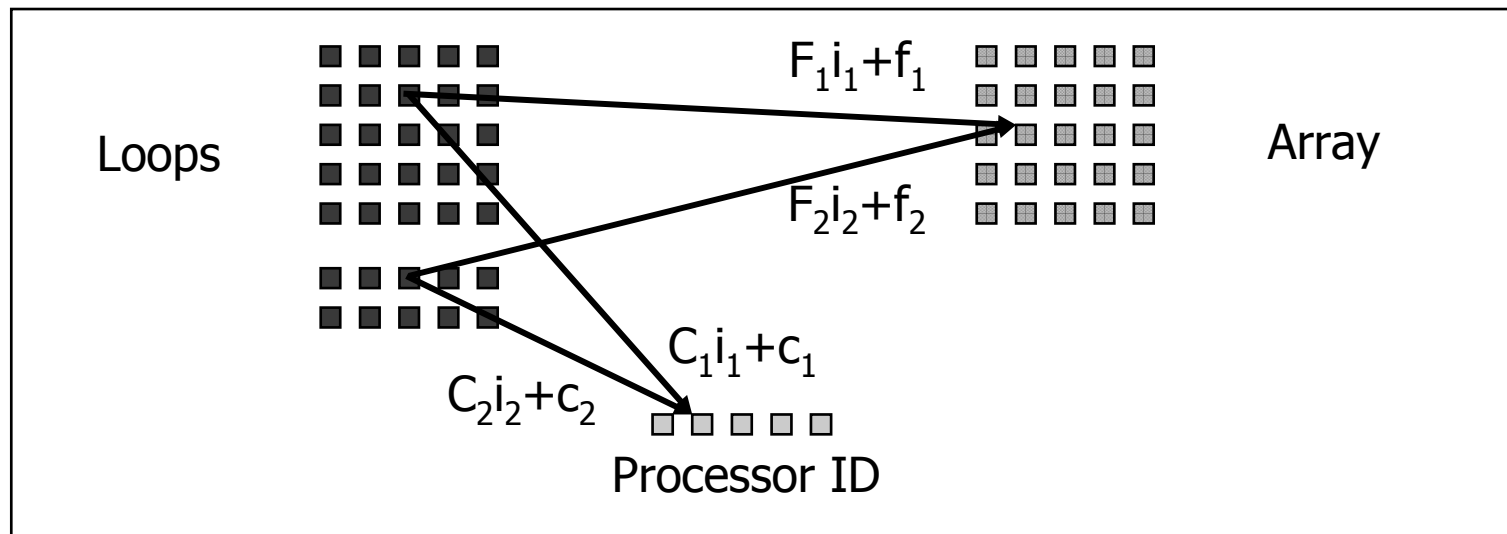


CS 243
Lecture 13
Affine Transforms

1. Loop Permutation as an Example
2. Seven Primitive Transforms
3. Advanced Topic: Pipelining & Blocking

Readings: Chapter 11.1 - 11.7

Affine Partitioning



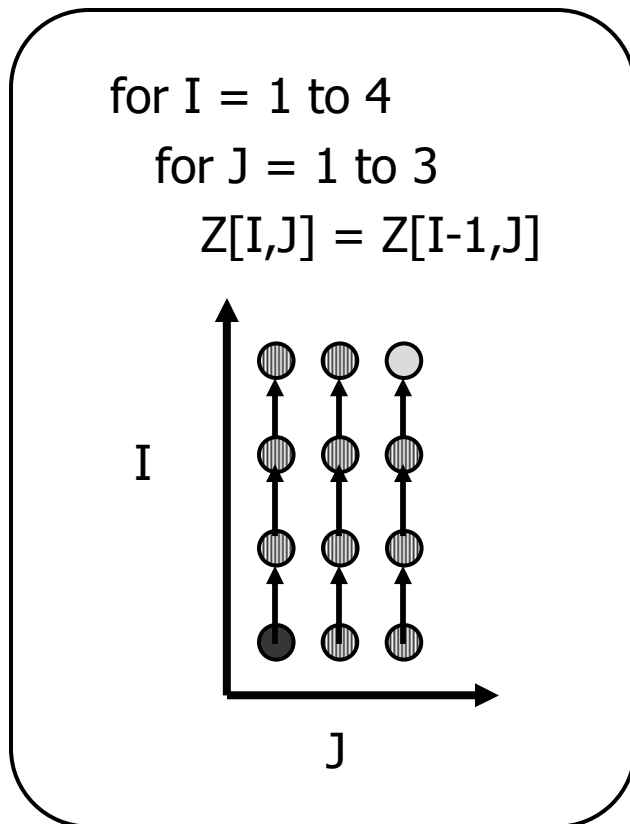
For every pair of data dependent accesses $F_1 i_1 + f_1$ and $F_2 i_2 + f_2$

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

1. Example: Loop Interchange (Loop Permutation)



Data dependent operations:

$\forall i, j, i', j'$ such that,

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

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

Find C_1, C_2, c for statement in loop

$$[C_1 \ C_2] \begin{bmatrix} i \\ j \end{bmatrix} + c = [C_1 \ C_2] \begin{bmatrix} i' \\ j' \end{bmatrix} + c$$

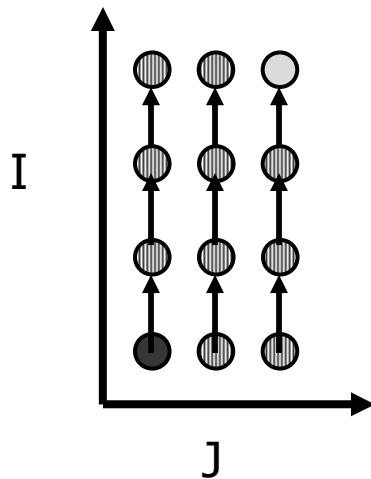
$$[C_1 \ C_2] \begin{bmatrix} i'-i \\ j'-j \end{bmatrix} = 0 \quad [C_1 \ C_2] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0$$

One solution: $[C_1 \ C_2] = [0 \ 1] \quad c = 0$

Therefore: $p = [0 \ 1] \begin{bmatrix} i \\ j \end{bmatrix} + c \quad p = j$

Code Generation: Execute each partition in sequential order Must be correct!

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

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

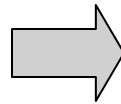
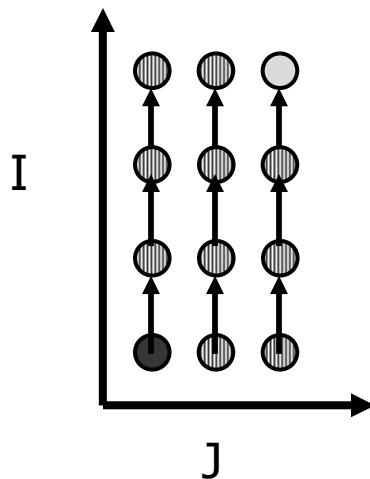
$$\begin{bmatrix} p \\ i' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$

Geometric Interpretation

for I = 1 to 4

for J = 1 to 3

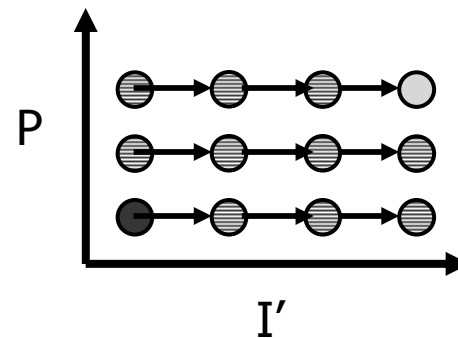
$Z[I,J] = Z[I-1,J]$



for P = 1 to 3

for I' = 1 to 4

$Z[I',P] = Z[I'-1,P]$



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

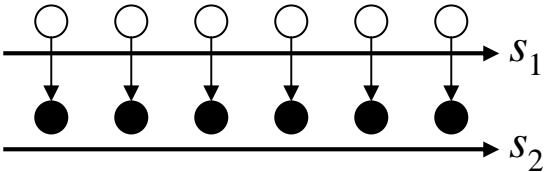
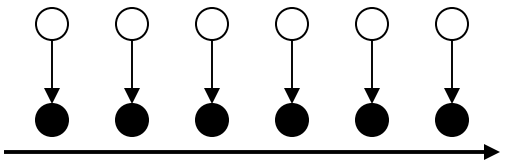
Affine Partitioning Algorithm: Maximize Degree of Parallelism with No Communication

- Find data dependences
- For each pair of data dependent operations
 - Set up equations $F_1i_1+f_1 = F_2i_2+f_2$
to capture relations of dependent iterations
 - Reduce the number of unknowns – lots of identities.
 - Set up equations for affine partitions $C_1i_1+c_1 = C_2i_2+c_2$
 - Each operation gets its own C, c
 - Work on C by dropping the c' ,
Rewrite constraints as $Ax = 0$, where x is the unknown Cs
 - Nullity of A is the rank C
 - Find the basis vectors
 - Find the constant c

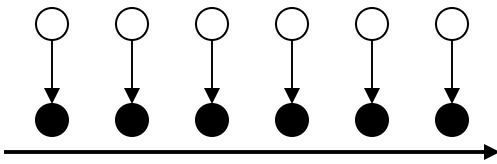
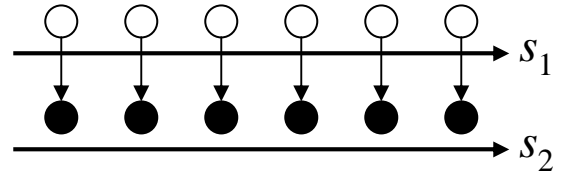
2. Primitive Loop Transformations

- Key idea:
If you draw the dependence graph,
you can eyeball the code and get the solution
derived using linear algebra
- Seven source-level transformations
 - Unimodular transform: Reversal, permutation, skewing
 - Fusion, fission, re-indexing, scaling
- Affine partitioning can express arbitrary combinations of these seven primitives.

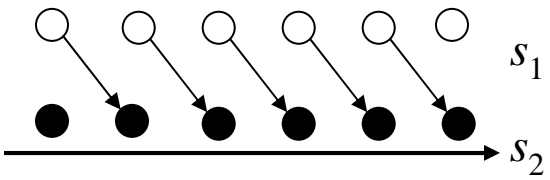
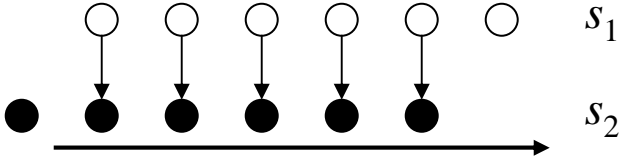
Fusion

Source Code	Partition	Transformed Code
<pre>for (i=1; i<=N; i++) Y[i] = Z[i]; /*s1*/ for (j=1; j<=N; j++) X[j] = Y[j]; /*s2*/</pre> 	$s_1 : p = i$ $s_2 : p = j$	<pre>for (p=1; p<=N; p++) { Y[p] = Z[p]; X[p] = Y[p]; }</pre> 

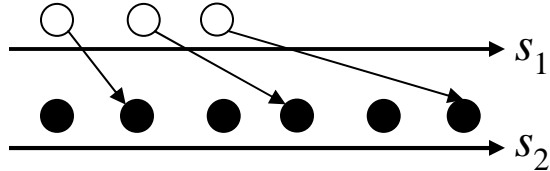
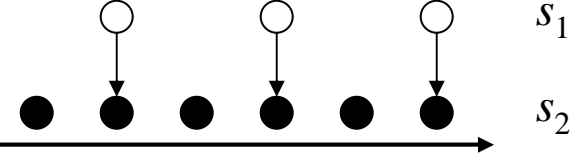
Fission

Source Code	Partition	Transformed Code
<pre>for (p=1; p<=N; p++) { Y[p] = Z[p]; X[p] = Y[p]; }</pre> 	$s_1 : i = p$ $s_2 : j = p$	<pre>for (i=1; i<=N; i++) Y[i] = Z[i]; /*s1*/ for (j=1; j<=N; j++) X[j] = Y[j]; /*s2*/</pre> 

Re-indexing

Source Code	Partition	Transformed Code
<pre> for (i=1; i<=N; i++) { Y[i] = Z[i]; /*s1*/ X[i] = Y[i-1]; /*s2*/ } </pre> 	$s_1 : p = i$ $s_2 : p = i - 1$	<pre> if (N>=1) X[1]=Y[0]; for (p=1; p<=N; p++) { Y[p] = Z[p]; X[p+1] = Y[p]; } if (N>=1) Y[N]=Z[N]; </pre> 

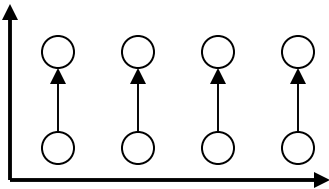
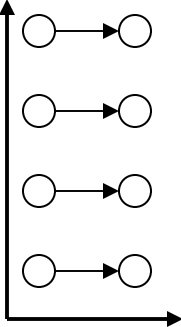
Scaling

Source Code	Partition	Transformed Code
<pre> for (i=1; i<=N; i++) Y[2*i] = Z[2*i]; /*s1*/ for (j=1; j<=2*N; j++) X[j] = Y[j]; /*s2*/ </pre> 	$s_1 : p = 2 \times i$ $s_2 : p = j$	<pre> for (p=1; p<=2*N; p++){ if (p mod 2 == 0) Y[p] = Z[p]; X[p] = Y[p]; } if (N>=1) Y[N]=Z[N]; </pre> 

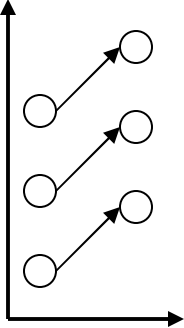
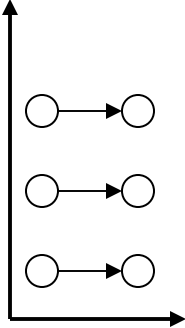
Reversal

Source Code	Partition	Transformed Code
<pre> for (i=0; i<=N; i++) Y[N-i] = Z[i]; /*s1*/ for (j=0; j<=N; j++) X[j] = Y[j]; /*s2*/ </pre>	$s_1 : p = i$ $s_2 : p = j$	<pre> for (p=0; p<=N; p++) { Y[p] = Z[N-p]; X[p] = Y[p]; } </pre>

Permutation

Source Code	Partition	Transformed Code
<pre>for (i=1; i<=N; i++) for (j=0; j<=M; j++) z[i,j] = z[i-1,j];</pre> 	$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$	<pre>for (p=0; p<=M; p++) for (q=1; q<=N; q++) z[q,p] = z[q-1,p];</pre> 

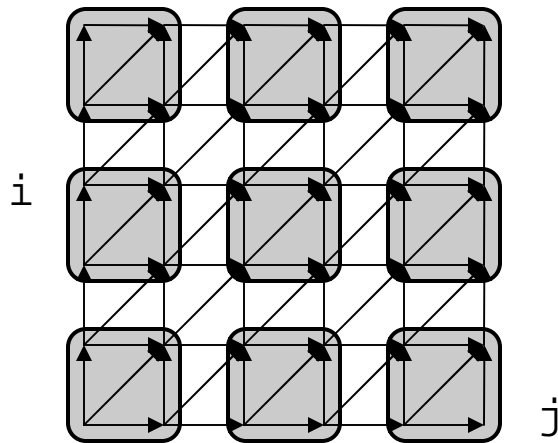
Skewing

Source Code	Partition	Transformed Code
<pre> for (i=1; i<=N+M-1; i++) for (j=max(1,i+N); j<=min(i,M); j++) Z[i,j] = Z[i-1,j-1]; </pre> 	$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	<pre> for (p=1; p<=N; p++) for (q=1; q<=M; q++) Z[p,q-p] = Z[p-1,q-p-1]; </pre> 

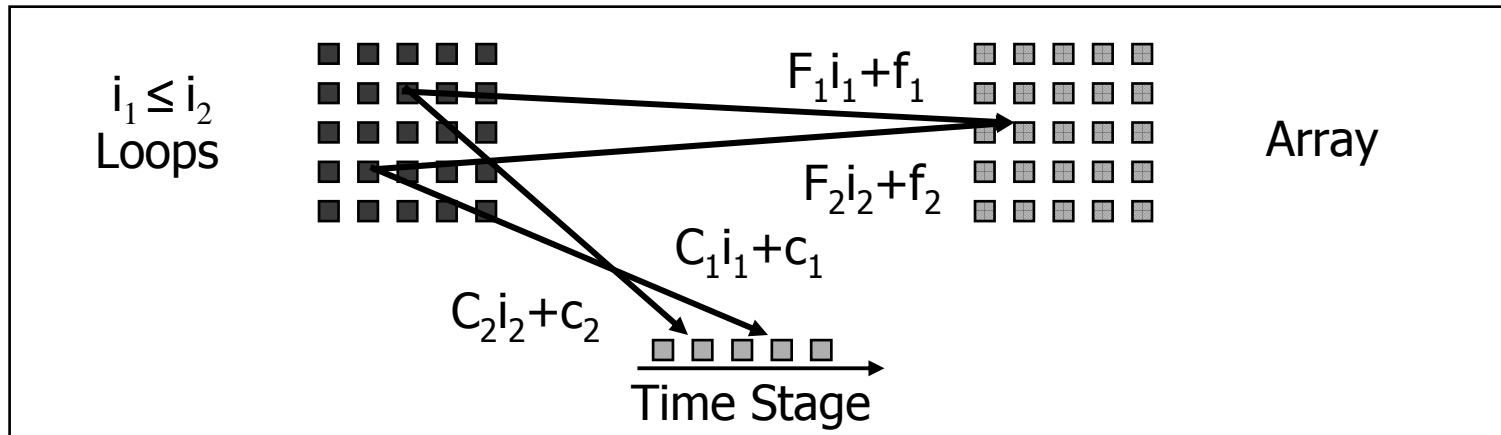
3. Advanced topic: Pipelining

SOR (Successive Over-Relaxation): An Example

```
for i = 0 TO m
  for j = 0 to n
    x[j+1] = 1/3 * (x[j] + x[j+1] + x[j+2])
```



Finding the Maximum Degree of Pipelining



For every pair of data dependent accesses $F_1 i_1 + f_1$ and $F_2 i_2 + f_2$

Let $B_1 i_1 + b_1 \geq 0$, $B_2 i_2 + b_2 \geq 0$ be the corresponding loop bound constraints,

Find C_1, c_1, C_2, c_2 :

$$\forall i_1, i_2 \quad B_1 i_1 + b_1 \geq 0, \quad B_2 i_2 + b_2 \geq 0$$

$$(i_1 \leq i_2) \wedge (F_1 i_1 + f_1 = F_2 i_2 + f_2) \rightarrow C_1 i_1 + c_1 \leq C_2 i_2 + c_2$$

with the objective of maximizing the rank of C_1, C_2

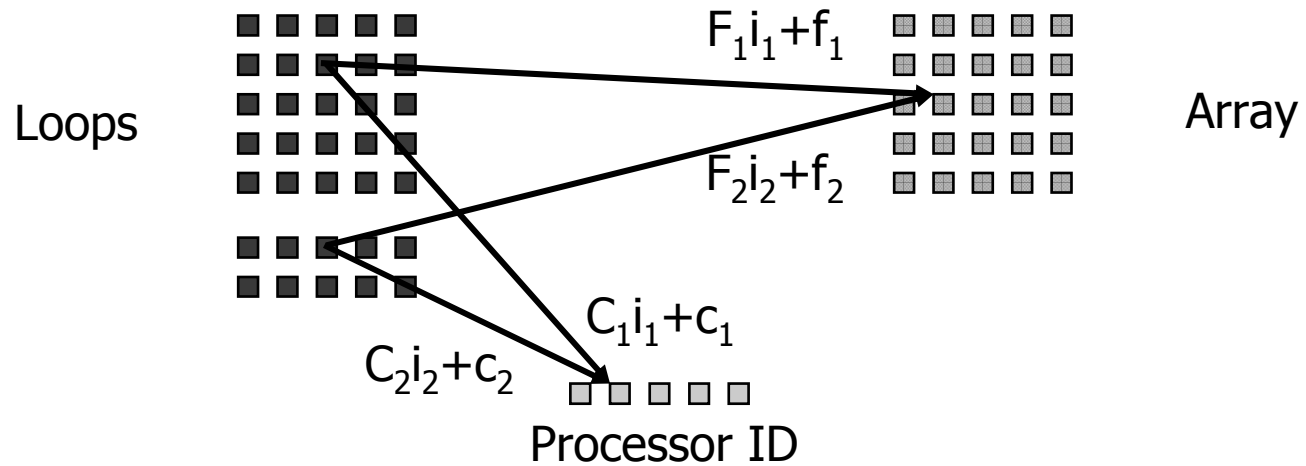
Key Insight

- Choice in time mapping => (pipelined) parallelism
- $\text{Rank}(C) - 1$ degree of parallelism with 1 degree of synchronization
- Can create blocks with $\text{Rank}(C)$ dimensions

- Find time partitions is not as straightforward as space partitions
 - Need to deal with linear inequalities
 - Solved using Farkas Lemma – no simple intuitive proof

Summary

Communication-Free



Pipelining

