LOG DEPTH CIRCUITS FOR DIVISION AND RELATED PROBLEMS†

†A preliminary version of this paper appeared in the proceedings of the 25th IEEE Symposium on the Foundations of Computer Science, 1984.

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

Dept. of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

Revised June 1985

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.

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 $O(\log n\ (\mathrm{loglog}\ n)^2)$ using a circuit for computing the product of $n^{O(1)}$ n-bit integers $\mathit{mod}\ 2^n + 1$, 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 $O(\log n\ \mathrm{loglog}\ n)$ 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 Turing machine 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 circuits mentioned above are log-space uniform, it follows that integer division has space complexity $O(\log n\ \mathrm{loglog}\ n)$. 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 $O(\log n)$. 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 circuit 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 $< \alpha_n >$ of circuits is *P-uniform* provided some deterministic Turing machine can compute the transformation $1^n \to \bar{\alpha}_n$ in time $n^{O(1)}$ where $\bar{\alpha}_n$ is the standard encoding [Ru81] of α_n. 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 the only parts of the circuits which are not log-space constructible may be generated in $O(\log n\ \mathrm{loglog}\ n)$ space using Reif's powering algorithm [Re84]. A useful notion of reducibility for circuits is the following definition [Co83].

Definition: f is NC^1 *reducible to* g if and only if there exists a log-space uniform circuit family $< \alpha_n >$ which computes f with $\mathrm{depth}(\alpha_n) = O(\log n)$ where, in addition to the usual nodes, oracle nodes for g are allowed.
An *oracle node* is a node which has some sequence $y_1, \ldots, y_r$ of input edges and $z_1, \ldots, z_s$ of output edges with associated function $(z_1, \ldots, z_s) = g(y_1, \ldots, y_r)$.
For the purpose of defining depth, the oracle node counts as depth $\lceil \log(r+s) \rceil$. An important consequence of this definition is that if f is NC^1 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". 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 $i = 0, \ldots, n$. The following result is adapted from [Ho79].

Theorem 3.1 DIVISION is NC^1 reducible to POWERING.

Proof: For integers x, y where $0 < x < 2^n$, $2 \le y < 2^n$ 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 $\lfloor t \rfloor + 1$ is $\left\lfloor \frac{x}{y} \right\rfloor$. Let $u = 1 - y\ 2^{-j}$ where $j \ge 2$ is an integer such that $2^{j-1} \le y < 2^j$. Thus $|u| \le \frac{1}{2}$. Consider the series $y^{-1} = 2^{-j}(1-u)^{-1} = 2^{-j}(1 + u + u^2 + \cdots)$.
Set $\bar{y}^{-1} = 2^{-j}(1 + u + \cdots + u^{n-1})$. Then $y^{-1} - \bar{y}^{-1} \le 2^{-j} \sum_{i \ge n} 2^{-i} < 2^{-n}$.

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 $u = 1 - y2^{-j}$
(2) Evaluate u^i , $i = 0, \ldots, n-1$ using the n-bit powering circuit.
(3) Compute $\bar{y}^{-1} = 2^{-j}(1 + u + \cdots + u^{n-1})$
(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). *qed*

Theorem 3.2 POWERING is NC^1 reducible to DIVISION.

Proof: Let x be an n-bit integer. We want to compute $x^0, \ldots, x^n$. We use a similar identity to the one in the previous reduction but in reverse, choosing a scaling factor so that none of the powers of x overlap in the resulting binary representation.

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

Note that $\sum_{i>n} 2^{2n^2(n-i)} x^i = 2^{-2n^2} x^{n+1} \sum_{j \geq 0} (x2^{-2n^2})^j$
which is $\ll \frac{1}{2} \sum_{j \geq 0} (2^{-n})^j < 1$.

The circuit for computing $x^0, \ldots, x^n$ 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 (position 0 contains the low order bit).

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

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

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

Lemma 4.1 For inputs x of n bits and $m \leq n$ the problems of computing $x \; mod \; m$, $\left\lfloor \frac{x}{m} \right\rfloor$, or $x^{-1} \; mod \; m$ (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 $a_{im} = 2^i \; mod \; m$ for $i = 0, \ldots, n-1$ and hardwire them into the circuit. Let $x = \sum_{i=0}^{n-1} x_i 2^i$. Then $x \; mod \; m = \sum_{i=0}^{n-1} x_i a_{im} \; mod \; m$. The circuit computes $y = \sum_{i=0}^{n-1} x_i a_{im}$ and reduces the result $mod \; m$ by subtracting off in parallel the multiples of m, $0 bulletm, 1 bulletm, \ldots, (n-1) bulletm$, 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 $i = 0, \ldots, n$ bit z_i is 1 if and only if $2\,(\, x_n \ldots x_{i+1} \; mod \; m\,) + x_i \geq m$.
To compute $x^{-1} \; mod \; m$ first compute $y = x \; mod \; m$ and then in parallel multiply y by each residue z modulo m and find the z for which the result is $equiv \; 1 \; mod \; m$. *qed*

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

Proof: It is a known fact of number theory (e.g. [Hu82]) that $\mathbf{Z}^*_{p^l}$ is cyclic except when $p = 2$ and $l > 2$, in which case $\mathbf{Z}^*_{p^l}$ 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 $l \leq 2$) in $O(\log n)$ space one can find a generator g for $\mathbf{Z}^*_{p^l}$ by brute force and then compute all pow-

ers of g up to $p^l - p^{l-1}$, the order of $\mathbf{Z}^*_{p^l}$, and hardwire them into the circuit. For each $4 < 2^l \le n$ in $O(\log n)$ space one can compute $(-1)^a 5^b \ mod\ 2^l$ for $a = 0, 1$ and $0 \le b < 2^{l-2}$ 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 $i = 1, \ldots, n$ in parallel.
(2) Compute $y_i = x_i / p^{j_i}$ for $i = 1, \ldots, n$.
(3) Compute $j = \sum_{i=1}^{n} j_i$.
Note that the y_i are now in $\mathbf{Z}^*_{p^l}$ and $\prod_{i=1}^{n} x_i\ equiv\ p^j \prod_{i=1}^{n} y_i\ mod\ p^l$.
(4) Test if $p \ne 2$ 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 $\bar{a} = a\ mod\ (p^l - p^{l-1})$.
(8) Read off $\prod_{i=1}^{n} y_i = g^{\bar{a}}\ mod\ p^l$ 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 $\bar{a} = a\ mod\ 2$ and $\bar{b} = b\ mod\ 2^{l-2}$.
(8) Read off $\prod_{i=1}^{n} y_i = (-1)^{\bar{a}} 5^{\bar{b}}\ mod\ 2^l$ from the table.

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

The table look-ups can be computed in $O(\log n)$ depth using selector trees, the modulo operations are computed as in Lemma 4.1, and the other steps can be computed using fast iterated addition circuits [Sa76] in $O(\log n)$ depth. *qed* 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 $a^b\ mod\ m$ where $m \le n$ is in NC^1.

Proof Sketch: By Chinese remaindering the problem can be reduced to computing $a^b\ mod\ p^l$, for each prime power factor p^l dividing m. This is solved by the same technique as above, taking discrete logarithms, multiplying by b, and then exponentiating $mod\ p^l$. *qed* Let $x_1, \ldots, x_n$ 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 $c_1, \ldots, c_n$ is: given inputs $c_1, \ldots, c_n$, and $x\ mod\ c_1, \ldots, x\ mod\ c_n$, compute $x\ mod\ \prod_{j=1}^{n} c_j$.

Lemma 5.1 CHINESE REMAINDERING for pairwise relatively prime $c_1, \ldots, c_n$ where $1 < c_1 < \cdots < c_n \le n^2$ 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 $v_i = \prod_{j \ne i} c_j$ by dividing c by c_i (by Lemma 4.1) for $i = 1, \ldots, n$ in parallel.
(3) Solve $v_i w_i equiv 1 \ \ (mod\ c_i)$ for $w_1, \ldots, w_n$ in parallel.
(4) Compute the interpolation constants, $u_i = v_i w_i$ for $i = 1, \ldots, n$.
Note that $u_i\ equiv\ 1\ mod\ c_i$ and $u_i\ equiv\ 0\ mod\ c_j$ for $j \ne i$.
(5) Compute $y = \sum_{i=1}^{n} (x\ mod\ c_i) u_i$ by multiplying in parallel and then computing a series sum.

It is necessary to reduce y modulo c to obtain the desired result. The largest multiple of c which is less than y can be estimated since

$$y = \sum_{i=1}^{n} (x \ mod \ c_i) v_i w_i = \sum_{i=1}^{n} \frac{(x \ mod \ c_i) w_i}{c_i} c.$$

(6) Compute $r_i = m\,(x\,mod\,c_i)\,w_i$ for $i = 1, \ldots, n$ where $m = 2^{lc1 \log_2 nrc1}$.

Thus $y = \sum_{i=1}^{n} \frac{r_i}{m\,c_i} c$.

(7) Compute $s_i = lf1 r_i / c_i rf1$ for $i = 1, \ldots, n$.

(8) Compute $s = \sum_{i=1}^{n} s_i$.

(9) Compute $t = lf1 s/m rf1$, i.e. truncate the right $lc1 \log_2 nrc1$ bits of s.

Note that $0 \le y - \frac{s}{m} c = \sum_{i=1}^{n} \left(\frac{r_i}{m\,c_i} - \frac{1}{m} \left\lfloor \frac{r_i}{c_i} \right\rfloor \right) c < \sum_{i=1}^{n} \frac{1}{m} c \le c$.

Thus $0 \le y - tc < 2c$.

(10) Set $x\ mod\ c$ to $y - tc$ if $0 \le y - tc < c$, otherwise set it to $y - (t + 1)c$.

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), (5), (6), and (8) can be computed by multiplying in parallel and then using multiple addition. Step (7) involves divisions of $O(\log n)$-bit integers by $O(\log n)$-bit integers and can be done using any reasonable division circuit (even linear depth would not hurt here). Step (10) requires simple multiplication, comparison and subtraction in parallel. Each of these steps is of depth $O(\log n)$. *qed* If we can compute $\prod_{i=1}^{n} x_i \ mod \ c_j$ for a set of relatively prime $c_1, \ldots, c_s$ 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 $M_1, M_2, \cdots$ is a *good modulus sequence* if and only if there are polynomials $q(n)$ and $r(n)$ such that for all n:

(i) $2^n \le M_n \le 2^{q(n)}$

(ii) For any prime p, $p^l | M_n$ implies that $p^l \le r(n)$.

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 $M_{n^2} \ge 2^{n^2} > \prod_{i=1}^{n} x_n$.

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 $c_i = p_i^{l_i}$ for $i = 1, \ldots, s$.

(3) Compute in parallel $b_{ij} = x_i \ mod \ c_j$ for $i = 1, \ldots, n$ and $j = 1, \ldots, s$.

(4) Compute $b_j = \prod_{i=1}^{n} b_{ij} \ mod \ c_j$ for $j = 1, \ldots, s$.

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

(5) Compute $\prod_{i=1}^{n} x_i \ mod \ M_{n^2}$ using the Chinese remaindering circuit for $c_1, \ldots, c_s$ 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 $O(\log n)$. *qed* 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, $p_1, \ldots, p_n$ 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 $2^n \le \prod_{i=1}^{n} p_i$.

Thus $\prod_{i=1}^{n} p_i$ for $n = 1, 2, \cdots$ 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. *qed*

Using the previous reductions we have:

Corollary 5.4 DIVISION and POWERING are computable by P-uniform Boolean circuits of depth $O(\log n)$. *qed*
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 $y = x^{2n}$.
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 (position 0 contains the low order bit).
Note that $2^{2n} > \binom{2n}{n} \geq 2^n$.
By elementary arithmetic (e.g. [Hu82]) 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^l dividing $\binom{2n}{n}$ satisfies $p^l < 2n$. From this we see that $\binom{2n}{n}$ for $n = 1, 2, \cdots$ forms a good modulus sequence and so the reduction is correct. *qed*

Corollary 6.2 DIVISION, POWERING, and ITERATED PRODUCT are all NC^1 equivalent. 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 $p_1 < \cdots < p_n$ not dividing y we can solve $y\, z\ equiv\ x\ mod\ p_i$ 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 $z\ equiv\ z_i\ mod\ p_i$ for each i. Such a z would be the only possible candidate for a solution to $yz = x$. If $u_i\ equiv \begin{cases} 1\ mod\ p_i \\ 0\ mod\ p_j,\ j \neq i \end{cases}$ then $z\ equiv \sum_{i=1}^{n} u_i\, z_i\ mod\ M$. If, in addition, $0 \leq u_i < M$ then $z = z^{(t)}$ for some t, $0 \leq t < np_n$ where $z^{(t)} = \sum_{i=1}^{n} u_i\, z_i - t\, M$. It follows that $y|x$ if and only if $existt$, $0 \leq t < np_n$ 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 $p_1, \ldots, p_n$, 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 $y_i = y\ mod\ p_i$ 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 $p_1, \ldots, p_n$ and the remaining primes among the first $3n$ as $q_1, \ldots, q_{2n}$.
(4) Compute $z_i = xy_i^{-1}\ mod\ p_i$ for each $i = 1, \ldots, n$.
(5) Compute $M_k = \prod_{i=1}^{n} p_i\ mod\ q_k$ for each $k = 1, \ldots, 2n$.
Note that $M_k\ equiv\ M\ mod\ q_k$.
(6) Compute $v_{ik} = \prod_{j \neq i} p_j\ mod\ q_k$ ($= M_k\ p_i^{-1}\ mod\ q_k$) for each $i = 1, \ldots, n$ and $k = 1, \ldots, 2n$.
(7) Compute $w_i = \prod_{j \neq i} p_j^{-1}\ mod\ p_i$ for each $i = 1, \ldots, n$.
(8) Compute $w_{ik} = w_i\ mod\ q_k$ for each $i = 1, \ldots, n$ and $k = 1, \ldots, 2n$.
(9) Compute $u_{ik} = v_{ik}\, w_{ik}\ mod\ q_k$ for each $i = 1, \ldots, n$ and $k = 1, \ldots, 2n$.

Note that u_{ik} *equiv* u_i *mod* q_k.
(10) Compute $z_{ik} = z_i \ mod \ q_k$ for each $i = 1, \ldots, n$ and $k = 1, \ldots, 2n$.
(11) Compute $z_k^{(t)} = \sum_{i=1}^{n} u_{ik} \, z_{ik} - t \, M_k \ mod \ q_k$ for each $k = 1, \ldots, 2n$ and $t = 1, \ldots, np_n$.
(12) Check if there exists a t such that for all k, $y_k \, z_k^{(t)}$ *equiv* $x \ mod \ q_k$. 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. *qed* In obtaining the $O(\log n)$ depth circuits for the problems of the previous sections we have avoided using the full power of P-uniformity as much as possible. This permitted us to focus on constructing "good modulus sequences" in attempting to produce log-space uniform circuits for these problems. However, this has made our circuits larger than necessary. By making fuller use of polynomial time constructibility, the $O(\log n)$ depth circuits can be simplified somewhat, yielding a reduction in their size. Since this simplification is most dramatic for the POWERING circuits, we describe this case in detail.

Theorem 8.1 POWERING can be computed by P-uniform circuits of depth $O(\log n)$ and size $O(n^5 \log^2 n)$.

Proof: Precompute and hardwire into the circuit:
(a) Primes $p_1, \ldots, p_s$ such that $\prod_{j=1}^{s} p_j > 2^{n^2}$.
(b) The $n+1$-bit under-approximations of the inverses of p_j, $\bar{p}_j^{-1}$, for $j = 1, \ldots, s$.
(c) For $i = 1, \ldots, n$, $j = 1, \ldots, s$, tables of $a^i \ mod \ p_j$ for each a, $0 \le a < p_j$.
(d) $M = \prod_{i=1}^{s} p_j$.
(e) The $2n^2$-bit under-approximation of the inverse of M, $\bar{M}^{-1}$.
(f) Interpolation constants $0 \le u_j < M$ such that u_j *equiv* $\begin{cases} 1 \ mod \ p_j \\ 0 \ mod \ p_i, i \ne j \end{cases}$ for $j = 1, \ldots, s$.

The circuit performs:

(1) Compute $t_j = x \bar{p}_j^{-1}$ and truncate to obtain $lf1t_jrf1$ for $j = 1, \ldots, s$.
(2) Compute $x \ mod \ p_j$ for $j = 1, \ldots, n$ as either $z_j = x - p_j lf1t_jrf1$ or $z_j - p_j$ whichever is between 0 and p_j.
(3) Read $y_{ij} = x^i \ mod \ p_j$ from the tables for $i = 1, \ldots, n$ and $j = 1, \ldots, s$.
(4) Compute $y_i = \sum_{j=1}^{s} y_{ij} u_j$ for $i = 1, \ldots, n$.
(5) Compute $t'_i = y_i \, \bar{M}^{-1}$ and truncate to obtain $lf1t'_irf1$ for $i = 1, \ldots, n$.
(6) Compute x^i for $i = 1, \ldots, n$ as either $z'_i = y_i - Mlf1t'_irf1$ or $z'_i - M$ whichever is between 0 and M.

Certainly $s < n^2$ and thus by the Prime Number Theorem $p_s = O(n^2 \log n)$. Since multiplication of n-bit integers can be done in size $O(n \log n \log\log n)$ [Sa76] and $O(\log n)$ depth, steps (1) and (2) can be computed in size $O(n^3 \log n \log\log n)$ as can steps (5) and (6). Step (4) computes n sums of $O(n^2)$ terms which are each multiplications of $O(\log n)$-bit integers by $O(n^2)$-bit integers. The multiplications in (4) cost a total of $O(n^5 \log n)$ size and the summations cost $O(n^5)$ in size. The table look-ups in step (3) are computed separately for each value of i and for each value of j. It follows that the circuit size for step (3) is dominated by the size of the tables, which contain $O(n \, s \, p_s) = O(n^5 \log n)$ entries of $O(\log n)$ bits each. Thus the total size of the circuit is $O(n^5 \log^2 n)$, as stated. *qed* The circuits for ITERATED PRODUCT are somewhat more complicated but are still the same size as the POWERING circuits.

Theorem 8.2 ITERATED PRODUCT can be computed by P-uniform circuits of depth $O(\log n)$ and size $O(n^5 \log^2 n)$.

Proof Sketch: These circuits are similar to the circuits in Theorem 8.1 above except for the following:
(i) The steps corresponding to (1) and (2) are performed on each x_i for $i = 1, \ldots, n$.
(ii) The steps corresponding to (4), (5) and (6) are performed so as to compute only one n^2-bit output.
(iii) The table look-ups in step (3) are replaced by circuits with tables of discrete logarithms for each p_j like the ones described in steps (5)-(8) of Part A in Theorem 4.2.

Applying changes (i) and (ii) to the arguments in Theorem 8.1, those portions of the circuit may be computed in size $O(n^4 \log n \log\log n)$ and $O(\log n)$ depth. The tables of discrete logarithms required by this circuit have only $O(sp_s)$ entries of $O(\log n)$ bits each, for a total size of $O(n^4 \log^2 n)$. However, the table for each p_j is accessed n different times in parallel (once for each x_i) so the accessing hardware is of size $O(n^5 \log^2 n)$. The other computations are easily within this size bound and the Theorem follows. *qed* DIVISION circuits may be constructed, using the reduction circuits of Theorem 3.1 and the POWERING circuits in Theorem 8.1, of the same asymptotic size as those for POWERING. By applying a trick suggested by Reif we obtain smaller, although more complicated, DIVISION circuits.

Theorem 8.3 DIVISION can be computed by P-uniform circuits of depth $O(\log n)$ and size $O(n^4 \log^3 n)$.

Proof Sketch: In computing DIVISION, the hard part is computing the series, $1 + u + u^2 + \cdots$, to at least n terms for a scaled n-bit u. Instead of computing each term separately, apply the identity

$$\prod_{i=0}^{k}(1 + u^{2^i}) = 1 + u + u^2 + \cdots + u^{2^{k+1}-1}$$

with $k = lf1 \log_2 nrf1$.

The circuit then computes only $O(\log n)$ powers from the POWERING circuit, adds 1 to each of the scaled results, and then computes the ITERATED PRODUCT of the $O(\log n)$ resulting terms. The powering portion of the circuit uses only $O(n^4 \log^3 n)$ size because there are fewer powers to be computed, resulting in smaller tables, and the iterated product portion of the circuit needs fewer simultaneous table accesses and also uses size $O(n^4 \log^3 n)$. *qed* 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 the conversion of integers from one fixed base to another, e.g. base 3 to base 5, when the digits of integers in bases other than 2 are represented by groups of bits. This problem can easily be seen to have P-uniform $O(\log n)$ depth circuits even without the machinery presented here. In our example all that is required is to precompute $3^0, \ldots, 3^{n-1}$ in base 5, hardwire them into the circuit, and on input $(b_{n-1} \ldots b_0)_3$ compute $\sum_{i=0}^{n-1} \sum_{j=1}^{b_i} 3^i$ in base 5. The base 5 summation circuits are simple modifications of standard fast binary summation circuits [Sa76]. 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 one base is not a power of the other. 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^*}- 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.

[Al84] Alt, H. *Comparison of arithmetic functions with respect to Boolean circuit depth.* 16th ACM STOC 1984, pp. 466-470. [AlBl83] 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 Abelian permutation group problems.* TR-181/85, Dept. of Computer Science, Univ. of Toronto, 1985. [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.