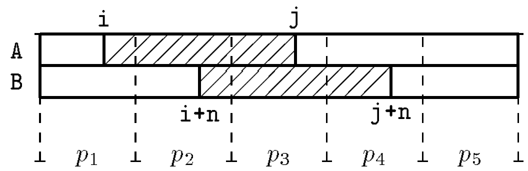
Efficient communications management

The placement of data and calculations determines what data communications are required.

Organising these comms efficiently is a complex matter. E.g:

- Consider simple shift of unknown size:

```
A(i:j) = ... + B(i+n:j+n) + ...
```



2 possible schemes:

(i) collective shift into temporary of same size as whole local segment of `B` (may involve local copying);

(ii) generate code with branches to perform most efficient comms for every case (complex control flow & tests).

- If shift is of known (small) size:

```
A(i:j) = ... + B(i+1:j+1) + ...
```

can just shift data required (without local copying), and partition the assignment into and parts.

Destination of communicated data can be:

(i) temporary array;

(ii) overlap area in `B`.

- Overlap scheme is most efficient, but requires:

  access patterns fairly local, and known statically;

  global analysis of accesses to array.

Can it be generalised?