Module 11
Arithmetic Operations

S. Lakshmivarahan
School of Computer Science
University of Oklahoma
USA-73019
Varahan@ou.edu

Arithmetic Operations

- Our goal is to examine faster algorithms to add and multiply two n-bit integers
- To this end, we start by examining the grade school method for add and multiply

Binary Addition
Grade School Method

- Recall the basic binary addition table

X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Binary Addition
Grade School Method

- Let $n=2^k$ and let a and b be two n-bit integers given by

$$a = a_{n-1}a_{n-2} \dots a_1 a_0$$

$$b = b_{n-1}b_{n-2} \dots b_1 b_0$$

- Let $s = s_n s_{n-1} \dots s_1 s_0$ be the (n+1) bit integer that represents the sum s = a+b

Binary Addition
Grade School Method

- Then,

	c_n	c_{n-1}	c_{n-2}		c_2	c_1	c_0
a=		a_{n-1}	a_{n-2}	…	a_2	a_1	a_0
b=		b_{n-1}	b_{n-2}	…	b_2	b_1	b_0
s=	s_n	s_{n-1}	s_{n-2}	…	s_2	s_1	s_0

- Where c_0=0 and s_n=c_n. The rest of the s_i, $0 \le i \le n-1$, are computed using the binary addition table.

Binary Addition
Grade School Method

- The total number of bit additions needed to compute s is 2n = O(n) and is known as the Boolean complexity.
- Note: In the unit cost model, the unit was taken as the time required to perform one basic operation- +,-,*,%, and ≥.
- However, in here our basic unit is the time taken to perform a one bit binary- AND, OR, NOT operation. Thus, the unit in this case (boolean complexity) is much smaller

Binary Addition
Grade School Method

- <u>Example</u>: Let n=4, a=11, b=15. Then s=26

	1	1	1	1	0
a=		1	0	1	1
b=		1	1	1	1
s=	1	1	0	1	0

Binary Multiply
Grade School Method

- The basic binary multiplication table

X	Y	Product
0	0	0
0	1	0
1	0	0
1	1	1

Binary Multiply
Grade School Method

- Let $n=2^k$ and let a and b be two n bit integers
- <u>Example:</u> n=4, a=11, b=15, ab=165

			a=		1	0	1	1	
			b=		1	1	1	1	
					1	0	1	1	← n-bit multiply
				1	0	1	1		← n-bit multiply
			1	0	1	1			← n-bit multiply
		1	0	1	1				← n-bit multiply
ab=	1	0	1	0	0	1	0	1	← (n-1) addition of the above n n-bit integers

Binary Multiply
Grade School Method

- Thus, it takes n^2 bit multiplications and (n-1) bit additions, each of which takes O(n) bit additions
- Thus, it takes $O(n^2)$ bit operations to multiply two n-bit integers

A faster method for multiplication of integers

- Let $n=2^k$ and let $a = a_{n-1}a_{n-2} \ldots a_2a_1a_0$ and $b=b_{n-1}b_{n-2}\ldots b_1b_0$ be the n bit integers to be multiplied.
- Consider the polynomials A(x) and B(x) in x of degree n-1, given by

$$A(x) = \sum_{i=0}^{n-1} a_i x^i, \quad a_i \in \{0,1\} \quad \rightarrow a = A(2)$$

$$B(x) = \sum_{i=0}^{n-1} b_i x^i, \quad b_i \in \{0,1\} \quad \rightarrow b = B(2)$$

A faster method for multiplication of integers

- Thus, there is a close correspondence between the integer arithmetic and polynomial arithmetic
- We now describe a divide and conquer based algorithm to multiply two polynomials
- This algorithm is due to Karatsuba and Ofman

A faster method for multiplication of integers

- First, divide the polynomials A(x) into two equal parts:

$$A(x) = A_L(x) + x^{n/2} A_H(x)$$

where

$$A_L(x) = \sum_{i=0}^{n/2-1} a_i x^i$$

$$A_H(x) = \sum_{i=0}^{n/2-1} a_i x^i$$

each of which is a polynomial of degree $(\frac{n}{2} - 1)$

- Similarly,

$$\mathrm{B}(x) = B_L(x) + x^{n/2} B_H(x)$$

A faster method for multiplication of integers

- As an example, let n = 4. Then

$$A(x) = a_0 + a_2x + a_2x^2 + a_3x^3$$

$$= (a_0+a_2x) + x^2(a_2+a_3x)$$

$$= A_L(x) + x^2A_H(x)$$

where

$$A_L(x) = a_0 + a_1x$$

$$A_H(x) = a_2 + a_3x$$

A faster method for multiplication of integers

- Let c = ab and let C(x) be the polynomial corresponding to the product c
- Thus, $C(x) = A(x) * B(x)$
- Then, substituting for A(x) and B(x):

$$C(x) = \left(A_L(x) + x^{n/2}A_H(x)\right) * (B_L(x) + x^{n/2}B_H(x)) \rightarrow ①$$

where $A_L = A_L(x)$ and so on

A faster method for multiplication of integers

- If we multiply the right hand side, then we get

$$C(x) = A_L B_L + x^{\frac{n}{2}}(A_L B_H + A_H B_H) + x^n A_H B_H$$

- Since x=2, multiplying by $2^{\frac{n}{2}}$ and 2^n is equivalent to shifting the binary point to the right by $\frac{n}{2}$ and *n* locations respectively. Hence, these multiplications do not take much time.

A faster method for multiplication of integers

- So, when we compute the r.h.s. of ①, we are left with 4 multiplications of polynomials of degree $\frac{n}{2} - 1$.

  $$A_L B_L, \;\; A_L B_H, \;\; A_H B_L, \;\; and \; A_H B_H,$$

  and three polynomial additions of polynomials of degree at most 2n-2.

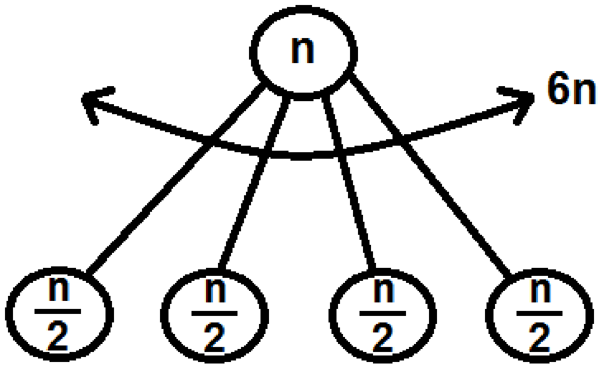
A faster method for multiplication of integers

- Recall, each polynomial addition takes linear time in the degree of the polynomial.
- Hence, the three additions together takes no more than

  $3x(2n-2) \leq 6n$ operations

A faster method for multiplication of integers

- If T(n) is the time to multiply two polynomials of degree (n-1) each, then we get the following recursive tree that describes the above algorithm.



- $\therefore T(n) = 4T\left(\frac{n}{2}\right) + 6n$, with $n=2^k$ and T(1) = 1 → 2

A faster method for multiplication of integers

- Solving ②, it follows that

$$T(n) = O\left(n^{\log_2 4}\right) = O(n^2) \rightarrow \text{ ③}$$

- That is, we divided but did not conquer it since ③ is also the time for the grade school method
- The conquering part is one that was introduced by Karatsuba and Ofman

Karatsuba-Ofman Algorithm

- To compute the product C(x), first compute three intermediate polynomials
- Let

$$\left.\begin{aligned} C_L &= A_L B_L \\ C_H &= A_H B_H \\ C_M &= (A_L + A_H)(B_L + B_H) \end{aligned}\right\} (4)$$

involving a total of 3 multiplications of polynomials of degree $(\frac{n}{2} - 1)$, (instead of the 4 such multiplications previously) and two additions.

Karatsuba-Ofman Algorithm

- This is similar in idea to the Strassen's method, which trades the expensive multiplication for the cheap additions.
- Then, assemble the product as

$$C(x) = C_L + [C_M - C_L - C_H]x^{\frac{n}{2}} + C_H x^n \ \rightarrow \ ⑤$$

- In ⑤, there are only 4 additions and there are no multiplications except those by x^n and $x^{\frac{n}{2}}$ which are very inexpensive.

Karatsuba-Ofman Algorithm

- Combining,

$$T(n) = 3T\left(\frac{n}{2}\right) + \alpha n \quad \rightarrow ⑥$$

where αn is the time to perform 6 additions of polynomials of degree less than or equal to 2n-2. Hence, the constant in ⑥ is $\alpha \leq 12$.

Karatsuba-Ofman Algorithm

- Solving ⑥, it follows that

$$T(n) = O(n^{\log_2 3})$$

$$= O(n^{1.58})$$

which is much faster than the grade school method

Homework

1. Multiplying the r.h.s. of ⑤, verify that you get the same answer as in ①.
2. Solve the recurrences in ② and in ⑤ exactly and verify the answers

Faster Multiplication

- The question now comes: can we multiply integers ever faster than O($n^{1.58}$).
- The answer is indeed Yes! The fastest known algorithm takes only $O(\log n)$ time and is due to Schonhage and Strassen (1971)

Faster Multiplication

- A detailed analysis of this parallel algorithm is given in Chapter 3 of the following book:
- S. Lakshmivarahan and S.H. Dhall (1990). Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems, McGraw Hill, New York

A Fast Parallel Algorithm for Integer Addition

- Let $s = a + b$ where

  $a = a_{n-1}\ a_{n-2} \ldots a_2\ a_1\ a_0$

  $b = b_{n-1}\ b_{n-2} \ldots b_2\ b_1\ b_0$

- And

  $s = s_n\ s_{n-1} \ldots s_2\ s_1\ s_0$

A Fast Parallel Algorithm for Integer Addition

- Recall that

  $s_i = a_i \oplus b_i \oplus c_i$, $0 \le i \le n-1$

  $s_n = c_n$, $c_0 = 0$ and for $1 \le i \le n$ (1)

  $c_i = (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1})$

  where

  $\wedge$ is the Boolean AND

  $\vee$ is the Boolean OR

  $\oplus$ is the exclusive OR

$\wedge$	**0**	**1**
0	0	0
1	0	1

$\vee$	**0**	**1**
0	0	1
1	1	1

$\oplus$	**0**	**1**
0	0	1
1	1	0

A Fast Parallel Algorithm for Integer Addition

- The expression for c_i can be rewritten as

$$c_i = g_i \wedge (p_i \wedge c_{i-1}) \quad \rightarrow \quad ②$$

where

$$g_i = a_i \wedge b_i, \text{ the generate bit}$$

$$p_i = a_i \vee b_i, \text{ the propogate bit}$$

- That is, c_i in ② is given by the first order linear Boolean recurrence

A Fast Parallel Algorithm for Integer Addition

- We can write ② as

$$c_i = f_i(c_{i-1}) \quad \rightarrow \text{③}$$

where the Boolean function depends on two Boolean parameters g_i and p_i, $1 \leq i \leq n$

A Fast Parallel Algorithm for Integer Addition

- Iterating ③, we readily obtain

$$c_1 = f_1(c_0)$$

$$c_2 = f_2(c_1) = f_2(f_1(c_0)) = f_1 \circ f_2(c_0)$$

$$c_3 = f_3(c_2) = f_1 \circ f_2 \circ f_3(c_0) \qquad ④$$

$$\dots$$

$$c_n = f_1 \circ f_2 \circ f_3 \circ \ \dots \circ f_n(c_0)$$

where $f_1 \circ f_2\ (c_0) = f_2(f_1(c_0))$

Property of the Composition

- $f_1 \text{ o } f_2\ (c_0)$ is the binary operation of the composition of f_1 and f_2 (from left to write)
- What is $f_1 \text{ o } f_2\ (c_0)$?
- Recall, $f_i \text{ o } f_j\ (x) = f_j(f_i(x))$

$$= f_j(g_i \vee (p_i \wedge x))$$

$$= g_j \vee \left(p_j \wedge [g_i \vee (p_i \wedge x)]\right)$$

$$= \left[g_j \vee \left(p_j \wedge g_i\right)\right] \vee [(p_j \wedge p_i) \wedge x]$$

A Fast Parallel Algorithm for Integer Addition

- Thus, $f_i \circ f_j\ (x) = \ G\ \vee(P \wedge x)\ \ \rightarrow\ ⑤$
- Where $G = g_i \vee \left(p_i \wedge g_j\right)$ is the new generate bit and $P = (p_i \wedge p_j)$ is the new propagate bit
- Thus, the functions f_i and f_j are composed, and the resulting function is of the same type as f_i and f_j
- This is called the reproducibility property under composition.

Homework

3. Verify that

$$f_1 \text{ o } f_2 \text{o } f_3(x) = (f_1 \text{ o } f_2)\text{o } f_3(x)$$
$$= f_1 \text{ o } (f_2 \text{o } f_3)(x)$$

4. Thus, the composition of functions is an associative, binary operation

Parallel Algorithm for Integer Addition

- Our goal is to compute the parameters of the series of n functions in ④ repeated here for convenience:

$$
\begin{aligned}
&f_1 \\
&f_1 \circ f_2 \\
&f_1 \circ f_2 \circ f_3 \qquad\qquad (6) \\
&\ldots \\
&f_1 \circ f_2 \circ f_3 \circ \ \ldots \circ f_n
\end{aligned}
$$

Parallel Algorithm for Integer Addition

- Since each function in ⑥ is of the type ⑤, we can evaluate them all at c_0 to get all the carry bits c_1 to c_n.
- Computing these composite functions where the composition of functions is an associative binary operation is known as the prefix problem.

Parallel Algorithm for Integer Addition

- There are circuit based efficient parallel algorithms to compute these composite functions in $O(\log n)$ time.
- To enable this, we now digress to describe the prefix problem and some of the ideas to compute them in parallel.

The Prefix Problem

- Let $\{x_i, x_2, \dots ,x_n\}$ be a set of n numbers, and let + denote an associative binary operation- such as addition or multiplication of numbers.
- Our goal is to compute ⑥.
- This is called the prefix problem, all partial sum problem, etc.

The Prefix Problem

- To compute these serially, let $(y_1, y_2, \ldots, y_n)$ be an array. Then,

$$y_1 = x_1$$

DO $i = 2\ to\ n$

$$y_i = y_{i-1} + x$$

END DO

is the serial algorithm to compute the n prefixes in ⑥

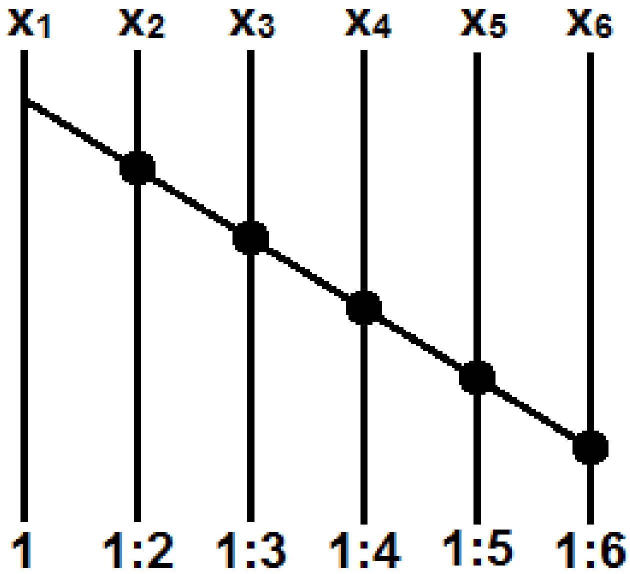
The Prefix Problem

- Clearly, the serial time $T_s(n) = n-1$.
- The question now is: can we compute ⑥ in parallel in a time faster than $O(n)$?
- To answer this, we first represent the serial algorithm as an n-input, n-output layered circuit as shown below.

The Prefix Problem

1. <u>Serial Circuit</u>

- Let n=6:



● denotes the operation of addition of two inputs into it

$i{:}j = x_i + x_{i+1} + \ldots + x_j$ for $i \leq j$

The Prefix Problem: Serial Circuit

- There are two parameters that describe the performance of this circuit- size and depth
- The size (n) = total number of operation nodes.
- For the serial circuit with n=6, size(6) = 5. In general, the size of the serial circuit denotes the total number of operations, and s(n) = n-1

The Prefix Problem: Serial Circuit

- The depth, d(n) of the serial circuit is the length of the longest path- it denotes the time.
- For the illustration with n=6, d(6) = 5. In general, d(n) = n-1

The Prefix Problem: Serial Circuit

- Another quantity of interest is called the size+depth
- In general for the serial circuits,

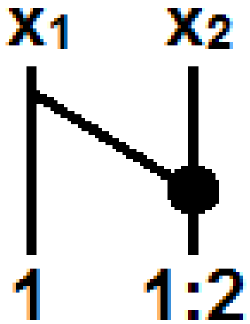
$$s(n) + d(n) = 2n-2$$

- We move to describing parallel circuits to compute the prefixes with depth of order of log n.

The Prefix Problem: DC Circuit

1. <u>Divide and Conquer Circuit</u>: (DC-Circuit)

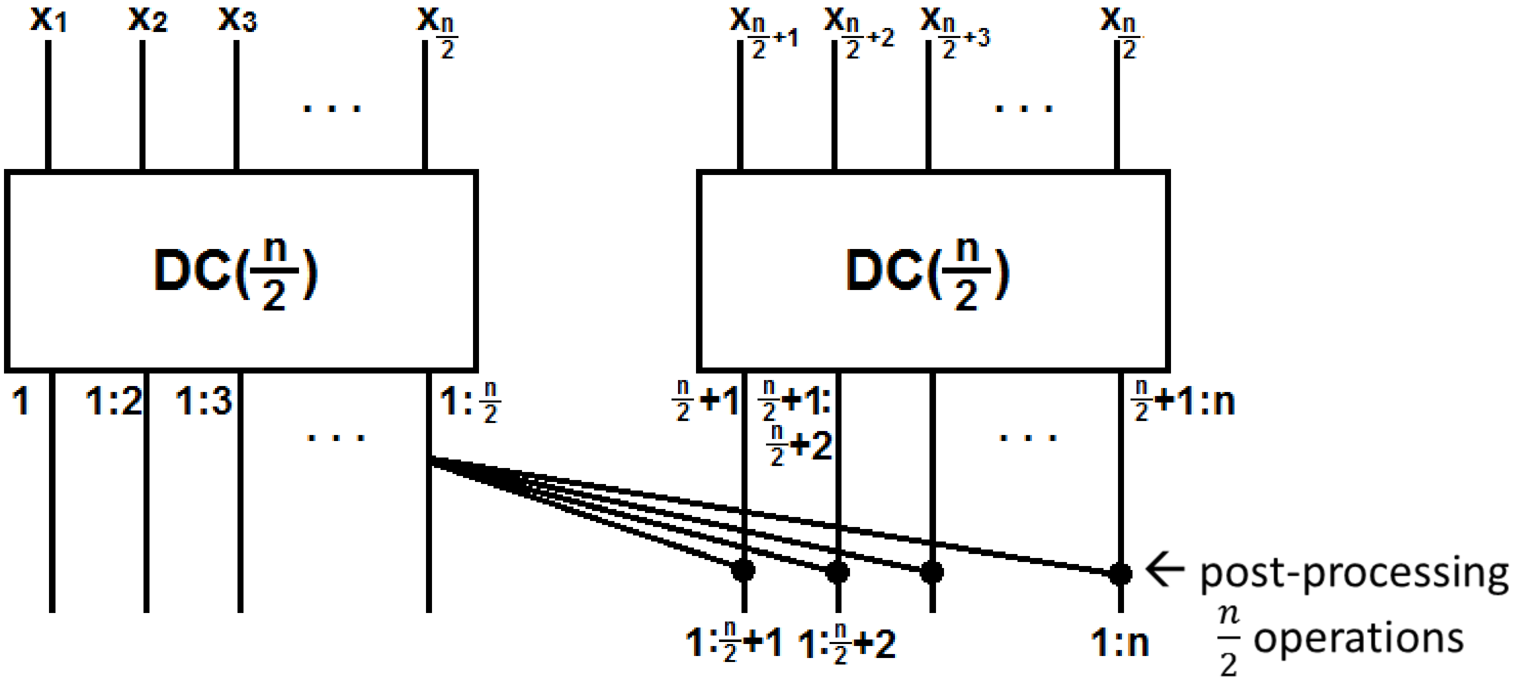
- Let $n=2^k$. For n=2, the DC circuit DC(2) is given by



with s(n)=1, d(n)=1

The Prefix Problem: DC Circuit

- For n>2,



The Prefix Problem: DC Circuit

- The structure is quite explanatory. The last output of the first DC($\frac{n}{2}$) box is combined with each out of the second DC($\frac{n}{2}$) box to get the overall correct results.

The Prefix Problem: DC Circuit

- Clearly,

$$\left.\begin{aligned} s(n) &= 2s\left(\frac{n}{2}\right) + \frac{n}{2},\ s(2) = 1 \\ d(n) &= d\left(\frac{n}{2}\right) + 1,\ d(2) = 1 \end{aligned}\right\} ⑧$$

- Solving, we get

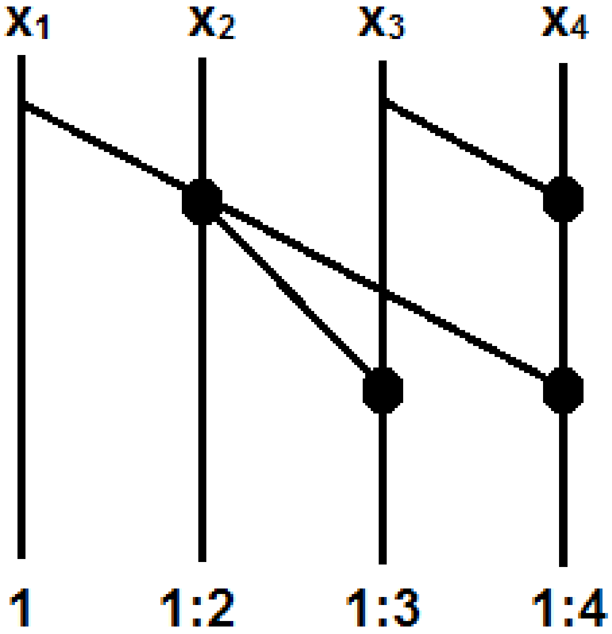
$$\left.\begin{aligned} s(n) &= \frac{n}{2}\log n \\ d(n) &= \log n \end{aligned}\right\} ⑨$$

The Prefix Problem

- **Homework**: Solve ⑧ and verify the answers in ⑨.
- While we have achieved our goal of $\log n$, it is achieved at the cost of performing a whole lot of operations of the order $n \log n$.

DC Circuit Examples

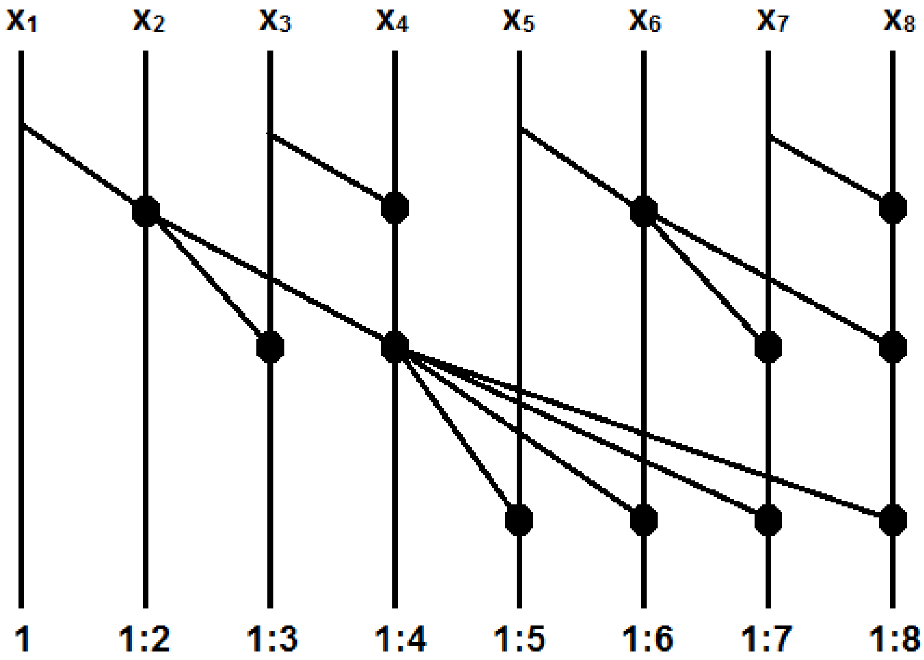
- DC(4):



s(4) = 4
d(n) = 2

DC Circuit Examples

- DC(8):



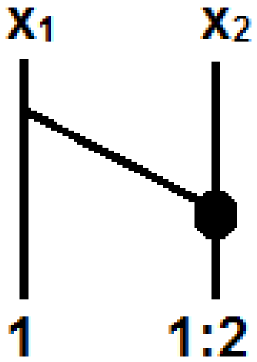
s(8) = 12
d(8) = 3

The Prefix Problem

- **<u>Homework</u>**: Draw DC(16). Compute s(16), d(16).
- We now provide a third class of circuits that, while it keeps the depth to be O($\log n$), it reduced the size to O(n) instead of O($n \log n$).

Prefix Problem: Brent-Kung Circuit

- Brent-kung prefix circuit: BK(N)
- Let $n=2^k$
- BK(2):

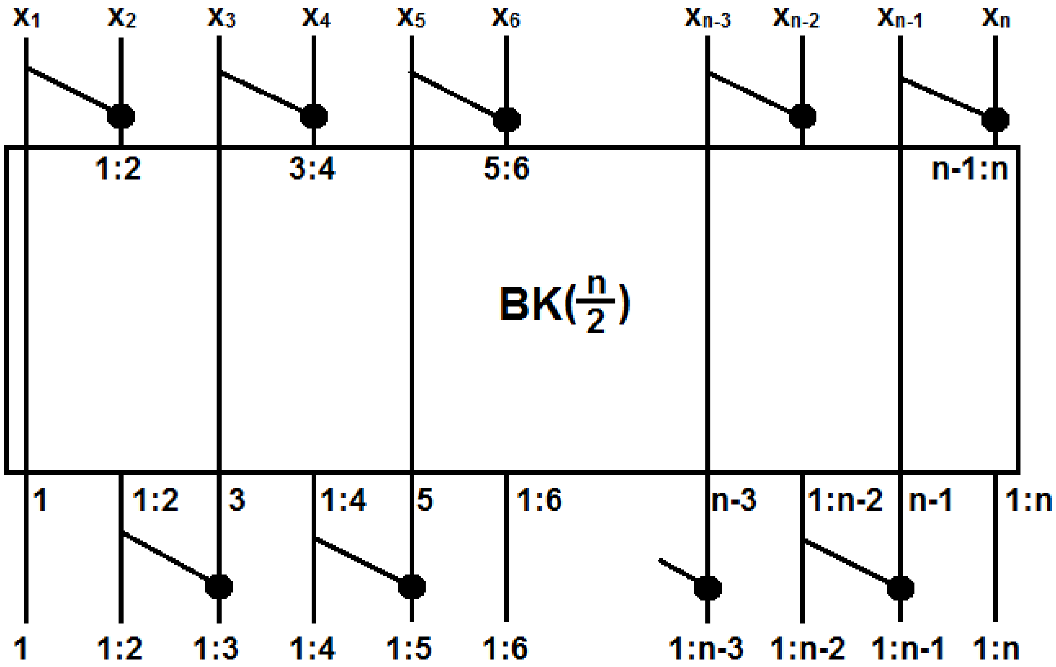


$s(n) = 1$
$d(n) = 1$

Prefix Problem: Brent-Kung Circuit

- BK(n)

Lines over the box don't enter the box



← Preprocessing $\frac{n}{2}$ operations

← Post processing $\frac{n}{2} - 1$ operations

Prefix Problem: Brent-Kung Circuit

- Now

$$\left.\begin{aligned} s(n) &= s\left(\frac{n}{2}\right) + (n-1) \\ d(n) &= d\left(\frac{n}{2}\right) + 2 \end{aligned}\right\} \quad (10)$$

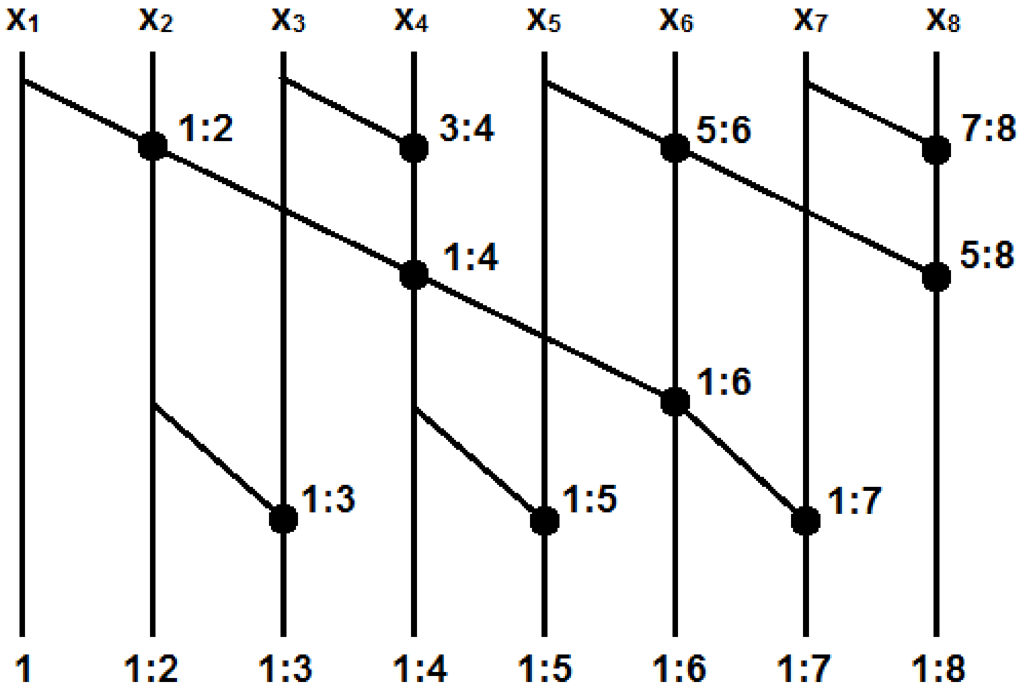
- Hence

$$\left.\begin{aligned} s(n) &= 2n - \log n - 2 \\ d(n) &= 2\log n - 2 \end{aligned}\right\} \quad (11)$$

Prefix Problem: Brent-Kung Circuit

- Homework: Solve ⑩
- Notice that while the depth is still O(log *n*), the size has reduced considerably to linear in n.

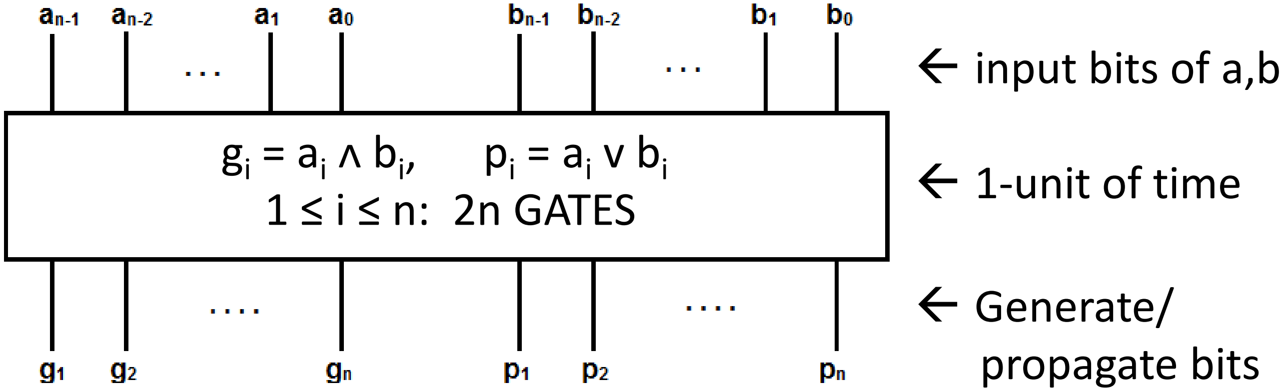
Example of BK(8)



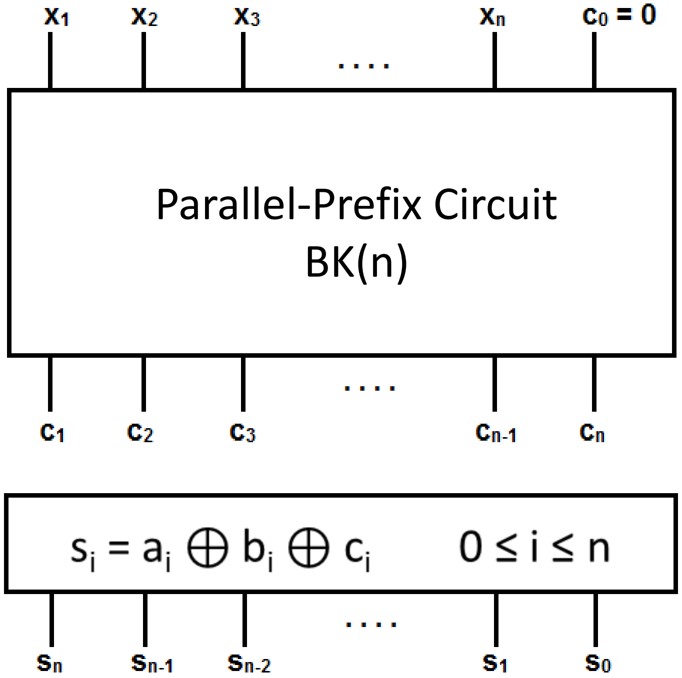
$s(n) = 11$
$d(n) = 4$

Back to Parallel Adder Circuit

- We can now use BK(n) to compute the prefixes in in $O(\log n)$ time.
- We now provide a layout of the parallel adder.



Parallel Adder Circuit



$x_i = (g_i, p_i)$
$1 \le i \le n$

← n carry bits

Parallel Adder Circuit

- Hence, overall depth is of the form $\propto + \beta \log n$ for some constants $\propto$ and β.
- This is the fastest known algorithm to add two n-bit integers.
- Reference: S. Lakshmivarahan and S. K. Dhall (1994). Parallel Computation using the Prefix Problem, Oxford University Press.