

# Parallel Prefix Algorithms

1. A Secret to turning serial into parallel
2. Suppose you bump into a parallel algorithm that surprises you → “there is no way to parallelize this algorithm” you say
3. Probably a variation on parallel prefix!

## Sum Prefix

Input  $x = (x_1, x_2, \dots, x_n)$

Output  $y = (y_1, y_2, \dots, y_n)$

$$y_i = \sum_{j=1}^i x_j$$

## Example

$x = (1, 2, 3, 4, 5, 6, 7, 8)$

$y = (1, 3, 6, 10, 15, 21, 28, 36)$

Prefix Functions-- outputs depend upon an *initial* string

**Clearly only  
Serial**

**Right?**

Prefix Functions -- outputs depend upon an *initial* string

Suffix Functions -- outputs depend upon a *final* string

Other Notations

+\  
“plus scan” APL

MPI\_scan

$$y = \begin{bmatrix} 1 \\ 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} x$$

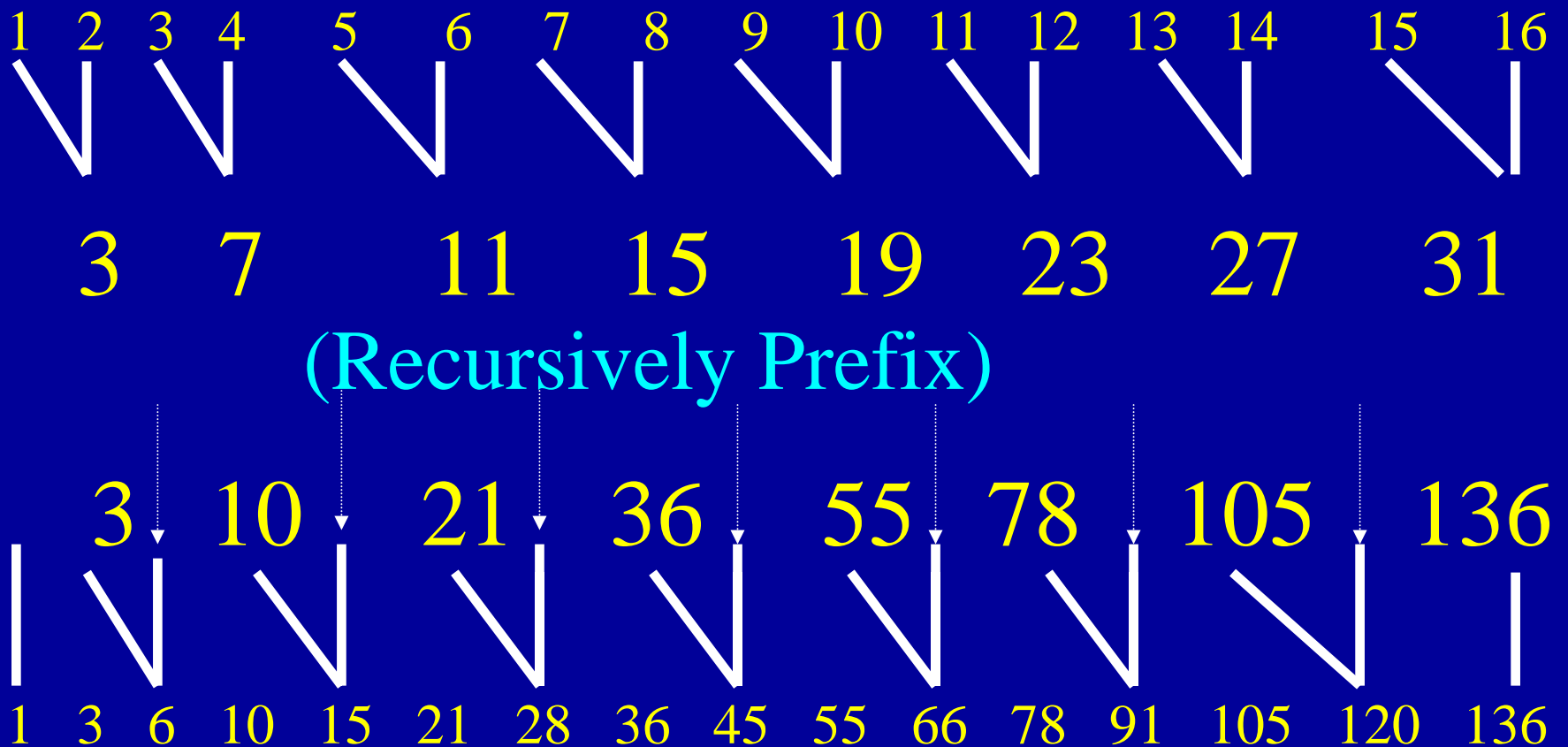
```
y(1) = x(1)
for i = 2 : n
    y(i) = x(i) + y(i-1)
end
```



**MATLAB:** y=cumsum(x)

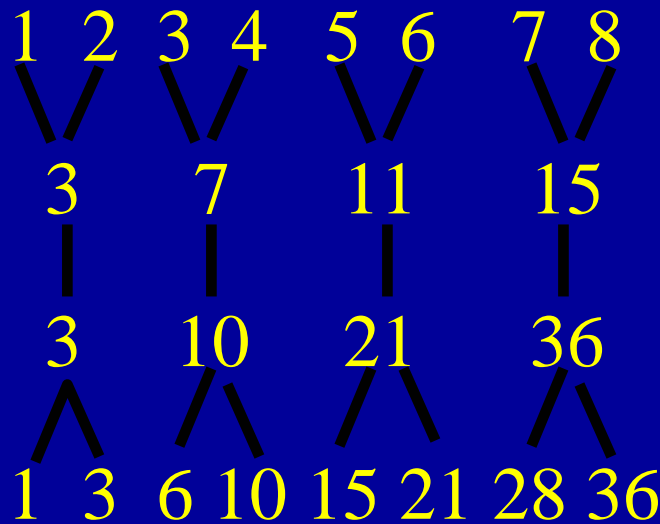
# $2\log_2 n$ Parallel Steps on $n$ Processors

Algorithm: 1. Pairwise sum    2. Recursively Prefix    3. Pairwise Sum



# Parallel Prefix

prefix( [1 2 3 4 5 6 7 8] ) = [1 3 6 10 15 21 28 36]



Pairwise sums

Recursive prefix

Update "odds"

- Any associative operator
- AKA: +\ (APL), cumsum(Matlab), MPI\_SCAN,

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

## Notice

# adds =  $2n$

# required =  $n$

Parallelism at the cost of  
more work!

# Any Associative Operation “ $\oplus$ ” on any inputs works

Associative:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

Sum (+)

Product (\*)

Max

Min

Input: Reals

All (=and) “ $\wedge$ ”

Any (= or) “ $\vee$ ”

Input: Bits  
(Boolean)

MatMul

Inputs: Matrices

# Fibonacci via Matrix Multiply Prefix

$$F_{n+1} = F_n + F_{n-1}$$

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

Can compute all  $F_n$  by matmul\_prefix on

$$\left[ \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right]$$

then select the upper left entry

# Arithmetic Modulo 2 (binary arithmetic)

$0+0=0$	$0*0=0$
$0+1=1$	$0*1=0$
$1+0=1$	$1*0=0$
$1+1=0$	$1*1=1$
Add = exclusive or	Mult = and

# Carry-Look Ahead Addition (Babbage 1800's)

Goal: Add Two n-bit Integers

Example						
1	0	1	1	1	Carry	
1	0	1	1	1	First Int	
1	0	1	0	1	Second Int	
<hr/>						
1	0	1	1	0	0	Sum

# Carry-Look Ahead Addition (Babbage 1800's)

Goal: Add Two n-bit Integers

Example							Notation				
1	0	1	1	1		Carry	$c_2$	$c_1$	$c_0$		
1	0	1	1	1	1	First Int	$a_3$	$a_2$	$a_1$	$a_0$	
1	0	1	0	1		Second Int	$a_3$	$b_2$	$b_1$	$b_0$	
<hr/>							<hr/>				
1	0	1	1	0	0	Sum	$s_3$	$s_2$	$s_1$	$s_0$	

# Carry-Look Ahead Addition (Babbage 1800's)

Goal: Add Two n-bit Integers

Example		Notation
1 0 1 1 1	Carry	$c_2$ $c_1$ $c_0$
1 0 1 1 1	First Int	$a_3$ $a_2$ $a_1$ $a_0$
1 0 1 0 1	Second Int	$a_3$ $b_2$ $b_1$ $b_0$
<hr/>		<hr/>
1 0 1 1 0 0	Sum	$s_3$ $s_2$ $s_1$ $s_0$

$c_{-1} = 0$  (addition mod 2)

for  $i = 0 : n-1$

$$s_i = a_i + b_i + c_{i-1}$$

$$c_i = a_i b_i + c_{i-1}(a_i + b_i)$$

end

$$s_n = c_{n-1}$$

# Carry-Look Ahead Addition (Babbage 1800's)

Goal: Add Two n-bit Integers

Example							Notation			
1	0	1	1	1		Carry	$c_2$	$c_1$	$c_0$	
1	0	1	1	1		First Int	$a_3$	$a_2$	$a_1$	$a_0$
1	0	1	0	1		Second Int	$a_3$	$b_2$	$b_1$	$b_0$
<hr/>						Sum	$s_3$	$s_2$	$s_1$	$s_0$
1	0	1	1	0	0					

$c_{-1} = 0$  (addition mod 2)

for  $i = 0 : n-1$

$$s_i = a_i + b_i + c_{i-1}$$

$$c_i = a_i b_i + c_{i-1}(a_i + b_i)$$

end

$$s_n = c_{n-1}$$

$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} a_i + b_i & a_i b_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix}$$

# Carry-Look Ahead Addition (Babbage 1800's)

Goal: Add Two n-bit Integers

Example		Notation
1 0 1 1 1	Carry	$c_2 \quad c_1 \quad c_0$
1 0 1 1 1	First Int	$a_3 \quad a_2 \quad a_1 \quad a_0$
1 0 1 0 1	Second Int	$a_3 \quad b_2 \quad b_1 \quad b_0$
1 0 1 1 0 0	Sum	$s_3 \quad s_2 \quad s_1 \quad s_0$

$c_{-1} = 0$  (addition mod 2)

for  $i = 0 : n-1$

$$s_i = a_i + b_i + c_{i-1}$$

$$c_i = a_i b_i + c_{i-1}(a_i + b_i)$$

end

$$s_n = c_{n-1}$$

$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} a_i + b_i & a_i b_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix}$$

Matmul prefix with binary arithmetic is equivalent to carry-look ahead!

Compute  $c_i$  by prefix, then

$s_i = a_i + b_i + c_{i-1}$  in parallel

# Tridiagonal Factor

$$\mathbf{T} = \begin{bmatrix} a_1 & b_1 & & & \\ c_1 & a_2 & b_2 & & \\ & c_2 & a_3 & b_3 & \\ & & c_3 & a_4 & b_4 \\ & & & c_4 & a_5 \end{bmatrix}$$

Determinants ( $D_0=1, D_1=a_1$ )  
 ( $D_k$  is the det of the  $k \times k$  upper left):

$$\begin{bmatrix} D_{n-1} & & \\ & D_n & \end{bmatrix}$$

$$D_n = a_n D_{n-1} - b_{n-1} c_{n-1} D_{n-2}$$

Compute  $D_n$  by `matmul_prefix`

$$\rightarrow \begin{bmatrix} D_n \\ D_{n-1} \end{bmatrix} = \begin{bmatrix} a_n & -b_{n-1}c_{n-1} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} D_{n-1} \\ D_{n-2} \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} 1 & & & \\ l_1 & 1 & & \\ & l_2 & 1 & \\ & & & \ddots \end{bmatrix} \begin{bmatrix} d_1 & b_1 \\ & d_2 & b_2 \\ & & d_3 & b_3 \\ & & & \ddots \end{bmatrix}$$

$$d_n = D_n / D_{n-1}$$

$$l_n = c_n / d_n$$

3 embarrassing  
 Parallels + prefix

# The “Myth” of $\log n$

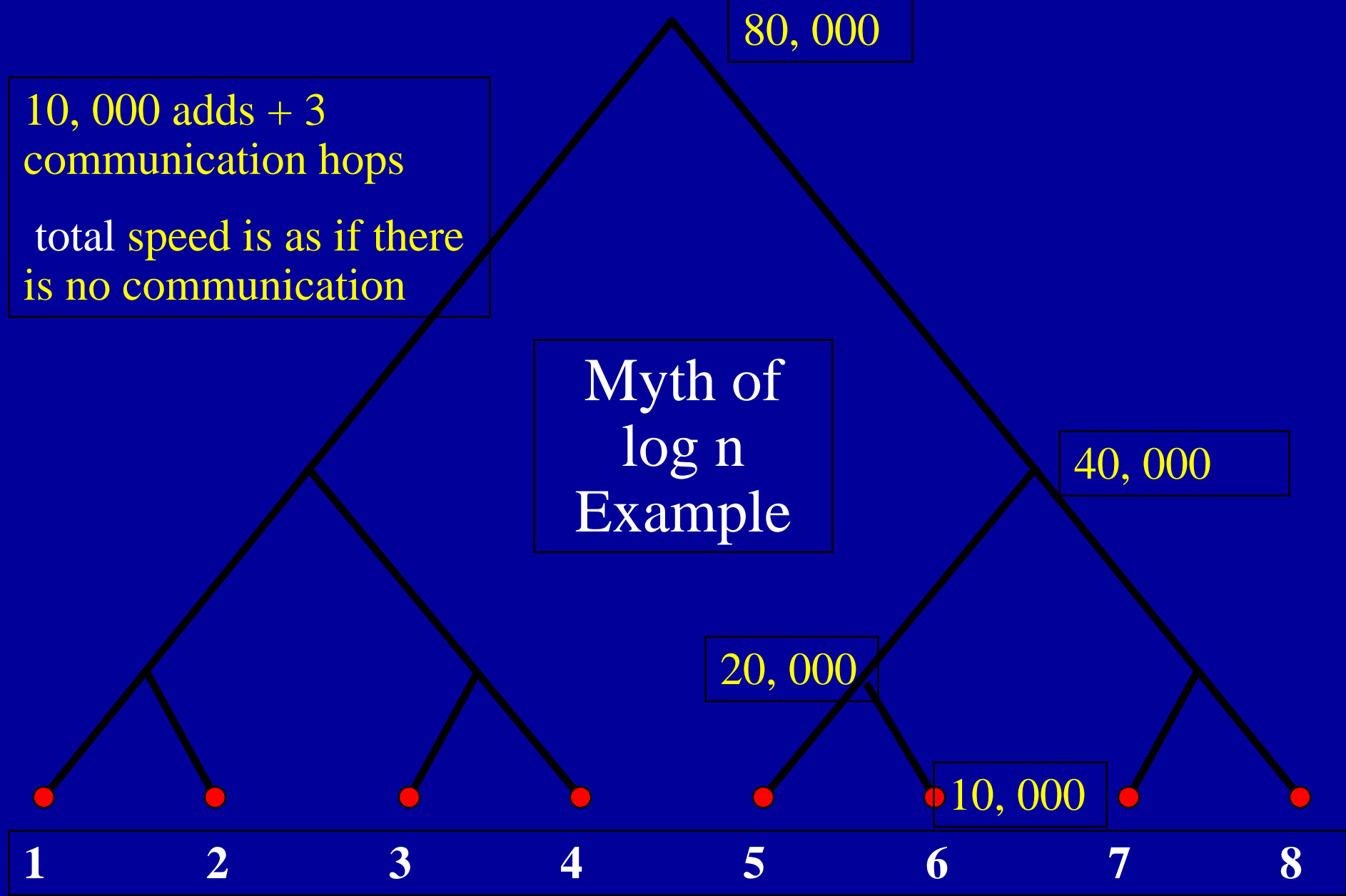
The  **$\log_2 n$  parallel steps** is not the main reason for the usefulness of parallel prefix.

Say  $n = 1000p$  (1000 summands per processor)

Time = (2000 adds) + ( $\log_2 P$  message passings)

 fast & embarrassingly parallel

(2000 local adds are serial for each processor of course)



$\log_2 n = \text{number of steps to add } n \text{ numbers (NO!!)}$

Any Prefix  
Operation May  
Be Segmented!

# Segmented Operations

Inputs = Ordered Pairs  
(operand, boolean)

e.g. (x, T) or (x, F)

Change of  
segment indicated  
by switching T/F

$\oplus_2$	(y, T)	(y, F)
(x, T)	( $x \oplus y$ , T)	(y, F)
(x, F)	(y, T)	( $x \oplus y$ , F)

e. g.	1	2	3	4	5	6	7	8
	T	T	F	F	F	T	F	T
Result	1	3	3	7	12	6	7	8

**Copy Prefix:**  $\mathbf{x} \oplus \mathbf{y} = \mathbf{x}$

(is associative)

**Segmented**

<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
<b>T</b>	<b>T</b>	<b>F</b>	<b>F</b>	<b>F</b>	<b>T</b>	<b>F</b>	<b>T</b>
<b>1</b>	<b>1</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>6</b>	<b>7</b>	<b>8</b>

# High Performance Fortran

SUM\_PREFIX ( ARRAY, DIM, MASK, SEG, EXC)

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix} \quad M = \begin{bmatrix} T & T & T & T & T \\ F & F & T & T & T \\ T & F & T & F & F \end{bmatrix}$$

$$\text{SUM\_PREFIX}(A) = \begin{bmatrix} 1 & 20 & 42 & 67 & 45 \\ 7 & 27 & 50 & 76 & 105 \\ 18 & 39 & 63 & 90 & 120 \end{bmatrix}$$

$$\begin{array}{l} \text{SUM\_SUFFIX}(A) \\ \text{SUM\_PREFIX}(A, \text{DIM} = 2) \end{array} = \begin{bmatrix} 1 & 3 & 6 & 10 & 15 \\ 6 & 13 & 21 & 30 & 40 \\ 11 & 23 & 36 & & \end{bmatrix}$$

$$\text{SUM\_PREFIX}(A, \text{MASK} = M) = \begin{bmatrix} 1 & 14 & 17 & . \\ 1 & 14 & 25 & . \\ 12 & 14 & 38 & \end{bmatrix}$$

# More HPF Segmented

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} \mathbf{T} & \mathbf{T} & \mathbf{F} & \mathbf{F} & \mathbf{F} \\ \mathbf{F} & \mathbf{T} & \mathbf{T} & \mathbf{F} & \mathbf{F} \\ \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} \end{bmatrix}$$

**T T | F | T T | F F**

**Sum\_Prefix (A, SEGMENTS = S)**

$$\begin{bmatrix} 1 & 13 & 3 \\ 6 & 20 \\ 11 & 32 \end{bmatrix}$$

# Example of Exclusive

$A = [1 \quad 2 \quad 3 \quad 4 \quad 5]$

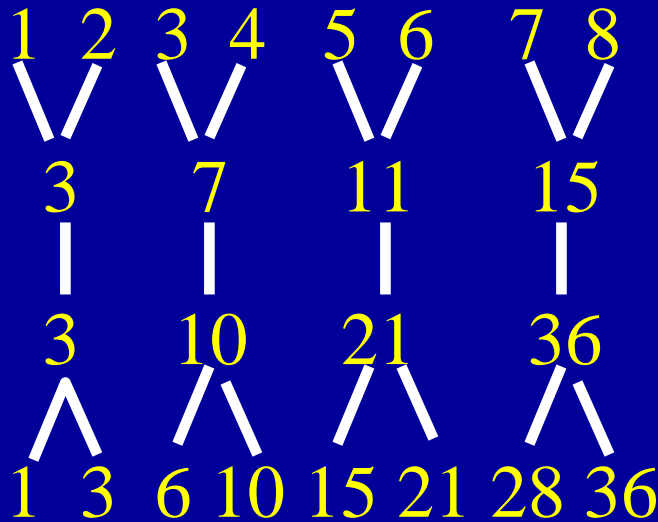
$\text{Sum\_Prefix}(A) \quad [1 \quad 3 \quad 6 \quad 10 \quad 15]$

$\text{Sum\_Prefix}(A, \text{EXCLUSIVE} = \text{TRUE})$   
 $[0 \quad 1 \quad 3 \quad 6 \quad 10]$

(Exclusive: Don't count myself)

# Parallel Prefix

prefix( [1 2 3 4 5 6 7 8] ) = [1 3 6 10 15 21 28 36]



Pairwise sums

Recursive prefix

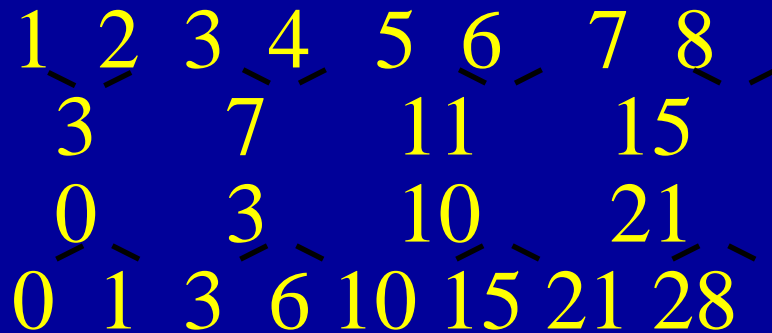
Update “evens”

- Any associative operator
- AKA: +\ (APL), cumsum(Matlab), MPI\_SCAN,

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

# Variations on Prefix

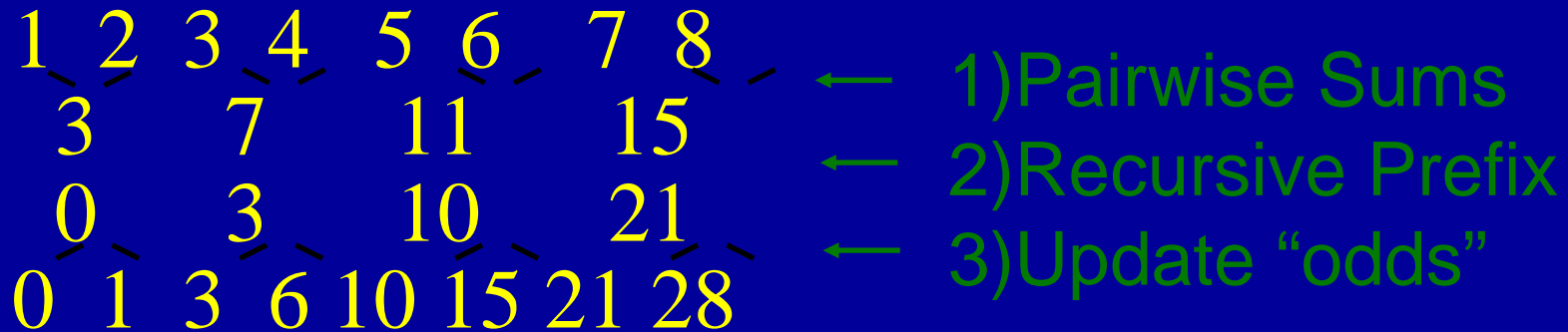
exclusive([1 2 3 4 5 6 7 8])=[0 1 3 6 10 15 21 28]



- ← 1) Pairwise Sums
- ← 2) Recursive Prefix
- ← 3) Update "odds"

# Variations on Prefix

exclusive( [1 2 3 4 5 6 7 8] ) = [0 1 3 6 10 15 21 28]

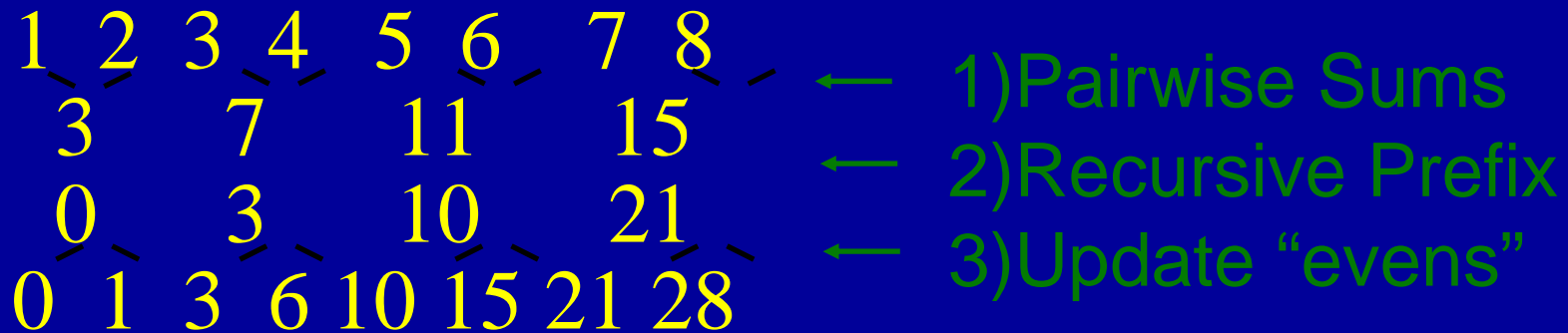


## The Family...

Directions	Inclusive Exc=0	Exclusive Exc=1	
Left	Prefix	Exc Prefix	

# Variations on Prefix

exclusive( [1 2 3 4 5 6 7 8] ) = [0 1 3 6 10 15 21 28]



## The Family...

Directions	Inclusive Exc=0	Exclusive Exc=1	
Left	Prefix	Exc Prefix	
Right	Suffix	Exc Suffix	

# Variations on Prefix

reduce( [1 2 3 4 5 6 7 8] ) = [36 36 36 36 36 36 36 36]

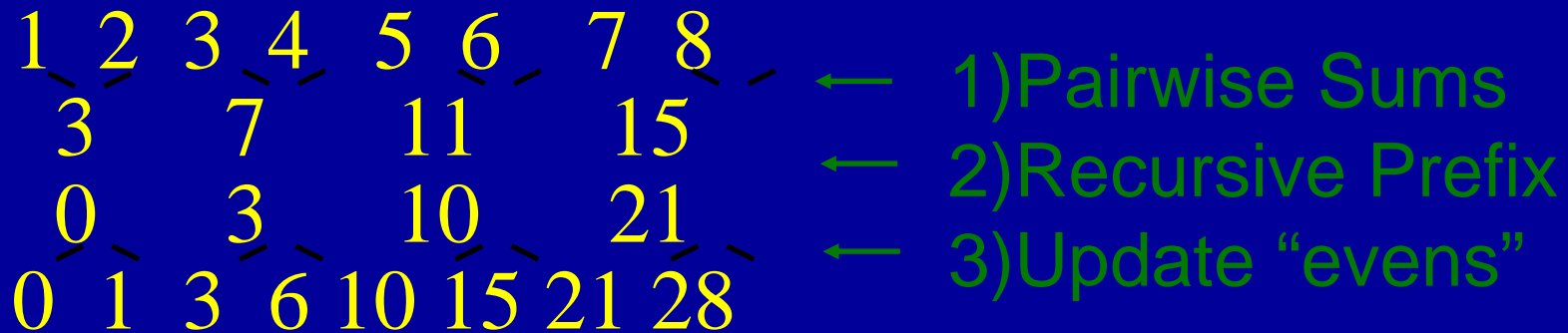


## The Family...

Directions	Inclusive Exc=0	Exclusive Exc=1
Left	Prefix	Exc Prefix
Right	Suffix	Exc Suffix
Left/Right	Reduce	Exc Reduce

# Variations on Prefix

exclusive( [1 2 3 4 5 6 7 8] ) = [0 1 3 6 10 15 21 28]



## The Family...

Directions	Inclusive Exc=0	Exclusive Exc=1	Neighbor Exc Exc=2
Left	Prefix	Exc Prefix	Left Multipole
Right	Suffix	Exc Suffix	Right " " "
Left/Right	<b>Reduce</b>	Exc Reduce	<b>Multipole</b>

# Multipole in 2d or 3d etc

Notice that left/right generalizes more readily to higher dimensions  
Ask yourself what Exc=2 looks like in 3d

## The Family...

<b>Directions</b>	<b>Inclusive Exc=0</b>	<b>Exclusive Exc=1</b>	<b>Neighbor Exc Exc=2</b>
Left	Prefix	<b>Exc Prefix</b>	Left Multipole
Right	Suffix	Exc Suffix	Right " " "
Left/Right	<b>Reduce</b>	Exc Reduce	<b>Multipole</b>

# Not Parallel Prefix but PRAM

Only concerned with minimizing  
parallel time (not communication)

Arbitrary number of processors

# Csanky's (1977) Matrix Inversion

Lemma 1:  $(\triangleleft^{-1})$  in  $O(\log^2 n)$  (triangular matrix inv)

Proof Idea: 
$$\begin{bmatrix} A & 0 \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -B^{-1}CA^{-1} & B^{-1} \end{bmatrix}$$

Lemma 2: *Cayley - Hamilton*

$$p(x) = \det(xI - A) = x^n + c_1x^{n-1} + \dots + c_n$$

( $c_n = \det A$ )

$$0 = p(A) = A^n + c_1A^{n-1} + \dots + c_nI$$

$$A^{-1} = (A^{n-1} + c_1A^{n-2} + \dots + c_{n-1})(-1/c_n)$$

**Powers of A via Parallel Prefix**

### Lemma 3: *Leverier's Lemma*

$$\begin{bmatrix} 1 & & & & \\ s_1 & 2 & & & \\ s_2 & s_1 & \cdot & & \\ \vdots & \vdots & \cdot & \cdot & \\ s_{n-1} & \cdot & \cdot & s_1 & n \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = - \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{bmatrix} \quad s_k = \text{tr}(A^k)$$

*Csanky*

- 1) Parallel Prefix powers of A
- 2)  $s_k$  by directly adding diagonals
- 3)  $c_i$  from lemas 1 and 3
- 4)  $A^{-1}$  obtained from lemma 2

Horrible for  $A=3I$  and  $n>50$  !!

Matrix multiply can be done in  $\log n$  steps  
on  $n^3$  processors with the pram model

Can be useful to think this way, but must  
also remember how real machines are  
built!