Vectorization II

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

Loop Vectorization

In which cases can loop be vectorized?

```
DO I = 1, N-1
  A(I) = A(I+1)*B(I)
ENDDO
```

```
A(1:128) = A(2:129)*B(1:128)
A(129:256) = A(130:257)*B(129:256)
....
```

Vectorization preserves semantics.

Loop Vectorization

Is semantics always preserved?

```
DO I = 2, N
  A(I) = A(I-1)*B(I)
ENDDO
```

```
A(2:129) = A(1:128)*B(2:129)
A(130:257) = A(129:256)*B(130:257)
....
```

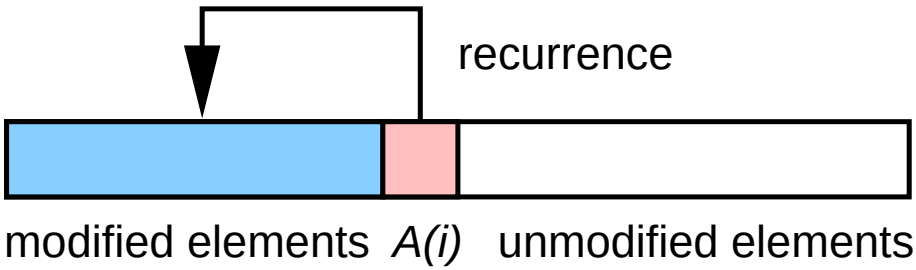
Vectorization has changed semantics!

Vectorization Inhibitors

Vectorization must be conservative; when in doubt, loop must *not* be vectorized.
Vectorization is inhibited by

- Function calls
- Input/output operations
- GOTOs into or out of loop
- Recurrences

  References to vector elements modified in previous iterations.



Goal is to maximize vectorization!

Vectorization Principles

- Loops over arrays

```
DO I = ...
  ... A(I) ...
ENDDO
```

- Induction variables

  Incremented/decremented by constant during each iteration

```
J = 1
DO I = 1, N
  J = J+3
  A(I) = A(J)*B(I)
ENDDO
```

  I and J are induction variables

Vectorization Principles

- Subscripts linear in induction variable

  Linear combination of induction variable and constants

```
DO I = ...
  ... A(I+C) ... A(D*I+C) ...
ENDDO
```

- Only innermost loop is vectorized

```
DO I = ...
  DO J = I+1, ...
    B(J-I) = A(I)*B(J)
  ENDDO
ENDDO
```

  `I` is constant in inner loop!

Compiler transforms loops to allow vectorization and/or to improve performance.

Partial Vectorization

Split loops to allow vectorization take place.

Non-vectorizable loop

```
DO I = 1, N
  A(I) = A(I-1)*X
  B(I) = C(I)**Y
ENDDO
```

Non-vectorizable loop

```
DO I = 1, N
  A(I) = A(I-1)*X
ENDDO
```

Vectorizable loop

```
DO I = 1, N
  B(I) = C(I)**Y
ENDDO
```

Loop Interchange

Interchange nested loops

```
DO I = 1, N
  DO J = 1, N
    ...
  ENDDO
ENDDO
```

```
DO J = 1, N
  DO I = 1, N
    ...
  ENDDO
ENDDO
```

- Increase iteration count
- Achieve most efficient vector stride (+1)
- Avoid non-vectorizable recurrences

Loop Distribution

Allow code in outer loop to be vectorized.

```
DO J = 1, N
  A(J,J) = X(J)*3
  DO I = 1, N
    B(I,J) = A(I,J)*C(I,J)
  ENDDO
ENDDO
```

```
DO J = 1, N
  A(J,J) = X(J)*3
ENDDO

DO J = 1, N
  DO I = 1, N
    B(I,J) = A(I,J)*C(I,J)
  ENDDO
ENDDO
```

Both loops can be vectorized!

Vectorization of Conditionals

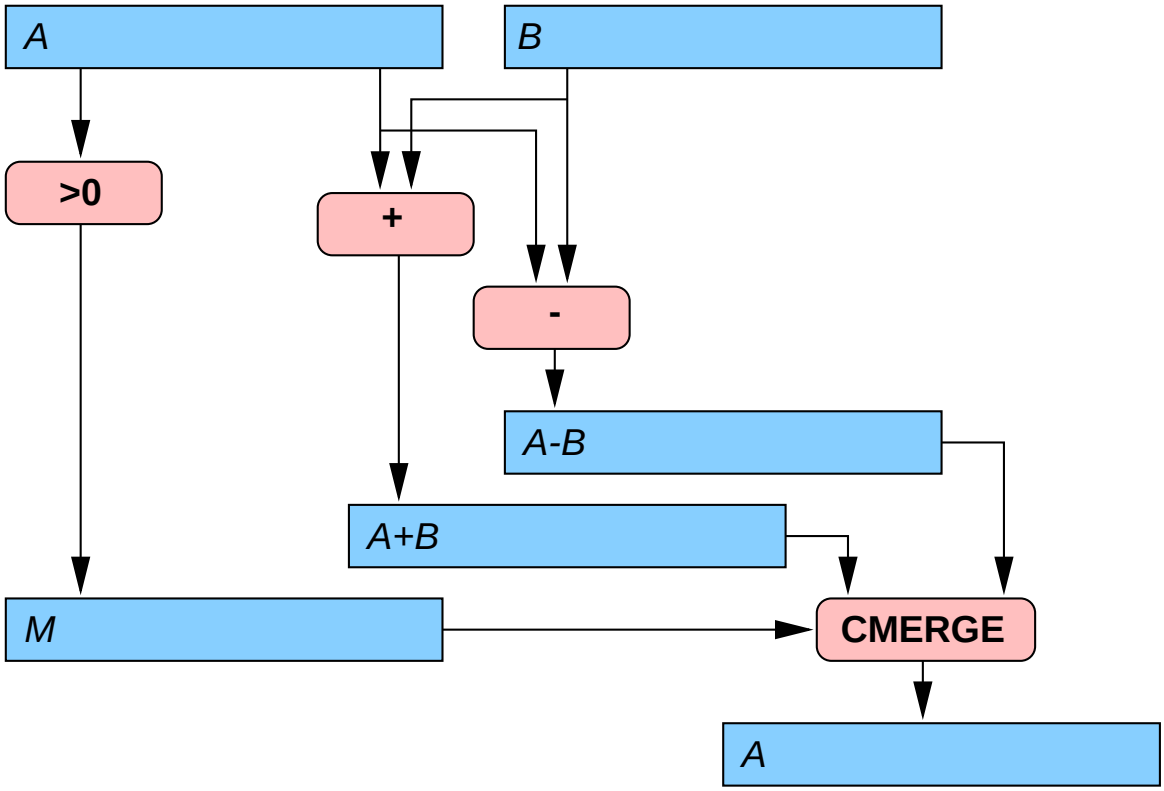
Merge different vectors.

```
DO I = 1, N
  IF (A(I).GT.0)
    A(I) = A(I)+B(I)
  ELSE
    A(I) = A(I)-B(I)
  ENDIF
ENDDO
```

```
M(1:N) = A(1:N).GT.0
A(1:N) = CMERGE(M, A+B, A-B)
```

Both branches are computed for all elements!

Conditional Vector Merge



Vector Reductions

Also some assignments into scalars can be vectorized.

```
S = 0
DO I = 1, N
  S = S+A(I)
ENDDO
```

```
S = VECTOR_SUM(A)
```

Only for predefined reduction operations!

Example

Matrix multiplication

Fortran 77 source program

```
4 DO I = 1, N
5   DO J = 1, N
6     C(I,J) = 0
7     DO K = 1, N
8       C(I,J) = C(I,J)+A(I,K)*B(K,J)
      ENDDO
    ENDDO
  ENDDO
```

(See additional slide)

Loop Distribution

Loop 4 $\Rightarrow$ loop 4-1 and loop 4-2

```
4-1 DO I = 1, N
5-1   DO J = 1, N
6-1     C(I,J) = 0
      ENDDO
    ENDDO

4-2 DO I = 1, N
5-2   DO J = 1, N
7-2     DO K = 1, N
8-2       C(I,J)=C(I,J)+A(I,K)*B(K,J)
        ENDDO
      ENDDO
    ENDDO
```

Loop 5-1 becomes vectorizable!

Loop Interchange

Interchange loop 4-1 and loop 5-1

```
5-1 DO J = 1, N
4-1   DO I = 1, N
6-1     C(I,J) = 0
      ENDDO
    ENDDO
```

Vector stride becomes one word!

Interchange loop 5-2 and loop 7-2

```
5-2 DO J = 1, N
7-2   DO K = 1, N
4-2     DO I = 1, N
8-2       C(I,J)=C(I,J)+A(I,K)*B(K,J)
        ENDDO
      ENDDO
    ENDDO
```

Type of vector operation is changed!

Types of Vector Operations

```
DO _ = 1, N
  C(I,J) = C(I,J)+A(I,K)*B(K,J)
ENDDO
```

- Index K

  I and J are fixed
  $\Rightarrow$ Scalar = Scalar + Vector$*$Vector
  $\Rightarrow$ 1 vector reduction!

- Index J

  I and K are fixed
  $\Rightarrow$ Vector = Vector + Scalar$*$Vector
  $\Rightarrow$ 1 vector-vector, 1 scalar-vector
  $\Rightarrow$ vector strides n

- Index I

  J and K are fixed
  $\Rightarrow$ Vector = Vector + Vector$*$Scalar
  $\Rightarrow$ 1 vector-vector, 1 scalar-vector
  $\Rightarrow$ vector strides 1

Transformation Result

```
5-1 DO J = 1, N
4-1   DO I = 1, N
6-1     C(I,J) = 0
      ENDDO
    ENDDO
5-2 DO J = 1, N
7-2   DO K = 1, N
4-2     DO I = 1, N
8-2       C(I,J) = C(I,J)+A(I,K)*B(K,J)
        ENDDO
      ENDDO
    ENDDO
```

- Vector loops 4-1 and 4-2
- Scalar loops 5-1, 5-2 and 7-2

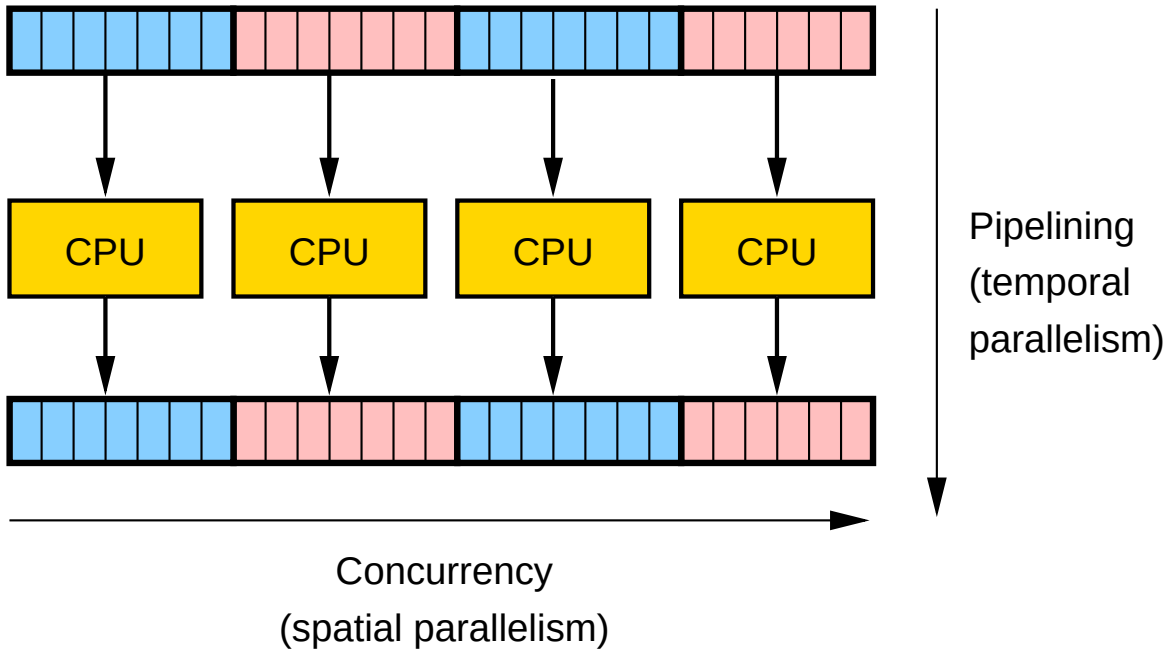
Vectorization Result

```
5-1 DO J = 1, N
      C(1:N,J) = 0
    ENDDO
5-2 DO J = 1, N
7-2   DO K = 1, N
        C(1:N,J)=C(1:N,J)+A(1:N,K)*B(K,J)
      ENDDO
    ENDDO
```

- Vector sources ("hoists") A, C
- Vector target ("sunk") C

Vector-Multiprocessors

Several vector processors operate on shared memory.



- Pipelining within processor.
- Concurrency among processors.

Most modern supercomputers!