Addition is $\mathcal{NC}$

We consider addittion of two n-bit binary numerals, a. and b. The digits of these numerals are a_i and b_i for $0 \leq i < n$. The sum we are trying to compute is s, whose digits are $\{s_i\}$ for $0 \leq i \leq n$. We let c_i be the i^{th} carry bit during the addition, for $0 \leq i \leq n$. We note that $s_i = (a_i + b_i + c_i)$ mod 2, where c_i is the i^{th} carry bit. The values of c_i and s_i can be computed by the traditional ripple method, as in the following program.

$c_0 = 0$
for($i = 0$ to $n - 1$)
{
$s_i = (a_i + b_i + c_i) \mod 2$
if($a_i + b_i == 0$) $c_{i+1} = 0$
else if($a_i + b_i == 1$) $c_{i+1} = c_i$
else if($a_i + b_i == 2$) $c_{i+1} = 1$
}
$s_n = c_n$

To have an $\mathcal{NC}$ algorithm, we must be able to compute all carry bits in $O(\log^k n)$ steps using $O(n^k)$ processors, for some constant k. For this problem, we can choose $k = 1$.

Changing a Sequential Algorithm to $\mathcal{NC}$

Consider the following straight line program.
$u = 1$
$v = u$
$x = v$
$y = x$
$z = y$

We can see that the value of each of the variables is 1, but that computation takes five steps by a sequential processor.

Our method is to store, at each variable, the actual valuue if we know it, otherwise instructions for how to find the value. Five processors, working simultaneously, can execute the following four steps resulting in a value of 1 for each variable.

Step 1:	Step 2:	Step 3:	Step 4:
value(u) = 1	value(v) = 1	value(x) = 1	value(z) = 1
value (v) = copy value(u)	value(x) = copy value(u)	value(y) = 1	
value (x) = copy value(v)	value(y) = copy value(u)	value(z) = copy value(u)	
value (y) = copy value(x)	value(z) = copy value(x)		
value (z) = copy value(y)			

Step 1 should be clear; the value of each variable except u is obtained by copying the value of another variable. The processor that writes that instruction does not yet know what that copied value will be.

Step 2 consists of four processors executing *composition*, just as in the document `oddNC.pdf`. The value of v is now 1, because its instruction is to copy the value of u, which is previously known to be 1. The actual value of x is not known, but by combining the first three lines of Step 1, we know that it is a copy of the value of u. The processor does not know that $u = 1$, since it would require two steps to fetch that value and write it to x, hence "copy value(u)" is written to x. Similarly, "copy value(x)" is written to z.

In Step 3, the values of x and y are determined, but the value of z is not: the instruction "copy value(u)" is stored in z. Step 4 finishes the algorithm.

Decreasing the number of steps from five to four does not seem like much, but more generally, if we have a chain of assignments with n variables, we can evaluate all of them in $O(\log n)$ steps instead of n by using n processors.

The $\mathcal{NC}$ Algorithm $\mathcal{A}$ for Addition

During the first step of $\mathcal{A}$, we compute a statement for each carry bit. Each statement will be one of the following three: value$(c_{i+1}) = 0$, value $(c_{i+1}) = 1$, or value $(c_{i+1}) =$ copy value c_i, depending on the value of $a_i + b_i$. We indicate the steps of $\mathcal{A}$ with the following pseudocode. For convenience, we assume $n = 2^m$ We use the notation rhs$[i]$ to denote the right hand side of the assignment of value(c_i), which is either 0, 1, or "copy value(c_j)" for some $j < i$.

for all $0 \le i \le n$ **in parallel** Step (1)
    if$(a_i + b_i == 0)$
        rhs$[i+1] = 0$
    else if$(a_i + b_i == 2)$
        rhs$[i+1] = 1$
    else
        rhs$[i+1] =$ "copy value(c_i)"
for (int $\ell = 0$; $\ell \le m$; ℓ++) // **sequentially**
    for all ($i =$ positive even multiple of 2^ℓ not more than n) **in parallel** (Step $\ell + 1$)
        if (rhs$[i] =$ "copy value(c_j)") // $j = i - 2^\ell$
            rhs$[i] =$ rhs$[j]$
for (int $\ell =$ m–1; $\ell \ge 0$; ℓ– –) // **sequentially**
    for all ($i =$ positive odd multiple of 2^ℓ less than n) **in parallel** (Step $2m - \ell + 1$)
        if (rhs$[i] =$ "copy value(c_j)") // $j = i - 2^\ell$
            rhs$[i] =$ rhs$[j]$
for (int $i = 0$;$i \le n$;n++)
    $s_i = (a_i + b_i + c_i) \mod 2$

The number of steps is $2m + 2 = O(\log n)$, and the number of processors needed does not exceed $n + 1$ at any step. Thus, $\mathcal{A}$ is an $\mathcal{NC}$ algorithm.

Example

We now work through an example instance of the addition problem, where $n = 32$

	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a		0	1	0	1	0	1	0	1	0	0	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1	0	0	1	1	1	1	0
b		0	1	1	0	1	0	1	0	1	1	0	1	0	0	1	0	1	1	1	1	1	0	0	0	0	0	1	1	1	0	1	1
a+b		0	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	1	0	0	1	0	1	2	2	1	2	1
c	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0
s	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	1	1	0	0	1

$$\begin{aligned} c_0 &= 0 \\ c_1 &= c_0 \\ c_2 &= 1 \\ c_3 &= c_2 \\ c_4 &= 1 \\ c_5 &= 1 \\ c_6 &= c_5 \\ c_7 &= 0 \\ c_8 &= c_7 \\ c_9 &= 0 \\ c_{10} &= 0 \\ c_{11} &= c_{10} \\ c_{12} &= c_{11} \\ c_{13} &= 1 \\ c_{14} &= 1 \\ c_{15} &= c_{14} \\ c_{16} &= c_{15} \\ c_{17} &= c_{16} \\ c_{18} &= c_{17} \\ c_{19} &= c_{18} \\ c_{20} &= c_{19} \\ c_{21} &= c_{20} \\ c_{22} &= c_{21} \\ c_{23} &= c_{22} \\ c_{24} &= c_{23} \\ c_{25} &= c_{24} \\ c_{26} &= c_{25} \\ c_{27} &= c_{26} \\ c_{28} &= c_{27} \\ c_{29} &= c_{28} \\ c_{30} &= c_{29} \\ c_{31} &= 1 \\ c_{32} &= 0 \end{aligned}$$

(1)

$$\begin{aligned} c_0 &= 0 \\ c_2 &= 1 \\ c_4 &= 1 \\ c_6 &= 1 \\ c_8 &= 0 \\ c_{10} &= 0 \\ c_{12} &= c_{10} \\ c_{14} &= 1 \\ c_{16} &= c_{14} \\ c_{18} &= c_{16} \\ c_{20} &= c_{18} \\ c_{22} &= c_{20} \\ c_{24} &= c_{22} \\ c_{26} &= c_{24} \\ c_{28} &= c_{26} \\ c_{30} &= c_{28} \\ c_{32} &= 0 \end{aligned}$$

(2)

$$\begin{aligned} c_0 &= 0 \\ c_4 &= 1 \\ c_8 &= 0 \\ c_{12} &= 0 \\ c_{16} &= 1 \\ c_{20} &= c_{16} \\ c_{24} &= c_{20} \\ c_{28} &= c_{24} \\ c_{32} &= 0 \end{aligned}$$

(3)

$$\begin{aligned} c_0 &= 0 \\ c_8 &= 0 \\ c_{16} &= 1 \\ c_{24} &= c_{16} \\ c_{32} &= 0 \end{aligned}$$

(4)

$$\begin{aligned} c_0 &= 0 \\ c_{16} &= 1 \\ c_{32} &= 0 \end{aligned}$$

(5)

$$\begin{aligned} c_0 &= 0 \\ c_{32} &= 0 \end{aligned}$$

(6)

$$c_{16} = 1$$

(7)

$$\begin{aligned} c_8 &= 0 \\ c_{24} &= 1 \end{aligned}$$

(8)

$$\begin{aligned} c_4 &= 1 \\ c_{12} &= 0 \\ c_{20} &= 1 \\ c_{28} &= 1 \end{aligned}$$

(9)

$$\begin{aligned} c_2 &= 1 \\ c_6 &= 1 \\ c_{10} &= 0 \\ c_{14} &= 1 \\ c_{18} &= 1 \\ c_{22} &= 1 \\ c_{26} &= 1 \\ c_{30} &= 1 \end{aligned}$$

(10)

$$\begin{aligned} c_1 &= 0 \\ c_3 &= 1 \\ c_5 &= 1 \\ c_7 &= 0 \\ c_9 &= 0 \\ c_{11} &= 0 \\ c_{13} &= 1 \\ c_{15} &= 1 \\ c_{17} &= 1 \\ c_{19} &= 1 \\ c_{21} &= 1 \\ c_{23} &= 1 \\ c_{25} &= 1 \\ c_{27} &= 1 \\ c_{29} &= 1 \\ c_{31} &= 1 \end{aligned}$$

In our tables, we delete the words "value" and "copy value" to save space. For each $0 \le t \le 2m + 1 = 11$, we show the output of Step t. In column (1), we show the output for c_i for each i. In column (2), we show entries for even i. Despite the fact that our pseudocode for $\mathcal{A}$ does not recalculate final (*i.e.,* constant) values, we show those previously calculated values in each column for uniformity of appearance.
In columns (3) through (6), we show the output for even multiples of 2^ℓ for $\ell = 1, 2, 3, 4$. In columns (7) through (11), we show the output for odd multiples of 2^ℓ for $\ell = 4, 3, 2, 1, 0$.