

LOG DEPTH CIRCUITS FOR DIVISION AND RELATED PROBLEMS

Paul W. Beame
Stephen A. Cook
H. James Hoover

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 1A4

Abstract: We present optimal depth Boolean circuits (depth $O(\log n)$) for integer division, powering, and multiple products. We also show that these three problems are of equivalent uniform depth and space complexity. In addition, we describe an algorithm for testing divisibility that is optimal for both depth and space.

1. Introduction

It is a well known fact that addition, subtraction and multiplication on modern computers are significantly faster operations than division. Circuit designers have been unable to match the efficiency of the circuits for addition and multiplication in division circuits. Until recently there seemed to be some theoretical justification for this inability since the best known circuits for the first three problems have $O(\log n)$ depth but division appeared to have only $O((\log n)^2)$ depth circuits.

Reif [Re83] reduced the division depth to $O(\log n (\log \log n)^2)$ using a circuit for computing the product of $n^{O(1)}$ n -bit integers *mod* $2^n + 1$, based on Fourier interpolation and evaluation. This circuit had slightly more than polynomial size, but a revised version of the result [Re84] yields polynomial size and $O(\log n \log \log n)$ depth circuits for the same problem.

We present simple circuits of depth $O(\log n)$ and polynomial size, using Chinese remaindering, for the division of two n -bit integers and for the product of n n -bit integers. Since the circuits we consider allow gates with fan-in at most two, our division and iterated product circuits are optimal in depth up to a constant factor.

Besides circuit depth complexity we are also interested in the deterministic space complexity of division. Borodin [Bo77] showed that if for all n a problem can be solved for n input bits by a circuit of depth $O(D(n))$ then it can be solved in space $O(D(n))$, provided the circuits are "log-space uniform" (i.e. some Turing machine, given any n on its input tape, can generate a description of the circuit for n inputs in $\log n$ space). Since Reif's cir-

cuits mentioned above are log-space uniform, it follows that integer division has space complexity $O(\log n \log \log n)$. Unfortunately our circuits for division may not quite be log-space uniform, and it remains an open question whether division has space complexity $\log n$.

Motivated by this question, we prove a number of results. First we show that the three problems division, powering, and iterated product are each strongly reducible to either of the others. Thus all three have the same uniform depth complexity and the same space complexity. Next we give a simple sufficient condition (that some "good modulus sequence" $\{M_n\}$ be log-space generable) for the three problems to have space complexity $\log n$. Finally we show that the problem of testing whether an n -bit integer is divisible by another does indeed have uniform depth complexity $O(\log n)$ and hence space complexity $O(\log n)$.

2. Circuits and Uniformity

We adopt the usual definition of fan-in 2 Boolean circuit families in which the n -th circuit has $g(n)$ inputs and $h(n)$ outputs where g and h are non-decreasing polynomially bounded functions. With this definition depth $O(\log n)$ implies polynomial size. Using the notion of uniformity (see the Introduction) we can define a basic complexity class:

Definition [Ru81]: The class NC^1 consists of all functions f computable by a log-space uniform circuit family of depth $O(\log n)$.

Thus every function in NC^1 has deterministic space complexity $O(\log n)$ [Bo77]. Using standard methods [Sa76] it is easy to see that multiplication of two n -bit integers and addition of n n -bit integers are each in NC^1 . It remains an open question whether division of two n -bit integers is in NC^1 .

Although log-space uniformity is desirable for theoretical reasons, there is a weaker kind of uniformity which provides a natural condition on cir-

cuit families. The builder of computer hardware may simply want to have fast circuits which are easy to construct. Once a circuit has been constructed it will be used over and over again. We thus propose the following definition:

Definition: A family $\langle \alpha_n \rangle$ of circuits is *P-uniform* provided some deterministic Turing machine can compute the transformation $1^n \rightarrow \bar{\alpha}_n$ in time $n^{O(1)}$.

Some of our circuits require internal constants which are polynomial-time computable but do not appear to be log-space computable, and thus are only P-uniform. However, even though they may not be log-space uniform they almost are, in that they can be generated in space $O(\log n \log \log n)$.

A useful notion of reducibility for circuits is the following definition [Co83].

Definition: f is *NC¹ reducible to g* if and only if there exists a log-space uniform circuit family $\langle \alpha_n \rangle$ which computes f with $\text{depth}(\alpha_n) = O(\log n)$ where, in addition to the usual nodes, oracle nodes for g are allowed.

An *oracle node* is a node which has some sequence y_1, \dots, y_r of input edges and z_1, \dots, z_s of output edges with associated function $(z_1, \dots, z_s) = g(y_1, \dots, y_r)$.

For the purpose of defining depth, the oracle node counts as $\text{depth} \lceil \log(r+s) \rceil$.

An important consequence of this definition is that if f is *NC¹ reducible to g* and g is computable by depth $O(\log^k n)$ uniform circuits then f is also computable by depth $O(\log^k n)$ uniform circuits. This applies whether "uniform" means "log-space uniform" or "P-uniform".

3. Powering and Division are Equivalent

Let x, y be n -bit positive integers. The DIVISION problem is to compute the n -bit representation of $\left\lfloor \frac{x}{y} \right\rfloor$.

The POWERING problem is to compute the n^2 -bit representation of x^i for $i=0, \dots, n$. The following result is adapted from [Ho79].

Theorem 3.1 DIVISION is *NC¹ reducible to POWERING*.

Proof: For integers x, y where $0 < x < 2^n, 2 \leq y < 2^n$ we wish to compute $\left\lfloor \frac{x}{y} \right\rfloor$. We first compute an under-approximation \bar{y}^{-1} of y^{-1} with error $< 2^{-n}$. Then we compute $t = x \bar{y}^{-1}$ which approximates $\frac{x}{y}$ with error < 1 , and determine which one of $\lfloor t \rfloor$ or

$$\lfloor t \rfloor + 1 \text{ is } \left\lfloor \frac{x}{y} \right\rfloor.$$

Let $u = 1 - y \cdot 2^{-j}$ where $j \geq 2$ is an integer such that $2^{j-1} \leq y < 2^j$. Thus $|u| \leq \frac{1}{2}$. Consider the series $y^{-1} = 2^{-j} (1-u)^{-1} = 2^{-j} (1+u+u^2+\dots)$. Set $\bar{y}^{-1} = 2^{-j} (1+u+\dots+u^{n-1})$. Then $y^{-1} - \bar{y}^{-1} \leq 2^{-j} \sum_{i \geq n} 2^{-i} < 2^{-n}$.

The circuit computes $\left\lfloor \frac{x}{y} \right\rfloor$ using scaled arithmetic of n^2 bits of precision as follows:

(1) Determine $j \geq 2$ such that $2^{j-1} \leq y < 2^j$ and compute $u = 1 - y \cdot 2^{-j}$

(2) Evaluate $u^i, i=0, \dots, n-1$ using the n -bit powering circuit.

(3) Compute $\bar{y}^{-1} = 2^{-j} (1+u+\dots+u^{n-1})$

(4) Compute $t = x \bar{y}^{-1}$ and truncate to obtain $\lfloor t \rfloor$. Note that $xy^{-1} \geq x \bar{y}^{-1} \geq xy^{-1} - 2^{-n} x$.

(5) Compute $r = x - y \lfloor t \rfloor$ and determine whether $\left\lfloor \frac{x}{y} \right\rfloor$ is $\lfloor t \rfloor$ or $\lfloor t \rfloor + 1$.

All of these steps have depth $O(\log n)$ except possibly the powering in step (2). ■

Theorem 3.2 POWERING is *NC¹ reducible to DIVISION*.

Proof: Let x be an n -bit integer. We want to compute x^0, \dots, x^n . We use a similar identity to the one in the previous reduction but in reverse.

$$\frac{2^{2n^3+2n^2}}{2^{2n^2}-x} = 2^{2n^3} \frac{1}{1-2^{-2n^2}x} = \sum_{i \geq 0} 2^{2n^2(n-i)} x^i$$

Note that $\sum_{i > n} 2^{2n^2(n-i)} x^i = 2^{-2n^2} x^{n+1} \sum_{j \geq 0} (x 2^{-2n^2})^j$ which is $\ll \frac{1}{2} \sum_{j \geq 0} (2^{-n})^j < 1$.

The circuit for computing x^0, \dots, x^n will implement the following procedure:

(1) Set $u = 2^{2n^3+2n^2}$ and compute $v = 2^{2n^2} - x$

(2) Evaluate $y = \left\lfloor \frac{u}{v} \right\rfloor$ using the $2n^3+2n^2$ -bit division circuit.

From the above identity it follows that $y = \sum_{0 \leq i \leq n} 2^{2n^2(n-i)} x^i$.

(3) Read off x^{n-i} as the bits in positions $2n^2 i$ to $2n^2(i+1) - 1$ from the right in y .

All of these steps have depth $O(\log n)$ except possibly the division in step (2). ■

4. Arithmetic Operations Modulo Small Integers

The results of this section are due to McKenzie and Cook [McCo84].

For x and m integers we write $x \bmod m$ for the unique integer y such that $y \equiv x \pmod m$ and $0 \leq y < m$.

Lemma 4.1 For inputs x of n bits and $m \leq n$ the problems of computing $x \bmod m$, $\left\lfloor \frac{x}{m} \right\rfloor$, or $x^{-1} \bmod m$ (if an inverse exists) are all in NC^1 .

Proof: Consider the mod computation first. In space $O(\log n)$ for each $m \leq n$ we may compute $a_{im} = 2^i \bmod m$ for $i=0, \dots, n-1$ and hardwire them into the circuit. Let $x = \sum_{i=0}^{n-1} x_i 2^i$. Then $x \bmod m = \sum_{i=0}^{n-1} x_i a_{im} \bmod m$. The circuit computes $y = \sum_{i=0}^{n-1} x_i a_{im}$ and reduces the result $\bmod m$ by subtracting off in parallel the multiples of m , $0 \cdot m, 1 \cdot m, \dots, (n-1) \cdot m$, and choosing the appropriate difference. Since y has $O(\log n)$ bits the circuit has $O(\log n)$ depth.

In order to compute $z = \left\lfloor \frac{x}{m} \right\rfloor$ use the above circuit and apply an NC^1 reduction from division to mod computation given by Alt and Blum [AlBl83]. Namely, for $i=0, \dots, n$ bit z_i is 1 if and only if $2(x_n \dots x_{i+1} \bmod m) + x_i \geq m$. To compute $x^{-1} \bmod m$ first compute $y = x \bmod m$ and then in parallel multiply y by each residue z modulo m and find the z for which the result is $\equiv 1 \pmod m$. ■

Theorem 4.2 Given integers x_1, \dots, x_n and $p^l \leq n$ a prime power where $0 \leq x_1, \dots, x_n < p^l$ the product $\prod_{i=1}^n x_i \bmod p^l$ can be computed in NC^1 .

Proof: It is a known fact of number theory (e.g. [Hu82]) that $\mathbb{Z}_{p^l}^*$ is cyclic except when $p=2$, in which case $\mathbb{Z}_{p^l}^*$ is generated by 5 and 2^l-1 . The basic idea of the algorithm is to hardwire in a table of discrete logarithms for each prime power $< n$ and then reduce the problem to one of computing iterated addition.

In $O(\log n)$ space it is possible to factor any number $\leq n$ and so determine whether it is a prime power. For each $p^l \leq n$ ($p \neq 2$ or $l > 2$) in $O(\log n)$ space one can find a generator g for $\mathbb{Z}_{p^l}^*$ by brute force and then compute all powers of g up to $p^l - p^{l-1}$ and hardwire them into the circuit. For each $4 < 2^l \leq n$ in $O(\log n)$ space one can compute $(-1)^a 5^b \bmod 2^l$ for $a=0,1$ and $0 \leq b < 2^{l-2}$ and

hardwire them into the circuit.

These tables may be used in either direction as tables of powers or of discrete logarithms.

The algorithm then proceeds as follows:

- (1) Compute the largest power, j_i , of p which divides x_i for $i=1, \dots, n$ in parallel.
- (2) Compute $y_i = x_i / p^{j_i}$ for $i=1, \dots, n$.
- (3) Compute $j = \sum_{i=1}^n j_i$.
- (4) Test if $p \neq 2$ or $p^l = 2, 4$. If either condition holds perform A else perform B.

Part A

- (5) Find each y_i in the table for p^l and read off its discrete logarithm, a_i .
- (6) Compute $a = \sum_{i=1}^n a_i$.
- (7) Compute $\bar{a} = a \bmod p^l - p^{l-1}$.
- (8) Read off $\prod_{i=1}^n y_i = g^{\bar{a}} \bmod p^l$ from the table.

Part B

- (5) Find each y_i in the table for 2^l and read off its representation as powers of 2^l-1 and 5, a_i and b_i .
- (6) Compute $a = \sum_{i=1}^n a_i$ and $b = \sum_{i=1}^n b_i$.
- (7) Compute $\bar{a} = a \bmod 2$ and $\bar{b} = b \bmod 2^{l-2}$.
- (8) Read off $\prod_{i=1}^n y_i = (-1)^{\bar{a}} 5^{\bar{b}} \bmod 2^l$ from the table.

- (9) Compute $\prod_{i=1}^n x_i = p^j \prod_{i=1}^n y_i \bmod p^l$.

The table look-ups can be computed in $O(\log n)$ depth, the modulo operations are computed as in Lemma 4.1, and the other steps can be computed using fast iterated addition circuits in $O(\log n)$ depth. ■

McKenzie and Cook also show how the above circuits may be used to compute iterated products for any small modulus by Chinese remaindering. It is interesting to note the following:

Theorem 4.3 For n -bit integers a and b , computing $a^b \bmod m$ where $m \leq n$ is in NC^1 .

Proof: Apply the same technique as above, taking discrete logarithms, multiplying by b and then exponentiating $\bmod m$. ■

5. Log Depth Circuits for Division and Iterated Product

Let x_1, \dots, x_n be n bit positive integers. The ITERATED PRODUCT problem is to compute $\prod_{i=1}^n x_i$. It is clear that POWERING is reducible to ITERATED PRODUCT (it is little more than a special case) and so POWERING and DIVISION will be computable in small depth if we can find small depth circuits for ITERATED PRODUCT. In order to solve this problem we will make use of Chinese remaindering and the circuits for arithmetic operations modulo small integers.

The Chinese remainder theorem yields a process for determining, given the values of an integer modulo a sequence of relatively prime numbers, the result of taking that integer modulo their product. More formally the CHINESE REMAINDERING problem for pairwise relatively prime integers c_1, \dots, c_n is: given inputs c_1, \dots, c_n , and $x \bmod c_1, \dots, x \bmod c_n$, compute $x \bmod \prod_{j=1}^n c_j$.

Lemma 5.1 CHINESE REMAINDERING for pairwise relatively prime c_1, \dots, c_n where $1 < c_1 < \dots < c_n \leq n^2$ is NC^1 reducible to the problem of computing $c = \prod_{i=1}^n c_i$.

Proof:

The circuit performs:

- (1) Call the oracle to obtain $c = \prod_{i=1}^n c_i$.
- (2) Compute $v_i = \prod_{j \neq i} c_j$ by dividing c by c_i (by Lemma 4.1) for $i=1, \dots, n$ in parallel.
- (3) Solve $v_i w_i \equiv 1 \pmod{c_i}$ for w_1, \dots, w_n in parallel.
- (4) Compute the interpolation constants, $u_i = v_i w_i$ for $i=1, \dots, n$.
- (5) Compute $y = \sum_{i=1}^n (x \bmod c_i) u_i$ by multiplying in parallel and then computing a series sum.
- (6) For each $t, 0 \leq t \leq nc_n$ compute $y_t = y - tc$.
- (7) Set $x \bmod c$ to be the unique y_t such that $0 \leq y_t < c$.

Since each c_i is small, representable in $O(\log n)$ bits, step (2) may be computed in depth $O(\log n)$, similarly step (3) can be computed by brute force. Steps (4) and (5) can be computed by multiplying in parallel and then using multiple addition. Steps (6) and (7) involve simple multiplication and then comparisons in parallel. Each of these steps is of depth $O(\log n)$. ■

If we can compute $\prod_{i=1}^n x_i \bmod c_j$ for a set of relatively prime c_1, \dots, c_s such that $\prod_{j=1}^s c_j > \prod_{i=1}^n x_i$, then the result of the interpolation process of Chinese remaindering will give the value of $\prod_{i=1}^n x_i$ exactly. This fact and the above lemma motivate the following definition.

Definition: A sequence M_1, M_2, \dots is a *good modulus sequence* if and only if there are polynomials $q(n)$ and $r(n)$ such that for all n :

- (i) $2^n \leq M_n \leq 2^{q(n)}$
- (ii) For any prime p , $p^l | M_n$ implies that $p^l \leq r(n)$.

Theorem 5.2 ITERATED PRODUCT is NC^1 reducible to the problem of computing any good modulus sequence $\{M_n\}$.

Proof: From the definition of good modulus sequence it is clear that $M_{n^2} \geq 2^{n^2} > \prod_{i=1}^n x_n$.

We obtain the following algorithm.

- (1) Call the good modulus sequence oracle to obtain M_{n^2} .
- (2) Factor M_{n^2} to obtain prime power factors $c_i = p_i^{k_i}$ for $i=1, \dots, s$.
- (3) Compute in parallel $b_{ij} = x_i \bmod c_j$ for $i=1, \dots, n$ and $j=1, \dots, s$.
- (4) Compute $b_j = \prod_{i=1}^n b_{ij} \bmod c_j$ for $j=1, \dots, s$.

Note that $b_j = \prod_{i=1}^n x_i \bmod c_j$.

- (5) Compute $\prod_{i=1}^n x_i \bmod M_{n^2}$ using the Chinese remaindering circuit for c_1, \dots, c_s to obtain the iterated product exactly.

Step (2) is brute force because the prime power factors are small and step (3) follows from Lemma 4.1. Using Theorem 4.2 for step (4) the entire circuit has depth $O(\log n)$. ■

The computational problem is now reduced to finding a good modulus sequence efficiently. The next theorem shows how this can be done.

Theorem 5.3 ITERATED PRODUCT is computable by P -uniform Boolean circuits of depth $O(\log n)$.

Proof: In polynomial time we can find the first n primes, p_1, \dots, p_n and compute their product. By the prime number theorem, $p_n = O(n \log n)$, so $\prod_{i=1}^n p_i = 2^{O(n \log n)}$.

Also trivially $2^n \leq \prod_{i=1}^n p_i$.

Thus $\prod_{i=1}^n p_i$ for $n=1,2,\dots$ forms a good modulus sequence.

We can compute this good modulus sequence in polynomial time, hardwire the values into the circuit and then apply Theorem 5.2 to get the desired result. ■

Using the previous reductions we have:

Corollary 5.4 DIVISION and POWERING are computable by P-uniform Boolean circuits of depth $O(\log n)$. ■

6. Iterated Product and Powering are Equivalent

As was previously stated POWERING is easily NC^1 reducible to ITERATED PRODUCT but the reducibility in reverse is far from obvious.

Theorem 6.1 ITERATED PRODUCT is NC^1 reducible to POWERING.

Proof We use the reduction of ITERATED PRODUCT to computing a good modulus sequence.

The algorithm proceeds as follows

- (1) Set $x = 2^{2n+1}$.
- (2) Use the powering circuit to compute $y = x^{2^n}$.

Note that $y = \sum_{i=0}^{2n} \binom{2n}{i} 2^{2ni}$.

- (3) Read off $\binom{2n}{n}$ as bits in positions $2n^2$ to $2n^2+2n-1$ from the right in y .

Note that $2^{2n} > \binom{2n}{n} \geq 2^n$.

By elementary arithmetic the exponent of the largest power of prime p dividing $n!$ is $\sum_{i>0} \left\lfloor \frac{n}{p^i} \right\rfloor$. Thus

the largest power dividing $\binom{2n}{n}$ is $\sum_{i>0} \left\{ \left\lfloor \frac{2n}{p^i} \right\rfloor - 2 \left\lfloor \frac{n}{p^i} \right\rfloor \right\}$.

Now each of these terms is ≤ 1 and the terms vanish when $p^i > 2n$ so that the largest power p^t dividing

$\binom{2n}{n}$ satisfies $p^t < 2n$. From this we see that

$\binom{2n}{n}$ for $n=1,2,\dots$ forms a good modulus sequence

and so the reduction is correct. ■

Corollary 6.2 DIVISION, POWERING, and ITERATED PRODUCT are all NC^1 equivalent.

7. Divisibility

Although the DIVISION problem has P-uniform $O(\log n)$ depth circuits, it is still unclear whether or not it has log-space uniform $O(\log n)$ depth circuits. Despite the fact that we are unable to answer this question it is possible to find such circuits for a closely related problem, DIVISIBILITY.

Let x, y be n -bit integers.

The output of the DIVISIBILITY problem is $\begin{cases} 1 & \text{if } y|x \\ 0 & \text{otherwise.} \end{cases}$

Theorem 7.1 DIVISIBILITY is in NC^1 , and hence has deterministic space complexity $O(\log n)$.

Proof: For each of n primes $p_1 < \dots < p_n$ not dividing y we can solve $yz \equiv x \pmod{p_i}$ to obtain z_i . If we could compute $M = \prod_{i=1}^n p_i$ then, as in Lemma 5.1, we could find the unique z such that $0 \leq z < M$ and $z \equiv z_i \pmod{p_i}$ for each i . Such a z would be the only possible candidate for a solution to $yz = x$.

If $u_i \equiv \begin{cases} 1 \pmod{p_i} \\ 0 \pmod{p_j, j \neq i} \end{cases}$ then $z \equiv \sum_{i=1}^n u_i z_i \pmod{M}$.

If, in addition, $0 \leq u_i < M$ then $z \equiv z^{(t)}$ for some t , $0 \leq t < np_n$ where $z^{(t)} = \sum_{i=1}^n u_i z_i - tM$. It follows that

$yz^{(t)} = x$ if and only if $\exists t, 0 \leq t < np_n$ such that $yz^{(t)} = x$. It is not necessary, however, to compute $z^{(t)}$ explicitly. We merely need to test the condition modulo sufficiently many primes. Since for any t , $|yz^{(t)} - x| < np_n 2^{n+1}M$, it suffices for the product of these primes to exceed $np_n 2^{n+1}M$. Note that the equation always holds modulo each of the primes p_1, \dots, p_n , so that it suffices to choose additional primes whose product exceeds $np_n 2^{n+1}$.

The resulting algorithm is:

- (1) Find the first $3n$ primes.
- (2) Compute $y_i = y \pmod{p_i}$ for each of these primes.
- (3) Select the first n primes from those found in (1) such that $y_i \neq 0$.

Note that since $y \leq 2^n$ it cannot have more than n different prime factors.

In the remainder of the algorithm we designate these primes as p_1, \dots, p_n and the remaining primes among the first $3n$ as q_1, \dots, q_{2n} .

- (4) Compute $z_i = xy_i^{-1} \pmod{p_i}$ for each $i=1, \dots, n$.

- (5) Compute $M_k = \prod_{i=1}^n p_i \pmod{q_k}$ for each $k=1, \dots, 2n$.

Note that $M_k \equiv M \pmod{q_k}$.

- (6) Compute $u_{ik} = \prod_{j \neq i} p_j \pmod{q_k} (= M_k p_i^{-1} \pmod{q_k})$

for each $i=1, \dots, n$ and $k=1, \dots, 2n$.

(7) Compute $w_i = \prod_{j \neq i} p_j^{-1} \bmod p_i$ for each $i=1, \dots, n$.

(8) Compute $w_{ik} = w_i \bmod q_k$ for each $i=1, \dots, n$ and $k=1, \dots, 2n$.

(9) Compute $u_{ik} = v_{ik} w_{ik} \bmod q_k$ for each $i=1, \dots, n$ and $k=1, \dots, 2n$.

Note that $u_{ik} \equiv u_i \bmod q_k$.

(10) Compute $z_{ik} = z_i \bmod q_k$ for each $i=1, \dots, n$ and $k=1, \dots, 2n$.

(11) Compute $z_k^{(t)} = \sum_{i=1}^n u_{ik} z_{ik} - t M_k \bmod q_k$ for each $k=1, \dots, 2n$ and $t=1, \dots, np_n$.

(12) Check if there exists a t such that for all k , $y_k z_k^{(t)} \equiv x \bmod q_k$. If such a t exists output 1 else output 0.

All the operations are computed modulo small primes in $O(\log n)$ depth and the remaining computations are simple tests in parallel which also have $O(\log n)$ depth. ■

8. Summary and Open Problems

From the $O(\log n)$ depth P-uniform circuits for the problems presented here, using the results of Alt [Al84], a large class of natural problems can now be shown to have $O(\log n)$ depth circuits. It is unknown whether any of these circuits may be made log-space uniform, which would imply that the problems are computable in deterministic log space.

An interesting problem related to powering is base conversion of integers from a fixed base, e.g. 3, to binary. This can be easily seen to have P-uniform $O(\log n)$ depth circuits even without the machinery presented here. All that is required is to precompute $3^0, \dots, 3^{n-1}$ in binary, hardwire them into the circuit, and on input $(b_{n-1} \dots b_0)_3$ compute $\sum_{i=0}^{n-1} b_i 3^i$. It is an open question whether this problem, which is reducible to powering for a fixed base, has $O(\log n)$ depth log-space uniform circuits when the base is not a power of 2.

The class of problems which are reducible to the decision problem, DIVISIBILITY, may be worth investigating since our results imply that such problems would have log-space uniform $O(\log n)$ depth Boolean circuits.

Finally, there is a stronger and in some ways more natural definition of uniform than log-space uniform. This stronger form was introduced by Ruzzo [Ru81] and called U_{E^r} -uniform (see [Co83]). If this condition is used to define NC^1 , then NC^1 can be characterized simply as the class of problems computable in time $O(\log n)$ on an alternating Turing machine. Unfortunately, it is not clear whether

all the results shown here still hold with the stronger condition. In particular, it would be interesting to know whether DIVISIBILITY, iterated product modulo small prime powers, and the reduction of iterated product to powering, have NC^1 circuits in this stronger sense.

References

- [Al84] Alt, H. *Comparison of arithmetic functions with respect to Boolean circuit depth*. 16th ACM STOC 1984, pp. 466-470.
- [AlB183] Alt, H., Blum, N. *On the Boolean circuit depth of division related functions*. Dept. of Computer Science, Pennsylvania State University, 1983.
- [Bo77] Borodin, A. *On Relating time and space to size and depth*. SIAM J. Comput. 6 (1977), pp. 733-744.
- [Co83] Cook, S. A. *The classification of problems which have fast parallel algorithms*. Lecture Notes in Computer Science, vol. 158 Springer-Verlag 1983.
- [Ho79] Hoover, H. J. *Some topics in circuit complexity*. M.Sc. thesis and TR-139/80, Dept. of Computer Science, Univ. of Toronto, Dec. 1979.
- [Hu82] Hua, L. K. *Introduction to Number Theory*. Springer-Verlag, 1982.
- [McCo84] McKenzie, P., Cook, S. A. *The parallel complexity of some Abelian permutation group problems*. Manuscript, March 1984.
- [Re83] Reif, J. *Logarithmic depth circuits for algebraic functions*. 24th IEEE FOCS 1983, pp. 138-145.
- [Re84] Reif, J. *Logarithmic depth circuits for algebraic functions*. Revised version 1984.
- [Ru81] Ruzzo, W. L. *On uniform circuit complexity*. JCSS 22 (1981), pp. 365-383.
- [Sa76] Savage, J. E. *The complexity of computing*. Wiley, 1976.