F90: array syntax

by definition data parallel and vectorizable

array = array expression

1 step: array expression is evaluated for all indices
2 step: assignment

but not faster than equivalent do loops
F90: array syntax

integer, parameter :: max=10
real, dimension(max) :: a, b

a(1:max) = (/ (real(i), i=1, max) / real(max-1)
b = 2.*a
b = sin(a)
b(1:max:2) = a(1:max:2)

F90: array syntax

type(relation_type) :: relation
integer, dimension(:), allocatable :: population ! declaration
allocate(population(0: number_of_elements)) ! allocation
population = population_of(relation) ! array as function result
F90: some elemental intrinsics

```fortran
occupied_lines = count(population(1:number_of_elements) /= 0 )
number_of_indices = sum(population)
largest_index = maxval(population)
real(kind_real_kind) :: y(n), mat(n,m), x(m)
y = matmul(mat, x)
```

F90: index vectors

```fortran
integer, dimension(n) :: population, index
integer, dimension(maxval(index)) :: pp
integer, dimension(n) :: permutation, inverse_permutation

population = pp(index)
!
```

! calculation of the inverse of a permutation
inverse_permutation(permutation) = (/ ( i, i = 1 , size(permutation) ) /)
F95: where

```fortran
subroutine div(A,C)
    integer,dimension(:) :: C
    integer,dimension(size(A)) :: B

    where ( A /= 0.0 ) B = 1.0 / A
endwhere
```

```
where ( A /= 0.0 )
    B = 1.0 / A
endwhere
```

---

F95: where

```fortran
w1 = minval (x)
w2 = maxval (x)
t = 0.0
do j = 1, n
    if (y(j) > w2) cycle
    if (w1 <= y(j)) then
        t = t + y(j)
    endif
enddo
```

```
w1 = minval (x)
w2 = maxval (x)
t = sum( z)
```

**sum of all values of y in the interval covered by x**

**traditional style**

**data parallel f90 style**

```
w1 = minval (x)
w2 = maxval (x)

where (w1 <= y .and. y <= w2)
    z = y
elsewhere
    z = 0.0
endwhere

t = sum( z)
```

**will be slower**
F95: forall .. end forall is a parallel do .. enddo

forall( i = 1:n, j = 1:n, a(i, j) .ne. 0.0 ) b(i, j) = 1.0 / a(i, j)

forall( i = 1:n, j = 1:n )
where( a(i, j) .ne. 0.0 ) b(i, j) = 1.0 / a(i, j)
end forall

F95: forall generalizes array syntax

! more general than array syntax:
forall( i = 1:n, j = 1:n ) h(i, j) = 1.0 / real(i + j - 1)
F95: forall ... end forall different from do ... enddo

! no recursion!
! Jakobi, not Gauss-Seidel

forall (i = 2:n-1, j = 2:n-1)
d(i, j) = 0.25*(c(i, j + 1) + c(i, j - 1) + c(i + 1, j) + c(i - 1, j)) - c(i, j)
c(i, j) = c(i, j) + eps*d(i, j)
end forall

optimization potential of forall is high
but not used of by todays compilers