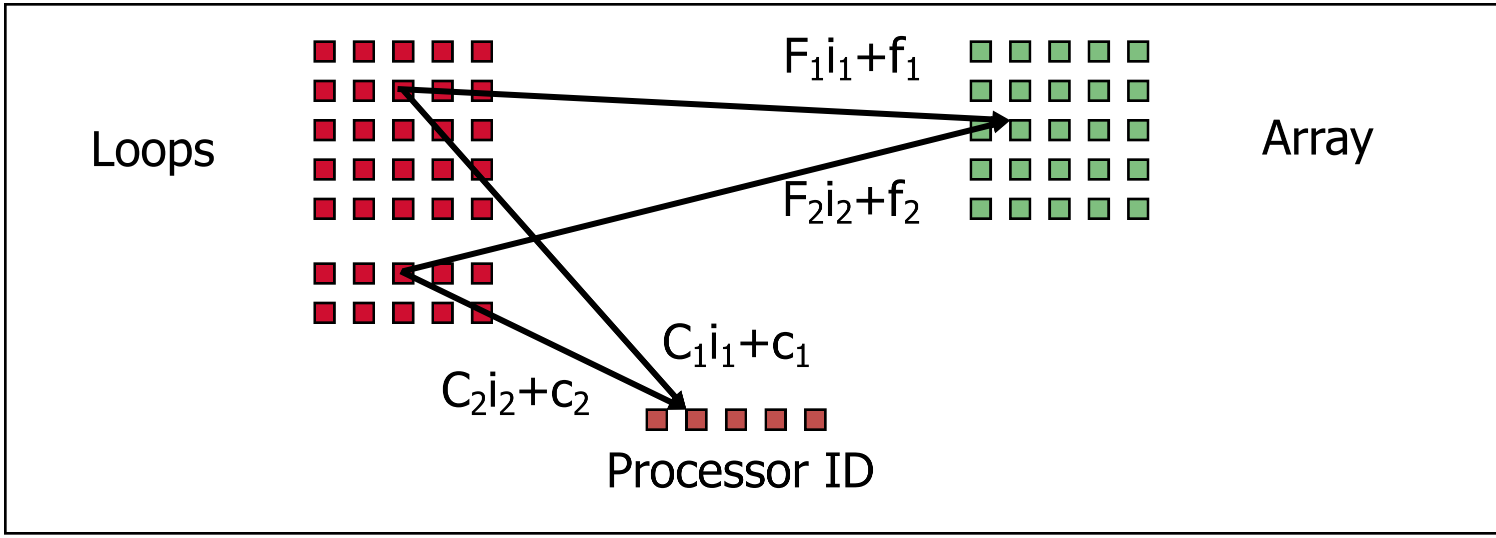
Lecture 11
Pipelined Parallelism

1. Intuition: Time mapping
2. Affine Time Partitioning Problem
3. Affine Time Partitioning Algorithm
4. Coarsest-Grain Parallelization

Readings: Chapter 11.8-11.9

Lecture 10: Maximum Parallelism & No Communication



C: Space partitioning of Computation to Processor ID

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

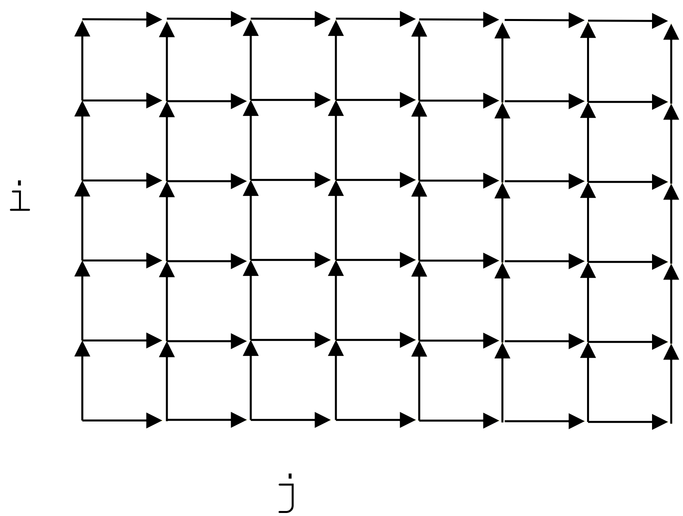
Find C_1, c_1, C_2, c_2:

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

with the objective of maximizing the rank of C_1, C_2

1. SOR (Successive Over-Relaxation): An Example

```
for i = 1 TO m
   for j = 1 to n
     A[i,j] = c * (A[i-1,j] + A[i,j-1])
```

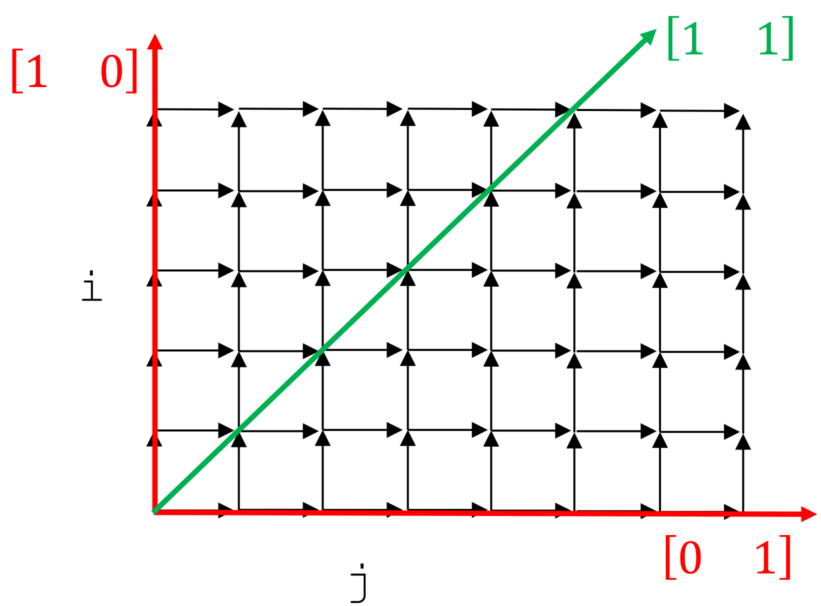


Quiz:

- Is there communication-free parallelism?
- Can you find parallelism with communication?

SOR (Successive Over-Relaxation): An Example

```
for i = 1 TO m
   for j = 1 to n
     A[i,j] = c * (A[i-1,j] + A[i,j-1])
```

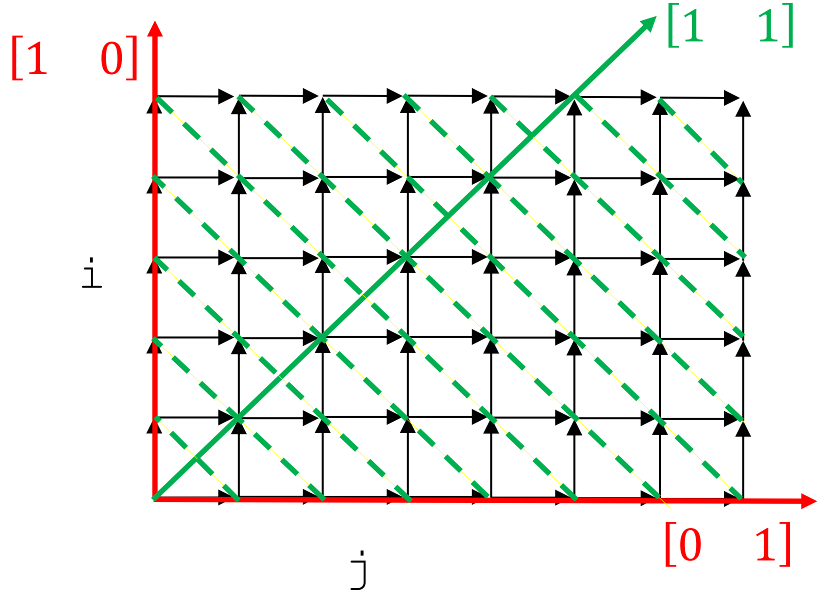


Focusing on sequential execution

- i is a legal outer loop
- Is j also a legal outer loop?
- Two independent basis vectors for the outer loop:
  [1 0], [0 1]
- Any combination of [1 0], [0 1] is a legal (DoAll) outer loop!
- Example: [1 1]

SOR (Successive Over-Relaxation): An Example

```
for i = 1 TO m
   for j = 1 to n
     A[i,j] = c * (A[i-1,j] + A[i,j-1])
```

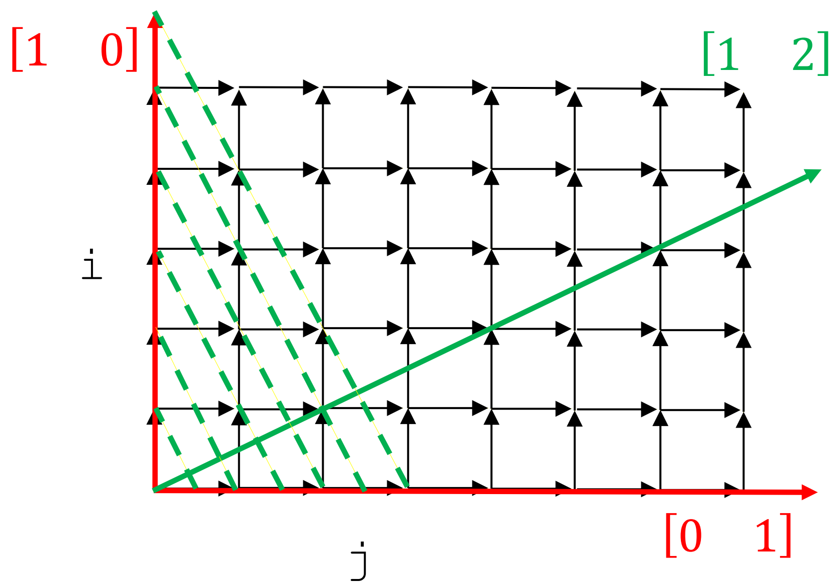


Focusing on sequential execution

- i is a legal outer loop
- Is j also a legal outer loop?
- Two independent basis vectors for the outer loop:
  [1 0], [0 1]
- Any combination of [1 0], [0 1] is a legal (DoAll) outer loop!
- Example: [1 1]
  - All original data dependences do not point backward in time
  - A wavefront of execution

SOR (Successive Over-Relaxation): An Example

```
for i = 1 TO m
   for j = 1 to n
     A[i,j] = c * (A[i-1,j] + A[i,j-1])
```

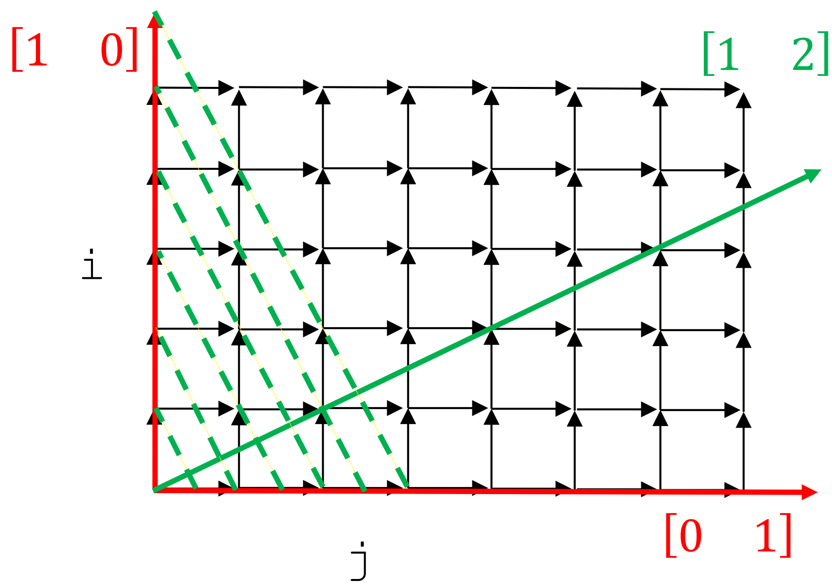


Focusing on sequential execution

- Two independent basis vectors for the outer loop:
  [1 0], [0 1]
- Any combination of [1 0], [0 1] is a legal (DoAll) outer loop!
- Example: [1 2]
  - All original data dependences do not point backward in time
  - A wavefront of execution

General Observation

```
for i = 1 TO m
  for j = 1 to n
    A[i,j] = c * (A[i-1,j] + A[i,j-1])
```

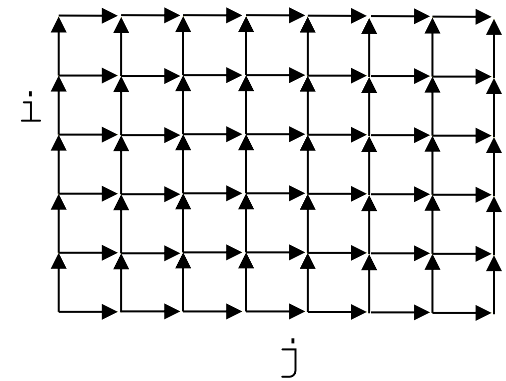


- Multiple legal outer loops:
  - Choice in execution → there is parallelism
- 2 independent basis vectors
  - The solution space has rank 2
  - 1 degree of pipelined parallelism
- *r* independent basis vectors
  - The solution space has rank *r*
  - *r* -1 degrees of pipelined parallelism

Choice means parallelism

Implementing Wavefronts Without Barriers

```
for i = 1 TO m
   for j = 1 to n
      A[i,j] = c * (A[i-1,j] + A[i,j-1])
```

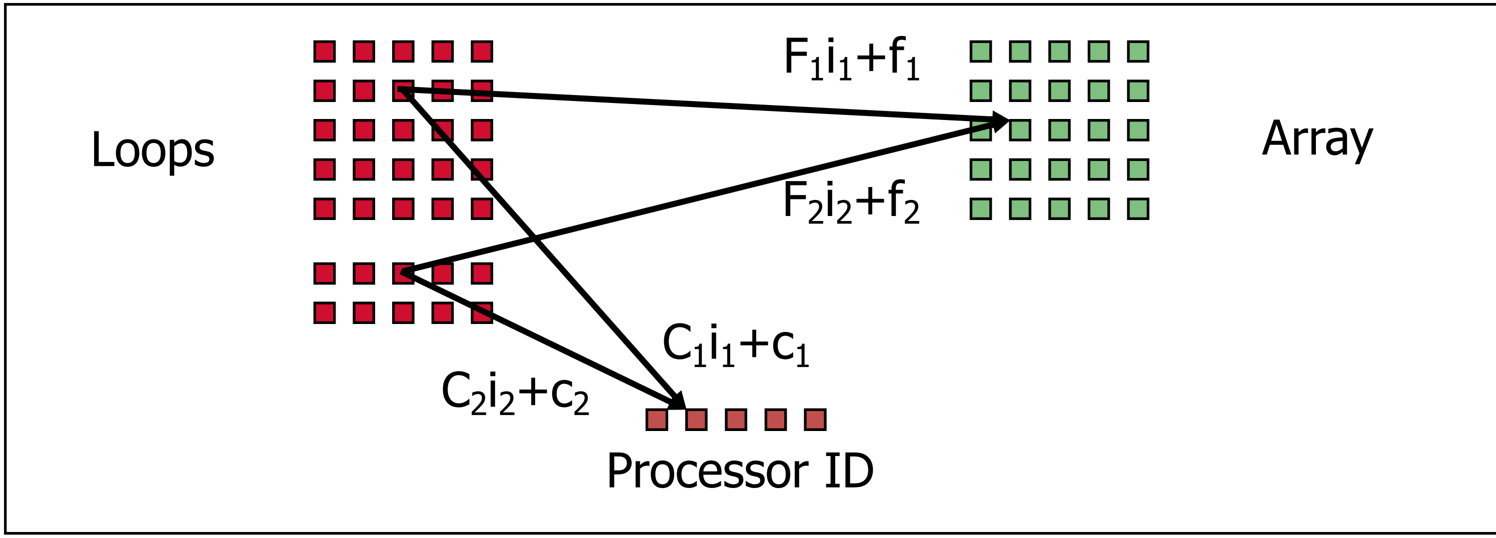


Assign each row i to a processor
Processor ID: p = i
Synchronization variable: t[p] = 0 (iterations executed)
WAIT: thread waits until the condition becomes true

```
for j = 1 to n
  if (p==1) or (WAIT(t[p-1]>=j))
    A[p,j] = c * (A[p-1,j] + A[p,j-1])
  t[p]++;
```

- Good locality
- Relaxed wavefront
- O(n) synchronization overhead

2. Recall: Maximum Parallelism & No Communication



C: Space partitioning of Computation to Processor ID

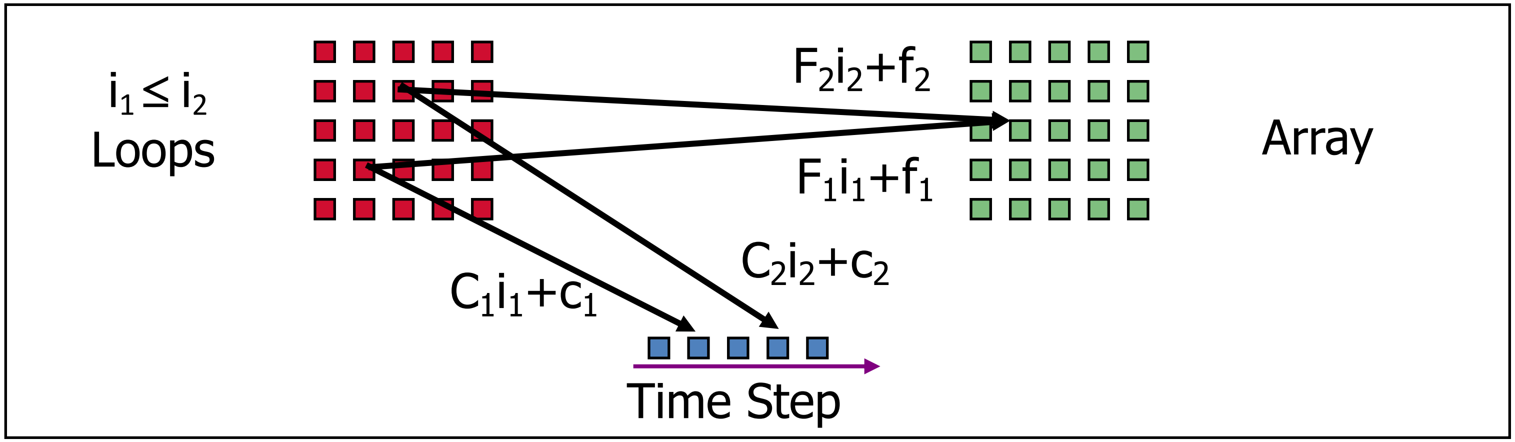
For every pair of data dependent accesses $F_1i_1+f_1$ and $F_2i_2+f_2$

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

Problem Statement: Maximum Pipelinable Parallelism



C: Time Partitioning of Computation to Time Step

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

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

Find C_1, c_1, C_2, c_2:

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

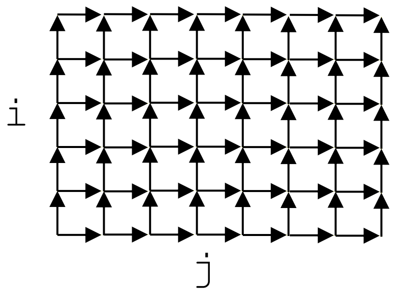
$$(i_1 \le i_2) \wedge (F_1 i_1 + f_1 = F_2 i_2 + f_2) \rightarrow C_1 i_1 + c_1 \le C_2 i_2 + c_2$$

with the objective of maximizing the rank of C_1, C_2

Example 1

(a)

```
for i = 1 TO m
 for j = 1 to n
  A[i,j]=c*(A[i-1,j]+ A[i,j-1])
```



2 independent time mappings:

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

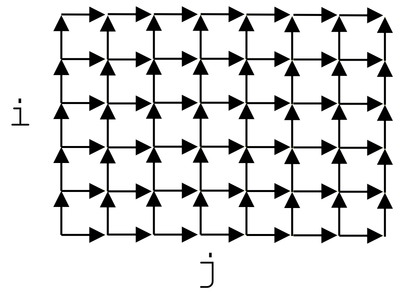
$$[t] = [0 \quad 1]\begin{bmatrix} i \\ j \end{bmatrix} + [0]$$

- Solving the equations
  - yields two independent basic vectors: $[1 \quad 0]$, $[0 \quad 1]$
  - 2 possible legal outer loops
  - 1 degree of pipelined parallelism

Example 1

(a)

```
for i = 1 TO m
 for j = 1 to n
  A[i,j]=c*(A[i-1,j]+ A[i,j-1])
```



2 time mappings:

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

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

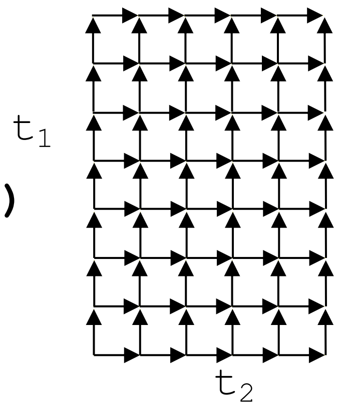
→ 2 legal permutations

(a) $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \end{bmatrix}$

(b) $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} i \\ j \end{bmatrix}$

(b)

```
for t1 = 1 TO n
 for t2 = 1 to m
  A[i,j]=c*(A[i-1,j] + A[i,j-1])
```

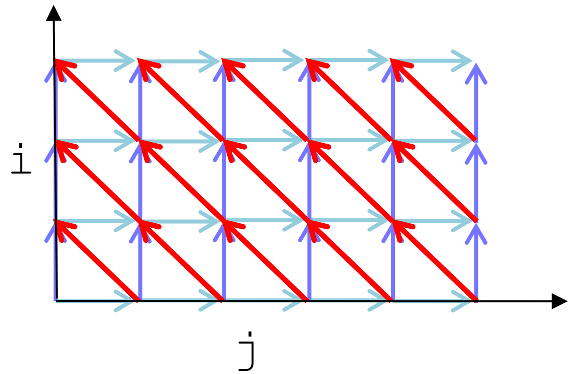


- 2 legal outer loops
- If all loops in a nest can be outermost → the loop nest is ***fully permutable***
- 2 ways to pipeline

Quiz: How many degrees of parallelism is there?

Example 2

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



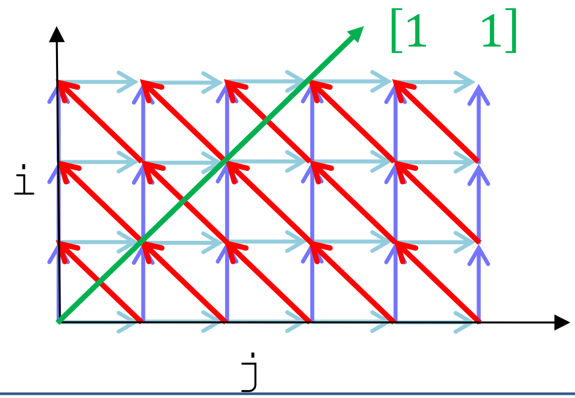
Is there communication-free parallelism?

Is the loop nest fully permutable?

Can we transform the code to make it permutable?

Applying Time Partitioning to Example 2

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



2 time mappings:

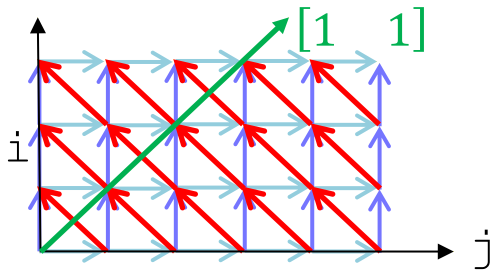
$$[t] = [1 \quad 0]\begin{bmatrix} i \\ j \end{bmatrix} + [0]$$

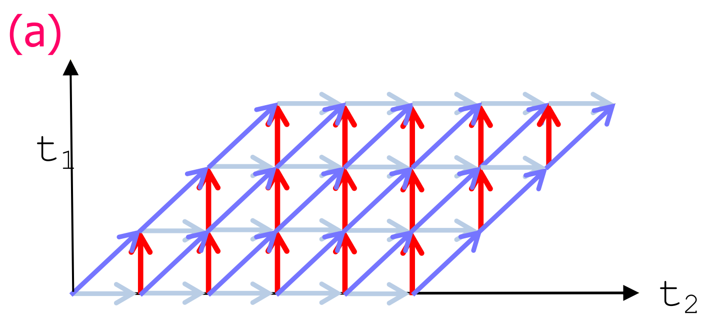
$$[t] = [1 \quad 1]\begin{bmatrix} i \\ j \end{bmatrix} + [0]$$

Intuitively:
The time mapping makes all
the dependences not point backwards.

Time Partitioning Results

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

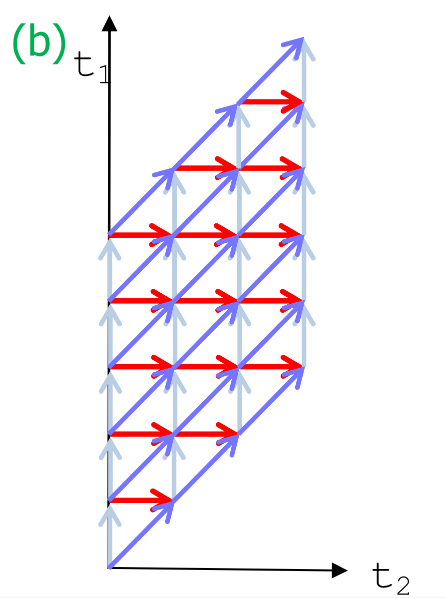




2 time mappings:

$$[t] = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}i \\ j\end{bmatrix} + [0]$$

$$[t] = \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}i \\ j\end{bmatrix} + [0]$$



2 permutations:

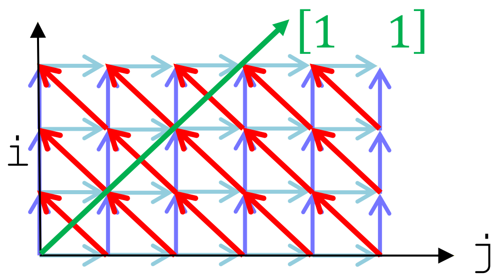
(a) $\begin{bmatrix}t_1 \\ t_2\end{bmatrix} = \begin{bmatrix}1 & 0 \\ 1 & 1\end{bmatrix}\begin{bmatrix}i \\ j\end{bmatrix}$

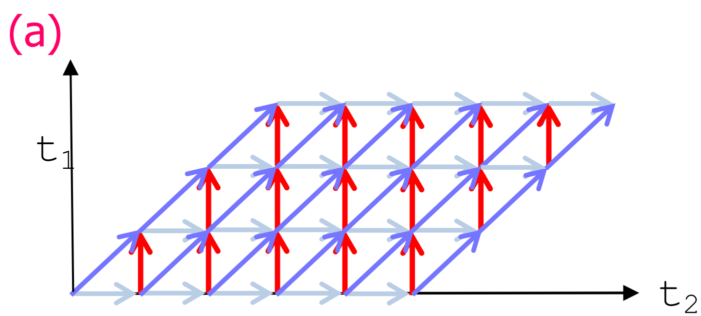
(b) $\begin{bmatrix}t_1 \\ t_2\end{bmatrix} = \begin{bmatrix}1 & 1 \\ 1 & 0\end{bmatrix}\begin{bmatrix}i \\ j\end{bmatrix}$

Quiz: How many degrees of parallelism?

Time Partitioning Results

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

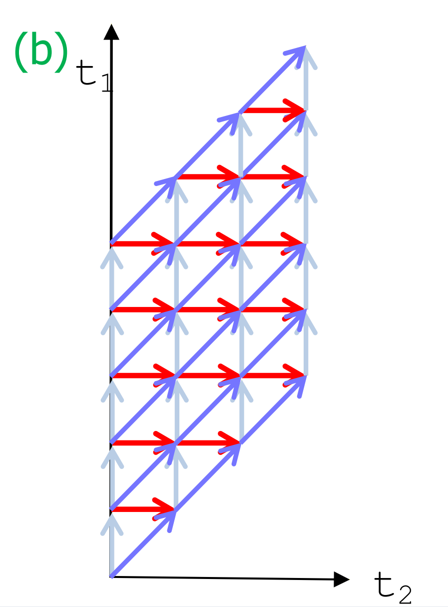




2 time mappings:

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

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



2 permutations:

(a) $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} i \\ j \end{bmatrix}$

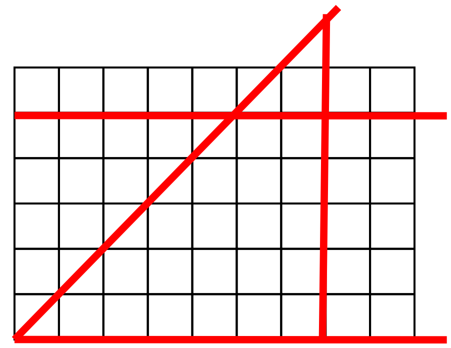
(b) $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} i \\ j \end{bmatrix}$

Intuitively: Transform so all dependences don't point backwards in both axes.

Skew transform: $\begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix}$ Quiz: Can we always skew to create permutable loop nests?

Recall

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

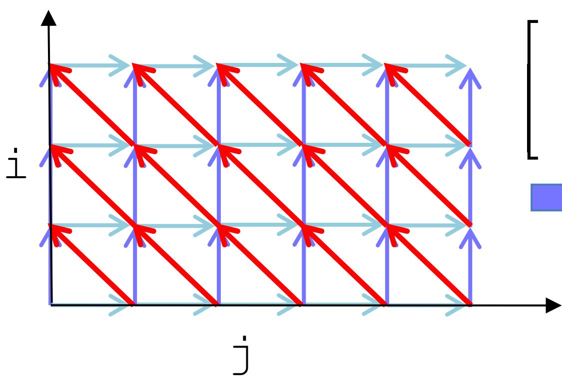


- Sequential execution order: lexicographic order
  - [0,0], [0,1], …, [0,6], [0,7],
    [1,1], …, [1,6], [1,7], …
- A loop transform is legal
  - if all data dependences in the original loop nest are honored with sequential execution in the new loop.
- Skewing by the number of iterations
  → fully permutable (a degenerate case)
  → the iterations execute sequentially

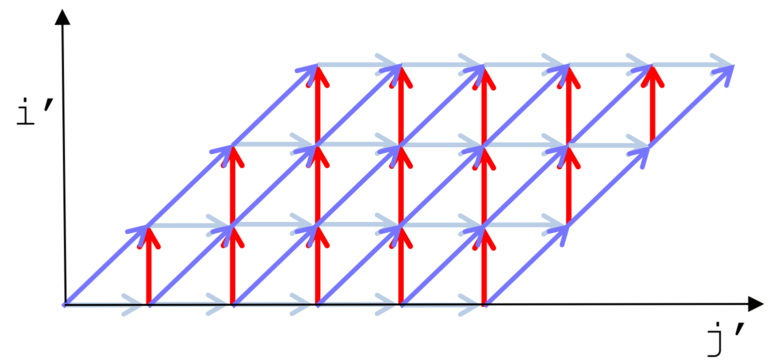
Code Generation

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

```
for i' = 0 TO m
 for j' = i' to i'+n
  X[j'-i'+1]
   =(X[j'-i']+X[j'-i'+1]+X[j'-i'+2])/3
```



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



Transformation

i' = i

j' = i + j

Substitutions: i = i' and j = j'-I'

Loop bounds (Using Fourier-Motzkin Elimination)

0 <= i' <=m

0 <= j'- i' <= n

Summary: Fully Permutable Loop Nests

- Definition:
  A loop nest is fully permutable
  if all the loops in the nest can be permuted arbitrarily
  without changing the semantics of the program

- Affine time partitioning algorithm finds fully permutable loop nests.

- Given rank *r* time mappings:
  - There are *r* independent legal outermost loops
  - Dependences do not point backward along *r*-axes
  - Rank *r* matrix (comprising *r* independent basis vectors) transforms the original loop
    into *r*-deep outermost fully permutable nest

- Generate (*r* -1) dimensional wavefront from permutable loop nest
  (*r* -1 degrees of parallelism)

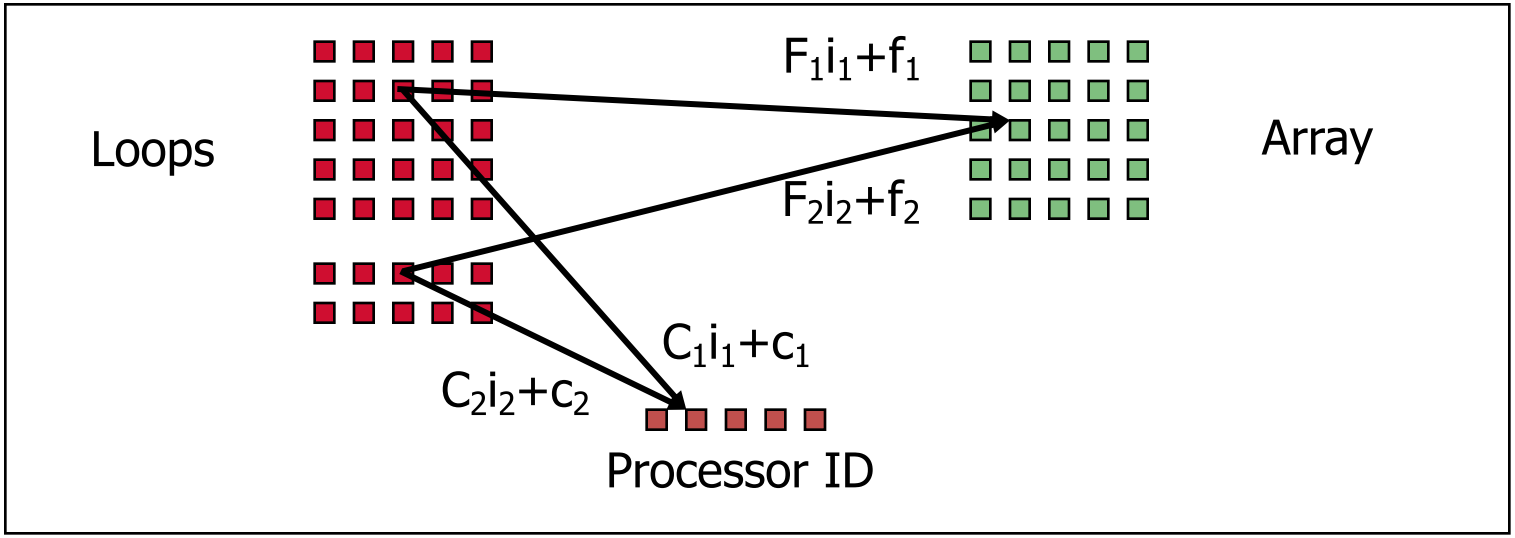
r-Dimensional Pipelineable Parallelism

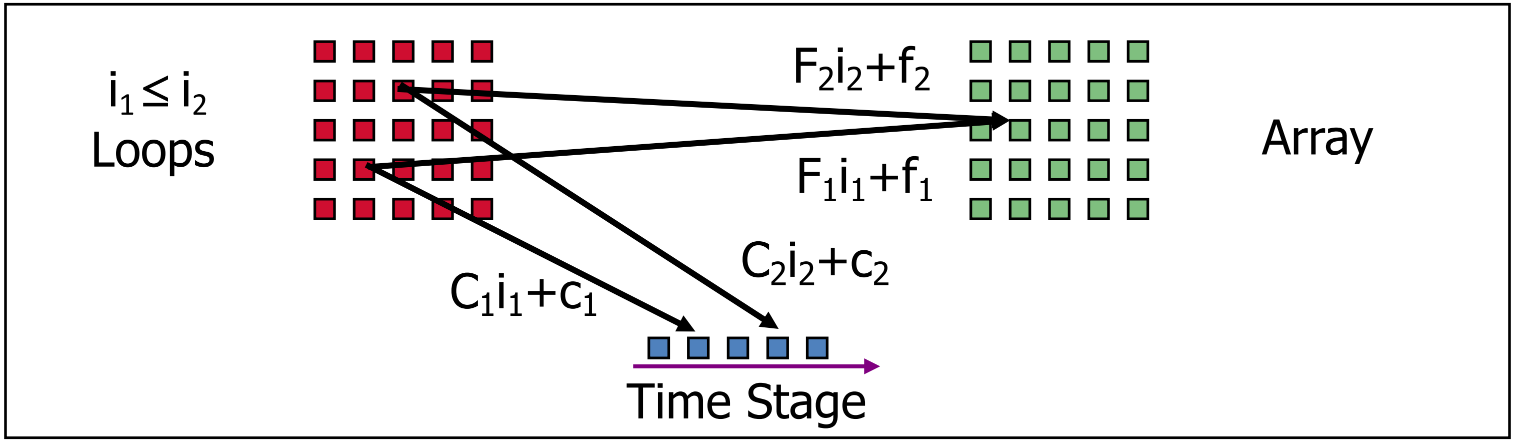
- *r*-dimensions of legal time mapping:
  - *r*-1 degrees of parallelism
    - $O(n^{r-1})$ parallelism, *n* is the number of iterations in each loop
    - $O(n)$ synchronization

- Synchronization
  - processor ID $(p_1, p_2, \ldots, p_{r-1})$:
    - *r*-1 outer loops map to each processor
    - Runs *r*th loop sequentially on each processor
  - iteration i_r for processor $(p_1, p_2, \ldots, p_{r-1})$, waits for its *r*-1 neighbors
    iteration i_r for processors $(p_1-1, p_2, \ldots, p_{r-1})$,
    $(p_1, p_2-1, \ldots, p_{r-1})$, …,
    $(p_1, p_2, \ldots, p_{r-1}-1)$.

3. How to Compute Time Partitioning?

Compare:





Comparing the Two Problems

Communication-Free Parallelism:

C: Space partitioning of Computation to Processor ID

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

Pipelining Parallelism:

C: Time mapping of Computation to Time

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:

Much harder!

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

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

with the objective of maximizing the rank of C_1, C_2

Farkas Lemma Comes to the Rescue!

Finding the possible time dimensions c:

Given matrix A, find a vector c such that

for all vectors x such that $Ax \geq 0$,

$c^T x \geq 0$

Farkas Lemma, 1901 (real domain)

The primal system of inequalities

$Ax \geq 0, \quad c^T x < 0$

has a real-valued solution x

or, the dual system

$A^T y = c, \quad y \geq 0$

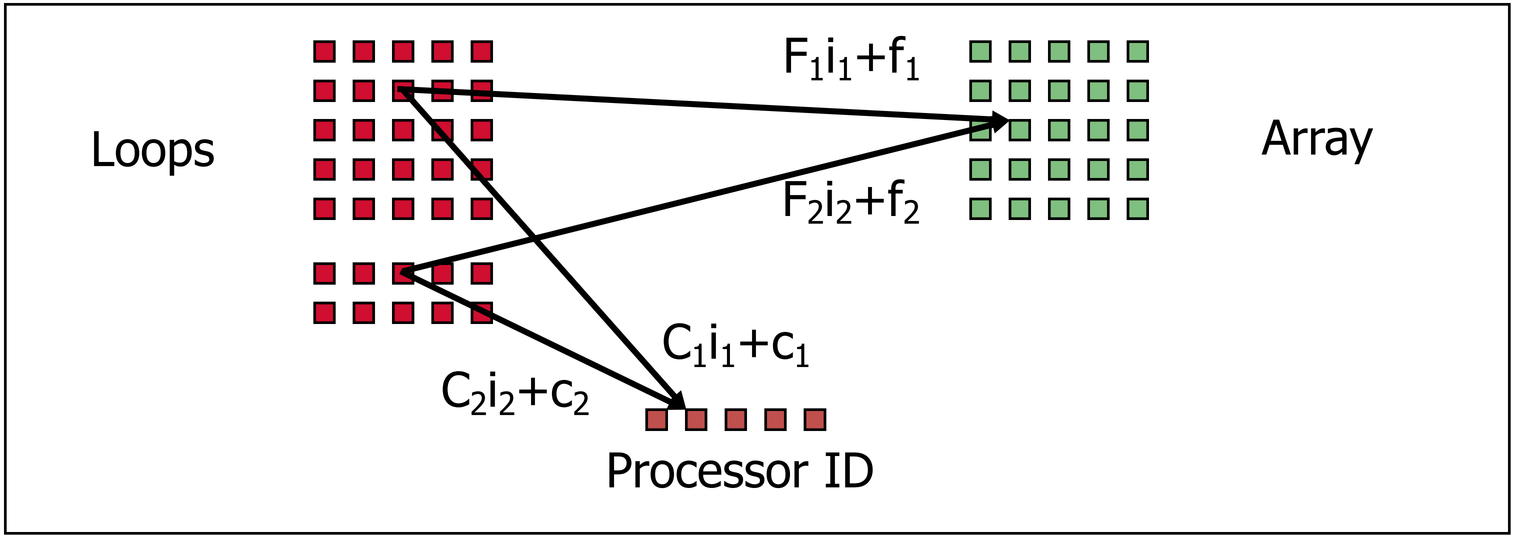
has a real-valued solution y, but never both.

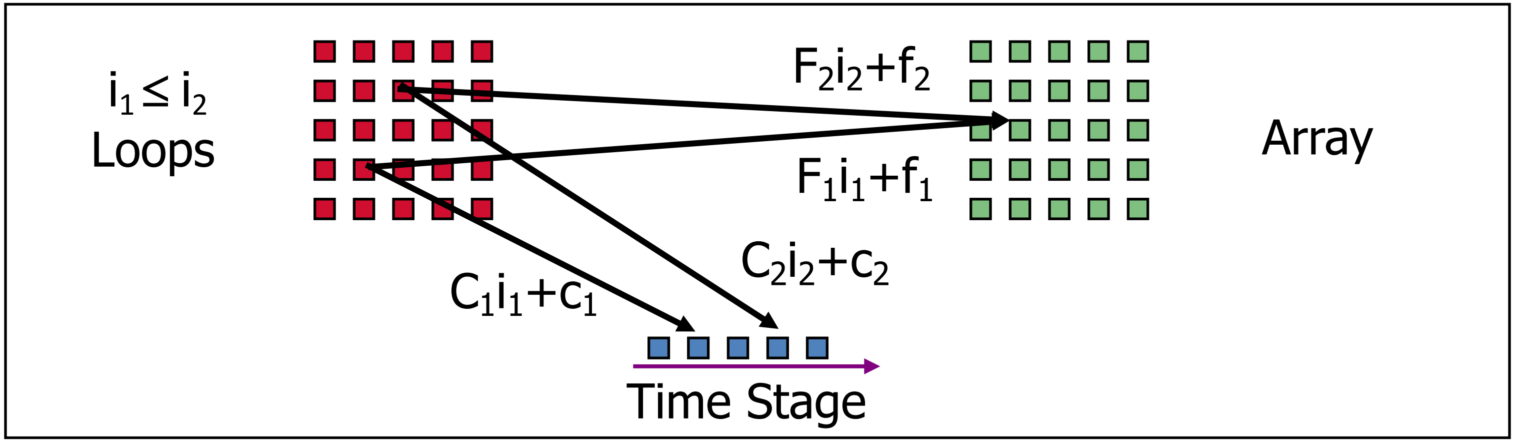
Time partitioning: Find c such that $A^T y = c, \quad y \geq 0$ Much easier!

Note: Farkas Lemma: a theorem of the alternative

(no intuitive proof exists)

Two Key Algorithms





Summary

Note: Time partitioning works for multiple loop nests & imperfect nesting

- Examples in the next class

Affine partitioning: fundamental concepts in parallelism & pipelining

- Can be solved using a mathematical algorithm (by a compiler)

- Can also be applied intuitively to programs by hand
  - Parallelism:
    - Assign all dependent operations to the same processor
    - With as many dimensions as possible
  - Pipelining
    - Find the largest possible outermost permutable loop nests
    - Transform loops so dependences don't point backwards in as many dimensions as possible.