High Performance Fortran (HPF):

The Language

Dr. Thomas Brandes

May 1995

HPF : The Language

1. Data Mapping
2. Dynamic Data Mapping
3. Data Parallelism
4. Intrinsics, HPF Library
5. Storage and Sequence Association
6. Extrinsic Procedures
7. HPF Subset

New Features in High Performance Fortran

- Directives for Data Mapping

  (ALIGN, DISTRIBUTE)

- Expressing Data Parallelism

  (FORALL, INDEPENDENT)

- Intrinsics and Library Procedures

  (PROCESSORS, ...)

. Extrinsic Procedures

  (Interface for other programming models / languages)

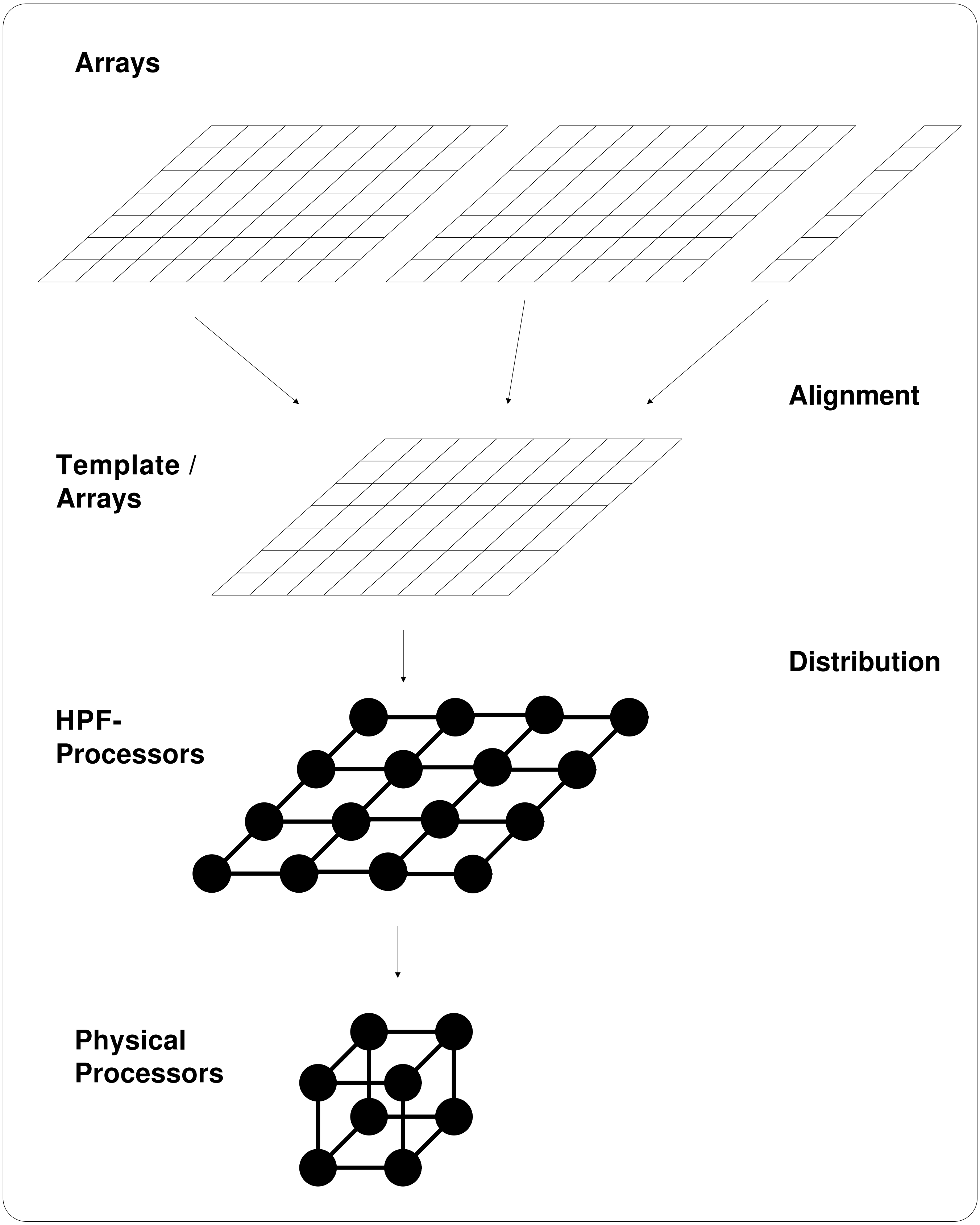
- Storage / Sequence Association

  (EQUIVALENCE, Reshaping)

- Subset High Performance Fortran

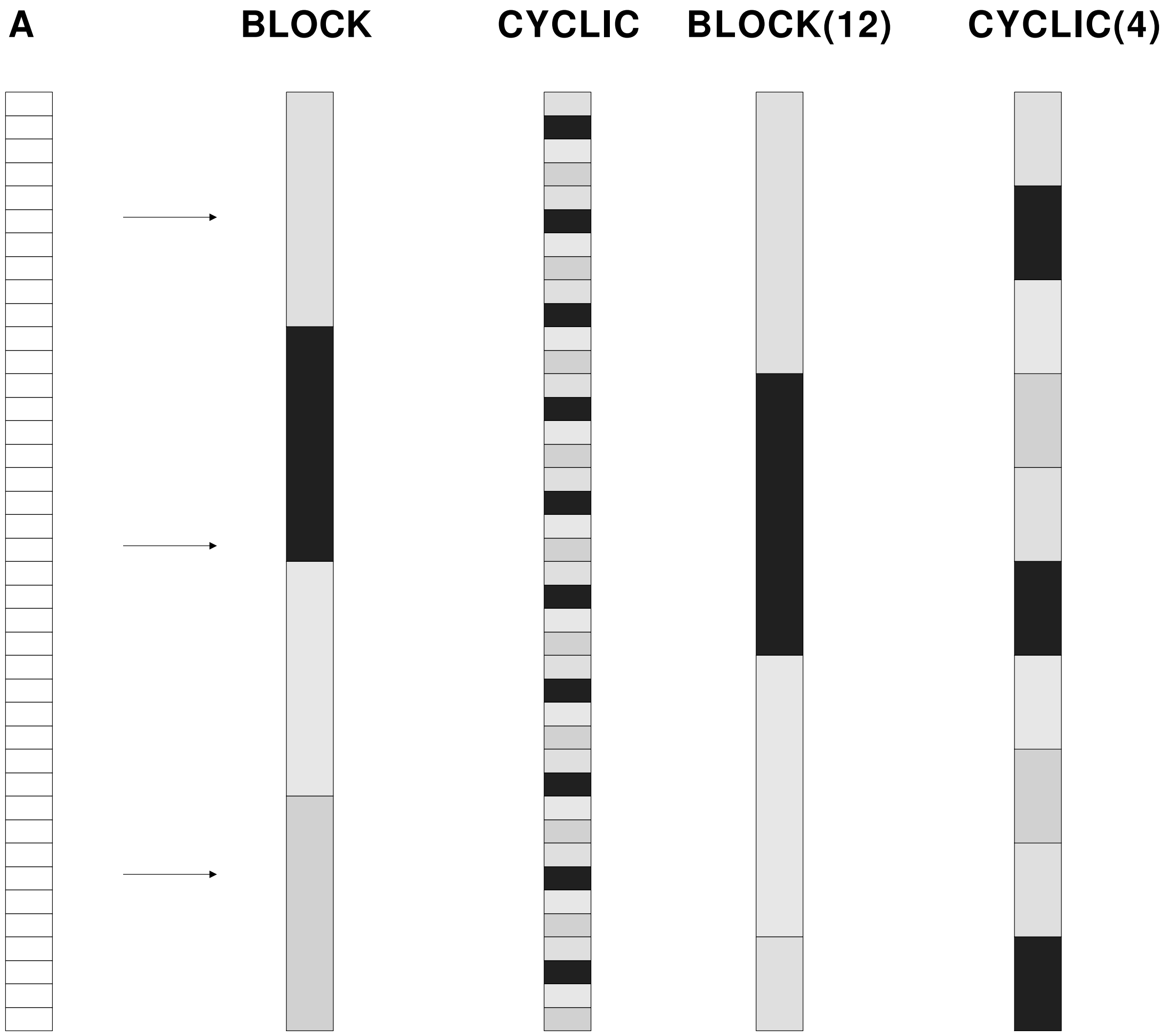
Considered: Parallel I/O

(no standardization)



1. Data Mapping

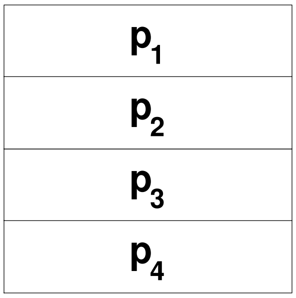
```
      REAL A(40)
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE A (...) ONTO P
```



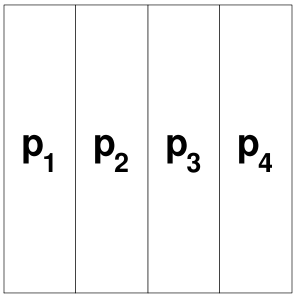
	BLOCK	CYCLIC	BLOCK(12)	CYCLIC(4)
P1	1:10	1:37:4	1:12	1;4, 17:20, 33:36
P2	11:20	2:38:4	13:24	5:8, 21:24, 37:40
P3	21:30	3:39:4	25:36	9:12, 25:28
P4	31:40	4:40:4	37:40	13:16, 29:32

HPF: Distributions (1)

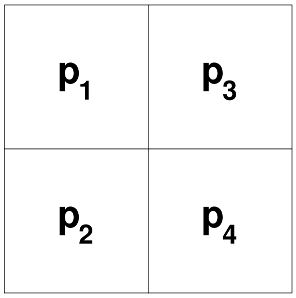
```
        REAL A(N,N)
!HPF$ PROCESSORS P(4),                    P (2,2)
!HPF$ DISTRIBUTE A (...) ONTO P
```



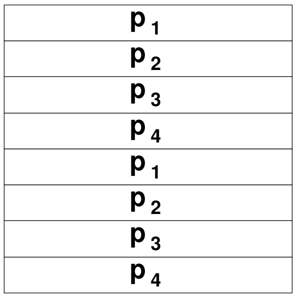
(BLOCK, *)



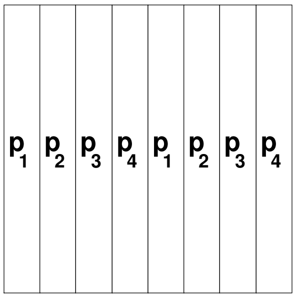
(*,BLOCK)



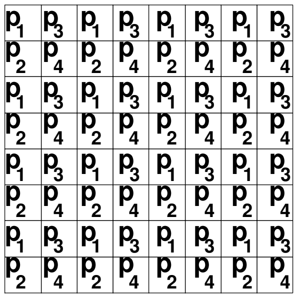
(BLOCK,BLOCK)



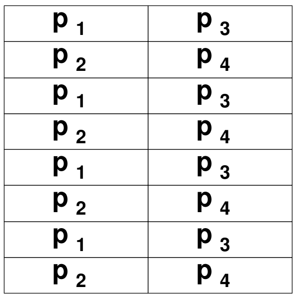
(CYCLIC,*)



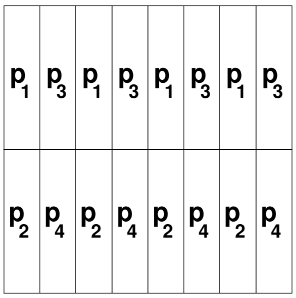
(*,CYCLIC)



(CYCLIC, CYCLIC)



(CYCLIC,BLOCK)



(BLOCK,CYCLIC)

HPF: Distributions (2)

Template

```
real U(N, N), V(N, N), W(N, N)
real Uold(N,N), Vold(N,N), Wold(N,N)
real A(3, N, N)
```

Solution 1 : all arrays are distributed

```
!HPF$ DISTRIBUTE (BLOCK, BLOCK) :: U, V, W
!HPF$ DISTRIBUTE (BLOCK, BLOCK) :: Uold, Vold, Wold
!HPF$ DISTRIBUTE (*, BLOCK, BLOCK) :: A
```

Solution 2 : all arrays are aligned to a distributed template

```
!HPF$ TEMPLATE SPACE (N, N)

!HPF$ ALIGN WITH SPACE (I,J) :: U(I,J), V(I,J), W(I,J)
!HPF$ ALIGN WITH SPACE (I,J) :: Uold(I,J), Vold(I,J), Wold(I,J)
!HPF$ ALIGN WITH SPACE (I,J) :: A(K,I,J)

!HPF$ DISTRIBUTE SPACE (BLOCK, BLOCK)
```

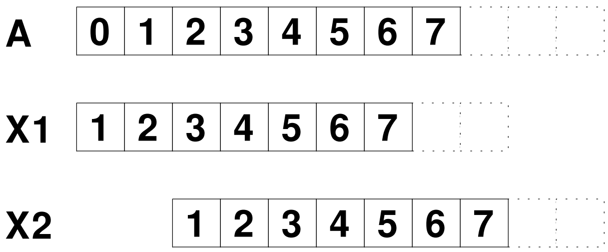
- compiler can generate better code
- easier to change the distribution of all arrays

Other Syntax:

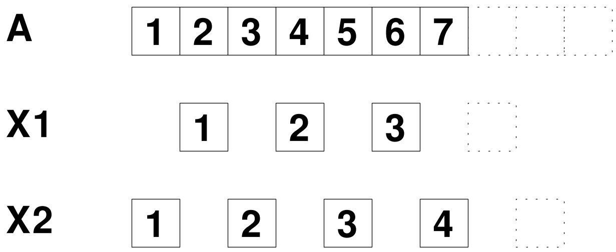
```
!HPF$ ALIGN WITH SPACE :: U, V, W, Uold, Vold, Wold
!HPF$ ALIGN WITH SPACE :: A(*,:,:)
```

Alignment of Data

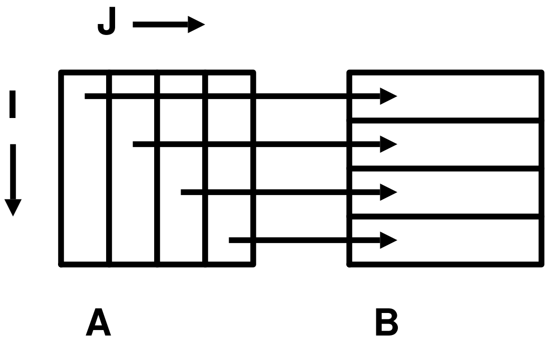
```
        real A(0:N+1), X1(N), X2(N)
c$hpf  align X1( I ) with A(I - 1)
c$hpf  align X2( I ) with A(I+1)
```




```
        real A(1:2*N), X1(N), X2(N)
c$hpf  align X1( I ) with A(2*I)
c$hpf  align X2( I ) with A(2*I - 1)
```

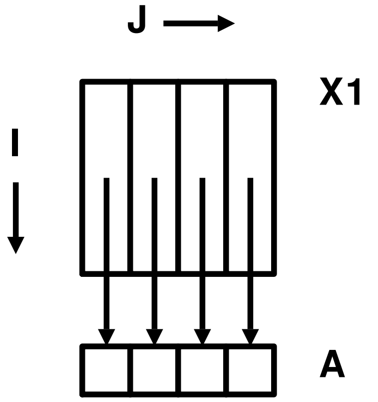



```
        real A(N,N), B(N,N)
c$hpf  align A(I,J) with B(J,I)
```

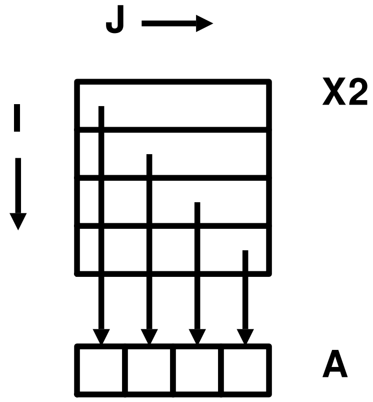


HPF: Alignment (1)

Array Collapse

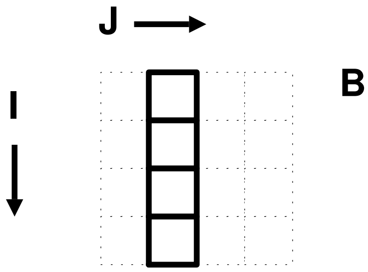


ALIGN X1(I,J) with A(J)

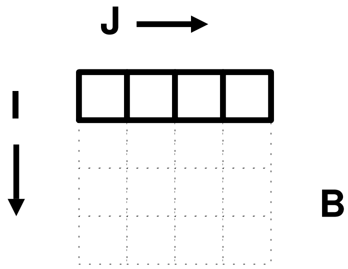


ALIGN X2(I,J) with A(I)

Array Embedding

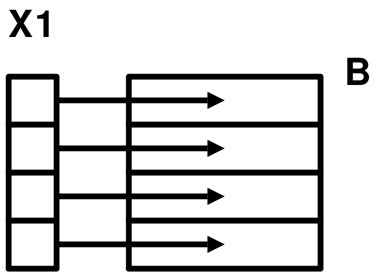


ALIGN X1(I) with B(I,2)

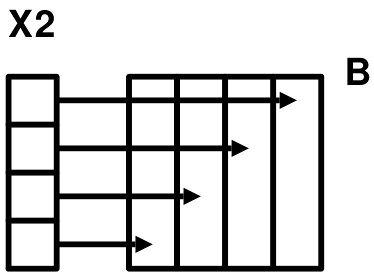


ALIGN X2(J) with B(1,J)

Array Replication



ALIGN X1(I) with B(I,*)



ALIGN X2(J) with B(*,J)

HPF: Alignment (2)

adapted from PARKBENCH, Low Level HPF Compiler Benchmarks, Kernel AA

```
          program AA

          parameter (N = 100000)
          real PX (13,N), Q
          integer I

!HPF$  PROCESSORS P(8)
!HPF$ TEMPLATE D(N)
!HPF$ ALIGN PX (*,I) WITH D(I)
!HPF$ DISTRIBUTE D(BLOCK) ONTO P


          FORALL (I=1:N)
          PX (1,I) =  10.0 * PX (13,I) + 0.2 * PX(12,i) +     &
                       2.6 * PX (11,I) + 0.25 * PX(10,I) +   &
                      0.11 * PX(9,I)   + 2.5   * PX(8,I) +    &
                      1.01 * PX(7,I) +           PX(3,I)  +    &
                       0.5 * (PX(5,I) + PX(6,I))
          stop
          end
```

Example : Data Mapping

Prescriptive mapping directives for dummy arguments

```
        subroutine UPDATE (U, V, N)
        real U(N, N), V(N, N)
        integer N
!HPF$ DISTRIBUTE (BLOCK, BLOCK) :: U, V
        ...
        U = (cshift (V, 1, -1) + cshift (V, 1, 1) + &
              cshift (V, 2, -1) + cshift (V, 2, 1) ) * 0.25
        ...
        end
```

- directives like mapping directives for non-dummy variables
- compiler can rely on the declared data mapping
- implicit remapping at surboutine boundaries possible

```
        real A(N,N), B(N,N)
        ....
!HPF$ DISTRIBUTE (CYCLIC,*) :: A, B
        ...
        call UPDATE (A, B, N)
        ...
```

Note : INTENT(IN), INTENT(OUT) allows optimizations

Implicit Redistribution at Subroutine Boundaries

Inherited mapping for dummy arguments

```
          subroutine UPDATE (U, V, N)
          real U(N, N), V(N, N)
          integer N
   !HPF$ INHERIT :: U, V
          ...
          U = (cshift (V, 1, -1) + cshift (V, 1, 1) + &
                cshift (V, 2, -1) + cshift (V, 2, 1) ) * 0.25
          ...
          end
```

- no implicit remapping
- code must be able to handle every distribution of the argument

```
         real A(N,N), B(N,N)
         ....
   !HPF$ DISTRIBUTE (CYCLIC,*) :: A, B
         ...
         call UPDATE (A, B, N)
         ...
```

Inherited Data Mapping

Descriptive mapping directives for dummy arguments

```
          subroutine UPDATE (U, V, N)
          real U(N, N), V(N, N)
          integer N
!HPF$ ALIGN U(:,:) WITH V(:,:)
          ...
          U = (cshift (V, 1, -1) + cshift (V, 1, 1) + &
                cshift (V, 2, -1) + cshift (V, 2, 1) ) * 0.25
          ...
          end
```

- there will be no remapping of the actual argument
- but directives assert that the actual argument will have a particular alignment or distribution

```
          real A(N,N), B(N,N)
          ....
!HPF$ DISTRIBUTE (CYCLIC,*) :: A, B
          ...
          call UPDATE (A, B, N)
          ...
```

Descriptive Data Mapping

Diretvies for dynamic redistribution

```
        real A(N,N), B(N,N)
        ....
!HPF$ DISTRIBUTE (CYCLIC,*) :: A, B
!HPF$ DYNAMIC :: A, B
        ...
!HPF$ REDISTRIBUTE (BLOCK, BLOCK) :: U, V
        ...
        U = (cshift (V, 1, -1) + cshift (V, 1, 1) + &
              cshift (V, 2, -1) + cshift (V, 2, 1) ) * 0.25
        ...
        end
```

Dynamic Data Mapping

How to specify data parallelism ?

a) Array Operations

```
A (2:N-1) = ( A(1:N-2) + A(3:N) ) * 0.5

A = ( CSHIFT (A, DIM=1, POS=1) + CSHIFT (A, 1, -1) ) * 0.5

B(1:N,1:N) = SPREAD (B(J, 1:N) , DIM=1, NCOPIES = N)
```

b) FORALL statement

- new in HPF
- included in Fortran 95

```
FORALL (I=2 : N-1)
  A (I) = ( A(I-1) + A(I+1) ) * 0.5
END FORALL

FORALL (I=1:N, K=1:N)
  B (I, K) = B (J, K)
END FORALL
```

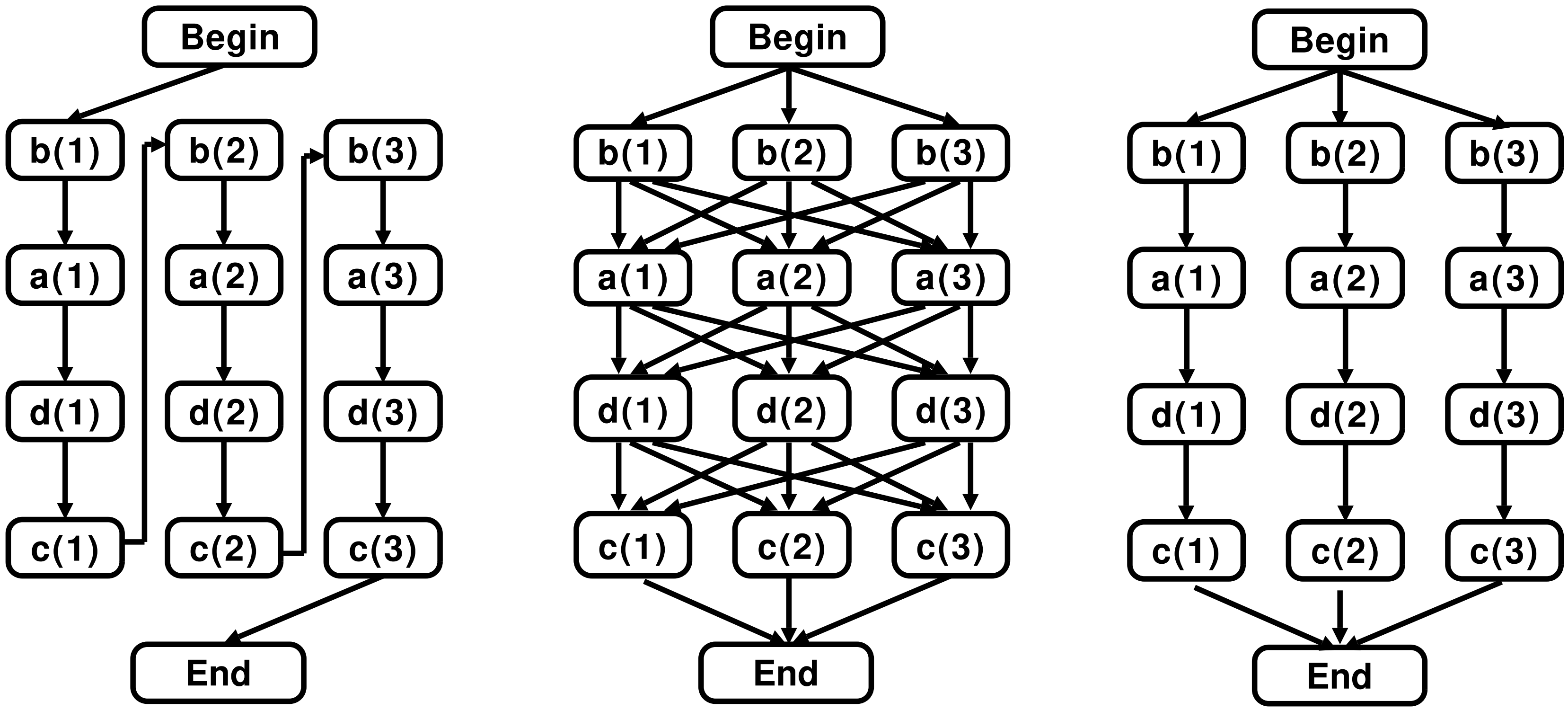
c) INDEPENDENT directive

```
!HPF$ INDEPENDENT
    DO i = 1, N
       Y( K(i) ) = X (i) + C ( K(i) ) * Y ( K(i) )
    END DO
```

```
DO I = 1, 3
  a(I) = b(I)
  c(I) = d(I)
END DO
```

```
FORALL (I=1:3)
  a(I) = b(I)
  c(I) = d(I)
END FORALL
```

```
!HPF$ INDEPENDENT
      DO I = 1, 3
        a(I) = b(I)
        c(I) = d(I)
      END DO
```



Precedence graphs for DO / FORALL / INDEPENDENT loop

```
REAL A(N)
```

	1	2	3	4	5	6	7	...
A	15	3	5	1	26	13	9	

```
DO I = 2, N
   A(I) = A(I-1) + 1
END DO
```

	1	2	3	4	5	6	7	...
A	15	16	17	18	19	20	21	

```
FORALL (I=2:N)
   A(I) = A(I-1) + 1        A(2:N) = A(1:N-1)+1
END FORALL
```

	1	2	3	4	5	6	7	...
A	15	16	4	6	2	27	14	

FORALL Statement ‡ DO Statement

PURE Attribute

Problem: call of subroutines / functions in parallel loops

```
FORALL (I=1,N)
   Y (i) = f (X(I), I)
END FORALL
```

```
PURE REAL FUNCTION f (x, i)
  REAL x
  INTEGER i
  IF (x > 0.0) THEN
   ....
  END IF
END FUNCTION
```

Pure procedures

- cannot have side effects on global data
- cannot have side effects on its arguments
- cannot contain I/O
- can only call pure procedures again
- cannot have local variables with SAVE attribute
- have also many other restrictions

===> behave like a mathematically pure function

Pure Procedures

New intrinsic functions

a) system / mapping inquiry functions

- NUMBER_OF_PROCESSORS ()
- PROCESSORS_SHAPE ()

b) computational intrinsics

- extension of MINLOC, MAXLOC (DIM argument)
- new reduction functions (IPARITY, IALL, IANY, ...)=
- array combining scatter functions
- array prefix and suffix functions
- sorting
- bit manipulation functions

Examples:

PROCESSORS_SHAPE() is (/ 4, 4 /)
NUMBER_OF_PROCESSORS() is 16

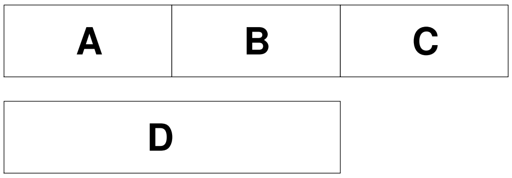
SUM_PREFIX (/1,3,5,7/) is (/1,4,9,16/)

PARITY (/ T, F, F, T, T, F, F, F, T, T /) is .TRUE.
PARITY (/ T, F, F, T, T, F, F, F, T, F /) is .FALSE.

IALL (/ 7, 3, 10 /) is 2
IANY (/ 7, 3, 10 /) is 15

Storage Association

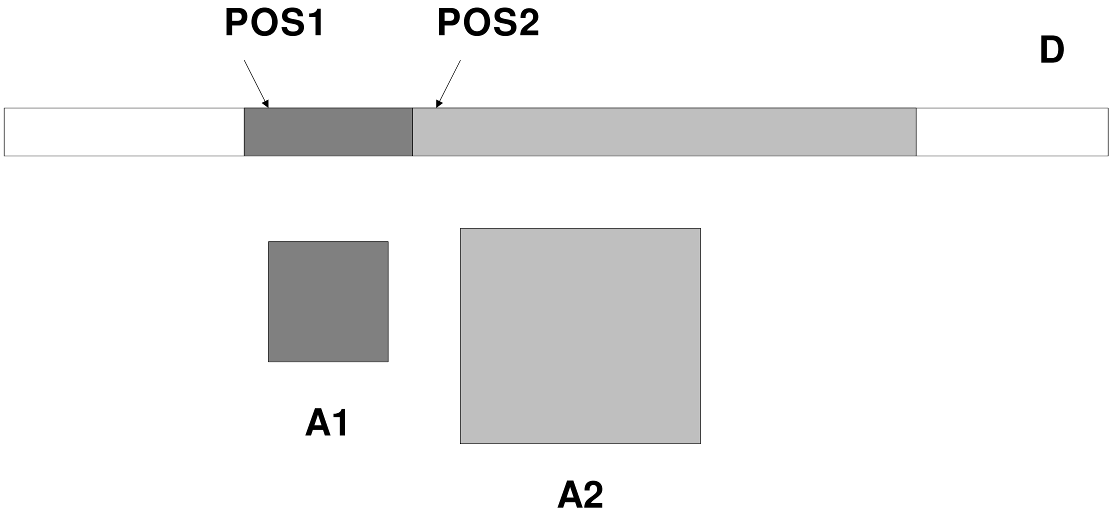
```
COMMON  /C1/ A, B, C
DIMENSION A(100), B(100), C(100), D(200)
EQUIVALENCE (A(1), D(1))
```



Sequence Association

```
COMMON /DATA/ D(100000)
...
CALL SUB ( D(POS1), D(POS2), N)
...

SUBROUTINE SUB (A1, A2)
REAL A1(0:N,0:N), A2(0:2*N,0:2*N)
...
```



Solution: sequential variables

5. Storage / Sequence Association

```
      real A(100,100)
!HPF$ distribute A(BLOCK, BLOCK)
      ...
      A = 5
      call SUB (A, 100)
      ...
```

SUB is an external subroutine !!

Possibility 1 :

* same concept as HPF (e.g. HPF, C*), single-thread

Possibility 2 :

GLOBAL

* multiple-thread-of-control (spawning)

Possibility 3:

LOCAL

* targeted to a single processor (local procedure)

Local Subroutines:

- SUB wil be called on every processor
- only the local data is visible
- synchronization at entry and leave of subroutine
- special inquiry functions for global view of data
- user has to guarantee consistency by himselves (replicated variables must be updated consistently)

Goal

- minimal starting set of features from Fortran 90 and HPF
- early release of compilers with HPF features

What is in the subset:

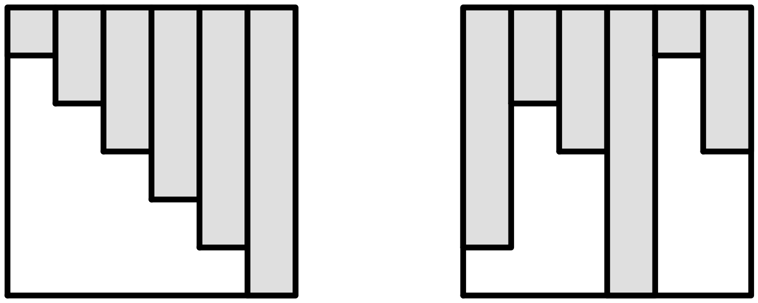
- Fortran 77
- MIL-STD-1753 (DO WHILE, END DO, INCLUDE, IAND, ...)
- Array Syntax (arithmetic, logical features)
- dynamic arrays (allocatable, automatic arrays)
- Intrinsic Procedures
. extended type declarations
- INTERFACE
- optional arguments
- syntax improvements

What is not in the subset:

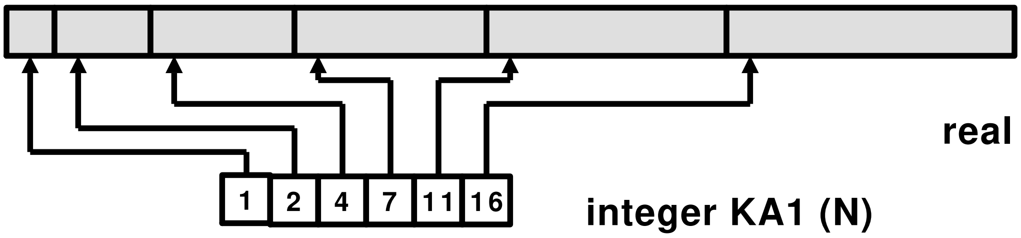
- derived types (TYPE ...)
- MODULE, USE
- REALIGN, REDISTRIBUTE, DYNAMIC
- PROCESSOR_VIEW
- PURE
- FORALL construct
- HPF_library
- EXTRINSIC, LOCAL
- POINTER, TARGET

7. Subset High Performance Fortran

real A1(N,N) real A2(N,N)



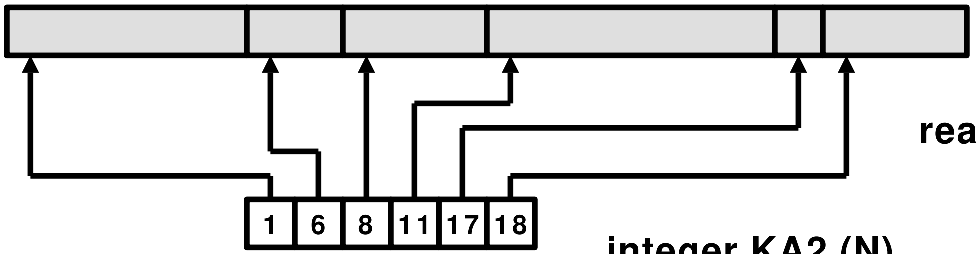
A1 A2



real DA1(M)

integer KA1 (N)

A (I,J) => DA (KA(I) + J - 1)



real DA2(M)

integer KA2 (N)

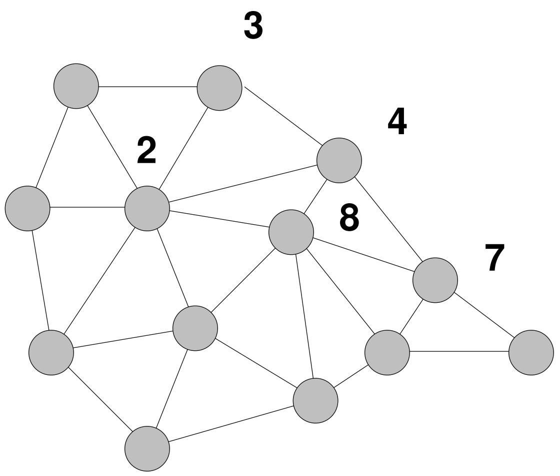
- work space arrays
- sparse data structures (?)

HPF : unsupported data mapping (1)

```
PARAMETER (atomMax = 8000, pMax = 250)

REAL x(atomMax), f(atomMax)

INTEGER inb (atomMax), partners (atomMax, pMax)
```



x(4) : position of atom 4
f(4) : force of atom 4

inb (4) = 4

partners (4,1) = 3
partners (4,2) = 2
partners (4,3) = 8
partners (4,4) = 7

```
DO i = 1, atomMax
   DO p = 1, inb (i)
      j = partners (i,p)
      force = nbforce (x(i), x(j)
      f(i) = f(i) + force
      f(j) = f(j) - force
   END DO
END DO
```

Problem 1 : data locality

Idea of solution : distribution by position (not by index)

Problem 2: load balancing (due to movement of atoms)

HPF : unsupported data mapping (2)