



WWW.TACC.UTEXAS.EDU



Advanced Fortran

Advanced Tips and Tricks

Kevin Schmidt, Susan Lindsey, Charlie Dey

Tips

Arrays as Indices

```
real, dimension(5) :: a = [ 1, 3, 5, 7, 9 ]  
real, dimension(2) :: i = [ 2, 4 ]  
  
print *, a(i)           ! prints 3. 7.
```

- Variable *i* is an array (vector)
- *a(i)* is [*a(i(1))*, *a(i(2))*, ...]

Tips

Stencils

$$A_i = (A_{i-1} + A_{i+1})/2$$

```
real, dimension(n) :: a
real                      :: t1, t2
...
t2 = a(1)
do i=2, n-1
    t1 = a(i-1)
    a(i) = (t2 + a(i+1))/2.
    t2 = t1
enddo
```

Applying what we learn, we can code it this way.
Requiring scalar variables

Tips

Stencils

$$A_i = (A_{i-1} + A_{i+1})/2$$

```
real, dimension(n) :: a  
...  
...  
a(2:n-1) = (a(1:n-2) + a(3:n))/2.
```

Or, we can apply this trick, using slicing.

Tips

Stencils

$$A_{i,j} = (A_{i-1,j} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1})/4$$

```
real, dimension(n,n) :: a, b
do j=2, n-1
  do i=2, n-1
    b(i,j) = 0.25 * (a(i-1,j) + a(i+1,j) +
a(i,j-1) + a(i,j+1))
  enddo
enddo

do j=2, n-1
  do i=2, n-1
    a(i,j) = b(i,j)
  enddo
enddo
```

Here we'd create a temporary matrix, b, run our calculations, then copy it back to the original matrix, a.

Tips

Stencils

$$A_{i,j} = (A_{i-1,j} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1})/4$$

```
real, dimension(n,n) :: a  
...  
a(2:n-1,2:n-1) = 0.25 * (a(1:n-2,2:n-1)  
                           + a(3:n,2:n-1) + a(2:n-1,1:n-2)  
                           + a(2:n-1,3:n))
```

Or... do it without a temporary variable.

Exercise 1.

Euler's Method, Convert the following code so it uses a Stencil.

```
program Euler_method
implicit none
real, dimension (20)::x,y
integer::i
real :: h = .05,xo = 0,yo = 0,f
x(1)=xo
y(1)=yo
print*, "for x=",x(1) , "y=",y(1)
```

```
i=2
do
  x(i)=x(1)+(i-1)*h
  y(i)=y(i-1)+h*f(x(i-1),y(i-1))
  if (x(i) > 3.0) exit
  print*, "for x=",x(i) , "y=",y(i)
  i=i+1
enddo

end program
real function f(x,y)
implicit none
real ::x,y
f=1/5.0*(y**2-x**2
end
```

Tips

introducing forall

$$A_{i,j} = (A_{i-1,j} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1})/4$$

```
forall (i=2:n-1, j=2:n-1) &
a(i,j) = 0.25 *
          &
          (a(i-1,j) + a(i+1,j) + a(i,j-1) + a(i,j+1))
```

We can also do it with a "forall" statement, where the fortran statement looks almost just like the original formula

What does Fortran stand for again?

Tips forall

- FORALL is more versatile than normal array assignments:
 - can access unusual sections:

```
FORALL(i=1:n) A(i,i) = B(i) ! diagonal

DO j = 1, n
  FORALL (i=1:j) A(i,j) = B(i) ! triangular
END DO
```

- can use indices in righthand side expression:

```
FORALL (i=1:n, j=1:n, i/=j) A(i,j) = REAL(i+j)
```

- can call PURE procedures:

```
FORALL (i=1:n:3, j=1:n:5) A(i,j) = SIN(A(j,i))
```

- can use indirection (vector subscripting),:

```
FORALL (i=1:n, j=1:n) A(VS(i),j) = i+VS(j)
```

Exercise 2.

Create a 5x5 random matrix, using forall statements replace the diagonal elements such that the matrix is diagonally dominant.

$$A = \begin{bmatrix} 3 & -2 & 1 \\ 1 & -3 & 2 \\ -1 & 2 & 4 \end{bmatrix}$$

is diagonally dominant because:

$$|a_{11}| \geq |a_{12}| + |a_{13}| \text{ since } |+3| \geq |-2| + |+1|$$

$$|a_{22}| \geq |a_{21}| + |a_{23}| \text{ since } |-3| \geq |+1| + |+2|$$

$$|a_{33}| \geq |a_{31}| + |a_{32}| \text{ since } |+4| \geq |-1| + |+2|.$$

Tips

ANY statement

```
integer, parameter :: n = 100
real, dimension(n,n) :: a, b, c1, c2

c1 = my_matmul(a, b) ! home-grown function
c2 = matmul(a, b)    ! built-in function
if (any(abs(c1 - c2) > 1.e-4)) then
    print *, 'There are significant differences'
endif
```

- `my_matmul(a,b)` is a custom matrix multiplier that we wrote
- `matmul` is provided by the compiler
- `abs(c1 - c2)`: Array syntax creates a temporary array
- `any` "creates" a scratch array of type logical and returns one logical

Tips

ALL statement

```
integer, parameter :: n = 100
real, dimension(n,n) :: a, b, c1, c2

c1 = my_matmul(a, b) ! home-grown function
c2 = matmul(a, b)    ! built-in function
if (all(abs(c1 - c2) <= 1.e-4)) then
    print *, 'There are NO significant
differences'
endif
```

- `my_matmul(a,b)` is a custom matrix multiplier that we wrote
- `matmul` is provided by the compiler
- `abs(c1 - c2)`: Array syntax creates a temporary array
- all "creates" a scratch array of type logical and returns one logical

Tips

WHERE statement

```
real, dimension(100,100) :: x
...
call random_number(x) ! array

where (x < 0.5)
    x = 0.
end where

where (x < 0.6)
    x = 0.
else where
    x = 1.
end where

where (x > 0.8) x = 2. * x
```

- Use arrays like scalars
- where, end where
- where, else where, end where
- where in one line

Tips

WHERE statement

```
real, dimension(4) ::  
  a,  
  b = [ 5, 6, 7, 8 ],  
  c = [ -1, 0, 1, 2 ]  
  ...  
where (c /= 0)  
  a = b / c  
else where ! When c(i) = 0  
  a = 0.  
  c = 1.  
end where
```

Put one array in the where condition, and apply condition to other conformable arrays

- Arrays must have the same shape (conformable: rank and number of elements)
- Code block executes when condition is true for individual elements of array
- Code block can contain :
 - Array assignments
 - Other where, any, or forall constructs

Tips

Array Analysis

```
integer, parameter :: n = 5
real, dimension(n,n) :: a

call random_number(a)

write (0,'(5(1x,f4.1))' ) &
 ((a(i,j), j=1, n), i=1, n)

write(0,*) maxval(a), maxloc(a)
write(0,*) maxloc(a,dim=1)
write(0,*) maxloc(a,dim=2)
```

Also see:

maxval, minval, sum, product
maxloc, minloc

[output]

0.0	0.8	0.3	0.7	0.0
0.0	0.3	0.9	0.3	0.1
0.4	0.9	0.1	0.0	0.6
0.7	0.8	0.9	0.9	0.9
1.0	0.8	0.7	0.1	0.1

0.9630555 5 1
5 3 4 4 4 (col)
2 3 2 5 1 (row)