Factoring High-Degree Polynomials by the Black Box Berlekamp Algorithm*

Erich Kaltofen and Austin Lobo

Department of Computer Science, Rensselaer Polytechnic Institute
Troy, New York 12180-3590; Inter-Net: {kaltofen,loboa}@cs.rpi.edu

Extended Abstract

1. Introduction

Modern techniques for solving structured linear systems over finite fields, which use the coefficient matrix as a black box and require an efficient algorithm for multiplying this matrix by a vector, are applicable to the classical algorithm for factoring a univariate polynomial over a finite field by Berlekamp (1967 and 1970). We report on a computer implementation of this idea that is based on the parallel block Wiedemann linear system solver (Coppersmith 1994 and Kaltofen 1993 and 1995). The program uses randomization and we also study the expected run time behavior of our method.

The asymptotically fastest known algorithm for factoring a polynomial over a finite field is by von zur Gathen and Shoup (1992). Shoup (1993) has subsequently implemented the equal degree part of that algorithm making use of FFT-based polynomial arithmetic. We will show that a sequential version of the black box Berlekamp algorithm is strongly related to their method and allows for the same asymptotic speed-ups, at least within a logarithmic factor.

It is also possible to realize the Niederreiter approach (Niederreiter and Göttfert 1994) by black box linear algebra (Gao and von zur Gathen 1994). That approach is similar to Berlekamp's (Fleischmann 1993) and the tradeoffs between both methods when using a black box linear system solver are quite subtle. We have not implemented the black box Niederreiter algorithm and therefore have no comparative data. It is now known that a method by Cantor and Zassenhaus (1981, §3) is competitive only if implemented with von zur Gathen's and Shoup's improvements (see Shoup 1993).

Our program is geared towards factoring high degree polynomials over moderately sized fields. Polynomials of degree 250 or less are factored most quickly by the standard Berlekamp algorithm. So far, we have factored polynomials of degree 10 001 modulo 127, in about 102.5 hours. Clearly, for such large degrees, FFT-based polynomial arithmetic becomes much more efficient than standard or Karatsuba-based polynomial arithmetic. We use a polynomial arithmetic package designed by Shoup (1993) in C++. His underlying integer arithmetic, by A. K. Lenstra, is geared towards long integers and thus slows the runs on our small residues somewhat.

We have converted our parallel block Wiedemann linear system solver (Díaz et al. 1993) into C++. The program now works with a generic black box matrix by accepting a function call for the matrix-times-vector product. We note that our implementation of Coppersmith's method has been highly successful in solving linear systems with sparsely populated coefficient matrices. A bottleneck step there has been the sequential Berlekamp/Massey algorithm for finding a recurrence polynomial. However, the black box matrix-times-vector products arising in the polynomial factoring algorithm are far more costly. Without blocking and parallelization our test runs would have been impossible. The bottleneck in the polynomial factoring application is the needed matrix-times-vector product. Higher speed would still be achievable by using more computers in parallel, which we do not have available to us.

Unlike Shoup's (1993) implementation, our algorithm computes a complete factorization of the input polynomial. This has necessitated some improvements in the overall approach. Instead of finding a single split and proceeding recursively, we "mine" each vector found in the so-called Butler/Berlekamp subalgebra (we have

*This material is based on work supported in part by the National Science Foundation under Grant No. CCR-90-06077.

In Proc. ISSAC '94, J. von zur Gathen and M. Giesbrecht (eds.), ACM Press, pp. 90–98 (1994).

about as many as the blocking factor of the linear solver) for all splits derivable from that vector. Lastly, we use factor refinement (Bach et al. 1993) to compute the full factorization. The problem remains to determine when all factors are found. Currently, our version of the block Wiedemann algorithm does not produce the rank of coefficient matrix as a by-product, which would yield the number of irreducible factors. However, the rank is available when the Berlekamp/Massey substep in the block Wiedemann algorithm is replaced by a Toeplitz-like linear solver (Kaltofen 1995, Theorem 7), at a negligible loss in efficiency. We plan to make this change in the near future.

When analyzing a sequential version of the black box Berlekamp algorithm, similarities to the distinct-degree factorization algorithm (Knuth 1981, §4.6.2) and the equal-degree factorization algorithm (see, e.g., von zur Gathen and Shoup 1992, Algorithm 3.6) become apparent. The reason for these connections is that an explicit formula for the minimum polynomial of the arising coefficient matrix can be derived. Certain substeps of the black box linear system solvers can then be related to both algorithms. We believe that these connections are significant because they explain how the Berlekamp algorithm in some sense generalizes both the distinct-degree and the equal-degree factorization algorithms, which is superior in performance to both methods so far. Furthermore, the degree of the minimum polynomial can be much smaller than the dimension of the corresponding matrix depending on the degree of the factors. This makes an adaptive version of the black box algorithm possible which performs many less matrix-times-vector products e.g., in the case where many factors share the same degree.

2. The Berlekamp Algorithm

The Berlekamp (1967, 1970) algorithm for factoring a monic squarefree polynomial $f \in \mathbb{F}_q[x]$ of degree $\deg(f) = n$ into its monic irreducible factors $f_1, \ldots, f_r$ is based on considering a subalgebra of $\mathbb{F}_q[x]/(f(x))$. In the following theorem we use a polynomial $v \in \mathbb{F}_q[x]$ with $\deg(v) < n$ as the canonical representative of an element in the quotient algebra $\mathbb{F}_q[x]/(f(x))$. Vectors are distinguished from polynomials by the vector mathematical accent $\vec{\ }$, which we also use as the coefficient vector constructor: if $u(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$ then $\vec{u} = [\, c_0, \quad c_1, \quad \ldots, \quad c_{n-1}\,]$.

Theorem 1. *For $v \in \mathbb{F}_p[x]$ with $\deg(v) < n$ the following conditions are equivalent:*

$$\forall i\colon v(x) \bmod f_i(x) \;\in\; \mathbb{F}_q, \tag{1}$$

$$\Updownarrow$$

$$v(x)^q = v(x^q) \;\equiv\; v(x) \pmod{f(x)}, \tag{2}$$

$$\Updownarrow$$

$$\vec{v} \cdot Q = \vec{v} \textit{ where } Q = \begin{bmatrix} \vdots \\ \overrightarrow{x^{iq} \bmod f(x)} \\ \vdots \end{bmatrix}_{i=0,\ldots,n-1} \in \mathbb{F}_q^{n \times n}. \tag{3}$$

If the characteristic of $\mathbb{F}_q$ is not equal to 2, then (1), (2), and (3) are also equivalent to

$$\Updownarrow$$

$$\forall i\colon v(x)^{(q-1)/2} \bmod f_i(x) \in \{-1, 0, 1\}. \tag{4}$$

The classical Berlekamp algorithm (1967) finds a non-constant polynomial v with the above properties by solving the linear system (3) and then computes the polynomial greatest common divisor $\mathrm{GCD}(v(x) - t, f)$ for all $t \in \mathbb{F}_q[x]$. If the vectors $\vec{v}$ participating in this process span the left null space of $Q - I$, the polynomial f is completely factored. For large fields of characteristic > 2, a modification by Cantor/Zassenhaus (1981, §4) and Camion (1981) based on (4) computes $\mathrm{GCD}(v^{(q-1)/2} \bmod f - 1, f)$ in order to split f. Large fields of characteristic 2 permit special tricks, but we shall not consider this situation here.

Historically, the above theorem dates back to before Berlekamp. The matrix Q was used for an irreducibility test by Petr in 1937 (see Schwarz 1956). Butler (1954) established parts (1)–(3) of Theorem 1, which Berlekamp used so cleverly for his now famous method. Our modification also uses a property of Petr's matrix.

Theorem 2. *Let $f \in \mathbb{F}_q[x]$ be monic and squarefree with monic irreducible factors $f_1, \ldots, f_r$ of degree $m_i = \deg(f_i)$. Let $\mu^Q(\lambda) \in \mathbb{F}_q[\lambda]$ be the minimum polynomial of the matrix Q as defined in (3). Then $\mu^Q(\lambda) = \mathrm{LCM}_{1 \le i \le r}(\lambda^{m_i} - 1)$, where LCM denotes the polynomial least common multiple.*

Proof. For any $w \in \mathbb{F}_q[x]$ with $\deg(w) < n$ we have

$$u = w^q \bmod f \iff \vec{u} = \vec{w} \cdot Q. \tag{5}$$

Now consider the algebra isomorphism

$$\begin{aligned} \Phi\colon \mathbb{F}_q[x]/(f(x)) &\longrightarrow \mathbb{F}_{q^{m_1}} \oplus \cdots \oplus \mathbb{F}_{q^{m_r}} \\ w &\longmapsto [\, \widehat{w}_1, \quad \ldots, \quad \widehat{w}_r\,] \end{aligned}$$

derived from the Chinese remainder theorem and the field isomorphisms

$$\psi_i\colon \mathbb{F}_q[x]/(f_i(x)) \longrightarrow \mathbb{F}_{q^{m_i}},$$

where element multiplication in the direct sum of fields $\mathbb{F}_{q^{m_1}} \oplus \cdots \oplus \mathbb{F}_{q^{m_r}}$ is defined componentwise. We shall suppose, as we may, that $\mathbb{F}_q \subset \mathbb{F}_{q^{m_i}}$ for all i and that each ψ_i restricted to $\mathbb{F}_q$ is the identity function. The operator

$$\begin{aligned}\mathcal{L}\colon \mathbb{F}_q[x]/(f(x)) &\longrightarrow \mathbb{F}_q[x]/(f(x))\\ w &\longmapsto w^q \bmod f\end{aligned}$$

is by (5) linear and has by virtue of the isomorphism Φ the corresponding operator $\mathcal{L}'$ on the direct sum, namely,

$$\begin{aligned}\mathcal{L}'\colon \mathbb{F}_{q^{m_1}} \oplus \cdots \oplus \mathbb{F}_{q^{m_r}} &\longrightarrow \mathbb{F}_{q^{m_1}} \oplus \cdots \oplus \mathbb{F}_{q^{m_r}}.\\ [\,\widehat{w}_1, \;\ldots,\; \widehat{w}_r\,] &\longmapsto [\,\widehat{w}_1^q, \;\ldots,\; \widehat{w}_r^q\,]\end{aligned}$$

Clearly, the minimum polynomials of Q, $\mathcal{L}$, and $\mathcal{L}'$ are all the same. Furthermore, the minimum polynomial of $\mathcal{L}'$ is the LCM of the minimum polynomials of $\mathcal{L}'_i$, the restrictions of $\mathcal{L}'$ to the i^{th} direct summand $\mathbb{F}_{q^{m_i}}$.

We derive the minimum polynomial of the restricted linear operator $\mathcal{L}'_i$ by observing that that polynomial must linearly generate the infinite sequence of elements in $\mathbb{F}_{q^{m_i}}$

$$a_0,\ a_0^q,\ a_0^{q^2}, \ldots,\ a_j = a_0^{q^j}, \ldots, \tag{6}$$

for any $a_0 \in \mathbb{F}_{q^{m_i}}$. However, since $a_j^{q^{m_i}} = a_j$, the sequence is linearly generated over $\mathbb{F}_q$ by $\lambda^{m_i} - 1 \in \mathbb{F}_q[\lambda]$. Finally, if a_0 generates a normal basis for $\mathbb{F}_{q^{m_i}}$ over $\mathbb{F}_q$ (see von zur Gathen and Giesbrecht 1990), then no proper divisor of $\lambda^{m_i} - 1$ over $\mathbb{F}_q[\lambda]$ can linearly generate (6) over $\mathbb{F}_q$. ⊠

3. The Black Box Algorithm

The theory and practice of "black box" linear algebra is based on the fact that a linear system can be solved efficiently when a fast algorithm for multiplying the coefficient matrix by a vector is known. Thus, these linear system solvers (Wiedemann 1986, Kaltofen and Saunders 1991, Coppersmith 1994, Kaltofen 1993 and 1995) are applicable not only to sparse systems but also to systems such as the one arising in the Berlekamp algorithm. For later reference, we now describe this method in some detail, supposing that q is not a power of 2. The strange grouping of the individual steps will make sense later.

Step BW1: This step and Step BW2 computes a vector $\vec{v}$ that satisfies Theorem 1. The algorithm used is Wiedemann's for finding a non-zero solution to $\vec{v}\cdot(Q - I) = 0$. In this step we compute $\mu^Q(\lambda) = \mu^{Q-I}(\lambda - 1)$.

Step BW1a: *Pick random projection row vectors $\vec{u}, \vec{b} \in \mathbb{F}_q^n$ and compute the sequence of field elements*

$$a_i = \vec{u}\cdot Q^i \cdot \vec{b}^{\,\mathrm{tr}} \quad \textit{for all } 0 \le i < 2n. \tag{7}$$

Let u be the polynomial corresponding to the coefficient vector $\vec{u}$. Then by Theorem 1 we have

$$\vec{u}\cdot Q^i = \overrightarrow{u^{q^i} \bmod f}, \tag{8}$$

which allows for fast computation of the a_i.

Step BW1b: *By any version of the Berlekamp/Massey algorithm (Massey 1969, Brent et al. 1980, Dornstetter 1987) compute a linear recurrence of degree no more than n which generates (7).* The polynomial corresponding to the linear recurrence is with high probability equal to μ^Q. *If the recurrence polynomial is equal to $\lambda^n - 1$, then f is definitely irreducible.*

Step BW2: Let $\mu_0^{Q-I}(\lambda) = \mu^{Q-I}(\lambda)/\lambda^k$, where λ^k is the highest power of λ dividing $\mu^{Q-I}(\lambda)$. For the random row vector $\vec{u}$ we have with high probability for some $l \ge 0$ that

$$\underbrace{\vec{u}\cdot\mu_0^{Q-I}(Q-I)\cdot(Q-I)^l}_{\vec{v}\neq 0}\cdot(Q-I) = 0. \tag{9}$$

Therefore, *compute $\vec{w} = \vec{u}\cdot\mu_{-1}^Q(Q)$, where $\mu_{-1}^Q(\lambda) = \mu^Q(\lambda)/(\lambda-1)^k$ with $(\lambda-1)^k$ being the highest power of $\lambda - 1$ dividing $\mu^Q(\lambda)$. If $\vec{w} = 0$, our randomizations have been unlucky. If not, compute*

$$\vec{w}\cdot(Q-I),\ \vec{w}\cdot(Q-I)^2, \ldots$$

until a zero vector appears or the degree of μ^{Q-I} is reached; in the latter case the randomizations were unlucky. If the zero vector is reached, the previously computed vector $\vec{v}$ is a non-zero solution to $\vec{v}\cdot(Q-I) = 0$. *If $\vec{v} = [a, 0, \ldots, 0]$ for some $a \in \mathbb{F}_q$, the randomizations have again been unlucky.*

Step BW3: Let v be the polynomial corresponding to the vector $\vec{v}$ computed in Step BW2. *For random $t \in \mathbb{F}_q$, compute $\mathrm{GCD}((v-t)^{(q-1)/2} - 1 \bmod f, f)$ until f is split.*

For very large q, the use of (8) for computing $\vec{u}\cdot Q$, which costs with FFT-based polynomial arithmetic $O(\log q \cdot n \log n \operatorname{loglog} n)$ field arithmetic operations in $\mathbb{F}_q$, is inefficient. A possible alternative is to compute $g(x) = (x^q \bmod f(x))$ and $u(x)^q \equiv u(g(x)) \pmod{f(x)}$, the latter with precomputed g in $O(n^2 \log n \operatorname{loglog} n)$ arithmetic steps. Indeed, the entire sequence can be computed in

$$O(\,(\log q + n\log n)\cdot n\log n \operatorname{loglog} n\,) \tag{10}$$

arithmetic operations using Algorithm 3.1 from von zur Gathen and Shoup (1992). We shall return to this last remark later.

The probabilistic analysis of the above algorithm poses several theoretical problems. For one, we must guarantee that a usable polynomial v in the subalgebra is found with probability bounded away from zero. Secondly, it is desired that the splits are in some sense even, so that a complete factorization can be found within a $O(\log r)$ time factor rather than a $\Omega(r)$ factor.

This is a problem similar to the expected versus the worst case running time of the quicksort algorithm. A straight-forward solution to these problems is the use of algorithms proposed in Kaltofen and Saunders (1991). The expected number of arithmetic steps in $\mathbb{F}_q$ is then asymptotically related to the function

$$(\min\{n^2, n\log q\} + \log q)\cdot n(\log n)^2 \operatorname{loglog} n \cdot \max\{1, \log_q n\} \quad (11)$$

Here the term $n\cdot\min\{n,\log q\}\cdot n\log n \operatorname{loglog} n$ accounts for the computation of the $3n$ q^{th}-powers in the sequence (7) by the two methods described above. Note that the preconditioning of $Q - I$ introduced in (Kaltofen and Saunders 1991) only adds an extra $O(n^2\log n \times \operatorname{loglog} n)$ field operations in the computation of the entire sequence. The term $\log q \cdot n\log n \operatorname{loglog} n$ accounts for the computation of $x^q \bmod f(x)$ and $(v(x) - t)^{(q-1)/2} \bmod f(x)$. The extra factor $\log n$ is derived from the cost of recursively splitting the factors. Finally, the factor $\max\{1, \log_q n\}$ arises from the required field extensions in the linear system solver (not the factorizer), because the random elements needed by Kaltofen and Saunders (1991) must be sampled from a field of cardinality $\Omega(n)$.

The running time measure (11) only has historic value. The first author invented that variant of the black box algorithm as a more space efficient version of the Berlekamp algorithm: it requires only $O(n)$ storage for field elements; and it has served as a starting point for von zur Gathen's and Shoup's break-through algorithm of

$$O((n+\log q)\cdot n(\log n)^2 \operatorname{loglog} n)$$

expected field operations. However, Steps BW1–BW3 as described here constitute an interesting variant of Berlekamp algorithm. Its merits over other algorithms from an implementation point of view will be discussed in Section 4. In the remainder of this section we shall explore surprising connections to the distinct-degree and the Cantor/Zassenhaus polynomial factorization algorithms, thereby justifying good probabilistic behavior of our algorithm.

As stated above, Steps BW1a and BW2 can be realized more efficiently by use of von zur Gathen's and Shoup's Algorithm 3.1. They use their algorithm for computing a distinct-degree factorization of f. The relation is not coincidental, as we shall explain now. Consider the algebra isomorphism Φ of the proof of Theorem 2. The component corresponding to the factor f_i in the sequence $u_k = u_0^{q^k} \in \mathbb{F}_q[x]/(f(x))$, $k \geq 0$ has the minimum polynomial $\lambda^{m_i} - 1$, which implies that

$$f_i \text{ divides } \operatorname{GCD}(u_0^{q^{m_i}} - u_0, f).$$

In fact, all factors of f whose degrees divide m_i are factors of the right side GCD. If the components of $\Phi(u_0)$ corresponding to the other factors are normal elements, those factors cannot divide $u_0^{q^{m_i}} - u_0$. A suitable interpretation of the distinct-degree factorization algorithm (Knuth 1981, §4.6.2) is now possible: it sequentially filters the factors of degree $m = 1, 2, \ldots$ by use of their minimum polynomials $\lambda^m - 1$ and the sequence starting at $u_0 = x \bmod f(x)$. Note that our Step BW1 does not perform this sequential filter but computes a common minimum polynomial for all components of $\Phi(u_0)$, thus saving potentially $\Omega(n)$ costly GCD computations.

If the degrees of the factors are known in advance, the polynomial μ^Q can be computed via Theorem 2 and Step BW1 can be omitted. This is true, e.g., for the equal degree factorization problem, where all $m_i = m = n/r$. Then $\mu^Q(\lambda) = \lambda^m - 1$ and

$$\mu_{-1}^Q(\lambda) = 1 + \lambda + \lambda^2 + \cdots + \lambda^{m-1},$$

provided that m is not divisible by the characteristic of $\mathbb{F}_q$. The polynomial w corresponding to $\vec{w}$ computed in Step BW2 is by (8) equal to

$$w = u + u^q + u^{q^2} + \cdots + u^{q^{m-1}} \bmod f,$$

the well-known trace of u in $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ in a variant of the Cantor/Zassenhaus algorithm (Ben-Or 1981). Indeed, w is polynomial in the subalgebra of Theorem 1 with $w \bmod f_i$ being random elements in $\mathbb{F}_q$. Our general algorithm also maps the i^{th} component of $\Phi(u)$ into $v \bmod f_i \in \mathbb{F}_q$, but the argument is a bit more involved.

Let us suppose that the algorithm has correctly computed μ^Q in Step BW1. With a sufficient probability this is true (see Wiedemann 1986). Let p be the characteristic of $\mathbb{F}_q$, and let p^k be that highest power of p that divides at least one of the factor degrees m_j, where $1 \leq j \leq r$. Furthermore, let m_i be one of the factor degrees that is divisible by p^k. We will establish the following claim: with probability no less than $1 - 1/q$ we have in (9) that $l = p^k - 1$ and $v \bmod f_i$ is a random non-zero element in $\mathbb{F}_q$, and that $v \bmod f_j = 0$ for all j where m_j is not divisible by p^k. We begin the proof of the claim by observing that since k is maximal and

$$\begin{aligned}\lambda^{m_i} - 1 &= (\lambda^{m_i/p^k} - 1)^{p^k}\\ &= (1 + \lambda + \cdots + \lambda^{m_i/p^k - 1})^{p^k}(\lambda - 1)^{p^k}, \quad (12)\end{aligned}$$

we must have by Theorem 2

$$\mu_{-1}^Q(\lambda)\cdot(\lambda - 1)^{p^k - 1} = \mu^Q(\lambda)/(\lambda - 1).$$

Next, we define the co-factor of the trace polynomial:

$$\mu_{-1}^Q(\lambda)\cdot(\lambda - 1)^{p^k-1} = (1 + \lambda + \cdots + \lambda^{m_i - 1}) \cdot \underbrace{\Big(\sum_j h_{i,j}\lambda^j\Big)}_{h_i^*(\lambda)\,\in\,\mathbb{F}_q[\lambda]}. \quad (13)$$

From (12) and (13) it follows that $\lambda-1$ cannot be a root of $h_i^*(\lambda)$; that is, $\sum_j h_{i,j} \neq 0$. Now, let $\tau_i = u + u^q + \cdots + u^{q^{m_i-1}} \bmod f$ and $\hat{\tau}_i = \tau_i \bmod f_i$. For a random vector $\vec{u}$ the element $\hat{\tau}_i$, as the trace in $\mathbb{F}_{q^{m_i}}$ over $\mathbb{F}_q$ of the corresponding random i^{th} component of the image $\Phi(u)$ (see Proof of Theorem 2), is a random element in $\mathbb{F}_q$. Then

$$v' \bmod f_i = \sum\nolimits_k h_{i,k} \hat{\tau}_i^{q^k} = \hat{\tau}_i \cdot \sum\nolimits_k h_{i,k}.$$

is also a random element in $\mathbb{F}_q$, where

$$\vec{v}\,' = \vec{u} \cdot \mu_{-1}^Q(Q) \cdot (Q-I)^{p^k-1}.$$

Thus, with probability $1-1/q$ the random field element $v' \bmod f_i \neq 0$, in particular $\vec{v}\,' \neq 0$. Under those circumstances the zero vector will be reached in Step BW2 after exactly p^k multiplications with $Q-I$; that is, $l = p^k - 1$ in (9). For if $l < p^k - 1$ then we would have

$$0 \neq \vec{v}\,' = \vec{v} \cdot (Q-I)^{p^k-1-l} = 0,$$

a contradiction. The claim that in the case where $l = p^k - 1$ we have $v \bmod f_j = 0$ when p^k does not divide m_j follows similarly from the fact that then $(1 + \lambda + \cdots + \lambda^{m_j-1}) \cdot (\lambda - 1)$ is a factor of (13).

In conclusion of the just established claim, we are guaranteed that with probability no less than $1-1/q-1/(q-1)$ there are 2 indices $i \neq j$ with $v \bmod f_i \neq v \bmod f_j$. Again there are two cases to consider. If not all m_j are divisible by p^k, then we may choose i such that p^k divides m_i and j such that p^k does not divide m_j. Note that in this case with probability $1-1/q$ the $\mathrm{GCD}(v, f)$ is non-trivial. On the other hand, if all m_i have the same maximal power divisor p^k, one concludes that with probability no less than $1-1/q$ a random non-zero component is computed, which is distinct from any other component with probability $1-1/(q-1)$. Let us assume now that $v \bmod f_i \neq v \bmod f_j$. In Step BW3 the field elements $v-t \bmod f_i$ and $v-t \bmod f_j$ then have opposite quadratic residuosity with probability $1/2 - 1/(2q)$ (Rabin 1980) and f is split.

In order to compute the full factorization of f, the algorithm must continue after having determined the first split. There are several tricks on how to use the polynomial v of the Butler subalgebra (Theorem 1) to generate more than one split, one of which we have utilized in our implementation. However, from an asymptotic and worst case expected time point of view it is sufficient to just call the algorithm recursively on the factors discovered in Step BW3, because the splits in the algorithm are sufficiently random as proven above. Let us assume that the sequence (7) is computed in the fastest known time (10). We wish to argue that the complete factorization of f can be computed within at most a $O(\log r)$ factor of (10). For that we must show that the recursion tree has expected depth $O(\log r)$. However, this is easily established from the above claims if we make a modification in Step BW3 and first compute $g = \mathrm{GCD}(v, f)$ and $h = f/\mathrm{GCD}(v, f)$ before splitting h by $\mathrm{GCD}((h-t)^{(q-1)/2} - 1 \bmod f, f)$. In summary, we have the following theorem.

Theorem 3. *We have a probabilistic algorithm that factors a polynomial of degree n over $\mathbb{F}_q$ using an expected number of*

$$O(\ (n \log n + \log q) \cdot n (\log n)^2 \operatorname{loglog} n\ )$$

operations in $\mathbb{F}_q$.

Note that the complexity accomplished in Theorem 3 is within at most a $O(\log n)$ factor of the algorithm by von zur Gathen and Shoup. Of course, we must make use of one of their subalgorithms. We remark that the $\log_q n$ factor of (11) is saved by the analysis of Wiedemann (1986, §VI), while we do not know how to fully amortize the recursive descent by von zur Gathen's and Shoup's (1992, §4) game of "balls and bins."

4. Implementation

We have implemented the black box Berlekamp algorithm using the blocked version of Wiedemann's algorithm for a linear system solver. The blocked solver is run in a distributed computing environment, providing the speedup over the sequential algorithm of §3 which is necessary for inputs of the size that we consider. We make no assumptions about the number of irreducible factors or about their multiplicities and their degrees. Thus we have constructed a general-purpose factorizer which performs the complete factorization of high-degree polynomials modulo primes of moderate size.

In our linear system solver, the matrix-times-vector product is computed by a call to a generic black box (see Figure 1). This black box accepts a vector $\vec{u}_{in}$ and generates the vector $\vec{u}_{out}$, where $u_{out} = u_{in}^p - u_{in} \bmod f$. Note that we do not use the simplified black box calls of Steps BW1a and BW2 in §3 because our implementation employs our general purpose black box linear system package. In that package, the black box call is treated as a function argument mapping a vector into a vector. Static data for p and f are supplied by an initialization function.

The linear solver is an implementation of the block Wiedemann algorithm (see Coppersmith 1994, Kaltofen 1993 and 1995, Díaz et al. 1993). Its purpose is to generate upon demand vectors $\vec{v}$ in the left null space of $Q-I$. There are four distinct parts to the linear solver (see Figure 1). In the first part, m pairs of random row

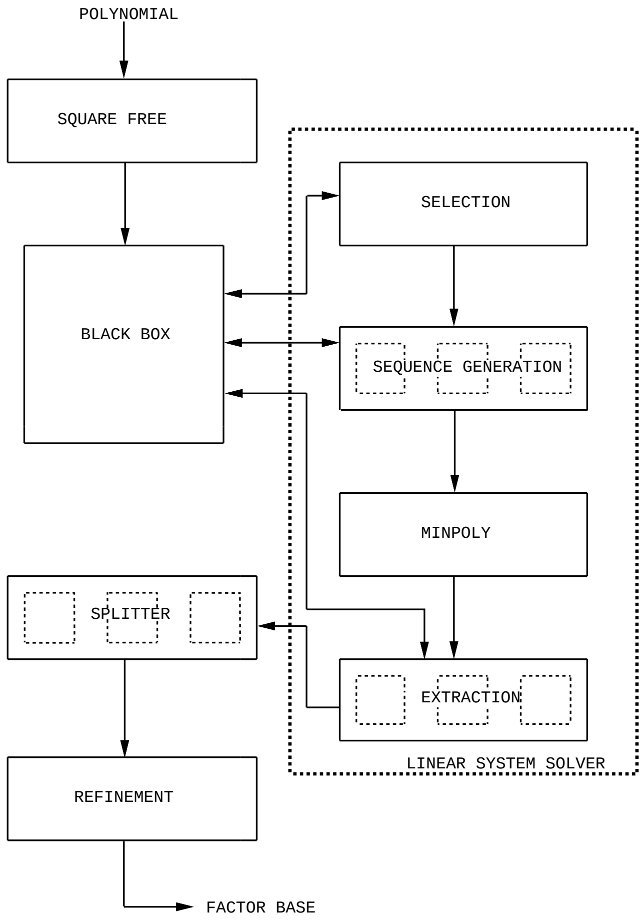

Figure 1: Software architecture of the black box factorizer.

vectors $\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_m$ and $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m$ all in $\mathbb{F}_p^n$ are selected subject to the criterion that $\boldsymbol{a}^{(0)} = \vec{\boldsymbol{u}} \cdot (Q - I) \cdot \vec{\boldsymbol{b}}^{\mathrm{tr}} \in \mathbb{F}_p^{m \times m}$ is nonsingular.

In the second part, the sequence $\boldsymbol{a}^{(i)} = \vec{\boldsymbol{u}} \cdot (Q - I)^{i+1} \cdot \vec{\boldsymbol{b}}^{\mathrm{tr}}$ for $i = 1, 2, \ldots, L = \lfloor 2N/m \rfloor + 3$ is computed (cf. (7)). This step can be speeded to L parallel calls of the matrix-times-vector black box function by computing the m rows of $\boldsymbol{a}^{(i)}$ in parallel as $\boldsymbol{a}^{(i)}_{j,*} = \vec{u}_j \cdot (Q - I)^{i+1} \cdot \vec{\boldsymbol{b}}^{\mathrm{tr}}$ and then merging the results. Parts 1 and 2 are the counterpart of Step BW1a in §3.

In the third part, a block version of the Berlekamp/Massey algorithm (Coppersmith 1994) computes a generator matrix polynomial Λ for the sequence $\boldsymbol{a}^{(i)}$. This part corresponds to Step BW1b. The fourth and final part extracts as many as m vectors $\vec{v}$ by a Horner-type evaluation at $Q - I$ of a polynomial derived from Λ with coefficients that are m-dimensional vectors. This part corresponds to Step BW2 of §3.

By the theory developed in §3, we know that the matrix $Q - I$ does not satisfy the conditions required in Kaltofen's (1993 and 1995) analysis of the block Wiedemann algorithm. Despite this property, the algorithm seems to produce correct solution vectors $\vec{v}$. It appears to us that the rank estimates made in (Kaltofen 1993 and 1995) can be relaxed to account for the observed phenomenon, but we have not yet been able to prove this. Furthermore, unlike in the sequential case, we do not know how the degree of Λ is related to the degrees of the irreducible factors.

Candidates $\vec{v}_j$ from the solver go to the splitter where $\mathrm{GCD}(v_j^{(q-1)/2} - 1 \bmod f, f)$ is computed and hopefully f is split. After this the splits are refined and put into a factor base. Factor refinement will yield all r factors of f with high probability provided at least $2 \log_2 r$ random splits are available. One approach to obtaining so many splits is to increase the blocking factor m or, alternatively, to call the linear solver again. A far cheaper approach is to compute $\mathrm{GCD}((v_j - t)^{(q-1)/2} - 1 \bmod f, f)$ for several random $t \in \mathbb{F}_p$. This "mining" of a single vector for additional splits can be done at very low cost, and is easily implemented in a distributed computing environment. For small p sufficiently many distinct splits might not be obtained by this technique alone, in which case the randomized linear system solver will have to be called again. For factor refinement we use the simple algorithm in (Kaltofen 1985), which also runs in quadratic time like the more complex algorithm by (Bach et al. 1993).

If the number of factors r is small, all $t_k \in \mathbb{F}_q$ that result in splits $\mathrm{GCD}(v_j - t_k \bmod f, f)$ can be determined efficiently by finding a linear dependence of $v_j^k \bmod f$ for $0 \le k \le r$ (see Knuth 1981, §4.6.2, Exercise 14). However, the splitting and factor refinement phase of our implementation costs only a small fraction of the total time (see Figure 2 below) and it is therefore not crucial to speed it in that way. A more important question is how to verify the irreducibility of all produced factors quickly. However, this problem is solved if a solution to the block Berlekamp/Massey problem is computed by the generalized Levinson/Durbin method (see Kaltofen 1995, §6). Then we obtain a value s that with high probability is equal to the rank of $Q - I$. Note that $r = n - \mathrm{rank}(Q - I)$ and, as can be shown (ibid., Proof of Theorem 7), we always have $s \le \mathrm{rank}(Q - I)$. Therefore, once we find $n - s$ different factors, they all must be irreducible. We plan to add this modification to our algorithm.

The black box and assorted functions are built with member functions from C++ classes `ZZ_pX` for polynomials, `ZZ_p` for elements in $\mathbb{F}_q$, and `ZZ` for integers of arbitrary length. Vectors of field elements are defined and instantiated by means of macros. A class `Block` was designed for the sequence elements $\boldsymbol{a}^{(i)}$. The classes `ZZ_pX` and `ZZ_p` are from Shoup's (1993) implementa-

Degree n	Prime p	Task	Blocking Factor 8	16	32	Factor degrees
1025	32749	sequence	$0^h58'$	$0^h34'$	$0^h25'$	1, 1, 1, 1,
		minpoly	$0^h04'$	$0^h09'$	$0^h24'$	1, 2, 2, 5,
		evaluation	$0^h28'$	$0^h14'$	$0^h07'$	6, 11, 30,
		split/refine	$0^h12'$	$0^h14'$	$0^h11'$	153, 161,
		total time	$1^h42'$	$1^h12'$	$1^h07'$	180, 470
		work	$955^{\#}$	$1000^{\#}$	$1165^{\#}$	
2049	127	sequence	$1^h30'$	$0^h49'$	$0^h32'$	1, 1, 1, 1,
		minpoly	$0^h06'$	$0^h15'$	$1^h02'$	6, 6, 7,
		evaluation	$0^h43'$	$0^h22'$	$0^h12'$	63, 66, 74,
		split/refine	$0^h14'$	$0^h13'$	$0^h13'$	83, 371,
		total time	$2^h32'$	$1^h39'$	$1^h59'$	418, 951
		work	$1530^{\#}$	$1585^{\#}$	$1824^{\#}$	
4097	127	sequence	$7^h19'$	$3^h49'$		1, 1, 4, 5, 9,
		minpoly	$1^h10'$	$1^h57'$		21, 31, 42,
		evaluation	$3^h32'$	$1^h49'$		43, 85, 88,
		split/refine	$0^h44'$	$0^h32'$		139, 633, 863,
		total time	$12^h45'$	$8^h11'$		897, 1235
		work	$7613^{\#}$	$7821^{\#}$		
10001	127	sequence	$50^h52'$			1, 1, 1, 2, 3, 4,
		minpoly	$5^h45'$			7, 30, 35, 77,
		evaluation	$25^h04'$			121, 161, 621,
		split/refine	$1^h32'$			749, 1374,
		total time	$83^h14'$			2682, 4132
		work	$52858^{\#}$			

Figure 2: Parallel CPU time (hours^{h}minutes$'$) for factoring

$$(x^{\lceil n/2 \rceil} + x + 1) \cdot (x^{\lfloor n/2 \rfloor} + x + 1) \pmod{p}$$

on 86.1 MIPS computers. Work is measured in units of MIPS-hours$^{\#}$.

tion. The class ZZ was designed by A. K. Lenstra and in the C++ hierarchy `ZZ_pX` $\supset$ `ZZ_p` $\supset$ `ZZ`. For finding the linear recurrence, Λ we used a version of the Berlekamp/Massey module, written in ANSI C, which was mentioned in Díaz et al. (1993). This module is extremely fast and we preferred it to our C++ version even though p is restricted to a value within 15 bits, that is, 32749.

Our test cases (see Figure 2) are generated from trinomials of the form x^n+x+1 and are the product of two such trinomials differing in degree by 1. This is partly in the spirit of the challenge by von zur Gathen (1992); however, the primes are smaller than proposed there. Although trinomials and products are quite sparse, we do not take advantage of this fact in our modulo f computations. In fact, we have factored a random polynomial of degree 10 001 modulo 127 into 14 irreducible factors on 8 computers in 86.6 hours. The polynomials in Figure 2 are all trinomial products and the degrees of the irreducible factors are listed in the last column. Some jobs were rerun with blocking factors of 8,16 and 32.

In our experiment we chose the blocking factor to be a multiple of 8 but this is not a requirement of our implementation. We are currently experimenting with unequal row and column blocking factors, which result in a sequence, $\boldsymbol{a}^{(i)}$, of rectangular matrices for the purpose of reducing the number of matrix-times-vector products needed in the sequence generation step.

From the first two examples, it appears that 16 is an optimal value for the blocking factor m. Increasing m results in shorter times needed per parallel task, for sequence generation and evaluation. However, the time for finding the generator polynomial of the sequence increases and counters any decrease in the other two timings.

We did not have enough machines available to us for the long duration needed to run the large experiments with blocking factors greater than 8. We note however, that the memory requirements of the jobs are well within the capacity of our base machine which is a Sun 4 with 32 Mbytes of memory, rated at 86.1 MIPS.

We have computed a measure of the total work done by taking the product of the native MIPS (see Patterson and Hennessy 1990, pp. 40–42) of our base machine and the time taken for a task, and summing over all parallel and sequential tasks. The result is represented in the unit "MIPS-hour" which is the number of instructions executed by a 1-MIPS machine running for one hour.

5. Future Directions

The investigations described here are far from finished. There are several theoretical and many implementation issues that we hope to resolve in the future. In terms of theory, our understanding of Coppersmith's block Wiedemann method—while so essential to the success of our factoring tasks—is incomplete. We know that our black box matrices violate the conditions assumed in the only known mathematically proven analysis of the algorithm (Kaltofen 1993 and 1995), but the algorithm produces correct solutions. Moreover, the sequential algorithm of §3 can adapt to savings induced by equal-degree factors. However in the block version, a degree reduction of the recurrence equation induced by equal degree factors, for example, which in the sequential case is explained by Theorem 2, occurs for certain polynomials such as the Swinnerton-Dyer polynomials (Kaltofen et al. 1983) while for others (see Figure 2, $n = 1025$) it does not. We have no explanation for this phenomenon.

The bottleneck in our implementation is the cost for performing a matrix-times-vector product. It is that step which has forced us to use FFT-based arithmetic and to run our tests modulo small primes. It is possible to reduce to approximately one-half the total number of matrix-times-vector products in our approach by using higher blocking from the left, i.e., by choosing $m' \gg m$ rows in $\vec{\boldsymbol{u}}$, and by saving intermediate results of the sequence generation part for the polynomial evaluation part (see Kaltofen 1995, Appendix B). However, in order to push the fringe of achievable factorization problems significantly further the matrix-times-vector step has to be speeded. Currently known alternatives are to switch to the Niederreiter approach, or to utilize Algorithm 3.1 of von zur Gathen and Shoup (1992) in the sequence generation part of the block method, or to parallelize a version of the distinct-degree factorization algorithm (Shoup, private communication). However, we are not entirely sure that either change can substantially help.

Acknowledgement: We appreciate the suggestions for improvement made by the anonymous referees. We are especially grateful to Joachim von zur Gathen and Victor Shoup for many discussions on this topic.

Literature Cited

Bach, E., Driscoll, J., and Shallit, J., "Factor refinement," *J. Algorithms* **15**, pp. 199-222 (1993).

Ben-Or, M., "Probabilistic algorithms in finite fields," *Proc. 22nd IEEE Symp. Foundations Comp. Sci.*, pp. 394–398 (1981).

Berlekamp, E. R., "Factoring polynomials over finite fields," *Bell Systems Tech. J.* **46**, pp. 1853–1859 (1967). Republished in revised form in: E. R. Berlekamp, *Algebraic Coding Theory*, Chapter 6, McGraw-Hill Publ., New York 1968.

Berlekamp, E. R., "Factoring polynomials over large finite fields," *Math. Comp.* **24**, pp. 713–735 (1970).

Brent, R. P., Gustavson, F. G., and Yun, D. Y. Y., "Fast solution of Toeplitz systems of equations and computation of Padé approximants," *J. Algorithms* **1**, pp. 259–295 (1980).

Butler, M. C. R., "On the reducibility of polynomials over a finite field," *Quart. J. Math., Oxford Ser. (2)* **5**, pp. 102–107 (1954).

Camion, P., "Un algorithme de construction des idempotents primitifs d'ideaux d'algebres sur $\mathbb{F}_q$," *Ann. Discrete Math* **12**, pp. 55-63 (1982).

Cantor, D. G. and Zassenhaus, H., "A new algorithm for factoring polynomials over finite fields," *Math. Comp.* **36**, pp. 587–592 (1981).

Coppersmith, D., "Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm," *Math. Comput.* **62**/205, pp. 333–350 (1994).

Díaz, A., Hitz, M., Kaltofen, E., Lobo, A., and Valente, T., "Process scheduling in DSC and the large sparse linear systems challenge," in *Proc. DISCO '93*, Springer Lect. Notes Comput. Sci. **722**, edited by A. Miola; pp. 66–80, 1993. Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

Dornstetter, J. L., "On the equivalence between Berlekamp's and Euclid's algorithms," *IEEE Trans. Inf. Theory* IT-**33**/3, pp. 428–431 (1987).

Fleischmann, P., "Connections between the algorithms of Berlekamp and Niederreiter for factoring polynomials over $\mathbb{F}_q$," *Linear Algebra and Applications* **192**, pp. 101–108 (1993).

Gao, S. and von zur Gathen, J., "Berlekamp's and Niederreiter's polynomial factorization algorithm," in *Finite Fields: Theory, Applications and Algorithms*, Contemporary Mathematics, edited by L. Mullen and P. J.-S. Shiue; Amer. Math. Soc., Providence, Rhode Island, to appear, 1994.

von zur Gathen, J., "The polynomial factorization challenge," *SIGSAM Bulletin* **26**/2, pp. 22–24 (1992).

von zur Gathen, J. and Giesbrecht, M., "Constructing normal basis in finite fields," *J. Symbolic Comput.* **10**/6, pp. 547–570 (1990).

von zur Gathen, J. and Shoup, V., "Computing Frobenius maps and factoring polynomials," *Comput. Complexity* **2**, pp. 187–224 (1992).

Kaltofen, E., "Sparse Hensel lifting," *Proc. EUROCAL '85, Vol. 2, Springer Lec. Notes Comp. Sci.* **204**, pp. 4–17 (1985). Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

Kaltofen, E., "Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems," in *Proc. AAECC-10*, Springer Lect. Notes Comput. Sci. **673**, edited by G. Cohen, T. Mora, and O. Moreno; pp. 195–212, 1993. Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

Kaltofen, E., "Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems," *Math. Comput.*, to appear (1995). Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

Kaltofen, E., Musser, D. R., and Saunders, B. D., "A generalized class of polynomials that are hard to factor," *SIAM J. Comp.* **12**/3, pp. 473–485 (1983).

Kaltofen, E. and Saunders, B. D., "On Wiedemann's method of solving sparse linear systems," in *Proc. AAECC-9*, Springer Lect. Notes Comput. Sci. **539**; pp. 29–38, 1991. Available from `anonymous@ftp.cs.rpi.edu` in directory `kaltofen`.

Knuth, D. E., *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Ed. 2*; Addison Wesley, Reading, MA, 1981.

Massey, J. L., "Shift-register synthesis and BCH decoding," *IEEE Trans. Inf. Theory* IT-**15**, pp. 122–127 (1969).

Niederreiter, H. and Göttfert, R., "Factorization of polynomials over finite fields and characteristic sequences," *J. Symbolic Comput.* **16**/5, pp. 401–412 (1994).

Patterson, D. A. and Hennessy, J. L., *Computer Architecture A Quantitative Approach*; Morgan Kaufmann Pub., San Mateo, California, 1990.

Rabin, M. O., "Probabilistic algorithms in finite fields," *SIAM J. Comp.* **9**, pp. 273–280 (1980).

Schwarz, Št., "On the reducibility of polynomials over a finite field," *Quart. J. Math. Oxford Ser. (2)* **7**, pp. 110–124 (1956).

Shoup, V., "Factoring polynomials over finite fields: asymptotic complexity vs. reality," in *Proc. Int. IMACS Symp. on Symbolic Comput., New Trends and Developments*; Lille, France, pp. 124–129, 1993.

Wiedemann, D., "Solving sparse linear equations over finite fields," *IEEE Trans. Inf. Theory* IT-**32**, pp. 54–62 (1986).