Division in $O(\log n)$ Depth Using $O(n^{1+\epsilon})$ Processors

Johan Hastad* and Tom Leighton†
MIT

Abstract: Improving a result by Beame, Cook and Hoover we construct a family of P-uniform $O(\frac{1}{\epsilon^2}\log n)$ depth circuits for division with size $O(n^{1+\epsilon})$ for any $\epsilon > 0$. The key improvement is that we are able to do Chinese remaindering with $(n^{1+\epsilon})$ processors in $O(\log n)$ depth.

Warning: This is a very old half finished paper that has never been published and is not finally polished. Thus treat it as some set of ideas only we post for the possible benifit of anybody interested.

1.Introduction

Division seems to be harder than multiplication, addition or subtraction. The problem of constructing shallow circuits for division has been considered by several authors. Classical methods using Newton iterations yields circuits of depth $O(\log^2 n)$. The first improvement of this was given by Reif in [4],[5]. Using a clever construction involving discrete Fourier transforms he was able to construct polynomial size circuits of depth $O(\log n \log\log n)$. By using an elegant method relying on modular arithmetic Beame, Cook and Hoover [1] were able to construct circuits of depth $O(\log n)$. This is of course optimal but the construction had two imperfections. The circuits were of size $\Theta(n^5)$ and only P-uniform while Reif's circuits were small and logspace-uniform. The result of this paper is remove one of these imperfections in that we for any $\epsilon > 0$ construct circuits for division of depth $O(\log n)$ and size $O(n^{1+\epsilon})$. Unfortunately our circuits are still only P-uniform.

Our construction basically follows the same ideas as the construction in [1], but we introduce a number of improvements to make the construction more efficient. The most important of these improvements is to introduce a more efficient way of implementing Chinese remaindering. The other key idea is to use one extra level of modular arithmetic to reduce the sizes of the lookup tables.

The outline of the paper is the following. In section 2 we give a high level description of the algorithm. The efficient implementation of Chinese remaindering is described in section 3. We believe that this result is of independent interest and that Chinese remaindering could be used as a building block in other algorithms. Our algorithm will do a number of approximations during the computation and in section 4.1 we verify that this does not affect

* Royal Institute of Technology
† MIT

the accuracy of the answer. In section 4.2 we describe in more detail the implementation of the individual steps and prove the main theorem. Finally, in section 5 we have some concluding remarks and open problems.

2. Description of Algorithm.

Assume for simplicity that n is a power of 2. If this is not the case we can work with the smallest power of 2 larger than n. In this paper all logarithms will be to the base 2.

Since it is well known how to do multiplication efficiently in parallel, the key step is how to do inversion. Thus we will focus on this problem.

Inversion Algorithm.

1) Input y, an n-digit number to be inverted.
2) Put $z = 2^{-k}y$ where k is such that $\frac{1}{2} < z \leq 1$.
3) Define $x = 1 - z$. Now $\frac{1}{z} = \frac{1}{1-x} = \sum_{i=0}^{\infty} x^i$ and $0 \leq x < \frac{1}{2}$.
4) Compute $w = \sum_{i=0}^{2n-1} x^i = \prod_{i=0}^{\log n}(1 + x^{2^i})$ approximately by first finding approximations $\tilde{x}_r$ for the powers x^{2^r} in parallel as follows.
5) Let δ be a small number and assume for notational simplicity that $\delta \log n$ is an integer. Take primes $p_1, p_2 \ldots, p_s$ such that $\prod_{i=1}^{s} p_i > 2^{2n^{1+\delta}}$. By [3] we can do this with $p_i < 3n^{1+\delta}$ and hence $s < 3n^{1+\delta}$.
6) Let $l = \lfloor \frac{r}{\delta \log n} \rfloor$ and $t = r - l\delta \log n$. Set $a_0 = x$. For $j = 0, 1 \ldots, l-1$
   6a) Take the $2n$ bit approximation a_j of $x^{2^{j\delta \log n}}$. Compute $m_{ij} \equiv 2^{2n}a_j \pmod{p_i}$, $i = 1, 2 \ldots s$.
   6b) Compute $m_{ij}^{2^{\delta \log n}} \pmod{p_i}$, combine the results using Chinese remaindering and truncate to $2n$ bits to get an approximation a_{j+1} for $x^{2^{(j+1)\delta \log n}}$.

   Next j.

   Take a_l and compute $m_i \equiv 2^{2n}a_l \pmod{p_i}$. Compute $m_i^{2^t} \pmod{p_i}$ and convert back using Chinese remaindering to get a $2n$ bit approximation $\tilde{x}_r$ of x^{2^r}.
7) Compute $\tilde{w} = \prod_{i=0}^{\log n}(1 + \tilde{x}_i)$ using modular arithmetic.
8) Output $\tilde{w}2^{-k}$ to n digits accuracy.

We need to verify that this construction can be implemented as efficiently as stated and that the answer is correct. We will start by describing how to do Chinese remaindering in the next section. An important fact that we will make use of repeatedly is the result by Reif [4],[5] that multiplication of n-bit numbers can be done in depth $O(\log n)$ using $O(n \log n \log\log n)$ processors.

3. Efficient Chinese Remaindering.

Let p_i, $i = 1, 2 \ldots, s$ be relatively prime numbers, each consisting of at most m digits. Chinese remaindering consists of two separate problems. The first is that when given an integer x find x_i such $x \equiv x_i \pmod{p_i}$. The second problem is the inverse i.e. when given the x_i find the unique smallest x. Both problems, of course, have very efficient solutions using sequential computation since $x_i = x - \lfloor \frac{x}{p_i} \rfloor p_i$ and $x = \sum_{i=1}^{s} u_i x_i$ where u_i are numbers such that $u_i \equiv 1 \pmod{p_i}$ and $u_i \equiv 0 \pmod{p_j}$ for $j \neq i$. We note for the record that these numbers u_i can be constructed in polynomial time given the numbers p_i. Our only problem is how to do these computations efficiently in parallel.

As before δ is a small positive number and assume for notational simplicity that s^{δ} and $\frac{1}{\delta}$ are integers. The computation can be wieved as taking place on a tree with fanout s^{δ} and depth $\frac{1}{\delta}$. At the root there is the number x and at leaf i the number x_i. Each internal node v corresponds to the set of primes which belongs to the leaves in its subtree. Let P_v be the product of these primes, then the number x_v, associated with v is $x \pmod{P_v}$. Thus we have two computational problems, either we are given the number at the root and we want to compute the numbers at the leaves or the other way around. Let us first discuss the first problem.

We will do the computation level by level in the tree. All the computations on one level are of course done in parallel. At level i we have $s^{i\delta}$ nodes and to each node v corresponds $s^{1-i\delta}$ primes. Thus in this case P_v is a $s^{1-i\delta}m$ bit number. Let $f(v)$ be the father of v. Then we have to compute $x_v \equiv x \pmod{P_v}$, but this is just $x_{f(v)} - \lfloor \frac{x_{f(v)}}{P_v} \rfloor P_v$. This can be done in depth $O(\log(ms))$ and size $O((ms^{1-(i-1)\delta})^{1+\epsilon})$ for any $\epsilon > 0$ provided that we have precomputed all P_v and $1/P_v$ and hardwired them into the circuit. This means that we can construct a circuit of total size $(ms)^{1+\epsilon}$ for any $\epsilon > \delta$ and depth $O(\frac{1}{\delta} \log n)$ which computes all the x_i.

For the inverse problem we only have to traverse the tree in the opposite direction using the formulas $x_v = \sum_{i=1}^{s^{\delta}} u_{i,v} x_{s(i,v)}$ and $x_v = x_v - \lfloor \frac{x_v}{P_v} \rfloor P_v$ where $s(i,v)$ is the i'th son of v and $u_{i,v}$ is the suitable multiplier. The numbers $u_{i,v}$ can be precomputed and hardwired into the circuit and thus we get the same bounds for the circuits as in the other case.

Thus we have the following theorem:

Theorem: 1 *Chinese remaindering with s m-bit moduli can be done by P-uniform circuits of depth $O(\frac{1}{\epsilon} \log ms)$ and size $O((ms)^{1+\epsilon})$ for any $\epsilon > 0$.*

4. Error analysis and implementation details.

We have two things to analyze: That the approximations done in the algorithm does not affect the accuracy of the output and how efficiently the algorithm can be implemented. Let us start with the former.

4.1 Error analysis.

We do approximations in two places. First we approximate $\frac{1}{z}$ by w and then we only compute x^{2^r} only approximately. The first approximation is harmless since

$$|w - \frac{1}{z}| = \sum_{i=2n}^{\infty} x^i \le \sum_{i=2n}^{\infty} 2^{-i} = 2^{1-2n}.$$

To estimate the error in computing x^{2^r}, first observe that there is no overflow in step 6b and thus the only error comes from the truncation. We have

Lemma 1: $|x^{2^{j \log n}} - a_j| \le 2^{1-2n}$ *for sufficiently large* n.

Proof: We use induction over j. Clearly the lemma is true for $j = 0$. Now observe that since there is no overflow in the computation in 6b we know that

$$|a_{j+1} - a_j^{2^{\delta \log n}}| \le 2^{-2n}$$

and by the mean value theorem

$$|x^{2^{(j+1)\delta \log n}} - a_j^{2^{\delta \log n}}| \le |x^{2^{j\delta \log n}} - a_j| 2^{\delta \log n} \theta^{2^{\delta \log n} - 1}$$

for some θ between a_j and $x^{2^{j\delta \log n}}$. Since both these numbers are between 0 and $2^{-2^{j\delta \log n}}$ this latter number is $\le 2^{-2n}$ for sufficiently large n. This finishes the proof of Lemma 1.

$\square$

By a similar analysis we get

Lemma 2: $|x^{2^r} - \tilde{x}_r| \le 2^{2-2n}$ *for sufficiently large* n *and all* $r \le \log n$.

Finally let us see what Lemma 2 implies for $\tilde{w}$

Lemma 3: $|\tilde{w} - w| \le 2^{4-2n} \log n$ *for sufficiently large* n.

Proof: Let $w_i = \prod_{r=0}^{i}(1+x^{2^r}) \prod_{r=i+1}^{\log n}(1+\tilde{x}_r)$, then $w_{-1} = \tilde{w}$ and $w_{\log n} = w$. By Lemma 2,

$$|w_i - w_{i+1}| \le 3|\tilde{x}_r - x^{2^r}| \le 2^{4-2n}$$

proving Lemma 3.

$\square$

4.2 Details of Implementation.

Steps 1-5 are quite straightforward and we need to elaborate further on step 6. To do the computation of the powers in 6 we will again use modular arithmetic. We will describe the procedure for one fixed prime p. The essential step is to compute $m^{2^k} \pmod{p}$ for some $k, 1 \le k \le \delta \log n$ where $m, p \le n^{1+\delta}$.

Choose primes $q_i, i = 1, \ldots t$ such that $\prod_{i=1}^{t} q_i > n^{(1+\delta)n^\delta}$. This can be done with $q_i < 2n^\delta \log n$ and $t < 2n^\delta \log n$ by [3] as before. We can compute $m_i \equiv m \pmod{q_i}$ using the theorem in section 3. To compute $m_i^k \pmod{q_i}$ we store lookup tables for x^{2^k} for

$1 \leq x < q_i$ and $1 \leq k \leq \delta \log n$. This only requires $O(n^\delta \log^2 n)$ circuitry for each q_i and since we have at most $2n^\delta \log n$ q_i's for every p and $O(n^{1+\delta})$ p's the total size of the lookup table is bounded by $O(n^{1+4\delta})$ and these lookups can clearly be done in depth $O(\log n)$.

We have no trouble finding $m^{2^k} \pmod{\prod_{i=1}^t q_i}$ using Chinese remaindering and since $m^{2^k} < \prod_{i=1}^s q_i$ we know it over the integers and can compute $m^{2^k} \pmod{p}$ using $\frac{1}{p}$ as a precomputed multiplier. By Theorem 1 also this can be done in size $O(n^{1+2\delta})$ and this finishes the description and analysis of step 6.

To do the computation described by 7 we first compute $c_{ir} = 2^{2n} + 2^{2n}\tilde{x}_r \pmod{p_i}$ $i = 1, 2 \ldots s$. c_{ij} are $O(\log n)$ bit numbers and hence we can compute $\prod_{r=0}^{\log n} c_{ir} \pmod{p_i}$ in a binary tree. Each multiplication requiring only $O(\log \log n)$ depth giving a total depth of $O((\log \log n)^2)$ and a $O(\log^2 n)$ bound for the number of processors.

Since 8 obviously is easy this finishes the description of how to construct the circuit and we have the following theorem.

Theorem 2: *For any $\epsilon > 0$ division has P-uniform circuits of depth $O(\frac{1}{\epsilon^2} \log n)$ and size $O(n^{1+\epsilon})$.*

We have essentially already given the proof of the theorem. Let us just observe that the expensive part of the algorithm is step 6 where we do $\frac{1}{\delta}$ Chinese remainderings in sequence. Each of these require $O(\frac{1}{\delta} \log n)$ depth. To get the bound on the number of processors use $\delta = \frac{\epsilon}{4}$.

5. Open questions and discussion.

It would be interesting to reduce the number of processors even further. There seems to be two reasons that we cannot get better bounds than $O(n^{1+\epsilon})$ by the present techniques. The first bottleneck is Chinese remaindering. The condition of $O(\log n)$ depth seems to force us to use a large circuit. The other reason for the bound is that we can only go back and forth between normal and modular representation a constant number of times and hence we are forced to have $O(n^{1+\epsilon})$ bits around and hence that size circuit. These two problems suggest that to reduce the number of processors further e.g. to $O(n \log^c n)$ for some constant c new ideas must be used. We feel that this should be possible. It would be interesting to be able to do division as efficiently as multiplication which is the case in the sequential model.

Our construction is as well as that of Beame, Cook and Hoover only P-uniform and not logspace-uniform. This might seem unimportant if we want to build the circuits but a logspace-uniform construction would be very interesting from a theoretical point of view. The reason for this is that by a general theorem by Borodin [2], such a construction would imply that division is in logspace. For a discussion of this question and reductions to related problems we refer to [1].

Slight variations of the construction in this paper give small $O(\log n)$ depth circuits for problems which can be done efficiently using modular arithmetic. Examples of such problems are iterated multiplication and powering.

References

[1] Beame P.W., Cook S.A. and Hoover H.J. “Log Depth Circuits for Division and Related Problems” *Proceedings 25'th Annual Symposium on Foundations of Computer Science*, 1984, pp 1-6.

[2] Borodin A. “On Relating Time and Space to Size and Depth” *SIAM J. Computing* 6, (1977), pp 733-744.

[3] Davenport H. “Multiplicative Number Theory” *Springer Verlag* 1980.

[4] Reif J. “Logarithmic Depth Circuits for Algebraic Functions” *Proceedings 24'th Annual Symposium on Foundations of Computer Science*, 1983, pp 138-145.

[5] Reif J. “Logarithmic Depth Circuits for Algebraic Functions” *Revised version* 1984.