In Proceedings of CRYPTO-85, eds. E. Brickell and H. Williams, Springer, 1986, pp. 433-446

Efficient Parallel Pseudo-Random Number Generation

J. H. Reif[1]

J. D. Tygar[2]

Aiken Computation Laboratory

Harvard University

Cambridge, MA 02138

0. Abstract

We present a parallel algorithm for pseudo-random number generation. Given a seed of n^ϵ truly random bits for any $\epsilon > 0$, our algorithm generates n^c pseudo-random bits for any $c > 1$. This takes poly-log time using $n^{\epsilon'}$ processors where $\epsilon' = k\epsilon$ for some fixed small constant $k > 1$. We show that the pseudo-random bits output by our algorithm can not be distinguished from truly random bits in parallel poly-log time using a polynomial number of processors with probability $1/2 + 1/n^{O(1)}$ if the multiplicative inverse problem almost always can not be solved in **RNC**. The proof is interesting and is quite different from previous proofs for sequential pseudo-random number generators.

Our generator is fast and its output is provably as effective for **RNC** algorithms as truly random bits. Our generator passes all the statistical tests in KNUTH[14].

Moreover, the existence of our generator has a number of central consequences for complexity theory. Given a randomized parallel algorithm $\mathcal{A}$ (over a wide class of machine models such as parallel RAMs and fixed connection networks) with time bound $T(n)$ and processor bound $P(n)$, we show $\mathcal{A}$ can be simulated by a parallel algorithm with time bound $T(n) + O((\log n)(\log\log n))$, processor bound $P(n)n^{\epsilon'}$, and only using n^ϵ truly random bits for any $\epsilon > 0$.

[1]Supported in part by NSF grant NSF-MCS-79-21024 and ONR contract N0014-80-C-0674.

[2]Supported in part by a NSF graduate fellowship and NSF grant MCS-81-21431.

Also, we show that if the multiplicative inverse problem is almost always not in **RNC**, then **RNC** is within the class of languages accepted by uniform poly-log depth circuits with unbounded fan-in and strictly sub-exponential size $\bigcap_{\epsilon>0} 2^{n^\epsilon}$.

1. Introduction

A number of parallel randomized algorithms have appeared recently. These algorithms typically use a large number of random bits which must be generated in a small amount of time. Nonetheless, the area of parallel random bit generation remains unexplored.

In reality, our computers are deterministic and unable to generate truly random values. But we can give algorithms which will give pseudo-random bits on input of a random seed s_0. These pseudo-random bits satisfy conditions which suggest that for algorithmic purposes they are as effective as truly random bits.

What conditions should a pseudo-random bit sequence satisfy?

Improving an idea by SHAMIR[17], BLUM-MICALI[6] argue that the notion of "cryptographic strength" captures the important facets of random sequences. To demonstrate cryptographic strength they follow this schema:

1. Upper bound the computational resources by Resources A.
2. Assume that Problem B cannot be solved within the limits of Resources A.
3. Produce a Pseudo-Random Bit Generator G
4. Argue that if an opponent sees the first m_0 bits generated by Pseudo-Random Bit Generator G and can utilize Resources A to predict the remaining bits with an accuracy rate of $1/2 + \epsilon(m)$ (where m is the size of the seed and ϵ is a fixed function satisfying $\lim_{m\to\infty} \epsilon(m) = 0$), then the opponent will be able to solve Problem B limited to Resources A by consulting the bit-guessing oracle, a contradiction.

Several cryptographically-strong pseudo-random bit generators have been proposed (BLUM-BLUM-SHUB[5], BLUM-MICALI[6],) and many applications have been discussed (ALEXI-CHOR-GOLDREICH-SCHNORR[3], GOLDREICH-GOLDWASSER-MICALI[9], GOLDWASSER-MICALI-TONG[10], VAZIRANI-VAZIRANI[20], YAO[22].) These generators are all inherently sequential, require polynomial time, and their cryptographic strength relies on some unproven cryptographic assumption.

Notation

When we say a class of circuit is *uniform*, we mean that it is constructible in logarithmic space by a deterministic Turing Machine.

NC ($\mathbf{NC_U}$) is the class of languages accepted by (uniform, respectively) deterministic circuits with poly-log depth and polynomial size.

RNC ($\mathbf{RNC_U}$) is the class of languages accepted by (uniform, respectively) randomized circuits with one-sided error, poly-log depth, polynomial size, and acceptance probability greater than $1/2$.

We give more precise definitions of these terms in section 4.

Our Result

We present a new cryptographically-strong pseudo-random bit generator which runs in $\mathbf{NC_U}$ but which is secure against attacks taking parallel poly-log time if the multiplicative inverse problem almost always is not in **RNC**. While we use the schema described above for demonstrating the cryptographic strength of our random number generator, because of the inherent parallel nature of our generator, the technical details of our proof are quite different from those of previous proofs for sequential pseudo-random number generators. In particular, we prove that if the bits output by our pseudo-random bit generator can be predicted in **NC**, then we can solve the multiplicative inverse problem in **RNC** almost always and this requires that we construct an interesting, nontrivial, parallel algorithm for that problem. (See section 3.)

About the Assumption

While our assumption has not been proved, it is quite interesting to observe that it is *testable* in the following sense: If a **RNC** algorithm takes more than poly-log time using our pseudo-random bits instead of truly random bits then we can observe this event by timing. Thus one of two scenarios is possible: either every application of our generator to a **RNC** algorithm yields a poly-log algorithm using only a small number of random bits, or some application of our generator is discovered to exceed its poly-log time bounds and we can immediately derive a **NC** algorithm for multiplicative inverse.

About the Measure of Randomness

VALIANT-SYKUM-BERKOWITZ-RACKOFF[19] show that an NC-machine can evaluate any straight-line program which computes a multivariate polynomial which has degree polynomial in the length of the program. Thus if our assumption is correct, our pseudo-random bit generator is secure against any statistical test which can be so formulated as a straight-line program. This includes most standard statistical tests for random number generators. (KNUTH[14])

Applications

Our method for parallel pseudo-random bit generation is actually very practical. It requires, for any $\epsilon > 0$, only $O(\log n(\log\log(n)))$ added depth and a factor of n^ϵ for a bounded fan-in circuit. Here is an example: KARP-WIGDERSON[12] gives a deterministic algorithm for the maximal independent set problem in $O((\log n)^4)$ time using $O(n^3/(\log n)^3)$ processors. They also give a uniform randomized algorithm for the same problem running in $O((\log n)^3)$ expected time with $O(n^2)$ processors using $O(n^2)$ random bits. Our results immediately yield an uniform algorithm with $O((\log n)^3)$ running time and $O(n^{2+\epsilon'})$ processors using only n^ϵ random bits, where $\epsilon, \epsilon' > 0$ can be set arbitrarily small.

Recently KARP-UPFAL-WIGDERSON[13] have shown that finding a maximum graph matching is in $\mathbf{RNC_U}$, and ANDERSON[2] has shown that finding a maximal path is in $\mathbf{RNC_U}$. Our results also immediately yield efficient randomized uniform algorithms for these problems, using only n^ϵ bits for any $\epsilon > 0$.

In further work REIF-TYGAR[16], we have applied results given in this paper to prove randomness properties of rational linear iterative maps modulo 1.

Implications

An interesting theoretical application of our result is that $\mathbf{RNC_U}$ is contained within the class of languages recognized by uniform deterministic circuits of unbounded fan-in with poly-log depth and 2^{n^ϵ} size for any $\epsilon > 0$. (ADLEMAN[1] proved $\mathbf{RNC_U}$ is contained in (non-uniform) NC, but the previous best construction for bounding $\mathbf{RNC_U}$ by deterministic uniform circuits of poly-log depth required $2^{\Omega(n)}$ size.) This extends a result of YAO[22] for sequential polynomial time computations to poly-log time parallel computations.

2. Definitions and Results

Notation

We use the following notation throughout the paper:

N A positive composite integer such that each prime factor of N is greater than N^c for a fixed $c > 0$.

$\mathbf{Z}_N^*$ The multiplicative group of positive integers less than and relatively prime to N. (Note that the fact that N has only large factors implies that a random positive integer less than N is an element of $\mathbf{Z}_N^*$ with high probability.)

We will sometimes use x mod N to indicate the residue of x modulo N.

Definitions

A **NC-*machine*** (COOK[8]) is a deterministic parallel algorithm which runs on $n^{O(1)}$ P-RAM processors in time $(\log n)^{O(1)}$ for input of size n. (Note that $\mathbf{NC_U}$ is the class of languages accepted by **NC**-machines.)

A **RNC-*machine*** is a randomized parallel algorithm which runs on $n^{O(1)}$ P-RAM processors in time $(\log n)^{O(1)}$ for input of size n. (Note that $\mathbf{RNC_U}$ is the class of languages accepted by **RNC**-machines.)

Given $s_0 \in \mathbf{Z}_N^*$, the *multiplicative inverse* of s_0 modulo N is the s_0^{-1} such that $s_0 s_0^{-1} = 1 \bmod N$.

For a fixed N, given an arbitrary $k \in \mathbf{Z}_N^*$, the multiplicative inverse problem is to find the multiplicative inverse of k modulo N. Note that the input size to the problem is $n = \lceil \log N \rceil$

The problem of finding multiplicative inverses in poly-log depth has been studied extensively. (COOK[8], KANNAN-MILLER-RANDOLF[11], REIF[15], VON ZUR GATHEN[21].) Based on the lack of significant positive results obtained so far we conjecture:

Complexity Hypothesis

There exists an inifinite sequence of numbers $N_1, N_2, \ldots$ constructable in $\mathbf{NC_U}$ such that for each $n = 1, 2, \ldots$ we have $n = \lceil \log N_n \rceil$ and that for almost all n, no **RNC**-machine exists

which can on arbitrary input from $\mathbf{Z}^*_{N_n}$ solve the multiplicative inverse problem on any of those elements.

(Actually we could replace this complexity assumption with the weaker assumption that there exists a k such that for almost all n there exists an n' such that $n < n' < n^k$ and no RNC-machine can solve multiplicative inverse problem where the input can again range over arbitrary elements of $\mathbf{Z}^*_{N_{n'}}$. All the theorems in this paper would remain true under that weaker assumption.)

Definitions

A set S of bit sequences $\sigma = (b_1, \ldots, b_J)$ of length $J = n^{O(1)}$ pseudo-random bits is **RNC**-*cryptographically strong* if no **RNC**-machine can, on a random input $b_1, \ldots, b_i \in \sigma$ $(i < J, \sigma \in S)$ predict any one bit $b_{i+1}, \ldots, b_J$ with expected success of $1/2 + 1/n^{O(1)}$. Informally, the bit sequences are **RNC**-cryptographically strong if no **RNC**-machine can predict untransmitted bits with an expected success rate significantly better than $1/2$.

Theorem

If the complexity hypothesis holds there exists a deterministic **NC**-machine $\mathcal{G}$ which on an input seed of n bits outputs a **RNC**-cryptographically strong sequence of $J = n^{O(1)}$ pseudo-random bits. $\mathcal{G}$ can be computed by a bounded fan-in uniform boolean circuit of depth $O((\log n)(\log\log n))$ and size $n^{O(1)}$.

This theorem is proved in section 3.

Definition

A **RNC**-*statistical test* is a **RNC**-*machine* which attempts to distinguish truly random bit sequences from pseudo-random bit sequences. A statistical test succeeds if it correctly distinguishes the pseudo-random bit sequences from truly random bit sequences with probability at least $1/n^{O(1)}$.

By a technique due to YAO[22] we can show that no **RNC** statistical test can succeed on **RNC**-cryptographically strong bit sequences. Hence:

Corollary 1

If the complexity hypothesis holds then no **RNC**-statistical test can succeed on our pseudo-random bit generator $\mathcal{G}$.

Corollary 2

If the N_n are constructable in depth $h(n)$, then given a randomized parallel algorithm $\mathcal{A}$ (over a wide class of machine models such a parallel RAMS and fixed connection networks) with time bound $T(n)$ and processor bound $P(n)$, then $\mathcal{A}$ can be simulated by a parallel algorithm with time bound $T(n) + h(n) + O((\log n)(\log\log n))$, processor bound $P(n)n^{\epsilon'}$, and only using n^{ϵ} truly random bits for any $\epsilon > 0$, where $\epsilon' = O(\epsilon)$.

$\text{CIRCUIT}_U(D(n), S(n))$ is the class of languages accepted by uniform deterministic circuits with unbounded fan-in, depth $D(n)$, and size $S(n)$. (See section 4 for a precise definition of these complexity classes.)

Corollary 3

If the complexity hypothesis holds then

$$\mathbf{RNC}_U \subseteq \bigcup_{c>0} \bigcap_{\epsilon>0} \text{CIRCUIT}_U((\log n)^c, 2^{n^\epsilon})$$

This corollary is proved in section 4.

Corollary 4

There exists a cryptosystem where encryption and decryption can be done by a **NC**-machine on $n^{O(1)}$ bits given a secret shared key exactly n bits long (here n is a security parameter). If no **RNC**-machine can solve the multiplicative inverse problem then no **RNC**-machine can decrypt ciphertext exchanged in this cryptosystem.

We use the pseudo-random bits as a "one-time pad" — we take the sequential exclusive-or of the plaintext and the pseudo-random bits to produce the ciphertext and take the sequential exclusive-or of the ciphertext and the pseudo-random bits to obtain the plaintext again. Encryption and decryption both take parallel poly-log time but an opponent cannot decrypt the ciphertext with **RNC**-machine.

3. The Proof of the Main Theorem

Properties

We recall the following facts which we use implicitly (BEAME-COOKE-HOOVER[4], REIF[15], SHONHAGE-STRASSEN[18]):

- There exists a NC-machine for multiplication of two numbers in $\mathbf{Z}_N^*$.
- $2 \log p$ multiplications suffice to find the p^{th} power of a number in $\mathbf{Z}_N^*$.
- If $p < (\log N)^{O(1)}$, there exists a NC-machine for finding the p^{th} power of a number in $\mathbf{Z}_N^*$.

Fix $m = \lceil \log N \rceil$ throughout this section.

Let $\mathcal{G}$ be the NC-machine which performs the following operations:

Input: random elements $s_0, k \in \mathbf{Z}_N^*$.

Output: $b_1, \ldots, b_J$ where $J = m^{O(1)}$.

Method: In parallel each processor P_i $(i = 1, \ldots, J)$ calculates $s_i = k s_0^i \bmod N$ and $b_{J-i+1} = B(s_i)$ where

$$B(x) = \begin{cases} 0 & \text{if } x \leq N/2 \\ 1 & \text{if } x > N/2 \end{cases}$$

Lemma

If there exists a **RNC**-machine which can determine the value of b_J with probability 1 (i.e., no error) on input $b_1, \ldots, b_{J-1}$, then there exists a **RNC**-machine which can solve the multiplicative inverse problem for $\mathbf{Z}_{N_n}^*$.

Proof of Lemma

Suppose that MB (for "magic box") is an oracle which can determine the value of b_J with probability 1. Then given $s_0 \in \mathbf{Z}_N^*$ we can find $s_0^{-1} \bmod N$. We can find this by running in parallel the following algorithm on each processor P_j for $(0 \leq j \leq m)$:

Set $k \leftarrow 2^j$. In parallel set $b_i \leftarrow B(k s_0^{J-i-1})$ for $1 \leq i \leq J-1$. Note that $b_J = B(2^j s_0^{-1})$. Feed the sequence $(b_1, \ldots, b_{J-1})$ to MB to get b_J. Set the j^{th} most significant bit of δ to be

$B(2^j s_0^{-1})$. Define

$$\phi(\delta) = \left\lceil \frac{\delta N}{2^m} \right\rceil$$

Then $\phi(\delta) = s_0^{-1} \bmod N$. □

Theorem

If there exists a **RNC**-machine which can determine the value of b_J with probability at least $1/2+1/m^{O(1)}$ on input $b_1, \ldots, b_{J-1}$ then there exists a **RNC**-machine $\mathcal{H}$ which can solve the multiplicative inverse problem for $\mathbf{Z}^*_{N_n}$. $\mathcal{H}$ can be computed by a bounded fan-in boolean circuit of depth $O((\log n)(\log\log n))$ and size $n^{O(1)}$.

Proof of Theorem

Assume that that there exists a **RNC**-machine MB which can predict b_J with probability $1/2 + 2/m^c$. Let $H = 2(c+1)\lceil \log m \rceil$. Let δ and ϕ be as in the proof of the lemma.

Let $S = \{0, 1, \ldots, 2^{H-1} - 1\}$. For each $0 \leq y < x \leq m$, we will create, by randomized methods, two functions $F_{x,y} : S \to \{0,1\}^{x-y}$ and $G_{x,y} : S \to S$. Informally, values in S are guesses; $F_{x,y}$ is a rule for transforming a guess $j_x \in S$ into the x^{th} to y^{th} most significant bits of δ; and $G_{x,y}$ is a rule for transforming the guess $j_x \in S$ into the guess $j_y \in S$.

If a **RNC**-machine could find δ for arbitrary s_0, we could solve the multiplicative inverse problem. It will turn out that for some $j_m \in S$, that $F_{m,0}(j_m) = \delta$ with probability $1/2$. We can verify this occurence simply by checking whether $s_0\phi(\delta) = 1 \bmod N$. If we don't immediately find $s_0^{-1} \bmod N$, we simply form a new $F_{m,0}$ by randomized methods, and continue testing until we do find $s_0^{-1} \bmod N$.

Suppose we can determine j_x such that we know that $(2^x s_0^{-1} \bmod N)$ belongs to one of the two intervals

$$\left\{ \left[\left\lceil \frac{j_x N}{2^H} \right\rceil, \left\lfloor \frac{(j_x+1)N}{2^H} \right\rfloor \right], \left[\left\lceil \frac{(2^{H-1}+j_x)N}{2^H} \right\rceil, \left\lfloor \frac{(2^{H-1}+j_x+1)N}{2^H} \right\rfloor \right] \right\}$$

We can pick 2^H random values $\beta \in \mathbf{Z}^*_N$ and let v be MB's prediction for

$$B(2^x s_0^{-1} - \left\lfloor \frac{N j_x}{2^H} \right\rfloor + \beta \bmod N).$$

When β lies in the interval

$$\left[0, \left\lfloor \frac{(2^{H-1}-1)N}{2^H} \right\rfloor\right],$$

mark a vote for v, when β lies in the interval

$$\left[\left\lceil\frac{N}{2}\right\rceil, \left\lfloor\frac{(2^H-1)N}{2^H}\right\rfloor\right],$$

mark a vote the complement of v, and mark a *null* vote when β lies in other intervals. By assumption, MB predicts correctly with probability at least $1/2 + 2/m^c$.

We can assign a processor to calculate MB's prediction for each of the 2^H randomly chosen values of $\beta \in \mathbf{Z}_N^*$. This computation can be done in poly-log time for each β. The expected fraction of *null* votes is $2^{1-H} < 1/m^c$. Thus we have a bias of at least $2/m^c - 1/m^c = 1/m^c$ between **0** and **1** votes. Set $F_{x,x-1}(j_x)$ (our guess for $B(2^x s_0^{-1} \bmod N)$) to be a value which got the most votes. If our guess for $B(2^x s_0^{-1} \bmod N)$ is right, this immediately identifies which of the two intervals that $(2^x s_0^{-1} \bmod N)$ belongs to. 2^H tests are sufficient to make our guess correct with probability at least $1 - 1/2^m$. This result follows immediately from Chernoff bounds (CHERNOFF[7]); full details will appear in the complete paper. If our guess is right, that immediately determines the value of j_{x-1}; that is, we can determine that $(2^{x-1}s_0^{-1} \bmod N)$ lies in one of the two intervals

$$\left\{\left[\left\lceil\frac{j_{x-1}N}{2^H}\right\rceil, \left\lfloor\frac{(j_{x-1}+1)N}{2^H}\right\rfloor\right], \left[\left\lceil\frac{(2^{H-1}+j_{x-1})N}{2^H}\right\rceil, \left\lfloor\frac{(2^{H-1}+j_{x-1}+1)N}{2^H}\right\rfloor\right]\right\}$$

namely

$$j_{x-1} = G_{x,x-1}(j_x) = \lfloor j_x/2 \rfloor + 2^{H-2}(F_{x,x-1}(j_x))$$

We can calculate in parallel, for each $m \geq x \geq 1$, the functions $F_{x,x-1}$ and $G_{x,x-1}$, since the domain is finite and of polynomial size. If $x - y > 1$, then $F_{x,y}$ and $G_{x,y}$ can be recursively defined as

$$F_{x,y}(j_x) = F_{x,y}(G_{x,z}(j_x))2^{z-z} + F_{z,z}(j_z)$$

and

$$G_{x,y}(j_x) = G_{z,y}(G_{x,z}(j_x))$$

where $z = \lceil (x+y)/2 \rceil$. For each x, y pair $(0 \leq y < x \leq m)$ and each $j_x \in S$ we repeatedly calculate the appropriate compositions of these functions for all j_x in the domain of the functions. Thus we can compute $F_{m,0}$ in $\lceil \log m \rceil$ stages.

Some guess j_m is correct. Suppose that for all $1 \leq i \leq m$, that (1) $G_{i,i-1}(j_i)$ is the correct value of j_{i-1}. Then (2) $F_{m,0}(j_m)$ would be the correct value of δ. For each i, the probability that (1) is true for a particular j_i is $(1 - 2^{-m})$, so the probability that (2) is true is $(1-2^{-m})^{m-1} > 1 - (m-1)2^{-m} > 1/2$.

For some $j_m \in S$, it will be true that $F_{m,0}(j_m) = \delta$ with probability $1/2$. We can try all possible j_m in parallel, and find out if we have a correct value by checking whether $\phi(F_{m,0}(j_m))s_0 = 1 \bmod N$. (Of course, it might happen that an incorrect guess for j_m might give a correct value for δ but this can only speed the calculation.) In the event that we do not get the correct value for $s_0^{-1} \bmod N$, we simply form new $F_{x,y}$ and $G_{x,y}$ functions and continue until we do get the correct value. □

4. Randomized and Deterministic Parallel Complexity

Let C be a list of circuits $(C_1, C_2, \ldots)$ of unbounded fan-in where C_n has n inputs and size $S(n)$. We consider C to be *uniform* if there exists a $(\log S(n))$ space deterministic Turing machine which, given any n, outputs the circuit C_n. Let CIRCUIT$(D(n), S(n))$ be the class of all languages accepted by deterministic boolean circuits with unbounded fan-in, depth $D(n)$, and size $S(n)$. As usual we define

$$\mathbf{NC} = \bigcup_{k_1>0, k_2>0} \text{CIRCUIT}((\log n)^{k_1}, n^{k_2})$$

We allow a *randomized boolean circuit* C to have r special nodes each of which are assigned independent random bits chosen from $\{0,1\}$ with equal probability. C *accepts* an input $\omega \in \{0,1\}^n$ if C outputs 1 with probability $> 1/2$; otherwise C *rejects* the input. For simplicity, we consider only one-sided error randomized circuits which never ouput a 1 on an input they have rejected. (The construction below can easily be extended to two-sided error randomized circuits which have an acceptance probability of at least $1/2 + 1/n^k$ for some $k > 1$.) Let RCIRCUIT$(D(n), S(n))$ be the class of languages accepted by randomized circuits with unbounded fan-in, depth $D(n)$, and size $S(n)$. We define

$$\mathbf{RNC} = \bigcup_{k_1>0, k_2>0} \text{RCIRCUIT}((\log n)^{k_1}, n^{k_2})$$

We define CIRCUIT$_U$, $\mathbf{NC}_U$, RCIRCUIT$_U$, and $\mathbf{RNC}_U$ analogously — restricting the circuits to be uniform.

Corollary 3

If the complexity hypothesis holds then

$$\mathbf{RNC}_U \subseteq \bigcup_{c>0} \bigcap_{\epsilon>0} \text{CIRCUIT}_U((\log n)^c, 2^{n^\epsilon})$$

Proof

Let C be a (one-sided error) uniform randomized boolean circuit with n inputs, depth $D(n) = (\log n)^{k_1}$, and size $S(n) = n^{k_2}$. Fix any $\epsilon > 0$.

First suppose we had a source of $b = \lceil n^{\epsilon/2} \rceil$ truly random bits. Observe that C uses at most $S(n) = n^{k_2}$ random bits on each execution. Since $S(n) \leq b^{\epsilon'}$ where $\epsilon' = \lceil \epsilon/k_2 \rceil$ is constant, we can apply our parallel pseudo-random bit generator $\mathcal{G}$ to produce $S(n)$ pseudo-random bits in $(\log n)^{O(1)}$ parallel time using $n^{O(1)}$ processors and using the b truly random bits as the seed. We can view the execution of C on the given input ω as a statistical test. By Corollary 2, given an input $\omega \in \{0,1\}^n$, we need only execute C on ω for each of the 2^b possible pseudo-random bit sequences. We accept ω if C ever outputs 1.

Furthermore, we can avoid the use of a truly random seed by simply (1) enumerating all b-bit numbers in parallel; (2) executing the parallel pseudo-random bit generator using each of the b-bit numbers as a seed; and (3) executing C in parallel on ω on each of the resulting pseudo-random bit sequences. If C ever outputs 1 we accept ω. The resulting uniform circuit requires size $2^{b^2} \leq 2^{n^\epsilon}$ and depth $(\log n)^{O(1)} + O(D(n)) = (\log n)^{O(1)}$. □

Note that if we require that our simulation circuit have bounded fan-in, then to simulate a circuit accepting a language in $\mathbf{RNC_U}$, we require $n^{O(1)}$ (rather than $(\log n)^{O(1)}$ depth) and 2^{n^ϵ} size. This is an improvement over previous size bounds for $\mathbf{RNC_U}$.

6. Acknowledgements

We would like to thank Michael Rabin for being an inspiration to us in the fields of randomized algorithms and cryptography.

We are indebted to Silvio Micali for his many helpful and insightful comments on this manuscript.

Also, thanks to Benny Chor, Shafi Goldwasser, Johan Hastad, Brian O'Toole, Charles Rackoff, Les Valiant, and Vijay Vazirani for their comments.

7. Bibliography

[1] L. ADLEMAN *Two Theorems on Random Polynomial Time*, Proc. 19th IEEE Symposium on Foundations of Computer Science, Ann Arbor, MI, October 1978, pp. 75 – 83.

[2] R. ANDERSON, *A Parallel Algorithm for the Maximal Path Problem,* Proc. 17th ACM Symposium on Theory of Computing, Providence, RI, May 1985, pp. 33 – 37.

[3] W. ALEXI, B. CHOR, O. GOLDREICH, AND C. SCHNORR, *RSA/Rabin Bits Are 1/2 + 1/poly(log N) Secure,* Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, October 1984, pp. 449 – 457.

[4] P. BEAME, S. COOK, AND H. HOOVER, *Small Depth Circuits for Integer Products, Powers, and Division,* Proc. 25th IEEE Symposium on Foundations of Computer Science, Singer Island, FL, October 1984, pp. 1 – 6.

[5] L. BLUM, M. BLUM, AND M. SHUB, *A Simple Secure Pseudo-Random Number Generator,* Proc. of CRYPTO-82, Santa Barbra, CA, September 1982, pp. 112 – 117.

[6] M. BLUM AND S. MICALI, *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits,* SIAM J. Comp., 13 (1984), pp. 850 – 864.

[7] H. CHERNOFF, *A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations,* Ann. Math. Statist., 23 (1952), pp. 493 – 507.

[8] S. COOK, *Towards a Complexity Theory of Synchronous Parallel Computation,* (Presented at) Inter. Symp. Logic. Alg. (1980).

[9] O. GOLDREICH, S. GOLDWASSER, AND S. MICALI, *How to Construct Random Functions,* Proc. 25th Symposium IEEE Symposium Foundations of Computer Science, Singer Island, FL, October 1984, pp. 464 – 479.

[10] S. GOLDWASSER, S. MICALI, AND P. TONG, *Why and How to Establish a Private Code on a Public Network,* Proc. 23rd IEEE Symposium Foundations of Computer Science, Chicago, IL, October 1982, pp. 134 – 144.

[11] R. KANNAN, G. MILLER, AND L. RUDOLF *Sublinear Parallel Algorithms for the Greatest Common Divisor of Two Integers,* Proc. 25th IEEE Symposium Foundations of Computer Science, Singer Island, FL, October 1984, pp. 7 – 11.

[12] R. KARP AND A. WIGDERSON, *A Fast Parallel Algorithm for the Maximal Independent Set Problem,* Proc. 16th ACM Symposium on Theory of Computation, Washington, DC, May 1984, pp. 266 – 272.

[13] R. KARP, E. UPFAL, AND A. WIGDERSON, *Constructing a Perfect Graph Matching in RNC,* Proc. 17th ACM Symposium on Theory of Computing, Providence, RI, May 1985, pp. 22 – 32.

[14] D. KNUTH, *The Art of Computer Programming, vol. 2: Seminumerical Algorithms, 2nd ed.,* Addison-Wesley, Reading, MA, 1981.

[15] J. REIF, *Logarithmic Depth Circuits for Algebraic Functions,* Proc. 24th Symposium IEEE Foundations of Computer Science, Tuscon, AZ October 1983, pp. 138 – 145. Revised in Technical Report TR-84-18, Center for Research in Computing Technology, Harvard University. To appear in SIAM J. Comp.

[16] J. REIF AND J. TYGAR, *The Complexity of Chaotic Iterative Maps.* To appear.

[17] A. SHAMIR, *On the Generation of Cryptographically Strong Pseudo-Random Sequences,* ACM Trans. on Comp. Sys., 1, (1983), pp. 38-44.

[18] A. SHONHAGE AND V. STRASSEN, *Schnelle Multiplication grosser Zahlen,* Computing, 7 (1974), pp. 281 – 292.

[19] L. VALIANT, S. SYKUM, S. BERKOWITZ, AND C. RACKOFF, *Fast Parallel Computation of Polynomials Using Few Processors,* SIAM J. Comp., 12 (1983), pp. 641 – 644.

[20] U. VAZIRANI AND V. VAZIRANI, *Trapdoor Pseudo-Random Number Generators with Applications to Protocol Design,* Proc. 24th IEEE Symposium Foundations of Computer Science, Tuscon, AZ, October 1983, pp. 23 – 30.

[21] VON ZUR GATHEN, Private communication.

[22] A. YAO, *Theory and Applications of Trapdoor Functions,* Proc. 23rd IEEE Symposium Foundations of Computer Science, Chicago, IL, October 1982, pp. 80 – 91.