Improved Stage 2 to P±1 Factoring Algorithms

Peter Lawrence Montgomery, Alexander Kruppa

Improved Stage 2 to P ± 1 Factoring Algorithms

Peter L. Montgomery[1] and Alexander Kruppa[2]

[1] Microsoft Research, One Microsoft Way, Redmond, WA 98052 USA.
pmontgom@cwi.nl
[2] LORIA, Campus Scientifique, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France.
kruppaal@loria.fr

Abstract. Some implementations of stage 2 of the P–1 method of factorization use convolutions. We describe a space-efficient implementation, allowing convolution lengths around 2^{23} and stage 2 limit around 10^{16} while attempting to factor 230-digit numbers on modern PC's. We describe arithmetic algorithms on reciprocal polynomials. We present adjustments for the P+1 algorithm. We list some new findings.
Key words. Integer factorization, convolution, discrete Fourier transform, number theoretic transform, P–1, P+1, multipoint polynomial evaluation, reciprocal polynomials.

1 Introduction

John Pollard introduced the P–1 algorithm for factoring an odd composite integer N in 1974 [11, §4]. It hopes that some prime factor p of N has smooth $p-1$. It picks $b_0 \not\equiv \pm 1 \pmod N$ and coprime to N and outputs $b_1 = b_0^e \bmod N$ for some positive exponent e. This exponent might be divisible by all prime powers below a bound B_1. Stage 1 succeeds if $(p-1) \mid e$, in which case $b_1 \equiv 1 \pmod p$ by Fermat's little theorem. The algorithm recovers p by computing $\gcd(b_1 - 1, N)$ (except in rare cases when this GCD is composite). When this GCD is 1, we hope that $p - 1 = qn$ where n divides e and q is not too large. Then

$$b_1^q \equiv (b_0^e)^q = b_0^{eq} = (b_0^{nq})^{e/n} = \left(b_0^{p-1}\right)^{e/n} \equiv 1^{e/n} = 1 \pmod p, \qquad (1)$$

so p divides $\gcd(b_1^q - 1, N)$. Stage 2 of P–1 tries to find p when $q > 1$ but q is not too large. The search bound for q is called B_2.

Pollard [11] tests each prime q in $[B_1, B_2]$ individually. If q_1 and q_2 are successive primes, then look up $b_1^{q_2-q_1} \bmod N$ in a small table. Given $b_1^{q_1} \bmod N$, form $b_1^{q_2} \bmod N$ and test $\gcd(b_1^{q_2} - 1, N)$. He observes that one can combine GCD tests: if $p \mid \gcd(x, N)$ or $p \mid \gcd(y, N)$, then $p \mid \gcd(xy \bmod N, N)$. His stage 2 cost is two modular multiplications per q, one GCD with N at the end, and a few multiplications to build the table.

Montgomery [7] uses two sets S_1 and S_2, such that each prime q in $[B_1, B_2]$ divides a nonzero difference $s_1 - s_2$ where $s_1 \in S_1$ and $s_2 \in S_2$. He forms $b_1^{s_1} - b_1^{s_2}$ using two table look-ups, saving one modular multiplication per q. Sometimes one

$s_1 - s_2$ works for multiple q. Montgomery adapts his scheme to Hugh Williams's P+1 method and Hendrik Lenstra's elliptic curve method (ECM).

These changes lower the constant of proportionality, but stage 2 still uses $O(\pi(B_2) - \pi(B_1))$ (number of primes between B_1 and B_2) operations modulo N.

The end of [11] suggests an FFT continuation to P–1. Silverman [8, p. 844] implements it, using a circular convolution to evaluate a polynomial along a geometric progression. It costs $O(\sqrt{B_2} \log B_2)$ operations to build and multiply two polynomials of degree $O(\sqrt{B_2})$, compared to $O(B_2/\log B_2)$ primes below B_2, so [8] beats [7] when B_2 is large.

Montgomery's dissertation [9] describes an FFT continuation to ECM. He takes the GCD of two polynomials. Zimmermann [13] implements another FFT continuation to ECM, based on evaluating a polynomial at arbitrary points. These cost an extra factor of $\log B_2$ when the points are not a geometric progression. Zimmermann adapts his implementation to $P \pm 1$ methods.

Like [8], we evaluate a polynomial along geometric progressions. We exploit patterns in its roots to generate its coefficients quickly. Those patterns are not present in ECM, so these techniques do not apply there. We aim for low memory overhead, saving it for convolution inputs and outputs (which are elements of $\mathbb{Z}/N\mathbb{Z}$). Using memory efficiently lets us raise the convolution length ℓ. Many intermediate results are reciprocal polynomials, which need about half the storage and can be multiplied using weighted convolutions.

Doubling ℓ costs slightly over twice as much time per convolution, but each longer convolution extends the search for q (and effective B_2) fourfold. Silverman's 1989 implementation used 42 megabytes and allowed 250-digit inputs. It repeatedly evaluated a polynomial of degree 15360 at $8 \cdot 17408$ points in geometric progression, using $\ell = 32768$. This enabled him to achieve $B_2 \approx 10^{10}$.

Today's (2008) PC memories are 100 times as large as that used in [8]. With this extra memory, we achieve $\ell = 2^{23}$, a growth factor of 256. With the same number of convolutions (individually longer lengths but running on faster hardware) our B_2 advances by a factor of $256^2 \approx 6.6\text{e}4$. Supercomputers with huge shared memories do spectacularly.

Section 12 gives some new results, including a record 60-digit P+1 factor.

2 P+1 Algorithm

Hugh Williams [12] introduced a P+1 factoring algorithm. It finds a prime factor p of N when $p + 1$ (rather than $p - 1$) is smooth. It is modeled after P–1.

One variant of the P+1 algorithm chooses $P_0 \in \mathbb{Z}/N\mathbb{Z}$ and lets the indeterminate α_0 be a zero of the quadratic $\alpha_0^2 - P_0\alpha_0 + 1$. We hope this quadratic is irreducible modulo p. If so, its second root in $\mathbb{F}_{p^2}$ will be α_0^p. The product of its roots is the constant term 1. Hence $\alpha_0^{p+1} \equiv 1 \pmod{p}$ when we choose well.

Stage 1 of the P+1 algorithm computes $P_1 = \alpha_1 + \alpha_1^{-1}$ where $\alpha_1 \equiv \alpha_0^e \pmod{N}$ for some exponent e. If $\gcd(P_1 - 2,\ N) > 1$, then the algorithm succeeds. Stage 2 of P+1 hopes that $\alpha_1^q \equiv 1 \pmod{p}$ for some prime q, not too large, and some prime p dividing N.

Most techniques herein adapt to P+1, but some computations take place in an extension ring, raising memory usage if we use the same convolution sizes.

2.1 Chebyshev Polynomials

Although the theory behind P+1 mentions α_0 and $\alpha_1 = \alpha_0^e$, an implementation manipulates primarily values of $\alpha_0^n + \alpha_0^{-n}$ and $\alpha_1^n + \alpha_1^{-n}$ for various integers n rather than the corresponding values (in an extension ring) of α_0^n and α_1^n.

For integer n, the Chebyshev polynomials V_n and U_n are determined by $V_n(X + X^{-1}) = X^n + X^{-n}$ and $(X - X^{-1})U_n(X + X^{-1}) = X^n - X^{-n}$. The use of these polynomials shortens many formulas, such as

$$P_1 \equiv \alpha_1 + \alpha_1^{-1} \equiv \alpha_0^e + \alpha_0^{-e} = V_e(\alpha_0 + \alpha_0^{-1}) = V_e(P_0) \pmod{N}.$$

These polynomials have integer coefficients, so $P_1 \equiv V_e(P_0) \pmod{N}$ is in the base ring $\mathbb{Z}/N\mathbb{Z}$ even when α_0 and α_1 are not.

The Chebyshev polynomials satisfy many identities, including

$$\begin{aligned} V_{mn}(X) &= V_m(V_n(X)), \\ U_{m+n}(X) &= U_m(X)\,V_n(X) - U_{m-n}(X), \qquad (2) \\ U_{m+n}(X) &= V_m(X)\,U_n(X) + U_{m-n}(X), \\ V_{m+n}(X) &= V_m(X)\,V_n(X) - V_{m-n}(X), \qquad (3) \\ V_{m+n}(X) &= (X^2 - 4)\,U_m(X)\,U_n(X) + V_{m-n}(X). \end{aligned}$$

3 Overview of Stage 2 Algorithm

Our algorithm performs multipoint evaluation of polynomials by convolutions. Its inputs are the output of stage 1 (b_1 for P–1 or P_1 for P+1), and the desired stage 2 interval $[B_1, B_2]$.

The algorithm chooses a highly composite odd integer P. It checks for q in arithmetic progressions with common difference $2P$. There are $\phi(P)$ such progressions to check when $\gcd(q, 2P) = 1$.

We need an even convolution length $\ell_{\max}$ (determined primarily by memory constraints) and a factorization $\phi(P) = s_1 s_2$ where s_1 is even and $0 < s_1 < \ell_{\max}$. Sections 5, 9.1 and 11 have sample values.

Our polynomial evaluations will need approximately

$$s_2 \left\lceil \frac{B_2 - B_1}{2P(\ell_{\max} - s_1)} \right\rceil \approx \frac{\phi(P)}{2P} \frac{B_2 - B_1}{s_1(\ell_{\max} - s_1)} \qquad (4)$$

convolutions of length $\ell_{\max}$. We prefer a small $\phi(P)/P$ to keep (4) low. We also prefer s_1 near $\ell_{\max}/2$, say $0.3 \le s_1/\ell_{\max} \le 0.7$.

Using a factorization of $(\mathbb{Z}/P\mathbb{Z})^*$ as described in §5, it constructs two sets S_1 and S_2 of integers such that

(a) $|S_1| = s_1$ and $|S_2| = s_2$.

(b) S_1 is symmetric around 0. If $k \in S_1$, then $-k \in S_1$.
(c) If $k \in \mathbb{Z}$ and $\gcd(k, P) = 1$, then there exist unique $k_1 \in S_1$ and $k_2 \in S_2$ such that $k \equiv k_1 + k_2 \pmod{P}$.

Once S_1 and S_2 are chosen, it computes the coefficients of

$$f(X) = X^{-s_1/2} \prod_{k_1 \in S_1} (X - b_1^{2k_1}) \bmod N \tag{5}$$

by the method in §7. Since S_1 is symmetric around zero, this $f(X)$ is symmetric in X and $1/X$.

For each $k_2 \in S_2$ it evaluates (the numerators of) all

$$f(b_1^{2k_2+(2m+1)P}) \bmod N \tag{6}$$

for $\ell_{\max} - s_1$ consecutive values of m as described in §8, and checks the product of these outputs for a nontrivial GCD with N. This checks $s_1(\ell_{\max} - s_1)$ (not necessarily prime) candidates, hoping to find q.

For the P+1 method, replace (5) by $f(X) = X^{-s_1/2} \prod_{k_1 \in S_1} (X - \alpha_1^{2k_1}) \bmod N$. Similarly, replace b_1 by α_1 in (6). The polynomial f is still over $\mathbb{Z}/N\mathbb{Z}$ since each product $(X - \alpha_1^{2k_1})(X - \alpha_1^{-2k_1}) = X^2 - V_{2k_1}(P_1) + 1 \in (\mathbb{Z}/N\mathbb{Z})[X]$ but the multipoint evaluation works in an extension ring. See §8.1.

4 Justification

Let p be an unknown prime factor of N. As in (1), assume $b_1^q \equiv 1 \pmod{p}$ where q is not too large, and $\gcd(q, 2P) = 1$.

The selection of S_1 and S_2 ensures there exist $k_1 \in S_1$ and $k_2 \in S_2$ such that $(q - P)/2 \equiv k_1 + k_2 \pmod{P}$. That is,

$$q = P + 2k_1 + 2k_2 + 2mP = 2k_1 + 2k_2 + (2m + 1)P \tag{7}$$

for some integer m. We can bound m knowing bounds on q, k_1, k_2, as detailed in §5. Both $b_1^{\pm 2k_1}$ are roots of $f \pmod{p}$. Hence

$$f(b_1^{2k_2+(2m+1)P}) = f(b_1^{q-2k_1}) \equiv f(b_1^{-2k_1}) \equiv 0 \pmod{p}. \tag{8}$$

For the P+1 method, if $\alpha_1^q \equiv 1 \pmod{p}$, then (8) evaluates f at $X = \alpha_1^{2k_2+(2m+1)P} = \alpha_1^{q-2k_1}$. The factor $X - \alpha_1^{-2k_1}$ of $f(X)$ evaluates to $r^{-2k_1}(\alpha_1^q - 1)$, which is zero modulo p even in the extension ring.

5 Selection of S_1 and S_2

Let "+" of two sets denote the set of sums. By the Chinese Remainder Theorem,

$$(\mathbb{Z}/(mn)\mathbb{Z})^* = n(\mathbb{Z}/m\mathbb{Z})^* + m(\mathbb{Z}/n\mathbb{Z})^* \text{ if } \gcd(m, n) = 1. \tag{9}$$

This is independent of the representatives: if $S \equiv (\mathbb{Z}/m\mathbb{Z})^* \pmod m$ and $T \equiv (\mathbb{Z}/n\mathbb{Z})^* \pmod n$, then $nS + mT \equiv (\mathbb{Z}/(mn)\mathbb{Z})^* \pmod{mn}$. For prime powers, $(\mathbb{Z}/p^k\mathbb{Z})^* = (\mathbb{Z}/p\mathbb{Z})^* + \sum_{i=1}^{k-1} p^i(\mathbb{Z}/p\mathbb{Z})$.

We choose S_1 and S_2 so that $S_1 + S_2 \equiv (\mathbb{Z}/P\mathbb{Z})^* \pmod P$ which ensures that all values coprime to P, in particular all primes, in the stage 2 interval are covered. One way uses a factorization $mn = P$ and (9). Other choices are available by factoring individual $(\mathbb{Z}/p\mathbb{Z})^*$, $p \mid P$, into smaller sets of sums.

Let $R_n = \{2i - n - 1 : 1 \le i \le n\}$ be the arithmetic progression centered at 0 of length n and common difference 2. For odd primes p, a set of representatives of $(\mathbb{Z}/p\mathbb{Z})^*$ is R_{p-1}. Its cardinality is composite for $p \ne 3$ and the set can be factored into arithmetic progressions of prime length by $R_{mn} = R_m + mR_n$. If $p \equiv 3 \pmod 4$, alternatively $\frac{p+1}{4}R_2 + \frac{1}{2}R_{(p-1)/2}$ can be chosen as a set of representatives with smaller absolute values.

When evaluating (6) for all $m_1 \le m < m_2$ and $k_2 \in S_2$, the highest exponent coprime to P that is *not* covered at the low end of the stage 2 range will be $2\max(S_1 + S_2) + (2m_1 - 1)P$. Similarly, the smallest value at the high end of the stage 2 range not covered is $2\min(S_1 + S_2) + (2m_2 + 1)P$. Hence, for a given choice of P, S_1, S_2, m_1 and m_2, all primes in $[(2m_1 - 1)P + 2\max(S_1 + S_2) + 1, (2m_2 + 1)P + 2\min(S_1 + S_2) - 1]$ are covered.

Choose parameters that minimize $s_2 \cdot \ell_{\max}$ so that $[B_1, B_2]$ is covered, $\ell_{\max}$ is permissible by available memory, and, given several choices, $(2m_2 + 1)P + 2\min(S_1 + S_2)$ is maximal.

For example, to cover the interval $[1000, 500000]$ with $\ell_{\max} = 512$, we might choose $P = 1155$, $s_1 = 240$, $s_2 = 2$, $m_1 = -1$, $m_2 = 271$. With $S_1 = 231(\{-1, 1\}+\{-2, 2\})+165(\{-2, 2\}+\{-1, 0, 1\})+105(\{-3, 3\}+\{-2, -1, 0, 1, 2\})$ and $S_2 = 385\{-1, 1\}$, we have $\max(S_1 + S_2) = -\min(S_1 + S_2) = 2098$ and thus cover all primes in $[-3 \cdot 1155 + 4196 + 1,\ 541 \cdot 1155 - 4196 - 1] = [732, 620658]$.

6 Circular Convolutions and Polynomial Multiplication

Let R be a ring and ℓ a positive integer. All rings herein are assumed commutative with 1. A *circular convolution* of length ℓ over R multiplies two polynomials $f_1(X)$ and $f_2(X)$ of degree at most $\ell - 1$ in $R[X]$, returning $f_1(X)f_2(X) \bmod X^\ell - 1$. When $\deg(f_1) + \deg(f_2) < \ell$, this gives an exact product.

If R has a primitive ℓ-th root ω of unity, and if ℓ is not a zero divisor in R, then one convolution algorithm uses the discrete Fourier transform (DFT) [1, chapter 7]. Fix ω. A *forward DFT* evaluates all $f_1(\omega^i)$ for $0 \le i < \ell$. Another forward DFT evaluates all ℓ values of $f_2(\omega^i)$. Multiply these pointwise. Then an *inverse DFT* interpolates to find a polynomial $f_3 \in R[X]$ of degree at most $\ell - 1$ with $f_3(\omega^i) = f_1(\omega^i)f_2(\omega^i)$ for all i. Return f_3.

If ℓ is a power of 2 and we use a fast Fourier transform (FFT) algorithm for the forward and inverse DFTs, then the convolution takes $O(\ell \log \ell)$ operations in a suitable ring, compared to $O(\ell^2)$ ring operations for the naïve algorithm.

6.1 Convolutions over $\mathbb{Z}/N\mathbb{Z}$

The DFT cannot be used directly when $R = \mathbb{Z}/N\mathbb{Z}$, since we don't know a suitable ω. As in [13, p. 534], we consider two ways to do the convolutions.

Montgomery [8, §4] suggests a number theoretic transform (NTT). He treats the input polynomial coefficients as integers in $[0, N-1]$ and multiplies the polynomials over $\mathbb{Z}$. The product polynomial, reduced modulo $X^\ell - 1$, has coefficients in $[0,\ \ell(N-1)^2]$. Select distinct NTT primes p_j that each fit into one machine word such that $\prod_j p_j > \ell(N-1)^2$. Require each $p_j \equiv 1 \pmod{\ell}$, so a primitive ℓ-th root of unity exists. Do the convolution modulo each p_j and use the Chinese Remainder Theorem (CRT) to determine the product over $\mathbb{Z}$ modulo $X^\ell - 1$. Reduce this product modulo N. Montgomery's dissertation [9, chapter 8] describes these computations in detail.

The convolution codes need interfaces to (1) zero a DFT buffer (2) insert an entry modulo N in a DFT buffer, i.e. reduce it modulo the NTT primes, (3) perform a forward, in-place, DFT on a buffer, (4) multiply two DFT buffers pointwise, overwriting an input, and perform an in-place inverse DFT on the product, and (5) extract a product coefficient modulo N via a CRT computation and reduction modulo N.

The Kronecker-Schönhage convolution algorithm uses fast integer multiplication. See §11. Nussbaumer[10] gives other convolution algorithms.

6.2 Reciprocal Laurent Polynomials and Weighted NTT

Define a *reciprocal Laurent polynomial* (RLP) in X to be an expression $a_0 + \sum_{j=1}^{d} a_j \cdot (X^j + X^{-j}) = a_0 + \sum_{j=1}^{d} a_j V_j(X + X^{-1})$ for scalars a_j in a ring. It is *monic* if $a_d = 1$. It is said to have degree $2d$ if $a_d \neq 0$. The degree is always even. A monic RLP of degree $2d$ fits in d coefficients (excluding the leading 1).

While manipulating RLPs of degree at most $2d$, the *standard basis* is $\{1\} \cup \{X^j + X^{-j} : 1 \le j \le d\} = \{1\} \cup \{V_j(Y) : 1 \le j \le d\}$ where $Y = X + X^{-1}$.

Let $Q(X) = q_0 + \sum_{j=1}^{d_q} q_j(X^j + X^{-j})$ be an RLP of degree at most $2d_q$ and likewise $R(X)$ an RLP of degree at most $2d_r$. To obtain the product RLP $S(X) = Q(x)R(x) = s_0 + \sum_{j=1}^{d_s} s_j(X^j + X^{-j})$ of degree at most $2d_s = 2(d_q + d_r)$, choose a convolution length $\ell > d_s$ and perform a weighted convolution product [4] by computing $\tilde{S}(wX) = Q(wX)R(wX) \bmod (X^\ell - 1)$ for a suitable w. Suppose $\tilde{S}(wX) = \sum_{i=0}^{\ell-1} \tilde{s}_i X^i = S(wX) \bmod (X^\ell - 1)$. If $0 \le i \le d_s$, then the coefficient of X^i or X^{-i} in $Q(X)R(X)$ is s_i. The coefficient of X^i in $Q(wX)R(wX)$ is $s_i w^i$, whereas its coefficient of X^{-i} is s_i/w^i. When $0 \le i < \ell$, the coefficient $\tilde{s}_i$ of X^i in $\tilde{S}(wX)$ has a contribution $s_i w^i$ from X^i in $Q(wX)R(wX)$ (if $i \le d_s$) as well as $s_{\ell-i}/w^{\ell-i}$ from $X^{i-\ell}$ (if $d_s < \ell - i$). This translates to $\tilde{s}_i = s_i w^i$ when $0 \le i < \ell - d_s$, which we can solve for s_i. When instead $\ell - d_s \le i \le d_s$, we find $\tilde{s}_i = s_i w^i + s_{\ell-i}/w^{\ell-i}$. Replacing i by $\ell - i$ gives the system $\begin{pmatrix} 1 & w^{-\ell} \\ w^{-\ell} & 1 \end{pmatrix} \begin{pmatrix} s_i \\ s_{\ell-i} \end{pmatrix} = \begin{pmatrix} \tilde{s}_i/w^i \\ \tilde{s}_{\ell-i}/w^{\ell-i} \end{pmatrix}$. There is a unique solution when $w \neq 0$ and the matrix is invertible.

Input: RLPs $Q(X) = q_0 + \sum_{j=1}^{d_q} q_j(X^j + X^{-j})$ of degree at most $2d_q$,
and $R(X) = r_0 + \sum_{j=1}^{d_r} r_j(X^j + X^{-j})$ of degree at most $2d_r$,
both in standard basis. A convolution length $\ell > d_q + d_r$
Output: RLP $S(X) = s_0 + \sum_{j=1}^{d_s} s_j(X^j + X^{-j}) = Q(X)R(X)$ of degree at most $2d_s = 2d_q + 2d_r$ in standard basis. Output may overlap input
Auxiliary storage: NTT arrays M and M', each with ℓ elements per p_j.
A squaring may use the same array for M and M'

Zero M and M'
For each NTT prime p_j
    Choose w_j with $w_j^\ell \not\equiv 0, \pm 1 \pmod{p_j}$
    Set $M_{j,0} := q_0 \bmod p_j$ and $M'_{j,0} := r_0 \bmod p_j$
For $1 \le i \le d_q$ (any order)
    For each p_j
        Set $M_{j,i} := w_j^i q_i \bmod p_j$ and $M_{j,\ell-i} := w_j^{-i} q_i \bmod p_j$
Do similarly with R and M'
For each p_j
    Perform forward NTTs of length ℓ modulo p_j on $M_{j,*}$ and $M'_{j,*}$.
    Multiply elementwise $M_{j,*} := M_{j,*} M'_{j,*}$ and perform inverse NTT on $M_{j,*}$
    For $1 \le i \le \ell - d_s - 1$ set $M_{j,i} := w_j^{-i} M_{j,i} \pmod{p_j}$
    For $\ell - d_s \le i \le \lfloor \ell/2 \rfloor$
        Set $\begin{pmatrix} M_{j,i} \\ M_{j,\ell-i} \end{pmatrix} := \begin{pmatrix} 1 & -w^{-\ell} \\ -w^{-\ell} & 1 \end{pmatrix} \begin{pmatrix} w^{-i} M_{j,i}/(1 - w^{-2\ell}) \\ w^{i-\ell} M_{j,\ell-i}/(1 - w^{-2\ell}) \end{pmatrix} \bmod p_j$
For $0 \le i \le d_s$ perform CRT on $M_{*,i}$ residues to obtain s_i, store in output

Fig. 1. NTT-Based Multiplication Algorithm for Reciprocal Laurent Polynomials

This leads to the algorithm in Figure 1. It flows like the interface in §6.1.

Our code chooses the NTT primes $p_j \equiv 1 \pmod{3\ell}$. We require $3 \nmid \ell$. Our w_j is a primitive cube root of unity. Multiplications by 1 are omitted. When $3 \nmid i$, we use $w_j^i q_i + w_j^{-i} q_i \equiv -q_i \pmod{p_j}$ to save a multiply.

Substituting $X = e^{i\theta}$ where $i^2 = -1$ gives

$$Q(e^{i\theta})R(e^{i\theta}) = \left(q_0 + 2\sum_{j=1}^{d_q} \cos j\theta \right) \left(r_0 + 2\sum_{j=1}^{d_r} \cos j\theta \right) .$$

These cosine series can be multiplied using discrete cosine transforms, in approximately the same auxiliary space needed by the weighted convolutions. We did not implement that approach.

6.3 Multiplying General Polynomials by RLPs

In section 8 we will construct an RLP $h(X)$ which will later be multiplied by various $g(X)$. The length-ℓ DFT of $h(X)$ evaluates $h(\omega^i)$ for $0 \le i < \ell$. However since $h(X)$ is reciprocal, $h(\omega^i) = h(\omega^{\ell-i})$ and the DFT has only $\ell/2 + 1$ distinct coefficients. In signal processing, the DFT of a signal extended symmetrically

around the center of each endpoint is called a Discrete Cosine Transform of type I. Using a DCT–I algorithm [2], we could compute the coefficients $h(\omega^i)$ for $0 \le i \le \ell/2$ with a length $\ell/2+1$ transform. We have not implemented this.

Instead we compute the full DFT of the RLP (using $X^\ell = 1$ to avoid negative exponents). To conserve memory, we store only the $\ell/2+1$ distinct DFT output coefficients for later use.

7 Computing Coefficients of f

Assume the P+1 algorithm. The monic RLP $f(X)$ in (5), with roots α_1^{2k} where $k \in S_1$, can be constructed using the decomposition of S_1. The coefficients of f will always be in the base ring since $P_1 \in \mathbb{Z}/N\mathbb{Z}$.

For the P–1 algorithm, set $\alpha_1 = b_1$ and $P_1 = b_1 + b_1^{-1}$. The rest of the construction of f for P–1 is identical to that for P+1.

Assume S_1 and S_2 are built as in §5, say $S_1 = T_1 + T_2 + \cdots + T_m$ where each T_j has an arithmetic progression of prime length, centered at zero. At least one of these has even cardinality since $s_1 = |S_1| = \prod_j |T_j|$ is even. Renumber the T_j so $|T_1| = 2$ and $|T_2| \ge |T_3| \ge \cdots \ge |T_m|$.

If $T_1 = \{-k_1, k_1\}$, then initialize $F_1(X) = X + X^{-1} - \alpha_1^{2k_1} - \alpha_1^{-2k_1} = X + X^{-1} - V_{2k_1}(P_1)$, a monic RLP in X of degree 2.

Suppose $1 \le j < m$. Given the coefficients of the monic RLP $F_j(X)$ with roots $\alpha_1^{2k_1}$ for $k_1 \in T_1 + \cdots + T_j$, we want to construct

$$F_{j+1}(X) = \prod_{k_2 \in T_{j+1}} F_j(\alpha_1^{2k_2} X). \tag{10}$$

The set T_{j+1} is assumed to be an arithmetic progression of prime length $t = |T_{j+1}|$ centered at zero with common difference k, say $T_{j+1} = \{(-1-t)k/2 + ik : 1 \le i \le t\}$. If t is even, k is even to ensure integer elements. On the right of (10), group pairs $\pm k_2$ when $k_2 \ne 0$. We need the coefficients of

$$F_{j+1}(X) = \begin{cases} F_j(\alpha_1^{-k} X)\, F_j(\alpha_1^{k} X), & \text{if } t = 2; \\ F_j(X) \prod_{i=1}^{(t-1)/2} \big(F_j(\alpha_1^{2ki} X)\, F_j(\alpha_1^{-2ki} X)\big), & \text{if } t \text{ is odd.} \end{cases}$$

Let $d = \deg(F_j)$, an even number. The monic input F_j has $d/2$ coefficients in $\mathbb{Z}/N\mathbb{Z}$ (plus the leading 1). The output F_{j+1} will have $td/2 = \deg(F_{j+1})/2$ such coefficients.

Products such as $F_j(\alpha_1^{2ki} X)\, F_j(\alpha_1^{-2ki} X)$ can be formed by the method in §7.1, using d coefficients to store each product. The interface can pass $\alpha_1^{2ki} + \alpha_1^{-2ki} = V_{2ki}(P_1) \in \mathbb{Z}/N\mathbb{Z}$ as a parameter instead of $\alpha_1^{\pm 2ki}$.

For odd t, the algorithm in §7.1 forms $(t-1)/2$ such monic products each with d output coefficients. We still need to multiply by the input F_j. Overall we store $(d/2) + \frac{t-1}{2} d = td/2$ coefficients. Later these $(t+1)/2$ monic RLPs can be multiplied in pairs, with products overwriting the inputs, until F_{j+1} (with $td/2$ coefficients plus the leading 1) is ready.

All polynomial products needed for (10), including those in §7.1, have output degree at most $t\deg(F_j) = \deg(F_{j+1})$, which divides the final $\deg(F_m) = s_1$. The polynomial coefficients are saved in the (MZNZ) buffer of §9. The (MDFT) buffer allows convolution length $\ell_{\max}/2$, which is adequate when an RLP product has degree up to $2(\ell_{\max}/2)-1 \geq s_1$. A smaller length might be better for a particular product.

7.1 Scaling by a Power and its Inverse.

Let $F(X)$ be a monic RLP of even degree d, say $F(X) = c_0 + \sum_{i=1}^{d/2} c_i(X^i + X^{-i})$, where each $c_i \in \mathbb{Z}/N\mathbb{Z}$ and $c_{d/2} = 1$. Given $Q \in \mathbb{Z}/N\mathbb{Z}$, where $Q = \gamma + \gamma^{-1}$ for some unknown γ, we want the d coefficients (excluding the leading 1) of $F(\gamma X)\, F(\gamma^{-1}X)$ mod N in place of the $d/2$ such coefficients of F. We are allowed a few scalar temporaries and any storage internal to the polynomial multiplier.

Denote $Y = X + X^{-1}$. Rewrite, while pretending to know γ,

$$\begin{aligned} F(\gamma X) &= c_0 + \sum_{i=1}^{d/2} c_i(\gamma^i X^i + \gamma^{-i} X^{-i}) \\ &= c_0 + \sum_{i=1}^{d/2} \frac{c_i}{2}\Big((\gamma^i + \gamma^{-i})(X^i + X^{-i}) + (\gamma^i - \gamma^{-i})(X^i - X^{-i})\Big) \\ &= c_0 + \sum_{i=1}^{d/2} \frac{c_i}{2}\Big(V_i(Q)V_i(Y) + (\gamma - \gamma^{-1})U_i(Q)(X - X^{-1})U_i(Y)\Big). \end{aligned}$$

Replace γ by γ^{-1} and multiply to get

$$\begin{aligned} F(\gamma X)\, F(\gamma^{-1}X) &= G^2 - (\gamma - \gamma^{-1})^2(X - X^{-1})^2\, H^2 \\ &= G^2 - (Q^2 - 4)(X - X^{-1})^2\, H^2, \end{aligned} \tag{11}$$

where

$$G = c_0 + \sum_{i=1}^{d/2} c_i \frac{V_i(Q)}{2} V_i(Y), \qquad H = \sum_{i=1}^{d/2} c_i \frac{U_i(Q)}{2} U_i(Y).$$

This G is a (not necessarily monic) RLP of degree at most d in the standard basis, with coefficients in $\mathbb{Z}/N\mathbb{Z}$. This H is another RLP, of degree at most $d-2$, but using the basis $\{U_i(Y) : 1 \leq i \leq d/2\}$. Starting with the coefficient of $U_{d/2}(Y)$, we can repeatedly use $U_{j+1}(Y) = V_j(Y)U_1(Y) + U_{j-1}(Y) = V_j(Y) + U_{j-1}(Y)$ for $j > 0$, along with $U_1(Y) = 1$ and $U_0(Y) = 0$, to convert H to standard basis. This conversion costs $O(d)$ additions in $\mathbb{Z}/N\mathbb{Z}$.

Use (3) and (2) to evaluate $V_i(Q)/2$ and $U_i(Q)/2$ for consecutive i as you evaluate the $d/2+1$ coefficients of G and the $d/2$ coefficients of H. Using the memory model in §9, and the algorithm in Figure 1, write the NTT images of the standard-basis coefficients of G and H to different parts of (MDFT). Later retrieve the $d-1$ coefficients of H^2 and the $d+1$ coefficients of G^2 as you finish the (11) computation. Discard the leading 1.

8 Multipoint Polynomial Evaluation

We have constructed $f = F_m$ in (5). The monic RLP $f(X)$ has degree s_1, say $f(X) = f_0 + \sum_{j=1}^{s_1/2} f_j \cdot (X^j + X^{-j}) = \sum_{j=-s_1/2}^{s_1/2} f_j X^j$ where $f_j = f_{-j} \in \mathbb{Z}/N\mathbb{Z}$.

Assuming the P–1 method (otherwise see §8.1), compute $r = b_1^P \in \mathbb{Z}/N\mathbb{Z}$. Set $\ell = \ell_{\max}$ and $M = \ell - 1 - s_1/2$.

Equation (6) needs $\gcd(f(X), N)$ where $X = b_1^{2k_2+(2m+1)P}$, for several consecutive m, say $m_1 \le m < m_2$. By setting $x_0 = b_1^{2k_2+(2m_1+1)P}$, the arguments to f become $x_0 b_1^{2mP} = x_0 r^{2m}$ for $0 \le m < m_2 - m_1$. The points of evaluation form a geometric progression with ratio r^2. We can evaluate these for $0 \le m < \ell - 1 - s_1$ with one convolution of length ℓ and $O(\ell)$ setup cost [1, exercise 8.27].

To be precise, set $h_j = r^{-j^2} f_j$ for $-s_1/2 \le j \le s_1/2$. Then $h_j = h_{-j}$. Set $h(X) = \sum_{j=-s_1/2}^{s_1/2} h_j X^j$, an RLP. The construction of h does not reference x_0 — we reuse h as x_0 varies.

Let $g_i = x_0^{M-i} r^{(M-i)^2}$ for $0 \le i \le \ell - 1$ and $g(X) = \sum_{i=0}^{\ell-1} g_i X^i$.

All nonzero coefficients in $g(X)h(X)$ have exponents from $0 - s_1/2$ to $(\ell - 1) + s_1/2$. Suppose $0 \le m \le \ell - 1 - s_1$. Then $M - m - \ell = -1 - s_1/2 - m < -s_1/2$ whereas $M - m + \ell = (\ell - 1 + s_1/2) + (\ell - s_1 - m) > \ell - 1 + s_1/2$. The coefficient of X^{M-m} in $g(X)h(X)$, reduced modulo $X^\ell - 1$, is

$$\sum_{\substack{0 \le i \le \ell-1 \\ -s_1/2 \le j \le s_1/2 \\ i+j \equiv M-m \pmod{\ell}}} g_i h_j \quad = \sum_{\substack{0 \le i \le \ell-1 \\ -s_1/2 \le j \le s_1/2 \\ i+j = M-m}} g_i h_j = \sum_{j=-s_1/2}^{s_1/2} g_{M-m-j} h_j$$

$$= \sum_{j=-s_1/2}^{s_1/2} x_0^{m+j} r^{(m+j)^2} r^{-j^2} f_j = \sum_{j=-s_1/2}^{s_1/2} x_0^m r^{m^2} \left(x_0 r^{2m}\right)^j f_j = x_0^m r^{m^2} f(x_0 r^{2m}).$$

Since we want only $\gcd(f(x_0\, r^{2m}), N)$, the $x_0^m\, r^{m^2}$ factors are harmless.

We can compute successive $g_{\ell-i}$ with two ring multiplications each since the ratios $g_{\ell-1-i}/g_{\ell-i} = x_0\, r^{2i-s_1-1}$ form a geometric progression.

8.1 Adaptation for P+1 Algorithm

If we replace b_1 with α_1, then r becomes α_1^P, which satisfies $r + r^{-1} = V_P(P_1)$. The above algebra evaluates f at powers of α_1. However α_1, r, h_j, x_0, and g_i lie in an extension ring.

Arithmetic in the extension ring can use a basis $\{1, \sqrt{\Delta}\}$ where $\Delta = P_1^2 - 4$. The element α_1 maps to $(P_1 + \sqrt{\Delta})/2$. A product $(c_0 + c_1\sqrt{\Delta})(d_0 + d_1\sqrt{\Delta})$ where c_0, c_1, d_0, $d_1 \in \mathbb{Z}/N\mathbb{Z}$ can be done using four base-ring multiplications: $c_0 d_0$, $c_1 d_1$, $(c_0 + c_1)(d_0 + d_1)$, $c_1 d_1 \Delta$, plus five base-ring additions.

We define linear transformations E_1, E_2 on $(\mathbb{Z}/N\mathbb{Z})[\sqrt{\Delta}]$ so that $E_1(c_0 + c_1\sqrt{\Delta}) = c_0$ and $E_2(c_0 + c_1\sqrt{\Delta}) = c_1$ for all c_0, $c_1 \in \mathbb{Z}/N\mathbb{Z}$. Extend E_1 and E_2 to polynomials by applying them to each coefficient.

To compute r^{n^2} for successive n, we use recurrences. We observe

$$r^{n^2} = r^{(n-1)^2+2} \cdot V_{2n-3}(r + r^{-1}) - r^{(n-2)^2+2},$$
$$r^{n^2+2} = r^{(n-1)^2+2} \cdot V_{2n-1}(r + r^{-1}) - r^{(n-2)^2} \ .$$

After initializing the variables $\mathtt{r1}[i] := r^{i^2}$, $\mathtt{r2}[i] := r^{i^2+2}$, $\mathtt{v}[i] := V_{2i+1}(r + r^{-1})$ for two consecutive i, we can compute $\mathtt{r1}[i] = r^{i^2}$ for larger i in sequence by

$$\begin{aligned} \mathtt{r1}[i] &:= \mathtt{r2}[i-1] \cdot \mathtt{v}[i-2] - \mathtt{r2}[i-2], \\ \mathtt{r2}[i] &:= \mathtt{r2}[i-1] \cdot \mathtt{v}[i-1] - \mathtt{r1}[i-2], \\ \mathtt{v}[i] &:= \mathtt{v}[i-1] \cdot V_2(r + 1/r) - \mathtt{v}[i-2] \ . \end{aligned} \tag{12}$$

Since we won't use $\mathtt{v}[i-2]$ and $\mathtt{r2}[i-2]$ again, we can overwrite them with $\mathtt{v}[i]$ and $\mathtt{r2}[i]$. For the computation of r^{-n^2} where r has norm 1, we can use r^{-1} as input, by taking the conjugate.

All $\mathtt{v}[i]$ are in the base ring but $\mathtt{r1}[i]$ and $\mathtt{r2}[i]$ are in the extension ring. Each application of (12) takes five base-ring multiplications (compared to two multiplications per r^{n^2} in the P–1 algorithm).

We can compute successive $g_i = x_0^{M-i} r^{(M-i)^2}$ similarly. One solution to (12) is $\mathtt{r1}[i] = g_i$, $\mathtt{r2}[i] = r^2 g_i$, $\mathtt{v}[i] = x_0 r^{2M-2i-1} + x_0^{-1} r^{1+2i-2M}$. Again each $\mathtt{v}[i]$ is in the base ring, so (12) needs only five base-ring multiplications.

If we try to follow this approach for the multipoint evaluation, we need twice as much space for an element of $(\mathbb{Z}/N\mathbb{Z})[\sqrt{\Delta}]$ as one of $\mathbb{Z}/N\mathbb{Z}$. We also need a convolution routine for the extension ring.

If p divides the coefficient of X^{M-m} in $g(X)h(X)$, then p divides both coordinates thereof. The coefficients of $g(X)h(X)$ occasionally lie in the base ring, making $E_2(g(X)h(X))$ a poor choice for the gcd with N. Instead we compute

$$E_1(g(X)h(X)) = E_1(g(X))E_1(h(X)) + \Delta E_2(g(X))E_2(h(X)) \ . \tag{13}$$

The RLPs $E_1(h(X))$ and $E_2(\Delta h(X))$ can be computed once and for each the $\ell_{\max}/2 + 1$ distinct coefficients of its length-$\ell_{\max}$ DFT saved in (MHDFT). To compute $E_2(\Delta h(X))$, multiply $E_2(\mathtt{r1}[i])$ and $E_2(\mathtt{r2}[i])$ by Δ after initializing for two consecutive i. Then apply (12).

Later, as each g_i is computed we insert the NTT image of $E_2(g_i)$ into (MDFT) while saving $E_1(g_i)$ in (MZNZ) for later use. After forming $E_2(g(X))E_1(h(X))$, retrieve and save coefficients of X^{M-m} for $0 \le m \le \ell - 1 - s_1$. Store these in (MZNZ) while moving the entire saved $E_1(g_i)$ into the (now available) (MDFT) buffer. Form the $E_1(g(X))E_2(\Delta h(X))$ product and the sum in (13).

9 Memory Allocation Model

We aim to fit our major data into the following:

(MZNZ) An array with $s_1/2$ elements of $\mathbb{Z}/N\mathbb{Z}$, for convolution inputs and outputs. This is used during polynomial construction. This is not needed during P–1 evaluation. During P+1 evaluation, it grows to $\ell_{\max}$ elements of $\mathbb{Z}/N\mathbb{Z}$.

(MDFT) An NTT array holding $\ell_{\max}$ values modulo each prime p_j, for use during DWTs.
Section 7.1 does two overlapping squarings, whereas §7 multiplies two arbitrary RLPs. Each product degree is at most $\deg(f) = s_1$. The algorithm in Figure 1 needs $\ell \geq s_1/2$ and might use convolution length $\ell = \ell_{\max}/2$, assuming $\ell_{\max}$ is even. Two arrays of this length fit in (MDFT).
After f has been constructed, (MDFT) is used for NTT transforms with length up to $\ell_{\max}$.

(MHDFT) Section 8 scales the coefficients of f by powers of r to build h. Then it builds and stores a length-ℓ DFT of h, where $\ell = \ell_{\max}$. This transform output normally needs ℓ elements per p_j for P–1 and 2ℓ elements per p_j for P+1. The symmetry of h lets us cut these needs almost in half, to $\ell/2+1$ elements for P–1 and $\ell + 2$ elements for P+1.

During the construction of F_{j+1} from F_j, if we need to multiply pairs of monic RLPs occupying adjacent locations within (MZNZ) (without the leading 1's), we use (MDFT) and the algorithm in Figure 1. The outputs overwrite the inputs within (MZNZ).

During polynomial evaluation for P–1, we need only (MHDFT) and (MDFT). Send the NTT image of each g_i coefficient to (MDFT) as g_i is computed. When (MDFT) fills (with $\ell_{\max}$ entries), do a length-$\ell_{\max}$ forward DFT on (MDFT), pointwise multiply by the saved DFT output from h in (MHDFT), and do an inverse DFT in (MDFT). Retrieve each needed polynomial coefficient, compute their product, and take a GCD with N.

9.1 Potentially Large B_2

Table 1. Estimated memory usage (quadwords) while factoring 230-digit number.

Array name	Construct f. Both $P \pm 1$	Build h.		Evaluate f.	
(MZNZ)	$12(s_1/2)$	$12(s_1/2)$		0	(P–1)
				$12\ell_{\max}$	(P+1)
(MDFT)	$25\ell_{\max}$	$25\ell_{\max}$		$25\ell_{\max}$	
(MHDFT)	0	$25(\ell_{\max}/2+1)$	(P–1)	$25(\ell_{\max}/2+1)$	(P–1)
		$25(\ell_{\max}+2)$	(P+1)	$25(\ell_{\max}+2)$	(P+1)
Totals, if $s_1 = \ell_{\max}/2$	$28\ell_{\max} + O(1)$	$40.5\ell_{\max} + O(1)$	(P–1)	$37.5\ell_{\max} + O(1)$	(P–1)
		$53\ell_{\max} + O(1)$	(P+1)	$62\ell_{\max} + O(1)$	(P+1)

Nowadays (2008) a typical PC memory is 4 gigabytes. The median size of composite cofactors N in the Cunningham project **http://homes.cerias.purdue.edu/~ssw/cun/index.html** is about 230 decimal digits, which fits in

twelve 64-bit words (called *quadwords*). Table 1 estimates the memory requirements during stage 2, when factoring a 230-digit number, for both polynomial construction and polynomial evaluation phases, assuming convolutions use the NTT approach in §6.1. The product of our NTT prime moduli must be at least $\ell_{\max}(N-1)^2$. If $N^2\ell_{\max}$ is below $0.99 \cdot (2^{63})^{25} \approx 10^{474}$, then it will suffice to have 25 NTT primes, each 63 or 64 bits.

The P–1 polynomial construction phase uses an estimated $40.5\ell_{\max}$ quadwords, vs. $37.5\ell_{\max}$ quadwords during polynomial evaluation. We can reduce the overall maximum to $37.5\ell_{\max}$ by taking the (full) DFT transform of h in (MDFT), and releasing the (MZNZ) storage before allocating (MHDFT).

Four gigabytes is 537 million quadwords. A possible value is $\ell_{\max} = 2^{23}$, which needs 315 million quadwords. When transform length $3 \cdot 2^k$ is supported, we could use $\ell_{\max} = 3 \cdot 2^{22}$, which needs 472 million quadwords.

We might use $P = 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 = 111546435$, for which $\phi(P) = 36495360 = 2^{13} \cdot 3^4 \cdot 5 \cdot 11$. We choose $s_2 \mid \phi(P)$ so that s_2 is close to $\phi(P)/(\ell_{\max}/2) \approx 8.7$, i.e. $s_2 = 9$ and $s_1 = 4055040$, giving $s_1/\ell_{\max} \approx 0.48$.

We can do 9 convolutions, one for each $k_2 \in S_2$. We will be able to find $p \mid N$ if $b_1^q \equiv 1 \pmod{p}$ where q satisfies (7) with $m < \ell_{\max} - s_1 = 4333568$. As described in §5, the effective value of B_2 will be about $9.66 \cdot 10^{14}$.

Larger systems can search further in little more time.

10 Opportunities for Parallelization

Modern PC's are multi-core, typically with 2–4 CPUs (cores) and a shared memory. When running on such systems, it is desirable to utilize multiple cores.

While building $h(X)$ and $g(X)$ in §8, each core can process a contiguous block of subscripts. Use the explicit formulas to compute r^{-j^2} or g_i for the first two elements of a block, and the recurrences elsewhere.

If convolutions use NTT's and the number of processors divides the number of primes, then allocate the primes evenly across the processors. The (MDFT) and (MHDFT) buffers in §9 can have separate subbuffers for each prime. On NUMA architectures, the memory for each subbuffer should be allocated locally to the processor that will process it. Accesses to remote memory occur only when converting the h_j and g_i to residues modulo small primes, and when reconstructing the coefficients of $g(x)h(x)$ with the CRT.

11 Our Implementation

Our implementation is based on GMP-ECM, an implementation of P–1, P+1, and the Elliptic Curve Method for integer factorization. It uses the GMP library [5] for arbitrary precision arithmetic. The code for stage 1 of P–1 and P+1 is unchanged; the code for the new stage 2 has been written from scratch and will replace the previous implementation [13] which used product trees of cost $O\left(n(\log n)^2\right)$ modular multiplications for building polynomials of degree n and

a variant of Montgomery's POLYEVAL [9] algorithm for multipoint evaluation which has cost $O\left(n(\log n)^2\right)$ modular multiplications and $O(n \log n)$ memory. The practical limit for B_2 was about 10^{14} – 10^{15}.

GMP-ECM includes modular arithmetic routines, using e.g. Montgomery's REDC [6], or fast reduction modulo a number of the form $2^n \pm 1$. It also includes routines for polynomial arithmetic, in particular convolution products. One algorithm available for this purpose is a small prime NTT/CRT, using the "Explicit CRT" [3] variant which speed reduction modulo N after the CRT step but requires 2 or 3 additional small primes. Its current implementation allows only power-of-two transform lengths. Another is Kronecker-Schönhage's segmentation method [13], which is faster than the NTT if the modulus is large and the convolution length is comparatively small, and it works for any convolution length. Its main disadvantage is significantly higher memory use, reducing the possible convolution length.

On a 2.4 GHz Opteron with 8 GB memory, P–1 stage 2 on a 230-digit composite cofactor of $12^{254}+1$ with $B_2 = 1.2 \cdot 10^{15}$, using the NTT with 27 primes for the convolution, can use $P = 64579515$, $\ell_{\max} = 2^{24}$, $s_1 = 7434240$, $s_2 = 3$ and takes 1738 seconds while P+1 stage 2 takes 3356 seconds. Using multi-threading to use both cpus on the same machine, P–1 stage 2 with the same parameters takes 1753 seconds cpu and 941 seconds elapsed time while P+1 takes 3390 seconds cpu and 2323 seconds elapsed time. For comparison, the previous implementation of P–1 stage 2 in GMP-ECM [13] needs to use a polynomial $F(X)$ of degree 1013760 and 80 blocks for $B_2 = 10^{15}$ and takes 34080 seconds on one cpu of the same machine.

On a 2.6 GHz Opteron with 8 cores and 32 GB of memory, a multi-threaded P–1 stage 2 on the same input number with the same parameters takes 1661 seconds cpu and 269 seconds elapsed time, while P+1 takes 3409 seconds cpu and 642 seconds elapsed time. With $B_2 = 1.34 \cdot 10^{16}$, $P = 198843645$, $\ell_{\max} = 2^{26}$, $s_1 = 33177600$, $s_2 = 2$, P–1 stage 2 takes 5483 seconds cpu and 922 elapsed time while P+1 takes 10089 seconds cpu and 2192 seconds elapsed time.

12 Some Results

We ran at least one of $P \pm 1$ on over 1500 composite cofactors, including

(a) Richard Brent's tables with $b^n \pm 1$ factorizations for $13 \leq b \leq 99$;
(b) Fibonacci and Lucas numbers F_n and L_n with $n < 2000$, or $n < 10000$ and cofactor size $< 10^{300}$;
(c) Cunningham cofactors of $12^n \pm 1$ with $n < 300$;
(d) Cunningham cofactors c300 and larger.

The B_1 and B_2 values varied, with 10^{11} and 10^{16} being typical. Table 2 has new large prime factors p and the largest factors of the corresponding $p \pm 1$.

The 52-digit factor of $47^{146} + 1$ and the 60-digit factor of L_{2366} each set a new record for the P+1 factoring algorithm upon their discovery. The previous record was a 48-digit factor of L_{1849}, found by the second author in March 2003.

Table 2. Large $P \pm 1$ factors found

Input Method	Factor p found Largest factors of $p \pm 1$	Size
$73^{109} - 1$	76227040047863715568322367158695720006439518152299	c191
P–1	$12491 \cdot 37987 \cdot 156059 \cdot 2244509 \cdot 462832247372839$	p50
$68^{118} + 1$	75066863480377406210977101832004765805050737493250 89*	c151
P–1	$22807 \cdot 480587 \cdot 14334767 \cdot 89294369 \cdot 4649376803 \cdot 5380282339$	p52
$24^{142} + 1$	204890474274505790519896836864533701541268201046245 37	c183
P–1	$4959947 \cdot 7216081 \cdot 16915319 \cdot 17286223 \cdot 12750725834505143$	p53
$47^{146} + 1$	79864788660358229882201629788746313352749574950084 01	c235
P+1	$20540953 \cdot 56417663 \cdot 1231471331 \cdot 1632221953 \cdot 843497917739$	p52
L_{2366}	72551623773963590503713291617111603427921502614602177025052 3	c290
P+1	$932677 \cdot 62754121 \cdot 19882583417 \cdot 751245344783 \cdot 483576618980159$	p60

* = Found during stage 1

The 53-digit factor of $24^{142}+1$ has $q = 12750725834505143$, a 17-digit prime. To our knowledge, this is the largest prime in the group order associated with any factor found by the P–1, P+1 or Elliptic Curve methods of factorization.

The largest q reported in Table 2 of [8] is $q = 6496749983$ (10 digits), for a 19-digit factor p of $2^{895}+1$. That table includes a 34-digit factor of the Fibonacci number F_{575}, which was the P–1 record in 1989.

The largest P–1 factor reported in [13, pp. 538–539] is a 58-digit factor of $2^{2098} + 1$ with $q = 9909876848747$ (13 digits). Site **http://www.loria.fr/~zimmerma/records/Pminus1.html** has other records, including a 66-digit factor of $960^{119} - 1$ found by P–1 for which $q = 2110402817$ (only ten digits).

The first author ran stage 1 with $B_1 = 10^{11}$ for the p53 of $24^{142}+1$ in Table 2. It took 44 hours on a 2200 MHz AMD Athlon processor in 32-bit mode at CWI.

Table 3. Timing for stage 2 of $24^{142} + 1$ factorization

Operation	Minutes (per CPU)	Parameters
Compute f	22	$P = 198843645$
Compute h	2	$\ell_{\max} = 2^{26}$
Compute DCT–I(h)	8	$s_1 = 33177600$
Compute all g_i	6 (twice)	$s_2 = 1$
Compute $g * h$	17 (twice)	$m_1 = 246$
Test for non-trivial GCD	2 (twice)	
Total	$32 + 2 \cdot 25 = 82$	

Stage 2 was run by the second author on an 8-core, 32 Gb node of the Grid5000 network. Table 3 shows where the time went. The overall stage 2 time is $8 \cdot 82 = 656$ minutes, about 25% of the stage 1 CPU time.

13 Acknowledgements

We thank Paul Zimmermann for his advice and guidance.

We thank the reviewers for their comments.

We thank Centrum voor Wiskunde en Informatica (CWI, Amsterdam) and INRIA for providing huge amounts of computer time for this work.

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see `https://www.grid5000.fr`).

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Baszenski, G., Tasche, M.: Fast Polynomial Multiplication and Convolutions Related to the Discrete Cosine Transform. Linear Algebra and its Applications 252, 1–25 (1997)
3. Bernstein, D.J., Sorenson, J.P.: Modular Exponentiation via the Explicit Chinese Remainder Theorem. Math. Comp. 76, 443–454 (2007)
4. Crandall, R., Fagin, B.: Discrete Weighted Transforms and Large-Integer Arithmetic. Math. Comp. 62, 305–324 (1994)
5. Granlund, T.: GNU MP: The GNU Multiple Precision Arithmetic Library, `http://gmplib.org/`
6. Montgomery, P.L.: Modular Multiplication Without Trial Division. Math. Comp. 44, 519–521 (1985)
7. Montgomery, P.L: Speeding the Pollard and Elliptic Curve Methods of Factorization. Math. Comp. 48, 243–264 (1987)
8. Montgomery, P.L., Silverman, R.D.: An FFT Extension to the $P-1$ Factoring Algorithm. Math. Comp. 54, 839–854 (1990)
9. Montgomery, P.L.: An FFT Extension to the Elliptic Curve Method of Factorization. UCLA dissertation (1992), `ftp://ftp.cwi.nl/pub/pmontgom`
10. Nussbaumer, H.J.: Fast Fourier Transform and Convolution Algorithms. 2nd edition, Springer, Heidelberg (1982)
11. Pollard, J.M.: Theorems on factorization and primality testing. Proc. Cambridge Philosophical Society 76, 521–528 (1974)
12. Williams, H.C.: A $p+1$ Method of Factoring. Math. Comp. 39, 225–234 (1982)
13. Zimmermann, P., Dodson, B.: 20 Years of ECM. In: Hess, F., Pauli, S., Pohst, M. (eds.) Algorithmic Number Theory, 7th International Symposium, ANTS–VII, LNCS, vol. 4076, pp. 525–542. Springer, Heidelberg (2006)

A Responses to Reviewer Comments on ANTS-VIII paper 123

A.1 Reviewer 1

State-of-the-art implementation report for the p-1 and p+1 algorithms. Extensive discussion of details, with varying ranges of applicability outside factorization; somewhat stream-of-consciousness; reminded me of the Crandall-Pomerance book.

I was intrigued by the discussions of arithmetic on reciprocal polynomials, apparently speeding up the whole algorithm by nearly a factor of 2 at the cost of various interacting high-level and low-level complications. The implementation of this speedup is still in progress but is promised to be done in time for ANTS. I wonder whether a dual strategy, evaluating a random-looking polynomial at reciprocal points, would achieve similar speedups.

We did not consciously attempt to extend our results to non-factorization scenarios, such as discrete logarithms.

The end of section 6.2 of the November submission does mention that the DCT (discrete cosine transform) had not been implemented. We subsequently found and implemented a simpler, RLP multiplication algorithm that uses approximately the same time and storage. See the new §6.2.

As for the reviewer's speculation of a dual strategy, we invite him/her to investigate this.

A.2 Reviewer 2

This is an implementation report of some improved version of P–1 factoring methods. The results are quite impressive for these rather particular factoring methods and it appears that the main motivation for this revival of the P–1 methods is connected to the availability of large memory ressources, which are useful for the second stage computations.

The paper is technically correct and consistently written, the results are good.For a conference of the level of ANTS it would be important to have more transparent algorithmic analysis, and here the list of questions and remarks is remarkable. I will try to bring them in some order - which is not necessarily unique.

[1.] In the second paragraph of page 2, it is mentioned that the improvement of Montgomerry improve upon Pollard by a factor of two. Let us call this mehtod of Montgomerry the method A.

After this, the track of quantitative measures for improvements is simply lost. In paragraph four of the same page I read "like in [4], we evaluate a polynomial along geometric series ...". After reading the full paper I get the impression that the splitting in S_1, S_2 and the evaluations of the polynomial (X) involving more recent / adequate methods for multipoint evaluations and fourier transforms are the main improvements (see below some questions regarding these). The reader would like to know:

[a)] is it true that the ideas of the approach are not new, but use some better implementations?

The idea of using a convolution to evaluate a polynomial along a geometric progression is very old. Silverman's implemented evaluated one polynomial along one geometric progression. This work evaluates one polynomial along several geometric progressions. The way of choosing the polynomial and the geometric progressions is new.

Little has been written on this topic since [8]. In particular, the idea of using reciprocal polynomials is believed new. When we evaluate the polynomial at some α, we also evaluate it at $1/\alpha$. In §3, symmetry condition (b) ensures the roots of f occur in reciprocal pairs. For the P+1 method, these pairs are algebraic conjugates, and the coefficients of f are in the base ring.

The algorithms for multiplying and squaring reciprocal polynomials are believed new. The November, 2007 draft had an incomplete description of the DCT-II. Since then we discovered a method using a weighted convolution.

We have not seen recurrence (12) elsewhere.

[b)] even if this is so, one would like an estimate of the improvement factor with respect to method A. Is it a small constant, is it more?

We added remarks in 12 about the timing in an actual run. There are about 2.8e14 primes below 10^{16}. It was not practical to search that high with the old codes. We did not directly compare pairs of methods.

[c)] The analysis in section 9 is restricted to memory allocation. This leaves the impression that indeed the improvements go more in the direction of space use then run time. But is this really so, can one have explicite statements of the authors on the novel aspects of the algorithm?

The space improvements let us search further. They improve capacity (how far one can search in reasonable time).

Section 11 compares one-cpu timings with the old and beta GMP-ECM codes. $B_2 = 10^{15}$ was near the top of the feasible search range before this work. As this goes to press, we have run some cofactors to $B_2 = 10^{17}$. We hope to do some 10^{18} runs before the Banff conference.

[2.] One would like to have more clarity about the choice of the partial algorithms used. For this it is a good idea to refer to general literature. Fast fourier methods in all flavors have been studied, and in particular Shoup has done a good combination of implementation details and theoretical analysis in his papers of the 90'es and also in his recent book. It looks like the choices from Montgomerry's thesis are along the same line, but some neutral reference helps increase the trust in the optimality of the choice.

We added a reference to Nussbaumer's book.

To a lesser extent, the same question applies to the multipoint evaluations: the asymptotics of the methods used are the best in general but for points in arithmetic and geometric progressions O(n log (n)) can be achieved. Also optimal constants are known meanwhile - see for instance A. Bostan's thesis. Since there is no run time analysis, it is hard to estimate if a small improvement at this points may be relevant or not!

I guess A. Bostan is Alin Bostan, whose thesis appears to be
http://algo.inria.fr/bostan/these/These.pdf (in English).

Bostan shows that multipoint evaluation of a polynomial of degree $< n$ along a geometric progression of length n can be done in $M(n) + O(n)$ base field operations, where $M(n)$ is the time to multiply two polynomials both of degree $< n$. For P-1, our multi-point evaluation performs one cyclic convolution of length $2n$ in this notation, and pre-/postcomputation in $O(n)$, so its asymptotic complexity is equal to that of Bostan's method. For P+1, we need two convolutions but since the points of evaluation are in a quadratic extension of Z/NZ, it seems like this is the best we can manage.

[3.] A detailed revision of the paper should help the reader not only believe that the algorithms are consistent, but also follow gradually the ideas involved. For instance, in the second paragraph of Section 3 which is a presentation of the algorithm: "The algorithm chooses P with large P/phi(P) ...". So far we knew B_1, B_2, so one should understand here how P relates to these, how do the primes dividing it relate to the primes in the interval B_1, B_2, how large are the sets S_1, S_2, can one have some optimality proofs? I suspected that P is essentially split into the primes in the interval (B_1, B_2), but this is not said. I should mention that most of these questions are answered indirectly in section 5. It would help to add some basical answers in the overview section.

No claim of optimality is made.

The primes dividing P are typically small, below 40, far below B_1,

Sample values of parameters such as P appear in §9.1 and §12. There are more comments in the Overview (§3) about the how the parameters interrelate, including equation (4).

[4.] Last but not least, it is not clear if a generalization to ECM - with some major changes perhaps - is taken into consideration or is out of scope or hope?

No adaptation to ECM is imminent unless we can embed the additive elliptic curve's additive group into the multiplicative group in which computation is easy. [We can embed into a group ring, but machine computation in high-dimensional group rings is not feasible.] The polynomial arithmetic needs both additions and multiplications.

Present ECM codes manipulate a coordinate $x(P)$ rather than P itself, and there is no analogue of a geometric progression.

The revised introduction mentions that this paper's techniques do not apply to ECM.

I believe the paper deserves some serious rewritting for the final version.

Much of the text has changed.