DIVISION IN LOGSPACE-UNIFORM \text{NC}^1

\textbf{ANDREW CHIU}^1, \textbf{GEORGE DAVIDA}^1 \text{ AND BRUCE LITOW}^2

\textbf{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 logspace-uniform. In this paper we describe log-depth, polynomial size, logspace-uniform, \textit{i.e.}, \text{NC}^1 circuit family for integer division. In particular, by a well-known result this shows that division is in logspace. We also refine the method of the paper to show that division is in \text{dlogtime-uniform NC}^1.

\textbf{Mathematics Subject Classification}. 68Q05, 68Q10, 68Q15, 68Q17.

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 logspace-uniform family. See [20] for excellent coverage on these circuits. Notice that any circuit family

\hspace{1cm} Keywords and phrases: Parallel complexity, \text{NC}, integer division, uniformity.

\hspace{1cm} 1 EECS Department, University of Wisconsin-Milwaukee, Milwaukee, WI, U.S.A.;
\hspace{1cm} e-mail: davidascsuwm.edu

\hspace{1cm} 2 School of Information Technology, James Cook University, Townsville, Qld. 4811, Australia;
\hspace{1cm} e-mail: brucecsjcu.edu.au

\copyright EDP Sciences 2001
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 logspace-uniform, polynomial size, \( O(\log^2 n) \) depth circuits exist for division (see [20]). The question of the existence of \( \log n \) depth circuits was affirmatively resolved by Beame et al. in [4]. Unfortunately, their division circuits are not logspace-uniform, although they can be produced in P.

Following the Beame, Cook, Hoover result, Davida and Litow showed how to compute integer division by log-depth, polynomial size Boolean circuits in [8]. However, these circuits are not logspace-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 [16]). 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 [10]. Their division circuits also fail to be logspace-uniform.

In this paper we demonstrate, using a Chinese remainder approach that division can be computed by logspace-uniform, \( \log n \) depth circuits. Formally, we will prove:

**Theorem 1.1.** Integer division is in logspace-uniform \( \text{NC}^1 \).

Combining this result with Theorem 1.2 and Theorem 1.5 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 logspace-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 logspace-uniform, polynomial size, \( \log^k n \) depth circuits is called \( \text{NC}^k \), and the union over all \( \text{NC}^k \) is called NC. It is open
whether NC is a hierarchy, i.e., whether there exist $1 < k < k'$ such that $\text{NC}^k$ is properly contained in $\text{NC}^{k'}$. 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 [13]).

We use the variable $n$ to refer to the input size. Thus, when we say that a computation can be done in logarithmic space (or logarithmic depth) we mean that the algorithm uses $O(\log n)$ space (or $O(\log n)$ depth) on inputs of size $n$. Occasionally, we will need to refer to computations on inputs of size $n^{O(1)}$ (such as when we are operating on the representation of an $n$-bit number, when represented in Chinese remainder representation modulo a polynomial number of $O(\log n)$-bit primes). Note that the notions of logarithmic space and depth remain unchanged, regardless of whether the input is of length $n$ or $n^{O(1)}$.

The basic theory of NC and in particular, NC$^1$ and its associated uniformity issues can be found in [4,6,7,17]. 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 [4].

**Theorem 1.2.** Either division, iterated product and powering are all in logspace-uniform NC$^1$, or none is in logspace-uniform NC$^1$.

Throughout the paper, any occurrence of the unmodified term NC$^1$ should be taken to refer to logspace-uniform NC$^1$.

We will need to make use of some standard functions that are known to be in logspace-uniform NC$^1$. For proofs, you can consult standard texts such as [11,19].

**Theorem 1.3.** For fixed $k$, the following operations can be computed in logspace-uniform NC$^1$: (a) adding $n$ numbers, each of $n$ bits, (b) multiplying two $n$-bit numbers, (c) given an $n$-bit number $x$ and a number $m < n^k$, output $x \mod m$.

Thus in particular, given $n + 1$ numbers $m, x_1, \ldots, x_n$ all bounded above by $n^k$, $x_1 + \cdots + x_n \mod m$ can be computed in logspace-uniform NC$^1$.

**Theorem 1.4.** The computation of $x_1 \cdots x_n \mod p^t$ where $x_1, \ldots, x_n < p$, $p^t < n$ and $p$ is prime is in NC$^1$.

**Proof.** This is Theorem 4.2 in [4]. 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., $\{g, g^2, \ldots, g^{p-1}\} \mod p = \{1, \ldots, p - 1\}$. 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 $\mu_k$ such that $k \equiv g^{\mu_k} \mod p$.

To compute $x_1 \cdots x_n \mod p$, look up $\mu x_1, \ldots, \mu x_n$ in the row for $p$, compute in $NC^1$ by Theorem 1.3

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

and the required integer is the index of the entry in the row for $p$ containing $\mu$. The implied table look-up is clearly in $NC^1$.

The following theorem due to Borodin is from [6].

**Theorem 1.5.** $NC^d$ 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 [12,18]. The results that CRR rank and integer comparison can be computed in $NC^1$ were reported in [8]. 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($M$). 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 $x_i \equiv x \mod m_i$.

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 $z_i \equiv x_i \circ y_i \mod m_i$ 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 [8] 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 $\nu_i$ by $0 \leq \nu_i < m_i$ and $\nu_i \equiv x \cdot M_i^{-1} \mod m_i$. Let $(x_1, \ldots, x_n)$ be the CRR of $x$. Note that since

$$\sum_{i=1}^{n} M_i \cdot \nu_i \equiv x \mod m_i,$$
by the Chinese remainder theorem, there is an integer \( \rho(x, M) \) such that
\[\sum_{i=1}^{n} M_i \cdot \nu_i = \rho(x, M) \cdot M + x. \] (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 \( \rho(x, M) \) of integer \( x \) w.r.t. \( \text{CRR}(M) \). If the modulus product \( M \) is clear from context we will usually just write \( x \) instead of \( \rho(x, M) \). The rank has long been identified as an important parameter in CRR research. An alternative formulation of equation (1) will prove very useful.
\[\sum_{i=1}^{n} \nu_i / m_i = \rho(x, M) + x / M. \] (2)

We reserve \( \text{CRR}(M) \) 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 2.1.** The \( \text{CRR}(M) \) can uniquely represent every integer below \( 2^n \) and a base can be computed in \( O(\log n) \) space.

**Proof.** It is a standard fact from number theory that the \( n \)-th prime requires \( O(\log n) \) bits. It is obvious that the product of the first \( n \) primes exceeds \( 2^n \). Finally, it is straightforward to generate the list of the first \( n \) primes in \( O(\log n) \) space.

The next result is well-known, see for example Lemma 4.1 of [4].

**Theorem 2.2.** An integer \( x < 2^n \) in binary notation can be converted into its \( \text{CRR} \) in \( \text{CRR}(M) \) in \( \text{NC}^1 \).

**Proof.** Let \( x = y_{n-1} \cdot 2^{n-1} + \cdots + y_0 \), where \( y_0, \ldots, y_{n-1} \in \{0,1\} \). Clearly,
\[ x_i \equiv \sum_{j=0}^{n-1} 2^j \cdot y_j \mod m_i. \]

The theorem follows from Theorem 1.3 and Theorem 1.4.

The next result is the main motivation for studying rank.

**Lemma 2.1.** If rank can be computed in \( \text{NC}^3 \), then the \( \text{CRR} \) for \( x \mod m \) can be computed in \( \text{NC}^4 \) from the \( \text{CRR} \) for \( x \) in \( \text{CRR}(M) \) when \( m < n \).

**Proof.** Observe that \( 0 \leq \rho(x, M) < n \). From the definition of the rank \( \rho(x, M) \), equation (1) we have
\[ x \equiv \sum_{i=1}^{n} x_i \cdot \nu_i - \rho(x, M) \cdot M \mod m. \]
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 $S(m, i, j) = (j \cdot M_i^{-1} \mod m_i) \mod m$. Notice that $S(m, i, x_i) = \nu_i \mod m$. The summation $\sum_{i=1}^{n} x_i \cdot \nu_i \mod m$ can be computed in NC$^1$ from the foregoing facts and Theorem 1.3. It is clear that a table with the entries $(k \mod m, M \mod m)$, for $1 \leq k, m < n$ can be computed in O($\log n$) space. Assuming that the rank $k = \rho(x, M)$ can be computed in NC$^1$, and using the tables just described the CRR for $x \mod m$ can be computed in NC$^1$.

Our main application of Lemma 2.1 will be to computing the parity of numbers in CRR in logspace-uniform NC$^1$.

**Computing rank in NC$^1$**

**Theorem 2.3.** Rank can be computed in logspace-uniform NC$^1$.

**Proof.** This was shown in the proof of Lemma 2.5 of [8]. Alternatively, an independent proof that rank is computable in logspace was presented by Macarie in Lemma 2 of [15]. It is observed in [1] that the steps in Macarie’s algorithm can all be implemented in logspace-uniform TC$^0$, and hence in logspace-uniform NC$^1$. □

We can directly apply Theorem 2.3 to show that comparison is in NC$^1$.

**Theorem 2.4.** The integer comparison relation between $x$ and $y$ can be computed in logspace-uniform NC$^1$ from the CRRs of $x$ and $y$ of CRR($M$).

**Proof.** Assume that all integers are in CRR($M$). We let $[2^{-1}M]$ denote the number $a$ such that $2 \cdot a \equiv 1 \mod M$. Since $M$ is odd $[2^{-1}M]$ exists. Its CRR($M$) representation, in fact is just

$$([2^{-1}]m_1, \ldots, [2^{-1}]m_n),$$

which certainly can be computed in logspace-uniform NC$^1$.

Observe that

$$[M/2] = [(M - 1) \cdot [2^{-1}M]_M],$$

and since $M - 1$ in CRR($M$) is

$$(m_1 - 1, \ldots, m_n - 1),$$

it is clear that the CRR of $[M/2]$ can be computed in NC$^1$.

Note that $x \leq [M/2]$ iff $|[2 \cdot x]_M|_2 = 0$. Thus, by Theorem 2.3 and Lemma 2.1, we can decide whether $x \leq [M/2]$ in NC$^1$.

We see that both $x$ and $y$ are greater than $[M/2]$, or both are less than or equal to $[M/2]$, then $x \geq y$ iff $||x - y|_M|_2 = ||x|_2 - |y|_2|_2$. Our foregoing considerations show that this parity test can be carried out in NC$^1$. The remaining possibilities
are $x \leq \lceil M/2 \rceil$ and $y > \lceil M/2 \rceil$, in which case $x < y$, and $x > \lceil M/2 \rceil$ and $y \leq \lceil M/2 \rceil$, in which case $x > y$.

### 3. The NC$^1$ Division Algorithm

In this section we give an NC$^1$ algorithm for division. By Theorem 2.2, 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$(A)$ be based on $a_1, \ldots, a_r$ and CRR$(B)$ be based on $b_1, \ldots, b_s$, such that the two bases are disjoint. Let CRR$(C)$ 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$(A)$ of some $x < A$. The CRR $x_1, \ldots, x_r, y_1, \ldots, y_s$ in CRR$(C)$ is said to be the base extension of $x_1, \ldots, x_r$ in CRR$(A)$ provided that $x \equiv y_i \mod b_i$ for $i = 1, \ldots, s$. The base extension problem is the computation of $x_1, \ldots, x_r, y_1, \ldots, y_s$ in CRR$(C)$.

**Lemma 3.1.** 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 NC$^1$.

**Proof.** We extend $x_1, \ldots, x_r$ in CRR$(A)$ to CRR$(C)$. By Lemma 2.1 and Theorem 2.3, $y_1, \ldots, y_s$ can be computed in NC$^1$.

#### CRR Scaling

Let CRR$(A)$ 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 $[x/B]$ 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 NC$^1$.

**Theorem 3.1.** 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 NC$^1$.

**Proof.**

**Step 1.** It will be convenient to order the base of CRR$(A)$ 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$(C)$ denote the system with base $c_1, \ldots, c_q$.

Observe that $[x]_B$ is represented by $(x_1, \ldots, x_s)$ in CRR$(B)$. 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 NC$^1$. 
Step 2. Next, extend \((x_1, \ldots, x_s)\) in \(\text{CRR}(B)\) to
\[
(x_1, \ldots, x_s, |x_B|_{c_1}, \ldots, |x_B|_{c_q})
\]
in \(\text{CRR}(A)\). By Lemma 3.1 this can be done in \(\text{NC}^1\).

Step 3. Compute the CRR of \(x - |x_B|\) in \(\text{CRR}(A)\). 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 \(\text{NC}^1\).

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

Step 4. Note that \(B\) and \(C\) are coprime so \(|B^{-1}|_C\) exists. In particular, in \(\text{CRR}(C)\) \(|B^{-1}|_C\) 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 \(\text{NC}^1\) by Theorem 1.4.

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 \(\text{CRR}(C)\). 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 \(\text{CRR}(C)\), \(i.e.\) compute the CRR of \(|\ell|_C\) in \(\text{CRR}(C)\), but since \(\ell < C\) this is the CRR of \(\ell\) itself. This computation is in \(\text{NC}^1\).

Step 6. Extend the CRR of \(\ell\) in \(\text{CRR}(C)\) to its CRR in \(\text{CRR}(A)\). This is in \(\text{NC}^1\) by Lemma 3.1. Notice that \(\ell = |x/B|\) as required.

CRR Division

The CRR division algorithm described in this section is in \(\text{NC}^1\). 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 \(\text{NC}^1\) 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 \([x/y]\) given the CRRs for \(x\) and \(y\). More precisely, CRR division is the following problem. Given \(x\) and \(y\) in \(\text{CRR}(M)\), construct the CRR
of $[x/y]$. Actually, we will first compute an extension $A$ of $M$ and compute the CRR of $[x/y]$ in CRR($A$), which immediately yields the desired result.

Let $\alpha$ be a positive real. An $n$-bit underapproximation $\alpha'$ to $\alpha$ is a rational such that

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

**Lemma 3.2.** 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 $\beta$, 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 $1/\alpha$ for $n$ sufficiently large.

**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 $\beta$ we have $0 \leq T \leq t_i/A_i$. It follows from this that $T^2 \leq \prod_{i=1}^{n} t_i/A_i$, for $j = 1, \ldots, n + 2$. In turn this yields the inequalities

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

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

$$1/\alpha - \gamma \leq 2^{-2n} \sum_{k=1}^{n+2} k + 1/2^{n+2},$$
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 3.2. The motivation for these steps comes from Lemma 3.2.

1. Let $N = 3 \cdot n^2$. Compute $a_1, \ldots, a_N$ which are the $N$ consecutive primes greater than 3. Let CRR($A$) have $a_1, \ldots, a_N$ as its base and let $A = a_1 \cdots a_N$. By an easy extension of Theorem 2.1, and equation (3), for $n$ sufficiently large, $A$ is large enough so that all numbers involved in the approximation underlying our division method can be represented in CRR($A$) without wraparound.

2. Extend the CRRs of $x$ and $y$ in CRR($M$) to CRR($A$). All subsequent steps are performed in CRR($A$).

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 $\beta$.

5. Compute an integer $N$ such that

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

where $\gamma$ is the $n$-bit underapproximation to $1/\alpha$ of Lemma 3.2. 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 $\ell$ 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 3.2.** CRR division is in $\text{NC}^3$. 

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 2.1.

**Step 2.** This step is in \(NC^1\) by Lemma 3.1. We remind the reader that the phrase “computation of \(X\)” means the computation of the CRR of \(X\) in \(CRR(A)\).

**Step 3.** If \(y = 2\), let \(D = 4\). 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 1.4 to compute \(m_1, m_1 \cdot m_2, \ldots, m_1 \cdots m_{n-1}\) in parallel in \(NC^1\). 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 \(NC^1\) by Theorem 2.4 and Theorem 1.4. 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

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

Note that \(A_{n+2} = a_{2n^2+2n+1} \cdots a_{2n^2+3n}\). By Theorem 1.4, \(A_1, \ldots, A_{n+2}\) are computable in \(NC^1\). 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 \(NC^1\).

Let \(D = 2^k \cdot E\). Notice that \(2^k \leq 2 \cdot m_{j+1}\). First we deal with the divisor \(2^k\). We observe that since \(A\) and 2 are coprime, there exists a number \(a\) such that \(2^k \cdot a \equiv 1 \mod A\). This number \(a\) can be computed in \(CRR(A)\) by raising 2 to the \(k\)-th power for each modulus.

Compute \(|(D - y) \cdot A_i|_{2^k}\) in \(NC^1\) by Lemma 2.1. 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 NC$^1$ by Theorem 1.4. We can write

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

From this we see that

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

Now we can use scaling to deal with the factor of $E$ in the divisor. Compute $[U/E]$ in NC$^1$ by Theorem 3.1. It is clear that

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

so either $t_i = [U/E]$, or $t_i = [U/E] + 1$.

If $(D - y) \cdot A_i = [U/E] \cdot D < D$, then $t_i = [U/E]$, otherwise $t_i = [U/E] + 1$. This test can be computed in NC$^1$ by Theorem 2.4.

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 2.1.

**Step 5.** By Lemma 3.2,

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

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 1.3 and Theorem 1.4 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,
\[ \ell = \frac{x}{y} - \delta, \]
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 2.4, its computation is clearly in \(\text{NC}^1\).

CRR to binary

Theorem 3.3. Conversion from CRR to binary is in \(\text{NC}^k\).

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 \(\text{NC}^1\) by Theorem 1.4 and the divisions are in \(\text{NC}^1\) by Theorem 3.2.

Completion of the proof of Theorem 1

Proof. Integer division is in \(\text{NC}^1\) because each of the following steps is in \(\text{NC}^1\).
- Conversion from binary notation to CRR. See Theorem 2.2.
- CRR division. See Theorem 3.2.
- Conversion from CRR to binary notation. See Theorem 3.3.

4. Division is in dlogtime-uniform \(\text{NC}^1\)

Preliminaries

In this section we refine the logspace-uniform \(\text{NC}^1\) division circuit family and show that integer division is in fact computable by a dlogtime-uniform \(\text{NC}^1\) circuit family.

A fundamental discussion of dlogtime Turing machines and of dlogtime Boolean circuit uniformity is presented in [3]. A dlogtime Turing machine is deterministic and has a read-only input tape of length \(n\), a finite number of read/write work tapes of total length \(O(\log n)\) and a read/write address tape of length \(O(\log n)\). In one move the machine has access to one bit of input addressed by the contents of the address tape.

Dlogtime-uniform \(\text{NC}^1\) can be defined in terms of the extended connection language ECL of a circuit. An alternative formulation is given in [3]. The ECL of a circuit with \(n\) inputs is the set of tuples \((n, g, p, y)\), where \(g\) is the number of an input or gate, \(p \in \{L, R\}^*\), and \(y\) is the type of gate \(g\) if \(p\) is the empty string;
otherwise $y$ is the number of the gate reached from $g$ by following the **Left-Right** path indicated by $p$. The length of path $p$ is also required to be bounded above by $\log Z(n)$, where $Z(n)$ is the circuit size (see [20]).

A $n^{O(1)}$ size, $O(\log n)$ depth circuit family is said to be $\text{dlogtime}$-uniform if its ECL can be decided by a $\text{dlogtime}$ Turing machine.

It is well-known that tree and array combinations of the kind used in our construction of a logspace-uniform $\text{NC}^1$ division family remain in $\text{dlogtime}$-uniform $\text{NC}^1$. As pointed out by Allender and Barrington, the real issue in achieving $\text{dlogtime}$-uniform $\text{NC}^1$-uniformity in arithmetic problems is the precomputation of tables (see [2]). To guarantee logspace-uniformity it sufficed to show that all requisite tables could be generated in $O(\log n)$ space. Here we will show that the two crucial tables of primes and discrete logarithms can be computed by $\text{dlogtime}$-uniform $\text{NC}^1$ circuits.

**Table constructions**

**Theorem 4.1.** Given $k = O(n/\log n)$, the $k$-th prime can be computed by a $\text{dlogtime}$-uniform $\text{NC}^1$-uniform circuit.

**Proof.**

1. Estimate an upper bound, $C$, on the number of integers to test before finding the $k$-th prime. It is a standard result of number theory that $C = O(n)$.
2. For each $i, j < C$, construct the Boolean valued circuit $\text{DIV}[i, j]$ which returns 1 exactly when $i$ divides $j$.
3. For each $i < C$ construct the Boolean valued circuit $\text{PRI}[i]$ given by
   \[
   \neg \left( \bigvee_{j<i} \text{DIV}[i, j] \right) .
   \]
4. For each $i < C$, compute $i$’s position in the table of primes:
   \[
   \text{POS}[i] = \sum_{j<i} \text{PRI}[j] .
   \]
5. Find $i$ such that $\text{POS}[i] = k$.

This circuit has depth $O(\log n)$ since the steps are either $O(\log n)$ bit arithmetic or $O(\log n)$ depth tree reductions. Clearly, the circuit is in $\text{dlogtime}$-uniform $\text{NC}^1$ since each constituent circuit is in $\text{dlogtime}$-uniform $\text{NC}^1$.

**Theorem 4.2.** Given an integer $m < n$ and integer $h < m$, the discrete logarithm of $h$ modulo $m$ can be computed by a $\text{dlogtime}$-uniform $\text{NC}^1$-uniform circuit family.
Proof. The main difficulty is computing \( i^j \mod m \) in \( O(\log n) \) time. The standard divide and conquer approach takes \( O(\log(n) \cdot \log \log n) \) time, which is too slow.

Assuming that \( i^j \mod m \) can be computed by a dlogtime-uniform NC\(^1\) circuit, it is straightforward to build the discrete logarithm table.

1. For each \( i, j < m \), compute (using our assumption)
   \[
   \text{POW}[i, j] = i^j \mod m.
   \]

2. For each \( i < m \), compute
   \[
   \text{GEN}[i] = \bigvee_{j<m} (\text{POW}[i, j] \neq 1).
   \]

That is, \( \text{GEN}[i] = 1 \) exactly when \( i \) is a generator modulo \( m \).

3. Compute \( g \) to be least such that \( \text{GEN}[g] = 1 \).

4. Compute \( \ell \), where \( h = \text{POW}[g, \ell] \). That is, \( \ell \) is the discrete logarithm of \( h \) modulo \( m \) w.r.t. the generator \( g \).

We turn to the powering needed to build our discrete logarithm tables. We can write \( j < n \) in \( \log n \)-ary notation as

\[
 j = a_r \cdot (\log n)^r + \cdots + a_0 ,
\]

where \( a_0, \ldots, a_r < \log n \) and \( r = O(\log(n)/\log \log(n)) \). If we can compute \( i^a \mod m \) in \( O(\log \log n) \) time for \( a < \log n \), we can adapt the classical approach for computing \( i^j \) (see [12], p. 399) and obtain \( i^j \mod m \) by using \( O(\log(n)/\log \log(n)) \) powerings \( i^a \mod m \). The other operations at each of the six steps involve only arithmetic on \( O(\log n) \) bit numbers. Here is the basic algorithm.

1. \( J = j, Y = 1, Z = i \).
2. \( A = J \mod \log n, J = [J/\log n] \).
3. If \( A = 0 \) goto 6.
4. \( Y = Z^A \cdot Y \mod m \).
5. If \( J = 0 \), exit with result \( Y \).
6. \( Z = Z^\log n \mod m \), goto 2.

Each step involves either arithmetic on \( O(\log n) \) bit numbers, or powering to an exponent of size \( O(\log n) \). The arithmetic, by what has been done in Section 3 requires \( O(\log \log n) \) time. Since \( (\log n)^{\log(n)/\log \log(n)} = n \), the loop is executed at most \( \log(n)/\log \log(n) \) times. Thus, assuming that the powering can be done in \( O(\log \log n) \) time we obtain a dlogtime-uniform circuit.

Recalling Theorem 1.4, a discrete logarithm table for a \( O(\log \log n) \)-bit modulus can be computed by brute force in \( O(\log n) \) time. If these discrete logarithm tables are computed only once, then by Theorem 3.2 and the NC\(^1\) equivalence of division and iterated product, \( O(\log n / \log \log n) \) \( O(\log n) \)-bit iterated products
can be computed in $O(\log n)$ time, i.e. by a dlogtime-uniform NC$^1$ circuit family. This finishes the proof of the theorem.

**DISCUSSION**

This paper has introduced a new CRR technique namely scaling and has applied it to show that integer division and iterated product are in $O(\log n)$ space and in fact in dlogtime-uniform NC$^1$-uniform NC$^1$. Recently Hesse has further refined this result by showing that integer division and so also iterated product can be computed by by a uniform TC$^0$ (constant depth threshold) circuit family (see [9]). It seems unlikely that further restriction on division is possible.

Finally, we point out another consequence of division being computable by a logspace-uniform NC$^1$-uniform circuit family. It is known that the computation of the coefficients of a context-free grammar generating series is logspace-uniform NC$^1$-reducible to integer division, and so it follows that this problem is in logspace (see [5, 14]) for details on the generating series problem.

**REFERENCES**

[2] E. Allender and D.A. Mix Barrington, Uniform circuits for division: Consequences and problems (2000) allender@cs.rutgers.edu, barring@cs.umass.edu

Communicated by R. Baeza-Yates.
Received June 25, 2000. Accepted August 3, 2001.