DIMACS Technical Report 96-23
July 1996

Integer NC^1 is equal to Boolean NC^1

by

Stephen Bloch [1]
Math/Computer Science Dept.
Adelphi University
Garden City, NY 11415

[1]Visiting Researcher at DIMACS

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

We show that the product of n 3×3 matrices of n-bit integers can be computed in P-uniform FNC^1. Since this problem is complete [BOC92] for formulæ in $\{+, \cdot\}$ on n-bit integers, we conclude that "algebraic NC^1" on integers is equal to the usual Boolean notion of NC^1 functions.

1 Introduction

Throughout the history of parallel complexity theory, opinions have differed on how to charge basic arithmetic operations: should they be considered constant-cost, on grounds that for problems of practical size, they'll be carried out by fixed hardware components, or does that approach ignore significant scaling problems? Since addition, subtraction, multiplication, and (with strong enough uniformity) division are in FNC^1 (*i.e.* computable by uniform, multi-output, log-depth Boolean circuits), the two approaches obviously differ by at most a log factor of depth, but can this be improved?

We use the phrases "arithmetic circuit" and "algebraic circuit" interchangeably to refer to a circuit with (unit-cost) gates for the operations of a ring or semiring. An "arithmetic formula" or "algebraic formula" is defined as one would expect, and is clearly equivalent to an arithmetic circuit with fan-out 1. Formulæ and circuits therefore differ only in size, not in depth.

Jung [Jun85] gives an algorithm to simulate any log-depth arithmetic circuit family over the integers[1] with a Boolean circuit of depth $O(\log(n)\log^*(n))$. Eric Allender [All96] showed me a simplified version of this algorithm, using the more recent result [BOC92] that multiplication of n 3×3 matrices is complete for poly-size formula evaluation over arbitrary rings, and asked whether the depth bound could be improved to $O(\log(n))$. For at least the ring of integers, the answer is yes.

2 The Main Theorem

Let FNC^1 denote the function class computable by multi-output NC^1 circuit families (or, equivalently, the functions whose bit-graphs are NC^1 relations).

Theorem 1 *The problem of evaluating an $O(\log(n))$-depth arithmetic circuit family over n-bit integers is in P-uniform FNC^1.*

Proof: By [BOC92], the problem is reducible by p-projections (*a fortiori* in FNC^1) to finding the product of n 3×3 matrices of n-bit integers. By Lemma 2, this is in P-uniform FNC^1. ∎

The rest of this section is devoted to proving

Lemma 2 *The product of M 3×3 matrices of integers $\leq N$ can be computed by a P-uniform, bounded-fanin circuit family of depth $O(\log M)$, as long as $\log(N) < M^4$.*

[1] and mentions that the technique also works for $\mathbf{Q}$, $\mathbf{Z}$[x], and various other commutative rings

2.1 The Algorithm

The obvious algorithm for multiplying M matrices is a binary tree of depth $\log(M)$, multiplying two matrices at each vertex. Unfortunately, each entry in these matrices has length $n = O(\log N)$, so each matrix multiplication would take depth $O(\log\log N)$, for a total of $O(\log^2 M)$.

We solve this problem by using the Chinese Remainder Theorem to represent each n-bit number as a tuple of remainders mod various $\log(n)$-bit primes. These remainders, in turn, are represented as tuples of remainders mod various $\log\log(n)$-bit primes, and so on through $\log^*(n)$ stages[2] until each remainder is constant-length, and arithmetic operations on all the remainders can be performed in parallel by constant-depth, constant-fanin Boolean circuits. The basic algorithm is presented in pseudocode as function mainfunc in figure 2.1.

The analysis of mainfunc, however, only works when the numbers in the matrices are of comparable growth rate to the number of matrices. For application to Theorem 1, we need the *lengths* of the matrix entries comparable to the number of matrices, so we add prereduce and postrestore steps to reduce the size of the numbers, again by the Chinese Remainder Theorem, using the assumption $\log(N) < M^4$.

2.2 The M_i and N_i sequences

The parameter r in the algorithm indicates how many levels of nested prime moduli are in the representation. With parameter r, mainfunc can multiply M_r 3×3 matrices of numbers bounded by N_r, for suitable M_r and N_r. We shall impose four requirements:

1. $M_{i+1} > M_i$
2. $N_{i+1} > N_i$
3. the product of M_{i+1} 3×3 matrices of numbers $\leq N_{i+1}$ consists of numbers bounded by the product of all the primes $\leq N_i + 1$, thus ensuring that when mainfunc reduces its arguments, calls itself recursively, and restores the result, the result is correct over the integers. This requirement can be written algebraically as

$$N_{i+1}^{M_{i+1}} \cdot 3^{M_{i+1}-1} < \prod_{\text{prime } p \leq N_i+1} p$$

   or equivalently

$$(3N_{i+1})^{M_{i+1}} < 3 \cdot \left(\prod_{\text{prime } p \leq N_i+1} p \right).$$

4. the product of M 3×3 matrices of numbers $\leq N$ consists of numbers bounded by the product of all the primes $\leq N_r + 1$, thus ensuring that the correct answer over

[2] in principle; the algorithm as presented actually takes $\log\log n$ stages

Figure 1: the basic algorithm

$\mathsf{multiply}(A_1, \ldots, A_M, N)$:

assert each A_j is a 3×3 matrix of natural numbers $\leq N$
let r be such that $M_{r-1} < M \leq M_r$
for $i = 1$ to M in parallel,

  let $B_i = \mathsf{prereduce}(A_i, N_r, N)$

let $C = \mathsf{mainfunc}(B_1, \ldots, B_M, r)$
return $\mathsf{postrestore}(C, N_r, N)$

$\mathsf{mainfunc}(B_1, \ldots, B_M, r)$:

assert $M \leq M_r$
assert each B_j is a 3×3 matrix of natural numbers $\leq N_r$
(or tuples of such numbers, or tuples of tuples of such numbers, *etc.*)
let $p_1 < \cdots < p_k$ be the primes $\leq N_{r-1} + 1$
for $i = 1$ to M in parallel,

  let $C_i = \mathsf{reduce}(B_i, \langle p_1, \ldots, p_k \rangle)$

if $r = 1$,

  let D be the ordinary matrix product of $C_1, \ldots, C_M$
  (if entries are tuples, multiply pointwise in parallel)
  let $E = \mathsf{mod}(D, \langle p_1, \ldots, p_k \rangle)$

else

  for each vertex V of the M_{r-1}-ary tree with leaves $C_1, \ldots, C_M$,
    let $D = \mathsf{mainfunc}(\textit{ values of children}(V), r-1)$
    let $\textit{ value of } V = \mathsf{mod}(D, \langle p_1, \ldots, p_k \rangle)$
  let $E = \textit{ value of root of tree}$

return $\mathsf{restore}(E, \langle p_1, \ldots, p_k \rangle)$

Figure 2: support routines

prereduce(A, N_r, N):

> assert A is a 3×3 matrix of natural numbers $\leq N$
> (or tuples of such numbers, or tuples of tuples of such numbers, *etc.*)
> if $N_r \geq N$, return A
> else
>
> > let $p_1 < \cdots < p_k$ be all the primes $\leq N_r + 1$
> > return reduce$(A, \langle p_1, \ldots, p_k \rangle)$

postrestore(A, N_r, N):

> if $N_r \geq N$, return A
> else
>
> > let $p_1 < \cdots < p_k$ be all the primes $\leq N_r + 1$
> > return restore$(A, \langle p_1, \ldots, p_k \rangle)$

reduce(A, L):

> let $p_1, \ldots, p_k$ be the primes in L
> for $i = 1$ to 3, $j = 1$ to 3 in parallel,
>
> > let $B[i,j] = \langle A[i,j] \bmod p_1, \ldots, A[i,j] \bmod p_k \rangle$
> > (if $A[i,j]$ is a tuple, compute pointwise in parallel)
>
> return B

restore(A, L):

> let $p_1, \ldots, p_k$ be the primes in L
> for $i = 1$ to 3, $j = 1$ to 3, in parallel,
>
> > assert $A[i,j]$ is a tuple of k elements $a_1, \ldots, a_k$
> > do the following (if a's are tuples, do pointwise in parallel):
> >
> > > let $B[i,j]$ be the unique natural number $x < \prod_{t=1}^{k} p_t$ such that
> > > $(\forall t \leq k)(x \equiv a_t \pmod{p_t})$
>
> return B

mod(A, L):

> let $p_1, \ldots, p_k$ be the primes in L
> for $i = 1$ to 3, $j = 1$ to 3 in parallel,
>
> > assert $A[i,j]$ is a tuple of k elements $a_1, \ldots, a_k$
> > let $B[i,j] = \langle a_1 \bmod p_1, \ldots, a_k \bmod p_k \rangle$
> > (if a's are tuples, compute "mod" pointwise in parallel)
>
> return B

the integers can be correctly represented in the prime moduli chosen by prereduce and postrestore. This requirement can be written similarly as

$$(3N)^M < 3 \cdot \left(\prod_{\text{prime } p \le N_r+1} p \right).$$

These requirements leave us some freedom. For a first attempt, we shall choose $M_0 = 1$, $N_0 = 12$, *i.e.* the base level of representation involves remainders $\bmod\langle 2,3,5,7,11,13\rangle$. Since $\prod(2,3,5,7,11,13) = 30030$, we can then choose $M_1 = 2$ and $N_1 = 100$, which satisfies requirement 3 because $(3 \cdot 100)^2 = 90000 < 90090 = 3 \cdot \prod(2,3,5,7,11,13)$. We then choose $M_2 = 4$ and $N_2 = 540571993$, which the reader can check against requirement 3. Thenceforth, we choose

$$\begin{aligned} N_{i+1} &= N_i^2 \\ &\text{and} \\ M_{i+1} &= M_i^2 \end{aligned}$$

Clearly $M_i = 2^{2^{i-1}}$ and $N_i = 2^{\theta(2^i)}$; more precisely, there is a $c > 2$ such that for all $i \ge 2$, $N_i = 2^{2^{c+i}}$ (since $N_2 = 540571993 > 2^{29} > 2^{2^4} = 2^{2^{2+2}}$).

We must show that these M_i and N_i satisfy requirement 3. A corollary of the Prime Number Theorem is that

$$\lim_{x\to\infty} \frac{\vartheta(x)}{x} = 1,$$

where ϑ denotes Tchebyshev's ϑ function,

$$\vartheta(x) = \sum_{\text{prime } p \le x} \log(p).$$

Thus

$$\begin{aligned} \log\left(3\left(\prod_{\text{prime } p\le N_i+1} p\right)\right) &= \log(3) + \vartheta(N_i+1) \\ &= (1+o(1))(N_i+1) \\ &= (1+o(1))\left(2^{2^{c+i}}\right) \qquad (1) \end{aligned}$$

On the other side of requirement 3,

$$\begin{aligned} \log\left((3N_{i+1})^{M_{i+1}}\right) &= M_{i+1}(\log 3 + \log N_{i+1}) \\ &= 2^{2^i}\left(\log 3 + 2^{c+i+1}\right) \\ &= 2^{2^i+c+i+1} + O\left(2^{2^i}\right) \\ &= 2^{2^i+o\left(2^i\right)} \qquad (2) \end{aligned}$$

Since $c > 0$, it is clear that expression 1 asymptotically dominates expression 2, so requirement 3 is satisfied all but finitely often. If it so happens that the values of N_i we've chosen above aren't large enough to avoid the finitely many exceptions, go back and make N_0 (and hence N_1, N_2, and c) larger; the algorithm and the M_i's are unchanged.

We must also show that requirement 4 is satisfied. Recall that, by hypothesis,

$$
\begin{array}{rcl}
\log N & < & M^4 \\
& \leq & M_r^4 \\
& = & 2^{2^{r+1}} \\
& < & 2^{2^{c+r}-(2^{r-1}+1)} \text{ a.e. because } c > 1 \\
& = & N_r/(2M_r) \\
& < & \vartheta(N_r)/M_r - \log 3 \text{ a.e.} \\
M_r(\log 3 + \log N) & < & \vartheta(N_r) \text{ a.e.} \\
(3N)^{M_r} & < & \displaystyle\prod_{\text{prime } p \leq N_r} p \text{ a.e.} \\
& < & 3\left(\displaystyle\prod_{\text{prime } p \leq N_r} p\right)
\end{array}
$$

Since $M \leq M_r$, this requirement too is satisfied all but finitely often. In fact, each "a.e." above can be made true by increasing N_0 and hence c, so as in the previous paragraph, we do so as necessary.

2.3 Complexity Analysis

First, we observe that since `reduce` and `restore` can be done in (P-uniform) depth $O(\log\log(p_k))$, where p_k is the largest prime in question, the `prereduce` and `prerestore` functions take total depth $O(\log\log(N_r)) = O(c+r) < O(2^{r-1}) = O(\log(M))$, as desired.

Next, consider the depth needed for matrix multiplication itself. If we examine `mainfunc` and momentarily ignore all the calls to `reduce`, `restore`, and `mod`, we find that it reduces to a simple binary tree of matrix multiplications (with many independent multiplications happening in parallel at each vertex). Each matrix multiply involves only 2 matrices of numbers bounded by the constant N_0, and can therefore be done in constant depth. This tree obviously has depth $O(\log M_r)$, so the total depth of the algorithm, ignoring `reduce`, `restore`, and `mod`, is $O(\log M_r)$. Since r is chosen as small as possible subject to $M_r \geq M$, we get

$$
\begin{array}{rcl}
r & = & 1 + \lceil \log\log(M) \rceil \\
\log(M_r) & = & 2^{\log\log(M)+O(1)} \\
& = & O(\log M)
\end{array}
$$

as desired.

Finally, we must show that the total depth cost of reduce, restore, and mod throughout the algorithm is of the same order. Consider the calls made at the i-th level of the algorithm, *i.e.* reducing, mod-ding, and restoring mod primes $\leq N_{i-1}+1$. This happens each time a group of M_i matrices are multiplied together, *i.e.* once each $\log M_i$ levels of the binary tree, and it costs depth $O(\log\log p_{k_i}))$, where p_{k_i} is the largest prime $\leq N_{i-1}+1$. Thus the average cost of reduction, mod, and restoration per level of the multiplication tree is

$$\begin{aligned}
\sum_{i=1}^{r} \frac{O(\log\log(p_{k_i}))}{\log M_i} &\leq \sum_{i=1}^{\infty} \frac{O(\log\log(p_{k_i}))}{\log M_i} \\
&= \sum_{i=1}^{\infty} \frac{O(\log\log(N_{i-1}))}{\log M_i} \\
&\leq \sum_{i=1}^{\infty} \frac{O(\log\log(2^{2^{c+i-1}}))}{2^{i-1}} \\
&= \sum_{i=1}^{\infty} \frac{O(c+i-1)}{2^{i-1}} \\
&< \infty
\end{aligned}$$

In other words, inserting reduce, restore, and mod into the multiplication tree multiplies its depth by only a constant, and thus the depth of the entire algorithm is $O(\log M)$.

3 Applications and Open Questions

[CMTV96] defines counting classes $\#NC^1$, $GapNC^1$, PNC^1, and $C_=NC^1$ over NC^1 and shows that these classes have equivalent formulations over Boolean circuits, algebraic circuits, and several other models of parallel computation. (More precisely, $\#NC^1$ comprises exactly the functions computable by uniform $O(\log)$-depth algebraic formulæ over the semiring $\mathbf{N}$; $GapBP = GapNC^1$ comprises exactly the functions computable by uniform $O(\log)$-depth algebraic formulæ over the ring $\mathbf{Z}$; PNC^1 comprises exactly the relations expressible as $F_1 \leq F_2$, where F_1 and F_2 are $O(\log)$-depth algebraic formulæ over $\mathbf{N}$ or $\mathbf{Z}$; and $C_=NC^1$ comprises exactly the relations expressible as $F_1 = F_2$ similarly.) The present result shows that not only counting classes over NC^1, but FNC^1 itself, at least in the P-uniform case, is equivalent whether formulated over Boolean or $\mathbf{Z}$-arithmetic circuits. Furthermore, an FNC^1 Boolean circuit family can compute the value of an $O(\log)$-depth algebraic formula (over $\mathbf{N}$ or $\mathbf{Z}$), and therefore the number of accepting trees in an NC^1 Boolean circuit family. Thus at least in the P-uniform case, $\#NC^1 = GapNC^1 = FNC^1$, and $PNC^1 = C_=NC^1 = NC^1$.

[GLPG82] studies the problem of computing the first N terms of an M-order linear recurrence, and gives $O((\log M)(\log N))$-depth arithmetic circuits to do the job. Since their technique involves reducing the linear recurrence problem to various products of large numbers of $M \times M$ matrices, and since Lemma 2 holds with obvious modifications for square matrices of any fixed size, we can show that for any fixed M, the problem of computing the first N terms of an M-order linear recurrence is in $O(\log N)$ depth on P-uniform Boolean circuits.

An obvious direction for further research (perhaps by someone who knows more number theory than the author) is to replace the 3×3 matrices of this paper with $T\times T$. The algorithm should require only minimal changes, but the analysis may become much more complicated: in particular, it would be necessary to estimate how rapidly N_0, c, and the "finitely many exceptions" in section 2.2 grow with T. If all the requirements could be met with N_0, N_1, and N_2 growing only polynomially in T, then the depth of the multiplication tree (ignoring reduce *et al*) would be $O((\log T)(\log N))$. The infinite sum in section 2.3 increases linearly with c, which is $O(\log\log(N_2))$, for a total depth of $O((\log T)(\log\log T)(\log N))$; perhaps $O((\log T)(\log N))$ could be achieved with more tweaking and tighter analysis, thus (among other things) improving the arithmetic results of [GLPG82] to work on Boolean circuits.

Another intriguing open question is extending the result to a broader class of rings or semirings, *e.g.* rationals (with n-bit numerators and denominators), polynomials (with n-bit-integer coefficients), or families of finite (semi)rings with at most 2^n elements. The results of [GLPG82, BOC92] apply to an arbitrary ring. [Jun85] claims that the related technique there is applicable to $\mathbf{Q}$ and $\mathbf{Z}$[x], and comments on the implications of extending its results to the ring of integer matrices (such as probabilistic logspace $= NC^1$!)

It would likewise be very interesting to weaken the uniformity requirement in the present algorithm to ALOGTIME, or even LOGSPACE. The obvious way to do this is to weaken the uniformity requirement on FNC^1 algorithms for division, which would be a major result in itself. Another approach, as pointed out in [Jun85], would be to replace the moduli "all the primes $\leq N_i + 1$" in the algorithm with a more uniformly constructible sequence of numbers $\{b_j\}$ such that $(3N_{i+1})^{M_{i+1}} < 3\cdot\mathrm{LCM}\{b_j\}_{b_j\leq N_i+1}$ and $(3N)^{M_r} < 3\cdot\mathrm{LCM}\{b_j\}_{b_j\leq N_r+1}$, for some sequences $\{N_i\}$ and $\{M_i\}$ of suitable growth rates.

Acknowledgments

My deepest thanks to Eric Allender for suggesting the problem, for pointing me to related work, and for critiquing several attempted solutions. Thanks also to DIMACS for supporting the visit to Rutgers during which this work was done.

References

[All96] Eric Allender. Personal communication, June 1996.

[BOC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 22(1):54–58, February 1992.

[CMTV96] Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. In *Proceedings of the Eleventh Annual Computational Complexity Conference*, pages 12–21. IEEE Computer Society Press, 1996.

[GLPG82] Albert C. Greenberg, Richard E. Ladner, Michael S. Paterson, and Zvi Galil. Efficient parallel algorithms for linear recurrence computation. *Information Processing Letters*, 15(1), August 1982.

[Jun85] Hermann Jung. Depth efficient transformations of arithmetic into Boolean circuits. In Lothar Budach, editor, *Proceedings of 1985 Conference on Fundamentals of Computation Theory*, number 199 in Lecture Notes in Computer Science, pages 167–174. Springer-Verlag, 1985.