CS 314 Principles of Programming Languages

Lecture 20: Parallelism and Dependence Analysis

Prof. Zheng Zhang

Rutgers University

November 21, 2018

Class Information

- Project 2 deadline is extended to 11/25 Sunday.
- Midterm grades will be released immediately after Thanksgiving break.

Review: Parallelizing Affine Loops

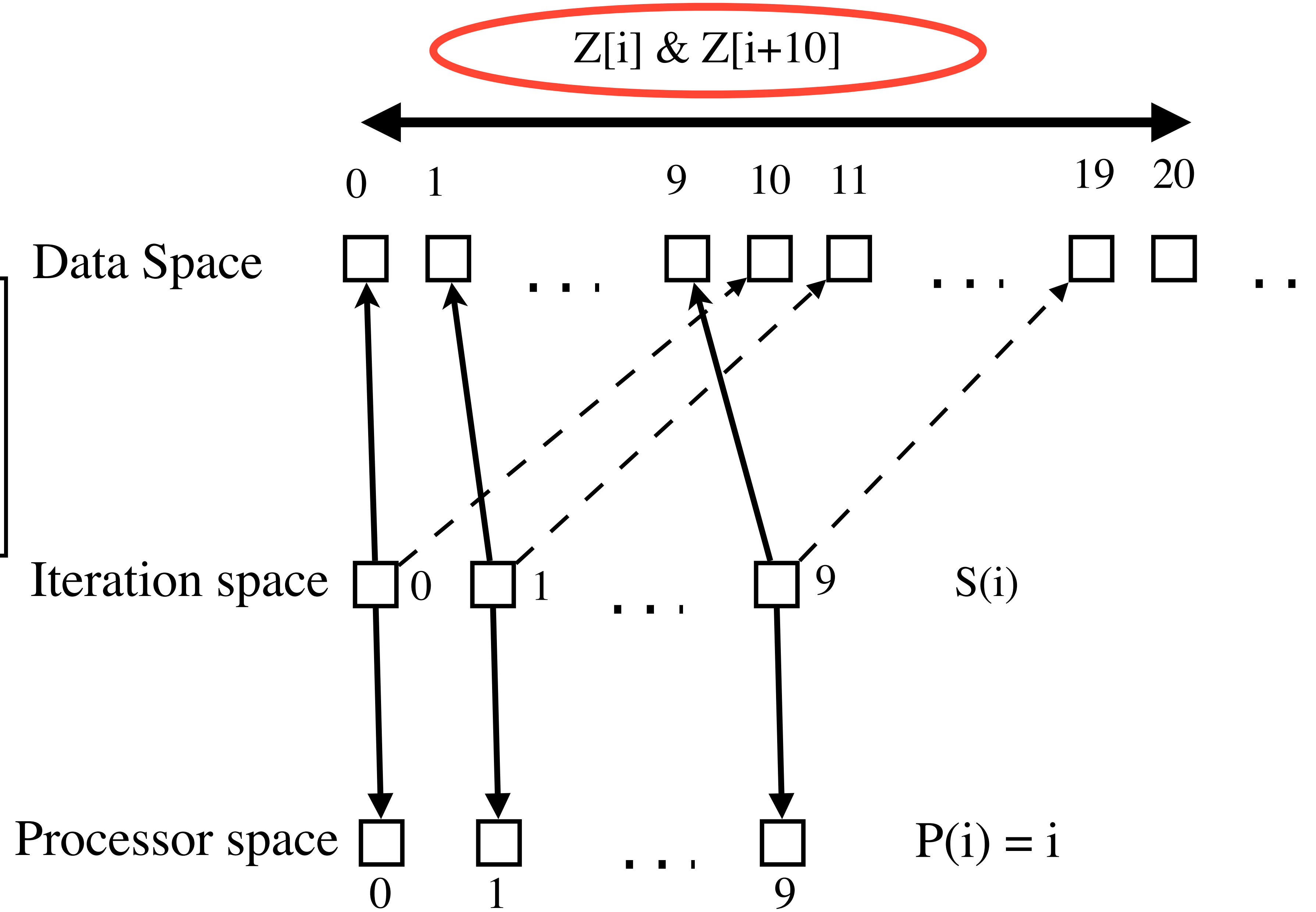
Three spaces

- Iteration space
  - The set of dynamic execution instances
    For instance, the set of value vectors taken by loop indices
  - A k-dimensional space for a k-level loop nest
- Data space
  - The set of array elements accessed
  - An n-dimensional space for an n-dimensional array
- Processor space
  - The set of processors in the system
  - In analysis, we may pretend there are unbounded # of virtual processors

Three Spaces

- **Iteration space, data space, and processor space**

```
float Z[100];
for (i=0; i<10; i++)
  S: Z[i+10] = Z[i];
```



Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

Example:

```
do i = 1, N
    do j = 1, N
       S1: A[i, j] = A[i, j - 1]
```

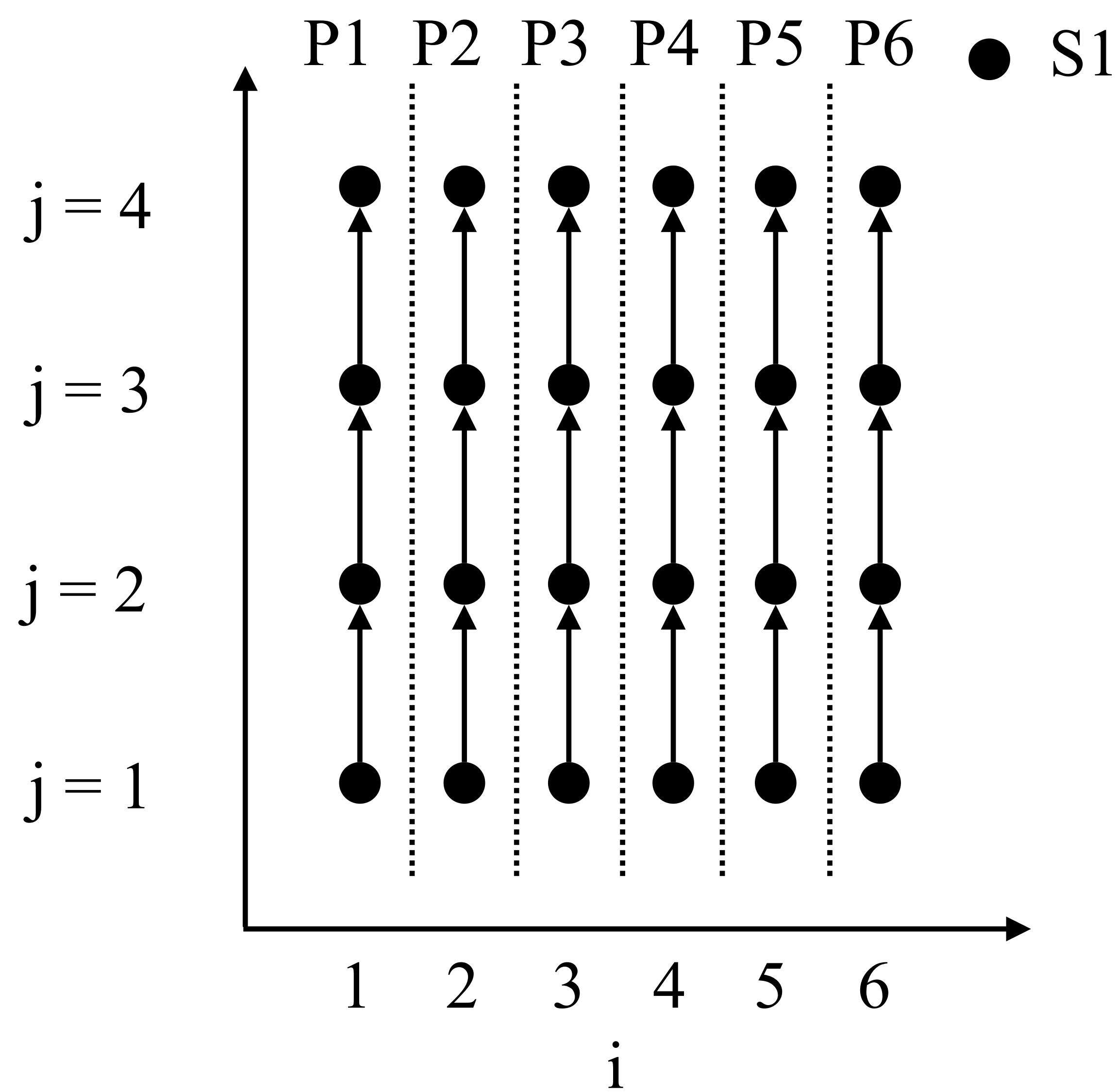
Write *in $S_1(1,1)$ to* ***Read*** *in $S_1(1,2)$*

↓

Write *in $S_1(i, j)$ to* ***Read*** *in $S_1(i, j+1)$*

Dependence from **S1** to **S1**

Communication is limited to the iterations within one processor.



Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

Example 1:

```
do i = 1, N
    do j = 1, N
       S1: A[i, j] = A[i, j - 1]
```

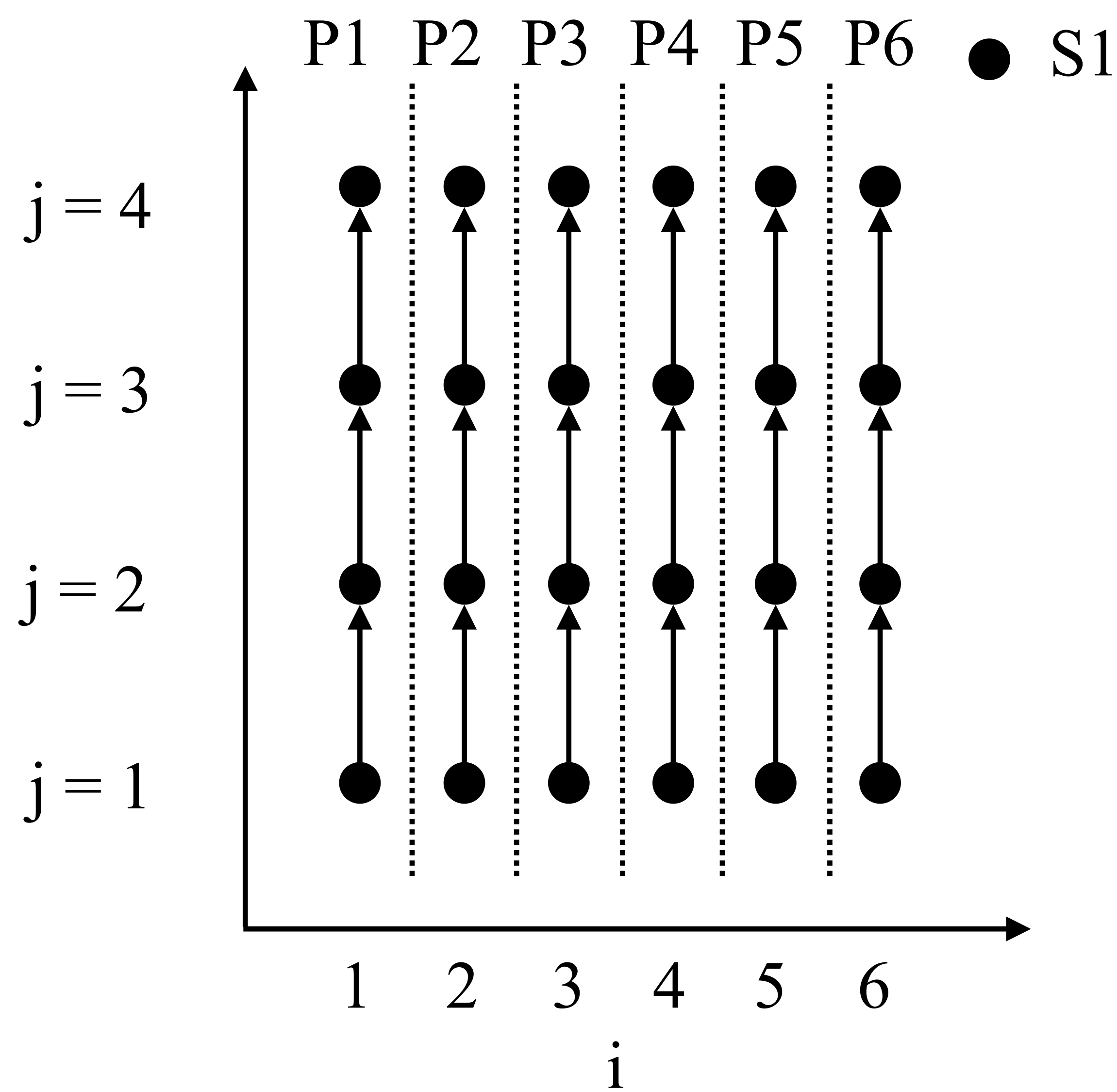
Write *in* $S_1(1,1)$ *to* ***Read*** *in* $S_1(1,2)$

↓

Write *in* $S_1(i, j)$ *to* ***Read*** *in* $S_1(i, j+1)$

Which loop can be parallelized?
The "i" loop or the "j" loop?

Answer: the "i" loop

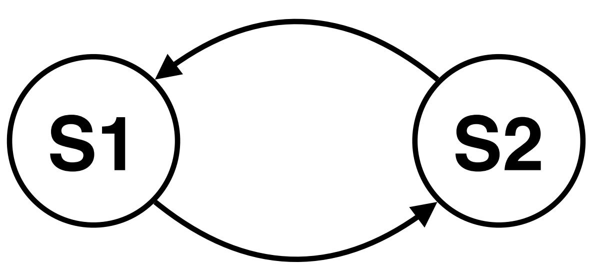


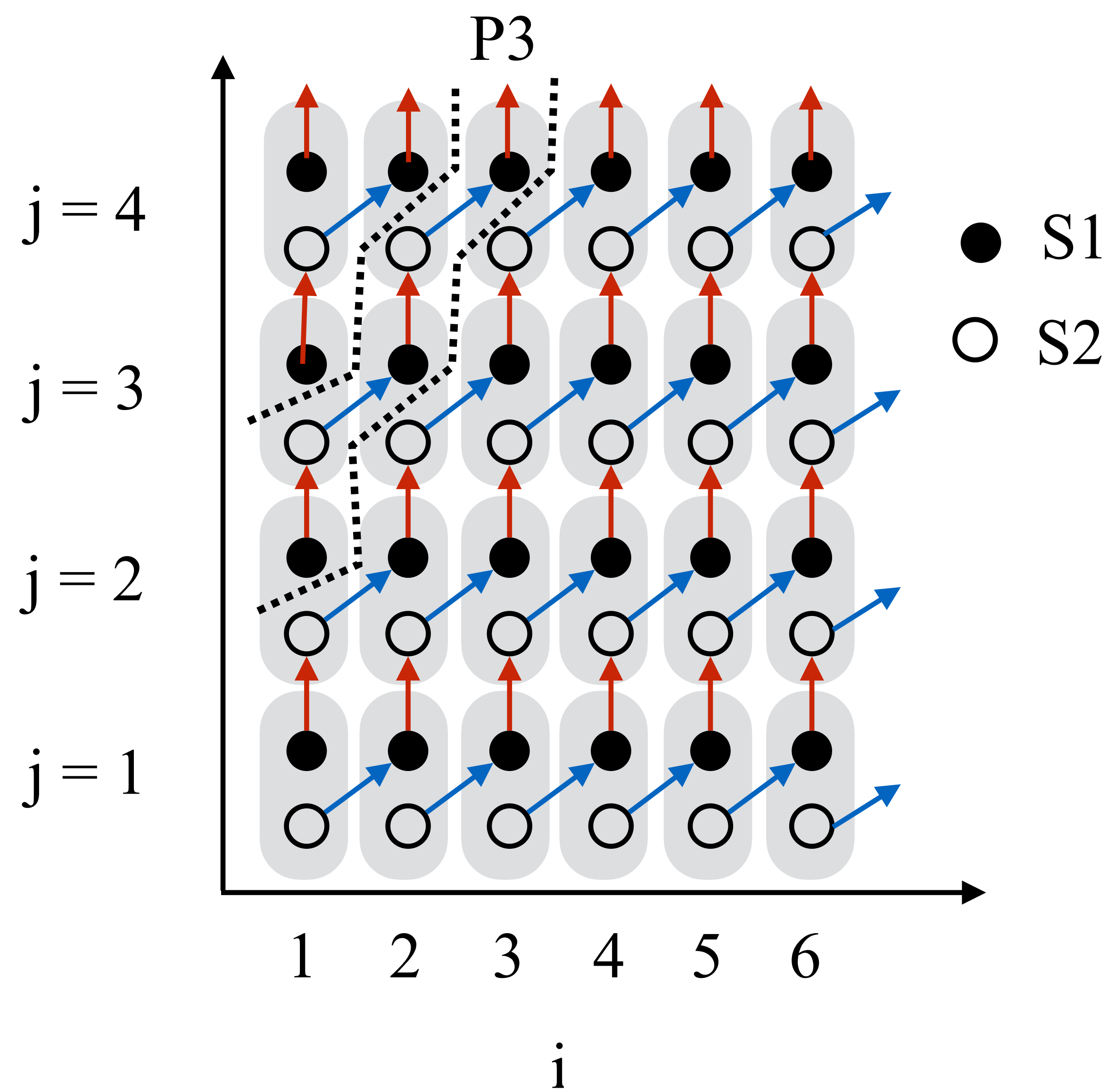
Synchronization-free Parallelism

Parallelize an application **without** allowing any *communication* or *synchronization* among (logical) processors.

<u>Example 2</u>:

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```





Dependence from **S1**(1,1) to **S2**(1,2)

Dependence from **S2**(1,1) to **S1**(2,1)

Review — Processing Space: Affine Partition Schedule

- Map an iteration to a processor using < **C**, d >

  C is a *n by m* matrix

  - m = d (the loop level)
  - n is the dimension of the processor grid

  $\vec{d}$ is a n-element constant vector

  $\vec{p} = C\vec{x} + \vec{d}$, where $\vec{x}$ is an iteration vector

Review: Processing Space: Affine Partition Schedule

- Map an iteration to a processor using < **C**, d >

$$\vec{p} = \mathbf{C}\,\vec{x} + \vec{d}, \textbf{ where } \vec{x} \textbf{ is an iteration vector}$$

- Example

```
for (i=1; i<=N; i++)
  S: Y[i] = Z[i];
```

$\mathbf{C} = [1], \mathbf{d} = [0]$

$\vec{p}(\ S(i)\ ) = 1*i + 0$

$= i$

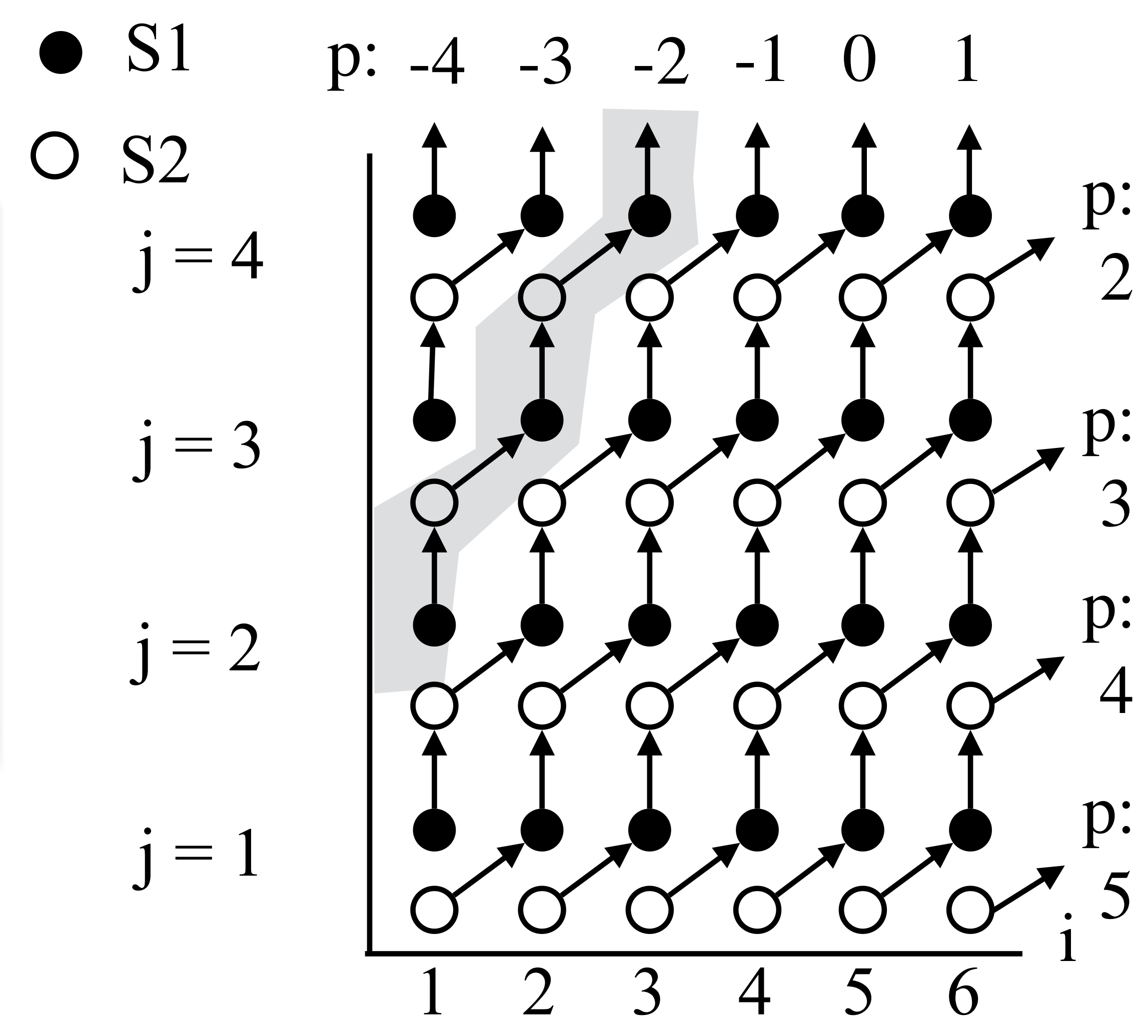
Map iteration **i** to Processor **i**

Review: Synchronization-free Parallelism

Example:

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        S1: X[i,j] = X[i,j] + Y[i-1, j];
        S2: Y[i,j] = Y[i,j] + X[i, j-1];
    }
```

$C_{11} = C_{21} = -C_{22} = -C_{12} = d_2 - d_1$



One Potential Solution:

Affine schedule for S1, p(S1): $[C_{11}\ C_{12}] = [1\ -1]$, $d_1 = -1$

i.e. (i, j) iteration of S1 to processor p = i - j - 1;

Affine schedule for S2, p(S2): $[C_{21}\ C_{22}] = [1\ -1]$, $d_2 = 0$

i.e. (i, j) iteration of S2 to processor p = i - j.

Code Generation

```
for (i=1; i<=6; i++)
   for (j=1; j<=4; j++){
      X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
      Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
   }
```

S1(i, j): processor p = i-j-1;
S2(i, j): processor p = i-j.

⇓

```
forall (p=-4; p<=5; p++)
   for (i=1; i<=6; i++)
      for (j=1; j<=4; j++){
         if (p== i-j-1)
            X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
         if (p== i-j)
            Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
      }
```

- Step 1: find processor ID ranges
  - S1: $-4 <= p <= 4$
  - S2: $-3 <= p <= 5$
  - Union: $-4 <= p <= 5$
- Step 2: generate code

Naive Code Generation

```
forall (p=-4; p<=5; p++)
    for (i=1; i<=6; i++)
        for (j=1; j<=4; j++){
            if (p== i-j-1)
                X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
            if (p== i-j)
                Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
        }
```

What are the issues with this code?

- Wider than necessary loop bounds
- Redundant tests in loop body

Remove Idle Iterations

Loop bounds are wider than they should have been

For example, when p=-4, only 1 of the 24 iterations has useful operations, i = 1, j =4 .

```
forall (p=-4; p<=5; p++)
   for (i=1; i<=6; i++)
      for (j=1; j<=4; j++){
         if (p== i-j-1)
            X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
         if (p== i-j)
            Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
      }
```

Make Loop Bounds Tighter

$-4 <= p <= 5$
$1 <= i <= 6$
$1 <= j <= 4$
$i-p-1=j$

↓ Fourier-Motzkin Elimination

S1

j: $i-p-1 <= j <= i-p-1$
$1 <= j <= 4$
i: $p+2 <= i <= p+5$ ← **Eliminate j**
$1 <= i <= 6$

Fourizer Motzkin Elimination

Eliminating variable z in inequality systems

- Match each of the lower bounds on z with each of its upper bounds
- Equivalent to projecting a polyhedron into reduced dimension space

-4 <= p <= 5
1<= i <= 6
1<= j <= 4
i-p-1=j

Suppose we want to eliminate j:

i-p-1<= j <= i-p-1
1<= j <= 4

i-p-1 <= i-p-1
i-p-1 <= 4

1 <= i-p-1
1 <= 4

p+2 <= i <= p+5

Make Loop Bounds Tighter

-4 <= p <= 5 1<= i <= 6 1<= j <= 4 i-p-1=j	-4 <= p <= 5 1<= i <= 6 1<= j <= 4 i-p=j

↓ Fourier-Motzkin Elimination ↓

S1

j:	i-p-1<= j <= i-p-1
	1<=j <= 4
i:	p+2<=i <= p+5 ← **Eliminate j**
	1<=i <= 6
p:	-4<= p<= 4 ← **Eliminate i**

S2

j:	i-p<=j <= i-p
	1<=j <=4
i:	p+1<=i <= 4+p
	1<=i <=6
p:	-3<= p <= 5

Make Loop Bounds Tighter

S1

j: i-p-1<= j <= i-p-1
1<=j <= 4
i: p+2<=i <= p+5
1<=i <= 6
p: -4<= p<= 4

S2

j: i-p<=j <= i-p
1<=j <=4
i: p+1<=i <= 4+p
1<=i <=6
p: -3<= p <= 5

Union result:
j: i-p-1<=j <= i-p
1<=j <=4
i: p+1<=i <= 5+p
1<=i <=6
p: -4<= p <= 5

Make Loop Bounds Tighter

```
forall (p=-4; p<=5; p++)
   for (i=1; i<=6; i++)
      for (j=1; j<=4; j++){
         if (p== i-j-1)
            X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
         if (p== i-j)
            Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
      }
```

Union result:

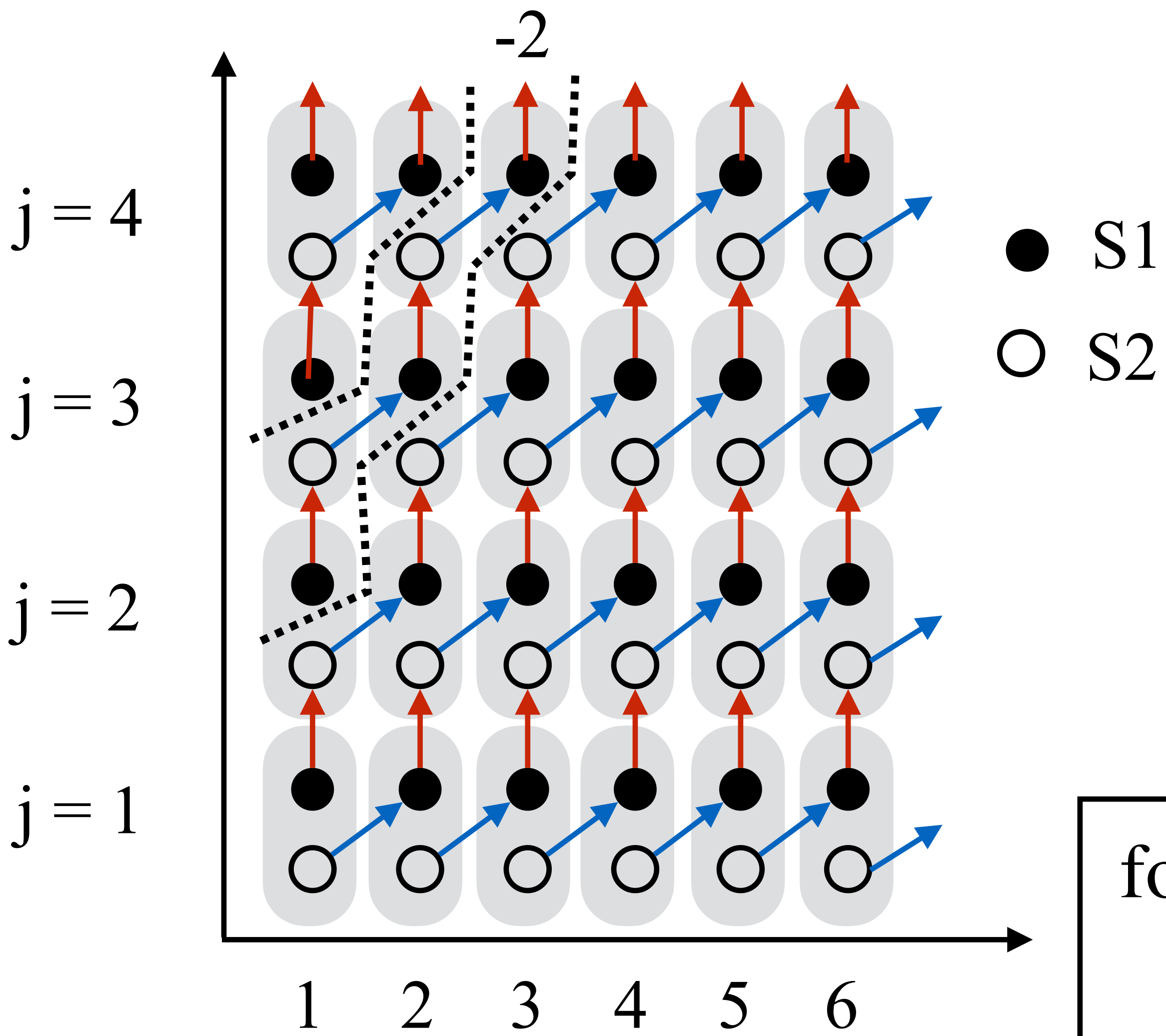
j: i-p-1<=j <= i-p
   1<=j <=4

i: **p+1**<=i <= 5+p
   1<=i <=6

p: -4<= p <= 5

```
for (p=-4; p<=5; p++)
   for (i=max(1,p+1); i<=min(6,5+p); i++)
      for (j=max(1,i-p-1); j<=min(4,i-p); j++){
         if (p== i-j-1)
            X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
         if (p== i-j)
            Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
      }
```

Make Loop Bounds Tighter



When p = -2,
i: [1, 3]

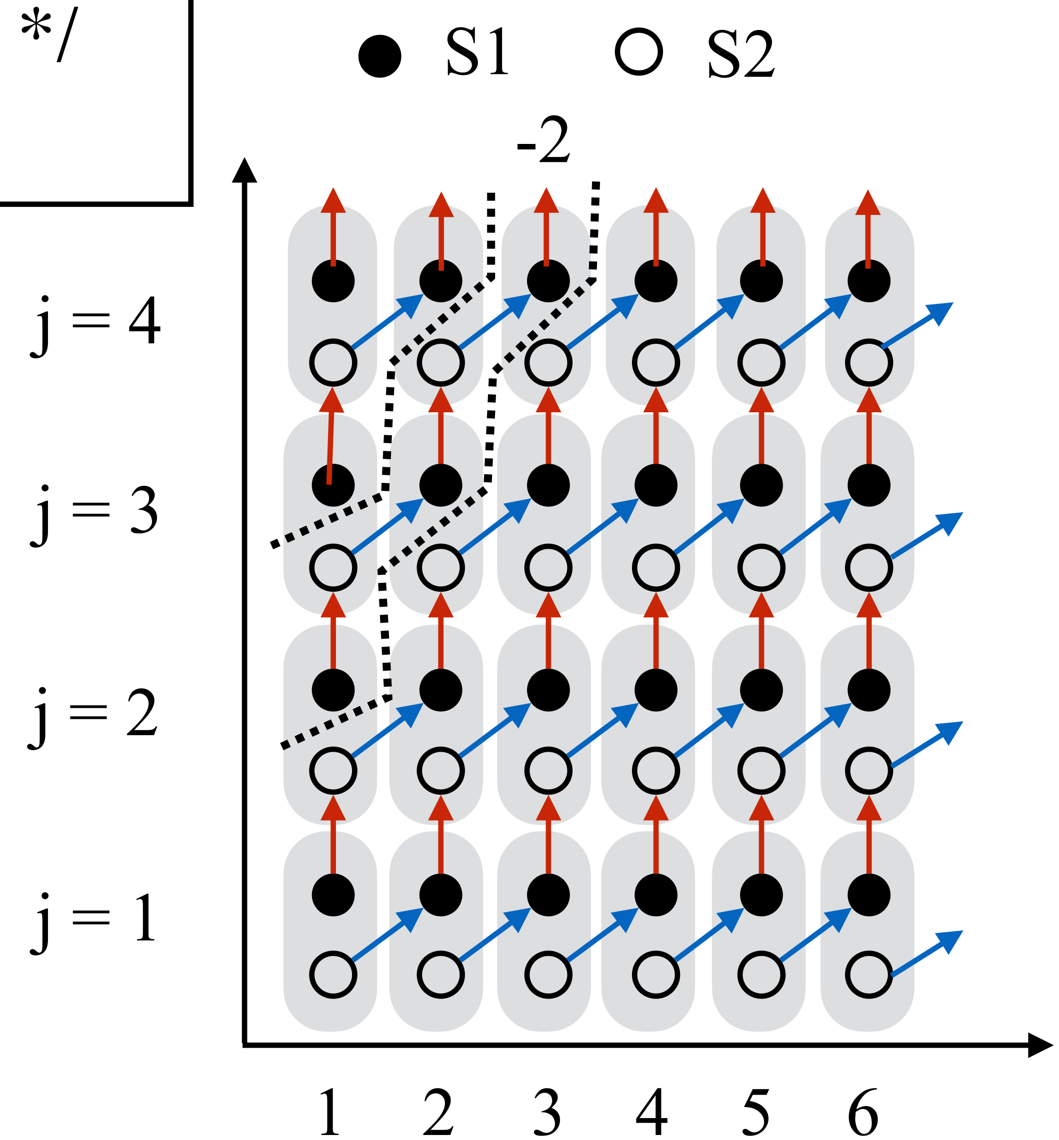
```
for (p=-4; p<=5; p++)
    for (i=max(1,p+1); i<=min(6,5+p); i++)
        for (j=max(1,i-p-1); j<=min(4, i-p); j++)
        {
            if (p== i-j-1)
                X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
            if (p== i-j)
                Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
        }
```

Remove Tests

```
for (p=-4; p<=5; p++)
   for (i=max(1,p+1); i<=min(6,5+p); i++)
      for (j=max(1,i-p-1); j<=min(4,i-p); j++){
         if (p== i-j-1)
            X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
         if (p== i-j)
            Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
      }
```

↓?

```
for (p=-4; p<=5; p++)
   for (i=max(1,p+1); i<=min(6,5+p); i++)
      j=i-p-1;
      X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
      j=i-p;
      Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
   }
```



Remove Tests

- **Reason for the tests**
  - The iteration spaces of statements intersect but do not completely overlap
- **Solution**
  - Split the iteration space at the boundaries of overlapping polyhedra.
  - Generate code for each of the subspaces.

Remove Tests

```
/*space 1*/
p=-4;   i=1;   j=4;
X[i,j]=X[i,j]+Y[i-1,j];   /*S1*/
```

```
/*space 2*/
for (p=-3; p<=4; p++)
    for (i=max(1,p+1); i<=min(6,5+p); i++)
        for (j=max(1,i-p-1); j<=min(4,i-p); j++){
            if (p== i-j-1)
                X[i,j] = X[i,j] + Y[i-1, j];    /* S1
                */
            if (p== i-j)
                Y[i,j] = Y[i,j] + X[i, j-1];    /* S2
                */
        }
```

```
/*space 3*/
p=5;   i=6;   j=1;
Y[i,j] = X[i,j-1] + Y[i,j];  /*S2*/
```

Split on "p":
- subspace 1: p = -4;
- subspace 2: -3 <= p <= 4;
- subspace 3: p = 5;

S1:

j: i-p-1<= j <= i-p-1
     1<=j <=4
i: p+2<=i <=5+p
     1<=i <= 6
p: **-4** <= p <= 4

S2:

j: i-p<=j <= i-p
     1<=j <=4
i: p+1<=i <= 4+p
     1<=i <=6
p: -3 <= p <= **5**

Remove Tests

```
/*space 1*/
p=-4;   i=1;   j=4;
X[i,j]=X[i,j]+Y[i-1,j];   /*S1*/
```

```
/*space 2*/
for (p=-3; p<=4; p++)
    for (i=max(1,p+1); i<=min(6,5+p); i++)
        for (j=max(1,i-p-1); j<=min(4,i-p); j++){
            if (p== i-j-1)
                X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
            if (p== i-j)
                Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
        }
```

```
/*space 3*/
p=5;   i=6;   j=1;
Y[i,j] = X[i,j-1] + Y[i,j];  /*S2*/
```

S1:

j: $i-p-1 <= j <= i-p-1$

$1<=j <=4$

i: $p+2<=i <=5+p$

$1<=i <= 6$

p: **-4** $<= p <= 4$

S2:

j: $i-p<=j <= i-p$

$1<=j <=4$

i: $p+1<=i <= 4+p$

$1<=i <=6$

p: $-3 <= p <=$ **5**

Remove Tests

p>=-3; p<=4

Split on "i":

subspace 2a: max(1, p+1)<= i < max(1, p+2); **only S2;**

subspace 2b: max(1, p+2) <= i <= min(6, 4+p); **both S1 and S2;**

subspace 2c: min(6, 4+p) < i <= min(5+p, 6); **only S1;**

S1:

j:	i-p-1<= j <= i-p-1
	1<=j <=4
i:	p+2<=i <=5+p
	1<=i <= 6
p:	-4 <= p <= 4

S2:

j:	i-p<=j <= i-p
	1<=j <=4
i:	p+1<=i <= 4+p
	1<=i <=6
p:	-3 <= p <= 5

Remove Tests

Split on "i":

subspace 2a: max(1,p+1)<= i < max(1, p+2);

subspace 2b: max(1,p+2) <= i <= min(6, 4+p);

subspace 2c: min(6, 4+p) < i <= min(5+p, 6).

```
/*space 2*/
for (p=-3; p<=4; p++)
    for (i=max(1,p+1); i<=min(6,5+p); i++)
        for (j=max(1,i-p-1); j<=min(4,i-p); j++){
            if (p== i-j-1)
                X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
            if (p== i-j)
                Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
        }
```

```
/*space 2*/
for (p=-3;  p<=4; p++){
  /*space 2a*/
  if (p>=0){
      i = p+1; j = 1;
      Y[i,j] = Y[i,j] + X[i, j-1];/* S2 */}
  /*space 2b*/
  for (i=max(1, p+2); i<=min(6, 4 +p); i++)
   {
   j=i-p-1;
   X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
   j=i-p
   Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */   }
  /*space 2c*/
  if (p<=1){
     i=5+p;   j=5;
     X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */}
}
```

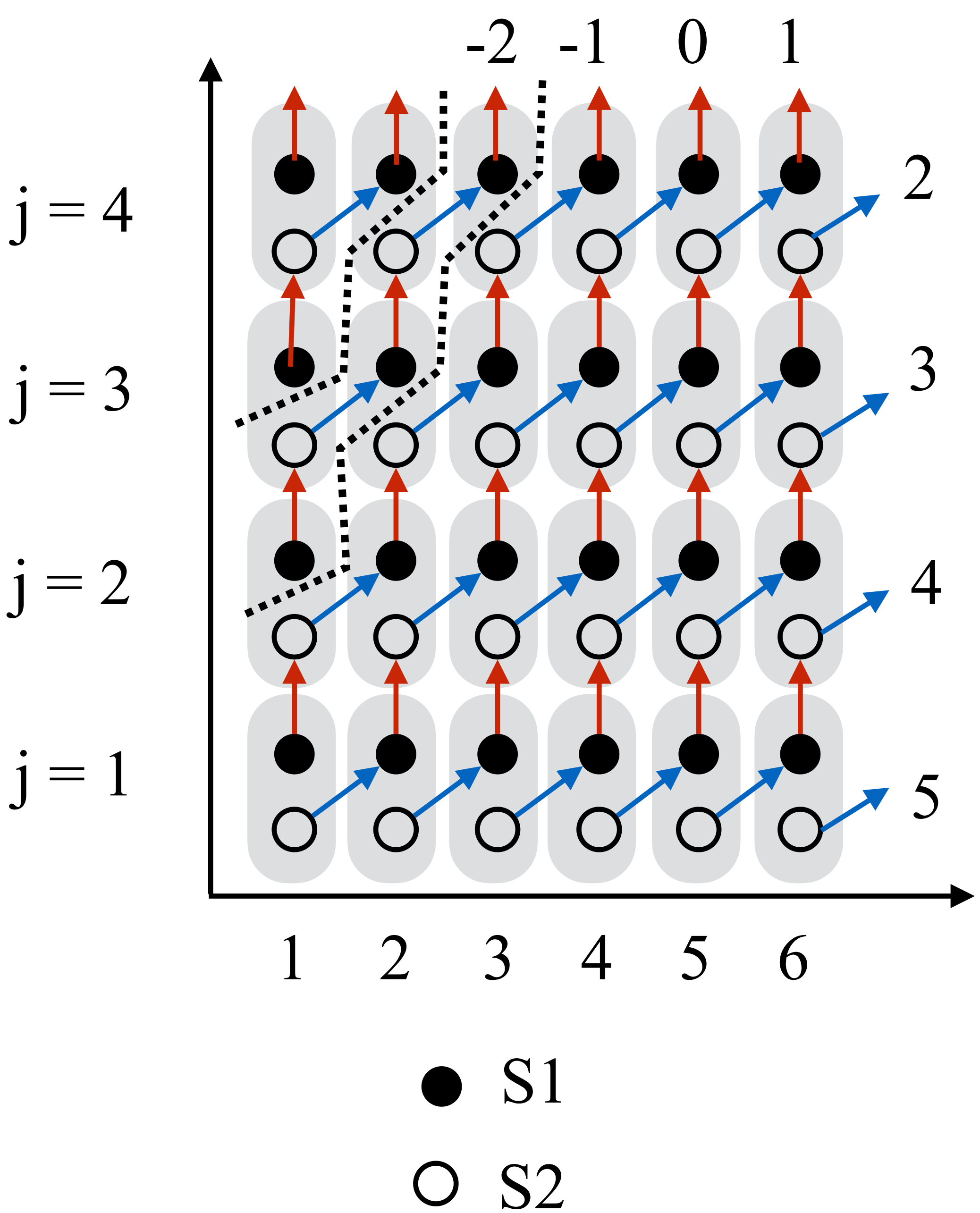
Remove Tests

Split on "i":

subspace 2a: max(1,p+1)<= i < max(1, p+2);

subspace 2b: max(1,p+2) <= i <= min(6, 4+p);

subspace 2c: min(6, 4+p) < i <= min(5+p, 6).




```
/*space 2*/
for (p=-3;  p<=4; p++){
  /*space 2a*/
  if (p>=0){
      i = p+1; j = 1;
      Y[i,j] = Y[i,j] + X[i, j-1];/* S2 */}
  /*space 2b*/
  for (i=max(1, p+2); i<=min(6, 4 +p); i++)
{
     j=i-p-1;
     X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
     j=i-p
     Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */   }
   /*space 2c*/
   if (p<=1){
      i=5+p;   j=5;
      X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */}
}
```

Code Generation and Optimization Summary

```
for (i=1; i<=100; i++)
    for (j=1; j<=100; j++){
        X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
        Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
    }
```

```
forall (p=-4; p<=5; p++)
    for (i=1; i<=6; i++)
        for (j=1; j<=4; j++){
            if (p== i-j-1)
                X[i,j] = X[i,j] + Y[i-1, j];    /* S1 */
            if (p== i-j)
                Y[i,j] = Y[i,j] + X[i, j-1];    /* S2 */
        }
```

```
/*space 1*/
if ( p == -4 )
  X[1,4]=X[1,4]+Y[0,4];   /*S1*/
/*space 2*/
for (p=-3;  p<=4; p++){
  /*space 2a*/
  if (p>0)
     Y[p+1,1] = Y[p+1,1] + X[p+1, 0];/* S2 */
  /*space 2b*/
  for (i=max(1,p+2); i<=min(6,4+p); i++)
    X[i,i-p-1] = X[i,i-p-1] + Y[i-1, i-p-1];    /* S1 */
    Y[i,i-p] = Y[i,i-p] + X[i, i-p-1];    /* S2 */   }
  /*space 2c*/
  if (p<=-1)
      X[5+p,5] = X[5+p,5] + Y[4+p, 5];    /* S1 */
}
/*space 3*/
if (p = =5)
  Y[6,1] = X[6,0] + Y[6,1];  /*S2*/
```

Next Class

Reading

- Scott, Chapter 7.2; ALSU Chapter 6.5