High Performance Fortran (HPF)

Source:

Chapter 7 of "Designing and building parallel programs" (Ian Foster, 1995)

Question

Can't we just have a clever compiler generate a parallel program from a sequential program?

Fine-grained parallelism

```
x = a*b + c*d
```

Trivial parallelism

```
for i := 1 to 100 do
   for j := 1 to 100 do
      C[i,j] := dotproduct(A[i,*], B[*,j]);
   od
od
```

Automatic parallelism

Automatic parallelization of *any* program is extremely hard

Solutions:

- Make restrictions on source program
- Restrict kind of parallelism used
- Use semi-automatic approach
- Use application-domain oriented languages

High Performance Fortran (HPF)

Designed by a forum from industry, government, universities

Extends Fortran 90

To be used for computationally expensive numerical applications

Portable to SIMD machines, vector processors, shared-memory MIMD and distributed-memory MIMD

Fortran 90 – Base language of HPF

Extends Fortran 77 with 'modern' features

- abstract data types, modules
- recursion
- pointers, dynamic storage

Array operators

- $A = B + C$
- $A = A + 1.0$
- $A(1{:}7) = B(1{:}7) + B(2{:}8)$
- WHERE $(X /= 0)$ $X = 1.0/X$

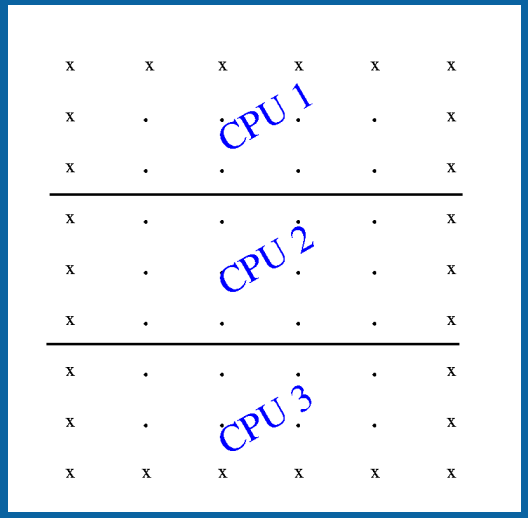
Data parallelism

Data parallelism: same operation applied to different data elements in parallel

Data parallel program: sequence of data parallel operations

Overall approach:

- Programmer does domain decomposition
- Compiler partitions operations automatically



Data may be regular (array) or irregular (tree, sparse matrix)

Most data parallel languages only deal with arrays

Data parallelism – Concurrency

Explicit parallel operations

```
A = B + C              ! A, B, and C are arrays
```

Implicit parallelism

```
do i = 1,m
   do j = 1,n
      A(i,j) = B(i,j) + C(i,j)
   enddo
enddo
```

Compiling data parallel programs

Programs are translated automatically into parallel SPMD (Single Program Multiple Data) programs

Each processor executes same program on a subset of the data

Owner computes rule:

- Each processor owns subset of the data structures
- Operations required for an element are executed by the owner
- Each processor may read (but not modify) other elements

Example

```
real s, X(100), Y(100) ! s is scalar, X and Y are arrays

X = X * 3.0          ! Multiply each X(i) by 3.0
do i = 2,99
  Y(i) = (X(i-1) + X(i+1))/2 ! Communication required
enddo
s = SUM(X)                    ! Communication required
```

Arrays X and Y are distributed (partitioned)

Scalar s is replicated on each machine

HPF primitives for data distribution

Directives:

- PROCESSORS: shape & size of abstract processors
- ALIGN: align elements of different arrays
- DISTRIBUTE: distribute (partition) an array

Directives affect performance of the program, not its result

Processors directive

```
!HPF$ PROCESSORS P(32)
!HPF$ PROCESSORS Q(4,8)
```

Mapping of abstract to physical processors not specified in HPF (implementation-dependent)

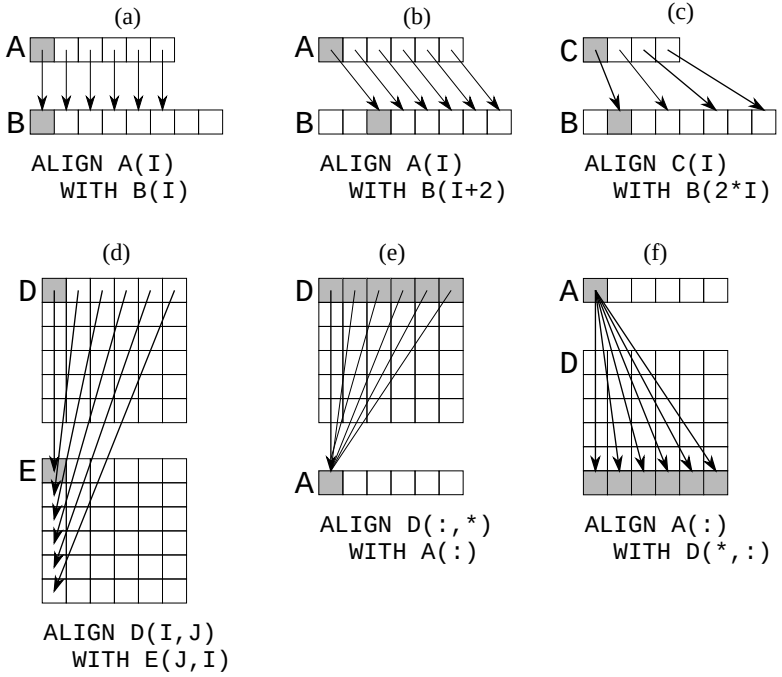
Alignment directive

Aligns an array with another array

Specifies that specific elements should be mapped to the same processor

```
real A(50), B(50), C(50,50)
!HPF$ ALIGN A(I) WITH B(I)
!HPF$ ALIGN A(I) WITH B(I+2)
!HPF$ ALIGN A(:) WITH C(*,:)
```

Figure 7.6 from Foster's book



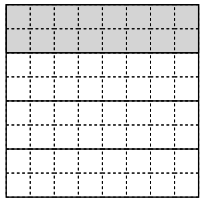
Distribution directive

Specifies how elements should be partitioned among the local memories

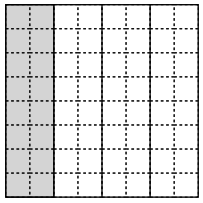
Each dimension can be distributed as follows:

`*`	`no distribution`
`BLOCK(n)`	`block distribution`
`CYCLIC(n)`	`cyclic distribution`

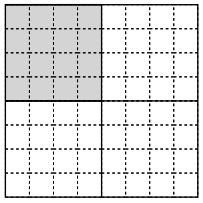
Figure 7.7 from Foster's book



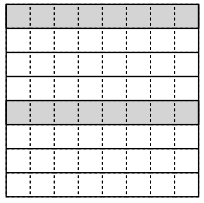
(BLOCK,*)



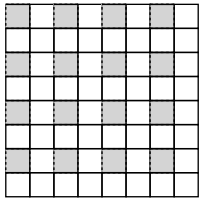
(*,BLOCK)



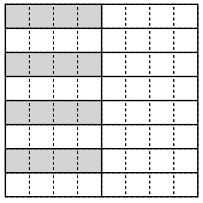
(BLOCK,BLOCK)



(CYCLIC,*)



(CYCLIC,CYCLIC)



(CYCLIC,BLOCK)

Example program

See Ian Foster, Program 7.2

```
      program hpf_finite_difference
!HPF$ PROCESSORS pr(4)                      ! use 4 CPUs
      real X(100, 100), New(100, 100)   ! data arrays
!HPF$ ALIGN New(:,:) WITH X(:,:)
!HPF$ DISTRIBUTE X(BLOCK,*) ONTO pr     ! row-wise

        New(2:99, 2:99) = (X(1:98, 2:99) + X(3:100, 2:99)
     $                   + X(2:99, 1:98) + X(2:99, 3:100))/4
        diffmax = MAXVAL(ABS(New-X))
        end
```

Example program (2)

Use block distribution instead of row distribution

```
      program hpf_finite_difference
!HPF$ PROCESSORS pr(2,2)                    ! use 2x2 grid
      real X(100, 100), New(100, 100)       ! data arrays
!HPF$ ALIGN New(:,:) WITH X(:,:)
!HPF$ DISTRIBUTE X(BLOCK, BLOCK) ONTO pr    ! block-wise

        New(2:99, 2:99) = (X(1:98, 2:99) + X(3:100, 2:99)
     $                   + X(2:99, 1:98) + X(2:99, 3:100))/4
        diffmax = MAXVAL(ABS(New-X))
        end
```

Performance

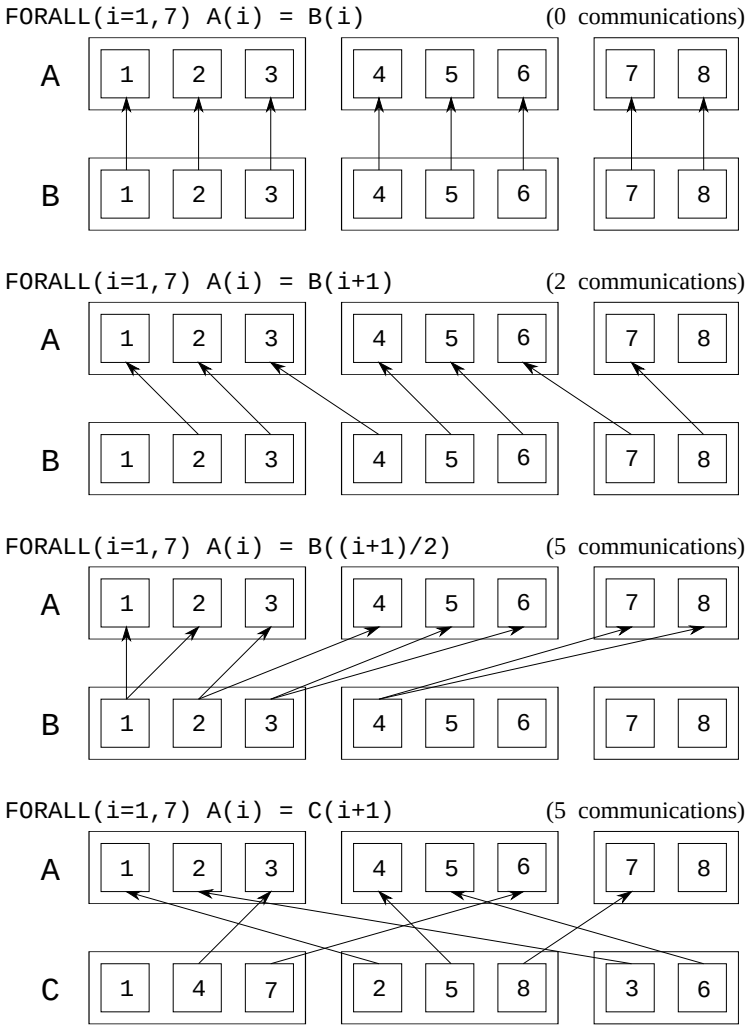
Distribution affects

- Load balance
- Amount of communication

Example (communication costs):

```
!HPF$ PROCESSORS pr(3)
      integer A(8), B(8), C(8)
!HPF$ ALIGN B(:) WITH A(:)
!HPF$ DISTRIBUTE A(BLOCK) ONTO pr
!HPF$ DISTRIBUTE C(CYCLIC) ONTO pr
```

Figure 7.9 from Foster's book



Conclusions

High-level model

- User specifies data distribution
- Compiler generates parallel program + communication

More restrive than general message passing model (only data parallelism)

Restricted to array-based data structures

HPF programs will be easy to modify, which enhances portability

Changing data distribution only requires changing directives