Integer Arithmetic

NEW! Now with Mesh Trees!!!

Βούλγαρης Παναγιώτης
Μωλ Πέτρος

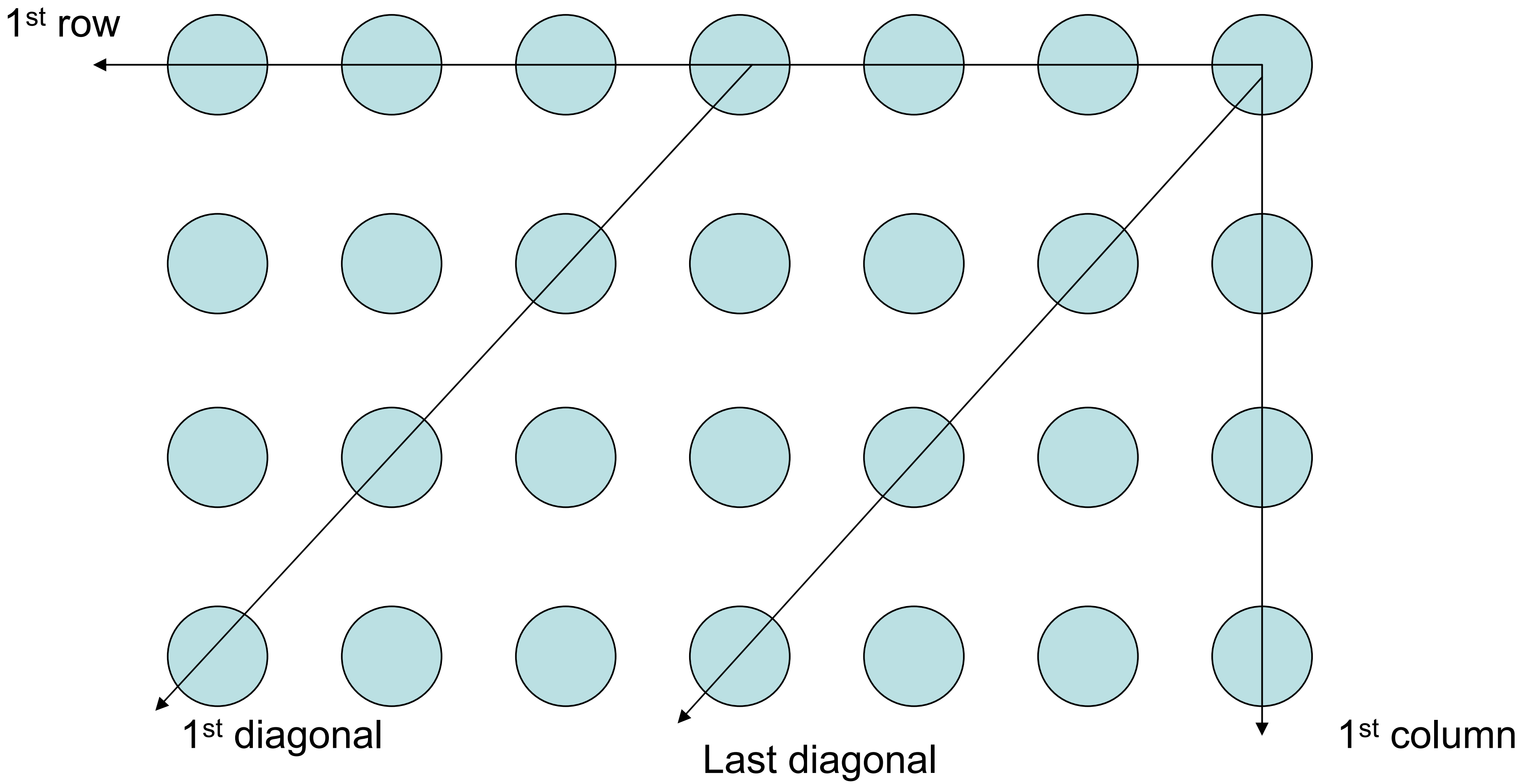
Complexity

	Time	Procs	Work	Efficiency
Multiply	O(logN)	O(N^2)	O((N^2) logN)	O(N)
With Pipline	The same	The Same	O(N^2)	O(N/(logN))

- Best serial O(NlogN)
- Really Efficient algorithms later with Furrier Transforms + Hypercube Networks
- All the algorithms have similar inefficiencies (or worst)

Multiply 2 N-bit numbers

- Mesh of trees N*(2N-1) with row, column and diagonal trees.



Multiply - The Steps

(Let a=a_1,…,a_N b=b_1,…,b_N)

- Enter a_i in root of I row tree

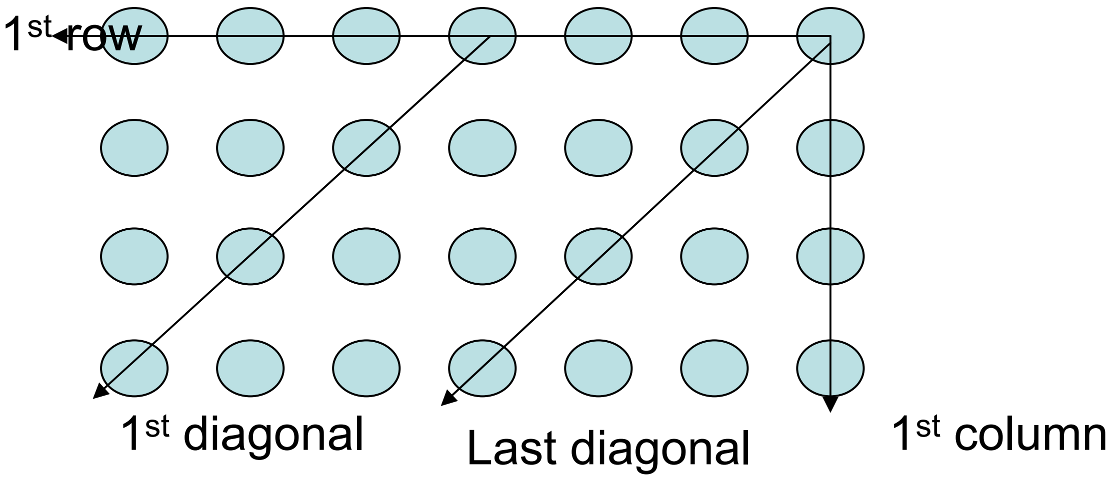
  Enter b_i in root of N-i+1 diagonal tree

  Propagate downwards leaves get a_i*b_j
- Sum the bits of each column in the column root (starting with less significant bit).

  2N-k col has s_k=s_{k,log(N+1)}….s_{k,1}
- Send all s_k to leaf in the first row of the column, and sum them with carry-save addition, the last two with carry lookahead.

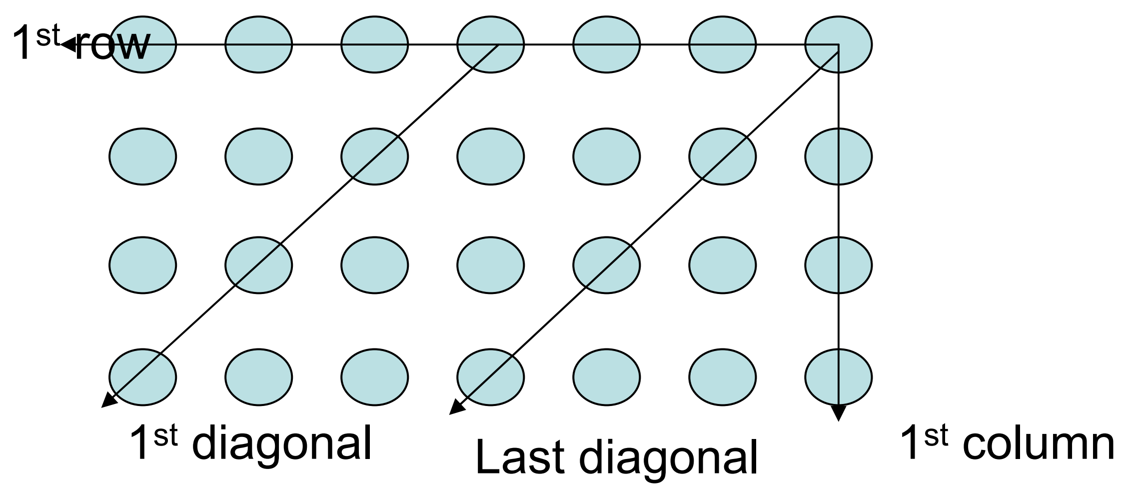
Step 1

- a=a_1,…a_N b=b_1,…,b_n
- Enter a_i in root of i row tree
  Enter b_i in root of N-i+1 diagonal tree
  Propagate downwards leaves get a_i*b_j
- We send them down the trees and multiply them when they intersect in the leaves.
- logN+1 steps to reach the leaves



Step 2

- Sum the bits of each column in the column root.
- Let 2N-k column has s_k=s_{k,logN+1}…s_{k,1}
- Let w_l=s_{2N-1,l…s_{1,l}
- w_l are the bits of s’ s as seen by rows from the least significant w_1 to the most significant w_{2N-1}



S's and W's

	s_{2N-1}	.	.	.	s_1
w_1->	s_{2N-1,1}	.	.	.	s_{1,1}
w_2->	s_{2N-1,2}	.	.	.	s_{1,2}
.	.	.	.	.	.
.	.	.	.	.	.
w_{2N-1}->	s_{2N-1,logN+1}	.	.	.	s_{1,logN+1}

Step 3

- Send the bits of s in the first row of the column. (logN)
- So we get w_i ‘s. We sum them and shift them right for each new we get with Carry-Save addition. (LogN)
- We sum the last two with Carry-lookahead. (2LogN)

Example

We get w_1. We shift it and get the last bit which is the last bit of the result. Now we have (w_1)/2 and we get w_2 we add the with Carry-Save addition and shift them getting the second bit of the result. And so on. (When we get the last of w's we have to make a "real" addition.)

Division in $O(\log^2 N)$

- Simple Newton Iteration
- $x_{i+1} = x_i + f(x_i)/f'(x_i)$
- $x=1/y \Rightarrow f(x)= 1-yx \quad f'(x)=-y$
- $x_{i+1}=x_i +1/y(1-yx_i)$
- With $1/y=x_i$
- $x_{i+1}=2x_i -y(x_i)^2$
- $O(\log N)$ operations in each step
- $O(\log N)$ steps for N bits $O(\log^2 N)$ complexity

Division in O(logN)

- Using Chinese Remaindering Theorem
- Faster asymptotically but to much look aheads, not usable mainly theoretical value.

Division Idea

- $y=1-\varepsilon \quad 0<\varepsilon<1/2$
- $1/y=1/(1-\varepsilon)=1+\varepsilon+\varepsilon^2+\varepsilon^3+\ldots.$
- $X_i=1+\varepsilon+\varepsilon^2+\ldots\varepsilon^i.$
- $|1/y-X_i|=\varepsilon^{i+1}+\ldots =< 1/2^{i+1}+\ldots=<2^{-i}$
- If we know N+logN bits of ε^i we sum them in logN (and find X_n). We can parallely compute ε^i so it suffices to compute ε^i in O(logN) steps and we get O(logN) complexity.

Chinese Remainder Theorem

- Let primes $p_1,p_2,\ldots p_s$
- Let $o=<X<P$
- The residue vector for every X with p_i's is unique and from it X can be reconstructed
- $X=\mathrm{Sum}\{i=1 \text{ to } s\} (\beta_i x_i \bmod P)$
- $\beta_i=(P/p_i)\, a_i \quad \alpha_i=(P/p_i)^{-1} \bmod p_i$

Prime Number Theorem

- Number of primes less than N is $\Theta(N/\log N)$

Idea

- Z^N is at most 2^{N^2} and it has at most N^2 bits
- $P=p_1,\ldots p_{N^2}$ the multiplication of the first N^2 primes
- Instead of Z^N we compute $Z^N \bmod p_i$ for every p_i which has $O(\log N)$ bits and we recontrsuct $Z^N \bmod P$ and then the result.
- Because p_i are $O(\log N)$ bits we can construct lookup tables for every operation we need to do. And each operation will be a lookup in a poly $(\log N)$ table.
- We see that although it is $\log(N)$ we must have precomputed to much information.