POLYLOG DEPTH CIRCUITS FOR INTEGER FACTORING AND DISCRETE LOGARITHMS

by

Jonathan Sorenson

Computer Sciences Technical Report #872

August 1989

Polylog Depth Circuits for Integer Factoring and Discrete Logarithms

Jonathan Sorenson

Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706
USA

August 29, 1989

Abstract.

We discuss parallel algorithms for integer factoring and discrete logarithms. In particular, we give polylog depth probabilistic circuits of subexponential size for both of these problems, which solves an open problem of Adleman and Kompella.

A positive integer is *B-smooth* if all of its prime divisors are at most B. We modify existing sequential algorithms for factoring and discrete logarithms which use such numbers. Analyzing the tradeoffs involved in choosing B, we show that for inputs of length n, setting $B = n^{\log^d n}$, with d positive a constant, gives:

1. Probabilistic boolean circuits of depth $O(\log^{2d+2} n)$ and size $e^{O(n/\log^d n)}$ for finding a proper divisor of a positive composite integer with probability at least $1/2 - o(1)$, and

2. Probabilistic boolean circuits of depth $O(\log^{2d+4} n)$ and size $e^{O(n/\log^d n)}$ for computing discrete logarithms in the finite field $\mathbf{Z}/p\mathbf{Z}$ for p a prime with probability $1 - o(1)$.

These are the first results of this type for both problems.

Sponsored by NSF grant CCR-8552596

University of Wisconsin-Madison Computer Sciences Technical Report #872

1 Introduction

We present polylog depth, subexponential size probabilistic boolean circuits to factor integers and compute discrete logarithms over prime finite fields. These are the first results of this type, and they answer a previously open question posed by Adleman and Kompella [AK88], which asked whether such circuits exist for these two problems.

Our results are motivated by much of the recent work done on parallel number theoretic algorithms [AK88, BS89, BCH86, BvzGH82, BK83, CG85, FT88, KMR87, PR85, Mul87, vzG87] and by the work done on the rigorous analysis of integer factorization [Dix81, Pom82, Pom87, Val89] and discrete logarithm algorithms [Adl79, Odl84, Pom87].

We base our integer factoring circuit on Dixon's algorithm [Dix81], which uses the difference of squares method. Our discrete logarithm circuit is based on the index-calculus method as described by Adleman [Adl79]. Faster methods exist, but they rely on unproven conjectures either for correctness or in their running time analyses. Such gaps in the theory behind these algorithms are acceptable if one is primarily interested in practical algorithms. However, our results are largely of theoretical interest, so using fully rigorous algorithms as the basis of our circuits is especially important. By choosing Dixon's and Adleman's algorithms, which are both fully rigorous, we satisfy this criterion.

Adleman's and Dixon's algorithms make use of smooth numbers, numbers which have only small prime factors, in a similar fashion. They start by choosing a bound B and finding all primes below that bound. Then they generate random elements with special properties until they get at least B of them which factor completely over the primes below B. For integer factoring, these special elements are random squares modulo N, where N is the number to be factored, and for discrete logarithms, these elements are random powers of the generator. Using these elements, they construct a system of linear equations whose solution leads to either a proper divisor of the input or the discrete logarithm of the input.

In optimizing the running times of these algorithms, one analyzes the tradeoff involved in choosing a value for B. If B is small, both finding all the primes and solving the system of equations are fast, but many random elements must be generated, since the probability an element factors over the primes below B is low. If B is large, then generating the elements is faster, but solving the system of equations takes a very long time.

Let $L(n) = e^{\sqrt{n \log n}}$. For sequential algorithms, the values of B that work best are $L(n)^c$ for small constants $c > 0$, where n is the length of the input. The resulting running times are proportional to a small power of B (2 or 3). Pomerance [Pom87] showed that Dixon's algorithm, improved using the techniques in [Pom82], runs in time $L(n)^{\sqrt{2}+o(1)}$. He also achieved the same running time for an improvement to Adleman's algorithm for discrete logarithms. Vallée [Val89] currently has the fastest rigorous integer factoring algorithm, based on the difference of squares method, which runs in time $L(n)^{\sqrt{4/3}+o(1)}$.

Our idea is to set $B = n^{d(n)}$, where $d(n) = \log^d n$ for $d > 0$ a fixed constant. We then prove the existence of:

1. Probabilistic boolean circuits of depth $O(\log^{2d+2} n)$ and size $e^{O(n/\log^d n)}$ for finding a proper divisor of a positive composite integer with probability at least $1/2 - o(1)$, and

2. Probabilistic boolean circuits of depth $O(\log^{2d+4} n)$ and size $e^{O(n/\log^d n)}$ for computing

discrete logarithms in the finite field $\mathbf{Z}/p\mathbf{Z}$ for p a prime, with probability $1 - o(1)$.

Considering the size of these circuits, it should be clear that unbounded fan-in is necessary to achieve such a small depth. However, we also give sublinear depth circuits with bounded fan-in of subexponential size for these problems. For these results, we use much larger functions for $d(n)$:

1. For factoring, we set $d(n) = n^{1/3+o(1)}$ to get bounded fan-in depth $O(n^{2/3+o(1)})$ and size $e^{n^{2/3+o(1)}}$, and
2. For discrete logarithms, we set $d(n) = n^{1/6+o(1)}$ to get bounded fan-in depth $O(n^{5/6+o(1)})$ and size $e^{n^{5/6+o(1)}}$.

For an introduction to factoring and discrete logarithm algorithms, see Koblitz [Kob87], which is also a good source for further references. For surveys of some of the currently best algorithms, along with detailed analyses, see Pomerance [Pom82, Pom87] and Odlyzko [Odl84].

The rest of this paper is organized as follows. In section 2, we discuss some notation and definitions. We present our circuits for factoring integers in section 3. In section 4 we give our circuits for computing discrete logarithms. We conclude with section 5, where we discuss open problems and directions for future research.

2 Preliminaries

In this section we discuss some notation and definitions.

All of our algorithms are given in the form of probabilistic boolean circuits. We define probabilistic boolean circuits similarly to standard boolean circuits, but we also allow gates which can toss a fair coin and use the result for their output. Our circuits compute functions as opposed to only recognizing languages. In addition, we also have two special bits of output; one called the *correctness* bit, and one called the *output* bit. The output bit is on if the rest of the circuit's output "means" something, or in other words, if the circuit computes something for the function value. The correctness bit signals whether or not this output is certain to be correct. The output may be correct without the correctness bit being set. If the output bit is off, then the value of the correctness bit is irrelevant.

We will say that a probabilistic circuit has *zero-sided error* if, whenever the output bit is on, then the correctness bit is also on. An example of such a circuit might be one which generates a quadratic non-residue modulo a prime. If one is found, then it is always "correct." If one is not found, then the output bit is off.

We say a circuit has *one-sided error* if the correctness bit is on at least some of the time when the output bit is on. An example in this case might be a circuit to test primality by attempting to factor the input, because if a divisor is found, then the circuit is certain that the input is not prime, but if no divisor is found, then the input is only probably prime.

Finally, a circuit has *two-sided error* if the correctness bit is never on. An example here might be a circuit which completely factors its input, but is not sure that the divisors it outputs are prime.

A circuit family is *$S(n)$-space uniform* if there exists a deterministic, $O(S(n))$ space bounded, off-line Turing machine which, when given 1^n on its input tape, can compute the description of the circuit for inputs of length n. For more information on parallel computation and circuits, see Cook [Cook85] and Karp and Ramachandran [KR88].

We assume that the basic arithmetic operations of addition, multiplication, and subtraction can all be done using logspace uniform circuits of depth $O(\log n)$ for inputs of length n. Division can also be done in depth $O(\log n)$, however the circuits are not known to be logspace uniform. They are, however, $\log^2 n$-space uniform, which suffices for our purposes. See Beame, Cook, and Hoover [BCH86].

A positive integer w is *y-smooth* if all of its prime divisors are no larger than y. We define $\Psi(x, y)$ to be the number of all y-smooth integers w with $1 \le w \le x$. We will refer to the set of primes below y as the *prime base* and the y-smooth integers as simply *smooth* when x and y are understood.

We write $\pi(x)$ to denote the number of all primes less than or equal to x.

$GF(q)$ is the finite field of q elements, where $q = p^e$ for a prime p and $e \ge 1$ an integer. The prime p is called the *characteristic* of the field. A finite field is *prime* or has *prime order* if $e = 1$. In this case, $GF(q) = GF(p) \cong \mathbf{Z}/p\mathbf{Z}$. $GF(q)^*$ is the multiplicative group of the field. An element $g \in GF(q)^*$ is a *generator* if every nonzero field element can be written as a power of g. Let $a \in GF(q)^*$ and assume that g is a generator for $GF(q)^*$. Then we can write $g^x = a$ for x an integer; x is called the *index* or *discrete logarithm* of a, x is unique modulo the order of the group $GF(q)^*$ which is $q - 1$, and we write $\operatorname{ind}_g(a) = x$.

We will use $\log n$ for the logarithm of n to the base 2, and $\ln n$ for the natural logarithm.

3 Factoring

In this section we present our polylog depth probabilistic circuit for integer factoring. Since our circuit is based on Dixon's random squares method, we will first review his algorithm.

It is based on the following idea. Given a composite integer N which is not a prime power, if we can find integers a and b, $a \not\equiv \pm b \pmod N$, such that $a^2 \equiv b^2 \pmod N$, then $\gcd(a + b, N)$ is a proper divisor of N. To find a and b, we first find some smooth squares modulo N. Then we find a linear dependency among them in factored form, which leads to a perfect square. Dixon [Dix81] proved that this method works with probability at least 1/2. To make this more precise, here is an outline of Dixon's factoring algorithm.

Dixon's Factoring Algorithm

Input: positive integer N, not a prime power.

1. Choose a positive bound B, find all the primes below B, and call this the prime base. Let $k = \pi(B)$, the size of the prime base.

2. Repeat until at least $k + 1$ B-smooth squares are found:
   a) Choose a random integer r uniformly from $[2, N - 1]$.
   b) Let $s = r^2 \bmod N$. Factor s over the prime base to see if it is smooth.

3. For each smooth square s_j, $1 \le j \le k + 1$, compute the indices e_{ij} such that $s_j = \prod_{i=1}^{k} p_i^{e_{ij}}$. Let $\mathbf{v}_j = [e_{1j}, \ldots, e_{kj}]^T \bmod 2$, a column vector over $GF(2)$.

4. Form $\mathbf{A} = [\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}]$ and solve the system $\mathbf{Ax} = 0$ for a nonzero vector $\mathbf{x}$ over $GF(2)$. Since $\mathbf{A}$ has more columns than rows, such a solution must exist. This can be found using Gaussian elimination.

5. Writing $\mathbf{x} = [x_1, \ldots, x_{k+1}]^T$, form $a = \prod_{j=1}^{k+1} r_j^{x_j}$ and $b = \sqrt{\prod_{j=1}^{k+1} s_j^{x_j}}$ modulo N. Then $a^2 \equiv b^2 \pmod N$ because $s_j \equiv r_j^2$, so $\gcd(a+b, N)$ is a proper divisor of N with probability at least 1/2.

For a proof of correctness, see Dixon [Dix81] or Pomerance [Pom87]. One important question to ask at this point is: What is the probability that an integer is smooth? The following lemma from [CEP83] answers this question.

Lemma 1 (Canfield, Erdös, Pomerance [CEP83]) *Let $u = \frac{\ln x}{\ln y}$. Then there exists a constant $c > 0$ such that if $u \geq c$, $x \geq 1$, then $\Psi(x, y) > xu^{-3u}$.*

Essentially, this lemma says that if w is chosen uniformly between 1 and x, then w is y-smooth with probability at least u^{-3u}, where $u = \ln x / \ln y$. Better estimates are available for this probability, but this will suffice for our purposes.

We can now give an estimated running time for Dixon's algorithm. Roughly speaking, if we set $u = n / \log B$, then the expected running time of the algorithm is about $Bu^{3u} + B^3$, ignoring polynomial factors. The first term reflects the time needed to find about B distinct B-smooth squares, since the probability a number is B-smooth is at least u^{-3u} by Lemma 1. The second term reflects the time for Gaussian elimination to find our solution $\mathbf{x}$. Optimizing gives $B = e^{O(\sqrt{n \log n})}$. For a more exact run time analysis see Pomerance [Pom82, Pom87].

We, however, are interested in a shallow circuit. Since the random squares can be generated simultaneously, the dominant cost is the depth needed to solve the system of linear equations. Using known methods [BvzGH82, Mul87, PR85] a system of m linear equations in m unknowns over $GF(2)$ can be solved in depth $O(\log^2 m)$ using $m^{O(1)}$ gates. For our application, $m \leq B$, so this can be done in depth $O(\log^2 B)$ using at most $B^{O(1)}$ gates. If we set $B = e^{O(\sqrt{n \log n})}$ as above, we get depth $O(n \log n)$, which is clearly not good enough. If we set $B = n^{O(1)}$, a polynomial in n, the depth is fine, but then we must generate $u^{3u} n^{O(1)}$ squares, which is no longer subexponential in n. We compromise by setting $B = n^{d(n)}$ where $d(n)$ is a non-decreasing function of n. In our analysis later, we will leave $d(n)$ as a parameter, allowing us to prove unbounded fan-in circuit results along with bounded fan-in results. As we mentioned earlier, to get the polylog depth circuit for factoring, we end up using $d(n) = \log^d n$ for d a positive constant.

The Replication Technique

Before we prove our main result, we must first develop a simple *replication technique.* The idea is to increase the probability with which a probabilistic circuit computes its output correctly by duplicating that circuit several times in parallel.

Let C_n be the circuit for inputs of length n in the probabilistic circuit family C. C_n computes a function probabilistically, so let $p(n)$ be the probability that C_n's output is a correct function value.

Let $q(n)$ be our target error probability. In other words, we want to duplicate C_n enough times so that with probability at least $1 - q(n)$, at least one of the C_n subcircuits computes its output correctly.

So, we construct a new probabilistic circuit family R, where each circuit R_n is constructed as follows. First, R_n runs $m(n)$ copies of C_n in parallel, where $m(n) \geq \frac{1}{p(n)} \ln(\frac{1}{q(n)})$. Then we use a simple fan-in structure to select a C_n subcircuit whose output and correctness bits are both set, if there is one. Numbering these subcircuits from 1 to $m(n)$, we choose the lowest numbered such subcircuit for the output of R_n. If no subcircuit generated a correct function value as output with certainty, then R_n either uses the output of the first circuit which generated an output, if there is one, or chooses some "best" circuit to copy. How this choice is made we leave to the specific application of the technique. If no C_n subcircuit generated an output, then R_n's output bit is turned off.

The probability that all of the C_n subcircuits failed is at most

$$(1 - p(n))^{m(n)} \leq \left(1 - \frac{1}{p(n)^{-1}}\right)^{-p(n)^{-1} \ln q(n)} \leq e^{\ln q(n)} = q(n),$$

as desired.

The size of the circuit R_n is $O(m(n) \cdot \text{size}(C_n))$, and the depth of R_n is $O(\log m(n) + \text{depth}(C_n))$ for the bounded fan-in case or the $\text{depth}(C_n) + O(1)$ for the unbounded fan-in case.

Notice that if C_n has zero-sided error, so does R_n, and if C_n had one-sided error, so does R_n. This technique can also be applied to circuits with two-sided error, but not in all cases; R_n must be modified to somehow choose which C_n subcircuit has the "best" output, if that is possible. If it is, then the resulting circuit R_n has two-sided error.

For example, if a circuit C_n produces correct output with probability $1/n$, we can copy C_n $2n \ln n$ times using this technique to get a new circuit which produces correct output with probability $1 - 1/n^2$.

We will apply this replication technique repeatedly throughout the rest of this paper.

Theorem 2 *Let $d(n)$ be a non-decreasing function of n, $d(n) \leq O(\sqrt{n/\log n})$. There exist both*

1. *an $n/d(n)$-space uniform probabilistic circuit family with bounded fan-in of depth $O(\sqrt{n}\log^{2.5} n + d(n)^2 \log^2 n + n/d(n))$ and size $e^{O(n/d(n))}$, and*

2. *an $n/d(n)$-space uniform probabilistic circuit family with unbounded fan-in of depth $O(d(n)^2 \log^2 n)$ and size $e^{O(n/d(n))}$*

which on input N, a positive composite integer of n bits, produce as output a proper factor of N with probability at least $1/2 - o(1)$.

Proof: We will describe how to parallelize each step of Dixon's factoring algorithm.

We can safely assume that N is not a prime power, since we can compute integer roots in $\mathcal{NC}^2$ (see Bach and Sorenson [BS89]).

Let $B = n^{d(n)}$.

Step 1.

To find all the primes in the prime base, we perform trial division in parallel. This can be done in depth $O(\log B) = O(d(n)\log n)$ and size $B^{O(1)}$.

Steps 2 and 3.

Next, we choose $r \in [2, N-1]$ uniformly and set $s = r^2 \bmod N$. For each i, compute e_i such that $p_i^{e_i}$ fully divides s. Then, if $\prod_{i=1}^{k} p_i^{e_i} = s$, s is B-smooth so we output s, r, and $\mathbf{v} = [e_1, \ldots, e_k]^T$.

The total depth of this subcircuit is at most $O(d(n)\log^2 n)$, and the probability that it generates an output is at least u^{-3u} where $u = n/(d(n)\log n)$, by Lemma 1, for n sufficiently large.

We then duplicate this subcircuit creating $k+1$ groups of $(2\ln B)u^{3u}$ subcircuits. Using the replication technique, the probability this results in at least $k+1$ smooth squares is at least $1-(k+1)/B^2 = 1-o(1)$.

The size is now $e^{O(n/d(n))}$ since

$$u^{3u} \quad \leq \quad n^{3n/(d(n)\log n)} \quad \leq \quad e^{O(n/d(n))}$$

and the k and B factors are absorbed in the constant of proportionality by the bound on $d(n)$. The unbounded fan-in depth is still $O(d(n)\log^2 n)$, but the bounded fan-in depth has grown to $O(n/d(n) + d(n)\log^2 n)$.

Step 4.

Now we construct the matrix $\mathbf{A} = [\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}]$. We then search for a non-zero solution to the equation $\mathbf{Ax} = 0$ over $GF(2)$, where we have reduced the entries in $\mathbf{A}$ modulo 2. This can be done by solving the $k+1$ linear equations $\mathbf{A}_j\mathbf{y} = \mathbf{v}_j$ in parallel, where $\mathbf{A}_j$ is simply $\mathbf{A}$ with the jth column removed. One of these equations must give a solution. If we find one for some index j, then $\mathbf{x} = [y_1, \ldots, y_{j-1}, 1, y_j, \ldots, y_k]^T$ is a solution to the original system, where $\mathbf{y} = [y_1, \ldots, y_k]^T$. This can be done in depth $O(d(n)^2\log^2 n)$ and size $B^{O(1)}$ (see [BvzGH82, Mul87, PR85]).

Step 5.

Now, using our solution $\mathbf{x}$, we form

$$a = \prod_{j=1}^{k+1} r_j^{x_j} \qquad \text{and} \qquad b = \left(\prod_{j=1}^{k+1} s_j^{x_j}\right)^{\frac{1}{2}} = \prod_{i=1}^{k} p_i^{\frac{1}{2}\sum_{j=1}^{k+1} x_j e_{ij}} \qquad \text{modulo } N.$$

Since a and b are random with $a^2 \equiv b^2 \pmod{N}$, by Dixon [Dix81], we have $\gcd(a+b, N)$ as a proper factor of N with probability at least $1/2$. To compute the greatest common divisor, we use the probabilistic circuit of Adleman and Kompella [AK88] which has unbounded fan-in depth of $O(\log^2 n)$, bounded fan-in depth of $O(\sqrt{n}\log^{2.5} n)$ and size $e^{O(\sqrt{n\log n})}$. It only succeeds with probability 1/2, but it has zero-sided error, so we can copy it $2\ln n$ times and apply the replication technique to decrease its error probability to $1/n$.

The total size of this integer factoring circuit is $e^{O(n/d(n))}$. The unbounded fan-in depth is dominated by solving the linear equations (Step 4) at $O(d(n)^2 \log^2 n)$. The bounded fan-in depth is $O(\sqrt{n} \log^{2.5} n + d(n)^2 \log^2 n + n/d(n))$.

Since our circuit is composed of large numbers of relatively small circuits together with simple fan-in structures, our circuit is clearly $n/d(n)$-space uniform. ■

If we set $d(n) = \log^d n$ for $d > 0$ a constant, we get an $O(\log^{2d+2} n)$ depth circuit of size $e^{O(n/\log^d n)}$, which is a solution to Adleman and Kompella's open problem [AK88]. The bounded fan-in version has optimal depth for $d(n) = n^{1/3}/\log^{2/3} n$ giving a depth of $O((n \log n)^{2/3})$ and size $e^{O((n \log n)^{2/3})}$.

Also notice that this factoring circuit has zero-sided error. If a proper divisor is found, then that is the output, and there is no doubt about its correctness. If only a trivial divisor is found, then the circuit simply turns off its output bit.

Complete Factorizations

Now we are interested in constructing a probabilistic circuit which takes as input an n-bit integer N and computes as output a complete factorization of N. Our motive is an application in the next section: we need this to compute discrete logarithms.

Once we have an algorithm to find a nontrivial divisor of a composite number, creating a sequential algorithm to completely factor that number is relatively easy. Our task is a bit more complex; we want to find a complete factorization in parallel. One approach might be to attempt to show that the factoring algorithm splits N into two divisors of roughly the same size; then we can factor these two divisors in parallel and repeat. The goal would be to show that this process would halt in $O(\log n)$ levels, where each level is the depth of one factoring circuit. This approach might work, however we opt for something a little simpler; by replicating the factoring circuit many times in parallel, there is a good chance that each prime power divisor appears as the output of one of these circuits.

Let $N = \prod_{i=1}^{m} q_i^{e_i}$ be the prime factorization of N. Let $B = n^{d(n)}$ as before. Assume that we removed all prime factors below B from N. Then, the number of distinct primes left is $m \leq \log_B N = n/(d(n) \log n)$.

Let s be a quadratic residue modulo N. Then modulo each prime power divisor $q_i^{e_i}$, s has exactly 2 square roots (we can safely assume $B > 2$ so that N is now odd). Thus, modulo N, s has exactly 2^m square roots. Let a and b be arbitrary square roots of s. Then $\gcd(a-b, N)$ will be the product of those prime powers $q_i^{e_i}$ for which $a \equiv b \pmod{q_i^{e_i}}$. Similarly, $\gcd(a+b, N)$ gives those prime powers where $a \equiv -b$.

Let d be the output of a factoring circuit. Then the probability d or N/d is a prime power is at least 2^{-m}, which is no less than $2^{-n/(d(n) \log n)}$. This observation is merely an extension of Dixon's theorem from [Dix81].

Corollary 3 *There exist $n/d(n)$-space uniform probabilistic boolean circuits of size $e^{O(n/d(n))}$, with unbounded fan-in depth $O(d(n)^2 \log^2 n)$ or bounded fan-in depth $O(\sqrt{n} \log^{2.5} n + d(n)^2 \log^2 n + n/d(n))$, which, when given as input a positive integer N of n bits in length, output a complete factorization of N as the product of prime powers, with probability $1 - o(1)$.*

Proof: We will now describe how to construct the circuit. Correctness will follow from the discussion above.

First we will construct a subcircuit which produces a prime factor of N with nonzero probability. This subcircuit is composed of four steps.

1. We remove all prime factors smaller than B from N.
2. We factor N using the factoring circuit from Theorem 2. By the discussion above, the probability that the result is a prime power is at least $2^{-n/(d(n)\log n)}$.
3. We next take a smallest integer root of the two divisors generated in the previous step using the $\mathcal{NC}$ root computation algorithm from [BS89]. (If one of those divisors was a prime power, we now have a prime divisor.)
4. We now take the integer roots, which we hope are primes, from the previous step and perform a prime test on them. To do this, we attempt to factor each $4n^2$ times in parallel. By the replication technique and Theorem 2, if we do not have a prime, then with probability at least $1 - e^{-n^2}$, for sufficiently large n, we will discover this fact by finding a proper divisor.

So this subcircuit will give a prime divisor of N as its output with probability $2^{-n/(d(n)\log n)}$. Notice that this subcircuit has two-sided error. The probability that it says it has found a prime divisor and is wrong is at most e^{-n^2}. This is small enough that we can essentially ignore it.

Second, we copy this subcircuit $(3 \log n)2^{n/(d(n)\log n)}$ times using the replication technique to get a circuit which finds some prime divisor of N with probability at least $1 - 1/n^3$. Our "best" subcircuit whose output gets copied is the first one which generates an output.

Third, we copy this circuit n^2 times to get n^2 prime divisors of N, obviously with some of them duplicated. By the above discussion and Dixon's theorem [Dix81], each prime is equally likely to occur, so clearly the probability that one of them is omitted from this list of n^2 primes is $o(1)$. We then sort this list and remove duplicates. We finish by raising the remaining primes to appropriate powers and computing their product. This should equal N. The probability that this results in a complete factorization is $1 - o(1)$.

The size and depth of the resulting circuit are dominated by the size and depth of the integer factoring circuits, with slightly larger constants of proportionality.

Note that there is a chance this circuit will generate output it believes to be correct, and yet this output may not be correct. Specifically, it may give N as the product of several factors, some of which may not be prime. The chance this occurs is $o(1)$. ∎

Notice that this circuit has two-sided error in the sense that it never knows if its output is indeed a complete factorization or not, so its correctness bit is always off.

4 Discrete Logarithms

In this section we give probabilistic circuits of polylog depth and subexponential size to compute discrete logarithms over certain finite fields. First, we outline the index-calculus method for computing discrete logarithms, which was rigorously analyzed by Adleman [Adl79]. Second, we present our circuit for prime finite fields. Last, we sketch the construction of a circuit for finite fields of small characteristic.

The index-calculus method for computing discrete logarithms has many algorithmic similarities to the random squares method for factoring integers. Thus, we will simply modify our integer factoring circuit, in a sense, to compute discrete logarithms. Now we will outline Adleman's index-calculus algorithm for the field $GF(p)$.

Adleman's Discrete Logarithm Algorithm

Input: p a prime, g a generator for $(\mathbf{Z}/p\mathbf{Z})^*$, and $a \in [2, p-1]$.

Phase I:

1. Find all the primes below B. Call this set of primes the prime base. Let $k = \pi(B)$. Let $p_1, \ldots, p_k$ be the primes of the prime base, and write $p_0 = 1$. Let $l = 2\log k + 3$.

2. For each i, $1 \leq i \leq k$, repeat (a) and (b) until at least l distinct B-smooth numbers are found, and for $i = 0$ until at least kl such numbers are found:
   a) Choose an integer r uniformly from $[1, p-1]$.
   b) Let $s = g^r p_i \bmod p$. Factor s over the prime base to see if it is smooth.

3. For each smooth number s_j, $1 \leq j \leq 2kl$, compute the indices e_{ij} such that $s_j = \prod_{i=1}^{k} p_i^{e_{ij}}$. Let $\mathbf{v}_j = [e_{1j}, \ldots, e_{kj}]$. If $s_j = g^{r_j} p_i$, $i > 0$, then subtract 1 from the ith entry in $\mathbf{v}_j$.

4. Form $\mathbf{A} = [\mathbf{v}_1, \ldots, \mathbf{v}_{2kl}]^T$ and $\mathbf{r} = [r_1, \ldots, r_{2kl}]^T$. With probability at least $1 - 1/2k$, $\mathbf{A}$ is a rectangular matrix of full rank (see [Pom87]). Find a solution to the equation $\mathbf{Ax} = \mathbf{r} \pmod{p-1}$. This can be done by factoring $p-1$ and using the Chinese remainder theorem. Solutions are found modulo each prime power factor and combined. Writing $\mathbf{x} = [x_1, \ldots, x_k]^T$, we have $\mathrm{ind}_g(p_i) = x_i$.

Phase II:

1. Choose $\rho \in [1, p-1]$ until $\alpha = ag^\rho \bmod p$ is B-smooth.

2. Factor α over the prime base to get the equation $\alpha = \prod_{i=1}^{k} p_i^{e_i}$. Taking logs of both sides then gives $\rho + \mathrm{ind}_g(a) = \sum_{i=0}^{k} e_i \mathrm{ind}_g(p_i)$, which determines $\mathrm{ind}_g(a)$.

For a rigorous analysis of the running time and correctness see Adleman [Adl79], Odlyzko [Odl84], or Pomerance [Pom87]. If $B = L(n)^c$ for some small constant c, then the sequential running time of this algorithm is $e^{O(\sqrt{n \log n})}$, just like integer factoring.

Theorem 4 *Let $d(n)$ be a non-decreasing function of n, $d(n) \le O(\sqrt{n/\log n})$. There exist both*

1. *an $n/d(n)$-space uniform probabilistic bounded fan-in circuit family with depth $O(d(n)^2\sqrt{n}\log^{4.5} n + n/d(n))$ and size $e^{O(n/d(n))}$, and*
2. *an $n/d(n)$-space uniform probabilistic unbounded fan-in circuit family with depth $O(d(n)^2 \log^4 n)$ and size $e^{O(n/d(n))}$*

which, on inputs p a prime, g a generator of $GF(p)^*$, and $a \in GF(p)^*$ of total length n in binary, solves the discrete logarithm problem in $GF(p)$ with probability $1 - o(1)$.

Proof: (sketch) The structure of the proof of this theorem strongly resembles that of Theorem 2, so we merely sketch the differences. Again, let $B = n^{d(n)}$.

Steps 2 and 3 compute random powers of the generator and also multiply by primes from the prime base. This uses Adleman and Kompella's modular exponentiation circuit for prime modulus [AK88], which has unbounded fan-in depth $O(\log^3 n)$, bounded fan-in depth $O(\sqrt{n}\log^{3.5} n)$, size $e^{O(\sqrt{n \log n})}$, and has zero-sided error. We apply the replication technique to reduce its error probability from $1/2$ to $o(1)$.

Once the system of linear equations is assembled, in Step 4 a solution is found modulo $p-1$. To do this, we first must factor $p-1$ using the circuit of Corollary 3, and solve the system by finding solutions modulo each prime power factor of $p-1$ and constructing the overall solution using the Chinese remainder theorem. These prime power factors may be quite large, so inverses must be computed using Adleman and Kompella's integer greatest common divisor circuit [AK88]. Hensel lifting may also be required; this can be done in $O(\log^2 n)$ depth using a polynomial number of processors (see Hardy and Wright [HW79, chapter VIII]). For more details in the sequential case, see Pomerance [Pom87] or Odlyzko [Odl84].

Finally, we add Phase II, which is simply choosing $\rho \in [1, p-1]$ until $g^\rho a$ is B-smooth. We can do this in parallel using techniques similar to what we use to generate random powers of the generator. Finally, we factor over the prime base to get a solution.

The overall size of the circuit is $e^{O(n/d(n))}$. For the unbounded fan-in case, the depth is dominated by the depth of the subcircuit for solving the system of linear equations modulo $p-1$, which is $O(d(n)^2\log^4 n)$. For the bounded fan-in case, the total depth is at most $O(n/d(n) + d(n)^2\sqrt{n}\log^{4.5} n)$. ■

As in the case of the factoring circuit, setting $d(n) = \log^d n$ for some constant $d > 0$ gives a polylog depth, subexponential size circuit for computing discrete logarithms. The bounded fan-in version has optimal depth for $d(n) = n^{1/6}/\log^{3/2} n$ for a total depth of $O(n^{5/6}\log^{3/2} n)$ and size $e^{O(n^{5/6}\log^{3/2} n)}$.

Notice that this circuit has zero-sided error, even though it uses a subcircuit for complete factorization, which has two-sided error.

We can also apply these methods to finite fields of small characteristic; we will now outline the changes necessary.

Our finite field is $GF(q) = GF(p^n)$. We will assume that we are given a representation for the field of the form $GF(p)[x]/(f)$, where p is the characteristic of the field, and $f(x)$

is a monic irreducible polynomial of degree n over $GF(p)$. Elements of the field, then, are represented as polynomials of degree less than n over $GF(p)$.

There are two main differences in the algorithm. The first is that an element is smooth if it factors as a polynomial into irreducibles of small degree. Thus, the prime base is the set of all monic irreducible polynomials with degree less than some bound $m \leq n$. We need something analogous to Lemma 1 to bound from below the probability that a randomly chosen polynomial is smooth. Odlyzko [Odl84] does give such a result. See also Pomerance [Pom87]. The second difference is that we need modular exponentiation in the field. This problem was shown to be in $\mathcal{NC}$ by Fich and Tompa [FT88], so that is not a difficulty.

Solving the system of linear equations is largely the same. Thus, we have a result similar to Theorem 4 for finite fields of small characteristic.

The computation of discrete logarithms for finite fields of moderate size characteristic is still largely an open problem in the sequential case. It seems that any new results there would imply parallel algorithms of the type given here, as long as the underlying methods use the index-calculus technique.

5 Conclusion

We have presented polylog depth probabilistic circuits to factor integers and compute discrete logarithms over finite fields of prime size or small characteristic, but there are still many unresolved questions.

Is it possible to extend our discrete logarithm results to fields of intermediate size characteristic? El Gamal gave sequential algorithms for the case $GF(p^m)$ for m fixed [EG85b, EG85a]. It might be possible to parallelize these algorithms using our methods.

Other methods are known for integer factoring and discrete logarithms aside from the difference of squares and the index-calculus methods. Might these techniques give polylog depth, subexponential size circuits?

Our circuits, though subexponential in size, are still quite large. Might it be possible to reduce these very large circuits to something around $e^{O(\sqrt{n \log n})}$ and still maintain a polylog depth?

Finally, Adleman and Kompella [AK88] suggested that their results might lead to sublinear space bounds for computing integer gcd's and modular exponentiation. Similarly, our results might also lead to sublinear space bounds for factoring and discrete logarithms. Intuitively, the parallel computation thesis, which asserts that sequential space and parallel depth (or time) are equivalent in some sense, implies that all of these problems might have some kind of sublinear space bounds. And in fact, if these circuits were deterministic instead of probabilistic, this would be the case. Unfortunately, the fact that these circuits are probabilistic seems to be more than simply an inconvenience in proving sublinear space bounds. We will now outline how a sublinear space bound proof would proceed if these circuits were in fact deterministic, followed by an explanation of the difficulty involved in applying this approach to probabilistic circuits.

Borodin [Bor77] proved that, given an $O(S(n))$-space uniform circuit of depth $O(S(n))$ to compute a function f for inputs of length n, then in fact f is computable in space $O(S(n))$.

The basic construction is to evaluate the circuit using a Turing machine, and the uniformity condition is used to give a subroutine for constructing the needed pieces of the circuit. The total space used is proportional to the stack space needed for evaluating the circuit, which is proportional to its depth, plus the space needed to construct the circuit pieces, which is bounded by the uniformity condition.

Adleman and Kompella's circuits for integer gcd and modular exponentiation are $n^{1/2+o(1)}$-space uniform of depth $n^{1/2+o(1)}$, our factoring circuit is $n^{2/3+o(1)}$-space uniform of depth $n^{2/3+o(1)}$, and our discrete logarithm circuit for prime fields is $n^{5/6+o(1)}$-space uniform of depth $n^{5/6+o(1)}$. If these were deterministic circuits, we could apply Borodin's methods to get corresponding sequential space bounds of $n^{1/2+o(1)}$, $n^{2/3+o(1)}$, and $n^{5/6+o(1)}$.

Converting these results to the standard probabilistic Turing machine model as given by Gill [Gill77] apparently does not work. The reason is that, during the evaluation of the circuit by the Turing machine, the output of any one gate may be needed more than once. Each time the gate's output is evaluated, we must get the same answer. Thus, if that gate performs a coin toss, the same result must be used each time, which seems to imply that the result of the toss must be saved, which in turn means storing the result on tape. This is fine, but the number of coin tosses used by these circuits is much too large for this approach to give sublinear space bounds. One way around this problem is to redefine the probabilistic Turing machine which evaluates the circuit to use a read-only, two-way tape for its source of random coin tosses. But then we have transformed the original problem to that of analyzing the differences between two models of probabilistic Turing machines.

Attempts to prove a sublinear nondeterministic space bound will run into a similar situation. So using this approach to prove sublinear space bounds for these problems seems difficult at best, whether it be on a probabilistic or nondeterministic Turing machine.

Currently, no bound better than linear, deterministic space is known for any of these problems, except for Chor and Goldreich's CREW PRAM algorithm [CG85], which gives an $O(n \log\log n/\log n)$ deterministic space bound for integer GCD after applying Borodin's methods.

Acknowledgements

I would like to thank my advisor Eric Bach and also Anne Condon, João Meidânis, and Victor Shoup for their many helpful comments and suggestions.

References

[Adl79] L. M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *20th Ann. IEEE Symp. Foundations of Comp. Sci.*, pages 55–60, 1979.

[AK88] L. M. Adleman and K. Kompella. Using smoothness to achieve parallelism. In *20th Ann. ACM Symp. Theory Comp.*, pages 528–538, 1988.

[BCH86] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15:994–1003, 1986.

[BK83] R. P. Brent and H. T. Kung. Systolic VLSI arrays for linear-time GCD computation. In F. Anceau and E. J. Aas, editors, *Proceedings of VLSI '83*, pages 145–154. Elsevier, 1983.

[Bor77] A. Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6(4):733–744, 1977.

[BS89] E. Bach and J. Sorenson. Sieve algorithms for perfect power testing. Technical Report #852, Computer Sciences Department, University of Wisconsin-Madison, 1989.

[BvzGH82] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. *Inform. Contr.*, 52:241–256, 1982.

[CEP83] E. R. Canfield, P. Erdös, and C. Pomerance. On a problem of Oppenheim concerning "Factorisatio Numerorum". *J. Number Theory*, 17:1–28, 1983.

[CG85] B. Chor and O. Goldreich. An improved parallel algorithm for integer GCD. Technical report, MIT Laboratory for Computer Science, 1985. *Algorithmica*, to appear.

[Cook85] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Inform. Contr.*, 64:2–22, 1985.

[Dix81] J. Dixon. Asymptotically fast factorization of integers. *Math. Comp.*, 36(153):255–260, 1981.

[EG85a] T. El Gamal. On computing logarithms over finite fields. In *Proceedings of Crypto '85*, pages 396–402, 1985.

[EG85b] T. El Gamal. A subexponential-time algorithm for computing discrete logarithms over $GF(p^2)$. *IEEE Trans. Inform. Theory*, 31(4):473–481, 1985.

[FT88] F. Fich and M. Tompa. The parallel complexity of exponentiating polynomials over finite fields. *J. ACM*, 35(4):651–667, 1988.

[Gill77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM J. Comput.*, 6:675–695, 1977.

[HW79] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, 5th edition, 1979.

[KMR87] R. Kannan, G. Miller, and L. Rudolph. Sublinear parallel algorithm for computing the greatest common divisor of two integers. *SIAM J. Comput.*, 16(1):7–16, 1987.

[Kob87] N. Koblitz. *A Course in Number Theory and Cryptography.* Springer-Verlag, New York, 1987.

[KR88] R. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report UCB/CSD 88/408, Computer Science Division, University of California, 1988. To appear in *Handbook of Theoretical Computer Science*, North-Holland.

[Mul87] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7(1):101–104, 1987.

[Odl84] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In *Proceedings of Eurocrypt '84*, 1984.

[Pom82] C. Pomerance. Analysis and comparison of some integer factoring algorithms. In H. W. Lenstra Jr. and R. Tijdeman, editors, *Computational Methods in Number Theory*, pages 89–141. 1982.

[Pom87] C. Pomerance. Fast, rigorous factorization and discrete logarithm algorithms. In D. S. Johnson, A. Nishizeki, A. Nozaki, and H. S. Wilf, editors, *Discrete Algorithms and Complexity: Proceedings of the Japan-US Joint Seminar*, pages 119–143. Academic Press, London, 1987.

[PR85] V. Pan and J. Reif. Efficient parallel solution of linear systems. In *17th Ann. ACM Symp. Theory Comp.*, pages 143–152, 1985.

[Val89] B. Vallée. Provably fast integer factoring with quasi-uniform small quadratic residues. In *21st Ann. ACM Symp. Theory Comp.*, pages 98–106, 1989.

[vzG87] J. von zur Gathen. Computing powers in parallel. *SIAM J. Comput.*, 16:930–945, 1987.