NC1 Division

A. Chiu [§], G. Davida [¶], and B. Litow, **communicating author** [‖]

September 8, 2000

Abstract

Beame, Cook and Hoover were the first to exhibit a log-depth, polynomial size circuit family for integer division. However, the family was not log-uniform. In this paper we describe log-depth, polynomial size, log-uniform, i.e., NC1 circuit family for integer division. An interesting consequence of this fact and known results from parallel complexity theory is that iterated integer product is in $O(\log n)$ space.

1 Parallel complexity of division

The problem

In this paper we consider the parallel complexity of nonnegative integer division. Conventional binary notation is assumed for both inputs and outputs. Integer division is simply the computation of the quotient $\lfloor x/y \rfloor$ of integers x and y. It has been known for some time that integer addition and multiplication can be carried out using Boolean circuits of polynomial size and logarithmic depth. That is, given two integers x and y whose binary notations each require at most n bits, there exist Boolean circuits of size $n^{O(1)}$ gates and $O(\log n)$ depth that compute their sum $x + y$ and product $x \cdot y$ in binary notation, respectively. In fact, one can say rather more than this. For each operation there exists a deterministic Turing machine, which uses $O(\log n)$ space and outputs the adjacency list for the circuit that handles n-bit integers. That is, the circuits form a log-uniform family. See [15] for excellent coverage on these circuits. Notice that any circuit family produced using $\log n$ space (we will always refer to deterministic Turing machines) necessarily has polynomial size.

The situation for division has been different until now. It has long been known that log-uniform, polynomial size, $O(\log^2 n)$ depth circuits exist for division. See [15]. The question of the existence of $\log n$ depth circuits was affirmatively resolved by Beame, Cook and Hoover in [1]. Unfortunately, their division circuits are not log-uniform, although they can be produced in P.

[§]EECS Dept. , University of Wisconsin-Milwaukee, Milwaukee, WI, USA

[¶]EECS Dept. , University of Wisconsin-Milwaukee, Milwaukee, WI, USA davida@cs.uwm.edu

[‖]School of Information Technology, James Cook University, Townsville, Qld. 4811, Australia bruce@cs.jcu.edu.au

Following the Beame, Cook, Hoover result, Davida and Litow showed how to compute integer division by log-depth, polynomial size Boolean circuits in [6]. However, these circuits are not log-uniform, although they can be computed using just slightly more space. Using a method due to Reif and chinese remaindering it is shown there that $\log n$ depth, $n^{O(1)}$ size division circuits can be produced using $O(\log(n) \cdot \log\log(n))$ space. See [12]. The Davida-Litow algorithm is based on chinese remainder (multi-residue) representation for intermediate computations. This chinese remainder approach is reexamined by Kaltofen and Hitz in [7]. Their division circuits also fail to be log-uniform.

In this paper we demonstrate, using a chinese remainder approach that division can be computed by log-uniform, $\log n$ depth circuits. Formally, we will prove

Theorem 1 *Integer division is in NC1.*

Combining this result with Theorem 2 and Theorem 5 we see that iterated product can be computed in $O(\log n)$ space. Some structural complexity interest attaches to this result, since iterated product appeared to be a candidate for a problem computable in P but not computable in $O(\log n)$ space.

Basic concepts

Our parallel computation model is the collection of log-uniform Boolean circuit families. This model has been widely used for exploring basic questions in parallel complexity. A Boolean circuit is a DAG (directed acyclic graph) whose nodes of indegree 0 are inputs, one bit per input, and whose remaining nodes each carry one of the labels: negation; conjunction; disjunction. These labelled nodes are called gates. Disjunction and conjunction gates have indegree 2 and negation gates have indegree 1. There is no constraint on outdegree. The gates having outdegree 0 are the outputs. The size of a circuit is the number of nodes and its depth is the number of levels obtained via a topological sort of the DAG. Since all gates at the same level can operate at the same instant, the time (treating each gate operation as $O(1)$ time cost) for outputs to appear is just the depth.

The class of problems computable by log-uniform, polynomial size, $\log^k n$ depth circuits is called NCk, and the union over all NCk is called NC. It is open whether NC is an hierarchy, i.e., whether there exist $1 \leq k < k'$ such that NCk is properly contained in NCk'. NC serves as a good model of problems that can be feasibly (polynomial size) computed in extremely fast ($log^k n$ depth) parallel time. NC has been criticised from the standpoint of realistic parallel algorithm engineering, but is generally regarded as a good theoretical benchmark for parallel time complexity of problems. See [9].

The main input size variable will be n and NC1 will refer to that size. However, we may need to express complexity in terms of other size variables. We use the expression 'NC1 in terms of c' to mean that a problem can be computed by circuits of $c^{O(1)}$ size, $O(\log c)$ depth and that the circuits can be computed in $O(\log c)$ space.

The basic theory of NC and in particular, NC1 and its associated uniformity issues can be found in [3, 13, 5, 1]. We will need the following facts from this theory. Iterated

product is the computation of the binary notations of all of the prefix products $x_1 \cdots x_i$ for $i = 1, \ldots, n$ where $x_1, \ldots, x_n$ are n-bit integers in binary notation. Powering is the computation of the binary notations for $x, x^2, \ldots, x^n$ where x is an n-bit integer. The next theorem is a principal result from [1].

Theorem 2 *Either division, iterated product and powering are all in NC1, or none is in NC1.*

The next fact is relatively straightforward to prove, see e.g., Lemma 4.5 [6].

Theorem 3 *If $x_1, \ldots, x_n, m < n$, the computation of $x_1 + \cdots + x_n$, $x_1 + \cdots + x_n \bmod m$ and $x_1 \cdot x_2 \bmod m$ are in NC1.*

Proof : A proof that $x_1 + \cdots + x_n$ is in NC1 may be found as Lemma 4.5 in [6]. Letting $y = x_1 + \cdots + x_n$, note that $y < n^2$. In parallel one can find $\lfloor y/m \rfloor$ by testing for the least $q < n^2$ such that $y \leq q \cdot m$. A single subtraction now yields $y \bmod m$. The same technique can be used to compute $x_1 \cdot x_2 \bmod m$. All binary notations are at most $O(\log n)$ bits so all of these operations are in NC1. □

Theorem 4 *The computation of $x_1 \cdots x_n \bmod p^\ell$ where $x_1, \ldots, x_n < p$, $p^\ell < n$ and p is prime is in NC1.*

Proof : This is Theorem 4.2 in [1]. We need only the case of prime moduli and sketch the proof here.

We build a table in $O(\log n)$ space with a row for each prime $p < n$. The row for p will have $p-1$ entries, one for each integer $1, \ldots, p-1$. Let g be a primitive element for p, i.e., $\{g, g^2, \ldots, g^{p-1}\} \bmod p = \{1, \ldots, p-1\}$. A primitive element can be found in $O(\log n)$ space by brute force, essentially. The k-th entry in the row for p will be μ_k such that $k \equiv g^{\mu_k} \bmod p$.

To compute $x_1 \cdots x_n \bmod p$, look up $\mu_{x_1}, \ldots, \mu_{x_n}$ in the row for p, compute in NC1 by Theorem 3

$$\mu \equiv \sum_{j=1}^{n} \mu_{x_j} \bmod p - 1 \,,$$

and the required integer is the index of the entry in the row for p containing μ. The implied table look-up is clearly in NC1. □

The following theorem due to Borodin is from [3].

Theorem 5 *NC1 is contained in $O(\log n)$ space.*

2 Chinese remainder representation

Preliminaries

For background information on chinese remainder representation, often called multi-residue arithmetic see [14, 8]. The results that CRR rank and integer comparison can be computed in NC1 were reported in [6]. A chinese remainder representation (CRR) is based

on a set $m_1, \ldots, m_n$ of pairwise coprime integers. The set $m_1, \ldots, m_n$ is called the CRR base and each m_i is called a modulus. We will denote this system by CRR($\mathcal{M}$). Let $M = m_1 \cdots m_n$. By the chinese remainder theorem, every integer $0 \leq x < M$ is uniquely represented by its CRR namely $(x_1, \ldots, x_n)$, where $0 \leq x_i < m_i$ and $x_i \equiv x \bmod m_i$.

It is evident that subject to wrap-around, i.e., where a result equals or exceeds M, or goes negative, addition subtraction and multiplication in CRR are inherently parallelised. That is the CRR of $x \circ y$ is $(z_1, \ldots, z_n)$ where $z_i \equiv x_i \circ y_i \bmod m_i$ and $\circ$ is addition, subtraction or multiplication. Note that in the event $\circ$ is subtraction and $x \geq y$ there is no problem. In particular, if $z_i = x_i - y_i$ is negative, we replace it with $m_i + z_i$.

The main obstacle to using CRR as a vehicle for parallel arithmetic has been the difficulty of efficiently implementing comparison. The approach in [6] has largely overcome that difficulty and is also the key to log-depth division. The second subsection is devoted to a CRR-based comparison algorithm. The main ingredient is the CRR concept of rank.

We proceed to develop more basic notation and ideas. Define $M_i = M/m_i$ and ν_i by $0 \leq \nu_i < m_i$ and $\nu_i \equiv x \cdot M_i^{-1} \bmod m_i$. Let $(x_1, \ldots, x_n)$ be the CRR of x. Note that since

$$\sum_{i=1}^{n} M_i \cdot \nu_i \equiv x \bmod m_i \ ,$$

by the chinese remainder theorem,

$$\sum_{i=1}^{n} M_i \cdot \nu_i = \rho(x, M) \cdot M + x \ . \tag{1}$$

Observe that the integer $\rho(x, M)$ satisfies $0 \leq \rho(x, M) < n$ since $M_i \cdot \nu_i < M$. This integer $\rho(x, M)$ is called the *rank* $\rho(x, \mathcal{M})$ of integer x w.r.t. CRR($\mathcal{M}$). If the modulus product M is clear from context we will usually just write $\rho(x)$ instead of $\rho(x, M)$. The rank has long been identified as an important parameter in CRR research. An alternative formulation of Eq. 1 will prove very useful.

$$\sum_{i=1}^{n} \nu_i / m_i = \rho(x, M) + x/M \ . \tag{2}$$

We reserve CRR($\mathcal{M}$) to denote the system based on the n consecutive primes $3 = m_1 < \cdots < m_n$ and reserve M to be $M = m_1 \cdots m_n$.

Theorem 6 *The CRR($\mathcal{M}$) can uniquely represent every integer below 2^n and a base can be computed in $O(\log n)$ space.*

Proof : A weak version of the prime number theorem implies that for sufficiently large x, $\pi(x) \geq x/\ln x$, where $\pi(x)$ is the number of primes below x. It is an easy consequence of this theorem and a simple underestimate based on Stirling's approximation for factorials that for x sufficiently large, the product Π_{2^x} of all primes up to 2^x satisfies

$$\Pi_{2^x} \geq 2^{2^{x-1}} \ .$$

The first claim of the lemma follows from the lower bound on Π_{2^x} by letting $x = 1+\log n$ and noting that the product of the n consecutive primes greater than 3 is at least as large as the product of all of the primes below $2^x = n$. The second claim follows directly from the prime number theorem because we have $m_n < x$ where x satisfies $x/\ln x \leq n+1$ ($n+1$ because we must count the omitted prime 2.) This yields $x = O(n \log n)$, and all primes bounded above in this way can be produced in $O(\log n)$ space. □

Theorem 7 *An integer $x < 2^n$ in binary notation can be converted into its CRR in CRR($\mathcal{M}$) in NC1.*

Proof : Let $x = y_{n-1} \cdot 2^{n-1} + \cdots + y_0$, where $y_0, \ldots, y_{n-1} \in \{0, 1\}$. Clearly,

$$x_i \equiv \sum_{j=0}^{n-1} 2^j \cdot y_j \bmod m_i \ .$$

The theorem follows from Theorem 3 and Theorem 4. □

The next result is the main motivation for studying rank.

Lemma 1 *If rank can be computed in NC1, the the CRR for $x \bmod m$ can be computed in NC1 from the CRR for x in CRR($\mathcal{M}$) when $m < n$.*

Proof : Observe that $0 \leq \rho(x, \mathcal{M}) < n$. From the definition of the rank $\rho(x, \mathcal{M})$, Eq. 1 we have

$$x \equiv \sum_{i=1}^{n} x_i \cdot \nu_i - \rho(x, \mathcal{M}) \cdot M \bmod m \ .$$

A table $S(m, i, j)$ can be constructed in $O(\log n)$ space where, $i = 1, \ldots, n$, $j = 0, \ldots, m_i - 1$ and $S(m, i, j) = (j \cdot M_i^{-1} \bmod m_i) \bmod m$. Notice that $S(m, i, x_i) = \nu_i \bmod m$. The summation $\sum_{i=1}^{n} x_i \cdot \nu_i \bmod m$ can be computed in NC1 from the foregoing facts and Theorem 3. It is clear that a table with the entries $(k \bmod m, M \bmod m)$, for $1 \leq k, m < n$ can be computed in $O(\log n)$ space. Assuming that the rank $k = \rho(x, \mathcal{M})$ can be computed in NC1, and using the tables just described the CRR for $x \bmod m$ can be computed in NC1. □

Computing rank in NC1

That rank is in NC1 is proved in [6]. See Lemma 2.5 there. We reproduce a demonstration here for the sake of completeness. Throughout the remainder of the paper we adopt the notation from Tanaka and Szabo [14] that $|x|_m$ denotes the residue of x modulo m in the range $0, \ldots, m-1$. We make an exception to this notation which should not produce confusion. Given a CRR system CRR($\mathcal{A}$) the product of whose moduli is A, let $|x|_A$ denote both $x \bmod A$ and the CRR of x in CRR($\mathcal{A}$).

Let q be the least positive integer such that

$$\frac{n}{2^q} < \frac{1}{4} \ . \tag{3}$$

For $1 \leq j \leq n$, let σ_j be the truncation of the binary expansion of ν_j/m_j to the first q bits. We define $\sigma = \sum_{j=1}^{n} \sigma_j$. When necessary we will write things like $\sigma(y)$ to indicate that a quantity depends on the CRR of an integer y.

We let $\langle \alpha \rangle = \alpha - \lfloor \alpha \rfloor$ for real α.

Lemma 2 *The quantities $\sigma_1, \ldots, \sigma_n, \sigma, \lfloor \sigma \rfloor$ and $\langle \sigma \rangle$ can all be computed in NC1 from CRR data.*

Proof : Since $q = O(\log n)$ the required q-bit truncated binary expansion σ_i of ν_i/m_i can certainly be computed in $O(\log n)$ space. A table similar to the one described in the proof of Lemma 1 can be used to hold these expansions. Lemma 3 shows that σ, $\lfloor \sigma \rfloor$ and $\langle \sigma \rangle$ can be computed from $\sigma_1, \ldots, \sigma_n$ in NC1. □

Eq. 2, Eq. 3 and the triangle inequality yield

$$0 \leq \sum_{j=1}^{n} (\frac{\nu_j}{m_j} - \sigma_j) \leq n/2^q < 1/4 \; . \tag{4}$$

Lemma 3 *The following hold.*

1. *If $\langle \sigma \rangle \leq 3/4$, $\lfloor \sigma \rfloor = \rho(x)$.*
2. *If $1/4 \leq x/M \leq 3/4$, $\langle \sigma \rangle \leq 3/4$ and $\rho(x) = \lfloor \sigma \rfloor$.*
3. *If $2 \cdot x > M$, $\rho(x) = \lfloor \sigma \rfloor$.*

Proof : We treat the first item. Using Eq. 2 and Eq. 4, we can write

$$\sum_{j=1}^{n} \nu_j/m_j = \lfloor \sigma \rfloor + \langle \sigma \rangle + \alpha \; ,$$

where $0 \leq \alpha < 1/4$. This makes it plain that either $\rho(x) = \lfloor \sigma \rfloor$ or $\lfloor \sigma \rfloor + 1$. The first item now follows from Eq. 2.

Both items two and three follow directly from the same set-up as the proof of item one. This should be evident from

$$\langle \sigma \rangle = \rho(x) - \lfloor \sigma \rfloor + x/M - \alpha \; ,$$

and the fact that $\rho(x)$ is either $\lfloor \sigma \rfloor$ or $\lfloor \sigma \rfloor + 1$. □

Next, we define a binary tree T_n. It has n leaves and is constructed bottom-up like a Huffman tree, pairing nodes from left to right. Here is the pattern for $n = 10$.

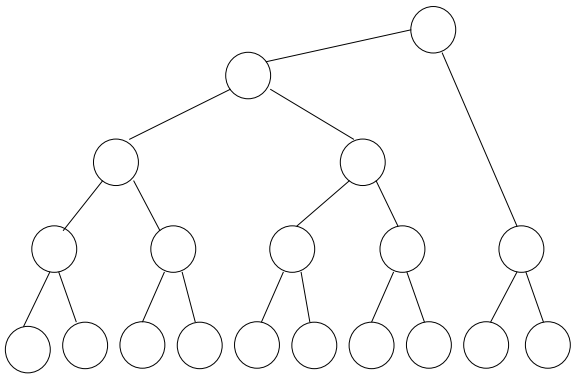

Figure 1: T_{10}

The leaves of T_n are labeled by ordered pairs of bits and an interior node receives two of these pairs as inputs and produces a pair as output which labels the parent node according to the following rules. Let (a_1, a_2) and (b_1, b_2) be the left and right input pairs to an interior node.

$$\begin{array}{ll} (1, a_2) & \text{if } a_1 = 1 \\ (1, b_2) & \text{if } a_1 = 0 \text{ and } b_1 = 1 \\ (0, 0) & \text{otherwise .} \end{array}$$

The leaves are labeled $0, \ldots, n-1$ from left to right and the k-th leaf receives the label

$$\begin{array}{ll} (0, 0) & \text{if } \langle \sigma(|2^k \cdot x|_M) \rangle > 3/4 \\ (1, ||2^k \cdot x|_M|_2) & \text{if } \langle \sigma(|2^k \cdot x|_M) \rangle \leq 3/4 \ . \end{array}$$

The next result was left implicit in [6], but we make it explicit here.

Lemma 4 *The leaves of T_n and the value at its root can be computed in NC1 from the CRR of x.*

Proof : Given values at the leaves it is evident that T_n is just a very simple NC1 circuit. By Lemma 4, each $|2^k|_M$ (the CRR of 2^k) can be computed in NC1. These values could also be handled through precomputation. By Lemma 3 each $|2^k \cdot x|_M$ can be computed in NC1 from $|2^k|_M$ and $|x|_M$. By Lemma 2 $\langle \sigma(|2^k \cdot x|_M) \rangle$ can be computed in NC1. The comparisons are clearly in NC1. Finally, by item 1, Lemma 3, and Lemma 1, $||2^k \cdot x|_M|_2$ can be computed in NC1. □

The key point about T_n is the following result whose proof we take from [6].

Lemma 5 *Assume $\langle \sigma(x) \rangle > 3/4$. The root of T_n receives the label $(1, 0)$ iff $2 \cdot x < M$.*

Proof : For purposes of reference we number the paragraphs in the proof.

P1.
First we show that if the root of T_n gets the label $(1, 0)$, $2 \cdot x < M$. By our assumption that $\langle \sigma \rangle > 3/4$ and Lemma 3, item 2 we know that either $x/M < 1/4$ or $3/4 < x/M$.

P2.
We first look at the case $x/M < 1/4$. There must be an integer h, $2 \leq h < n$ such that

$$M/2^{h+1} < x < M/2^h \ .$$

We see that

$$1/4 < 2^{h-1} \cdot x < 1/2 \ .$$

This certainly means that Lemma 3, item 2 implies

$$\langle \sigma(2^{h-1} \cdot x) \rangle \leq 3/4 \ ,$$

which means that the $h-1$-st leaf gets the label $(1, |2^{h-1} \cdot x|_2)$ and since $|2^{h-1} \cdot x|_M = 2^{h-1} \cdot x$, this label is $(1,0)$. Notice that if $k < h-1$ and $\langle \sigma(2^k \cdot x) \rangle \leq 3/4$, then the k-th leaf gets the label $(1,0)$. All leaves with index $k < h-1$ and $\langle \sigma \rangle > 3/4$ will get $(0,0)$ as label. By the rules for processing at a node it is easy to see that $(1,0)$ will be propagated to the root.

P3.
We have the case $x/M > 3/4$. Let $x = M - y$ so that $y/M < 1/4$. Note that $|2^k \cdot x|_M = |M - 2^k \cdot y|_M$. Using the argument of paragraph P2 there must be k such that

$$1/4 < 2^k \cdot y < 1/2 \ .$$

Thus

$$M - 2^k \cdot y = |M - 2^k \cdot y|_M = |2^k \cdot x|_M \ .$$

This implies that $1/2 < |2^k \cdot x|_M / M < 3/4$ so that Lemma 3, item 2 implies $\langle \sigma(|2^k \cdot x|_M) \rangle \leq 3/4$. This means that the k-th leaf gets $(1,1)$ as label since $|M - 2^k \cdot y|_2 = 1$. Now, $(1,1)$ propagates to the root in the same way as $(1,0)$.

P4.
Now we show that if $2 \cdot x < M$, the root of T_n gets the label $(1,0)$. By Lemma 3, item 2 we must have $x/M < 1/4$. Using the argument of paragraph P2 we see that the root of T_n gets the label $(1,0)$. □

Lemma 5 can be used to construct an NC1 algorithm to decide from the CRR for x whether $2 \cdot x < M$. It is clear that T_n is an NC1 circuit. It will be convenient, though not essential to assume that $(M-1)/2$ is odd. Let H be the least even integer such that $2 \cdot H > M$.

Here is the algorithm MIDWAY to decide whether $2 \cdot x < M$.

1. If $x = (M-1)/2$, EXIT YES.
2. If $\langle \sigma \rangle > 3/4$ GOTO 9.
3. Compute $a = |\sum_{j=1}^{n} \nu_j(x) \cdot M_j + \lfloor \sigma \rfloor \cdot M|_2$.
4. Let $y = |x + H|_M$.
5. If $\langle \sigma(y) \rangle > 3/4$ GOTO 8.

6. Compute $b = |\sum_{j=1}^{n} \nu_j(y) \cdot M_j + \lfloor \sigma(y) \rfloor \cdot M|_2$.

7. If $a = b$, EXIT YES, ELSE EXIT NO.

8. If the label of the root of T_n for y is not $(1,0)$, EXIT YES, ELSE EXIT NO.

9. If the label of the root of T_n for x is $(1,0)$, EXIT YES, ELSE EXIT NO.

Lemma 6 *Algorithm MIDWAY is a correct, NC1 algorithm.*

Proof : By Theorem 3 and Lemma 2, all of the steps of MIDWAY are in NC1. If $x \neq U$, $2 \cdot x < M$ iff $x + H < M$. By Lemma 3, item 1 a in step 3 is the parity of x and b in step 6 is the parity of y. In step 7, if $2 \cdot x < M$, i.e., $x + H < M$, it is clear that $a = b$. Recall here that H is even. If $2 \cdot x > M$, i.e., $x + H > M$, it is also clear that $a \neq b$. Step 8 is correct by Lemma 5 and the fact that $2 \cdot y < M$ iff $2 \cdot x \geq M$. Step 9 is correct by Lemma 5. □

Now we give an NC1 algorithm RANK to compute the rank $\rho(x)$.

1. If $\langle \sigma \rangle \leq 3/4$, EXIT($\lfloor \sigma \rfloor$).

2. If $2 \cdot x > M$, EXIT($\lfloor \sigma \rfloor$).

3. Compute $a = |\sum_{j=1}^{n} \nu_j \cdot M_j + \lfloor \sigma \rfloor \cdot M|_2$.

4. If $a = |y|_2$, EXIT($\lfloor \sigma \rfloor + 1$), ELSE EXIT($\lfloor \sigma \rfloor$).

Theorem 8 *Rank is in NC1.*

Proof : The steps in RANK are certainly in NC1 by Lemma 6, Lemma 2 and Theorem 3.

Step 1 is correct by Lemma 3, item 1. Step 2 is correct by Lemma 3, item 3. Step 4 is correct because $|x|_2 = ||y|_2 + 1|_2$. By Lemma 3, item 3, $|y|_2$ is in fact computed in Step 3. Now if $\rho(x) = \lfloor \sigma \rfloor$, $a = |x|_2$, so $a \neq |y|_2$, while if $\rho(x) = \lfloor \sigma \rfloor + 1$, $a = |y|_2$. □

The rank can be used to compute integer comparison in NC1. This is proved in [6], but we give a proof here.

Theorem 9 *The integer comparison relation between x and y can be computed in NC1 from the CRRs of x and y in CRR($\mathcal{M}$).*

Proof : Assume that all integers are in CRR($\mathcal{M}$). Note that $|2^{-1}|_M$ exists and can be computed in NC1 by Theorem 4. Also, $|M|_2$ can be computed in NC1 by Lemma 1. These facts imply that

$$H = \lfloor M/2 \rfloor = |(M-1) \cdot |2^{-1}|_M|_M$$

can be computed in NC1. Now, $x \leq H$ iff $||2 \cdot x|_M|_2 = 0$. By Lemma 1 and Theorem 8, this test can be carried out in NC1.

If either $x > H$ and $y > H$, or $x \leq H$ and $y \leq H$, $x > y$ iff $||x - y|_M|_2 = ||x|_2 - |y|_2|_2$. These tests are in NC1 by Lemma 1 and Theorem 8. Finally, if $x \leq H$ and $y > H$, $x < y$ and if $x > H$ and $y \leq H$, $y < x$. Again, all tests are in NC1. □

3 The NC1 division algorithm

In this section we give an NC1 algorithm for division. By Theorem 7, we can assume that the input integers x and y are in CRR. There are four major steps: CRR extension, CRR scaling, CRR division and CRR to binary conversion. All CRR bases consist exclusively of primes.

CRR base extension

Let CRR($\mathcal{A}$) be based on $a_1, \ldots, a_r$ and CRR($\mathcal{B}$) be based on $b_1, \ldots, b_s$, such that the two bases are disjoint. Let CRR($\mathcal{C}$) be based on $a_1, \ldots, a_r, b_1, \ldots, b_s$. Let $A = a_1 \cdots a_r$ and let $x_1, \ldots, x_r$ be the CRR in CRR($\mathcal{A}$) of some $x < A$. The CRR $x_1, \ldots, x_r, y_1, \ldots, y_s$ in CRR($\mathcal{C}$) is said to be the base extension of $x_1, \ldots, x_r$ in CRR($\mathcal{A}$) provided that $x \equiv y_i \bmod b_i$ for $i = 1, \ldots, s$.

Lemma 7 *Retaining all of the notation from the definition of the base extension problem, let $c = \max\{a_1, \ldots, a_r, b_1, \ldots, b_s\}$. The base extension problem is in NC1 in terms of c.*

Proof : We extend $x_1, \ldots, x_r$ in CRR($\mathcal{A}$) to CRR($\mathcal{C}$). The process applies equally to the extension from CRR($\mathcal{B}$). By Lemma 1 and Theorem 8, $y_1, \ldots, y_s$ can be computed in NC1 in terms of c. □

CRR scaling

Let CRR($\mathcal{A}$) be based on $a_1, \ldots, a_r$ where $A = a_1 \cdots a_r$, $x < A$ and $x_1, \ldots, x_r$ is the CRR of x. Let $\{b_1, \ldots, b_s\}$ be a nonempty subset of the base and let $B = b_1 \cdots b_s$. The computation of the CRR of $\lfloor x/B \rfloor$ from $x_1, \ldots, x_r$ is called the CRR *scaling problem.*

We give the proof of the next theorem in exacting detail because it is the most important CRR fact needed to show that division is in NC1.

Theorem 10 *We retain all of the notation from the definition of the scaling problem. Let $a = \max\{a_1, \ldots, a_r\}$. The CRR scaling problem is in NC1 in terms of a.*

Proof :

Step 1. It will be convenient to order the base of CRR($\mathcal{A}$) as a list so that $a_1, \ldots, a_r = b_1, \ldots, b_s, c_1, \ldots, c_q$. Note that $r = s + q$ and $s < r$. Let $C = c_1 \cdots c_q$ and let CRR($\mathcal{C}$) denote the system with base $c_1, \ldots, c_q$.

Observe that $|x|_B$ is represented by $(x_1, \ldots, x_s)$ in CRR($\mathcal{B}$). This CRR can be obtained from $x_1, \ldots, x_r$ by simply deleting $x_{s+1}, \ldots, x_r$ and so can clearly be computed in NC1 in terms of a.

Step 2. Next, extend $(x_1, \ldots, x_s)$ in CRR($\mathcal{B}$) to

$$(x_1, \ldots, x_s, ||x|_B|_{c_1}, \ldots, ||x|_B|_{c_q})$$

in CRR($\mathcal{A}$). By Lemma 7 this can be done in NC1 in terms of a.

Step 3. Compute the CRR of $x - |x|_B$ in CRR($\mathcal{A}$). This is explicitly given by

$$(0, \ldots, 0, |x_{s+1} - ||x|_B|_{c_1}|_{c_1}, \ldots, |x_r - ||x|_B|_{c_q}|_{c_q}) \ .$$

This computation is clearly in NC1 in terms of a.

Letting $x = \ell \cdot B + |x|_B$, we have just obtained the CRR of $\ell \cdot B$ in CRR($\mathcal{A}$).

Step 4. Note that B and C are coprime so $|B^{-1}|_C$ exists. In particular, in CRR($\mathcal{C}$) $|B^{-1}|_C$ has the CRR

$$(|B^{c_1-2}|_{c_1}, \ldots, |B^{c_q-2}|_{c_q}) \ ,$$

since $c_1, \ldots, c_q$ are primes. This CRR can be computed in NC1 in terms of a by Theorem 4.

Step 5. From Step 3,

$$(|x_{s+1} - ||x|_B|_{c_1}|_{c_1}, \ldots, |x_r - ||x|_B|_{c_q}|_{c_q})$$

is the CRR of $|\ell \cdot B|_C$ in CRR($\mathcal{C}$). At this point it is essential to note that $\ell < C$, otherwise $x \geq B \cdot C = A$, which is impossible.

Compute the CRR of $|\ell \cdot B|_C \cdot |B^{-1}|_C$ in CRR($\mathcal{C}$), i.e. compute the CRR of $|\ell|_C$ in CRR($\mathcal{C}$), but since $\ell < C$ this is the CRR of ℓ itself. This computation is in NC1 in terms of a.

Step 6. Extend the CRR of ℓ in CRR($\mathcal{C}$) to its CRR in CRR($\mathcal{A}$). This is in NC1 in terms of a by Lemma 7. Notice that $\ell = \lfloor x/B \rfloor$ as required. □

CRR division

The CRR division algorithm described in this subsection is in NC1. It follows the main lines of the Beame-Cook-Hoover approach, but in CRR rather than in binary. The key idea, due to Chiu is to use scaling as the building block for general division. In order to obtain a division algorithm in NC1 we still need to convert from CRR to binary notation. This conversion is discussed in the following subsection.

Let x and y be n-bit integers, i.e., $x, y < 2^n$. Our objective in this subsection is to compute the CRR of $\lfloor x/y \rfloor$ given the CRRs for x and y. More precisely, CRR division is the following problem. Given CRR($\mathcal{M}$) construct a CRR($\mathcal{A}$) such that for any two integers $x, y < 2^n$, the CRR in CRR($\mathcal{A}$) of $\lfloor x/y \rfloor$ is computed from their CRRs in CRR($\mathcal{M}$).

Let α be a positive real. An n-bit *underapproximation* α' to α is a rational such that

$$0 \leq \alpha - \alpha' \leq 1/2^n \ .$$

Lemma 8 *Let $1/2 \leq \alpha < 1$ and $\beta = 1 - \alpha$. If $t_1/A_1, \ldots, t_{n+2}/A_{n+2}$ are $2n$-bit underapproximations to β, then*

$$1 + \frac{t_1}{A_1} + \frac{t_1 \cdot t_2}{A_1 \cdot A_2} + \cdots + \prod_{i=1}^{n+2} \frac{t_i}{A_i}$$

is an n-bit underapproximation to $1/\alpha$.

Proof : Let $T = \beta - 1/2^{2n}$ and let

$$\gamma = 1 + \frac{t_1}{A_1} + \frac{t_1 \cdot t_2}{A_1 \cdot A_2} + \cdots + \prod_{i=1}^{n+2} \frac{t_i}{A_i} \; .$$

Since each t_i/A_i is a $2n$-bit underapproximation to β we have $0 \leq T \leq t_i/A_i$. It follows from this that $T^j \leq \prod_{i=1}^{j} t_i/A_i$, for $j = 1, \ldots, n+2$. In turn this yields the inequalities

$$1 + T + \cdots + T^{n+2} \leq \gamma \leq \sum_{k=0}^{\infty} \beta^k = 1/\alpha \; .$$

We can write

$$0 \leq 1/\alpha - \gamma \leq \sum_{k=0}^{\infty} \beta^k - (1 + T + \cdots + T^{n+2}) \; .$$

It is straightforward to show that for n sufficiently large and for $k = 1, \ldots, n+2$

$$\beta^k - T^k = \beta^k - (\beta - 1/2^{2n})^k \leq k/2^{2n} \; ,$$

and since $\beta \leq 1/2$

$$\sum_{k=n+3}^{\infty} \beta^k \leq 1/2^{n+2} \; .$$

It follows from these inequalities that

$$1/\alpha - \gamma \leq 2^{-2n} \cdot \sum_{k=1}^{n+2} k + 1/2^{n+2} \; ,$$

so that for n sufficiently large

$$1/\alpha - \gamma \leq 1/2^n \; .$$

□

We describe an algorithm for CRR division which is based on scaling. We retain the notation from the definition of the CRR division problem.

We describe the main steps without going into details about their CRR realisations. The details will be covered in the proof of Theorem 11. The motivation for these steps comes from Lemma 8.

1. Let $N = 2 \cdot n^2 + 3n + 1$. Compute $a_1, \ldots, a_N$ which are the N consecutive primes greater than 3. Let CRR($\mathcal{A}$) have $a_1, \ldots, a_N$ as its base and let $A = a_1 \cdots a_N$.
2. Extend the CRRs of x and y in CRR($\mathcal{M}$) to CRR($\mathcal{A}$). All subsequent steps are performed in CRR($\mathcal{A}$).

3. Compute an integer D such that $1/2 \leq y/D < 1$. We let $\alpha = y/D$ and $\beta = 1 - \alpha$.

4. Compute $t_1/A_1, \ldots, t_{n+2}/A_{n+2}$ as $2n$-bit underapproximations to β.

5. Compute an integer N such that

$$\gamma = \frac{N}{A_1 \cdots A_{n+2}} ,$$

where γ is the n-bit underapproximation to $1/\alpha$ of Lemma 8. We write $\gamma = 1/\alpha - \epsilon$, where $0 \leq \epsilon \leq 1/2^n$.

6. Note that $1/\alpha = D/y$, so $\gamma = D/y - \epsilon$. That is

$$x \cdot \gamma/D = \frac{x}{y} - \frac{x \cdot \epsilon}{D} .$$

Since $0 \leq x\epsilon < 1$ and $D \geq 1$,

$$x \cdot \gamma/D = \frac{x}{y} - \delta ,$$

where $0 \leq \delta < 1$.

7. Compute

$$\ell = \lfloor x \cdot \gamma/D \rfloor = \lfloor \frac{x \cdot N}{D \cdot A_1 \cdots A_{n+2}} \rfloor .$$

8. Let ℓ be the result of the computation in the previous step. If $x - \ell \cdot y < y$, decide that $\ell = \lfloor x/y \rfloor$, otherwise $\ell + 1 = \lfloor x/y \rfloor$.

Theorem 11 *CRR division is in NC1.*

Proof : We examine each of the steps of the algorithm.

Step 1. This step, which is a precomputation, i.e., it is done once for all n-bit divisions. requires a straightforward generalisation of Theorem 6.

Step 2. This step is in NC1 by Lemma 7. We remind the reader that the phrase 'computation of X' means the computation of the CRR of X in CRR($\mathcal{A}$).

Step 3. If $y = 2$, let $D = 2^2$. If $y > 2$ proceed as follows. Find $j < n$ such that $m_1 \cdots m_j \leq y < m_1 \cdots m_j \cdot m_{j+1}$. We use Theorem 4 to compute $m_1, m_1 \cdot m_2, \ldots, m_1 \cdots m_{n-1}$ in parallel in NC1. Each of these is compared to y in parallel. We can use a simple binary tree subcircuit to find the index j. The tree has depth $O(\log n)$.

Find the least integer k such that $y < 2^k \cdot m_1 \cdots m_j$. This is in NC1 by Theorem 9 and Theorem 4. Note that

$$1/2 \leq \frac{y}{2^k \cdot m_1 \cdots m_j} < 1 .$$

We let $D = 2^k \cdot m_1 \cdots m_j$.

Step 4. For $i = 1, \ldots, n+2$, define A_i by

$$A_i = a_{n+2(i-1)\cdot n+1} \cdots a_{n+2i\cdot n} \ .$$

Note that $A_{n+2} = a_{2n^2+2n+1} \cdots a_{2n^2+3n}$. By Theorem 4, $A_1, \ldots, A_{n+2}$ are computable in NC1. Define t_i by

$$t_i = \lfloor \frac{(D-y)\cdot A_i}{D} \rfloor \ .$$

We show that $t_1, \ldots, t_{n+2}$ can be computed in NC1.

Let $D = 2^k \cdot E$. Notice that $2^k \leq 2 \cdot m_{j+1}$. First we deal with the divisor 2^k. Compute

$$|(D-y)\cdot A_i|_{2^k}$$

in NC1 by Lemma 1. Now compute

$$U = \lfloor \frac{(D-y)\cdot A_i}{2^k} \rfloor = ((D-y)\cdot A_i - |(D-y)\cdot A_i|_{2^k}) \cdot |2^{-k}|_A$$

in NC1 by Theorem 4. We can write

$$(D-y)\cdot A_i = 2^k \cdot U + |(D-y)\cdot A_i|_{2^k} \ .$$

From this we see that

$$t_i = \lfloor \frac{U}{E} + \frac{|(D-y)\cdot A_i|_{2^k}}{D} \rfloor \ .$$

Now we can use scaling to deal with the factor of E in the divisor. Compute $\lfloor U/E \rfloor$ in NC1 by Theorem 10. It is clear that

$$\frac{|(D-y)\cdot A_i|_{2^k}}{D} < 1 \ ,$$

so either $t_i = \lfloor U/E \rfloor$, or $t_i = \lfloor U/E \rfloor + 1$.

If $(D-y)\cdot A_i - \lfloor U/E \rfloor \cdot D < D$, $t_i = \lfloor U/E \rfloor$, otherwise $t_i = \lfloor U/E \rfloor + 1$. This test can be computed in NC1 by Theorem 9.

Notice that t_i/A_i is a $2n$-bit under approximation to $\beta = 1 - \alpha = 1 - y/D$ since

$$\frac{t_i}{A_i} = 1 - y/D - \epsilon/A_i \ ,$$

where $0 \leq \epsilon < 1$, and $A_i > 2^{2n}$ by Theorem 6.

Step 5. By Lemma 8,

$$\gamma = 1 + \frac{t_1}{A_1} + \cdots + \frac{t_1 \cdots t_{n+2}}{A_1 \cdots A_{n+2}}$$

is an n-bit under approximation of $1/\alpha$. We can write

$$\gamma = \frac{N}{A_1 \cdots A_{n+2}} \ ,$$

where

$$N = A_1 \cdots A_{n+2} + t_1 \cdot A_2 \cdots A_{n+2} + t_1 \cdot t_2 \cdot A_3 \cdots A_{n+2} + \cdots + t_1 \cdots t_{n+2} \ .$$

By Theorem 3 and Theorem 4 we can compute N and $A_1 \cdots A_{n+2}$.

Step 6. No comment needed.

Step 7. This step can be carried out in the same manner as Step 4. We point out, recalling from Step 4 that $D = 2^k \cdot E$ and $E \cdot A_1 \cdots A_{n+2}$, is a product of moduli so the use of scaling is valid.

Step 8. By the observation made about Step 6 in the description of the algorithm,

$$\ell = \lfloor \frac{x}{y} - \delta \rfloor \ ,$$

where $0 \leq \delta < 1$. Thus, either $\ell = \lfloor x/y \rfloor$ or $\lfloor x/y \rfloor - 1$. This establishes the validity of the test in this step and by Theorem 9, its computation is clearly in NC1. □

CRR to binary

Theorem 12 *Conversion from CRR to binary is in NC1.*

Proof : If $x = y_{n-1} \cdot 2^{n-1} + \cdots y_0$ is the binary expansion of x, then clearly

$$y_i = \lfloor x/2^i \rfloor - 2 \cdot \lfloor x/2^{i+1} \rfloor \ .$$

The computation of the powers of 2 is in NC1 by Theorem 4 and the divisions are in NC1 by Theorem 11. □

Completion of the proof of Theorem 1

Proof : Integer division is in NC1 because each of the following steps is in NC1.

- Conversion from binary notation to CRR. See Theorem 7.
- CRR division. See Theorem 11.
- Conversion from CRR to binary notation. See Theorem 12.

□

4 Remarks

It appears likely, based on material in [4] that integer division can be computed by the apparently more restrictive U_E uniform $O(\log n)$ depth circuits. See [15] for a definition

of this kind of uniformity. However, we do not pursue this here because it seems to us that the most important consequence of an NC1 division algorithm is that iterated integer product is in $O(\log n)$ space. Since it is known that the computation of the coefficients of a context-free grammar generating series is NC1-reducible to integer division, it follows that this problem is also in NC1. See [2, 11].

References

[1] P. Beame, S. Cook, and H. Hoover. Log depth circuits for division and related problems. *SIAM J. Comp.*, 15,4:994–1003, 1986.

[2] A. Bertoni, M. Goldwurm, and P. Massazza. Counting problems and algebraic formal power series in noncommuting variables. *Inf. Proc. Lett.*, 34:117–121, 1990.

[3] A. Borodin. On relating time and space to size and depth. *SIAM J. Comp.*, 6:733–744, 1977.

[4] Andrew Chiu. Complexity of parallel arithmetic using the chinese remainder representation. Master's thesis, U. Wisconsin-Milwaukee, 1995. G. Davida, supervisor.

[5] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[6] G. Davida and B. Litow. Fast parallel arithmetic via modular representation. *SIAM J. Comp.*, 20,4:756–765, 1991.

[7] M. A. Hitz and E. Kaltofen. Integer division in residue number systems. *IEEE Trans. Computers*, 44(8):983–989, 1995.

[8] D. Knuth. *The Art of Computer Programming.* Addison-Wesley, 1969.

[9] C. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theor. Comp. Sci.*, 71:95–132, 1990.

[10] B. Litow. On iterated integer product. *Inf. Proc. Lett.*, 42,5:269–272, 1992.

[11] B. Litow. Computing context-free grammar generating series. in press, Inf. and Comp., 2000.

[12] J. Reif. Logarithmic depth circuits for algebraic functions. *SIAM J. Comput.*, 15:231–242, 1986.

[13] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

[14] R. Tanaka and N. Szabo. *Residue Arithmetic and its Application to Computer Technology.* McGraw-Hill, 1968.

[15] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.