Fortran 90
and
HPF
(High-Performance Fortran)

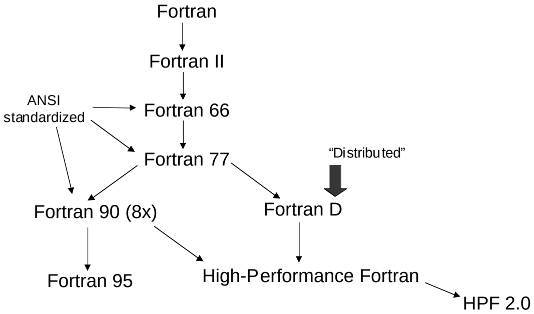
Reference Links

- Fortran 90 for the Fortran 77 programmer
  http://www.nsc.liu.se/~boein/f77to90/f77to90.html
- Edinburgh Parallel Computing Center course notes
  http://www.epcc.ed.ac.uk/epcc-tec/hpf/
- Design and Building Parallel Programs, by Ian Foster, chapter 7
  http://www-unix.mcs.anl.gov/dbpp/

Fortran Historically

- “Formula Translation”
- A venerable language, used in (some) scientific computing circles
- One of the oldest surviving languages with the same name
- Invented by John Backus of IBM in 1956 (who since became a proponent of functional programming)
- Contemporary of Algol-60

Fortran Background



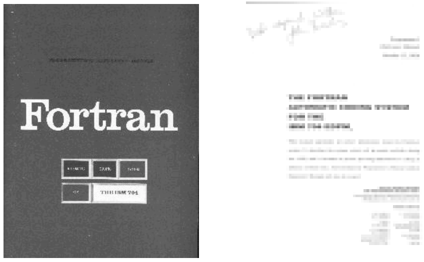
Backus Quote

I don't know what the technical characteristics of the standard language for scientific and engineering computation in the year 2000 will be

... but I know it will be called Fortran.

John Backus, 1980’s

Original Fortran Manual



Typical Fortran Program

```
      program gauss
c
c      this program does a gauss-seidel iteration to solve a set of
c      simultaneous equations with lhs coefficients a and rhs b

      double precision  a(6, 6), x(6), b(6), sum, oldx, maxdiff, max_error

      integer  i, j, n, steps
      max_error = 0.00001d0

c
c      initial guess
c
      do 100 i = 1, n
         x(i) = 0.0d0
100     continue

      steps = 0
```

```
150     continue
      maxdiff = 0.0d0
      do 300 i = 1, n
         oldx = x(i)
         sum = 0.0d0
         do 200 j = 1, n
            if( i .ne. j ) then
               sum = sum + a(i, j)*x(j)
            endif
200           continue
         x(i) = (b(i) - sum)/a(i, i)
         maxdiff = max(maxdiff, abs(x(i)-oldx))
300        continue

      steps = steps + 1

      if( maxdiff .gt. max_error ) then
         go to 150
      endif
```

Fortran Coding Sheet
(historical artifact)

Fortran is Conservative

- It added things like recursion, dynamic memory, and pointers 20 or more years after their appearance in other languages.

Fortran is Radical

- It adds things like array operations, array cross sections, distribution, and compiles them for parallel machines.

- It is one of the most optimizable and optimized languages.

Fortran is Surprising

- Given recent emphases on use for parallel computing, it is surprising that Fortran retains features that present obstacles:
  - Explicit ways to share (ALIAS) memory locations:
    - COMMON
    - EQUIVALENCE

Recent Variants

- Fortran 90:
  Oriented toward comprehensive array operations

- HPF (High-Performance Fortran):
  Oriented toward FORALL type construct

Sample Comparison

- Fortran 90:

```
X(2:N-1) = X(1:N-2) + X(2:N-1) + X(3:N)
```

- HPF:

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

Sample Comparison with Conditional Execution

- Fortran 90:

```
WHERE( X(1:N) .NE. 0.0) Y(1:N) = 1.0 / X(1:N)
```

- HPF:

```
FORALL(I=1:N, X(I).NE.0.0) Y(I) = 1.0 / X(I)
```

Fortran 90; Data Parallelism

- Entire arrays or sections of arrays can be operated on:
  - pairwise, or
  - by reduction operators
  - vector-valued subscripts (e.g. for permutation)
  - “where” construct for selective operations
  - “stride” for non-contiguous chunks of arrays (gather/scatter)

Example of F77 vs. F90

(from Ian Foster, DBPP)

F77

```
 real X(100, 100), New(100, 100), diffmax
 do i = 2, 99
   do j = 2, 99
     New(i, j) = (X(i-1, j) + X(i+1, j) + X(i, j-1) + X(i, j+1))/4
   enddo
 enddo
 diffmax = 0.0
 do i = 1, 100
   do j = 1, 100
     diff = abs(New(i, j)-X(i, j))
     if(diff .gt. diffmax) diffmax = diff
     enddo
   enddo
enddo
```

F90

```
 real X(100, 100), New(100, 100), diffmax
 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
```

F90 Vector Sectioning

- /1, 7, 3, 2/ denotes a constant vector

- V(/1, 7, 3, 2/), where V is a vector, denotes the vector
  V(1), V(7), V(3), V(2)

- When a vector subscript is used on the LHS of an assignment
  V(/1, 7, 3, 2/) = W
  each index must be distinct.

Array Intrinsic Functions

- maxval(A)
- maxloc(A)
- sum(A)
- dot_product(A, B)
- transpose(A)
- cyclic_shift(A, shift, dim)

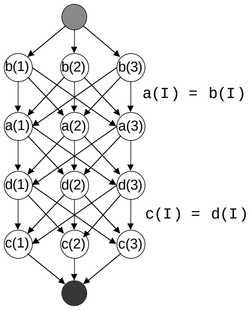
HPF: FORALL Statement

- Similar to DO statement, except
- Body can be executed in any order or in parallel; order is undefined
- Barrier between each statement in body

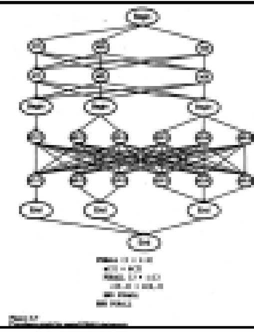
```
REAL, DIMENSION(N, N) :: A, B
   ...
FORALL (I = 2:N-1, J = 2:N-1)
  A(I,J) = 0.25*(A(I,J-1)+A(I,J+1)+A(I-1,J)+A(I+1,J))
  B(I,J) = A(I,J)
END FORALL
```

FORALL Semantics

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



Nested FORALL Semantics



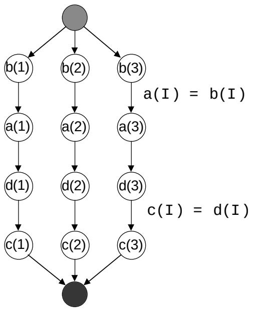
HPF: “INDEPENDENT” Directive

- Specifies that loop bodies (either DO or FORALL) are to be regarded as executable in parallel, based on programmer knowledge.
- The compiler can optimize based on this.
- With DO, there is no implied barrier between individual statements, as in the case of FORALL.

“INDEPENDENT” Semantics

(contrast to FORALL)

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



“INDEPENDENT” Directive

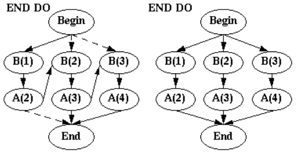
- Below it is *possible* that the same A(J) *could* be assigned to twice (non-deterministically).
- INDEPENDENT says either:
  - it can’t happen (due to additional knowledge of data), or
  - it doesn’t matter

```
!HPF$   INDEPENDENT
  DO I = 1 TO N
     A(INDEX(I)) = B(I)
  END DO
```

“INDEPENDENT” with DO

```
DO i= 2, 4
  A(i) = B(i-1)
END DO
```

```
!HPF$ INDEPENDENT
 DO i= 2, 4
  A(i) = B(i-1)
 END DO
```



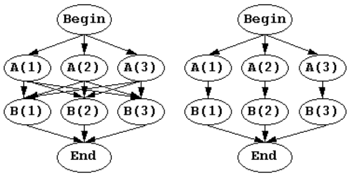
source: http://www.epcc.ed.ac.uk/epcc-tec/documents/hpf-course/hpf-course-66.html

---

“INDEPENDENT” with FORALL

```
ALL(i=1:3,A.NE.0.0) &
(i) = 1.0/A(i)
```

```
!HPF$ INDEPENDENT
      FORALL(i=1:3,A.NE.0.0) &
        B(i) = 1.0/A(i)
```



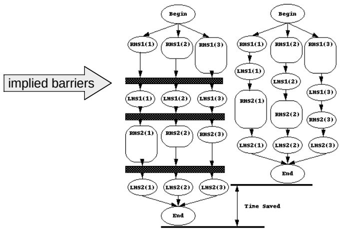
source: http://www.epcc.ed.ac.uk/epcc-tec/documents/hpf-course/hpf-course-66.html

---

“INDEPENDENT” with FORALL

```
! *NOT INDEPENDENT*
FORALL(i=1:3)
  LHS1(i) = RHS1(i)
  LHS2(i) = RHS2(i)
END FORALL
```

```
!HPF$ INDEPENDENT
   FORALL(i=1:3)
    LHS1(i) = RHS1(i)
    LHS2(i) = RHS2(i)
  END FORALL
```



source: http://www.epcc.ed.ac.uk/epcc-tec/documents/hpf-course/hpf-course-66.html

---

“NEW” Directive

- Prescribes variables for which new storage is allocated for **each iteration**.

```
!HPF$   INDEPENDENT, NEW(temp)
        DO I = 1 TO N
          temp = A(I) + B(I)
          A(I) = temp
          B(I) = temp
        END DO
```

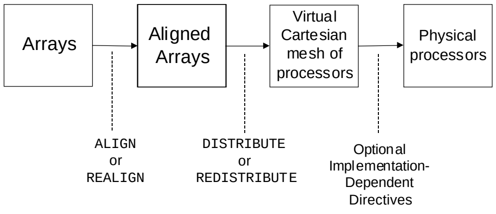
“PURE” Attribute

- Used to tell compiler that a function has no side-effects (and thus can be called in parallel with other functions):

```
PURE REAL FUNCTION MY_EXP(X)
    REAL, INTENT(IN) :: X
    MY_EXP = 1 + X + X*X / 2.0 + X**3 / 6.0
    END FUNCTION MY_EXP
```

indicates input vars

HPF Data Mapping Model



Arrays

Aligned Arrays

Virtual Cartesian mesh of processors

Physical processors

ALIGN
or
REALIGN

DISTRIBUTE
or
REDISTRIBUTE

Optional
Implementation-
Dependent
Directives

Why is “alignment” so important?

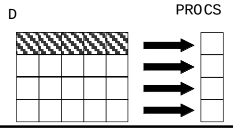
HPF **Alignment** Directives

```
       REAL A(1000), B(1000), C(1000), X(500), Y(0:501)
       INTEGER INX(1000)
!HPF$  PROCESSORS PROCS(10)

!HPF$  ALIGN X(I) WITH Y(I-1)

!HPF$  ALIGN X(I) WITH PROCS(I/50)

!HPF$  ALIGN D(:, *) WITH PROCS(:)
```

Virtual Processor Array

Array Alignment

Processor Alignment



HPF “Distribute” Directives

```
!HPF$   PROCESSORS PROCS(4)
!HPF$   DISTRIBUTE A(CYCLIC) ONTO PROCS
```

1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

```
!HPF$   DISTRIBUTE B(BLOCK) ONTO PROCS
```

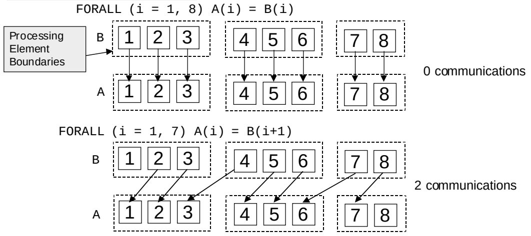
1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4

```
!HPF$   DISTRIBUTE C(BLOCK(2)) ONTO PROCS
```

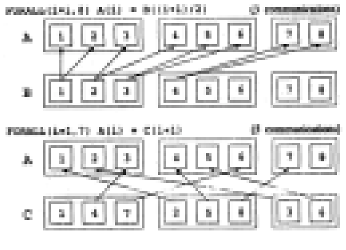
1	1	2	2	3	3	4	4	1	1	2	2	3	3	4	4

HPF Directives Allow Computation of Communication Costs

Blocked allocation



HPF Directives Allow Computation of Communication Costs

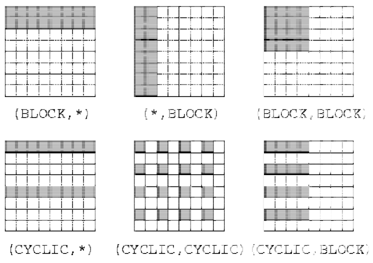


Blocked allocation

Cyclic allocation

```
!HPF$  ALIGN B(:) WITH A(:)
!HPF$  DISTRIBUTE A(BLOCK) ONTO pr
!HPF$  DISTRIBUTE C(CYCLIC) ONTO pr
```

2-D DISTRIBUTE



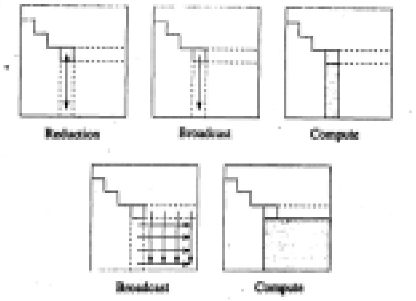
Other HPF Primitives

- PROCESSORS_SHAPE()
  e.g. /4, 8/ means 4x8 array
- NUMBER_OF_PROCESSORS()
  e.g. 32
- NUMBER_OF_PROCESSORS(1)
  e.g. 4
- NUMBER_OF_PROCESSORS(2)
  e.g. 8

Example:
Parallelism in Gaussian Elimination

- Gaussian elimination is one way to solve a system of equations.
- For i = 1, 2, …., n:
  - The maximum of column i is computed
  - row i is reserved and divided by the maximum.
  - Row operations are performed on all rows by subtracting a factor times the reserved row.
  - Rows are permuted
- For i = n, n-1, …., 1:
  - Back substitute

Example:
Parallelism in Gaussian Elimination



Gaussian Elimination using HPF (after Ian Foster, DBPP)

```
subroutine solve(n, A, X)
integer n
real A(n, n+1), X(n), Fac(n), Row(n+1), maxval
integer Indx(n), Itmp(1), i, j, k, max_indx

Indx = 0
do i = 1, n
  Itmp = MAXLOC(ABS(A(:, i)), Indx.EQ.0)
  max_indx = Itmp(1)
  Indx(max_indx) = i
  Fac = A(:, i) / A(max_indx, i)
  Row = A(max_indx, :)

  do j = 1, n
    FORALL (k=i:n+1, Indx(j).EQ.0) A(j, k) = A(j, k) - Fac(j)*Row(k)
    enddo
  enddo

FORALL (j=1:n) A(Indx(j), :) = A(j, :)

do j = n, 1, -1
  X(j) = A(j, n+1) / A(j, j)
  A(1:j-1, n+1) = A(1:j-1, n+1) - A(1:j-1, j)*X(j)
  enddo
end
```

To Compile HPF on turing

- `pghpf filename.f`
- (pg = “Portland Group”)