Fortran 90 and HPF

High-Performance Fortran

Reference Links

- http://www.nsc.liu.se/~boein/f77to90/f77to90.html
- http://loki.stockton.edu/~stk7857/Fortran.htm
- http://www.epcc.ed.ac.uk/epcc-tec/documents/hpf-course/hpf-course.book_1.html
- http://www.utexas.edu/cc/parallel/HPF/ (for pghpf compiler, available on turing; also see for more links)

Fortran

- "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

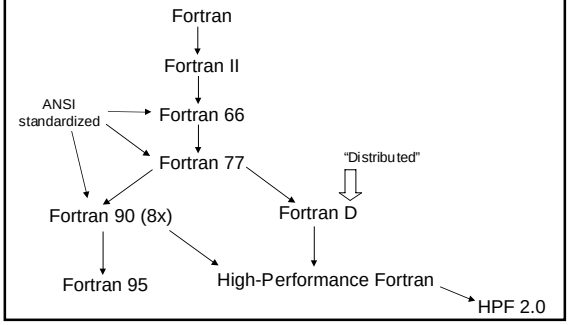
Backus Quote

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

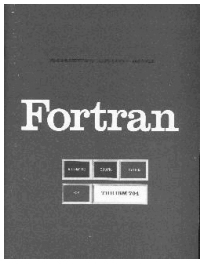
... but I know it will be called Fortran.

John Backus, 1980's

Fortran Background



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 adds 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

Fortran 90

- Data parallel:
  - 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)

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 assigment
  V(/1, 7, 3, 2/) = W
  each index must be distinct.

Array Intrinsic Functions

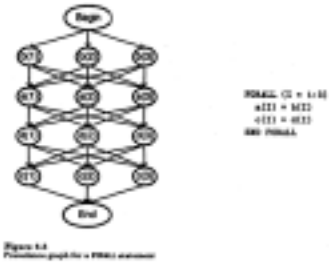
- maxval(A)
- maxloc(A)
- sum(A)
- dot_product(A, B)
- transpose(A)
- cyclic_shift(A, shift, dim)

HPF FORALL Statement

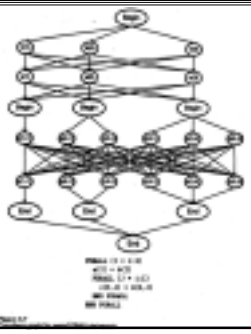
- Similar to DO statement, except
- Body can be executed in any order or in parallel
- Barrier between each statement in body

```
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



Nested FORALL Semantics



FORALL in Gaussian Elim.

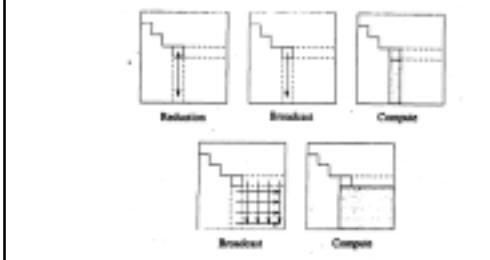


Figure 7.11 Communication and computation in the various phases of the HPF Gaussian elimination algorithm. Arrows represent communication, and shading indicates tasks involved in computation in each phase. The five phases are described in Section 7.8.

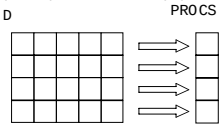
HPF Alignment Directives

```
      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

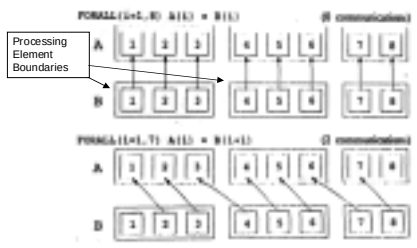
```
!HPF$   DISTRIBUTE B(BLOCK) ONTO PROCS
```

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

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

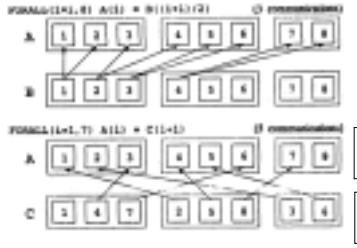
1	1	2	2	3	3	4	4	1	1	2	2	3	3	4	4

The HPF Directives Allow Computation of Communication Costs



Blocked allocation

The HPF Directives Allow Computation of Communication Costs



Blocked allocation

Cyclic allocation

HPF Data Mapping

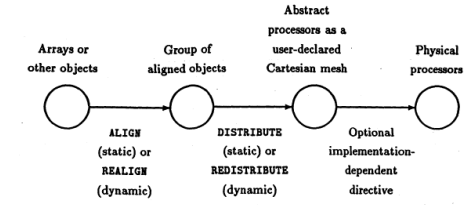


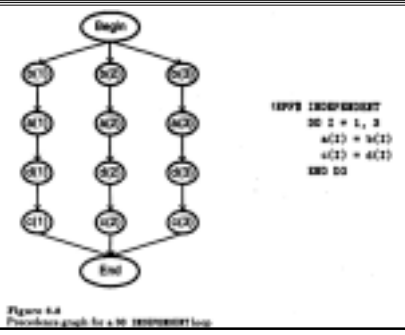
Figure 4.4
The HPF data mapping model

"Independent" Directive

- Specifies that loop bodies are to be regarded as executable in parallel.
- The compiler can optimized based on this.
- Below it is possible that the same A(J) could be assigned to twice (non-deterministically).
- INDEPDENT says it doesn't matter

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

"Independent" Semantics



"NEW" Directive

- Prescribes variables for which new storage is allocated.

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