Data Dependences

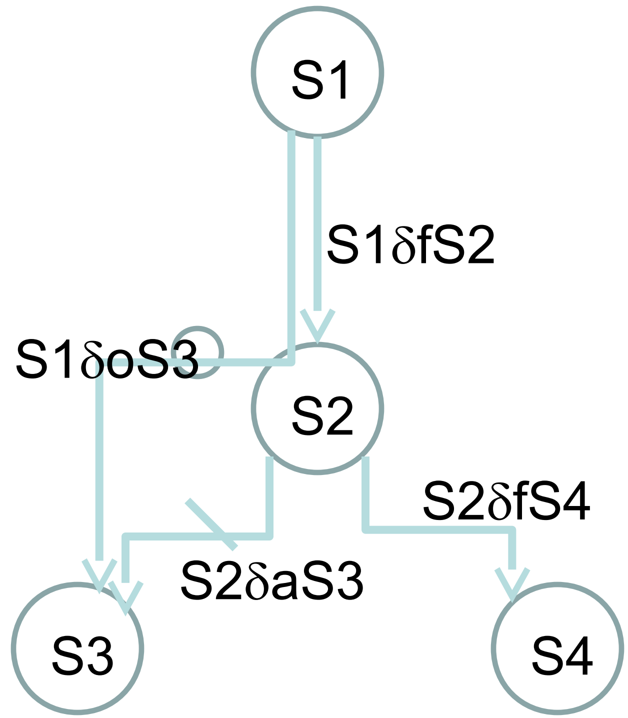
S1: a = b + c

S2: d = a * 2

S3: a = c + 2

S4: e = d + c + 2

stmt	read	write	
S1	**b,c**	**a**	
S2	**a**	**d**	**S1δfS2**
S3	**c**	**a**	**S1δoS3, S2δaS3**
S4	**d,c**	**e**	**S2δfS4**



Data Dependence Types

flow-dependence: occurs when a variable which is assigned a value in one statement, e.g. S1, is read by another statement, e.g. S2, later. Written as $S1\delta fS2$.

anti-dependence: occurs when a variable read by one statement, e.g. S1, is assigned an updated value in another statement, e.g. S2, later. Written as $S1\delta aS2$.

output-dependence: occurs when a variable which is assigned a value in one statement, e.g. S1, is later re-assigned an updated value in another statement, e.g. S2, later. Written as $S1\delta oS2$.

Data Dependences

S1: a = b + c

S2: d = a * 2

S3: a = c + 2

S4: e = d + c + 2

IN(Si): the set of memory locations read by the statement Si.

OUT(Si): the set of memory locations written by the statement Si.

(note a specific memory location may be both IN(Si) and OUT(Si).)

- A data dependence exists between two statements Si and Sj when the OUT set of one of the statements has a non-empty intersection with the IN or OUT set of the other

stmt	read	write		
S1	b,c	a		
S2	a	d	$OUT(S1) \cap IN(S2) = \{a\} \neq \emptyset$	→ $S1\delta fS2$
S3	c	a	$OUT(S1) \cap OUT(S3) = \{a\} \neq \emptyset$	→ $S1\delta oS3$,
			$IN(S2) \cap OUT(S3) = \{a\} \neq \emptyset$	→ $S2\delta aS3$
S4	d,c	e	$OUT(S2) \cap IN(S4) = \{d\} \neq \emptyset$	→ $S2\delta fS4$

Data Dependences in Loops

- Associate a dynamic instance to each statement. For example

```
    For i = 1 to 50
S1:   A(i) = B(i-1) + C(i)
S2:   B(i) = A(i+2) + C(i)
    EndFor
```

- Statements S1 and S2 are executed 50 times. We say S2(10) to mean the execution of S2 when i = 10
- Dependences are based on dynamic instances of statements

Data Dependences in Loops

- Unrolling loops can help one figure out dependences:

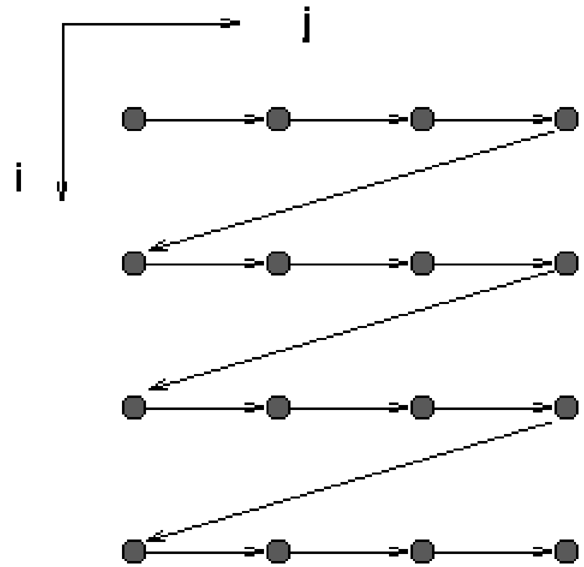
```
S1(1):      A(1) = B(0) + C(1)
S2(1):      B(1) = A(3) + C(1)
S1(2):      A(2) = B(1) + C(2)
S2(2):      B(2) = A(4) + C(2)
S1(3):      A(3) = B(2) + C(3)
S2(3):      B(3) = A(5) + C(3)
.....................

S1(50):    A(50) = B(49) + C(50)
S2(50):    B(50) = A(52) + C(50)
```

Iteration Spaces

- Nested loops define an iteration space:

```
For i = 1 to 4
   for j = 1 to 4
      A(i,j) = A(i,j) + C(j)
   Endfor
Endfor
```



- Sequential execution (traversal order):
- Dimensionality of iteration space = loop nest level; arbitrary convex shapes are allowed
- Change in order of execution is valid if no dependences are violated

Dependences in Loop Nests

L_j and U_j are affine functions of enclosing loop indices $I_1, \cdots, I_{j-1}$

```
For I1 = L1, U1
 For I2 = L2, U2
  …
   For In = Ln, Un
     S(I1,…,In)
```

There is a dependence in a loop nest if there are iterations $\vec{I} = (i_1, i_2, \cdots, i_n)$ and $\vec{J} = (j_1, j_2, \cdots, j_n)$ and some memory location M such that:

– $\vec{I}$ and $\vec{J}$ are valid iteration instances

– $\vec{I} \prec \vec{J}$, i.e., $\vec{I}$ executes before $\vec{J}$ or $\vec{I}$ precedes $\vec{J}$

– S($\vec{I}$) and S($\vec{J}$) reference M

Dependence Level

We say that $\vec{I} \prec_u \vec{J}$ (for $u = 1, \ldots, n$) if:

– $I_k = J_k$ for $k = 1, \ldots, u - 1$ and,

– $I_u < J_u$

u is defined as the level of the dependence.

If $I_k = J_k$ for all values of k, then we say that the dependence is loop independent.

If $\vec{I} \prec_u \vec{J}$, then the dependence is at level u; we also refer to this as a dependence carried by the loop i_u

Dependence Distance Vector

- Suppose there is a dependence from $S(i_1, i_2, \cdots, i_n)$ in a $S(j_1, j_2, \cdots, j_n)$
- We define the dependence ***distance vector*** as

$$(j_1 - i_1, j_2 - i_2, \cdots, j_n - i_n)$$

- Legality of transformations is defined in terms of distance vectors or approximations to distance vectors, such as direction vectors and dependence levels
- Level of a dependence is position of rightmost positive component in dependence vector

Dependence Direction Vector

- We define sign of a real number x as follows:
  - sign(x) = 0   if x=0
  - sign(x) = +   if x > 0
  - sign(x) = -   if x < 0
- Suppose we have a dependence from S(i1, …, in) to S(j1, …, jn). The dependence distance is (j1-i1, …, jn-in)
- The ***direction vector*** for this dependence is:

(sign(j1-i1), sign(j2-i2), …, sign(jn-in))

For loop nests that have loops that count up, the first (left-most) non-zero component of any dependence vector must be positive

Example of Dependences

```
    For i = 1 to 5
        For j = i to 5
S:          A(i,j) = A(i,j-3) + A(i-2,j) + A(i-1,j+2) + A(i+1,j-1)
        EndFor
    EndFor
```

RHS reference	Type	Distance Vector	Direction Vector	Level
A(i,j-3)	Flow	(0,3)	(0,+)	2
A(i-2,j)	Flow	(2,0)	(+,0)	1
A(i-1,j+2)	Flow	(1,-2)	(+,-)	1
A(i+1,j-1)	Anti	(1,-1)	(+,-)	1

Example of Dependences

```
    For i = 1 to 5
        For j = i to 5
S:          A(i,j) = A(i,j-3) + A(i-2,j) + A(i-1,j+2) + A(i+1,j-1)
        EndFor
    EndFor
```

RHS reference	Type	Distance Vector	Direction Vector	Level
A(i,j-3)	Flow	(0,3)	(0,+)	2
A(i-2,j)	Flow	(2,0)	(+,0)	1
A(i-1,j+2)	Flow	(1,-2)	(+,-)	1
A(i+1,j-1)	Anti	(1,-1)	(+,-)	1

Example of Dependences

```
    For i = 1 to 5
        For j = i to 5
S:          A(i,j) = A(i,j-3) + A(i-2,j) + A(i-1,j+2) + A(i+1,j-1)
        EndFor
    EndFor
```

RHS reference	Type	Distance Vector	Direction Vector	Level
A(i,j-3)	Flow	(0,3)	(0,+)	2
A(i-2,j)	Flow	(2,0)	(+,0)	1
A(i-1,j+2)	Flow	(1,-2)	(+,-)	1
A(i+1,j-1)	Anti	(1,-1)	(+,-)	1

Example of Dependences

```
    For i = 1 to 5
        For j = i to 5
S:          A(i,j) = A(i,j-3) + A(i-2,j) + A(i-1,j+2) + A(i+1,j-1)
        EndFor
    EndFor
```

RHS reference	Type	Distance Vector	Direction Vector	Level
A(i,j-3)	Flow	(0,3)	(0,+)	2
A(i-2,j)	Flow	(2,0)	(+,0)	1
A(i-1,j+2)	Flow	(1,-2)	(+,-)	1
A(i+1,j-1)	Anti	(1,-1)	(+,-)	1

Example of Dependences

```
    For i = 1 to 5
        For j = i to 5
S:          A(i,j) = A(i,j-3) + A(i-2,j) + A(i-1,j+2) + A(i+1,j-1)
        EndFor
    EndFor
```

RHS reference	Type	Distance Vector	Direction Vector	Level
A(i,j-3)	Flow	(0,3)	(0,+)	2
A(i-2,j)	Flow	(2,0)	(+,0)	1
A(i-1,j+2)	Flow	(1,-2)	(+,-)	1
A(i+1,j-1)	Anti	(1,-1)	(+,-)	1

Validity (Legality) of Loop Transformations

- In a program, let there be a dependence

  from some instance of statement S1 (called source)

  to some instance of statement S2 (called sink)

- A transformation of a program is said to be valid if in the transformed version, every sink executes after its corresponding source.

Transformations: Loop interchange

- Interchange: Exchanges two loops in a perfectly nested loop, also known as permutation

```
For I = 1, N
  For J = 1, M
     S(I,J)
```

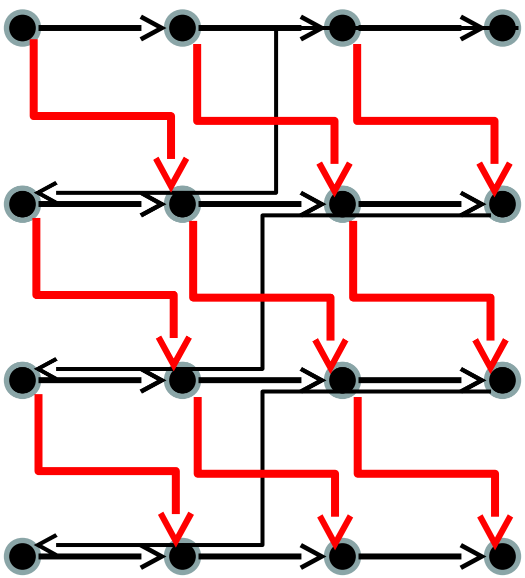
```
For J = 1, M
  For I = 1, N
     S(I,J)
```

- Interchange can improve locality
- It is not always legal; can change the dependence level, direction and distance. (d1,d2) becomes (d2,d1)

Valid Loop Interchange

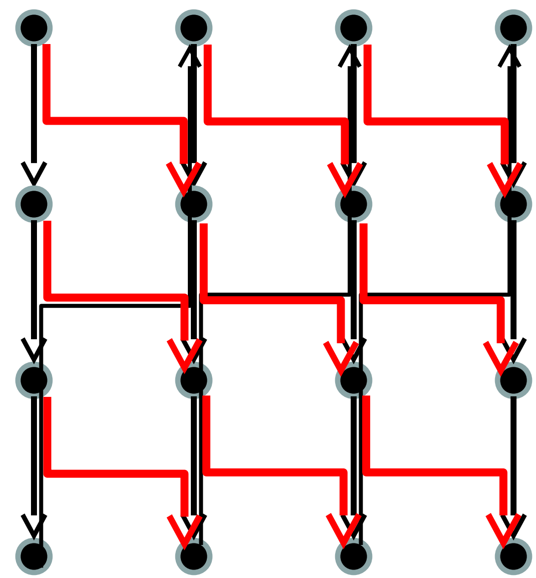
A

```
for (int i = 0; i <= n; i++)
    for (int j = 0; j <= n; j++)
        A( i, j) = A( i-1, j-1) * .9
```



B

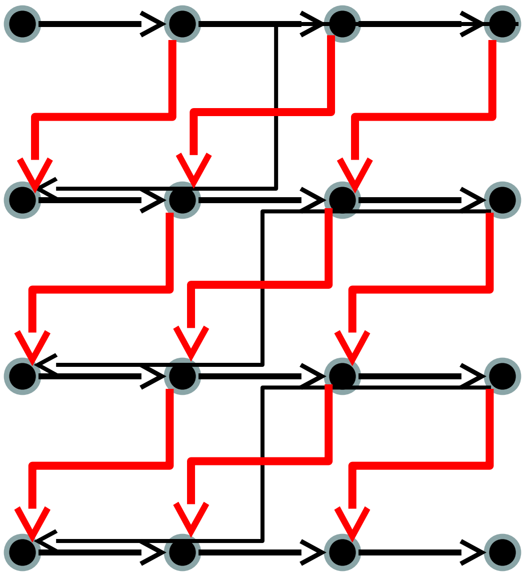
```
for (int j = 0; j <= n; j++)
    for (int i = 0; i <= n; i++)
        A( i, j) = A( i-1, j-1) * .9
```



Invalid Loop Interchange

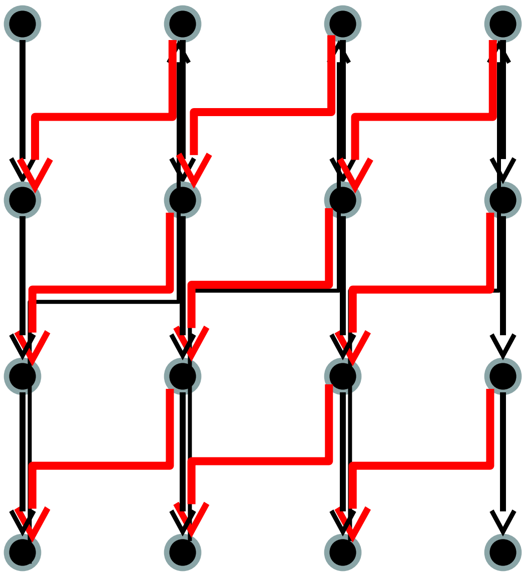
A

```
for (int i = 0; i <= n; i++)
    for (int j = 0; j <= n; j++)
        A( i, j) = A( i-1, j+1) * .9
```



B

```
for (int j = 0; j <= n; j++)
    for (int i = 0; i <= n; i++)
        A( i, j) = A( i-1, j+1) * .9
```



Validity Condition for Loop Interchange

- Loop interchange is valid for a 2D loop nest if none of the dependence vectors has any negative components
- Interchange is legal: (1,1), (2,1), (0,1), (3,0)
- Interchange is not legal: (1,-1), (3,-2)
- Equivalent view: After permutation, inner dimension becomes outer dimension and vice versa; so permute the dependence vectors of original loop and check if they are still valid dependence vectors.
- (1,1) -> (1,1): valid
- (1,-1) -> (-1,1): Not a valid dependence vector; hence loop interchange will result in violation of dependence

Validity Condition for Loop Permutation

- Generalization for higher-dimensional loops: permute all dependence vectors exactly the same way as the intended loop permutation: if any permuted vector is lex. Negative, permutation is illegal
- Example: d1 = (1,-1,1) and d2 = (0,2,-1)
- ijk -> jik? (1,-1,1) -> (-1,1,1): illegal
- ijk -> kij? (0,2,-1) -> (-1,0,2): illegal
- ijk -> ikj? (0,2,-1) -> (0,-1,2): illegal
- No valid permutation:
  - j cannot be outermost loop (-1 component in d1)
  - k cannot be outermost loop (-1 component in d2)

Validity Condition for Loop Unroll/Jam

- Sufficient condition can be obtained by observing that complete unroll/jam of a loop is equivalent to a loop permutation that moves that loop innermost, without changing order of other loops
- If such a loop permutation is valid, unroll/jam of the loop is valid
- Example: 4D loop ijkl; d1 = (1,-1,0,2), d2 = (1,1,-2,-1)
  - i: d1-> (-1,0,2,1) => invalid to unroll/jam
  - j: d1-> (1,0,2,-1); d2 -> (1,-2,-1,1) => valid to unroll/jam
  - k: d1 -> (1,-1,2,0); d2 -> (1,1,-1,-2) => valid to unroll/jam
  - l: d1 and d2 are unchanged; innermost loop always unrollable

Validity Condition for Loop Distribution

- Sufficient (but not necessary) condition: A loop with two statements can be distributed if there are no dependences from any instance of the later statement to any instance of the earlier one. Generalizes to more statements.
- Example: Loop distribution is not valid (executing all S1 first and then all S2)

```
    For I = 1, N
S1:  A(I) = B(I) + C(I)
S2:  E(I) = A(I+1)* D(I)
    EndFor
```

- Example: Loop distribution is valid

```
    For I = 1, N
S1:  A(I) = B(I) + C(I)
S2:  E(I) = A(I-1)* D(I)
    EndFor
```

Validity Condition for Tiling

- A contiguous band of loops can be tiled if they are fully permutable
- A band of loops is fully permutable of all permutations of the loops in that band are legal
- Example: d = (1,2,-3)
  - Tiling all three loops ijk is not valid, since the permutation kij is invalid ((-3,1,2) is lex. –ve)
  - 2D tiling of band ij is valid (jik: (2,1,-3) is lex. +ve)
  - 2D tiling of band jk is valid (ikj: (1,-3,2) is lex. +ve)

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
   for(k = 0;k < n; k++)
        Loop_body(I,j,k)
```

→

```
for (it = 0; it < n; it+=T)
 for (jt = 0; tj < n; j+=T)
  for (i = it; i < it+T; i++)
   for (j = jt; j < jt+T; j++)
    for(k = 0;k < n; k++)
        Loop_body(I,j,k)
```

Transformations: Loop Parallelization

- A loop in a loop nest can be executed in parallel if it does not carry a dependence

```
For I = 1, N
  For J = 1, M
     A(I,J)= A(I-1,J)
```

- The dependence distance is (1,0); the outer (I) loop carries the dependence.
- The inner (J) loop does not carry any dependences

```
For I = 1, N
  ForPAR J = 1, M
    A(I,J)= A(I-1,J)
```

Examples of Parallelization

```
For I = 1, 10
   A(I) = A(I) + 5
```

Parallel

```
For I = 1, 10
   A(I) = A(I-1) + 5
```

Not parallel

```
For I = 1, 10
   A(I) = A(I-10) + 5
```

Parallel (Why?)