18.337 Parallel Prefix

The Parallel Prefix Method

- This is our first example of a parallel algorithm
- Watch closely what is being optimized for
  - Parallel steps
- Beautiful idea with surprising uses
- Not sure if the parallel prefix method is used much in the real world
  - Might maybe be inside MPI scan
  - Might be used in some SIMD and SIMD like cases
- The real key: What is it about the real world that differs from the naïve mental model of parallelism?

Students early mental models

- Look up or figure out how to do things in parallel
- Then we get speedups!
  - NOT!

Parallel Prefix Algorithms

1. A theoretical (may or may not be practical) 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!

Example of a prefix

Sum Prefix

Input $x = (x1, x2, \ldots, xn)$

Output $y = (y1, y2, \ldots, yn)$

$$y_i = \Sigma_{j=1:I}\, x_j$$

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

What do you think?

- Can we really parallelize this?
- It looks like this sort of code:

```
y=0;
for i=2:n, y(i)=y(i-1)+x(i); end
```

- The ith iteration of the loop is not at all decoupled from the (i-1)st iteration.
- Impossible to parallelize right?

A clue?

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

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

Is there any value in adding, say, 4+5+6+7?

Note if we separately have 1+2+3, what can we do?

Suppose we added 1+2, 3+4, etc. pairwise, what could we do?

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

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

Other Notations

1. +\ “plus scan” APL (“A Programming Language” source of the very name “scan”, an array based language that was ahead of its time)
2. MPI_scan
3. MATLAB command: y=cumsum(x)
4. MATLAB matmul: y=tril(ones(n))*x

Parallel Prefix Recursive View

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

1 2 3 4 5 6 7 8

3 7 11 15 — Pairwise sums

3 10 21 36 — Recursive prefix

1 3 6 10 15 21 28 36 — Update “odds”

- Any associative operator

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

```
function prefix!(x, *)
    n=length(x)
    if n<=1
        return()
    end

    for i=2:2:n    # even: pairwise sums
        x[i] *= x[i-1]
    end

    prefix!(view(x,2:2:n),*) # recursive prefix (in place!)

    for i = 3:2:n  # odd: x[i-1] is a cumsum,x[i] is raw data
        x[i] *= x[i-1]
    end
end
```

What does this reveal? What does this hide?

Operation Count

- Notice
- # adds = 2n
- # required = n

- Parallelism at the cost of more work!

Any Associative Operation works

Associative:

$(a+b)+c = a+(b+c)$

Sum (+)

Product (*)

Max

Min

Input: Reals

All (=and)

Any (= or)

Input: Bits (Boolean)

MatMul

Inputs: Matrices

Fibonacci via Matrix Multiply Prefix

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

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

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

Carry-Look Ahead Addition (Babbage 1800's)

Example	
1 0 1 1 1	Carry
1 0 1 1 1	First Int
1 0 1 0 1	Second Int
1 0 1 1 0 0	Sum

Goal: Add Two n-bit Integers

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	b_3	b_2	b_1	b_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

(addition mod 2)

$c_{-1} = 0$

for $i = 0 : n-1$

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

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

end

$s_n = c_{n-1}$

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

$c_{-1} = 0$

(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

$s_n = c_{n-1}$

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

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

$c_{-1} = 0$ (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!

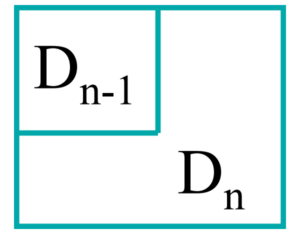
Compute c_i by prefix, then

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

Tridiagonal Factor

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

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



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

Compute D_n by matmul_prefix

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

$$T = \begin{bmatrix} 1 & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} d_1 & b_1 & l_1 \; 1 \\ & & \\ & & \end{bmatrix}$$

$d_2 \quad b_2$

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

$$l_n = c_n / d_n$$

3 embarassing
Parallels + prefix

The “Myth” of log n

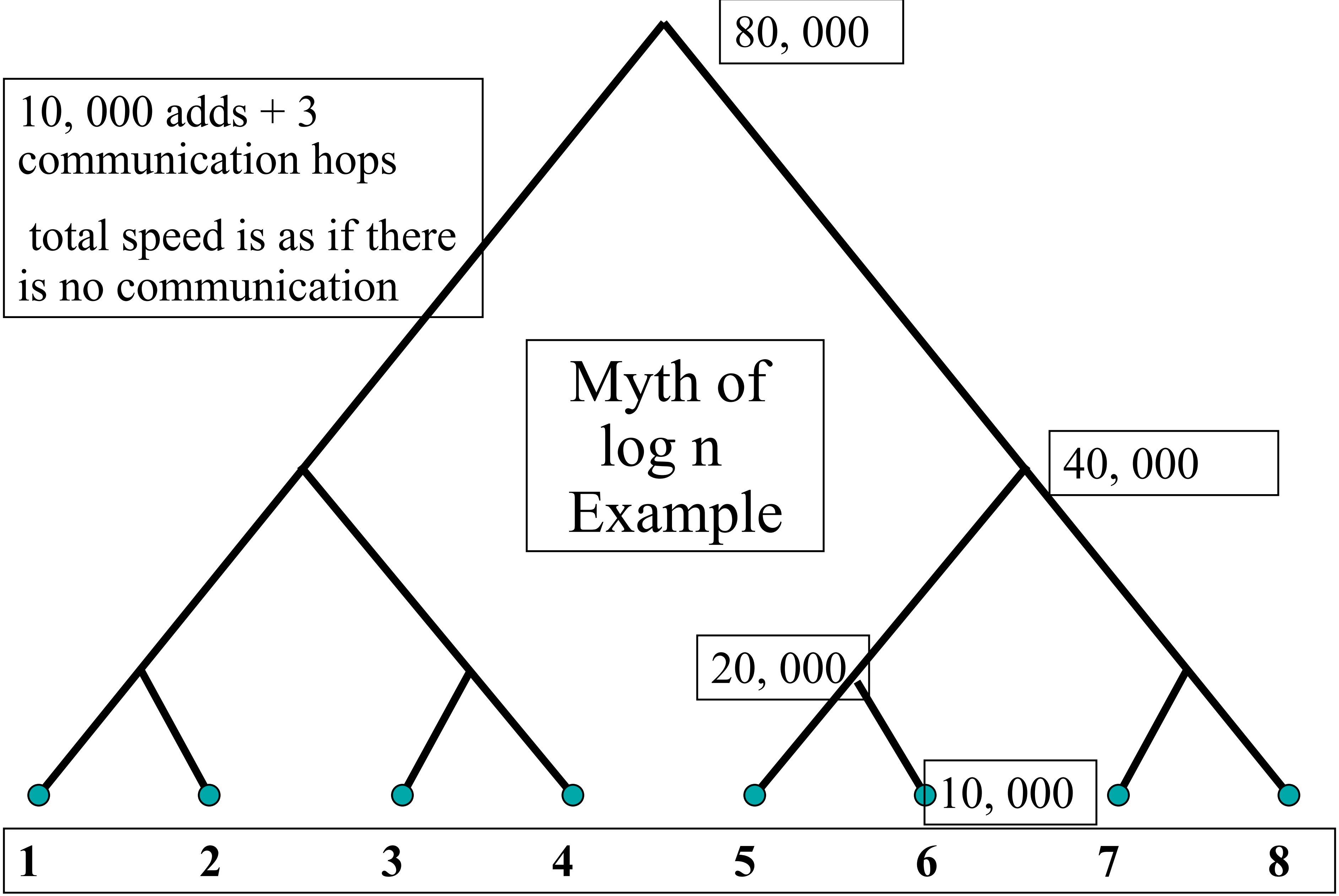
The $\log_2 n$ parallel steps is not the main reason for the usefulness of parallel prefix.

Say n = 1000p (1000 summands per processor)

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

fast & embarassingly parallel

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



$\log_2 n$ = number of steps to add n 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

$+_2$	(y, T)	(y, F)
(x, T)	(x+y, T)	(y, F)
(x, F)	(y, T)	(x+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: $\mathbf{x} + \mathbf{y} = \mathbf{x}$

(is associative)

Segmented

1	**2**	|	**3**	**4**	**5**	|	**6**	|	**7**	|	**8**
T	**T**	|	**F**	**F**	**F**	|	**T**	|	**F**	|	**T**
1	**1**	|	**3**	**3**	**3**	|	**6**	|	**7**	|	**8**

High Performance Fortran

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

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

$$\text{SUM\_SUFFIX(A)} \qquad \begin{matrix} 1 & 3 & 6 & 10 & 15 \end{matrix}$$

$$\text{SUM\_PREFIX(A, DIM = 2)} = \begin{matrix} 6 & 13 & 21 & 30 & 40 \\ 11 & 23 & 36 & & \end{matrix}$$

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

More HPF

Segmented

	1	2	3	4	5
A =	6	7	8	9	10
	11	12	13	14	15

T T | F | T T | F F

	T	T	F	F	F
S =	F	T	T	F	F
	T	T	T	T	T

Sum_Prefix (A, SEGMENTS = S)

1	13	3
6	20	
11	32	

Example of Exclusive

A = 1 2 3 4 5

Sum_Prefix(A) 1 3 6 10 15

Sum_Prefix(A, EXCLUSIVE = TRUE)

0 1 3 6 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]

1 2 3 4 5 6 7 8

Pairwise sums

3 7 11 15

Recursive prefix

3 10 21 36

Update "evens"

1 3 6 10 15 21 28 36

- Any operator
- AKA: +\ (APL), cumsum(Matlab), MPI_SCAN,

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

Variations on Prefix

1 2 3 4 5 6 7 8

3 7 11 15

0 3 10 21

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]

```
1  2  3  4   5  6    7  8
  3     7      11     15
  0     3      10     21
0  1  3  6 10 15 21 28
```

1)Pairwise Sums
2)Recursive Prefix
3)Update “odds”

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]

1 2 3 4 5 6 7 8 ← 1)Pairwise Sums
3 7 11 15 ← 2)Recursive Prefix
0 3 10 21 ← 3)Update "evens"
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]

1 2 3 4 5 6 7 8 ← 1)Pairwise Sums

3 7 11 15 ← 2)Recursive Reduce

36 36 36 36 ← 3)Update "odds"

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]

```
1  2   3  4   5  6   7  8     ← 1)Pairwise Sums
  3      7      11     15     ← 2)Recursive Prefix
  0      3      10     21     ← 3)Update “evens”
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	**Reduce**	Exc Reduce	**Multipole**

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...

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

Not Parallel Prefix but PRAM

- Only concerned with minimizing parallel time (not communication)
- Arbitrary number of processors
- One element per processor

Csanky's (1977) Matrix Inversion

Lemma 1: $(\triangle^{-1})$ 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*

$$p(x) = \det(xI - A) = x^n + c_1 x^{n-1} + \ldots + c_n$$

$\pm$

$(c_n = \det A)$

$$0 = p(A) = A^n + c_1 A^{n-1} + \ldots + c_n I$$

$$A^{-1} = (A^{n-1} + c_1 A^{n-2} + \ldots + c_{n-1})(-1/c_n)$$

Lemma 3: *Leverier's Lemma*

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

Csanky 1) Parallel Prefix powers of A

2) s_k by directly adding diagonals

3) c_i from lemmas 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!

- Parallel steps are not the whole story
- Nobody puts one element per processor