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 use the discrete cosine transform to multiply reciprocal polynomials. We present adjustments for the P+1 algorithm. We list some new findings.
Key words. Integer factorization, convolution, discrete cosine transform, DCT–II, discrete Fourier 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 [**?**, section 4]. It hopes that some prime factor p of N has smooth $p-1$. It picks b_0 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 where 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

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

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 [**?**] tries 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 the GCD tests: if either $p \mid \gcd(x, N)$ 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 [**?**] 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 [**?**] suggests an FFT continuation to P–1. Silverman [**?**, p. 844] implements it, using a circular convolution to evaluate a polynomial along a geometric progression. Montgomery's dissertation [**?**] describes an FFT continuation to ECM. He takes the GCD of two polynomials. Zimmermann [**?**] implements another FFT continuation to ECM, based on evaluating a polynomial at arbitrary points. Zimmermann adapts his implementation to $P \pm 1$ methods.

Like [**?**], we evaluate a polynomial along geometric progressions. We exploit patterns in its roots to generate its coefficients quickly. 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 the discrete cosine transform.

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 (2007) PC memories are 100 times as large as that used in [**?**]. 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 $256^2 \approx 6.6e4$.

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

2 P+1 Algorithm

Hugh Williams [**?**] 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 $\gcd(P_1^2 - 4,\, N) > 1$, 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.

Let n be an integer. The Chebyshev polynomials V_n and U_n satisfy the formal identities

$$V_n(X + X^{-1}) = X^n + X^{-n},$$
$$(X - X^{-1})U_n(X + X^{-1}) = X^n - X^{-n}.$$

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 $[B_1, B_2]$.

It chooses an odd integer P with large $P/\phi(P)$, a convolution length $\ell_{\max}$ and a factorization $\phi(P) = s_1 s_2$ so that s_1 is even and close to $\ell_{\max}/2$, perhaps $0.3 \le s_1/\ell_{\max} \le 0.7$. We require $\ell_{\max} > s_1$. It will do s_2 convolutions of length $\ell_{\max}$; each evaluates a polynomial of degree s_1 on $\ell_{\max} - s_1$ points.

Using a factorization of $(\mathbb{Z}/P\mathbb{Z})^*$ as described in section **??**, 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

$$f(X) = X^{-s_1/2} \prod_{k_1 \in S_1} (X - b_1^{2k_1}) \bmod N \qquad (4)$$

by the method in section **??**. 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

$$f(b_1^{2k_2+(2m+1)P}) \bmod N \tag{5}$$

for $\ell_{\max} - s_1$ consecutive values of m as described in section **??**, and checks the product of these outputs for a nontrivial GCD with N.

For the P+1 method, replace (**??**) 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 (**??**). The polynomial f is still over $\mathbb{Z}/N\mathbb{Z}$, but the multipoint evaluation works in an extension ring. See section **??**.

4 Justification

Let p be an unknown prime factor of N. As in (**??**), 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,

$$q = P + 2k_1 + 2k_2 + 2mP = 2k_1 + 2k_2 + (2m + 1)P \tag{6}$$

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

$$f(b_1^{2k_2+(2m+1)P}) = f(b_1^{q-2k_1}) \equiv f(b_1^{-2k_1}) \equiv 0 \pmod p. \tag{7}$$

For the P+1 method, if $\alpha_1^q \equiv 1 \pmod p$, then (**??**) 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,

$$(\mathbb{Z}/(mn)\mathbb{Z})^* = n(\mathbb{Z}/m\mathbb{Z})^* + m(\mathbb{Z}/n\mathbb{Z})^* \text{ if } \gcd(m, n) = 1. \tag{8}$$

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 (**??**). 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 an 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 $p \neq 3$ and the set can be

factored into arithmetic progressions of prime length by $R_{nm} = R_n + nR_m$. 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 (**??**) 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.

For choosing a value of P which covers a desired $[B_1, B_2]$ interval, we can test candidate P from a table. This table could contain values so that P and $\phi(P)$ are increasing, and each P is maximal for its $\phi(P)$. We can select those P which, in order, cover the desired $[B_1, B_2]$ interval with the user-specified $\ell_{\max}$, minimize $s_2 \cdot \ell$ and maximize $(2m_2 + 1)P + 2\min(S_1 + S_2)$.

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 commutative ring with 1 and let ℓ be a positive integer. 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 the ring $R[X]$, returning $f_1(X)f_2(X) \bmod X^\ell - 1$.

When $\deg(f_1) + \deg(f_2) < \ell$, the convolution 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) [**?**, chapter 7]. Fix ω. A *forward DFT* evaluates all $f_1(\omega^i)$ for $0 \le i \le \ell - 1$. 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 [**?**, p. 534], we consider two ways to do the convolutions.

Montgomery [**?**, section 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 some 63-bit NTT primes p_j 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

ω_j modulo p_j 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 [**?**, chapter 8] describes these computations in detail.

The convolution code should supply functions to (1) zero a DFT buffer (or a DCT buffer – see section **??**), (2) add one residue modulo N to a DFT buffer at a time, i.e. reduce it modulo the NTT primes, (3) perform a forward DFT on a buffer, (4) compute a point-wise product of the DFT transforms in two buffers, overwriting an input, and performing the inverse DFT on the product, and (5) extracting residues modulo N from the convolution product by performing the CRT computation and reduction modulo N.

Another method for computing convolutions uses fast integer multiplication. See section **??**.

6.2 Reciprocal Laurent Polynomials and Discrete Cosine Transform

Define a *reciprocal Laurent polynomial* (RLP) in X to be an expansion $a_0 + \sum_{j=1}^{d} a_j V_j(X + X^{-1}) = a_0 + \sum_{j=1}^{d} a_j \cdot (X^j + X^{-j})$ 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 $\{1\} \cup \{X^j + X^{-j} : 1 \leq j \leq d\} = \{1\} \cup \{V_j(Y) : 1 \leq j \leq d\}$ where $Y = X + X^{-1}$.

Let ℓ be a positive integer. Let R be a ring in which 2ℓ is not a zero divisor Assume R has a primitive 4ℓ-th root ω_ℓ of unity for which $\omega_\ell^{2\ell} = -1$. Set $\gamma_\ell = \omega_\ell + \omega_\ell^{-1}$. Then $V_\ell(\gamma_\ell) = 0$. We omit the subscript ℓ when it is clear from context.

Let $Q(X) = q_0 + \sum_{j=1}^{\ell-1} q_j(X^j + X^{-j}) = q_0 + \sum_{j=1}^{\ell-1} q_j V_j(X + 1/X)$ be a RLP of degree at most $2\ell - 2$. Define its *discrete cosine transform* of length ℓ to be $\tilde{Q} = [\tilde{q}_1, \tilde{q}_3, \ldots, \tilde{q}_{2\ell-1}]$ where $\tilde{q}_k = Q(V_k(\gamma_\ell))$. This is also called a DCT–II.

The $\tilde{Q}$ vector has values of $Q(X)$ when X is an odd power of ω_ℓ. Those are the roots of V_ℓ. In the ring of RLPs, we have $Q_3(X) \equiv Q_1(X)Q_2(X) \pmod{X^\ell + X^{-\ell}}$ precisely when the $\tilde{Q}_3$ vector is the pointwise product of the $\tilde{Q}_1$ and $\tilde{Q}_2$ vectors.

This leads to a polynomial multiplication algorithm. Given RLPs Q_1 and Q_2 with $\deg(Q_1) \leq 2(d_1 - 1)$ and $\deg(Q_2) \leq 2(d_2 - 1)$, choose $\ell \geq d_1 + d_2 - 1$. Take length-ℓ DCTs of Q_1 and Q_2. Find an RLP Q_3 of degree at most $2(\ell-1)$ such that $\tilde{Q}_3$ is the pointwise product of $\tilde{Q}_1$ and $\tilde{Q}_2$. The error $Q_0(X) := Q_1(X)Q_2(X) - Q_3(X)$ has degree at most $\max(2d_1 + 2d_2 - 4, 2\ell - 2) < 2\ell$. But $Q_0(X)$ is a multiple of $X^\ell + X^{-\ell}$ so $Q_0(X) = 0$.

We need to transition between Q and $\tilde{Q}$. Steidl and Tasche [**?**, section 3] do this when the length ℓ is a power of 2, say $\deg(Q) < \ell = 2^r$, as well as for $\ell = 3 \cdot 2^r$. If n_1 and n_2 are nonnegative integers, let $n_1 \oplus n_2$ denote their bitwise exclusive OR. If n is an (up to) r-bit binary integer, let $\mathrm{bitrev}_r(n)$ denote its bit reversal. For an odd integer a, define $\mathrm{aloc}(a) = \mathrm{bitrev}_r(\lfloor a/4 \rfloor \oplus \lfloor a/2 \rfloor)$. Figure **??** sketches how to get $\tilde{Q}$ from Q.

Input: Array with 2^r ring elements.
  $Q(X)$ starts at index 0, standard basis.
Outputs:
  Evaluations $Q(V_a(\gamma)) = Q(X) \bmod (X - V_a(\gamma))$ at index aloc(a) for odd a.

for s **from** 1 **to** r **do**
  $n := 2^{r-s}$;
// Now array has the remainders $Q(X) \bmod (V_{2n}(X) - V_{2an}(\gamma))$
// at index aloc(a) for odd a with $1 \le a < 2^s$.
  for a **from** 1 **to** $2^s - 1$ **by** 2 **do**
    $b := a$; $c := 2^{s+1} - a$;
    $k :=$ aloc(a);
    Retrieve $t_a := Q(X) \bmod (V_{2n}(X) - V_{2an}(\gamma))$, which starts at index k.
    $t_b := t_a \bmod (V_n(X) - V_{bn}(\gamma))$;
    $t_c := t_a \bmod (V_n(X) - V_{cn}(\gamma))$;
    Store t_b and t_c, each length n, at indices k and $k + n$, respectively.
  end for
end for

Fig. 1. Forward DCT–II (without final permutation) of length 2^r

Within the t_b and t_c computations, we observe

$$V_{bn}(\gamma) + V_{cn}(\gamma) = V_{(bn+cn)/2}(\gamma)V_{(bn-cn)/2}(\gamma) = V_{2^r}(\gamma)V_{(bn-cn)/2}(\gamma) = 0.$$

The product of the polynomial moduli is $(V_n(X) - V_{bn}(\gamma))\ (V_n(X) - V_{cn}(\gamma)) = V_n(X)^2 - V_{an}(\gamma)^2 = V_{2n}(X) - V_{2an}(\gamma)$, which is the modulus used for t_a.

We can compute $V_{an}(\gamma)$ for successive a with one ring multiply each since a ranges over an arithmetic progression. Replacing $c_0 + \sum_{m=1}^{2n-1} c_m V_m(X)$ by its two remainders $(c_0 \pm c_n V_{an}(\gamma)) \cdot 1 + \sum_{m=1}^{n-1} (c_m - c_{2n-m} \pm c_{n+m} V_{an}(\gamma))\, V_m(X)$ takes n multiplications per a and can be done in-place. Each s needs about $2^{s-1}(1+n) = 2^{s-1} + 2^{r-1}$ multiplies. Many optimizations to the standard FFT apply, such as avoiding multiplies by ± 1.

The pseudocode for an inverse transform can reverse the steps, forming t_a from t_b and t_c. The formula $t_a := (t_b + t_c)/2 + (V_n(X)/V_{an}(\gamma))(t_b - t_c)/2$ divides by $V_{an}(\gamma)$. Instead assume

$$t_b = nU_{bn}(\gamma)(Q(X) \bmod (V_n(X) - V_{bn}(\gamma))),$$
$$t_c = nU_{cn}(\gamma)(Q(X) \bmod (V_n(X) - V_{cn}(\gamma))).$$

When $s = r$ (so $n = 1$), the code can scale input remainders by $U_a(\gamma)$ so this pattern holds when $n = 1$. Given these, the reverse inner loop computes $t_a := V_n(X)(t_c - t_b) + V_{cn}(\gamma)(t_b + t_c)$ to preserve the pattern. At the end, during the backward $s = 1$ loop, the only case is $n = 2^{r-s}$ and $a = 1$. The output $Q(X)$ is scaled by a factor of TBD.

As of November, 2007, the DCT has not been implemented. We plan to have it working before the February, 2008 ANTS deadline.

6.3 Multiplying General Polynomials by RLPs

In section **??** we will construct an RLP $h(X)$ which will later be multiplied by various $g(X)$. Normally we use the DCT-II to multiply two RLP's and the DFT to multiply two general polynomials.

A DCT-II of length $\ell/2$ and a DFT of length ℓ use different points of evaluation, so they cannot be combined for multiplication of a polynomial by an RLP.

A DCT-I of length $\ell/2+1$ can mix its outputs with those of a DFT, each evaluating a polynomial at ℓ-th roots of unity. We achieve this by computing a full DFT of the RLP (after using $X^\ell = 1$ to avoid negative exponents). To conserve memory, we store only the $\ell+1$ distinct DFT output coefficients. This exploits $h(\omega) = h(\omega^{-1})$ for all ℓ-th roots of unity ω.

In the scrambled output of a decimation-in-time FTT of length $\ell = 2^r$, the distinct DFT coefficients $h(\omega^i)$ for $0 \le i \le \ell/2$ are at even indices and index 1. We store only these. For $0 < 2i < \ell/2$, coefficients at index $2i$ and index $m_i - 2i$, where $m_i = 2^{\lfloor \log_2(i) \rfloor + 3} - 2^{\lfloor \log_2(i) \rfloor + 1} - 1$, correspond to $h(\omega^{\text{bitrev}_r(2i)})$ and $h(\omega^{\ell - \text{bitrev}_r(2i)})$ and thus are equal. For the pointwise product, we can multiply the FFT coefficients of g at index $2i$ and $m_i - 2i$ by the coefficient of h that was at index $2i$.

7 Computing Coefficients of f

Assume the P+1 algorithm. The monic RLP $f(X)$ in (**??**), 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 section **??**, 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

$$F_{j+1}(X) = \prod_{k_2 \in T_{j+1}} F_j(\alpha_1^{2k_2} X). \tag{9}$$

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

$$F_{j+1}(X) = \begin{cases} F_j(\alpha_1^{-2k} X)\, F_j(\alpha_1^{2k} X), & \text{if } t = 2; \\ F_j(X) \prod_{i=1}^{(t-1)/2} \big(F_j(\alpha_1^{4ki} X)\, F_j(\alpha_1^{-4ki} X)\big), & \text{if } t \text{ is odd.} \end{cases}$$

Let $d = \deg(F_j)$, an even number. The monic input F_j has $d/2$ coefficients in $\mathbb{Z}/N\mathbb{Z}$ (not counting the leading 1). The output F_{j+1} will have $td/2 = \deg(F_{j+1})/2$ such coefficients.

Products such as $F_j(\alpha_1^{4ki}X)\,F_j(\alpha_1^{-4ki}X)$ can be formed by the method in section **??**, using d coefficients to store each product. The interface can pass $\alpha_1^{4ki} + \alpha_1^{-4ki} = V_{4ki}(P_1) \in \mathbb{Z}/N\mathbb{Z}$ as a parameter instead of $\alpha_1^{\pm 4ki}$.

For odd t, the algorithm in section **??** 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 (**??**), including those in section **??**, have output degree at most $t\deg(F_j) = \deg(F_{j+1})$, which divides the final $\deg(F_m) = s_1$. If we use multiplications modulo $X^\ell - 1$, for some $\ell > \deg(F_{j+1})$, then the products will be exact. Since $\ell_{\max} > s_1 \geq \deg(F_{j+1})$ and $\ell_{\max}$ is a tolerable convolution length, we can always use $\ell = \ell_{\max}$, but a smaller ℓ 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 $F(\gamma X)\,F(\gamma^{-1}X) \bmod N$ 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

$$\begin{aligned} F(\gamma X)\,F(\gamma^{-1}X) &= G^2 - (\gamma - \gamma^{-1})^2(X - X^{-1})^2\,H^2 \\ &= G^2 - (Q^2 - 4)(X - X^{-1})^2\,H^2, \end{aligned} \tag{10}$$

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 $\{U_i(Y) : 1 \le i \le d/2\}$. 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 (**??**) and (**??**) 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 section **??**, write the standard-basis coefficients of G to one DCT buffer and those of H to the other. Take two forward DCTs, square both, and take the inverse DCTs. Retrieve the $d-1$ coefficients of H^2 and the $d+1$ coefficients of G^2 as you finish the (**??**) computation. Discard the leading 1.

8 Multipoint Polynomial Evaluation

We have constructed $f = F_m$ in (**??**). 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 section **??**), compute $r = b_1^P \in \mathbb{Z}/N\mathbb{Z}$. Set $\ell = \ell_{\max}$ and $M = \ell - 1 - s_1/2$.

Equation (**??**) 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 [**?**, 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_0d_0, c_1d_1, $(c_0 + c_1)(d_0 + d_1)$, $c_1d_1\Delta$, 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

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

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

$$\begin{aligned} \mathtt{r1}[i] &:= \mathtt{r2}[i-1] \cdot \mathtt{v}[i-2] - \mathtt{r2}[i-2], \\ \mathtt{r2}[i] &:= \mathtt{r2}[i-1] \cdot \mathtt{v}[i-1] - \mathtt{r1}[i-2], \\ \mathtt{v}[i] &:= \mathtt{v}[i-1] \cdot V_2(r + 1/r) - \mathtt{v}[i-2] \ . \end{aligned} \tag{11}$$

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 (**??**) 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 (**??**) 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 (**??**) 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

$$E_1(g(X)h(X)) = E_1(g(X))E_1(h(X)) + \Delta E_2(g(X))E_2(h(X)) \ .$$

The RLPs $E_1(h(X))$ and $E_2(\Delta h(X))$ can be computed once and their DCT transforms (convolution length $\ell_{\max}$) saved in the two DCT buffers. 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, before applying (**??**).

Later, as each g_i is computed we insert $E_2(g_i)$ into a DFT input buffer while saving $E_1(g_i)$ for later use. After forming $E_2(g(X))E_1(h(X))$, retrieve and save coefficients of X^{M-m} for $0 \le m \le \ell - 1 - s_1$. Insert saved $E_1(g_i)$ into the (now fresh) DFT input buffer. Form the $E_1(g(X))E_2(\Delta h(X))$ product and the sum.

Or we can compute one coordinate of g_i at a time, at the cost of computing each $\mathtt{v}[i]$ twice, by applying E_1 (or E_2) to each line in (**??**).

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 $\mathbb{Z}/N\mathbb{Z}$ (if we compute both coordinate of each g_i together, saving one of them), or $\ell_{\max} - s_1$ elements (if we compute the coordinates individually).

(MDCT1), (MDCT2) Two buffers, each holding $\ell_{\max}/2 + 1$ outputs from a DCT (convolution length $\ell_{\max}$). Each can instead hold a pointwise product. These are work areas during forward and inverse DCTs.
After the construction of h from f in section **??**, the length-$\ell_{\max}$ DFT of h (or of $E_1(h)$ and $E_2(\Delta h)$) is computed once and stored here (using only (MDCT1) for P–1 but both buffers for P+1). Then f and h are discarded. The DFT of h evaluates $h(\omega)$ for all $\ell_{\max}$-th roots of unity, but needs only $\lfloor \ell_{\max}/2 \rfloor + 1$ locations due to $h(X) = h(1/X)$.

(MDFT) A buffer holding $\ell_{\max}$ outputs from a DFT (convolution length $\ell_{\max}$). This can also hold a pointwise product. It is a work area during forward and inverse DFTs. This is used during polynomial evaluation, both $P \pm 1$.

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), then we can write one DCT input to each DCT buffer. After two forward DCTs, a pointwise product, and an inverse DCT, retrieve all wanted coefficients of the product, overwriting the inputs within (MZNZ).

During polynomial evaluation for P–1, we need only (MDCT1) and (MDFT). Send 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 DCT output from h in (MDCT1), and do an inverse DFT in (MDFT). Retrieve each polynomial, compute their product, and take a GCD with N.

9.1 Potentially Large B_2

Nowadays (2007) a typical PC memory is 4 gigabytes. The median size of composite cofactors N in the Cunningham project **http://homes.cerias.purdue.**

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

Array name	Construct f. Both $P \pm 1$	Evaluate f. P–1	Evaluate f. P+1
(MZNZ)	$12s_1/2$	0	$12\ell_{\max}$ (or $12(\ell_{\max} - s_1)$)
(MDCT1)	$25(1 + \ell_{\max}/2)$	$25(1 + \ell_{\max}/2)$	$25(1 + \ell_{\max}/2)$
(MDCT2)	$25(1 + \ell_{\max}/2)$	0	$25(1 + \ell_{\max}/2)$
(MDFT)	0	$25\ell_{\max}$	$25\ell_{\max}$
Totals, if $s_1 = \ell_{\max}/2$	$28\ell_{\max} + O(1)$	$37.5\ell_{\max} + O(1)$	$62\ell_{\max} + O(1)$ (or $56\ell_{\max} + O(1)$)

edu/~ssw/cun/index.html is about 230 decimal digits, which fits in twelve 64-bit words (called *quadwords*). Table **??** 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 section **??**. The product of our NTT prime moduli must be at least $\ell_{\max}(N-1)^2$. If N is below $0.99 \cdot (2^{63})^{25} \approx 10^{474}$, then it will suffice to have 25 NTT primes, each 63 bits.

The P–1 polynomial construction phase uses an estimated $28\ell_{\max}$ quadwords, vs. $37.5\ell_{\max}$ quadwords during polynomial evaluation. 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 (**??**) with $m < \ell_{\max} - s_1 = 4333568$. As described in section **??**, the effective value of B_2 will be about $9.66 \cdot 10^{14}$.

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 section **??**, 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 DCT and DFT buffers in section **??** 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 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 [**?**] 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 [**?**] 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 [**?**] or fast reduction modulo 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 (but without the DCT variant). Its current implementation allows only for power-of-two transform lengths. Another is Kronecker-Schönhage's segmentation method [**?**], 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 8GB memory, P–1 stage 2 on a 250-digit composite number with $B_2 = 1.2 \cdot 10^{15}$, using the NTT for the convolution, can use $P = 64579515$, $\ell_{\max} = 2^{24}$, $s_1 = 7434240$, $s_2 = 3$ and takes 46 minutes.

On the same machine, P+1 stage 2 on a 243-digit number with $B_2 = 10^{15}$ can use $P = 111546435$, $l = 2^{23}$, $s_1 = 3649536$, $s_2 = 10$ and takes 86 minutes.

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 $n < 300$.

The B_1 and B_2 values varied, with 10^{11} and 10^{15} being typical. Table **??** 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.

The 49-digit factor of $75^{128} + 1$ has $q = 3288338823576187$, a 16-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.

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

Input Method	Factor p found Largest factors of $p \pm 1$	Size
$68^{118}+1$	750668634803774062109771018320047658050507374932508 9*	c151
P–1	$22807 \cdot 480587 \cdot 14334767 \cdot 89294369 \cdot 4649376803 \cdot 5380282339$	p52
$73^{109}-1$	762270400478637155683223671586957200064395181522 99	c191
P–1	$12491 \cdot 37987 \cdot 156059 \cdot 2244509 \cdot 462832247372839$	p50
$75^{128}+1$	218063787707856565609056954981253627319444343552 1	c183
P–1	$37489 \cdot 19791649 \cdot 162960727 \cdot 3288338823576187$	p49
$47^{146}+1$	798647886603582298822016297887463133527495749500 8401	c235
P+1	$20540953 \cdot 56417663 \cdot 1231471331 \cdot 1632221953 \cdot 843497917739$	p52
L_{2366}	725516237739635905037132916171116034279215026146021770250523	c290
P+1	$932677 \cdot 62754121 \cdot 1988258341 \cdot 751245344783 \cdot 483576618980159$	p60

* = Found during stage 1

The largest q reported in Table 2 of [**?**] 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 [**?**, 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).

13 Acknowledgements

We thank Paul Zimmermann for his advice and guidance.

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

References

1. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.
2. Peter L. Montgomery, Modular Multiplication without Trial Division, *Math. Comp.*, v. **44**(1985), pp. 519–521
3. Peter L. Montgomery, Speeding the Pollard and Elliptic Curve Methods of Factorization, *Math. Comp.*, v. **48**(1987), pp. 243–264.
4. Peter L. Montgomery and Robert D. Silverman, An FFT Extension to the P–1 Factoring Algorithm, *Math. Comp.*, v. **54**(1990), pp. 839–854.
5. Peter L. Montgomery. An FFT Extension to the Elliptic Curve Method of Factorization. UCLA dissertation, 1992. Available at **ftp://ftp.cwi.nl/pub/pmontgom**.
6. J.M. Pollard, Theorems on factorization and primality testing, *Proc. Cambridge Philosophical Society*, volume 76(1974), pp. 521–528.

7. G. Steidl and M. Tasche, "A polynomial approach to fast algorithms for discrete Fourier-cosine and Fourier-sine transforms", *Math. Comp.*, v. **56**(1991), pp. 281–296.
8. H. C. Williams, A $p+1$ Method of Factoring, *Math. Comp.*, v. **39**(1982), pp. 225–234.
9. Paul Zimmermann and Bruce Dodson. 20 Years of ECM, In F. Hess, S. Pauli, M. Pohst (Eds.), *Algorithmic Number Theory*, 7th International Symposium, ANTS-VII volume 4076 of Lecture Notes in Computer Science, pp. 525–542. Springer-Verlag, 2006.