December 1983

Report No. STAN-CS-83-991

# Parallel Algorithms for Arithmetics, Irreducibility and Factoring of *GFq*-Polynomials

by

Moshe Morgensteren and Eli Shamir

**Department of Computer Science**

Stanford University
Stanford, CA 94305

# PARALLEL ALGORITHMS FOR ARITHMETICS, IRREDUCIBILITY AND FACTORING OF G&-POLYNOMIALS

Moshe Morgensteren and Eli Shamir
Institute of Mathematics and Computer Science
The Hebrew University, Jerusalem

## Abstract

A new algorithm for testing irreducibility of polynomials over finite fields without gcd computations makes it possible to devise efficient parallel algorithms for polynomial factorization. We also study the probability that a random polynomial over a finite field has no factors of small degree.

## 1. Introduction.

Efficient sequential algorithms were recently developed [3,4,5] for basic polynomial arithmetic on $GF_q$, the Galois field with $q$ elements; for testing, searching irreducible $f \in GF_q[x]$; for root finding and factoring $f$.

Here we discuss parallel algorithms for these tasks where $P$ processors can work in parallel on one problem instance.- We use the PRAM model of parallel computation, for which each processor has a direct access in one step to the common memory or equivalently to any other processor. This is very convenient for writing and analyzing parallel programs.

Performance-wise, the best one can hope is to speed-up the sequential processing time $TS(Q)$ so that $PT(Q) = O[TS(Q)/P]$, where the parallel time $TP(Q)$ for the problem $Q$ also includes the communication steps between the processors, which are interleaved with the computation steps of each processor. The total number of communications CM(Q) is an important measure by itself. Having CM(Q) = $O[TS(Q)]$ is a reasonable requirement for a good parallelization; otherwise, even if we assume $P$ communications are processed in one time unit, we get a communication time lower-bound which exceeds $O[TS(Q)/P]$.

The natural input-length for a problem instance $f$ is $\deg(f) = n$. In fact $n + 1$ is the number of coefficients which specify $f$.

We usually assume:

$$\text{The number of processors is } n/w(n) \text{ with } 1/w(n) = O(1) \text{ as } n \to \infty. \tag{1.1}$$

This restriction is amply justified on practical and theoretical grounds. In our context, the restriction (1.1) effects primarily the computation of

$$(f,g) = \text{gcd of } f \text{ and } g \tag{1.2}$$

a procedure which figures prominently in irreducibility and factoring problems. As far as we know, the procedure for $(f, g)$ resists a significant speed-up with $P = n/w(n)$ processors. However with $n^k$ processors ($k$ = 13 or so) a speed-up is achievable [9] (but is it practical?).

The preparation of this paper was supported in part by National Science Foundation grant MCS-83-00984.

The success in getting a maximal speed of n/w(n) for irreducibility and factoring-actually even an improvement of the sequential **algorithms**—hinges on the scheduling of the gcd-procedure calls. We either obviate them altogether, or 'reduce them and schedule them so that processors do not remain idle.

The plan of the article is as follows: Section 2 treats the basic polynomial arithmetics and **gcd,** Section 3 treats irreducibility for one $f$, Section 4 describes a search for irreducible polynomials, Section 5 treats factorization of $f$. Finally in Section 6 we derive in an elementary method estimates on the probability distribution of the lowest-factor degree of a random $GF_q$ polynomial.

## 2. Basic Polynomial Arithmetics.

**Notation:** $L(N) = \log(N) \cdot \log\log(N)$.

Ingenious sequential algorithms were developed for polynomial arithmetics in $F(z)$, bringing the sequential time down to N . L(n) or $\deg(f) \leq N$.

The parallel implementations follow closely the sequential algorithms. One has to allocate efficiently $P =$ N/w(N) processors and an almost maximal speed-up is obtained, the arguments are somewhat simpler for w(N) = i , $(P(N) = N)$ . The proofs are quite routine, somewhat tedious. We do not reproduce the proofs here; only the results are stated-they are not surprising.

**Multiplication:** (done with Fourier Transform [7]).

$$TP = w(N)L(N), \qquad CM = OIN . L(N)]. \tag{2.1}$$

**Division with remainder:** $f = q . g + r$, $\deg(r) < \deg(g)$.

The sequential algorithm is based on Newton's approximation up to degree N of $g^{-1}$.

$$TP \approx w(N) . L(N) + L(N/w(N)) \log(N/w(N)), \tag{2.2}$$

$$CM = O(TS) = O(N . L(N)). \tag{2.3}$$

**Residues:** $f_i$ (mod $g$), $1 \leq i \leq k$.

One has to compute the approximation to $g^{-1}$ once, then only **muliplications** are needed.

$$TP = k . w(N) \cdot L(N) + L(N/w(N)) \log(N/w(N)), \tag{2.4}$$

$$CM = O(TS) = O(k . N . L(N)). \tag{2.5}$$

**Remark:** The relative contribution of the second term in (2.2), (2.4) is largest when w(N) = 1, when it gives L(N) *log(N). The speed-up is short of its maximal value by a factor of log(N) . For $k \geq \log(N)$ in (2.4), this is immaterial and we do get a maximal speed-up.

**Greatest Common Divisor — (f, g):**

The sequential algorithm in [1] has

$$TS = O(N . L(N) *\log(N)). \tag{2.6}$$

It is based on Euclid's algorithm which is rigidly sequential, i.e., we cannot parallelize the sequence of multiplications/divisions, only within a single such operation-which gives a meager speed-up. It will be highly interesting to obtain a formal proof that one cannot get a significant speed-up for the gcd with N/w(N) processors.

## 3. Irreducibility.

**Fact 3.1:** $x^{q^k} - x$ is the product of all irreducible monic polynomials of degree $s$ in $GF_q[x]$ with $s \mid k$. In fact their distinct roots in the extension field exhaust all elements of $GF_{q^k}$, which is a subfield of $GF_{q^n}$ iff $k \mid n$.

Thus for a monic $f$ of degree $n$ in $GF_q[x]$, $f$ is irreducible, $\Leftrightarrow f \mid (x^{q^n} - x)$ and

$$(f, x^{q^k} - x) = 1, \qquad k < n. \tag{3.1}$$

Rabin's algorithm [ 5] is a straightforward translation of this fact. A better sequential algorithm, which obviate the use of gcd, is

**Algorithm 3.1 — Testing irreducibility of $f$, $\deg(f) = n$.**

```
     begin
1.   $u := x, v := 1$;
2.   for $j := 1$ to $n - 1$ do [$u := u^q \pmod f$, $v := v \cdot (u - x) \pmod f$];
3.   $u := (u^q - x) \pmod f$;
     (* now $u = (x^{q^n} - x) \pmod f$, $v = \prod_{i=1}(x^{q^i} - x) \pmod f$ *)
4.   $f$ is irreducible iff $u = 0$ and $v \neq 0$.
     end
```

**Correctness:** If $f$ is irreducible then $f \mid (x^{q^n} - x)$ so $u = 0$ and $f$ does not divide the product. which gives v. Conversely, if u = 0 then $f \mid (x^{q^n} - x)$, so all factors of $f$ have multiplicity **1.** Every real factor (of degree $k < n$) must divide a factor $(x^{q^k} - x)$ of v. Thus if $f$ is reducible then v = 0, but $v \neq 0$ so $f$ is irreducible.

The sequential time for Algorithm 3.1 is clearly (lines 2 and 3):

$$TS = n \cdot \log q \cdot n \cdot L(n) = n^2 L(n) \log q. \tag{3.2}$$

The parallel time and the number of communications with $n/w(n)$ processors, are

$$TP = w(n) \cdot n L(n) \cdot \log q, \qquad CM = O(TS). \tag{3.3}$$

Indeed, lines 2, 3 are parallelizable with the maximal speed-up factor.

Ben-Or [3] gave an algorithm which tests for factors of degree $1, 2, \ldots, [n/2]$ until a factor is found; if none is found, then $f$ is i&educible. The worst case of [3] is worse than **(3.2),** but $EXP(TS)$, the expectation taken over $GF_q[x]$, is much better, since $EXP(L_n(f))$ = logn, where

$$L,(f) = \text{degree of the lowest factor of } f,\ \deg(f) = n,$$

cf. [3] and Section 6 below,

We present now an algorithm which is better or -equal than both in all respects: worst-case,. average and tail estimate for $TS$. It also leads to a maximal speed-up in the parallel search for an irreducible $f$ (Section. 4).

**Algorithm 3.2 — Testing irreducibility of $f$, $\deg(f)$ = n.**

```
begin
1.  u := $x$; v := 1;
2.  for $i$ := 1 to $d := C \cdot \log n$ do
3.  begin u := $u^q$(mod f); v := v . (u $-$ $x$)(mod f); end;
4.  if $(f, v) \neq 1$ then return ‘$f$ is reducible’;
    repeat
5.  for $i$ := 1 to $d$ do
6.  begin u := $u^q$(mod $f$); $v$ := $v$ . ($u$ $-$ $x$)(mod $f$); end
7.  if $(f, v) \neq 1$ then return ‘$f$ is reducible’;
8.  . d := 2 . $d$;
    until $d \geq n - 1$;
9.  return ‘$f$ is irreducible’;
end
```

**Correctness:** In phase 0, lines l-4, we test for factors in the range $1 \leq s \leq d_0 = C$ . log $n$. *In* phase m, which is the mth execution of lines 5-8, we test for factors in the range

$$2^{m-1} \cdot d_0 < s \leq 2^m \cdot d_0 . \qquad (3.4)$$

The beauty of this scheme is that the number of steps in the last executed phase, say phase $m$, which is

$$0[2^{m-1} \bullet \quad \text{do*logq=neL(n)]}$$

dominates the number of steps in all previous phases. As a consequence we get

$$\text{Worst-case TS} = O(n^2 \,.\, L(n) \,.\, \log \;; \qquad (3.5)$$

$$EXP(TS) = 0\,(n \,.\, \log n \cdot L(n) \,.\, \log q)\,; \qquad (3.6)$$

$$\text{Prob } \{TS \geq n^\delta \cdot n \cdot L(n) \cdot \log q\} \leq n^{-\delta} \qquad (3.7)$$

Indeed for $n$ sufficiently large the event in the braces is contained in {L,(f) $\geq n^\delta\}$ which has probability $\leq n^{-\delta}$ by Section 6.

Parallelization for a single (even random) $f$ will not do better than parallelization of Algorithm 3.1 since at least one call of gcd remains uncracked. But for searching an irreducible $f$, which involves testing many $f$ s, the story is different.

## 4. Parallel Search for an Irreducible $f$.

Finding irreducible $f$ of degree $n$ over $GF_q$ is needed for realizing arithmetic in $GF_{q^n}$ *as* polynomial arithmetic (mod $f$) in $GF_q$. Other uses for data protection is explained in [6].

The basic idea is to test $n$ (for tail estimates, $N$ = n . *A(n))* random polynomials, chosen independently. The probability $p$ of a monic $f$ of degree $n$ to be irreducible is about l/n. The expected number of tests is $n$. For tail estimates of the success probability

$$. \text{ Prob \{irreducible } f \text{ is not found among } N\}$$
$$= \text{Prob \{number of successes deviates from } N \,.\, p \text{ by } N \,.\, p\}$$
$$\leq e^{-\frac{1}{2} N \cdot p} = e^{-\frac{1}{2} \cdot \lambda(n)} \qquad (4.1)$$

by Chernoff bound [2]. If $\lambda(n) = C \cdot \log n$, the tail estimate is $n^{-C}$. One can continue to derive tail estimates, but in the following discussions we limit ourselves to expectations.

After choosing a set $M$ of n random polynomials (we can use the $k$th processor to choose the $k$th coefficient), the testing scheme is to run one phase (say phase $k$) of Algorithm 3.2 to conclusion on all $f \in$ M, before the next phase (k + 1) starts. The allocation of processors to polynomials is always even-handed, with difference $\leq 1$ between any pair. When a phase concludes, some polynomials drop out, the processors are then eventually reallocated to the remaining polynomials.

**Claim 4.1:** (See Section 6.) Upon running phase $k \geq 1$ of Algorithm 3.2 on $f$, the probability of $f$ to remain (and so the expected number of remaining $f$ s) is divided by $2 \pm O(n^{-c+1})$.

Thus for running the next phase, the expected number of processors per polynomial is doubled, but also the number of steps in the 'for' loop in lines 5-6 is doubled. The operations in line 6 are maximal parallelizable (line 7 we can speed-up only by a factor of log $n$). Thus the expected parallel time is about the same for the various phases and in the interesting case $p(n) \doteq n$ it is log $n$ . $n$ . L(n) . log $q$ for most phases. Thus

**Theorem 4.2:** *For the parallel search of an irreducible polynomial of degree n with n/w(n) processors.*

$$EXP(TP) = O[w(n) \cdot n \log^2 n \cdot L(n) \log q], \tag{4.2}$$
$$CM = O(TS) = O[n^2 \cdot \log^2 n \cdot L(n) \cdot \log q]. \tag{4.3}$$

*In particular $EXP(TP)$ is log-linear for n processors.*

## 5. Parallel Factorization of Polynomials Over $GF_q$.

The algorithms in the literature [4,5,3] have two phases.

**Phase A:** Returns

$$g_1, \ldots, g_t \quad \text{and} \quad d_1, \ldots, d_t \tag{5.1}$$

where $g_i$ is a product of all distinct factors of $f$ of some fixed degree $d_i$, and $d_1 < d_2 < d_t$, so clearly $\binom{t}{2} = 1 + 2 + \ldots + t \leq \deg(f)$ = N.

**Phase B:** (the harder one) factors a typical g of (5.1).

**Phase A:** Phase A is done in [4,5,3] without considerations of parallelization, with a long and badly scheduled sequence of gcd's.

We give another deterministic algorithm for phase **A with** a careful scheduling of gcd's, which gives a maximal speed-up.

**Theorem 5.1:** *For P = N/w(N) processors, a parallel algorithm can be found which returns (5.1) with*

$$TP = O\big(w(N) \cdot N \cdot L(N)[\log q + \log N]\big) \tag{5.3}$$
$$CM = O(TS) = O\big(N^2 \cdot L(N) \cdot [\log q + \log N]\big) \tag{5.4}$$

**Proof and Algorithm:** We accomplish the task in three parts.

**Algorithm 5.1 — Part I —** First decomposition.

```
begin
 1 .   $r := x$;
 2.    for $i := 1$ to $N$ do [$r := r^q (\mathrm{mod}\ f)$; $h_i := (r - x)$];
 3.    for $j := 0$ step $p$ to $N$ do
           computein one (gcd) time unit
 4.    $a_{j \cdot p+1} := (f, h_{j \cdot p+1}), \ldots, a_{j \cdot p+p} := (f, h_{j \cdot p+p})$;
end.
```

The parallel cost of line 2 is indeed (5.3). The cost of line 4 is $w(N) \cdot TS$ (gcd) . So the total time is indeed given in (5.3).

Now we have $a_i = (f, x^{q^i} - x)$, which is a product of all the distinct factors of $f$ of degree dividing $i$. Clearly by climbing from below and successive divisions the gs of (5.1) **can be obtained.** But the sequence of divisions is too long, so a better idea is to *first* get rid of superfluous $a_i$ -those for which $f$ does not have an honest factor of degree $i$. Such $a_i$ are characterized by the **condition** $a_i \mid \prod_{j<i} a_j$.

**Algorithm 5.1 — Part II —** (gets rid of **supefluous** $a_i$).

```
begin
 1.    $r := 1, j := 0$;
 2.    for $i := 1$ to $N$ do
 3.    if $a_i \nmid r$ then [$j := j + 1$; $b_j := a_i$; $r := r \cdot a_i\ (\mathrm{mod} f)$];
end
```

This part returns $b_1, \ldots, b_t$ where, $t$ is clearly the same as in (5.1).

The cost is dominated by Part I. Now finally we can et the $g_i$s in Part III.

**Algorithm 5.1 — Part III —** (finding the $g_i$s).

```
begin
 1.    $g_1 := b_1$;
 2.    for j := 2 to $t$ do
 3.    begin for $i := 1$ to $j - 1$ do [if $g_i \mid b_j$ then $b_j := b_j / g_i$]; $g_j := b_j$; end
end.
```

Correctness is clear. We do $\binom{t}{2}$ divisions and $\binom{t}{2} \leq N$ by (5.2). Using the results **from** Section 2 for $TP$ (division), we get **the** estimate (5.3).

As for CM, all the procedures we use from Section 2 which are used have CM = $O(TS)$, the total of these is included in (5.4), while the intermediate communications are dominated by (5.4).

**Phase B:** (Let g be a typical $g_i$ of (5.1) with degree (g) $= l$, g has $k$ irreducible factors of degree $d$, $l = k \cdot d$).

For Phase B we follow Ben-Or [3]. Be works with the assumption that $q$ is much greater than $n$, but remarks that it can be dropped. We shall work without this **assumption.**

Ben-Or's idea is to find a polynomial which is called a **'separating polynomial'** for g, **then to** decompose g with a recursive procedure 'SEPARATE', and continue in the same way with the two factors of g.

He shows that if we choose randomly a polynomial y $\in F_q$ [z]/(g) then

$$\mathrm{Tr}(y) = y + y^q + \cdots + y^{q^{d-1}} (\mathrm{mod}\ g) \tag{5.5}$$

is a separating polynomial for g with probabiity $1 - 1/q^{k-1}$. The cost of computing $\mathrm{Tr}(y)$ is [illegible] . $l$ . $L(l)$ . log $q)$ and we need at most 2 .$(l/d)$ = 2 .$k$ such polynomials during the execution of SEPARATE.

So the expected total cost of computing separating polynomials for g is $\leq O(l^2$ . $L(l)$ log $q)$. Ben-Or shows also that the expected cost of SEPARATE is 0 (log $k$ .$l$ .$L(l)$ .[log $q$ + log $l$]).

Going back to the original $f$ (in the notations of Phase A) the total cost of factoring all the $g_i$s of $f$ is

$$. . \sum_{i=1}^{t} l_i^2 \,.\, L(l_i)[\log q + \log l_i] \leq C \cdot N \cdot L(N)[\log q + \log N] \cdot \sum_{i=1}^{t} l_i$$

$$\leq CN^2 L(N)[\log q + \log N]\,. \tag{5.6}$$

But this is also the sequential cost of Phase $\mathbf{A}$, so this is the **total** cost for factoring $f$.

**Theorem 5.2:** ***For $P = N/w(N)$ processors, the following parallel algorithm factors a*** polyno-***mial $f$,*** $\deg(f) \leq n$***, within***

$$TP = O[w(N)\,.\,N \cdot L(n)\ .\,(\log q + \log N)] \tag{5.7}$$

$$\mathrm{CM}\ .\ O[N^2\ .\ [illegible]\ .\ (\log q + \log N)] = O(TS). \tag{5.8}$$

**Proof.** First we run Phase $\mathbf{A}$, which returns $g_1, \ldots, g_t$ and $d_1, \ldots, d_t$ of (5.1). Then we run Phase B, which is

```
begin
    $S := \{g_1, \ldots, g_t\}$
    repeat
    1. Allocate processors to subsets of S;
    2. Compute a separating polynomial for each element of S;
    3. s := result of applying SEPARATE to each element of S;
    until all elements of S are irreducible
end
```

For the allocation step, we group all the polynomials into subsets with sum of degrees $\leq P/2$. Then we work sequentially on each subset with all $P$ processors. When the degrees of the polynomials in the subsets is small, or if $P$ is small, one processor will do subroutines 2 and 3 on each polynomial. If the degree of a polynomial is high, and $P$ is large $(P \geq \sqrt{N}$ . log $N)$, we have available log N processors for each polynomial to speed-up the arithmetic operations in 2 and 3 by log N. This allocation'scheme suffices to give the claimed performance (5.7), (5.8), as one can easily verify.

## 6. Probability Distribution of the Lowest Factor Degree.

For the analysis of Algorithm 3.2 in the sequential form or in the parallel form adapted for search of irreducible $f$ in Section 4, we need asymptotic estimates for some probabilities.

Let $f$ be a monic polynomial of degree $n$ in $GF_q[z]$ .

$$L(f) = \min\{\deg(h) : h \text{ divides } f\} \tag{6.1}$$

$$A(n, r) = \{L(f) > r\}. \tag{6.2}$$

The probability estimates for these events will hold with high precision for $r \geq c \cdot \log n$. They are based on the following easily verified lemma.

**Lemma 6.1** [5,3]:

$$\text{Prob } \{\text{a monic } f \text{ of degree m is irreducible}\} = \frac{1}{m} + O(q^{-(m/2)}) \tag{6.3}$$

Note that if $m \geq 2c \cdot \log n$, the error is $O(n^{-c})$.

Now $f = g \cdot h$, g is irreducible with $\deg(g) = t(g) = L(f)$. If $A(n, r-1)$ holds then either $L(g) = r$ and $L(h) \geq r$ or $L(f) \geq r + 1$ so

$$A(n, r-1) = \{L(g) = r\} \cdot A(n-r, r-1) + A(n, r) \tag{6.4}$$

(+ denotes disjoint union). If $r \geq c \cdot \log n$ (so we can use (6.3) and $r \ll n$ so that $n - r$ and $n$ axe effectively the same, we get when we pass to probabilities

$$PA(n, r-1) = \frac{1}{r} PA(n, r-1) + PA(n, r), \tag{6.5}$$

$$\frac{r-1}{r} PA(n, r-1) = PA(n, r) \qquad \text{so} \qquad r \cdot PA(n, r) \doteq \text{constant}, \tag{6.6}$$

which holds with a high precision, $O(n^{-c})$ for the specified range of $r$; in particular we get from the recursion in (6.6) with high-precision

$$PA(n, 2r) = \frac{1}{2} PA(n, r). \tag{6. 7}$$

To study relatively large values of $r$ even close to $n$, we set

$$B(n, k) = \{L(f) \geq n/k\} \tag{6.8}$$

again

$$B(n, k) = \{n/k \leq \deg(g) < n/(k-\delta))\} \cdot B(n - (n/k), k(1 - (1/k))) + B(n, k - 6). \tag{6.9}$$

$(B(n - (n/k), k(1 - 1/k))$ is the event that a polynomial of degree $n - (n/k)$ has the lowest factor degree $\geq n/k)$ . Now we pass to probabilities for $k$ not too small:

$$PB(n, k) \doteq \frac{kk - \delta}{m} \cdot n \cdot \left(\frac{1}{k-\delta} - \frac{1}{k}\right) PB\left(n - \frac{n}{k}, k\left(1 - \frac{1}{k}\right)\right) + PB(n, k - \delta), \tag{6.10}$$

$$PB(n, k) = \frac{\delta}{k} PB\left(n \cdot \left(1 - \frac{1}{k}\right), k\left(1 - \frac{1}{k}\right)\right) + PB(n, k - \delta). \tag{6.11}$$

If we take $P(n, 1) = 1/n =$ Prob $\{f$ of degree $n$ is irreducible$\}$ then the recurrence relation (6.11) happens to be solved precisely by

$$PB(n, k) = \frac{k}{n}. \tag{6.12}$$

One is led to this solution from the assumption that $PB(n, k)$ is a function of $k/n$ only. Then one gets one-step recurrence relations analogous to (6.6)

$$PB(n, k) = \left(1 - \frac{\delta}{k}\right) PB(n, k - \delta) \qquad \text{and} \qquad \frac{PB(n,k)}{k} = \text{constant}, \tag{6.6$'$}$$

which imply also

$$PB(n, 2k) = 2\, PB(n, k). \tag{6.7$'$}$$

These relations hold for $k$ not too close to $n$. For the other range *we* used $PA(n, r)$ above. There is a wide overlap of the two ranges. Clearly $EXP(L(f)) = O(\log n)$. More effort is required in order to show that the value is asymptotically log $n$.

Another approach to the asymptotic extimates of the distribution of $A(n, r)$ and other events related to the factoring pattern of a random $f$ is to use the fact [3] that probability-wise it is 'quite' close to the cycle decomposition pattern of a random-permutation (of $n$ objects), and use known results about permuations [8]. There are several drawbacks to this approach: (1) 'quite close' is not close enough for our purpose, (ii) the proofs in [8] are quite involved especially if one wants to use them for $L(f) \geq n/k$, where the limit also grows to infinity with $n$. *In* fact one can easily see how to employ our elementary approach to derive sharp estimates on the distribution of the smallest cycle length in a random permutation of $n$ objects. In both cases (polynomials or permutations) we can also get the 'r-smallest' (factor or cycle) distribution.

## References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Algorithms, Addison- Wesley, Reading, Mass., 1974.

[2] H. Chemoff, "A Measure of Asymptotic Efficiency for Tests of Hypothesis," Ann. Math. Stat. 23 **(1952)**, 493-507.

[3] M. Ben-Or, "Probabilistic Algorithms in Finite Fields," ***Proceedings 21st Annual IEEE FOCS*** **(1981)**, 334-398.

[4] E. R. Berlekamp, "Factoring Polynomials over Large Finite Fields," Math. Comp. **24 (1970), 713–735.**

[5] . M. 0. Rabin, "Probabilistic Algorithms 'in Finite Fields," SLAM ***J. on Comp.*** 9 **(1980), 273–280.**

[6] M. 0. Rabin, "Fingerprints by Random Polynomials," Preprint 1982.

[7] A . **Schönhage,** "Schnelle Multiplikation von Polynomen," Acts ***Informatica*** **7 (1977),** 395-398.

[8] L. A. Shepp, S. P. Lloyd, "Ordered Cycle Length in a Random Permutation," ***Trans.*** Amer. ***Math. Soc.*** **121 (1966), 340–357.**

[9] **J.** von **zur** Gathen, "Parallel Algorithms for Algebraic Problems," ***Proceedings 15th Annual ACM STOC*** **(1983),** 17-23.