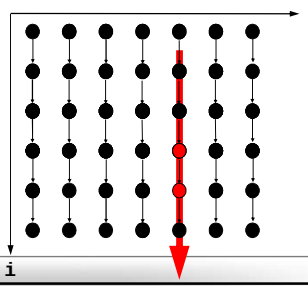
Example 3 Interchange and unrolling

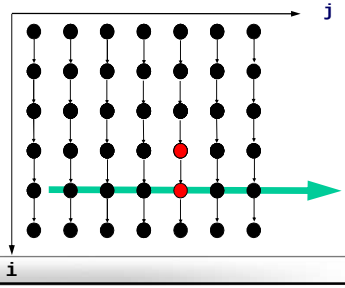
```
DO j=1,n
  DO i=2,n
    b(i,j) = SQRT(x(i,j))
    a(i,j) = a(i-1,j)*b(i,j)+z(i,j)
    y(i,j) = SIN(a(i,j))
  END DO
END DO
```



Vectorization not possible

Example 3 Interchange and unrolling ...

```
DO i=2,n
  DO j=1,n
    b(i,j) = SQRT(x(i,j))
    a(i,j) = a(i-1,j)*b(i,j)+z(i,j)
    y(i,j) = SIN(a(i,j))
  END DO
END DO
```



Vectorization possible !

Example 3 Interchange and unrolling (case n=100)

```
!CDIR NODEP
DO j = 1, 100
  b(2,j) = SQRT(x(2,j))
  a(2,j) = a(1,j)*b(2,j) + z(2,j)
  y(2,j) = SIN(a(2,j))
END DO

DO i = 2, 99, 2
!CDIR NODEP
  DO j = 1, 100
    b(i+1,j) = SQRT(x(i+1,j))
    b(i+2,j) = SQRT(x(i+2,j))
    a(i+1,j) = a(i,j)*b(i+1,j) + z(i+1,j)
    a(i+2,j) = a(i+1,j)*b(i+2,j) + z(i+2,j)
    y(i+1,j) = SIN(a(i+1,j))
    y(i+2,j) = SIN(a(i+2,j))
  END DO
END DO
```

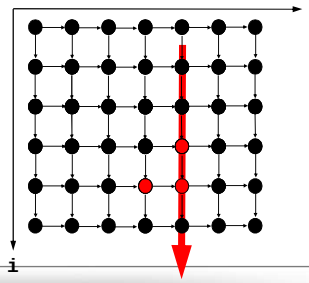
Case i = 2 separately because of unrolling

Interchange and unrolling

For simple cases: done automatically by the compiler

Example 4 2D recursion

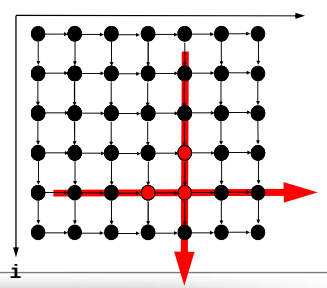
```
DO j=2,n
  DO i=2,m
    x(i,j)=rhs(i,j)-a(i,j)*x(i-1,j)-b(i,j)*x(i,j-1)
  END DO
END DO
```



Vectorization not possible

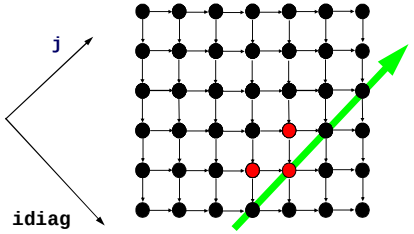
Example 4 … 2D recursion

```
DO i=2,m
  DO j=2,n
    x(i,j)=rhs(i,j)-a(i,j)*x(i-1,j)-b(i,j)*x(i,j-1)
  END DO
END DO
```



Vectorization not possible

Example 4 … 2D recursion



Vectorization possible !

```
DO idiag=1,m+n-1
!CDIR NODEP
   DO j = max(1,idiag+1-m), min(n,idiag)
     i=idiag+1-j
     x(i,j)= rhs(i,j)-a(i,j)*x(i-1,j)-b(i,j)*x(i,j-1)
   END DO
END DO
```


MITgcm – kpp_calc

```
        DO j = jmin, jmax
            jm1 = j - 1
            jp1 = j + 1
            DO i = imin, imax
               im1 = i - 1
               ip1 = i + 1
               shsq(i,j,k) = p5 * (
     $              (uVel(i,  j,  k,bi,bj)-uVel(i,  j,  kp1,bi,bj)) *
     $              (uVel(i,  j,  k,bi,bj)-uVel(i,  j,  kp1,bi,bj)) +
     $              (uVel(ip1,j,  k,bi,bj)-uVel(ip1,j,  kp1,bi,bj)) *
     $              (uVel(ip1,j,  k,bi,bj)-uVel(ip1,j,  kp1,bi,bj)) +
     $              (vVel(i,  j,  k,bi,bj)-vVel(i,  j,  kp1,bi,bj)) *
     $              (vVel(i,  j,  k,bi,bj)-vVel(i,  j,  kp1,bi,bj)) +
     $              (vVel(i,  jp1,k,bi,bj)-vVel(i,  jp1,kp1,bi,bj)) *
     $              (vVel(i,  jp1,k,bi,bj)-vVel(i,  jp1,kp1,bi,bj)) )
                …
            END DO
         END DO
```

Short test :

Exec. Time 0.27 sec

9000 MFlops

Vectorisation Ratio 99.6%

MITgcm – kpp_calc

```
!CDIR OUTERUNROLL=16
        DO j = jmin, jmax
            jm1 = j - 1
            jp1 = j + 1
            DO i = imin, imax
               im1 = i - 1
               ip1 = i + 1
               shsq(i,j,k) = p5 * (
     $              (uVel(i,  j,  k,bi,bj)-uVel(i,  j,  kp1,bi,bj)) *
     $              (uVel(i,  j,  k,bi,bj)-uVel(i,  j,  kp1,bi,bj)) +
     $              (uVel(ip1,j,  k,bi,bj)-uVel(ip1,j,  kp1,bi,bj)) *
     $              (uVel(ip1,j,  k,bi,bj)-uVel(ip1,j,  kp1,bi,bj)) +
     $              (vVel(i,  j,  k,bi,bj)-vVel(i,  j,  kp1,bi,bj)) *
     $              (vVel(i,  j,  k,bi,bj)-vVel(i,  j,  kp1,bi,bj)) +
     $              (vVel(i,  jp1,k,bi,bj)-vVel(i,  jp1,kp1,bi,bj)) *
     $              (vVel(i,  jp1,k,bi,bj)-vVel(i,  jp1,kp1,bi,bj)) )
                …
            END DO
         END DO
```

MITgcm – kpp_calc

```
.        do j = 5, 516, 16
.  !CDIR    NODEP
.           do i = 1, 516
.              shsq(i-3,j-3,k) = 5.00000000000000e-
.       1         001*((((uvel(i-3,j-3,k,
.       2         bi,bj)-uvel(i-3,j-3,kp1,bi,bj))*
.       3         (uvel(i-3,j-3,k,bi,bj)-
.       4         uvel(i-3,j-3,kp1,bi,bj)))+((uvel(i-2,j-3,k,
                …
.              shsq(i-3,j-2,k) =
                …
.              shsq(i-3,j-1,k) =
                …
.              shsq(i-3,j+11,k) =
                …
.              shsq(i-3,j+12,k) =
                …
.            end do
.          end do
```

Transformation listing **(-Wf,-L transform)**

MITgcm – kpp_calc

```
!CDIR OUTERUNROLL=16
      DO j = jmin, jmax
         jm1 = j - 1
         jp1 = j + 1
         DO i = imin, imax
            im1 = i - 1
            ip1 = i + 1
            shsq(i,j,k) =
     $           (uVel(i,  j,  k,bi,bj)-uVel(i,  j,  kp1,bi,bj)) *
     $           (uVel(i,  j,  k,bi,bj)-uVel(i,  j,  kp1,bi,bj)) +
     $           (uVel(ip1,j,  k,bi,bj)-uVel(ip1,j,  kp1,bi,bj)) *
     $           (uVel(ip1,j,  k,bi,bj)-uVel(ip1,j,  kp1,bi,bj)) +
     $           (vVel(i,  j,  k,bi,bj)-vVel(i,  j,  kp1,bi,bj)) *
     $           (vVel(i,  j,  k,bi,bj)-vVel(i,  j,  kp1,bi,bj)) +
     $           (vVel(i,  jp1,k,bi,bj)-vVel(i,  jp1,kp1,bi,bj)) *
     $           (vVel(i,  jp1,k,bi,bj)-vVel(i,  jp1,kp1,bi,bj)) )
               …
            END DO
         END DO
```

Short test :

Exec. Time 0.16 sec ***(- 40% !!)***

10000 MFlops

Vectorisation Ratio 99.7%

FORTRAN90/SX Compiler Directives

FORTRAN90/SX Compiler Directives

!cdir vector_directive

!cdir parallel_directive automatic parallelisation/multitasking

Compiler Directive (local effect) ⟷ Compiler Option (global effect)

Related NEC manuals

- SUPER-UX Performance Tuning Guide
- FORTRAN90/SX Programmer's Guide
- NEC SX Series Programming Environment Ready Reference

!CDIR NODEP

Maybe the most important directive…

The NODEP directive is used to let the compiler know that there are no dependencies between variables in a loop or array expression, so that those statements can be vectorized.

Example

```
!CDIR NODEP
DO i=1, n
 a(ind2(i)) = a(ind2(i)) + 2.0
END DO
```


!CDIR NOVECTOR

Usage

1) To test the effect of vectorisation on accuracy

2) To avoid unnecessary vectorisation

Example

```
!CDIR NOVECTOR
DO I=1,1000
       IF( A(I)-B(I) .LT. 1.0E-10) EXIT
       Z(I) = A(I)-B(I)
END DO
```

- If branch-out occurs at element 2, it makes no sense to vectorise the loop → Please help the compiler

!CDIR NOOVERLAP(X,Y)

Example

```
…
!CDIR NOOVERLAP(X,Y)
REAL, TARGET,  DIMENSION(n) :: Y
REAL, POINTER, DIMENSION(n) :: X
…
DO i=2, n
  X(i) = Y(i-1)+1
END DO
```

- Since X and Y may occupy overlapping areas of memory, the loop is not vectorised.

  If not, specify the directive to vectorise the loop.

!CDIR COLLAPSE

Example

```
!CDIR COLLAPSE
DO k=1,l
  DO j=1,m
    DO i=1,n
      a(i,j,k) = ...
    END DO
  END DO
END DO
```

→

```
DO i1=1,n*m*l
   a(i1,1,1) = ...
END DO
```

• The compiler does not check whether the arrays occupy contiguous areas !

If not, execution result will be incorrect.

!CDIR UNROLL=n

Example

```
!CDIR UNROLL=4
DO i=1,n
 a(i) = b(ind(i))+s(i)
END DO
```

→

```
i1 = iand(max0(n,0),3)
DO i = 1, i1
  a(i) = b(ind(i)) + s(i)
END DO
!CDIR NODEP
!CDIR NOUNROLL
DO i = i1 + 1, n, 4
  a(i) = b(ind(i)) + s(i)
  a(i+1) = b(ind(i+1))+s(i+1)
  a(i+2) = b(ind(i+2))+s(i+2)
  a(i+3) = b(ind(i+3))+s(i+3)
END DO
```

- Compiler takes care to handle any remainders not divisible by 4

!CDIR UNROLL=n …

Example 2

```
k = ...
!CDIR UNROLL=2
DO j=1,99,2
  DO i=1,n
    a(i) = a(i) + b(i,k) * c(k,j) + b(i,k)*c(k,j+1)
  END DO
END DO
```

- The load and store count of a(i) and the load count of b(i,k) are halved by unrolling

!CDIR ALTCODE

For each vectorizable loop

- The compiler generates the vector and the scalar version.
- At execution time, the most efficient version is selected based on data dependencies and/or vector length.