Parallel Algorithm and Complexity - Samir Datta

Scribed: Soham Chatterjee

sohamchatterjee999@gmail.com
Website: sohamch08.github.io

2023

Contents

1 Addition of Two Numbers in Binary

Problem: ADD_{2n}
Input: Two n bit numbers $a = a_{n-1} \cdots a_1 a_0$ and $b = b_{n-1} \cdots b_1 b_0$
Output: $s = s_n \cdots s_1 s_0$ where $s \stackrel{\text{def}}{=} a + b$

1.1 Sequential (Ripple Carry)

For sum of any position i the two bits a_i, b_i and the carry generated by the previous position c_{i-1} is added. For the initial position we can set $c_0 = 0$. If we add two bits at most 2 bits is created. The right bit is called the sum bit and the left bit is the carry bit. $a_i + b_i + c_{i-1} = c_i s_i$. Then

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

Time Complexity: This algorithm takes $O(n)$ time complexity

1.2 Parallel (Carry Look Ahead Adder)

There is a carry that ripples into position i if and only if there is some position $j < i$ to the right where this carry is generated, and all positions in between propagate this carry. A carry is generated at position i if and only if both input bits a_i and b_i are on, and a carry is eliminated at position i, if and only if both input bits a_i and b_i are off. This leads to the following definitions:

For $0 \leq i < n$, let

$g_i = a_i \wedge b_i$	position i generates a carry
$p_i = a_i \vee b_i$	position i propagates a carry that ripples int o it

So we can set for $1 \leq i \leq n$

$$c_i = \bigvee_{j=0}^{i-1} \left(g_j \wedge \bigwedge_{k=j+1}^{i-1} \right)$$

Now the sumbits are calculated as before $s_i = a_i \oplus b_i \oplus c_{i-1}$ for $0 \leq i \leq n-1$ and $s_n = c_n$
Time Complexity: This algorithm takes $O(1)$ time complexity

Definition 1.1 (AC^0)**.** *The class of circuits consists of the gates $(\vee_n, \wedge_n, \neg_1)$ (The subscript n or 1 denotes the fanin) of polynomial size and depth $O(1) = O(\log^0 n)$*

Theorem 1.1. $ADD_{2n} = IterADD_{2,n} \in AC^0$

2 Iterated Addition

Problem: $IterADD_{k,m}$
Input: k many m-bit numbers $a_1, \ldots, a_k$
Output: The sum of the input numbers

Definition 2.1 (Length Respecting)**.** *Let $f : \{0,1\}^* \to \{0,1\}^*$. f is length respecting if for all $x, y \in \{0,1\}^*$ $|f(x)| = |f(y)|$*

Definition 2.2 (Constant Depth Reduction)**.** *let $f, g : \{0,1\}^* \to \{0,1\}^*$ be length respecting. Then f is constant depth reducible to g or $f \leq_{cd} g$ if there is an unbounded fanin constant depth circuit computing f from the bits of g.*

2.1 Iterated Addition of Logarithmically many n-bit numbers

Theorem 2.1. *$IterADD_{\log n, n} \leq_{cd} IterADD_{\log\log n, O(n)}$*

Proof: We will denote $\log n = l$. We are given l many n-bit numbers $a_1, \ldots, a_l$, where $bin(a_i) = a_{i,n-1} \ldots a_{i,1}a_{i,0}$. We add all the l many bits at ith position of all numbers. we know if we add m bits then we have at most $\log m$ many bits. So adding the l many bits will take $\log l = \log\log n$ many bits. $s_k = \sum_{i=1}^{l} a_{i,k}$. Hence $bin(s_k) = s_{k,\log l-1} \ldots s_{k,1}s_{k,0}$. Hence $\sum_{i=1}^{l} a_{i,k} = \sum_{j=0}^{\log l-1} s_{k,j}2^j$

$$\sum_{i=1}^{l} a_i = \sum_{i=1}^{l}\sum_{k=0}^{n-1} a_{i,k}2^k = \sum_{k=0}^{n-1}\sum_{i=1}^{l} a_{k,k}2^k = \sum_{k=0}^{n-1}\sum_{j=0}^{\log\log n-1} s_{k,j}2^j \cdot 2^k = \sum_{j=0}^{\log\log n-1}\sum_{k=0}^{n-1} s_{k,j}2^{j+k}$$

So this is converted to addition of $\log\log n$ many numbers of at most $n + \log\log n = O(n)$ many bits. ■

Recursing like this we have $IterADD_{\log\log n, n} \leq_{cd} IterAdd_{2,O(n)}$. Hence

Theorem 2.2. *$IterADD_{\log n, n} \leq_{cd} IterAdd_{2,O(n)}$ and therefore $IterADD_{\log n, n} \in AC^0$*

Remark: Apart from this $O(\log^* n)$ method to prove $IterADD_{\log n, n} \in AC^0$ there is also another method in Vinay Kumar's Lecture Notes

2.2 Iterated Addition of n many n-bit numbers

We know $IterAdd_{n,n} \leq_{cd} IterAdd_{n,1}$ but we dont know anything about $IterAdd_{n,n} \leq_{cd} IterAdd_{\log n, n}$. If that happens it will put $IterAdd_{n,n}$ to AC^0.
Remark: $IterAdd_{n,1}$ is also known as $BCOUNT_n$.

Theorem 2.3. *$IterAdd_{n,n} \leq_{cd} IterAdd_{n,1}$*

Proof: Let we given n many n-bit numbers $a_1, \ldots, a_n$, where $bin(a_i) = a_{i,n-1} \ldots a_{i,1}a_{i,0}$. First we compute $s_k = \sum_{i=1}^{n} a_{i,k}$ using $BCOUNT_n$ for all $0 \leq k \leq n$. Now it becomes addition of $\log n$ many $O(n)$ bit numbers which we already know is in AC^0 by Theorem 2.2. Hence $IterAdd_{n,n} \leq_{cd} BCOUNT_n$ ■

3 $IterAdd_{n,n} \equiv BCOUNT_n \equiv Threshold_{n,n} \equiv Majority_n \equiv MULT_n$

Problem: $MULT$
Input: 2 n-bit numbers $a = a_0, \ldots, a_{n-1}$, $b = b_0, \ldots, b_{n-1}$
Output: $c = a \cdot b$

Theorem 3.1. *$MULT_{n,n} \leq IterAdd_{n,n}$*

Proof: Given a, b where $bin(a) = a_{n-1} \cdots a_1 a_0$ and $bin(b) = b_{n-1} \cdots b_1 b_0$ then obviously

$$a \cdot b = \sum_{i=0}^{n-1} a \cdot b_i \cdot 2^i$$

Define for all $0 \leq i \leq n-1$

$$c_i = \begin{cases} 0^{n-i-1}a_{n-1} \cdots a_1 a_0 0^u & \text{when } b_i = 1 \\ 0^{2n-1} & \text{otherwise} \end{cases}$$

i.e. $c_i = a \cdot 2^i$ if $b_i = 1$. Each c_i is of $2n-1 = O(n)$ many bits long. Hence we have $a \cdot b = \sum_{i=0}^{n-1} c_i$. Hence now we can use the $IterAdd_{n,n}$ gate to add the n many $O(n)$ many bits to find the multiplication of a and b. Therefore $MULT_n \leq IterAdd_{n,n}$. ■

Problem: $Majority_n$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if at least half of the bits are 1

Theorem 3.2. *$Majority \leq MULT$*

Proof: Given $a_0, \ldots, a_{n-1}$. Take the number a such that $bin(a) = a_{n-1} \cdots a_1 a_0$. Denote $l := \log n$. Define

$$A = \sum_{i=0}^{n-1} a_i \cdot 2^{li} \text{ and } B = \sum_{i=0}^{n-1} 2^{li}$$

where both A and B consists of n blocks of length l. We took l length block because summation of n bits takes at most l bits. Let $C = A \cdot B$ We represent C in binary as l length blocks where $C = \sum_{i=0}^{2n-1} c_i \cdot 2^{li}$. Each c_i is a l length block. Then the middle block c_{n-1} have exactly the computation of the sum of the a_i. Therefore $c_{n-1} = \sum_{i=0}^{n-1} a_i$.

A and B are constructed in constant depth and fed into $MULT$ gates yielding C. Now we have to compare c_{n-1} with $\frac{n}{2}$ which can be done in constant depth. ■

Problem: $ExactThreshold_{n,m}$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if $\sum_{i=0}^{n-1} = m$

We have another similar problem but we have greater than instead of equality.

Problem: $Threshold_{n,m}$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if $\sum_{i=0}^{n-1} \geq m$

Theorem 3.3. *$BCOUNT \leq ExactThreshold \leq Threshold \leq Majority$*

Proof: $\underline{BCOUNT \leq ExactThreshold}$: Let $\sum_{i=0}^{n-1} a_i = \sum_{i=0}^{l=\log n} s_i \cdot 2^i$. Let for all $0 \leq j \leq l$, R_j denote the set of all numbers $r \in \{0, \ldots, n\}$ whose j-th bit is 1 in its binary representation . Then we can say

$$s_j = \bigvee_{r \in R_j} \left[\!\!\left[\sum_{i=0}^{n-1} a_i = r \right]\!\!\right]$$

Now R_j don't depend on the input but only on the input n so it can be hardwired this into the circuit. Thus we have a circuit for $BCOUNT$ which uses $ExactThreshold$.

$\underline{ExactThreshold \leq Threshold}$: We know for any r and a variable x certainly

$$[\![x = r]\!] = [\![x \geq r]\!] \wedge [\![x < r+1]\!]$$

With this we have a constant depth circuit for $ExactThreshold$ using the $Threshold$ gates.

$\underline{Threshold \leq Majority}$: We are given $a_0, \ldots, a_{n-1}$. Let we want to find $\sum_{i=0}^{n-1} \geq m$ then we have this following relations

$$\sum_{i=0}^{n-1} a_i \geq m \iff \begin{cases} Maj_{2n-2m}\left(a_0, \ldots, a_n, \underbrace{1, \ldots, 1}_{n-2m}\right) & \text{wher } m < \frac{n}{2} \\ Maj_{2m}\left(a_0, \ldots, a_n, \underbrace{0, \ldots, 0}_{n-2m}\right) & \text{wher } m \geq \frac{n}{2} \end{cases}$$

This Maj_{2n-2m} and Maj_{2m} can be constructed in constant depth. ■

Remark: Hence using the theorems above we have the final relation

$$Majority \leq MULT \leq IterAdd_{n,n} \leq BCOUNT \leq ExactThreshold \leq Threshold \leq Majority$$

which gives the following corollary

Corollary 3.4. *$IterAdd_{n,n} \equiv BCOUNT \equiv Threshold \equiv Majority \equiv MULT$*

Definition 3.1 (TC^0)**.** *Constant depth polynomial size unbounded fanin circuit family using the gates $\wedge, \vee, \neg, Maj$.*
Alternating Definition: Constant depth polynomial size unbounded fanin circuit family using the gates $\neg, Maj$.

Theorem 3.5. *Both the definitions of TC^0 are equivalent.*

Theorem 3.6. *$IterAdd_{n,n}$, $BCOUNT$, $MULT \in TC^0$*

Proof: By Corollary 3.4 we have the result. ■

4 $BCOUNT \equiv UCOUNT \equiv SORT$

5 Parallel Random-Access Machine (PRAM)