# 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 $i$th 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 $IterAdd_{n,n} \equiv BCOUNT_n \equiv Threshold_{n,m} \equiv Majority_n \equiv MULT_n \equiv SORT_{n,n}$

**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 $bin(a) = a_{n-1} \cdots a_1a_0$ and $bin(b) = b_{n-1} \cdots b_1b_0$ 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.

*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. ■

**Problem:** *SORT*
**Input:** $n$ numbers with $n$ bits each
**Output:** The sequence of the input numbers in non-decreasing order.

**Definition 3.2.** *We define* $un_n(k) \triangleq 1^k 0^{n-k}$ *where* $k \in \{0, \ldots, n\}$

**Problem:** *UCOUNT*
**Input:** $a_0, \ldots, a_{n-1} \in \{0, 1\}$

**Output:** $un_n\left(\sum_{i=0}^{n-1} a_i\right)$

**Lemma 3.7.** $BCOUNT \leq UCOUNT$

***Proof:*** Given $a_0, \ldots, a_{n-1}$ suppose $b_1 \ldots b_n = un_n\left(\sum_{i=0}^{n-1} a_i\right)$. Also let $b_0 = 1$ and $b_{n+1} = 1$. Then in the number $b_0 b_1 \ldots b_n b_{n+1}$ there is at least one 1 from left and at least one 0 from right. Now take

$$d_j = b_j \wedge \neg(b_{j+1})$$

for all $0 \leq j \leq n$. Then if $d_j = 1$ that means $b_j = 1$ and $b_{j+1} = 0$ hence $b_j$ is the last bit which is 1 afterwards every bit is 0. Hence there are in total $j$ many 1's except $b_0$. Therefore $\sum_{i=0}^{n-1} a_i = j$. So we take all $r \in 0, \ldots, n$ such that the $j$th bit of $r$ is on then define

$$c_j = \bigwedge_{i \in R_j} d_i$$

Hence if $c_j = 1$ then we can say $\sum_{i=0}^{n-1} a_i$ is such a number whose $j$th bit is 1. Thus we take $BCOUNT(a_0, \ldots, a_{n-1}) = c_{\log n - 1} \cdots c_1 c_0$ ■

**Theorem 3.8.** $UCOUNT \equiv BCOUNT$

**Theorem 3.9.** $UCOUNT \leq SORT \leq UCOUNT$

***Proof:*** $UCOUNT \leq SORT$: Given $a_0, \ldots, a_{n-1}$ define $A_i = a_i \underbrace{0 \cdots, 0}_{n-1}$ for all $0 \leq i \leq n-1$. Sorting these numbers the sequence of most significant bits in the ordered sequence is the $un_n \left( \sum_{i=0}^{n-1} a_i \right)$ reversed.

$SORT \leq UCOUNT$: Given $a_i = a_{i,n-1} \cdots, a_{i,1} a_{0,0}$ for all $1 \leq i \leq n$. Define

$$c_{i,j} \triangleq [\![ (a_i < a_j) \vee ((a_i = a_j) \wedge i \leq j) ]\!]$$

$\forall\ 0 \leq i, j \leq n-1$. Now we define for all $1 \leq j \leq n$

$$c_j \triangleq UCOUNT(c_{0,j} \ldots c_{n-1,j})$$

Hence first of all the number of 1's in $c_{0,j} \ldots c_{n-1,j}$ is the number of $a_i$'s with value strictly less than $a_j$ or if equal then index is less than or equal to $j$. Hence it represents the position of $a_j$ when the numbers are sorted. Therefore $c_j$ is the $n$-bit unary representation of the position of $a_j$ in the ordered output sequence. If the ordered sequence is $a'_1, \ldots, a'_n$ then $a'_{c_j} = a_j$. Let $a'_i = a'_{i,n-1} \cdots a'_{i,0} a'_{i,1} a'_{i,0}$ for $1 \leq i \leq n$. Then

$$a'_{i,k} = 1 \iff [\![ a'_i = a_j ]\!] \wedge [\![ a_{j,k} = 1 ]\!]$$

Hence $a'_{i,k} = \bigwedge_{1 \leq j \leq n} \left( [\![ c_j = 1^i 0^{n-i} ]\!] \wedge a_{j,k} \right)$. ■

**Corollary 3.10.** $SORT \in TC^0$

# 4 Parallel Random-Access Machine (PRAM)

In [KR90] many Parallel Machine Models are very well written and explained

**Definition 4.1** (EREW-PRAM). *E:= Exclusive, R := Read, W := Write*

**Question 1.** *Can $n$, $n$-bit numbers be sorted in $O(\log n)$ time on an EREW-PRAM? Bir arithmetic is allowed and each with polynomially many processors. Bit operation takes $O(1)$ time. Hence is $SORT \in EREW[\log n, poly(n)]$?*

**Question 2.** *What about if bit arithmetic are not allowed?*

**Question 3.** *If comparisons are allowed can we say anything?*

**Answer:** AKS (Ajtai–Komlós–Szemerédi) in their paper [Pad11] showed in $O(\log n)$ time

**Question 4.** *If number of processors is $O(n)$ what can be said?*

There are also other parallel machine models: CRCW (C:= Concurrent), CREW, CROW (O:= Owner), OROW. Concurrent means everyone can access the memory concurrently. Owner means only the processor who is the owner of a memory can access it.

In CRCW allows simultaneous read and writes. Hence we have to resolve write conflicts. Some commonly used methods of resolving write conflicts are $COMMON$, $ARBITRARY$, $PRIORITY$ models.

**Definition 4.2** ($NC^1$). *Class of languages accepted by circuits of depth $O(\log n)$ and size $n^{O(1)}$ and fanin 2*

**Definition 4.3.** *Formulas are trees i.e. circuits with fanout 1*

**Theorem 4.1.** *In $NC^1$ formulas and circuits are equivalent.*

**Theorem 4.2.** $NC^1 \subseteq OROW[\log n, poly(n)]$

***Proof:*** For any circuit $C \in NC^1$ we create the OROW PRAM where each gate of $C$ is a processor in the PRAM and each processor has the writing access of their own memory and for any edge $u \to v$ in $C$ processor $v$ has the reading access of the memory of $u$. With this OROW PRAM each processor can read memory from its children then writes the computed value in his memory location thus it computes $C$. Since $C$ has size $n^{O(1)}$ the PRAM has $n^{O(1)}$ many processors and since the circuit has depth $O(\log n)$ the OROW PRAM takes $O(\log n)$ time to compute. ■

**Open Question 4.3.** $NC^1 \stackrel{?}{\supseteq} OROW[\log n, poly(n)]$

**Definition 4.4** ($AC^k$, $NC^k$)**.** *The class of circuits consists of the gates $(\vee, \wedge, \neg)$ (The subscript $n$ or $1$ denotes the fanin) of polynomial size and depth $O(1) = O(\log^k n)$*
*Similarly for $NC^k$ but with fanin 2*

**Theorem 4.4.** $AC^1 = CRCW[\log n, poly(n)]$

***Proof:*** $AC^1 \subseteq CRCW[\log n, poly(n)]$: (COMMON) For any gate all its parents have similarly reading access of the memory of the gate and all its childs has writing access. But we use COMMON model to resolve write conflicts. For an $OR$ gate if anyone evaluates to be 0 then it ignores and if its 1 then it writes 1. Thus everybody writes 1. Every $OR$ gate by has 0 previously in the memory. Similarly for $AND$ gate its the opposite.
$AC^1 \supseteq CRCW[\log n, poly(n)]$: [Vol99, Theorem 2.56], [SV84] ■

# 5 Some Circuit Complexity Class Relations

**Theorem 5.1.** $AC^0 \subseteq NC^1$

***Proof:*** For all unbounded gate in $AC^1$ circuit $C$ with fanin $s$ we can replace each gate with $s$ many same gates fo fanin 2 and depth log s. Thus we have $AC^0 \subseteq NC^1$ ■

**Theorem 5.2.** $TC^0 \subseteq NC^1$

***Proof:*** We will first show that using Redundant Algebra $ADD_{2n} \in NC^0$. In Redundant Algebra with base 4 while adding two digits the result can be at most in normal integers $-6, \ldots, 6$. We only have to check for $\pm 6, \pm 5, \pm 3$. Other case we dont have to watch out.

$$6 = 12 \qquad 5 = 11 \qquad 3 = 1\bar{1}$$
$$\bar{6} = \bar{1}\bar{2} \qquad \bar{5} = \bar{1}\bar{1} \qquad \bar{3} = \bar{1}1$$

For $\pm 4$ it is the normal representation in this system ie 10 and $\bar{1}0$ respectively. Now whenever we add two digits in this system in the sum result can be at most 2 digits. Among them we call the right most digit the sum digit or $s$ and the left digit the carry digit $c$ becasue it becomes the carry for the addition. Now see in all of the numbers the sum digit $|s| \leq 2$ and carry digit $|c| \leq 1$ So whenever we add a carry generated before to the current sum no new carry is generated becasue of the carry. So We dont have to look for carry generation and propagation. The carry generated at the previous position will add to the sum digit in the current position after getting added to that it will not further propagate after getting added the final

digit at the current place will be between 3 and $\overline{3}$ So we proved that addition of two $n$-bit numbers using Redundant algebra is in $NC^0$

Now we will show converting a number $n$ from base 2 to base 4 is in $NC^0$. let

$$n = \sum_{k=0}^{m} a_k 2^k = \sum_{k=0}^{\lfloor \frac{m}{2} \rfloor} (a_{2k+1} \times 2 + a_{2k})\, 4^k \qquad \text{if } m \text{ is odd then } a_{m+1} = 0$$

So we just take two bits in binary multiply the left one with 2 and add to the right one and we have the digit at base 4 in that position. So we can change base in $NC^0$. From now on we will by default assume all the addition is done using Redundant Algebra.

Then we are done in showing $TC^0 \subseteq NC^1$. We will first show that adding $n$ bits is in $NC^1$ ie $BCOUNT \in NC^1$. So we have $n$ bits $a_n, a_{n-1}, \ldots, a_1$. We will group the bits into groups of two bits and add the two bits in each group. We can do the addition for all groups in parallel. And since we have already shown addition of two $k$-bit numbers is in $NC^0$ this takes constant depth and size since we are adding two bits. Now the number of bits are halved. We do tha same process again and again. Each time it takes $poly(n)$ size and constant depth and at each iteration the number of numbers becomes half. So this whole process takes $O(\log n)$ many iterations. So adding $n$ bits takes $poly(n)$ size and depth $O(\log n)$. So $BCOUNT \in NC^1$

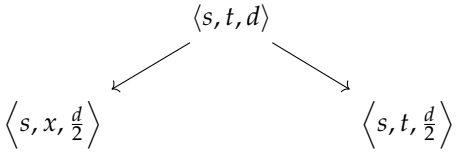
Now We will show that $MAJORITY \in NC^1$. Let the addition of $n$ bits we got is $a$ and the Redundant Algebra representation of $\lceil \frac{n}{2} \rceil$ is $b$. Now we can calculate $-b$ with reversing the sign of every digit in $b$ ie if $b = \sum_{i=0}^{m} a_k \times 4^k$ then $-b = \sum_{i=0}^{m} \overline{a_k} \times 4^k$. Now we add $a$ and $-b$ which can be done in $NC^0$. And now we will look for the left most negative digit. If there is nonw we output 1 i.e. majority of the bits are 1 and if there exists a negative bit then output is 0. Hence we have $MAJORITY \in TC^0$. So $TC^0 \subseteq NC^1$. ■

**Definition 5.1** ($SAC^k$)**.** *The class of circuits consists of the gates* $(\vee, \wedge, \neg)$ *of polynomial size and depth* $O(1) = O(\log^k n)$ *but semi unbounded fanin i.e either* $\wedge$ *gates have unbounded fanin and* $\vee$ *gate have bounded fanin or* $\wedge$ *gates have bounded fanin and* $\vee$ *gates have unbounded fanin.*

**Theorem 5.3.** $AC^0 \subseteq TC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq SAC^1 \subseteq AC^1 \subseteq NC^2 \subseteq AC^2 \subseteq \cdots \subseteq NC^i \subseteq AC^i \subseteq \cdots \subseteq NC = AC \subseteq P$

***Proof:*** $NC^1 \subseteq L$: The idea is to simply evaluate the circuit using a depth-first search, which can be defined inductively as follows: visit the output gate of the circuit, visit the gates of the left subcircuit, visit the gates of the right subcircuit. At any moment, we store the number of the current gate, the path that led us there (as a sequence of left's and right's) and any partial values that were already computed. For example, we could have $L$, $R$ (0), $R$ (1), $L$, $L$, 347. This would mean that we are visiting gate 347 and we got there by going left, right, right, left and left. The 0 after the first R indicates that the left input of the second gate on the path evaluated to 0. When we are done visiting a gate (and computed its value), we return to the previous gate on the path. Note that we can recompute the number of that gate by following the path from the beginning. The space requirements of this algorithm are therefore determined by the maximum length of the path, which is equal to the depth of the circuit. The uniformity of the circuit is used when traveling through the circuit and evaluating its gates.

$NL \subseteq SAC^1$: We know $PATH$ is $NL - complete$. Let there is a path $s \rightsquigarrow t$ of length $d$. Then there exists a vertex $x$ in between $s$ and $t$ such that there exists a path of length $\frac{d}{2}$ from $s \rightsquigarrow x$ and $x \rightsquigarrow t$.



Since there exists one such vertex $x$ and in circuit we will add all this in a big $\vee$ gate for all vertices in the graph. We can represent this as a circuit

$$\langle s,t,d\rangle = \bigvee_{x\in V}\left(\left\langle s,x,\frac{d}{2}\right\rangle \wedge \left\langle s,t,\frac{d}{2}\right\rangle\right)$$

Like this we extend it to 1 length paths. In this circuit we are using $\vee$ gates with unbounded fanin and $\wedge$ gates with 2 fanin. Hence we have $NL \subseteq SAC^1$ ■

# References


[SV84] Larry Stockmeyer and Uzi Vishkin. "Simulation of Parallel Random Access Machines by Circuits". In: *SIAM Journal on Computing* 13.2 (1984), pp. 409–422. DOI: 10.1137/0213027. eprint: https://doi.org/10.1137/0213027. URL: https://doi.org/10.1137/0213027.

[KR90] Richard M. KARP and Vijaya RAMACHANDRAN. "CHAPTER 17 - Parallel Algorithms for Shared-Memory Machines". In: *Algorithms and Complexity*. Ed. by JAN VAN LEEUWEN. Handbook of Theoretical Computer Science. Amsterdam: Elsevier, 1990, pp. 869–941. ISBN: 978-0-444-88071-0. DOI: https://doi.org/10.1016/B978-0-444-88071-0.50022-9. URL: https://www.sciencedirect.com/science/article/pii/B9780444880710500229.

[Vol99] Heribert Vollmer. "Relations to Other Computation Models". In: *Introduction to Circuit Complexity: A Uniform Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 35–78. ISBN: 978-3-662-03927-4. DOI: 10.1007/978-3-662-03927-4_3. URL: https://doi.org/10.1007/978-3-662-03927-4_3.

[Pad11] "Ajtai–Komlós–Szemerédi Sorting Network". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 16–16. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_2379. URL: https://doi.org/10.1007/978-0-387-09766-4_2379.