PARALLEL POWERING

Joachim von zur Gathen
Department of Computer Science
University of Toronto

Abstract

A fast parallel computation for large powers of an integer modulo another integer is presented, assuming that the modulus has only small prime factors.

1. Introduction

Consider the problem of computing $a^b \bmod m$ in parallel, where a, b, m are n-bit integers. This problem comes up as a subroutine in many computational problems, e.g. factoring integers, primality tests, cryptographic schemes, and factoring polynomials over finite fields. The method of "repeated squaring" does not yield fast parallel computations, i.e. of parallel time $(\log n)^{O(1)}$.

We present boolean circuits for this problem. Provided that the modulus m has only small prime factors $p \le n$, the depth is $O(\log^2 n)$ with polynomial size.

In section 2 we present the technique to be used in the setting of polynomials. Section 3 has some auxiliary parallel algorithms, for factoring numbers and Chinese remaindering. The fact that factoring can be achieved fast in parallel puts the constraint of "small prime factors" into perspective: it is a severe restriction, and for many applications the more interesting case is that of large prime factors. However, the present result is the first one that provides an exponential parallel speed-up of the sequential methods for modular integer powering. A slightly different problem - computing in parallel the high-order bits of a large power of an integer - is considered in Alt [1984].

In section 4 we then adapt the polynomial technique to the case of integers. The resulting parallel algorithm for computing a_b mod m requires a special case of this problem to be "hard-wired", and yields a P-uniform family of boolean circuits of depth $O(\log^2 n)$.

2. Powering polynomials

The idea of the integer powering algorithm is based on a method which is quite natural for polynomials. In this section, we explain this method and its inherent limitations.

We work over an arbitrary ground field F of characteristic zero. The model of computation (for this section only) is an arithmetic circuit with inputs and constants from F, and +, -, *, / as operations. Consider the following four problems, where $u \in F$, and $f, g \in F[x]$ of degree n are inputs, and b is an n-bit integer.

POWER-1: Compute u^b.

POWER-2: Compute $f^b \bmod g$.

POWER-3: Compute $f^b \bmod x^n$.

POWER-4: Compute $f^b \bmod x^n$, if f has constant term 1.

Theorem 2.1. Problems POWER-1, POWER-2, POWER-3 cannot be computed on any family of arithmetic circuits of depth $(\log n)^{O(1)}$.

Proof. It is well-known that the rational function accociated with a gate of depth d (in an arithmetic circuit) has degree at most 2^{d-1} (Kung [1976]). Hence the depth d_n of any circuit computing u^{2^n} satisfies $d_n > n$, and the claim for POWER-1 follows. POWER-1 is reducible in depth $O(\log n)$ to both POWER-2 and POWER-3. ☐

This result does not extend to finite fields e.g. if $F = \mathbb{Z}_2$ then

$$a^b = a, \ \forall b \ge 1, \ a \in F.$$

Allowing only constant term 1 in f rules out the above reduction from POWER-1 to POWER-3, and in fact we have the following fast parallel algorithm for POWER-4.

Algorithm PARALLEL POLYNOMIAL POWERING

Input : $f \in F[x]$, $\deg f \le n$, f has constant term 1, char $F = 0$, b a n-bit integer.

Output : $f^b \bmod x^n$

0272-5428/84/0000/0031$01.00 © 1984 IEEE

1. Compute $h = \log f \mod x^n$.
   Consider f as a power series with only finitely many non-zero terms and compute the first n terms of $\log f = \sum_{i \geq 1} \frac{(-1)^{i+1}}{i} (f-1)^i$.
2. Compute $b \cdot h$
3. Compute $\exp(b \cdot h) \mod x^n$.
   As in step 1, using the fact that $\exp g = \sum_{i \geq 0} \frac{1}{i!} g^i$, where g has zero constant term; i.e. we compute $\sum_{0 \leq i \leq n} \frac{1}{i!} g^i \mod x^n$.

Theorem 2.2. Algorithm PARALLEL POLYNOMIAL POWERING can be implemented on an arithmetic circuit of depth $O(\log n)$.

Proof. In steps 1 and 3 we have to compute $h^i \mod x^n$ for various polynomials h (with $h(0) = 0$ and $1 \leq i \leq n$). This can obviously be done in depth $O(\log^2 n)$, and in fact in depth $O(\log n)$ by Reif [1984] (assuming roots of unity are available), and Eberly [1984] (for arbitrary fields). □

Theorem 2.1 shows that the condition "constant term 1" for substituting into the logarithm power series is not just technical, but in our context related to the parallel computational complexity.

3. Chinese remaindering

To solve the powering problem, we have to present parallel computations for some other problems, which may be of independent interest. Inputs are the various n-bit numbers denoted as $a, b, a_1, \ldots, a_n, m, m_1, \ldots, m_n$, and output an n-bit number c as below:

POWER : $c \equiv a^b \bmod m$,

MODINV : $\begin{cases} ac \equiv 1 \bmod m & \text{if } gcd(a,m) = 1, \\ c = 0 & \text{otherwise,} \end{cases}$

("modular inverse")

LAGRANGE : $c \equiv 0 \bmod m_2 \cdots m_n$,

$$c \equiv \begin{cases} 1 \bmod m_1 & \text{if } gcd(m_1, m_2 \cdots m_n) = 1, \\ 0 \bmod m_1 & \text{otherwise,} \end{cases}$$

("Lagrange interpolation coefficient"),

CHINREM : If $gcd(m_i, m_j) = 1$ for all $i \neq j$,
then $c \equiv a_i \bmod m_i$ for all i.
Otherwise c is arbitrary.
("Chinese remainder algorithm")

FACTOR : compute the prime factors of m (with multiplicities).

Using DIV for the problem of integer division with remainder, we have an NC^1-reduction (see Cook [1983] for terminology):

$$\text{CHINREM} \leq \text{LAGRANGE} \leq \text{MODINV} + \text{DIV}.$$

It follows by using an inverse d of $m_2 \cdots m_n \bmod m_1$; then $c = d \cdot (m_2 \cdots m_n)$ is sufficient. The iterated product $m_2 \cdots m_n$ is reducible to division (see Beame-Cook-Hoover [1984]).

At the present time, none of these problems is known to be in NC - i.e solvable in depth $(\log n)^{O(1)}$ - or complete for P. All problems - except possibly FACTOR - are in P.

In their work on Abelian permutation group membership, McKenzie-Cook [1983] had to solve systems of linear congruences with the constraint that the modulus m was known to have only small prime power factors, i.e. $m = \prod p_i^{e_i}$ with $p_i^{e_i} = O(n)$ for all i. They then showed that this problem is in RNC^3. We now use "SF" to denote the condition that $m, m_1, \ldots, m_n$ have "small factors", i.e. only prime factors $p \leq n$. This is a little more generous than the condition in McKenzie-Cook [1983].

We assume that there exist polynomial-size circuits of depth $\text{div}(n)$ that compute the division with remainder for n-bit integers. By Beame-Cook-Hoover [1984] $\text{div}(n) = O(\log n)$ for P-uniform circuits, and $\text{div}(n) = O(\log n \log\log n)$ for log-space uniform circuits, by Reif [1984].

Theorem 3.1. The following problems can be computed by boolean circuits of polynomial size and the stated depth:

(i) SF-FACTOR in depth $O(\text{div}(n))$,

(ii) SF-MODINV, SF-LAGRANGE, and SF-CHINREM in depth $O(\log n \cdot \text{div}(n))$.

Proof. We leave away the "SF-". First consider FACTOR. The algorithm is obvious: For each number $p \leq n$, test whether p is prime and determine its multiplicity in m. This can be performed in depth $\text{div}(n)$, by computing $p, p^2, p^3, \ldots, p^n$ and testing for each i whether p^i divides m.

Now consider MODINV in the special case where $m = p^e$, p prime. We have the following algorithm:

1. Test if $p \mid a$, and return $c = 0$ if "yes".
2. Find by exhaustive search $c_0 \in \mathbf{N}$ such that
$$1 \leq c_0 < p$$
$$ac_0 \equiv 1 \bmod p.$$
3. Newton iteration: Set $k = \lceil \log_2 e \rceil$, and compute $c_1, \ldots, c_k$ as follows:
$$c_i = 2c_{i-1} - ac_{i-1}^2 \bmod p^{2^i},$$
$$1 \leq c_i < p^{2^i}.$$

(Then $ac_i \equiv 1 \bmod p^{2^i}$, by the Newton formula

$$ac_i - 1 \equiv \frac{1}{c_{i-1}^2}(c_i - (2c_{i-1} - ac_{i-1}^2) - (c_i - c_{i-1})^2) \bmod p^{2^i}.)$$

4. Return $c = c_k \bmod p^e$.

The circuit depth for this algorithm is dominated by step 3:

$$O(\log e \cdot (\log n + \mathrm{div}(n))) = O(\log n \, \mathrm{div}(n)).$$

Next we consider LAGRANGE. If $m_1 = p^e$ for a small prime p, then we can solve LAGRANGE in depth $O(\log n \, \mathrm{div}(n))$ by the reduction to MODINV + DIV. For the general case $m = p_1^{e_1} \cdots p_r^{e_r}$, with small pairwise distinct primes $p_1, \ldots, p_r$ (and no p_i dividing $m_2 \cdots m_n$), we compute $c_1, \ldots, c_r$ such that

$$c_i \equiv \begin{cases} 1 \bmod p_i^{e_i} \\ 0 \bmod p_1^{e_1} \ldots p_{i-1}^{e_{i-1}} p_{i+1}^{e_{i+1}} \ldots p_r^{e_r} m_2 \ldots m_n. \end{cases}$$

Each c_i is given by one of the special LAGRANGE problems already solved, and

$$c = c_1 + \cdots + c_r$$

solves the current problem. Therefore LAGRANGE and CHINREM can be solved in depth $O(\log n \, \mathrm{div}(n))$.

Now for the general case of MODINV, where $m = p_1^{e_1} \cdots p_r^{e_r}$ with small primes p_i, we have the algorithm:

1. for $i \le r$ compute c_i such that $ac_i \equiv 1 \bmod p_i^{e_i}$,
2. compute c such that for all $i \le r$ $c \equiv c_i \bmod p_i^{e_i}$.

Then

$$\forall i \le r \quad ac \equiv ac_i \equiv 1 \bmod p_i^{e_i}$$
$$\Longrightarrow ac \equiv 1 \bmod m. \quad \square$$

4. Powering integers

Now we consider the problem POWER: computing $a^b \bmod m$. We adopt the algorithm PARALLEL POLYNOMIAL POWERING to solve this problem in P-uniform depth $O(\log^2 n)$ if m has only small prime factors.

Let $1 \le q \le s$ be integers, and consider the truncated exponential and logarithmic power series:

$$E_q(y) = \sum_{0 \le i \le q} \frac{1}{i!} y^i \in \mathbf{Q}[y],$$

$$L_s(x) = \sum_{1 \le i \le s} \frac{(-1)^{i+1}}{i} x^i \in \mathbf{Q}[x].$$

(Thus $E_q(y) \equiv exp(y) \bmod y^{q+1}$, and $L_s(x) \equiv log(1+x) \bmod x^s$.) The next three lemmas establish the usefulness of these series for integer powering. Lemma 4.1 has implicitly been used in Theorem 2.2.

Lemma 4.1. Let $0 \le q \le s$ and $b \in \mathbf{N}$. Then we have in $\mathbf{Q}[x]$:

$$E_q(bL_s(x)) \equiv (1+x)^b \bmod x^{q+1}.$$

Proof. We prove the lemma by induction on q. The case $q = 0$ (or $q = 1$) is easily checked. For the inductive step, we have

$$\frac{\partial}{\partial x}(E_q(bL_s(x)) - (1+x)^b)$$
$$= \frac{\partial E_q}{\partial y}(bL_s(x)) \cdot b \cdot \frac{\partial}{\partial x}\Big(\sum_{1 \le i \le s} \frac{(-1)^{i+1}}{i} x^i\Big) - b(1+x)^{b-1}$$
$$= \Big(\sum_{0 \le i \le q-1} \frac{1}{i!}(bL_s(x))^i\Big) \cdot b \cdot \Big(\sum_{1 \le i \le s} (-1)^{i+1} x^{i-1}\Big) - b(1+x)^{b-1}$$
$$\equiv b(1+x)^{b-1}\Big((1+x)\Big(\sum_{1 \le i \le s} (-1)^{i-1} x^{i-1}\Big) - 1\Big)$$
$$= b(1+x)^{b-1} \cdot (-1)^{s-1} \cdot x^s \equiv 0 \bmod x^q,$$

where we have used the induction hypothesis in the third transformation. Now one verifies that

$$(E_q(bL_s(x)) - (1+x)^b)(0) = E_q(bL_s(0)) - 1 = 0,$$

and the claim follows. $\square$

From now on we will always have $s = q$. The idea is now essentially to substitute a prime number p for x in the congruence of Lemma 4.1, and hope to obtain a similar congruence in $\mathbf{Z}$. The problem is that the left hand side has denominators in the terms of order greater than n. Lemma 4.2 bounds the multiplicity of p in integers of the form $i!$ and $j_1 \cdots j_i$, and Lemma 4.3 states that we do not lose too much of the initial precision $q+1$ given by Lemma 4.1.

We will have to consider $t = \frac{q!}{p^r}$, where $p^r \mid q!$ and $p^{r+1} \nmid q!$. We write $t = \frac{q!}{gcd(q!, p^q)}$, and note that $gcd(p,t) = 1$.

Lemma 4.2. Let $d, i, p, t \in \mathbf{N}$ such that p is prime, and $t = \frac{q!}{gcd(q!, p^q)}$. Then

(i) $i! \mid p^i t$,

(ii) If $h = \left\lfloor \frac{d}{p} \right\rfloor$, and $j_1, \ldots, j_i \ge 1$ with $j_1 + \cdots + j_i \le d$, then

$$j_1 \cdots j_i \mid p^h t^i.$$

Proof. (i) The multiplicity of p in $i!$ is

$$\left\lfloor \frac{i}{p} \right\rfloor + \left\lfloor \frac{i}{p^2} \right\rfloor + \cdots < \frac{i}{p} + \frac{i}{p^2} + \cdots = \frac{i}{p} \cdot \frac{p}{p-1}$$

$$= \frac{i}{p-1} \leq i.$$

(ii) We have to bound the multiplicity of p in $j_1 \cdots j_i$. We use induction on d. The base case $d < p$ is clear. We can assume that d and each j_s is divisible by p, since otherwise the claim follows by induction. Furthermore, we can assume that

$$\frac{j_1}{p}, \ldots, \frac{j_m}{p} \text{ are not divisible by } p,$$

$$\frac{j_{m+1}}{p}, \ldots, \frac{j_i}{p} \text{ are divisible by } p,$$

for some m, $0 \leq m \leq i$. Then

$$m \leq i \leq \frac{d}{p} = h,$$

$$p^2(i-m) \leq j_{m+1} + \cdots + j_i \leq d - pm = p(h-m),$$

$$i \leq m + \frac{h-m}{p},$$

$$j_1 \cdots j_m \mid p^m t^m,$$

$$\frac{j_{m+1}}{p} + \cdots + \frac{j_i}{p} \leq h - m.$$

Set $l = \left\lfloor \frac{h-m}{p} \right\rfloor$. By the induction hypothesis,

$$\frac{j_{m+1}}{p} \cdots \frac{j_i}{p} \mid p^l t^{i-m},$$

$$j_{m+1} \cdots j_i \mid p^{l+i-m} t^{i-m},$$

$$j_1 \cdots j_i \mid p^{l+i} t^i,$$

$$l+i \leq \frac{h-m}{p} + m + \frac{h-m}{p} = h - (h-m)\left(1 - \frac{2}{p}\right) \leq h.$$

(In fact, the exponent $h = \frac{d}{p}$ is required when $j_1 = \cdots = j_i = p$ and $d = pi$.) □

Lemma 4.3. Let $b, p, q, t, z \in \mathbf{N}$, $k = \left\lfloor \frac{q+1}{p} \right\rfloor$ such that p is prime, $t = \frac{q!}{gcd(q!, p^q)}$, and $p \mid b$ and $p \mid z$. Then

$$t^{q+1} E_q(bL_q(z)) \in \mathbf{Z},$$

$$t^{q+1} E_q(bL_q(z)) \equiv t^{q+1}(1+z)^b \bmod p^{q-k+1}.$$

Proof. Write

$$E_q(bL_q(x)) = \sum_{0 \leq d \leq q^2} e_d x^d \in \mathbf{Q}[x]$$

with $e_d \in \mathbf{Q}$, and

$$S(x) = \sum_{0 \leq d \leq q} e_d x^d \in \mathbf{Q}[x].$$

From Lemma 4.1, we know that

$$x^{q+1} \mid E_q(bL_q(x)) - (1+x)^b \text{ in } \mathbf{Q}[x]$$

and hence

$$x^{q+1} \mid S(x) - (1+x)^b \text{ in } \mathbf{Q}[x],$$

$$S(x) \in \mathbf{Z}[x],$$

$$\exists\, g(x) \in \mathbf{Z}[x] \quad g(x) \cdot x^{q+1} = S(x) - (1+x)^b,$$

$$S(z) \equiv (1+z)^b \bmod p^{q+1},$$

$$t^{q+1} S(z) \equiv t^{q+1}(1+z)^b \bmod p^{q+1}.$$

It remains to prove that

$$t^{q+1} e_d z^d \equiv 0 \bmod p^{q-k+1}$$

for any d with $q < d \leq q^2$.

$$E_q(bL_q(x)) = \sum_{0 \leq i \leq q} \frac{1}{i!} \left(b \sum_{1 \leq j \leq q} \frac{(-1)^{j+1}}{j} x^j\right)^i$$

$$= \sum_{0 \leq d \leq q^2} \left(\sum_{\substack{0 \leq i \leq q \\ 1 \leq j_1, \ldots, j_i \leq q \\ j_1 + \cdots + j_i = d}} \frac{b^i}{i!} \frac{(-1)^{j_1 + \cdots + j_i + i}}{j_1 \cdots j_i} \right) x^d$$

$$= \sum_{0 \leq d \leq q^2} e_d x^d.$$

Fix some d, $q < d \leq q^2$, and set $h = \left\lfloor \frac{d}{p} \right\rfloor$. Then

$$e_d = \sum_{\substack{0 \leq i \leq q \\ 1 \leq j_1, \ldots, j_i \leq q \\ j_1 + \cdots + j_i = d}} \frac{b^i}{i!} \frac{(-1)^{j_1 + \cdots + j_i + i}}{j_1 \cdots j_i}$$

and by Lemma 4.2 we have

$$\frac{b^i t}{i!} \in \mathbf{Z},$$

$$\frac{p^h t^i}{j_1 \cdots j_i} \in \mathbf{Z},$$

for any $i, j_1, \ldots, j_i$ in the sum for e_d. It follows that

$$p^h t^{q+1} e_d \in \mathbf{Z},$$

$$p^h t^{q+1} e_d z^d \equiv 0 \bmod p^d.$$

Also, $p^d \mid z^d$ and $t^{q+1} e_d z^d \in \mathbf{Z}$, and hence

$$t^{q+1} e_d z^d \equiv 0 \bmod p^{d-h}.$$

Furthermore,

$$d - h = d - \left\lfloor \frac{d}{p} \right\rfloor \geq q + 1 - \left\lfloor \frac{q+1}{p} \right\rfloor = q + 1 - k.$$

Putting this together with the above congruence for $S(x)$, we get

$$t^{q+1} E_q(bL_q(z)) - t^{q+1}(1+z)^b$$

$$= t^{q+1}\left(S(z) + \sum_{d > q} e_d z^d\right) - t^{q+1}(1+z)^b$$

$$\equiv 0 \bmod p^{q-k+1},$$

which is the claim of the lemma. □

We now describe a parallel algorithm which, on input n-bit numbers a, b, m, computes $a^b \bmod m$, if m has small prime factors.

Algorithm PARALLEL INTEGER POWERING

1. Factor $m = p_1^{e_1} \cdots p_r^{e_r}$. By the assumption, we have $p_i \le n$. It is sufficient to compute

$$a^b \bmod p_i^{e_i}$$

for each i, since we can then use CHINREM. So from now on assume $m = p^e$ with p prime.

2. Find $l \ge 0$ such that

$$p^l \mid a,\ p^{l+1} \nmid a.$$

It is sufficient to compute

$$\left(\frac{a}{p^l}\right)^b \bmod p^{e-le}.$$

Replacing a by $\frac{a}{p^l}$, we now assume that $p \nmid a$, and compute $a^b \bmod p^e$.

3. Find b_0 such that $0 \le b_0 < p$ and $b \equiv b_0 \bmod p$. Compute

$$c_0 \equiv a^{b_0} \bmod p^e.$$

It is now sufficient to compute $a^{b-b_0} \bmod p^e$, and we can assume that $p \mid b$.

4. We assume that $u^{p^{e-1}} \bmod p^e$ is given ("hard-wired") for $1 \le u < p$. Find $a_0, f, g, z \in \mathbf{N}$ such that $1 \le a_0 < p$ and $0 \le f, g, z < p^e$, and

$$a_0 \equiv a \bmod p,$$

$$g \equiv a_0^{p^{e-1}} \bmod p^e,$$

$$fg \equiv 1 \bmod p^e,$$

$$z \equiv af - 1 \bmod p^e.$$

(Then $g \equiv a \bmod p$, and $z \equiv 0 \bmod p$.)

5. Compute $(1+z)^b \bmod p^e$ as follows. Set $q = \left\lceil e\frac{p}{p-1} \right\rceil$. Using binary search, compute $r, t \in \mathbf{N}$ such that

$$p^r \mid q!,\ p^{r+1} \nmid q!,\ t = \frac{q!}{p^r}.$$

For all j, $1 \le j \le q$, compute z^j, then

$$s = b \cdot \sum_{1 \le j \le q} (-1)^{j+1} \frac{tz^j}{j} \quad (= tbL_q(z)),$$

$$v = \sum_{0 \le i \le q} \frac{ts^i}{i!} t^{q-i} \quad (= t^{q+1} E_q(bL_q(z))).$$

Compute $w \in \mathbf{N}$ such that $1 \le w < p^e$ and

$$wt^{q+1} \equiv 1 \bmod p^e,$$

6. Compute $d, y \in \mathbf{N}$ such that $0 \le d \le p-2$, $1 \le y < p^e$, and

$$b \equiv d \bmod p-1,$$

$$y \equiv g^d \bmod p^e.$$

Return

$$c \equiv vwy \bmod p^e.$$

Theorem 4.4. The above algorithm correctly computes c such that

$$c \equiv a^b \bmod m.$$

The algorithm can be performed by a P-uniform family α_n of boolean circuits, where α_n has depth $O(\log^2 n)$ and size $n^{O(1)}$, and works for inputs a, b, m which have at most n bits each, and where each prime factor p of m satisfies $p \le n$.

Corollary 4.5.

$$\text{SF-POWER} \in \text{NC}^2 \ (P\text{-uniform}). \quad \square$$

Proof of Theorem. Correctness: First note that each summand in s is an integer, $p \mid s$, and by Lemma 4.2, also each summand for v is integral. Set $k = \left\lfloor \frac{q}{p} \right\rfloor$. By Lemma 4.3,

$$v \equiv t^{q+1}(1+z)^b \bmod p^{q-k+1},$$

$$q - k + 1 > q - \frac{q}{p} = q \cdot \frac{p-1}{p} \ge e\frac{p}{p-1} \cdot \frac{p-1}{p} = e,$$

$$wv \equiv wt^{q+1}(1+z)^b \equiv (af)^b \bmod p^e.$$

There exists $h \in \mathbf{Z}$ such that $b = d + (p-1)h$, and the order of the multiplicative group of units in $\mathbf{Z}_{p^e}$ is $\psi(p^e) = p^{e-1}(p-1)$. Therefore

$$a_0^{p^{e-1}(p-1)h} \equiv 1 \bmod p^e,$$

$$a^b \equiv a^b (fg)^b \equiv (af)^b g^b \equiv vwg^{d+(p-1)h}$$

$$\equiv vwy\, a_0^{p^{e-1}(p-1)h} \equiv vwy \equiv c \bmod p^e.$$

It remains to estimate the circuit depth and size. In step 4, we assume that $u^{p^{e-1}} \bmod p^e$ is given. This can be hard-wired for all $p \le n$, $1 \le u < p$, and $e \le n$ in size $n^{O(1)}$. The resulting circuit family is P-uniform, i.e. the n-th circuit can be constructed by a polynomial time bounded Turing machine on input n in unary. (See Ruzzo [1981], Cook [1983], Beame-Cook-Hoover [1984] for discussions of uniformity.) Beame-Cook-Hoover [1984] have $O(\log n)$ P-uniform circuits for division with remainder and iterated product of n-bit integers, and the size for each step of the algorithm is polynomial. We get the following estimates for the circuit depth:

Step 1: $O(\log^2 n)$.

Step 2: $O(\log n)$.

Step 3: b_0, c_0 : $O(\log n)$.

Step 4: a_0, z : $O(\log n)$,
g : $O(\log n)$ (table look-up),
f : $O(\log^2 n)$.

Step 5: q : $O(\log\log n)$,
$q!, r, t, s, v$: $O(\log n)$,
w : $O(\log^2 n)$.

Step 6: c, y : $O(\log n)$. $\square$

Remarks. **4.6.** The algorithm requires $u^{p^{e-1}} \bmod p^e$ hard-wired for $1 \le u,p,e \le n$. I do not know how to solve this special powering problem fast in parallel, and SF-POWER is not known to be in NC under log-space uniformity. However, when $m = 2^n$, then $a_0 = 1$, step 4 only uses $u = 1$, and only the calculation of w in step 5 requires more than $O(\log n)$ depth. Using the log-space uniform algorithm of Reif [1984] - with depth $O(\log n \cdot \log\log n)$ and polynomial size - for division with remainder and iterated product, it follows that $a^b \bmod 2^n$ can be computed in log-space uniform depth $O(\log^2 n \cdot \log\log n)$ and polynomial size. The algorithm similarly simplifies for the case $m = 3^n$, using representatives $\{-1,0,1\}$ for $\mathbf{Z}/3\mathbf{Z}$.

4.7. We know that the number of units mod p^e is

$$\#\mathbf{Z}_{p^e}^{\times} = \varphi(p^e) = (p-1)p^{e-1}.$$

In fact, if $e \ge 2$, and $p \ge 3$, then

$$G = \{u^{p^{e-1}} \bmod p^e : 1 \le u < p\},$$

$$H = \{v \bmod p^e : 1 \le v \le p^e \text{ and } v \equiv 1 \bmod p\}$$

are subgroups of $\mathbf{Z}_{p^e}^{\times}$, with order $p-1$ (namely $G \cong \mathbf{Z}_p^{\times}$) resp. p^{e-1}, and $\mathbf{Z}_{p^e}^{\times} = G \times H$. The algorithm essentially solves the powering problem in H. Since G is small, one can compute powers in G in depth $O(\log p)$, but in Remark 1 we ask to actually compute G.

4.8. An obvious question is to remove the condition of m having only small prime factors. Allan Borodin has proposed the following: can we compute fast in parallel the i-th bit of a^b, given n-bit numbers a,b,i? Alt [1984] shows that this is the case for the high-order bits of a^b.

Example 4.9. Let $q=3$, $p=2$ in Lemma 4.3. Then $t=3$, $k=2$, $t^{q+1}=81$. Let

$$F(x) = E_3(2L_3(x)) - (1+x)^2$$

$$= -\frac{1}{6}x^4 + \frac{5}{3}x^5 - \frac{23}{18}x^6 + \frac{7}{9}x^7 - \frac{2}{9}x^8 + \frac{4}{81}x^9.$$

We knew from Lemma 4.2 that if z is even, then $81 \cdot F(z) \in \mathbf{Z}$, and from Lemma 4.3 that

$$81 \cdot F(z) \equiv 0 \bmod 2^2.$$

In fact, $81 \cdot F(2) = 8 \cdot 373$. However, for $z = 1$ we find $81 \cdot F(1) = 67$; therefore the exp-log-approach does not seem to work directly to solve the problem we hard-wired in step 4.

References

H. Alt, Comparison of arithmetic functions with respect to boolean circuit depth. Proc. 16th Ann. ACM Symp. Theory of Computation, Washington DC, 1984, 466-470.

P.W. Beame, S.A. Cook and H.J. Hoover, Log depth circuits for division and related problems. Proc. 25th Ann. IEEE Symp. Foundations of Computer Science, Singer Island FL, 1984.

S.A. Cook, The classification of problems which have fast parallel algorithms. Proc. Int. Conf. Foundations of Computation Theory, Borgholm, 1983, Springer Lecture Notes Computer Science, vol. 158, 78-93.

W. Eberly, Very fast parallel matrix and polynomial arithmetic. Proc. 25th Ann. IEEE Symp. Foundations of Computer Science, Singer Island FL, 1984.

H.T. Kung, New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. J. ACM **23** (1976), 252-261.

P. McKenzie and S.A. Cook, The parallel complexity of the Abelian permutation group membership problem. Proc. 24th Ann. IEEE Symp. Foundations of Computer Science, Tucson, 1983, 154-161.

J. Reif, Logarithmic depth circuits for algebraic functions. Tech. Rep. TR-35-82, Harvard University, Revised version, 1984. See also Proc. 24th Annual IEEE Symp. Foundations of Computer Science, Tucson, 1983, 138-145.

W.L. Ruzzo, On uniform circuit complexity. J. Computer System Sciences **22** (1981), 365-383.

PARALLEL POWERING

Joachim von zur Gathen
Department of Computer Science
University of Toronto

Abstract

A fast parallel computation for large powers of an integer modulo another integer is presented, assuming that the modulus has only small prime factors.

1. Introduction

Consider the problem of computing $a^b \bmod m$ in parallel, where a, b, m are n-bit integers. This problem comes up as a subroutine in many computational problems, e.g. factoring integers, primality tests, cryptographic schemes, and factoring polynomials over finite fields. The method of "repeated squaring" does not yield fast parallel computations, i.e. of parallel time $(\log n)^{O(1)}$.

We present boolean circuits for this problem. Provided that the modulus m has only small prime factors $p \leq n$, the depth is $O(\log^2 n)$ with polynomial size.

In section 2 we present the technique to be used, in the setting of polynomials. Section 3 has some auxiliary parallel algorithms, for factoring numbers and Chinese remaindering. The fact that factoring can be achieved fast in parallel puts the constraint of "small prime factors" into perspective: it is a severe restriction, and for many applications the more interesting case is that of large prime factors. However, the present result is the first one that provides an exponential parallel speed-up of the sequential methods for modular integer powering. A slightly different problem - computing in parallel the high-order bits of a large power of an integer - is considered in Alt [1984].

In section 4 we then adapt the polynomial technique to the case of integers. The resulting parallel algorithm for computing $a_b \bmod m$ requires a special case of this problem to be "hard-wired", and yields a P-uniform family of boolean circuits of depth $O(\log^2 n)$.

2. Powering polynomials

The idea of the integer powering algorithm is based on a method which is quite natural for polynomials. In this section, we explain this method and its inherent limitations.

We work over an arbitrary ground field F of characteristic zero. The model of computation (for this section only) is an arithmetic circuit with inputs and constants from F, and +, -, *, / as operations. Consider the following four problems, where $u \in F$, and $f, g \in F[x]$ of degree n are inputs, and b is an n-bit integer.

POWER-1: Compute u^b.

POWER-2: Compute $f^b \bmod g$.

POWER-3: Compute $f^b \bmod x^n$.

POWER-4: Compute $f^b \bmod x^n$, if f has constant term 1.

Theorem 2.1. Problems POWER-1, POWER-2, POWER-3 cannot be computed on any family of arithmetic circuits of depth $(\log n)^{O(1)}$.

Proof. It is well-known that the rational function accociated with a gate of depth d (in an arithmetic circuit) has degree at most 2^{d-1} (Kung [1976]). Hence the depth d_n of any circuit computing u^{2^n} satisfies $d_n > n$, and the claim for POWER-1 follows. POWER-1 is reducible in depth $O(\log n)$ to both POWER-2 and POWER-3. □

This result does not extend to finite fields e.g. if $F = \mathbf{Z}_2$ then

$$a^b = a, \ \forall b \geq 1, \ a \in F.$$

Allowing only constant term 1 in f rules out the above reduction from POWER-1 to POWER-3, and in fact we have the following fast parallel algorithm for POWER-4.

Algorithm PARALLEL POLYNOMIAL POWERING

Input : $f \in F[x]$, deg $f \leq n$, f has constant term 1, char $F = 0$, b a n-bit integer.

Output : $f^b \bmod x^n$

0272-5428/84/0000/0031$01.00 © 1984 IEEE

1. Compute $h = \log f \mod x^n$.
   Consider f as a power series with only finitely many non-zero terms and compute the first n terms of $\log f = \sum_{i \geq 1} \frac{(-1)^{i+1}}{i} (f-1)^i$.
2. Compute $b \cdot h$
3. Compute $\exp(b \cdot h) \mod x^n$.
   As in step 1, using the fact that $\exp g = \sum_{i \geq 0} \frac{1}{i!} g^i$, where g has zero constant term; i.e. we compute $\sum_{0 \leq i \leq n} \frac{1}{i!} g^i \mod x^n$.

Theorem 2.2. Algorithm PARALLEL POLYNOMIAL POWERING can be implemented on an arithmetic circuit of depth $O(\log n)$.

Proof. In steps 1 and 3 we have to compute $h^i \mod x^n$ for various polynomials h (with $h(0) = 0$ and $1 \leq i \leq n$). This can obviously be done in depth $O(\log^2 n)$, and in fact in depth $O(\log n)$ by Reif [1984] (assuming roots of unity are available), and Eberly [1984] (for arbitrary fields). □

Theorem 2.1 shows that the condition "constant term 1" for substituting into the logarithm power series is not just technical, but in our context related to the parallel computational complexity.

3. Chinese remaindering

To solve the powering problem, we have to present parallel computations for some other problems, which may be of independent interest. Inputs are the various n-bit numbers denoted as $a, b, a_1, \ldots, a_n, m, m_1, \ldots, m_n$, and output an n-bit number c as below:

POWER : $c \equiv a^b \bmod m$,

MODINV : $\begin{cases} ac \equiv 1 \bmod m & \text{if } gcd(a,m) = 1, \\ c = 0 & \text{otherwise,} \end{cases}$

("modular inverse")

LAGRANGE : $c \equiv 0 \bmod m_2 \cdots m_n$,

$$c \equiv \begin{cases} 1 \bmod m_1 & \text{if } gcd(m_1, m_2 \cdots m_n) = 1, \\ 0 \bmod m_1 & \text{otherwise,} \end{cases}$$

("Lagrange interpolation coefficient"),

CHINREM : If $gcd(m_i, m_j) = 1$ for all $i \neq j$,
then $c \equiv a_i \bmod m_i$ for all i.
Otherwise c is arbitrary.
("Chinese remainder algorithm")

FACTOR : compute the prime factors of m (with multiplicities).

Using DIV for the problem of integer division with remainder, we have an NC^1-reduction (see Cook [1983] for terminology):

$$\text{CHINREM} \leq \text{LAGRANGE} \leq \text{MODINV} + \text{DIV}.$$

It follows by using an inverse d of $m_2 \cdots m_n \bmod m_1$; then $c = d \cdot (m_2 \cdots m_n)$ is sufficient. The iterated product $m_2 \cdots m_n$ is reducible to division (see Beame-Cook-Hoover [1984]).

At the present time, none of these problems is known to be in NC - i.e solvable in depth $(\log n)^{O(1)}$ - or complete for P. All problems - except possibly FACTOR - are in P.

In their work on Abelian permutation group membership, McKenzie-Cook [1983] had to solve systems of linear congruences with the constraint that the modulus m was known to have only small prime power factors, i.e. $m = \prod p_i^{e_i}$ with $p_i^{e_i} = O(n)$ for all i. They then showed that this problem is in RNC^3. We now use "SF" to denote the condition that $m, m_1, \ldots, m_n$ have "small factors", i.e. only prime factors $p \leq n$. This is a little more generous than the condition in McKenzie-Cook [1983].

We assume that there exist polynomial-size circuits of depth $\text{div}(n)$ that compute the division with remainder for n-bit integers. By Beame-Cook-Hoover [1984] $\text{div}(n) = O(\log n)$ for P-uniform circuits, and $\text{div}(n) = O(\log n \log\log n)$ for log-space uniform circuits, by Reif [1984].

Theorem 3.1. The following problems can be computed by boolean circuits of polynomial size and the stated depth:

(i) SF-FACTOR in depth $O(\text{div}(n))$,

(ii) SF-MODINV, SF-LAGRANGE, and SF-CHINREM in depth $O(\log n \cdot \text{div}(n))$.

Proof. We leave away the "SF-". First consider FACTOR. The algorithm is obvious: For each number $p \leq n$, test whether p is prime and determine its multiplicity in m. This can be performed in depth $\text{div}(n)$, by computing $p, p^2, p^3, \ldots, p^n$ and testing for each i whether p^i divides m.

Now consider MODINV in the special case where $m = p^e$, p prime. We have the following algorithm:

1. Test if $p \mid a$, and return $c = 0$ if "yes".
2. Find by exhaustive search $c_0 \in \mathbf{N}$ such that
$$1 \leq c_0 < p$$
$$ac_0 \equiv 1 \bmod p.$$
3. Newton iteration: Set $k = \lceil \log_2 e \rceil$, and compute $c_1, \ldots, c_k$ as follows:
$$c_i = 2c_{i-1} - ac_{i-1}^2 \bmod p^{2^i},$$
$$1 \leq c_i < p^{2^i}.$$

(Then $ac_i \equiv 1 \bmod p^{2^i}$, by the Newton formula

$$ac_i - 1 \equiv$$

$$\frac{1}{c_{i-1}^2}(c_i - (2c_{i-1} - ac_{i-1}^2) - (c_i - c_{i-1})^2) \bmod p^{2^i}.)$$

4. Return $c = c_k \bmod p^e$.

The circuit depth for this algorithm is dominated by step 3:

$$O(\log e \cdot (\log n + \mathrm{div}(n))) = O(\log n \ \mathrm{div}(n)).$$

Next we consider LAGRANGE. If $m_1 = p^e$ for a small prime p, then we can solve LAGRANGE in depth $O(\log n \ \mathrm{div}(n))$ by the reduction to MODINV + DIV. For the general case $m = p_1^{e_1} \cdots p_r^{e_r}$, with small pairwise distinct primes $p_1, \ldots, p_r$ (and no p_i dividing $m_2 \cdots m_n$), we compute $c_1, \ldots, c_r$ such that

$$c_i \equiv \begin{cases} 1 \bmod p_i^{e_i} \\ 0 \bmod p_1^{e_1} \ldots p_{i-1}^{e_{i-1}} p_{i+1}^{e_{i+1}} \ldots p_r^{e_r} m_2 \ldots m_n. \end{cases}$$

Each c_i is given by one of the special LAGRANGE problems already solved, and

$$c = c_1 + \cdots + c_r$$

solves the current problem. Therefore LAGRANGE and CHINREM can be solved in depth $O(\log n \ \mathrm{div}(n))$.

Now for the general case of MODINV, where $m = p_1^{e_1} \cdots p_r^{e_r}$ with small primes p_i, we have the algorithm:

1. for $i \le r$ compute c_i such that $ac_i \equiv 1 \bmod p_i^{e_i}$,
2. compute c such that for all $i \le r$ $c \equiv c_i \bmod p_i^{e_i}$.

Then

$$\forall i \le r \quad ac \equiv ac_i \equiv 1 \bmod p_i^{e_i}$$

$$\Longrightarrow ac \equiv 1 \bmod m. \ \square$$

4. Powering integers

Now we consider the problem POWER: computing $a^b \bmod m$. We adopt the algorithm PARALLEL POLYNOMIAL POWERING to solve this problem in P-uniform depth $O(\log^2 n)$ if m has only small prime factors.

Let $1 \le q \le s$ be integers, and consider the truncated exponential and logarithmic power series:

$$E_q(y) = \sum_{0 \le i \le q} \frac{1}{i!} y^i \in \mathbf{Q}[y],$$

$$L_s(x) = \sum_{1 \le i \le s} \frac{(-1)^{i+1}}{i} x^i \in \mathbf{Q}[x].$$

(Thus $E_q(y) \equiv exp(y) \bmod y^{q+1}$, and $L_s(x) \equiv log(1+x) \bmod x^s$.) The next three lemmas establish the usefulness of these series for integer powering. Lemma 4.1 has implicitly been used in Theorem 2.2.

Lemma 4.1. Let $0 \le q \le s$ and $b \in \mathbf{N}$. Then we have in $\mathbf{Q}[x]$:

$$E_q(bL_s(x)) \equiv (1+x)^b \bmod x^{q+1}.$$

Proof. We prove the lemma by induction on q. The case $q = 0$ (or $q = 1$) is easily checked. For the inductive step, we have

$$\frac{\partial}{\partial x}(E_q(bL_s(x)) - (1+x)^b)$$

$$= \frac{\partial E_q}{\partial y}(bL_s(x)) \cdot b \cdot \frac{\partial}{\partial x}\Big(\sum_{1 \le i \le s} \frac{(-1)^{i+1}}{i} x^i\Big) - b(1+x)^{b-1}$$

$$= \Big(\sum_{0 \le i \le q-1} \frac{1}{i!}(bL_s(x))^i\Big) \cdot b \cdot \Big(\sum_{1 \le i \le s} (-1)^{i+1} x^{i-1}\Big) - b(1+x)^{b-1}$$

$$\equiv b(1+x)^{b-1}\Big((1+x)\Big(\sum_{1 \le i \le s} (-1)^{i-1} x^{i-1}\Big) - 1\Big)$$

$$= b(1+x)^{b-1} \cdot (-1)^{s-1} \cdot x^s \equiv 0 \bmod x^q,$$

where we have used the induction hypothesis in the third transformation. Now one verifies that

$$(E_q(bL_s(x)) - (1+x)^b)(0) = E_q(bL_s(0)) - 1 = 0,$$

and the claim follows. $\square$

From now on we will always have $s = q$. The idea is now essentially to substitute a prime number p for x in the congruence of Lemma 4.1, and hope to obtain a similar congruence in $\mathbf{Z}$. The problem is that the left hand side has denominators in the terms of order greater than n. Lemma 4.2 bounds the multiplicity of p in integers of the form $i!$ and $j_1 \cdots j_i$, and Lemma 4.3 states that we do not lose too much of the initial precision $q+1$ given by Lemma 4.1.

We will have to consider $t = \frac{q!}{p^r}$, where $p^r \mid q!$ and $p^{r+1} \nmid q!$. We write $t = \frac{q!}{gcd(q!, p^q)}$, and note that $gcd(p,t)=1$.

Lemma 4.2. Let $d,i,p,t \in \mathbf{N}$ such that p is prime, and $t = \frac{q!}{gcd(q!, p^q)}$. Then

(i) $i! \mid p^i t$,

(ii) If $h = \left\lfloor \frac{d}{p} \right\rfloor$, and $j_1, \ldots, j_i \ge 1$ with $j_1 + \cdots + j_i \le d$, then

$$j_1 \cdots j_i \mid p^h t^i.$$

Proof. (i) The multiplicity of p in $i!$ is

$$\left\lfloor \frac{i}{p} \right\rfloor + \left\lfloor \frac{i}{p^2} \right\rfloor + \cdots < \frac{i}{p} + \frac{i}{p^2} + \cdots = \frac{i}{p} \cdot \frac{p}{p-1}$$

$$= \frac{i}{p-1} \leq i.$$

(ii) We have to bound the multiplicity of p in $j_1 \cdots j_i$. We use induction on d. The base case $d < p$ is clear. We can assume that d and each j_s is divisible by p, since otherwise the claim follows by induction. Furthermore, we can assume that

$$\frac{j_1}{p}, \ldots, \frac{j_m}{p} \text{ are not divisible by } p,$$

$$\frac{j_{m+1}}{p}, \ldots, \frac{j_i}{p} \text{ are divisible by } p,$$

for some m, $0 \leq m \leq i$. Then

$$m \leq i \leq \frac{d}{p} = h,$$

$$p^2(i-m) \leq j_{m+1} + \cdots + j_i \leq d - pm = p(h-m),$$

$$i \leq m + \frac{h-m}{p},$$

$$j_1 \cdots j_m \mid p^m t^m,$$

$$\frac{j_{m+1}}{p} + \cdots + \frac{j_i}{p} \leq h - m.$$

Set $l = \left\lfloor \frac{h-m}{p} \right\rfloor$. By the induction hypothesis,

$$\frac{j_{m+1}}{p} \cdots \frac{j_i}{p} \mid p^l t^{i-m},$$

$$j_{m+1} \cdots j_i \mid p^{l+i-m} t^{i-m},$$

$$j_1 \cdots j_i \mid p^{l+i} t^i,$$

$$l+i \leq \frac{h-m}{p} + m + \frac{h-m}{p} = h - (h-m)(1-\frac{2}{p}) \leq h.$$

(In fact, the exponent $h = \frac{d}{p}$ is required when $j_1 = \cdots = j_i = p$ and $d = pi$.) □

Lemma 4.3. Let $b, p, q, t, z \in \mathbf{N}$, $k = \left\lfloor \frac{q+1}{p} \right\rfloor$ such that p is prime, $t = \frac{q!}{gcd(q!, p^q)}$, and $p \mid b$ and $p \mid z$. Then

$$t^{q+1} E_q(bL_q(z)) \in \mathbf{Z},$$

$$t^{q+1} E_q(bL_q(z)) \equiv t^{q+1}(1+z)^b \bmod p^{q-k+1}.$$

Proof. Write

$$E_q(bL_q(x)) = \sum_{0 \leq d \leq q^2} e_d x^d \in \mathbf{Q}[x]$$

with $e_d \in \mathbf{Q}$, and

$$S(x) = \sum_{0 \leq d \leq q} e_d x^d \in \mathbf{Q}[x].$$

From Lemma 4.1, we know that

$$x^{q+1} \mid E_q(bL_q(x)) - (1+x)^b \text{ in } \mathbf{Q}[x]$$

and hence

$$x^{q+1} \mid S(x) - (1+x)^b \text{ in } \mathbf{Q}[x],$$

$$S(x) \in \mathbf{Z}[x],$$

$$\exists\, g(x) \in \mathbf{Z}[x] \quad g(x) \cdot x^{q+1} = S(x) - (1+x)^b,$$

$$S(z) \equiv (1+z)^b \bmod p^{q+1},$$

$$t^{q+1} S(z) \equiv t^{q+1}(1+z)^b \bmod p^{q+1}.$$

It remains to prove that

$$t^{q+1} e_d z^d \equiv 0 \bmod p^{q-k+1}$$

for any d with $q < d \leq q^2$.

$$E_q(bL_q(x)) = \sum_{0 \leq i \leq q} \frac{1}{i!} \Big(b \sum_{1 \leq j \leq q} \frac{(-1)^{j+1}}{j} x^j\Big)^i$$

$$= \sum_{0 \leq d \leq q^2} \Bigg(\sum_{\substack{0 \leq i \leq q \\ 1 \leq j_1, \ldots, j_i \leq q \\ j_1 + \cdots + j_i = d}} \frac{b^i}{i!} \frac{(-1)^{j_1 + \cdots + j_i + i}}{j_1 \cdots j_i} \Bigg) x^d$$

$$= \sum_{0 \leq d \leq q^2} e_d x^d.$$

Fix some d, $q < d \leq q^2$, and set $h = \left\lfloor \frac{d}{p} \right\rfloor$. Then

$$e_d = \sum_{\substack{0 \leq i \leq q \\ 1 \leq j_1, \ldots, j_i \leq q \\ j_1 + \cdots + j_i = d}} \frac{b^i}{i!} \frac{(-1)^{j_1 + \cdots + j_i + i}}{j_1 \cdots j_i}$$

and by Lemma 4.2 we have

$$\frac{b^i t}{i!} \in \mathbf{Z},$$

$$\frac{p^h t^i}{j_1 \cdots j_i} \in \mathbf{Z},$$

for any $i, j_1, \ldots, j_i$ in the sum for e_d. It follows that

$$p^h t^{q+1} e_d \in \mathbf{Z},$$

$$p^h t^{q+1} e_d z^d \equiv 0 \bmod p^d.$$

Also, $p^d \mid z^d$ and $t^{q+1} e_d z^d \in \mathbf{Z}$, and hence

$$t^{q+1} e_d z^d \equiv 0 \bmod p^{d-h}.$$

Furthermore,

$$d - h = d - \left\lfloor \frac{d}{p} \right\rfloor \geq q + 1 - \left\lfloor \frac{q+1}{p} \right\rfloor = q + 1 - k.$$

Putting this together with the above congruence for $S(x)$, we get

$$t^{q+1} E_q(bL_q(z)) - t^{q+1}(1+z)^b$$

$$= t^{q+1}(S(z) + \sum_{d > q} e_d z^d) - t^{q+1}(1+z)^b$$

$$\equiv 0 \bmod p^{q-k+1},$$

which is the claim of the lemma. □

We now describe a parallel algorithm which, on input n-bit numbers a, b, m, computes $a^b \bmod m$, if m has small prime factors.

Algorithm PARALLEL INTEGER POWERING

1. Factor $m = p_1^{e_1} \cdots p_r^{e_r}$. By the assumption, we have $p_i \le n$. It is sufficient to compute

$$a^b \bmod p_i^{e_i}$$

for each i, since we can then use CHINREM. So from now on assume $m = p^e$ with p prime.

2. Find $l \ge 0$ such that

$$p^l \mid a,\ p^{l+1} \nmid a.$$

It is sufficient to compute

$$\left(\frac{a}{p^l}\right)^b \bmod p^{e-lc}.$$

Replacing a by $\frac{a}{p^l}$, we now assume that $p \nmid a$, and compute $a^b \bmod p^e$.

3. Find b_0 such that $0 \le b_0 < p$ and $b \equiv b_0 \bmod p$. Compute

$$c_0 \equiv a^{b_0} \bmod p^e.$$

It is now sufficient to compute $a^{b-b_0} \bmod p^e$, and we can assume that $p \mid b$.

4. We assume that $u^{p^{e-1}} \bmod p^e$ is given ("hard-wired") for $1 \le u < p$. Find $a_0, f, g, z \in \mathbf{N}$ such that $1 \le a_0 < p$ and $0 \le f, g, z < p^e$, and

$$a_0 \equiv a \bmod p,$$

$$g \equiv a_0^{p^{e-1}} \bmod p^e,$$

$$fg \equiv 1 \bmod p^e,$$

$$z \equiv af - 1 \bmod p^e.$$

(Then $g \equiv a \bmod p$, and $z \equiv 0 \bmod p$.)

5. Compute $(1+z)^b \bmod p^e$ as follows. Set $q = \left\lceil e\frac{p}{p-1} \right\rceil$. Using binary search, compute $r, t \in \mathbf{N}$ such that

$$p^r \mid q!,\ p^{r+1} \nmid q!,\ t = \frac{q!}{p^r}.$$

For all j, $1 \le j \le q$, compute z^j, then

$$s = b \cdot \sum_{1 \le j \le q} (-1)^{j+1} \frac{tz^j}{j} \quad (= tbL_q(z)),$$

$$v = \sum_{0 \le i \le q} \frac{ts^i}{i!} t^{q-i} \quad (= t^{q+1} E_q(bL_q(z))).$$

Compute $w \in \mathbf{N}$ such that $1 \le w < p^e$ and

$$wt^{q+1} \equiv 1 \bmod p^e,$$

6. Compute $d, y \in \mathbf{N}$ such that $0 \le d \le p-2$, $1 \le y < p^e$, and

$$b \equiv d \bmod p-1,$$

$$y \equiv g^d \bmod p^e.$$

Return

$$c \equiv vwy \bmod p^e.$$

Theorem 4.4. The above algorithm correctly computes c such that

$$c \equiv a^b \bmod m.$$

The algorithm can be performed by a P-uniform family α_n of boolean circuits, where α_n has depth $O(\log^2 n)$ and size $n^{O(1)}$, and works for inputs a, b, m which have at most n bits each, and where each prime factor p of m satisfies $p \le n$.

Corollary 4.5.

$$\text{SF-POWER} \in \text{NC}^2\ (P\text{-uniform}). \quad \square$$

Proof of Theorem. Correctness: First note that each summand in s is an integer, $p \mid s$, and by Lemma 4.2, also each summand for v is integral. Set $k = \left\lfloor \frac{q}{p} \right\rfloor$. By Lemma 4.3,

$$v \equiv t^{q+1}(1+z)^b \bmod p^{q-k+1},$$

$$q - k + 1 > q - \frac{q}{p} = q \cdot \frac{p-1}{p} \ge e\frac{p}{p-1} \cdot \frac{p-1}{p} = e,$$

$$wv \equiv wt^{q+1}(1+z)^b \equiv (af)^b \bmod p^e.$$

There exists $h \in \mathbf{Z}$ such that $b = d + (p-1)h$, and the order of the multiplicative group of units in $\mathbf{Z}_{p^e}$ is $\psi(p^e) = p^{e-1}(p-1)$. Therefore

$$a_0^{p^{e-1}(p-1)h} \equiv 1 \bmod p^e,$$

$$a^b \equiv a^b (fg)^b \equiv (af)^b g^b \equiv vwg^{d+(p-1)h}$$

$$\equiv vwy\ a_0^{p^{e-1}(p-1)h} \equiv vwy \equiv c \bmod p^e.$$

It remains to estimate the circuit depth and size. In step 4, we assume that $u^{p^{e-1}} \bmod p^e$ is given. This can be hard-wired for all $p \le n$, $1 \le u < p$, and $e \le n$ in size $n^{O(1)}$. The resulting circuit family is P-uniform, i.e. the n-th circuit can be constructed by a polynomial time bounded Turing machine on input n in unary. (See Ruzzo [1981], Cook [1983], Beame-Cook-Hoover [1984] for discussions of uniformity.) Beame-Cook-Hoover [1984] have $O(\log n)$ P-uniform circuits for division with remainder and iterated product of n-bit integers, and the size for each step of the algorithm is polynomial. We get the following estimates for the circuit depth:

Step 1: $O(\log^2 n)$.

Step 2: $O(\log n)$.

Step 3: $b_0, c_0 : O(\log n)$.

Step 4: $a_0, z : O(\log n)$,
$g : O(\log n)$ (table look-up),
$f : O(\log^2 n)$.

Step 5: $q : O(\log\log n)$,
$q!, r, t, s, v : O(\log n)$,
$w : O(\log^2 n)$.

Step 6: $c, y : O(\log n)$. $\square$

Remarks. 4.6. The algorithm requires $u^{p^{e-1}} \bmod p^e$ hard-wired for $1 \le u, p, e \le n$. I do not know how to solve this special powering problem fast in parallel, and SI-POWER is not known to be in NC under log-space uniformity. However, when $m = 2^n$, then $a_0 = 1$, step 4 only uses $u = 1$, and only the calculation of w in step 5 requires more than $O(\log n)$ depth. Using the log-space uniform algorithm of Reif [1984] - with depth $O(\log n \cdot \log\log n)$ and polynomial size - for division with remainder and iterated product, it follows that $a^b \bmod 2^n$ can be computed in log-space uniform depth $O(\log^2 n \cdot \log\log n)$ and polynomial size. The algorithm similarly simplifies for the case $m = 3^n$, using representatives $\{-1, 0, 1\}$ for $\mathbb{Z}/3\mathbb{Z}$.

4.7. We know that the number of units mod p^e is

$$\#\mathbb{Z}_{p^e}^{\times} = \varphi(p^e) = (p-1)p^{e-1}.$$

In fact, if $e \ge 2$, and $p \ge 3$, then

$$G = \{u^{p^{e-1}} \bmod p^e : 1 \le u < p\},$$

$$H = \{v \bmod p^e : 1 \le v \le p^e \text{ and } v \equiv 1 \bmod p\}$$

are subgroups of $\mathbb{Z}_{p^e}^{\times}$, with order $p-1$ (namely $G \cong \mathbb{Z}_p^{\times}$) resp. p^{e-1}, and $\mathbb{Z}_{p^e}^{\times} = G \times H$. The algorithm essentially solves the powering problem in H. Since G is small, one can compute powers in G in depth $O(\log p)$, but in Remark 1 we ask to actually compute G.

4.8. An obvious question is to remove the condition of m having only small prime factors. Allan Borodin has proposed the following: can we compute fast in parallel the i-th bit of a^b, given n-bit numbers a, b, i? Alt [1984] shows that this is the case for the high-order bits of a^b.

Example 4.9. Let $q=3, p=2$ in Lemma 4.3. Then $t=3$, $k=2$, $t^{q+1}=81$. Let

$$F(x) = E_3(2L_3(x)) - (1+x)^2$$

$$= -\frac{1}{6}x^4 + \frac{5}{3}x^5 - \frac{23}{18}x^6 + \frac{7}{9}x^7 - \frac{2}{9}x^8 + \frac{4}{81}x^9.$$

We knew from Lemma 4.2 that if z is even, then $81 \cdot F(z) \in \mathbb{Z}$, and from Lemma 4.3 that

$$81 \cdot F(z) \equiv 0 \bmod 2^2.$$

In fact, $81 \cdot F(2) = 8 \cdot 373$. However, for $z = 1$ we find $81 \cdot F(1) = 67$; therefore the exp-log-approach does not seem to work directly to solve the problem we hard-wired in step 4.

References

H. Alt, Comparison of arithmetic functions with respect to boolean circuit depth. Proc. 16th Ann. ACM Symp. Theory of Computation, Washington DC, 1984, 466-470.

P.W. Beame, S.A. Cook and H.J. Hoover, Log depth circuits for division and related problems. Proc. 25th Ann. IEEE Symp. Foundations of Computer Science, Singer Island FL, 1984.

S.A. Cook, The classification of problems which have fast parallel algorithms. Proc. Int. Conf. Foundations of Computation Theory, Borgholm, 1983, Springer Lecture Notes Computer Science, vol. 158, 78-93.

W. Eberly, Very fast parallel matrix and polynomial arithmetic. Proc. 25th Ann. IEEE Symp. Foundations of Computer Science, Singer Island FL, 1984.

H.T. Kung, New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. J. ACM **23** (1976), 252-261.

P. McKenzie and S.A. Cook, The parallel complexity of the Abelian permutation group membership problem. Proc. 24th Ann. IEEE Symp. Foundations of Computer Science, Tucson, 1983, 154-161.

J. Reif, Logarithmic depth circuits for algebraic functions. Tech. Rep. TR-35-82, Harvard University, Revised version, 1984. See also Proc. 24th Annual IEEE Symp. Foundations of Computer Science, Tucson, 1983, 138-145.

W.L. Ruzzo, On uniform circuit complexity. J. Computer System Sciences **22** (1981), 365-383.