

CSE 6220 INTRODUCTION TO HIGH PERFORMANCE COMPUTING

APPLICATIONS OF PREFIX SUMS

Ümit V. Çatalyürek

School of Computational Science and Engineering
Georgia Institute of Technology

Polynomial Evaluation Problem

- $P(x) = \sum_{i=0}^{n-1} a_i x^i$
- Given a_0, a_1, \dots, a_{n-1} and x_0
- Compute: $P(x_0)$
- Serial Time?
- $T(n, 1) = \Theta(n)$

Polynomial Evaluation Problem

- Assume coefficients are block distributed, i.e., P_i contains $a_{i\frac{n}{p}}, a_{i\frac{n}{p}+1}, \dots, a_{(i+1)\frac{n}{p}-1}$
1. P_0 reads x_0 and broadcast it
 2. Perform n -element parallel prefix with 1 as the first element, x_0 as all other elements, and X (multiplication) as the operation.
 3. Each processor computes $\frac{n}{p}$ terms locally and adds them.
 4. Sum the partial results, one on each processor.

Polynomial Evaluation Problem

1. $O((\tau + \mu) \log p)$
2. Computation time = $\Theta\left(\frac{n}{p} + \log p\right)$
 Communication time = $\Theta((\tau + \mu) \log p)$
3. $O\left(\frac{n}{p}\right)$
4. $O((\tau + \mu) \log p)$

\Downarrow

Computation time = $\Theta\left(\frac{n}{p} + \log p\right)$
 Communication time = $\Theta((\tau + \mu) \log p)$
 Efficient if $p = O\left(\frac{n}{\log n}\right)$

Fibonacci Sequences

$$f_0 = f_1 = 1$$

$$f_i = f_{i-1} + f_{i-2}$$

$$S_i = S_{i-1} + x_i$$

$$\begin{bmatrix} f_i & f_{i-1} \end{bmatrix} = \begin{bmatrix} f_{i-1} & f_{i-2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Fibonacci Sequences

$$V_i = V_{i-1} \times M \text{ where}$$

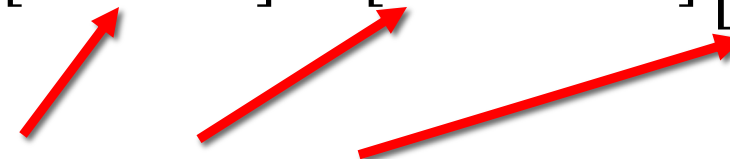
$$V_i = [f_i \quad f_{i-1}] \text{ and}$$

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\Rightarrow V_i = V_1 \times M^{i-1}$$

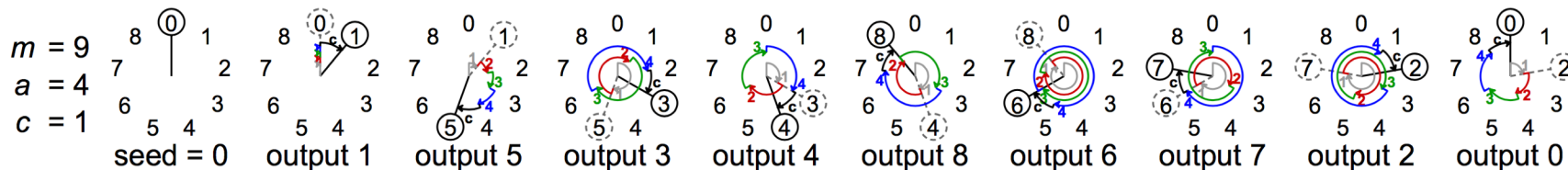
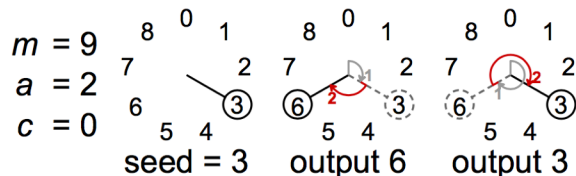
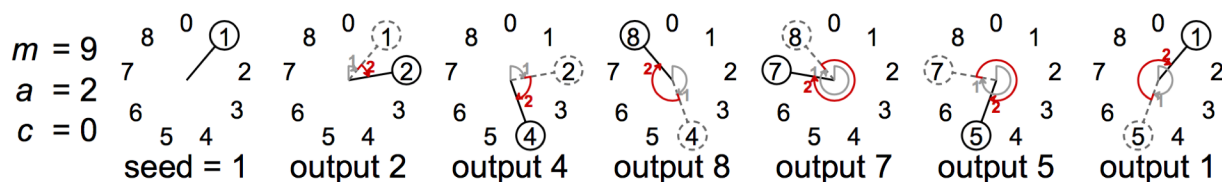
- Compute $M, M^2, M^3, \dots, M^{n-1}$
- Using parallel prefix
 - Operation is matrix multiplication

Generalization

- $x_i = ax_{i-1} + bx_{i-2}$
 - $[x_i \quad x_{i-1}] = [x_{i-1} \quad x_{i-2}] \begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix}$
 - $V_i = V_{i-1} \times M$
- 

Linear Congruential Random Number Generator (1948)

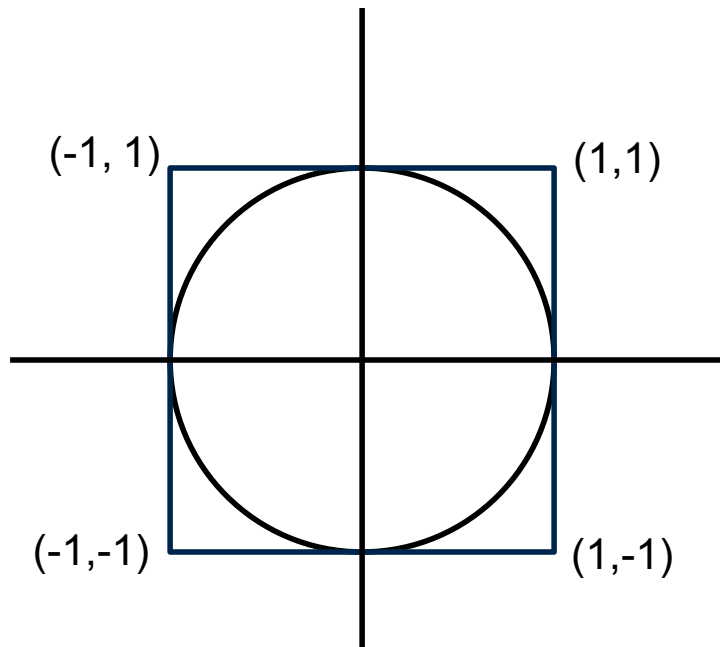
- $x_0 = \text{seed}$
- $x_i = (ax_{i-1} + c) \bmod m$
- $x_0, x_1, x_2, \dots, x_{n-1}$



Linear Congruential Random Number Generator (Lehmer, 1948)

- $x_0 = \textit{seed}$
- $x_i = (ax_{i-1} + c) \bmod m$
- $x_0, x_1, x_2, \dots, x_{n-1}$
- $\begin{bmatrix} x_i & 1 \end{bmatrix} = \begin{bmatrix} x_{i-1} & 1 \end{bmatrix} \begin{bmatrix} a & 0 \\ c & 1 \end{bmatrix} \bmod m$
- $V_i = V_{i-1} \times M \bmod m$

Keeping Results Consistent – Monte Carlo Example

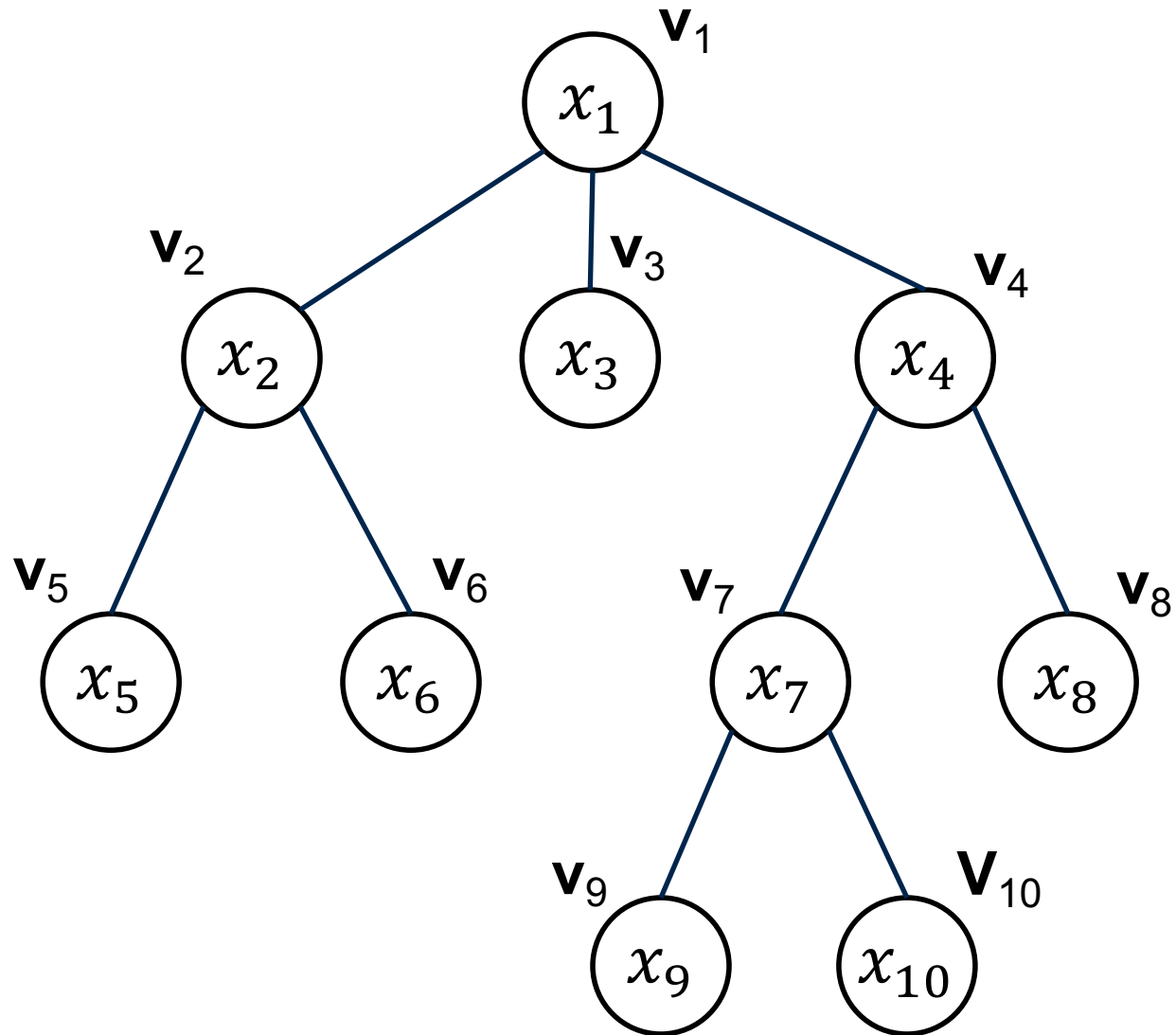


$$\frac{\text{Circle}}{\text{Square}} = \frac{\pi}{4}$$

Keeping Results Consistent – Monte Carlo Example

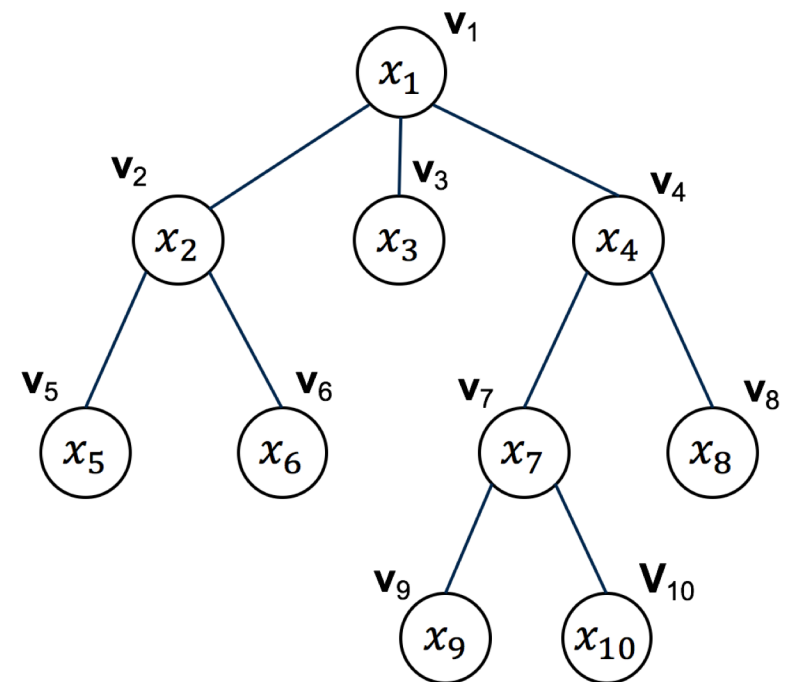
- In practice, many applications do not know in advance how many random numbers they eventually consume.
- To accommodate, random numbers are often generated in a way that corresponds to interleaved decomposition.
- With interleaved decomposition, one has to be careful to ensure output of the program remains the same regardless of number of processors used.
- In the example of computing π using the Monte Carlo method, two consecutive random numbers are used to create a random point. The parallel algorithm has to take this into account to ensure the points generated are the same whether running in serial or parallel, and regardless of how many processors are used.

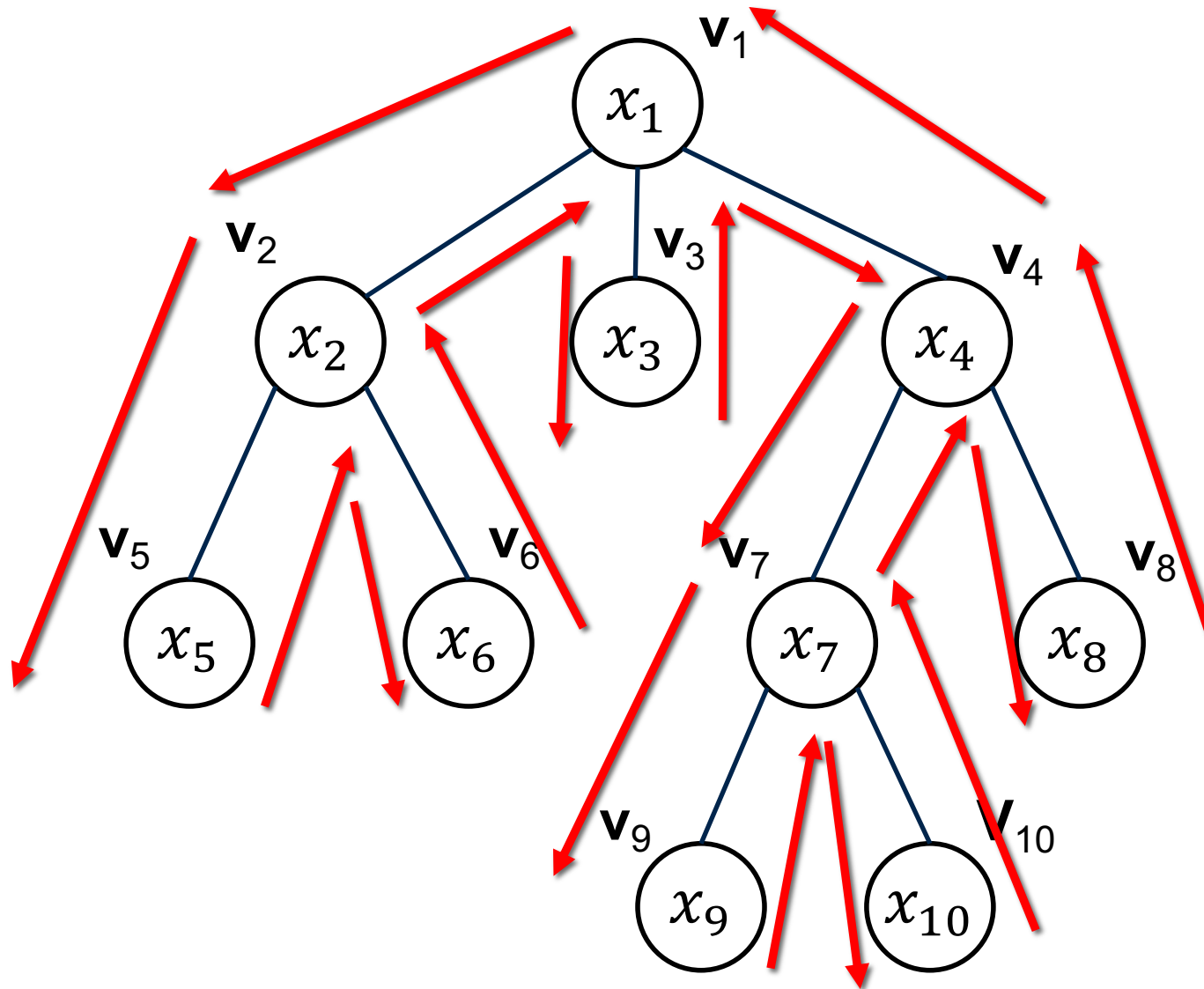
Tree Accumulation



Tree Accumulation

- **Upward Accumulation (UA):** For a node v , sum all numbers in the subtree
- $UA(v_4) = x_4 + x_7 + x_8 + x_9 + x_{10}$
- Do this for all nodes
- **Downward Accumulation (DA):** For each node v , sum all numbers in ancestors of v
- $DA(v_7) = x_7 + x_4 + x_1$
- Serial Computation Time?
- $T(n, 1) = \Theta(n)$





Euler Tour: $v_1 v_2 v_5 v_2 v_6 v_2 v_1 v_3 v_1 v_4 v_7 v_9 v_7 v_{10} v_7 v_4 v_8 v_4 v_1$

Tree Accumulation: UA

- ET: $v_1 v_2 v_5 v_2 v_6 v_2 v_1 v_3 v_1 v_4 v_7 v_9 v_7 v_{10} v_7 v_4 v_8 v_4 v_1$
- $|ET| = 1 + 2(n - 1) = 2n - 1$
- Assume ET is block partitioned among processors.
 - First occurrence of v_i put x_i
 - Otherwise put 0

v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
x_1	x_2	x_5	0	x_6	0	0	x_3	0	x_4	x_7	x_9	0	x_{10}	0	0	x_8	0	0

- PS: resulting prefix sums array
- i_f : index of first occurrence of v
- i_l : index of last occurrence of v
- $UA(v) = PS[i_l] - PS[i_f - 1]$

Tree Accumulation: DA

- ET: $v_1 v_2 v_5 v_2 v_6 v_2 v_1 v_3 v_1 v_4 v_7 v_9 v_7 v_{10} v_7 v_4 v_8 v_4 v_1$
- Assume ET is block partitioned among processors.
 - First occurrence of v_i put x_i
 - Last occurrence of v_i put $-x_i$
 - Otherwise put 0

v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
x_1	x_2	x_5	0	x_6	$-x_2$	0	x_3	0	x_4	x_7	x_9	0	x_{10}	$-x_7$	0	x_8	$-x_4$	$-x_1$

- PS: resulting prefix sums array
- i_f : index of first occurrence of v
- $DA(v) = PS[i_f]$
- Correct?
- Not really: leaf nodes, their values are not subtracted.

Tree Accumulation: DA

- ET: $v_1 v_2 v_5 v_2 v_6 v_2 v_1 v_3 v_1 v_4 v_7 v_9 v_7 v_{10} v_7 v_4 v_8 v_4 v_1$
- Assume ET is block partitioned among processors.
 - Internal nodes:
 - First occurrence of v_i put x_i
 - Last occurrence of v_i put $-x_i$
 - Otherwise put 0
 - Leaf nodes: put 0

v_1	v_2	v_5	v_2	v_6	v_2	v_1	v_3	v_1	v_4	v_7	v_9	v_7	v_{10}	v_7	v_4	v_8	v_4	v_1
x_1	x_2	0	0	0	$-x_2$	0	0	0	x_4	x_7	0	0	0	$-x_7$	0	0	$-x_4$	$-x_1$

- PS: resulting prefix sums array
- i_f : index of first occurrence of v
- If v_i is an internal node
 - $DA(v_i) = PS[i_f]$
- If v_i is a leaf node
 - $DA(v_i) = PS[i_f] + x_i$

Tree Accumulation: Algorithm

1. Create array A
 1. Right-shift permutation on last node in subarray of ET
 2. Left-shift permutation on first node in subarray of ET
 3. Create location portion of A (slide 14 & 16)
2. Compute PS from A using parallel prefix
3. Compute answers using formula (slide 14 & slide 16)

Complexity?

1. Communication time = $\Theta(\tau + \mu)$
 Computation time = $\Theta\left(\frac{n}{p}\right)$
2. Computation time = $\Theta\left(\frac{n}{p} + \log p\right)$
 Communication time = $\Theta((\tau + \mu) \log p)$
3. Computation time = $\Theta\left(\frac{n}{p}\right)$

Sequence Alignment

- An important problem in computational biology.
- DNA sequences: Strings over {A,C,G,T}.
- Goal: To find out how “well” the sequences align.
- Alignment: Stacking chars of each sequence into columns.
- Gaps (-) may be inserted for missing characters.

Example alignment and score computation

- Alignment has a score that shows quality.
- Every column of an alignment is a match, mismatch or a gap.
- Matches are preferred and hence have a positive score, others have a negative score.
- Example: If match = 1, mismatch = 0 and gap = -1, then for the following alignment

A	T	G	A	–	C	C
A	–	G	A	A	T	C
1	-1	1	1	-1	0	1

$$\text{Score}(\text{ATGACC}, \text{AGAATC}) = 2$$

Problem definition

Input:

- Two sequences $A = a_1, a_2, \dots, a_m$ and $B = b_1, b_2, \dots, b_n$
- Scores for match/mismatch ($f(a, b)$) and gap (g)

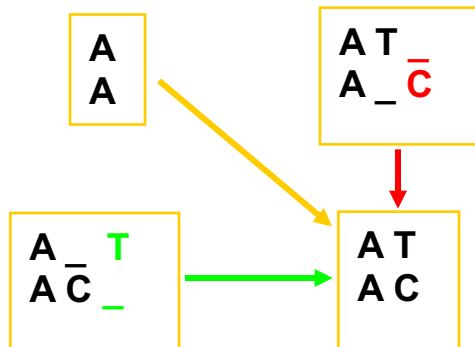
Output:

Dynamic programming solution:

- T = Table of size $(m + 1) \times (n + 1)$
- $T[i, j]$ = best score between a_1, a_2, \dots, a_i and b_1, b_2, \dots, b_j
- $$T[i, j] = \max \begin{cases} T[i - 1, j - 1] + f(a_i, b_j) \\ T[i - 1, j] - g \\ T[i, j - 1] - g \end{cases}$$
- Sequential time $= O(mn)$

Dynamic programming matrix

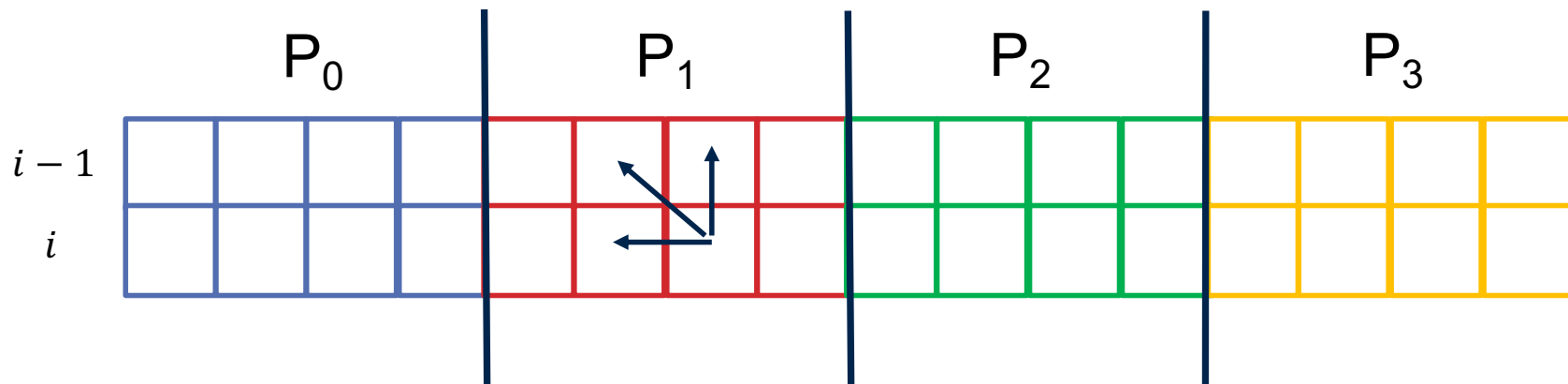
$$T[i, j] = \max \begin{cases} T[i-1, j-1] + f(a_i, b_j) \\ T[i-1, j] - g \\ T[i, j-1] - g \end{cases}$$



	0	1	2	3	4	5	6
0		Gap	A	T	G	C	T
1	Gap	0	-1	-2	-3	-4	-5
2	A	-1	2 ⁽²⁾	1 ⁽⁰⁾	0	0	0
3	C	-2	1 ⁽⁰⁾	2 ⁽⁰⁾	1	2	1
4	C	-3	0	1 ⁽⁰⁾	2	3	2
5	G	-4	0	0	3 ⁽²⁾	2	3
6	C	-5	0	0	2	5 ⁽²⁾	4
7	T	-6	0	2	1	4	7 ⁽²⁾

Solution using parallel prefix

- We compute each row of T in parallel:



- $$w[j] = \max \begin{cases} T[i-1, j-1] + f(a_i, b_j) \\ T[i-1, j] - g \end{cases}$$

- $$T[i, j] = \max \begin{cases} w[j] \\ T[i, j-1] - g \end{cases}$$

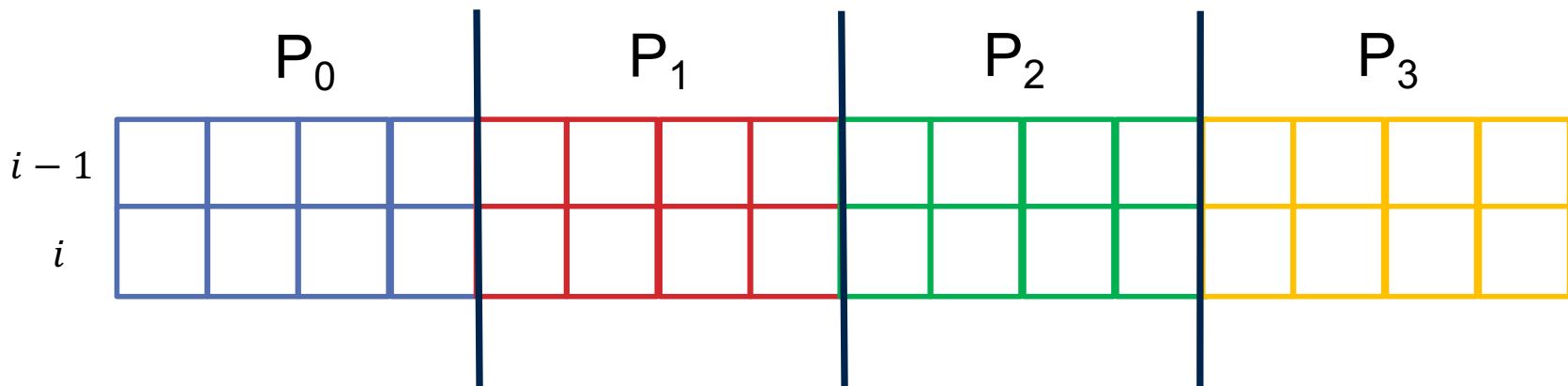
Solution using parallel prefix

- $x[j] = T[i, j] + j \cdot g$
- $x[j] = \max \begin{cases} w[j] + jg \\ T[i, j-1] - g + jg \end{cases}$
- $x[j] = \max \begin{cases} w[j] + jg \\ T[i, j-1] + (j-1)g \end{cases}$
- $x[j] = \max \begin{cases} w[j] + jg \\ x[j-1] \end{cases}$

Parallel Algorithm: Compute row i from $i-1$

1. Use right shift to send last element of $(i - 1)^{\text{th}}$ row on each processor. $O(\tau + \mu)$
2. Create vector A

$$A[j] = jg + \max \begin{cases} T[i - 1, j - 1] + f(a_i, b_j) \\ T[i - 1, j] - g \end{cases} \quad O\left(\frac{n}{p}\right)$$



Parallel Algorithm: Compute row i from $i-1$

1. Use right shift to send last element of $(i - 1)^{\text{th}}$ row on each processor. $O(\tau + \mu)$

2. Create vector A

$$A[j] = jg + \max \begin{cases} T[i - 1, j - 1] + f(a_i, b_j) \\ T[i - 1, j] - g \end{cases} \quad O\left(\frac{n}{p}\right)$$

3. Compute parallel prefix on A using max as operator and store results in x

4. Compute $T[i, j]$'s using $O\left(\frac{n}{p} + (\tau + \mu) \log p\right)$

$$T[i, j] = x[j] - jg \quad O\left(\frac{n}{p}\right)$$

$$\text{Computation Time} = O\left(\frac{mn}{p} + m \log p\right)$$

$$\text{Communication Time} = O((\tau + \mu)m \log p)$$

$$\text{Efficient when } p = O\left(\frac{n}{\log n}\right)$$