High Performance Fortran

- A small set of extensions to Fortran 90 supporting data parallelism
- Most extensions are structured comments
  - !HPF$ ALIGN = fine-grain locality
  - !HPF$ DISTRIBUTE = agglomeration
- A few new statements and functions
  - FORALL, HPF library
- Designed by academia, industry & gov't

Data-Parallel Languages

- Abstractions like data parallelism split the work between programmer & compiler
- Programmer solves the problem abstractly
  - Aggregate operations on large, distributed data structures
- Compiler maps the concepts (massive parallelism) to physical machine
  - Fill in the grungy details

How HPF Is Implemented

- Analysis
  - Dataflow, dependence & mapping analysis
- Computation Partitioning
  - Use data mapping info to create locality
- Communication Introduction
  - Lots of optimization to package communication
- Code Generation
  - Lots more optimization and really ugly details

HPF Language Features

- All of Fortran 90
- Parallelism: FORALL and INDEPENDENT
- Mapping: ALIGN and DISTRIBUTE
- But:
  - No parallel I/O or explicit message-passing
  - Little support for irregular computations or non-data parallelism

Fortran 90

- FORTRAN 77 plus …
- Array expressions & assignments

  b = MATMUL(a,b) / SUM(a) + c (1:n)
- Dynamic memory allocation

  ALLOCATE(a(1:n,1:m))
- Explicit Procedure Interfaces
- User-defined types & pointers
- Syntax Improvements

Data Parallelism

- HPF exploits data parallelism through:
  - **Array expressions:** Taken from Fortran 90
  - **FORALL:** Tightly-coupled parallel execution based on an index space
    - **PURE:** Procedures without side effects that may be called in FORALL
  - **INDEPENDENT:** Assertion that iterations do not interfere with each other
  - **HPF library**: Other useful operations

FORALL Statement

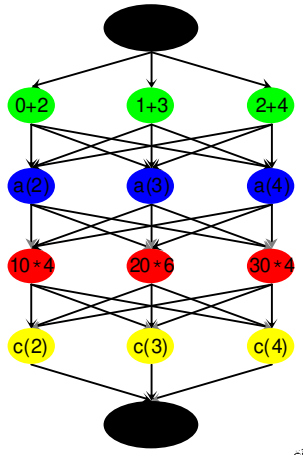
Initially,

a = [0, 1, 2, 3, 4]
b = [0, 10, 20, 30, 40]
c = [-1, -1, -1, -1, -1]

```
FORALL ( i = 2:4 )
    a(i) = a(i-1) + a(i+1)
    c(i) = b(i) * a(i+1)
END FORALL
```

Afterwards,

a = [0, 2, 4, 6, 4]
b = [0, 10, 20, 30, 40]
c = [-1, 40, 120, 120, -1]



INDEPENDENT Assertion

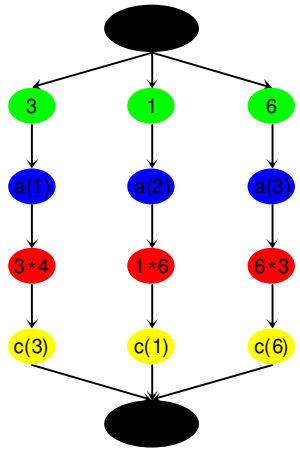
Initially,

a = [0, 2, 4, 6, 1, 3, 5, 7]
b = [6, 5, 4, 3, 2, 3, 4, 5]
c = [-1,-1,-1,-1,-1,-1,-1,-1]

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

Afterwards,

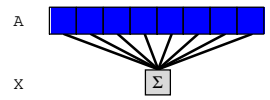
a = [3, 1, 6, 6, 1, 3, 5, 7]
b = [6, 5, 4, 3, 2, 3, 4, 5]
c = [6,-1,12,-1,-1,18,-1,-1]



HPF Library

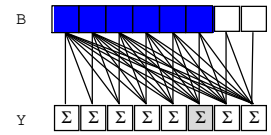
X = SUM(A)

$$x = \sum_{i=1}^{n} a_i$$



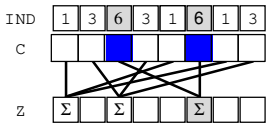
Y = SUM_PREFIX(B)

$$y_j = \sum_{i=1}^{j} b_i$$



Z = SUM_SCATTER(C, Z, IND)

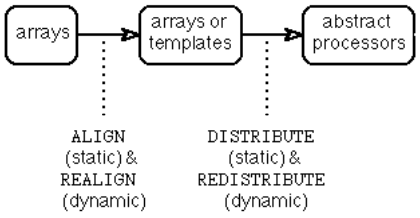
$$z_j = \sum_{i \ni ind_i = j} c_i$$



Uses of Parallel Features

- FORALL extends array assignments
  - New shapes (main diagonal, lower triangle, irregular regions)
  - PURE allows complex elemental computations
- INDEPENDENT gives compiler more information
  - Safe to run DO in parallel
- HPF library provides aggregate operations

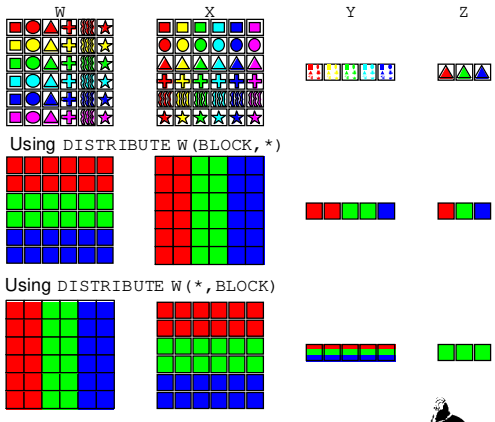
Data Mapping



- HPF uses a two-phase data mapping:
  - **ALIGN:** Creates a relationship between objects
  - **DISTRIBUTE:** Partitions an object between processors

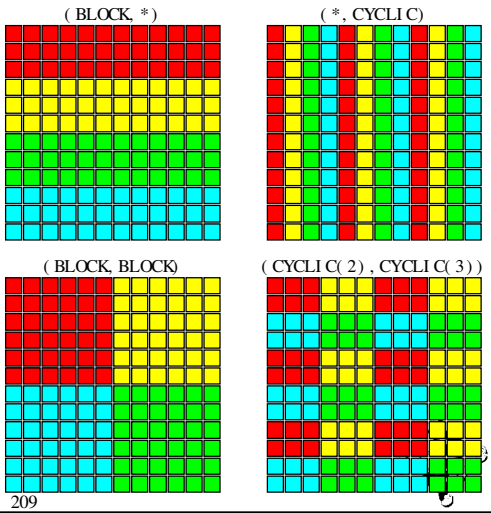
ALIGN Directive

```
REAL W(6,6),X(6,6),Y(5),Z(3)
!HPF$ALIGN X(I,J)WITH W(J,I)
!HPF$ALIGN Y(K)WITH W(K,*)
!HPF$ALIGN Z(L)WITH X(3,2*L-1)
```



DISTRIBUTE Directive

```
REAL W(12,12),X(12,12),
REAL Y(12,12),Z(12,12)
!HPF$ DISTRIBUTE W(BLOCK,*)
!HPF$ DISTRIBUTE X(*,CYCLIC)
!HPF$ DISTRIBUTE Y(BLOCK,BLOCK)
!HPF$ DISTRIBUTE
   Z(CYCLIC(2),CYCLIC(3))
```



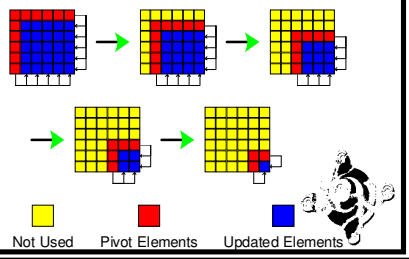
Uses of Mapping Features

- ALIGN matches individual elements of different arrays
  - Usually machine-independent
  - Use to "copy" array partitioning
- DISTRIBUTE agglomerates & maps arrays
  - Tune for individual machines
  - BLOCK for local, structured communication
  - CYCLIC for load balance

Gaussian Elimination Algorithm

- Problem:
  - Find N unknowns x_i to solve N linear equations
- Approach:
  - Solve Eqn. 1 for x_1, eliminate x_1 elsewhere
  - Solve Eqn. 2 for x_2, eliminate x_2 elsewhere, etc.

$$2.00\,x_1 + 1.00\,x_2 + 0.50\,x_3 + 0.25\,x_4 = 3.75 \quad \rightarrow \quad x_1 = 1.00$$
$$2.00\,x_1 + 2.00\,x_2 + 2.00\,x_3 + 2.00\,x_4 = 8.00 \quad \rightarrow \quad x_2 = 1.00$$
$$2.00\,x_1 + 3.00\,x_2 + 4.50\,x_3 + 6.75\,x_4 = 16.5 \quad \rightarrow \quad x_3 = 1.00$$
$$2.00\,x_1 + 4.00\,x_2 + 8.00\,x_3 + 16.0\,x_4 = 30.0 \quad \rightarrow \quad x_4 = 1.00$$



Gaussian Elimination: HPF Issues

- Each step classically data-parallel
  - Update: `a(i,j)=a(i,j)-a(i,K)/a(K,K)*a(K,j)`
  - Pivot: MAXLOC reduction, vector assignments
- Communication is structured but nonlocal
  - Reduction tree and broadcast tree
- Elements go inactive in later steps
  - DISTRIBUTE CYCLIC for load balance
  - ALIGN temporary with pivot row

Gaussian Elimination: HPF Program

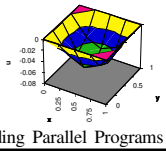
```
REAL a(n,n), tmp(n)
!HPF$ DISTRIBUTE a(CYCLIC,CYCLIC)
!HPF$ ALIGN tmp(i) WITH a(*,i)

DO k = 1, n-1

   ! Select the pivot
   ipivot = MAXLOC( ABS(a(k:n,k)) ) + k - 1

   ! Swap the rows
   tmp(k:n) = a(ipivot,k:n)
   a(ipivot,k:n) = a(k,k:n)
   a(k,k:n) = tmp(k:n)

   ! Update the submatrix
   FORALL ( i=k+1:n, j=k+1:n ) &
      & a(i,j) = a(i,j) - a(i,k)/tmp(k)*tmp(j)

END DO
```

Conjugate Gradient Algorithm I

- **Problem:**
  - Numerically solve a partial differential equation
- **Approach:**
  - Divide (continuous) space into a (discrete) grid & guess solution
  - Find error and search direction; minimize along search direction
  - Repeat until convergence

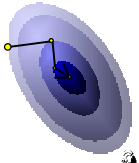
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2x^2 + 2x - 2y^2 + 2y$$

$u = 0$ if $x = 0, x = 1, y = 0,$ or $y = 1$



2-Dimensional Search

3-Dimensional Search



9-Dimensional Search

???

Conjugate Gradient Algorithm II

$u = \langle$ initial guess $\rangle$
$r = f - A * u$
$\delta = \max(|r|)$
$\iota = 0; \rho = 0$
WHILE $(\delta > \varepsilon)$ **DO**
  $\iota = \iota + 1; \rho_{old} = \rho$
  $\rho = r \cdot r$
  IF $(\iota=1)$ **THEN** $p = r$ **ELSE** $p = r + \rho/\rho_{old}\, p$
  $q = A * p$
  $\alpha = \rho / (p \cdot q)$
  $u = u + \alpha\, p$
  $r = r - \alpha\, q$
  $\delta = \max(|r|)$
END WHILE

Conjugate Gradient: HPF Issues

- Each step classically data-parallel
  - Reductions for dot products
  - Array assignments to p and r
  - Sparse matrix-vector multiply is data-parallel
- Communication is probably well-structured
  - Sparse matrix-vector multiply depends on PDE
- Simple mapping suffices for regular grids
  - Local PDE operators suggest BLOCK for all

Conjugate Gradient: HPF Program I

```
DO WHILE (err > tol)
   i = i + 1; rho_old = rho
   rho = SUM( r(1:n-1,1:n-1)**2 )
   IF  (i=1)  THEN
      p = r
   ELSE
      p = r + rho/rho_old * p
   END IF
   CALL a_times_vector(p, q)
   alpha = rho / SUM(p*q)
   u = u + alpha * p
   r = r - alpha * q
   err = MAXVAL(ABS(r(1:n-1,1:n-1)))
END DO
```

Conjugate Gradient: HPF Program II

```
REAL u(0:n,0:n), r(0:n,0:n), p(0:n,0:n)
REAL q(0:n,0:n), f(0:n,0:n)
!HPF$ DISTRIBUTE u(BLOCK,*)
!HPF$ ALIGN WITH u(:,:) :: r(:,:), p(:,:)
!HPF$ ALIGN WITH u(:,:) :: q(:,:), f(:,:)
INTERFACE
   SUBROUTINE a_times_vector( x, y )
   REAL, INTENT(IN) :: x(:,:)
   REAL, INTENT(OUT) :: y(:,:)
   !HPF$ DISTRIBUTE x *(BLOCK,*)
   !HPF$ ALIGN y(:,:) WITH *x(:,:)
   END SUBROUTINE
END INTERFACE
```

HPF Summary

- Parallelism comes from the data, esp. arrays
- Concurrent updates of many array elements
- Scalable to larger problems using larger arrays
- Locality from machine-independent alignment of arrays and machine-dependent distribution of data

HPF Addendum

- HPF has an "escape hatch" into lower-level languages
  - EXTRINSIC(HPF_LOCAL)
- HPF 2.0 is considering more extensions
  - Task parallelism, parallel I/O, irregular data distributions, computation mapping
- HPF 2.0 is considering some reductions
  - "Kernel HPF"