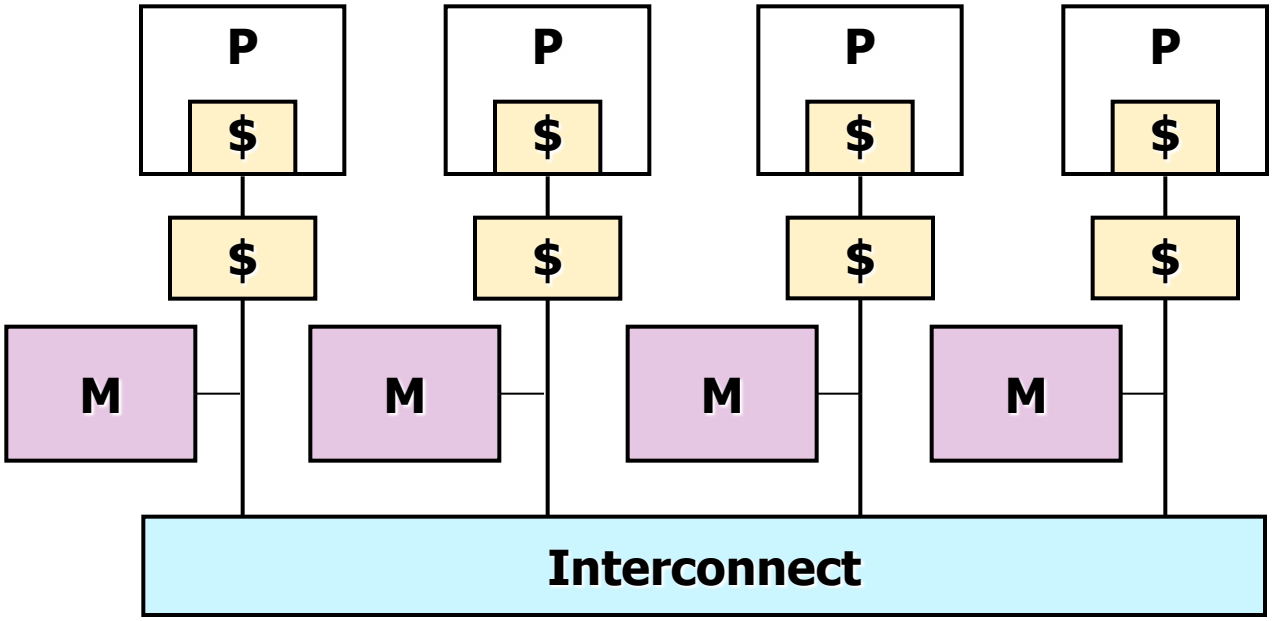
Lecture 10
Loop Transformations for Communication-Free Parallelism

1. Examples of Loop Transformations
2. Affine Partitioning
3. Code Generation

Readings: Chapter 11–11.3, 11.6–11.7.4, 11.9-11.9.6

Shared Memory Machines

Performance on Shared Address Space Multiprocessors:
Parallelism & Locality



Concept Development

Two major shifts in paradigms:

- Optimizing parallelism → Optimizing locality
- Loop transformations: discovery → heuristics → mathematics

Parallelism and Locality

- Parallelism DOES NOT necessarily imply speed up!
- The key is to optimize locality
  - Multiprocessor performance:
    - Operations using the same data are executed on the same processor
    - This results in parallelism with minimum communication
  - Sequential processor performance:
    - Minimize cache misses
    - Operations using the same data are executed close in time
- Shift focus from finding "independent operations" to finding "dependent operations"

Origin of Loop Transformations

Discovery: Many different loop transformations are useful (Kuck et al.)

- Loop interchange: inner → outer; outer → inner loop
- Loop reversal: run iterations backwards
- Loop skewing: add a constant to the loop index of the inner loop
- Loop fusion: take two loops and fuse into 1
- Loop fission: take one loop and split into multiple

Key question: how to combine these to optimize parallelism, locality?

Heuristics → Mathematics

Heuristics: Many papers on source-to-source transforms (Code → Code)

Mathematics

- **Parallel algorithm research**
  - Systolic Arrays: invented by Kung and Leiserson, 1978
    - Over 100 papers published by many mathematicians
  - Geometric (matrix) transforms to map computation onto VLSI while minimizing communication
- **Applying matrix transforms to programs** (parallelism & locality)
  - Matrix transformation approach
    (Code→ matrices → matrices → Code)
  - "A Data Locality Optimization Algorithm" for uni & multiprocessors
    - Michael Wolf & Monica Lam, PLDI 1991;
      Most influential paper award, 2001
  - Affine transforms: Amy Lim and Monica Lam, 1998, 1999
    - Built on 4 theses
      (Michael Wolf, Jennifer Anderson, Saman Amarasinghe, Amy Lim)

Lecture Outline

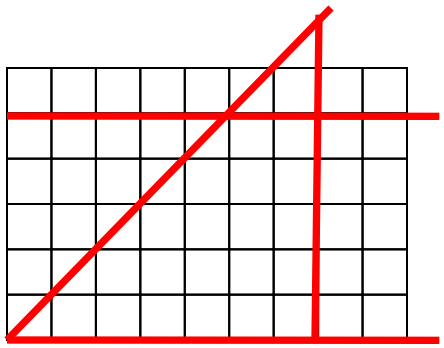
1. Problem formulation with matrix transforms

2. Affine transforms for communication-free parallelism

Affine transforms for parallelism with minimum communication (next class)

Iteration Space

```
FOR i = 0 to 5
   FOR j = i to 7
      …
```



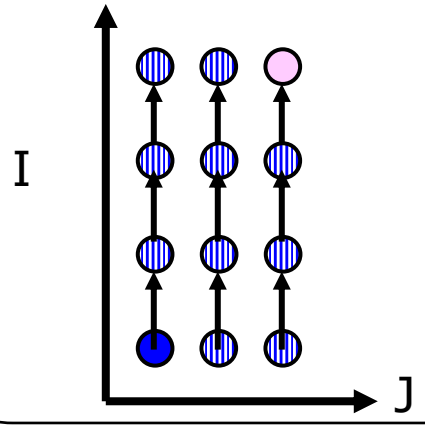
- n-deep loop nests: n-dimensional polytope
- Iterations: coordinates in the iteration space
- Assume: iteration index is incremented in the loop
- Sequential execution order: lexicographic order
  - [0,0], [0,1], …, [0,6], [0,7],
    [1,1], …, [1,6], [1,7], …

- A loop transform is legal
  - if all data dependences in the original loop nest are honored with sequential execution in the new loop.

Example 1

(a)

```
for I = 1 to 4
   for J = 1 to 3
      Z[I,J] = Z[I-1,J]
```



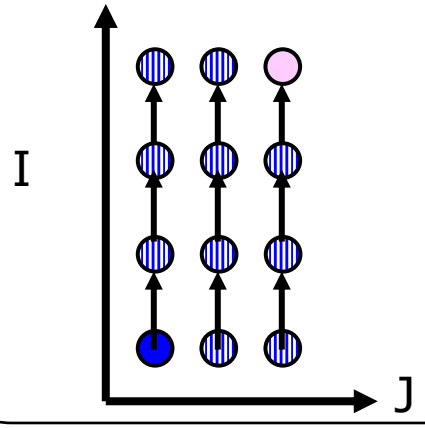
Quiz

1. Is the inner or outer loop a doall loop?
2. How to parallelize for a multiprocessor?
3. Data locality on a uniprocessor, or on a multiprocessor?
4. What is the expected performance?

Transform 1: Loop Permutation (Loop Interchange)

(a)

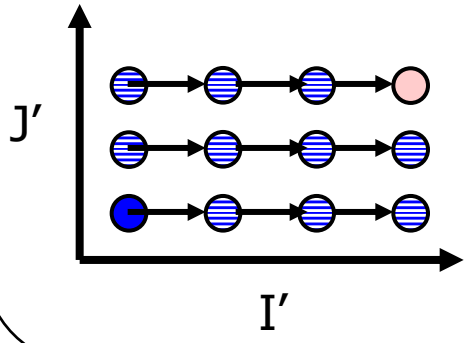
```
for I = 1 to 4
  for J = 1 to 3
    Z[I,J] = Z[I-1,J]
```



$$\begin{bmatrix} j' \\ i' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

(b)

```
for J' = 1 to 3
  for I' = 1 to 4
    Z[I',J'] = Z[I'-1,J']
```



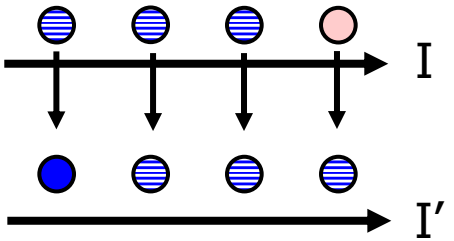
Quiz

1. Is the transform legal?
2. Is the inner loop or outer loop a doall loop in (b)?
3. Is (b) good for uniprocessor locality?
4. For multiprocessor parallelism & locality?

Key: Assign dependent operations on the same processor and close in time.

Transform 2: Loop Fusion

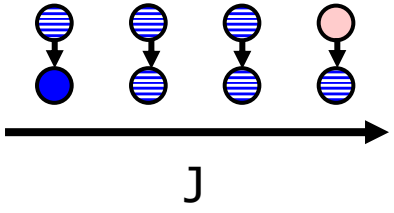
```
for I = 1 to 4
    T[I]= A[I]+B[I]     (s1)
for I' = 1 to 4
    C[I']= T[I'] x T[I'] (s2)
```



s1: $\begin{bmatrix} j \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} i \end{bmatrix}$

s2: $\begin{bmatrix} j \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} i' \end{bmatrix}$

```
for J = 1 to 4
    T[J]= A[J]+B[J]  (s1)
    C[J]= T[J] x T[J] (s2)
```



Loop Transformations

- Many loop transforms have been proposed for parallelism & locality
- *n*-deep loop nest transforms → *n*-dimensional matrix transforms
  - Interchange: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
  - Skewing: $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$
  - Reversal: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
  - Matrix multiplication of the above
- Cross statement transforms
  - Loop fusion: combining loops
  - Loop fission: separating loops
  - Re-indexing: shifting operations to different iterations in a loop

Loop Transformations

- It is easy to check if a transform is legal
- But which combinations of transformations should we apply
  - Many heuristics for many years!
- Key idea:
  - Unify them with affine transforms and solve for the unknowns!

Lecture Outline

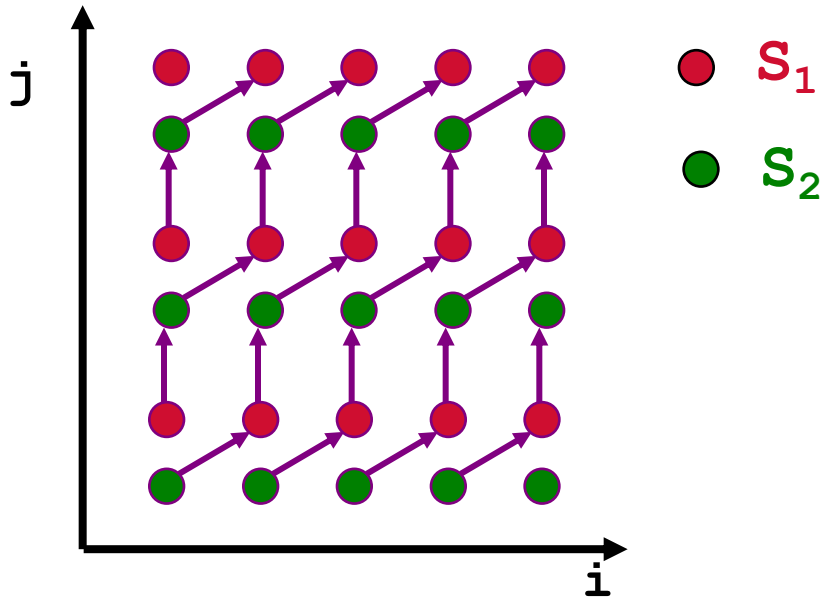
1. Problem formulation with matrix transforms

2. Affine transforms for communication-free parallelism

Affine transforms for parallelism with minimum communication (next class)

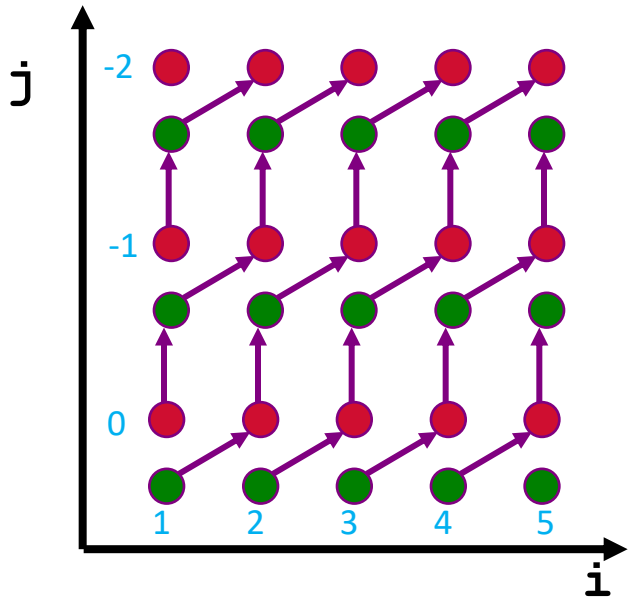
2. Affine Partitioning: A Contrived but Illustrative Example

```
FOR j = 1 TO n
   FOR i = 1 TO n
      A[i,j] = A[i,j]+B[i-1,j];          (S1)
      B[i,j] = A[i,j-1]*B[i,j];          (S2)
```



2. Affine Partitioning: A Contrived but Illustrative Example

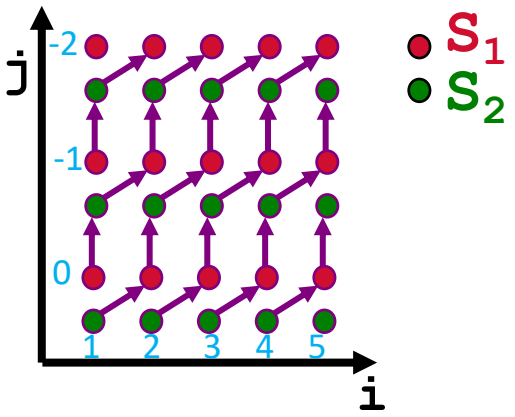
```
FOR j = 1 TO n
   FOR i = 1 TO n
      A[i,j] = A[i,j]+B[i-1,j];          (S1)
      B[i,j] = A[i,j-1]*B[i,j];          (S2)
```



S_1 $p = [1 \quad -1]\begin{bmatrix} i \\ j \end{bmatrix} + 0$

S_2 $p = [1 \quad -1]\begin{bmatrix} i \\ j \end{bmatrix} + 1$

Generate: SPMD (Single–Program Multiple Data) Code



Let p be the processor's ID number

```
if (1-n <= p <= n) then
   if [1 <= p) then
      B[p,1] = A[p,0] * B[p,1];                          (S2)
   for i1 = max[1,1+p) to min[n,n-1+p) do
      A[i1,i1-p] = A[i1,i1-p] + B[i1-1,i1-p];            (S1)
      B[i1,i1-p+1] = A[i1,i1-p] * B[i1,i1-p+1];          (S2)
   if (p <= 0) then
      A[n+p,n] = A[n+p,N] + B[n+p-1,n];                  (S1)
```

Communication-Free Parallelization Algorithm

- Key Constraint:
  Communication-free parallelism =>
  Operations using the same data
  must be mapped to the same processor!
- Two-step Algorithm

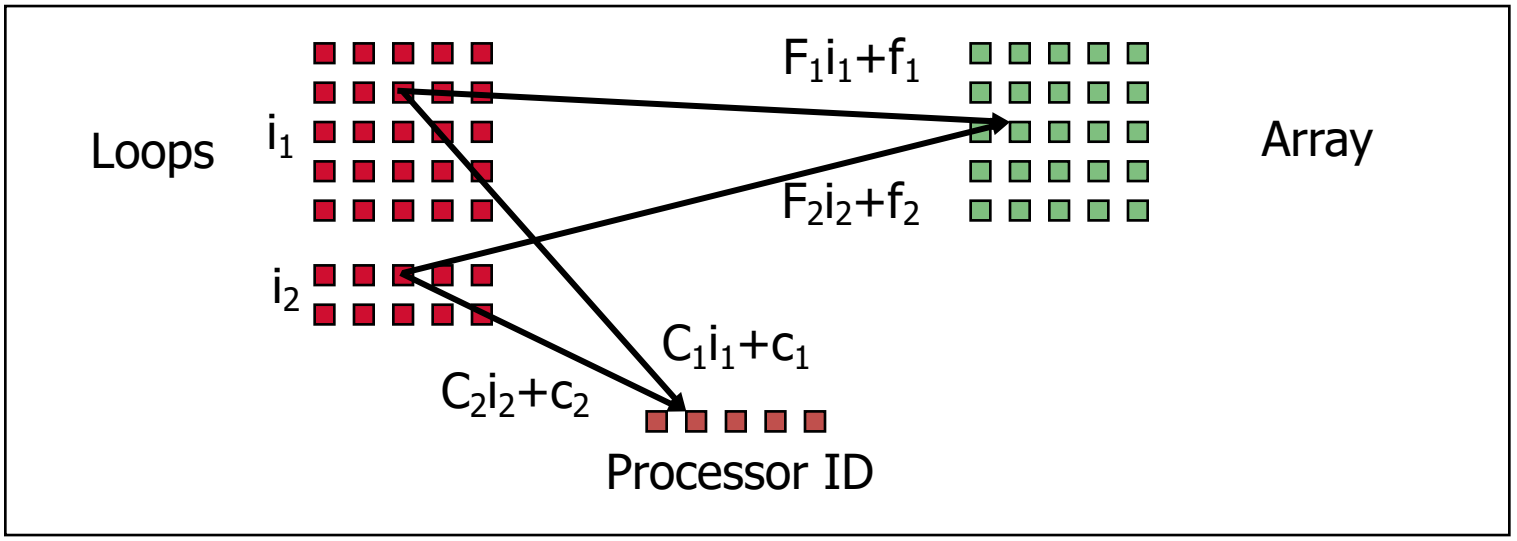
1. Given affine array indices in programs with arbitrarily nested loops
   - Find an affine mapping for each instruction to a processor ID such that there is no communication between processors
   - Eg:

$$\mathbf{S_1} \quad p = [1 \quad -1] \begin{bmatrix} i \\ j \end{bmatrix} + 0$$

$$\mathbf{S_2} \quad p = [1 \quad -1] \begin{bmatrix} i \\ j \end{bmatrix} + 1$$

2. Generate the SPMD code

Maximum Parallelism & No Communication



Let i_1 and i_2 be loop indices of 2 (not necessarily distinct) loops

For every pair of *data dependent* accesses $F_1i_1+f_1$ and $F_2i_2+f_2$

Find processor partitioning C_1, c_1, C_2, c_2:

$$\forall\ i_1, i_2 \quad F_1 i_1 + f_1 = F_2 i_2 + f_2 \rightarrow C_1 i_1 + c_1 = C_2 i_2 + c_2$$

with the objective of maximizing the rank of C_1, C_2

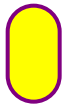
Quiz: Can you always find such a mapping?

Rank of Partitioning = Degree of Parallelism

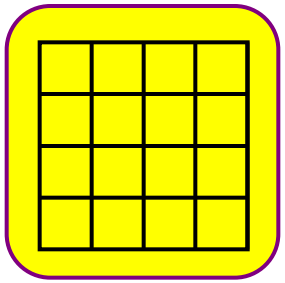
iterations

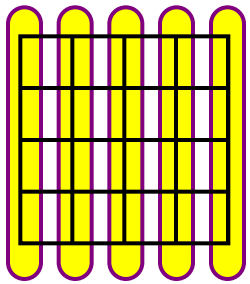
Affine Mapping $\begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$ $\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$ $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$

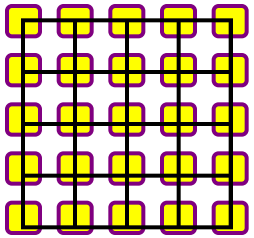
Rank 0 1 2



Mapped to same processor

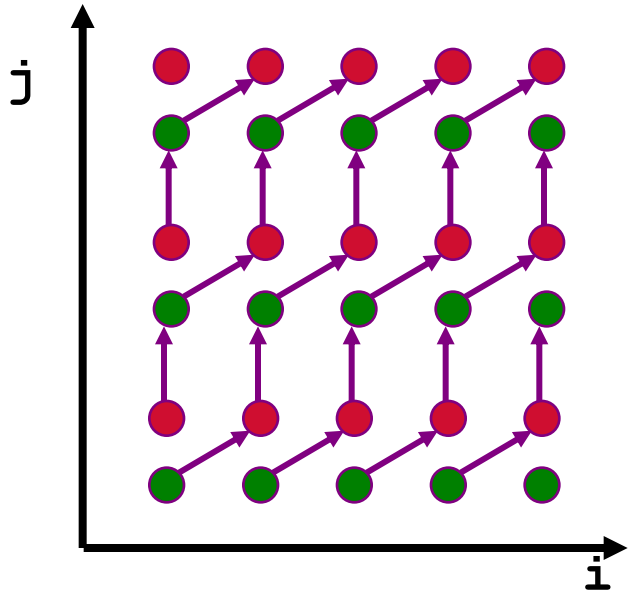






Solving Equations Give the Desired Partitioning (in the book)

```
FOR j = 1 TO n
   FOR i = 1 TO n
      A[i,j] = A[i,j]+B[i-1,j];          (S1)
      B[i,j] = A[i,j-1]*B[i,j];          (S2)
```



S_1 $p = [1 \quad -1]\begin{bmatrix} i \\ j \end{bmatrix} + 0$

S_2 $p = [1 \quad -1]\begin{bmatrix} i \\ j \end{bmatrix} + 1$

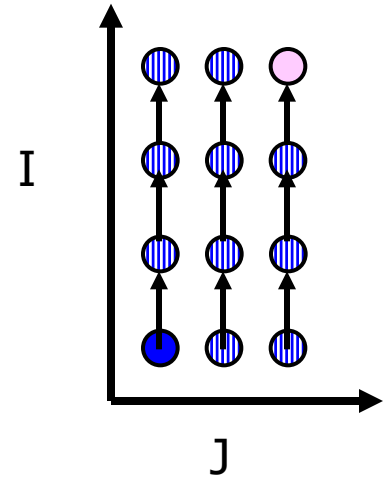
3. Step 2: Code Generation

- Naive
  - Each processor visits all the iterations in the original execution order
  - Executes only if it owns that iteration
  - Quiz: why is it correct?

- Optimization
  - Removes unnecessary looping and condition evaluation

Example 1: Loop Permutation

```
for I = 1 to 4
  for J = 1 to 3
    Z[I,J] = Z[I-1,J]
```



1. Find affine partitioning: c_1, c_2, c_0 such that

$$p = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + c_0$$

2. Suppose iteration i,j & i', j' refer to same location

$$i = i' - 1$$
$$j = j'$$

3. No communication means:

$c_1\, i + c_2\, j + c_0 = c_1\, i' + c_2\, j' + c_0$

$c_1(i'-1) + c_2\, j' + c_0 = c_1\, i' + c_2\, j' + c_0$

$c_1 = 0$

$p = c_2\, j + c_0$

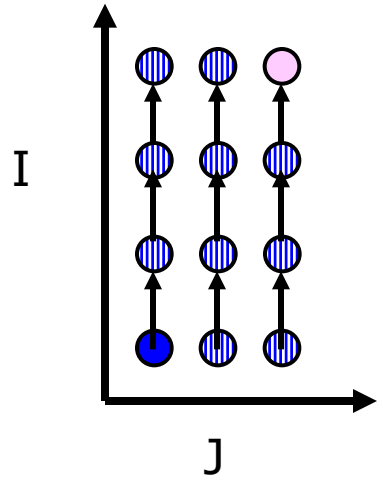
4. Pick simplest c_0 and non-zero c_2 :

$c_2 = 1$, $c_0 = 0$

$p = j$

Example 1: Code Generation

```
for I = 1 to 4
   for J = 1 to 3
      Z[I,J] = Z[I-1,J]
```



p = j

```
for P = 1 to 3
   for I = 1 to 4
      for J = 1 to 3
         if (j == P)
            Z[I,J] = Z[I-1,J]
```

Optimization

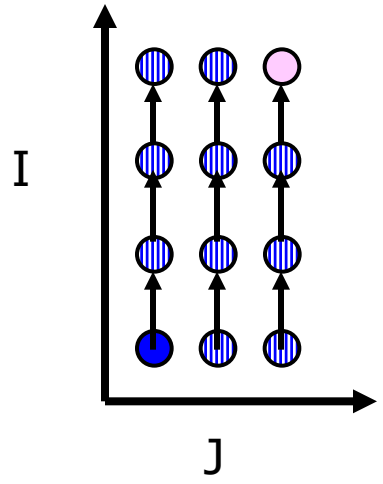
```
for P = 1 to 3
   for I = 1 to 4
      Z[I,P] = Z[I-1,P]
```

SPMD (single program multiple data) code:

```
for I = 1 to 4
      Z[I,P] = Z[I-1,P]
```

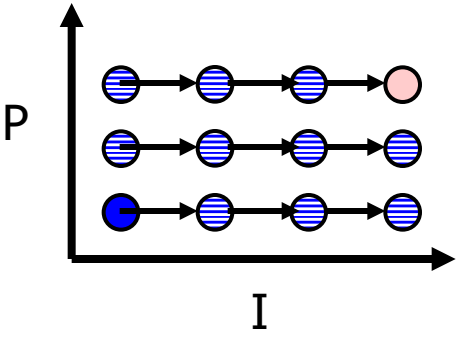
Equivalent to Loop Permutation (Loop Interchange)

```
for I = 1 to 4
  for J = 1 to 3
    Z[I,J] = Z[I-1,J]
```



$$\begin{bmatrix} p' \\ i' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

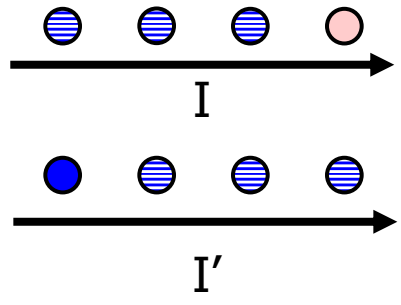
```
for P = 1 to 3
  for I = 1 to 4
    Z[I,P] = Z[I-1,P]
```



Our algorithm performs the same effect in 2 steps (partitioning + code generation)
Useful for handling multiple statements in a program

Example 2: Loop Fusion

```
for I = 1 to 4
    T[I]= A[I]+B[I]     (s1)
for I' = 1 to 4
    C[I']= T[I'] x T[I'] (s2)
```



1. Find affine partitioning: $c_{1,1}$, $c_{1,0}$, $c_{2.1}$, $c_{1,0}$, such that

s1: $[p] = [c_{1,1}][i] + c_{1,0}$

s2: $[p] = [c_{2,1}][i'] + c_{2,0}$

2. Suppose iteration i & i' refer to the same location

$i = i'$

3. No communication means:

$c_{1,1}\, i + c_{1,0} = c_{2,1}\, i' + c_{2,0}$

$c_{1,1} = c_{2,1}$

$c_{1,0} = c_{2,0}$

4. Pick simplest, non-zero values for $c_{1,1}$, $c_{2,1}$:

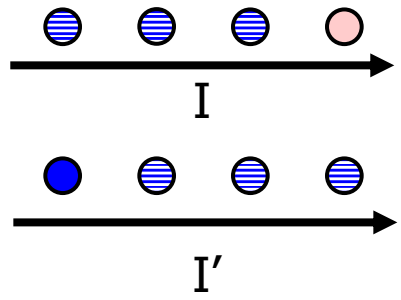
$c_{1,1} = c_{2,1} = 1$, $c_{1,0} = c_{2,0} = 0$

$p = i$; $p = i'$

Quiz: How would you transform this code?

Loop Fusion

```
for I = 1 to 4
    T[I]= A[I]+B[I]      (s1)
for I' = 1 to 4
    C[I']= T[I'] x T[I'] (s2)
```



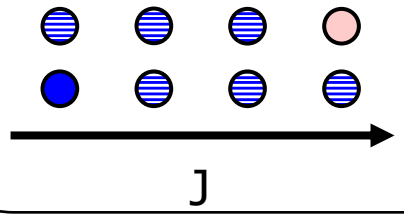
s1: $[p] = [1]\,[i]$

s2: $[p] = [1]\,[i']$

```
for P = 1 to 4
  for I = 1 to 4
    if (I == P)
      T[I]= A[I]+B[I]      (s1)
  for I' = 1 to 4
    if (I' == P)
      C[I']= T[I'] x T[I']  (s2)
```

Optimization

```
for P = 1 to 4
    T[P]= A[P]+B[P]   (s1)
    C[P]= T[P] x T[P]  (s2)
```

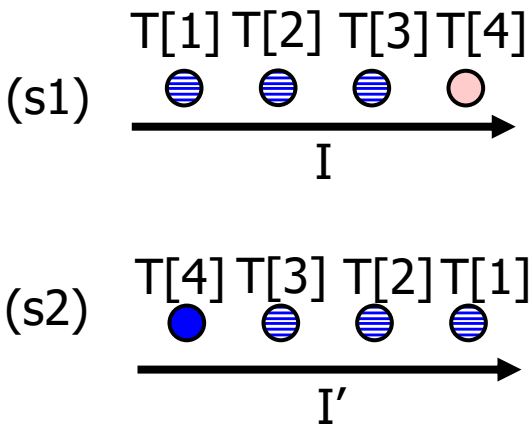


Example 3

Do you know how the algorithm works without simulating it?

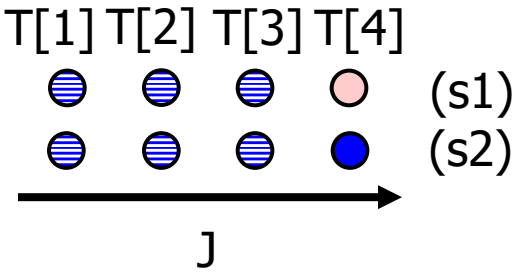
```
for I = 1 to 4
  T[I]= A[I]+B[I]          (s1)
for I' = 1 to 4
  C[I']= T[5-I'] x T[5-I']  (s2)
```

```
for J = 1 to 4
    T[J]= A[J]+B[J]     (s1)
    C[   ]= T[J] x T[J] (s2)
```



s1: [j]=[1][i]

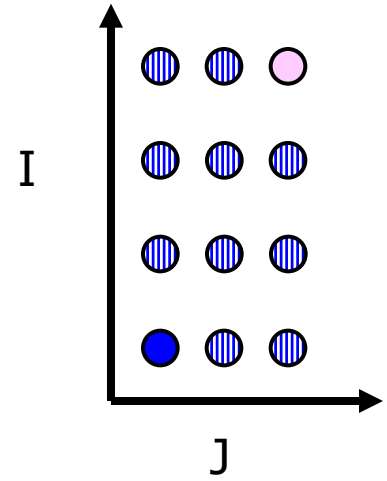
s2: [j]=[ ][i']+[ ]



Homework: Derive the answer by simulating the affine partitioning algorithm.

Example 4: 2 Nested, DoAll Loops

```
for I = 1 to 4
   for J = 1 to 3
      Z[I,J] = Z[I,J]+1
```



1. Find affine partitioning: c_1, c_2, c_0 such that

$$p = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + c_0$$

2. Suppose iteration i,j & i', j' refer to same location

$$i = i'$$
$$j = j'$$

3. No communication means:

$$c_1 i + c_2 j + c_0 = c_1 i' + c_2 j' + c_0$$

$$c_1 i' + c_2 j' + c_0 = c_1 i' + c_2 j' + c_0$$

4. No constraints

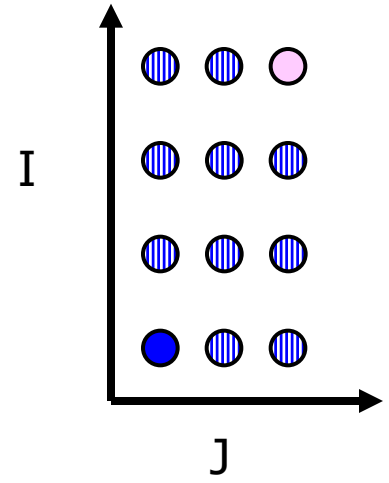
Two basis vectors: $[c_1\ c_2]=[1\ 0]$, or $[c_1\ c_2] = [0\ 1]$

Two answers for p: two degrees of parallelism

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

Example 4: 2 Nested DoAll Loops

```
for I = 1 to 4
  for J = 1 to 3
    Z[I,J] = Z[I,J]+1
```



$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

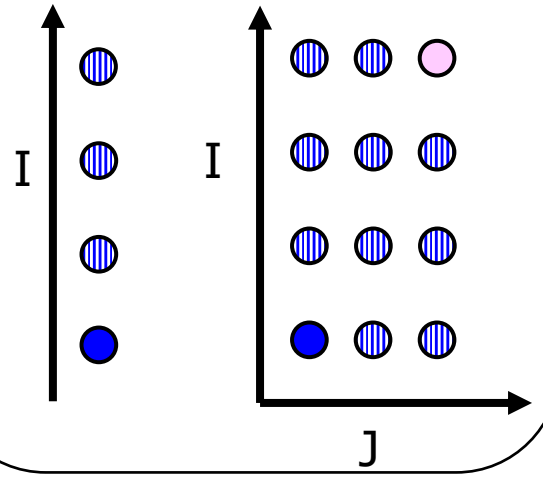
```
for p1 = 1 to 4
  for p2 = 1 to 3
    for I = 1 to 4
      for J = 1 to 3
        if (I==p1 & J == p2)
          Z[I,J] = Z[I,J]+1
```

Optimization

```
for p1 = 1 to 4
  for p2 = 1 to 3
    Z[p1,p2] = Z[p1,p2]+1
```

Example 5: Imperfectly Nested Loops

```
for I = 1 to 4
    Z[I,1] = Y[I];
    for J = 1 to 3
        Z[I,J] = Z[I,J]+1
```



$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [i] + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

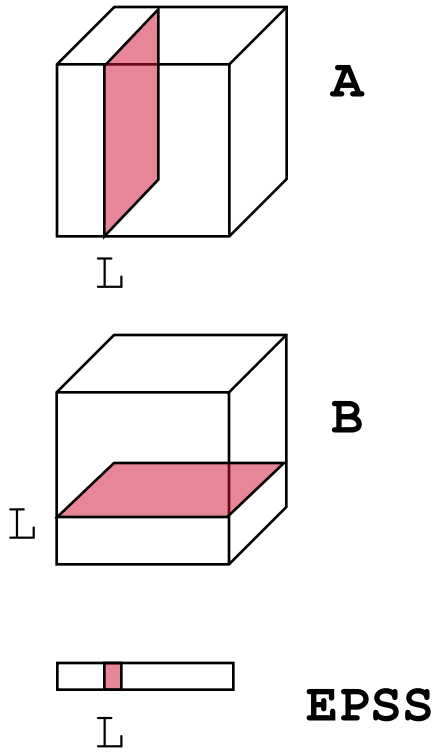
```
for p1 = 1 to 4
  for p2 = 1 to 3
    for I = 1 to 4
      if (I==p1 & 1 == p2)
          Z[I,1] = Y[I]
      for J = 1 to 3
        if (I==p1 & J == p2)
            Z[I,J] = Z[I,J]+1
```

Optimization

```
for p1 = 1 to 4
  for p2 = 1 to 3
    if p2 == 1
      Z[p1,1] = Y[p1]
    Z[p1,p2] = Z[p1,p2]+1
```

Optimizing Arbitrary Loop Nesting Using Affine Partitions (chotst, NAS)

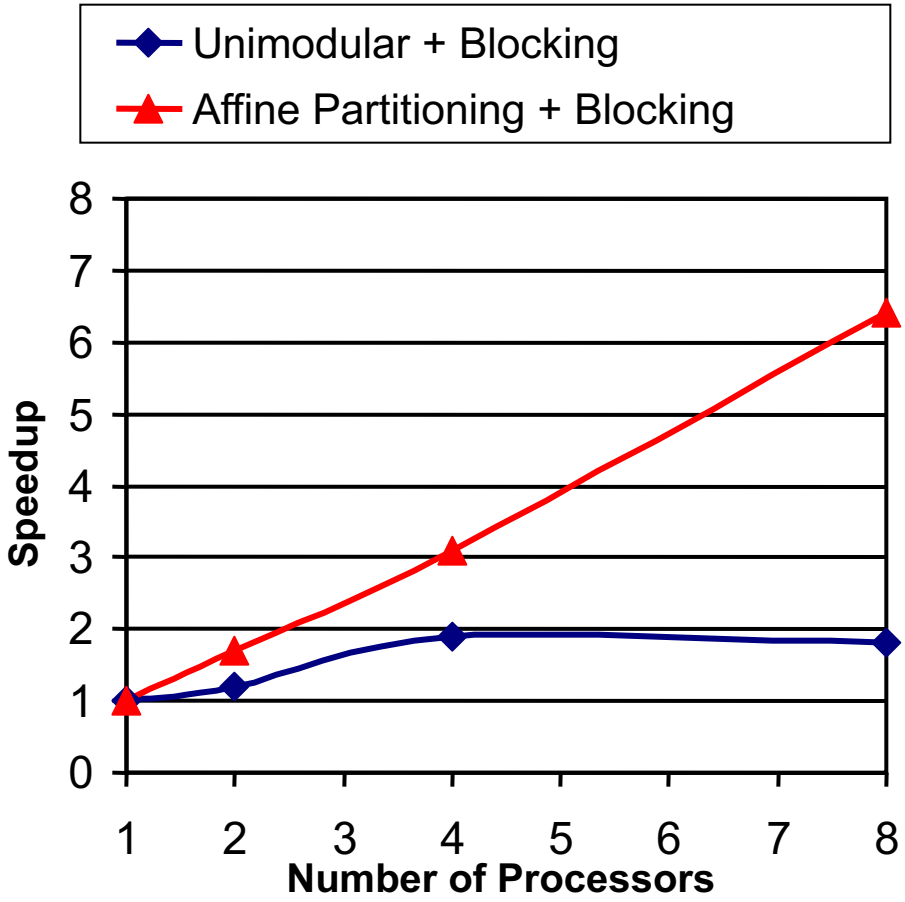
```
      DO 1 J = 0, N
       I0 = MAX ( -M, -J )
          DO 2 I = I0, -1
             DO 3 JJ = I0 - I, -1
                DO 3 L = 0, NMAT
3                  A(L,I,J) = A(L,I,J) - A(L,JJ,I+J) * A(L,I+JJ,J)
             DO 2 L = 0, NMAT
2               A(L,I,J) = A(L,I,J) * A(L,0,I+J)
          DO 4 L = 0, NMAT
4            EPSS(L) = EPS * A(L,0,J)
          DO 5 JJ = I0, -1
             DO 5 L = 0, NMAT
5               A(L,0,J) = A(L,0,J) - A(L,JJ,J) ** 2
          DO 1 L = 0, NMAT
1            A(L,0,J) = 1. / SQRT ( ABS (EPSS(L) + A(L,0,J)) )

       DO 6 I = 0, NRHS
          DO 7 K = 0, N
             DO 8 L = 0, NMAT
8               B(I,L,K) = B(I,L,K) * A(L,0,K)
             DO 7 JJ = 1, MIN (M, N-K)
                DO 7 L = 0, NMAT
7                  B(I,L,K+JJ) = B(I,L,K+JJ) -  A(L,-JJ,K+JJ) * B(I,L,K)
          DO 6 K = N, 0, -1
             DO 9 L = 0, NMAT
9               B(I,L,K) = B(I,L,K) * A(L,0,K)
             DO 6 JJ = 1, MIN (M, K)
                DO 6 L = 0, NMAT
6                  B(I,L,K-JJ) = B(I,L,K-JJ) - A(L,-JJ,K) * B(I,L,K)
```



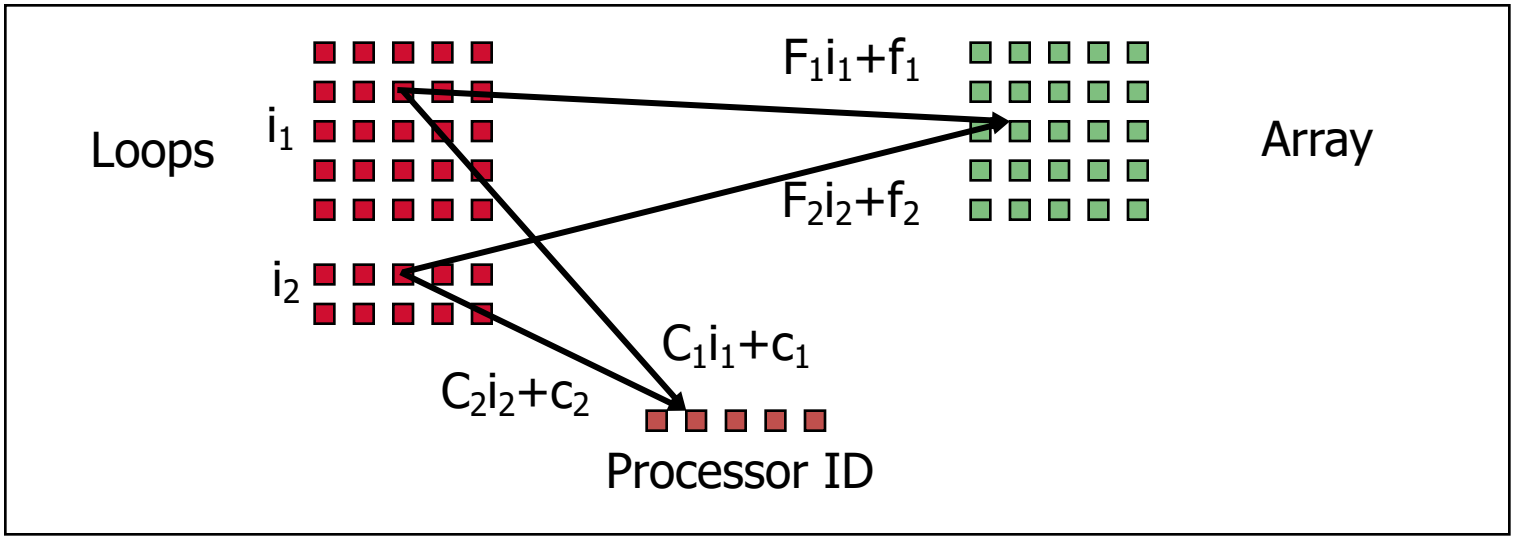
Result of affine transform: 1 outermost loop parallelism: index L

Chotst: Results with Affine Partitioning + Blocking

(Unimodular: a subset of affine partitioning for perfect loop nests)



Maximum Parallelism & No Communication



Let i_1 and i_2 be loop indices of 2 (not necessarily distinct) loops

For every pair of data dependent accesses $F_1i_1+f_1$ and $F_2i_2+f_2$

Find C_1, c_1, C_2, c_2:

$$\forall\ i_1, i_2 \quad F_1 i_1 + f_1 = F_2 i_2 + f_2 \rightarrow C_1 i_1 + c_1 = C_2 i_2 + c_2$$

with the objective of maximizing the rank of C_1, C_2

Summary

- Affine transforms: a mathematical model
  - Succinct & expressive notation
    - Unifies arbitrary combinations of many loop transforms
    - Loop interchange, reversal, skewing, fusion, fission
  - **Optimal** algorithm for communication-free parallelism
    - Solving **equations** while maximizing the rank
  - Next class: Parallelism with minimum communication
    - Solving **linear inequalities** while maximizing the rank.