

# Improved Stage 2 to $P \pm 1$ Factoring Algorithms

Peter L. Montgomery<sup>1</sup> and Alexander Kruppa<sup>2</sup>

<sup>1</sup> Microsoft Research, One Microsoft Way, Redmond, WA USA. pmontgom@cwi.nl

<sup>2</sup> 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 (2007) PC's. Our implementation uses a modular variant of the discrete cosine transform to multiply two reciprocal polynomials. We present adjustments for the  $P + 1$  algorithm. We list some new findings.

**Keywords.** Integer factorization, convolution, discrete cosine transform, discrete Fourier transform,  $P + 1$ ,  $P - 1$ , multipoint polynomial evaluation, reciprocal polynomials.

## 1 Introduction

The  $P - 1$  algorithm for factoring an odd composite integer  $N$  was introduced by John Pollard in 1974 [9, section 4]. It hopes that some prime factor  $p$  of  $N$  has smooth  $p - 1$ . It picks  $b_0$  coprime to  $N$  and outputs  $b_1 = b_0^e \pmod N$  for some exponent  $e$ . This exponent might be the product of 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 instead hope that  $p - 1 = qn$  where  $n$  divides  $e$  and  $q$  is not too large. Then

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

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 [9] suggests trying each prime  $q$  in  $[B_1, B_2]$  individually. If  $q_1$  and  $q_2$  are successive primes, then look up  $b_1^{q_2 - q_1} \pmod N$  in a small table. Given  $b_1^{q_1} \pmod N$ , form  $b_1^{q_2} \pmod N$  and test  $\gcd(b_1^{q_2} - 1, N)$ . There is one modular multiplication and one GCD per candidate  $q$ . He also observes that one can combine the GCD tests. If either  $p \mid \gcd(x, N)$  or  $p \mid \gcd(y, N)$ , then  $p \mid \gcd(xy \pmod N, N)$ . The cost of stage 2 drops to two modular multiplications per  $q$ , one GCD with  $N$  at the end, and a few multiplications to build the table.

Montgomery [6] 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 [9] suggests an FFT continuation to  $P - 1$ . Silverman [7, p. 844] was first to implement one, using a circular convolution to evaluate a polynomial along a geometric progression.

Montgomery's dissertation [8] describes an FFT continuation to ECM. He takes the GCD of two polynomials. Zimmermann [11] implements another FFT continuation to ECM, based on evaluating a polynomial at arbitrary points. Zimmermann adapts his implementation to handle  $P \pm 1$  methods too.

Like [7], our implementation evaluates a polynomial  $f$  along geometric progressions. We take advantage of patterns in the roots of  $f$  to speed generation of its coefficients. 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  $\ell$ . Many intermediate results are reciprocal polynomials, which need about half the storage and can be multiplied using the discrete cosine transform.

Doubling  $\ell$  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 8.17408 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 [7]. With this extra memory, we achieve  $\ell = 3 \cdot 2^{22}$  a growth factor of 384. With the same number of convolutions (individually longer lengths but running on faster hardware) our  $B_2$  advances by  $384^2 \approx 1.5e5$ .

The largest  $q$  reported in Table 2 of [7] 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 [11, 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).

We give some timing data and list some new results.

## 2 $P + 1$ algorithm

Hugh Williams [10] 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  $\alpha_0$  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  $\alpha_0^p$ . 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 of this presentation holds for the  $P + 1$  method, but some computations take place in an extension ring, raising memory needs by 50% if we use the same convolution size.

## 2.1 Chebyshev polynomials

Although the theory behind  $P + 1$  mentions  $\alpha_0$  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  $\alpha_0^n$  and  $\alpha_1^n$ .

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

$$\begin{aligned} V_n(X + X^{-1}) &= X^n + X^{-n}, \\ (X - X^{-1})U_n(X + X^{-1}) &= X^n - X^{-n}. \end{aligned}$$

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  $\alpha_0$  and  $\alpha_1$  are not.

The Chebyshev polynomials satisfy many identities, including

$$\begin{aligned} V_{mn}(X) &= V_m(V_n(X)), \\ U_{m+n}(X) &= U_m(X) V_n(X) - U_{m-n}(X), \end{aligned} \tag{2}$$

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

## 3 Overview of stage 2 algorithm

Our algorithm uses convolutions, as described in section 6. Early in stage 2, select a maximum convolution length  $\ell_{\max}$ , perhaps based on user-specified parameters and available memory. This  $\ell_{\max}$  might be a power of 2.

Also choose an odd integer  $P$  with large  $P/\phi(P)$ . This  $P$  might be the product of the first several odd primes.

Choose a factorization  $\phi(P) = s_1 s_2$  where

- (a)  $s_1$  is even.
- (b)  $s_1$  is close to  $\ell_{\max}/2$ . This might mean  $0.3 \leq s_1/\ell_{\max} \leq 0.7$ . Require  $\ell_{\max} > s_1$ .

- (c) You can afford a coefficient array with  $s_1/2$  elements of  $\mathbb{Z}/N\mathbb{Z}$ .
- (d) You are willing to do at least  $\lceil s_2/2 \rceil$  convolutions of length  $\ell_{\max}$ .

Then construct two sets  $S_1$  and  $S_2$  of integers such that

- (a)  $|S_1| = s_1$  and  $|S_2| = s_2$ .
- (b)  $S_1$  and  $S_2$  are symmetric around 0. If  $k \in S_1$  (resp.  $k \in S_2$ ), then  $-k \in S_1$  (resp.  $-k \in S_2$ ).
- (c)  $0 \notin S_1$ . This is assured since  $s_1 = |S_1|$  is even.
- (d) 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, compute the coefficients of

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

This  $f(X)$  is symmetric in  $X$  and  $1/X$ .

If  $k_2 \in S_2$  and  $k_2 \geq 0$ , evaluate (the numerators of) all

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

for  $\ell_{\max} - s_1$  (or more) consecutive values of  $m$ , and check each output for a nontrivial GCD with  $N$ . When  $k_2 = 0$ , use only positive values of  $2m + 1$ ; when  $k_2 > 0$ , use negative  $2m + 1$  as well.

For the  $P + 1$  method, replace (4) by  $f(X) = X^{-s_1/2} \prod_{k_1 \in S_1} (X - \alpha_1^{2k_1}) \pmod{N}$ . Similarly, replace  $b_1$  by  $\alpha_1$  in (5).

## 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,

$$q = P + 2k_1 + 2k_2 + 2mP = 2k_1 + 2k_2 + (2m + 1)P \quad (6)$$

for some integer  $m$ . We can bound  $m$  knowing bounds on  $q$ ,  $k_1$ ,  $k_2$ . Both  $b_1^{\pm 2k_1}$  are roots of  $f$ . These satisfy

$$b_1^{-2k_1} = b_1^{-q+2k_2+(2m+1)P} \equiv b_1^{2k_2+(2m+1)P} \pmod{p}, \quad (7)$$

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

When  $k_2 > 0$ , or  $k_2 = 0$  and  $2m + 1 > 0$ , line (7) has a point of evaluation of  $f$  for finding  $p$  if  $m$  is within the search limits. Line (8) has a point of evaluation of  $f$  for finding  $p$  when  $k_2 < 0$ , or  $k_2 = 0$  and  $2m + 1 < 0$ .

## 5 Selection of $S_1$ and $S_2$

By the Chinese Remainder Theorem,

$$(\mathbb{Z}/(mn)\mathbb{Z})^* = n(\mathbb{Z}/m\mathbb{Z})^* + m(\mathbb{Z}/n\mathbb{Z})^* \text{ if } m \perp n, \quad (9)$$

where “+” denotes the set of sums. This is independent of choice of 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). More choices are available by factoring individual  $(\mathbb{Z}/p\mathbb{Z})^*$ ,  $p \mid P$ , into smaller sets of sums.

Let  $R_n = \{2i - n - 1 : 1 \leq i \leq n\}$  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 size 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  $F(b_1^{2k_2+(2m+1)P})$  for all  $m_1 \leq m \leq m_2$  and  $k_2 \in S_2$ , the highest exponent coprime to  $P$  not covered at the low end of the stage 2 range will be  $2 \max(S_1) + 2 \max(S_2) + (2(m_1 - 1) + 1)P = 2 \max(S_1 + S_2) + (2m_1 - 1)P$ . Similarly, the smallest value at the high end of the stage 2 range not covered is  $2 \min(S_1 + S_2) + (2m_2 + 3)P$ . Hence, for a given choice of  $P$ ,  $S_1$ ,  $S_2$ ,  $m_1$  and  $m_2$ , we can be certain that all primes in  $[(2m_1 - 1)P + 2 \max(S_1 + S_2) + 1, (2m_2 + 3)P + 2 \min(S_1 + S_2) - 1]$  are covered.

For choosing a value of  $P$  which covers a desired  $[B_1, B_2]$  interval, we can test candidate  $P$  values from a table. This table should contain values so that  $P$  and  $\phi(P)$  are increasing, and each  $P$  is maximal for its  $\phi(P)$ . We can select those  $P$  values which, in order, cover the desired  $[B_1, B_2]$  interval with the user-specified  $\ell_{\max}$ , minimize  $s_2$ , minimize  $\ell_{\max}$  and maximize  $(2m_2 + 3)P + 2 \min(S_1 + S_2)$ .

For example, to cover the interval  $[1000, 500000]$  with  $\ell_{\max} = 512$ , we might choose  $P = 1155$ ,  $s_1 = 240$ ,  $s_2 = 2$ ,  $m_1 = -1$ ,  $m_2 = 270$ . 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  $[732, 631360]$ .

## 6 Circular convolutions and polynomial multiplication

Let  $R$  be a commutative ring with 1 and let  $\ell$  be a positive integer. A *circular convolution* of length  $\ell$  over  $R$  multiplies two polynomials  $f_1(X)$  and  $f_2(X)$  of degree at most  $\ell - 1$  in the ring  $R[X]$ , returning  $f_1(X)f_2(X) \pmod{X^\ell - 1}$ .

When  $\deg(f_1) + \deg(f_2) < \ell$ , the convolution gives an exact product.

If  $R$  has a primitive  $\ell$ -th root  $\omega$  of unity, and if  $\ell$  is not a zero divisor in  $R$ , then one convolution algorithm uses the Discrete Fourier Transform (DFT) [1,

chapter 7]. Fix  $\omega$ . A *forward DFT* evaluates all  $f_1(\omega^i)$  for  $0 \leq i \leq \ell - 1$ . Another forward DFT evaluates all  $\ell$  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  $\ell$  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. The FFT algorithm is heavily used when  $R = \mathbb{C}$ , along with  $\omega = \exp(-2\pi i/\ell)$ .

## 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  $\omega$ . As in [11, p. 534], we consider two ways to do the convolutions.

Montgomery [7, 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 DFT primes  $p_j$  such that  $\prod_j p_j > \ell(N - 1)^2$ . Require each  $p_j \equiv 1 \pmod{\ell}$ , so a primitive  $\ell$ -th root of unity  $\omega_j$  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 [8, chapter 8] describes these computations in detail.

The other uses fast integer multiplication. See section 10.

**IS THIS ORGANIZATION ACCURATE?** The convolution codes are organized so individual stages can be invoked as the application is ready, not necessarily all back-to-back. The stages include:

- (\*) Zero a DFT input buffer.
- (\*) Insert a coefficient into a DFT input buffer. For the NTT, this step reduces the input modulo each NTT prime and stores the remainders. This stage may be repeated.
- (\*) Perform a forward DFT on a DFT buffer (after all nonzero coefficients have been inserted).
- (\*) Pointwise multiply two DFT output buffers, storing the results in a third (which may overlap an input). Perform inverse DFT on the product buffer.
- (\*) Retrieve a coefficient from a DFT output buffer. For the NTT, this means CRT reconstruction and reduction modulo  $N$ . This stage may be repeated.

Wherever we see "DFT" above, there should be similar functionality for the discrete cosine transform in the next section.

## 6.2 Reciprocal polynomials and the discrete cosine transform

Define a *reciprocal Laurent polynomial* 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 said to have degree  $2d$  if  $a_d \neq 0$ . This degree is always even. It is *monic* if  $a_d = 1$ .

We can store a monic reciprocal Laurent polynomial of degree  $2d$  with  $d$  coefficients (excluding the leading 1).

While manipulating reciprocal Laurent polynomials 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  $Y = X + X^{-1}$ .

Fix an even convolution length  $\ell$ . A reciprocal Laurent polynomial  $f$  satisfies  $f(1/X) = f(X)$ . The DFT (when modified to accept Laurent inputs) evaluates both  $f(\omega)$  and  $f(\omega^{-1})$  for each  $\ell$ -th root  $\omega$ , but these values are the same. This length- $\ell$  forward DFT has only  $\ell/2 + 1$  distinct outputs. The discrete cosine transform (DCT-I), with  $\ell/2 + 1$  inputs and  $\ell/2 + 1$  outputs, streamlines these computations. When multiplying two reciprocal Laurent polynomials, the pointwise multiplier and inverse transform simplify too.

More formally, fix an even convolution length  $\ell$  and a primitive  $\ell$ -th root of unity  $\omega$ . If  $\mathbf{a} = [a_0, a_1, \dots, a_{\ell/2}]$ , let  $\text{DCT } 1_\ell(\mathbf{a}) = [b_0, b_1, \dots, b_{\ell/2}]$  where

$$b_k = a_0 + \sum_{j=1}^{\ell/2-1} (\omega^{jk} + \omega^{-jk})a_j + a_{\ell/2}(-1)^k \quad (0 \leq k \leq \ell/2).$$

If we define  $A(X) = a_0 + \sum_{j=1}^{\ell/2-1} a_j(X^j + X^{\ell-j}) + a_{\ell/2}X^{\ell/2}$ , then  $b_k = A(\omega^k) = A(\omega^{-k})$  for all  $k$ . That is,

$$\text{DCT } 1_\ell(\mathbf{a}) = [A(1), A(\omega), A(\omega^2), \dots, A(\omega^{\ell/2})].$$

Suppose we iterate the  $\text{DCT } 1_\ell$ . Evaluate

$$\begin{aligned} B(X) &= b_0 + \sum_{j=1}^{\ell/2-1} (X^j + X^{\ell-j})b_j + b_{\ell/2}X^{\ell/2} \\ &= A(1) + \sum_{j=1}^{\ell/2-1} (X^j + X^{\ell-j})A(\omega^j) + A(-1)X^{\ell/2} \end{aligned}$$

at  $X = \omega^i$ , getting

$$B(\omega^i) = A(1) + \sum_{j=1}^{\ell/2-1} (\omega^{ij} + \omega^{-ij})A(\omega^j) + A(\omega^{\ell/2})\omega^{-i\ell/2} = \sum_{j=-\ell/2}^{\ell/2-1} \omega^{-ij}A(\omega^j),$$

which is  $\ell$  times the coefficient of  $X^i$  in  $A(X)$ . That is,  $\text{DCT } 1_\ell(\text{DCT } 1_\ell(\mathbf{a})) = \ell\mathbf{a}$ . Hence the inverse DCT is the same as the forward DCT, except for a factor  $1/\ell$ .

A modular variant of the DCT lets  $\omega$  be a primitive  $(2N)$ -th root of unity (same  $\omega$  as used in the DFT). Replace  $2 \cos j\pi/N$  by  $V_j(\omega + \omega^{-1})$  and  $N$  by  $\ell$  throughout the equations.

Suppose  $f_1(X)$  and  $f_2(X)$  are reciprocal Laurent polynomials of degrees at most  $2d_1$  and  $2d_2$ , where  $d_1 + d_2 < \ell_{\max}/2$ . To multiply these, choose an even

convolution length  $\ell$  where  $2d_1 + 2d_2 < \ell \leq \ell_{\max}$ , and a primitive  $\ell$ -th root  $\omega$ . Using an inverse DFT (same  $\omega$ ), we can find a polynomial  $f_3$  such that

$$f_3(\omega^k) = f_3(\omega^{-k}) = f_1(\omega^k)f_2(\omega^k) \quad (0 \leq k \leq \ell/2).$$

This  $f_3$  is determined modulo  $X^\ell - 1$ . Reduce the exponents within  $f_3$  modulo  $\ell$ , so they are in the interval  $[-d_1 - d_2, \ell - 1 - d_1 - d_2]$ . Then

$$X^{d_1+d_2}(f_1(X)f_2(X) - f_3(X))$$

is a polynomial of degree at most  $\max(2d_1 + 2d_2, \ell - 1) = \ell - 1$ . This polynomial vanishes modulo  $X^\ell - 1$ , and must be the zero polynomial. That is,  $f_3(X) = f_1(X)f_2(X)$ .

The inverse DCT does this computation, starting after the pointwise product. The memory model in section 9 has two DCT buffers each holding  $\ell_{\max}/2 + 1$  elements. Use one DCT buffer for  $f_1(\omega^k)$  evaluations and the other for  $f_2(\omega^k)$  computations. Write the pointwise product to one buffer, and do the inverse DCT there.

As of October 2007, our DCT implementation is incomplete. Rather, we are using a scheme which multiplies arbitrary polynomials and is based on fast integer multiplication. See section 10,

## 7 Computing coefficients of $f$

Assume the  $P + 1$  algorithm. The monic reciprocal Laurent polynomial  $f(X)$  in (4), with roots  $\alpha_1^{2k}$  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 5, say  $S_1 = T_1 + T_2 + \dots + T_m$  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  $|T_2| \geq |T_3| \geq \dots \geq |T_m|$ .

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 reciprocal Laurent polynomial in  $X$  of degree 2.

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

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

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  $T_{j+1} = \{(-1 - t + 2i)k : 1 \leq i \leq t\}$ . On the right of (10), group pairs  $\pm k_2$  when  $k_2 \neq 0$ . We need the coefficients of

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

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), and 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 7.1, 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 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 reciprocal Laurent polynomials can be multiplied in pairs, with products overwriting the inputs, until  $F_{j+1}$  (with  $td/2$  coefficients plus the leading 1) is ready.

For large  $t$ , we could expand the right of (10) using  $O(\log t)$  polynomial multiplications (rather than  $O(t)$ ), but this gains little when  $t \leq 11$  and the larger  $T_j$  are processed early (i.e., while  $\deg(F_j)$  is small).

All polynomial products needed for (10), including those in section 7.1, 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  $\ell$  might be better for a particular product.

## 7.1 Scaling by a power and its inverse.

Let  $F(X)$  be a monic reciprocal Laurent polynomial 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  $\gamma$ , 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  $\gamma$ ,

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

Replace  $\gamma$  by  $\gamma^{-1}$  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

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

This  $G$  is a reciprocal (not necessarily monic) Laurent polynomial of degree at most  $d$  in the standard basis, with coefficients in  $\mathbb{Z}/N\mathbb{Z}$ . This  $H$  is another reciprocal Laurent polynomial, 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 section 9, 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 (11) computation. Discard the leading 1 from the result.

## 8 Multipoint polynomial evaluation

We have constructed  $f = F_m$  in (4). The monic reciprocal Laurent polynomial  $f(X)$  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 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 (5) needs  $\gcd(f(X), N)$  where  $X = b_1^{2k_2+(2m+1)P}$ , for several consecutive  $m$ , say  $m_1 \leq m < m_2$ . 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  $0 \leq m < m_2 - m_1$ . The points of evaluation form a geometric progression with ratio  $r^2$ . We can evaluate these for  $0 \leq m < \ell - 1 - s_1$  with one convolution of length  $\ell$  and  $O(\ell)$  setup cost [1, exercise 8.27].

To be precise, set  $h_j = r^{-j^2} f_j$  for  $-s_1/2 \leq j \leq s_1/2$ . Then  $h_j = h_{-j}$ . Set  $h(X) = \sum_{j=-s_1/2}^{s_1/2} h_j X^j$ , a reciprocal Laurent polynomial. 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  $0 \leq i \leq \ell - 1$  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  $0 \leq m \leq \ell - 1 - s_1$ . 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

$$\begin{aligned} \sum_{\substack{0 \leq i \leq \ell-1 \\ -s_1/2 \leq j \leq s_1/2 \\ i+j \equiv M-m \pmod{\ell}}} g_i h_j &= \sum_{\substack{0 \leq i \leq \ell-1 \\ -s_1/2 \leq j \leq s_1/2 \\ i+j = M-m}} g_i h_j = \sum_{j=-s_1/2}^{s_1/2} g_{M-m-j} h_j \\ &= \sum_{j=-s_1/2}^{s_1/2} x_0^{m+j} r^{(m+j)^2} r^{-j^2} f_j = \sum_{j=-s_1/2}^{s_1/2} x_0^m r^{m^2} (x_0 r^{2m})^j f_j = x_0^m r^{m^2} f(x_0 r^{2m}). \end{aligned}$$

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  $\alpha_1$ , then  $r$  becomes  $\alpha_1^P$ , which satisfies  $r + r^{-1} = V_P(P_1)$ . The above algebra evaluates  $f$  at powers of  $\alpha_1$ . However  $\alpha_1$ ,  $r$ ,  $h_j$ ,  $x_0$ , 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  $\alpha_1$  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.

Some multiplication involves powers of  $\alpha_1$  and  $r$ . These have norm 1, which may allow simplifications. For example,

$$(c_0 + c_1\sqrt{\Delta})^2 = 2c_0^2 - 1 + 2c_0c_1\sqrt{\Delta}$$

needs only two multiplications and three additions if  $c_0^2 - c_1^2\Delta = 1$ .

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

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

Since we won't use  $\mathbf{v}[i-2]$  and  $\mathbf{r2}[i-2]$  again, we can overwrite them with  $\mathbf{v}[i]$  and  $\mathbf{r2}[i]$ .

For the computation of  $r^{-n^2}$  where  $r$  has norm 1, we can use  $r^{-1}$  as input, by taking the conjugate.

All  $\mathbf{v}[i]$  are in the base ring but  $\mathbf{r1}[i]$  and  $\mathbf{r2}[i]$  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  $\mathbf{r1}[i] = g_i$ ,  $\mathbf{r2}[i] = r^2 g_i$ ,  $\mathbf{v}[i] = x_0 r^{2M-2i-1} + x_0^{-1} r^{1+2i-2M}$ . Again each  $\mathbf{v}[i]$  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.

To lift these needs, assume that the original  $P_0^2 - 4$  is a quadratic non-residue modulo a prime  $p$  we are searching for. Then

$$\begin{aligned} \Delta &= P_1^2 - 4 = (\alpha_1 - \alpha_1^{-1})^2 \equiv (\alpha_0^e - \alpha_0^{-e})^2 \\ &= U_e(P_0)^2 (\alpha_0 - \alpha_0^{-1})^2 = U_e(P_0)^2 (P_0^2 - 4) \pmod{N} \end{aligned}$$

is another quadratic non-residue modulo  $p$  since we assume  $p \mid N$  and  $\gcd(P_1^2 - 4, N) = 1$ .

Define  $h(X)$  and  $g(X)$  as in the  $P - 1$  case, with  $b_1$  replaced by  $\alpha_1$ . Write them in the form

$$\begin{aligned} h(X) &= h^{(0)}(X) + h^{(1)}(X)\sqrt{\Delta}, \\ g(X) &= g^{(0)}(X) + g^{(1)}(X)\sqrt{\Delta}, \end{aligned}$$

where  $h^{(0)}(X)$ ,  $h^{(1)}(X)$ ,  $g^{(0)}(X)$ ,  $g^{(1)}(X) \in (\mathbb{Z}/N\mathbb{Z})[X]$ .

If  $p$  divides the coefficient of  $X^{M-m}$  in

$$\begin{aligned} g(X)h(X) &= \left(g^{(0)}(X) + g^{(1)}(X)\sqrt{\Delta}\right) \left(h^{(0)}(X) + h^{(1)}(X)\sqrt{\Delta}\right) \\ &= \left(g^{(0)}(X)h^{(0)}(X) + g^{(1)}(X)h^{(1)}(X)\Delta\right) \\ &\quad + \left(g^{(0)}(X)h^{(1)}(X) + g^{(1)}(X)h^{(0)}(X)\right) \sqrt{\Delta}, \end{aligned}$$

then  $p$  must divide the coefficient of  $X^{M-m}$  in both coordinates

$$\begin{aligned} &g^{(0)}(X)h^{(0)}(X) + g^{(1)}(X)h^{(1)}(X)\Delta, \\ &g^{(0)}(X)h^{(1)}(X) + g^{(1)}(X)h^{(0)}(X). \end{aligned}$$

We test the second of these coordinates of a coefficient in of  $g(X)h(X)$  instead of testing the coefficient in  $g(X)h(X)$  itself [7, p. 852].

The reciprocal Laurent polynomials  $h^{(0)}(X)$  and  $h^{(1)}(X)$  can be computed once and their DCT transforms (convolution length  $\ell_{\max}$ ) saved in the two DCT buffers. We can adapt (12) to compute one coordinate of  $\mathbf{r1}[i]$  and the same coordinate of  $\mathbf{r2}[i]$ , along with  $\mathbf{v}[i]$ , in three base-ring multiplications.

As each  $g_i$  is computed (in the extension ring), we can insert coefficients of  $g^{(1)}(X)$  into a DFT input buffer while corresponding coefficients of  $g^{(0)}(X)$  are saved for later use. After forming  $g^{(1)}(X)h^{(0)}(X)$ , retrieve and save coefficients

of  $X^{M-m}$  for  $0 \leq m \leq \ell - 1 - s_1$ . Insert saved  $g^{(0)}(X)$  coefficients into the (now fresh) DFT input buffer. Form the  $g^{(0)}(X)h^{(1)}(X)$  product and the sum.

Or we can compute process one coordinate of  $g_i$  at a time, at the cost of computing each  $v[i]$  twice.

## 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), or  $\ell_{\max} - s_1$  elements (if we compute the coordinates separately).
- (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 8, the DCT of  $h$  is computed once and stored here (using only (MDCT1) for  $P - 1$  but both buffers for  $P + 1$ ). Then  $f$  and  $h$  are discarded.
- (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  $P \pm 1$ .

Another potentially big array is a table with powers of the roots of unity  $\omega_j$  used by the DFT. We might restrict our convolution lengths to divisors of  $\ell_{\max}$ . Then, rather than one table with all  $\ell_{\max}$  powers of an  $\omega_j$ , use two tables with  $O(\sqrt{\ell_{\max}})$  entries each. Multiply two table entries to get any power we need. Using smaller tables may improve cache performance too. We need only DCT products during polynomial construction, and only DFT products during polynomial evaluation, so the tables of constants can overlap. **The  $P + 1$  algorithm may need to store both  $U_i$  and  $V_i$  for selected  $i$ .**

During the construction of  $F_{j+1}$  from  $F_j$ , if we need to multiply pairs of monic reciprocal Laurent polynomials 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  $\langle g_i \rangle$  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 evaluation and take GCDs with  $N$ .

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 $56\ell_{\max} + O(1)$ )

MAKE A SIMILAR TABLE BASED UPON INTEGER MULTIPLICATION RATHER THAN NTT'S.

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

### 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.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 section 6.1. The product of our DFT prime moduli must be at least  $\ell_{\max}(N - 1)^2$ . If this is below  $0.99 \cdot (2^{63})^{25} \approx 10^{474}$ , then it will suffice to have 25 DFT 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  $\ell_{\max} = 3 \cdot 2^{22}$ , 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  $\phi(P) = 36495360 = 2^{13} \cdot 3^4 \cdot 5 \cdot 11$ . Choose  $s_2$  such that  $s_2 \mid \phi(P)$  and  $s_2$  is close to  $\phi(P)/(\ell_{\max}/2) \approx 5.8$ . We can use  $s_2 = 6$  and  $s_1 = 6082560$ .

This  $s_2$  is even, so  $S_2$  has three positive elements and three negative elements. Suppose we do  $c = 30$  convolutions overall (after building  $h$ ), ten for each  $k \in S_2$  with  $k > 0$  (both positive and negative  $2m + 1$ ). We will be able to find  $p \mid N$  if  $b_1^q \equiv 1 \pmod{p}$  where  $q$  satisfies (6) with  $|m| < \ell_{\max} - s_1 = 6500352$ . The effective value of  $B_2$  is about

$$\begin{aligned}
 B_2 &\approx \frac{c}{s_2} (2P) (\ell_{\max} - s_1) = c \frac{2P}{\phi(P)} s_1 (\ell_{\max} - s_1) \\
 &\approx c \frac{2P}{\phi(P)} \frac{\ell_{\max}^2}{4} \approx 30 \cdot 6.1 \cdot 40e12 \approx 7e15.
 \end{aligned}$$

HOW LONG WILL THIS TAKE?

Stage 1 to  $B_1 = 5 \cdot 10^{10}$  takes about 40 hours (Pentium, c205), low memory. Dividing by 2400, this advances  $B_1$  by 20 million each minute.

Paul Zimmermann (9 Feb 2007) reports 167.5 seconds per convolution when  $\ell_{\max} = 3 \cdot 2^{22}$  and 107.3 seconds per convolution when  $\ell_{\max} = 2^{23}$ , for 12-quadword operands, using the integer multiply FFT on an Opteron with 8 Gb. This (64-bit) processor advances  $B_1$  by 67 million each minute.

Using  $P - 1$  on the 1187-digit cofactor of the Fermat number  $2^{4096} + 1$ , we would need about  $189\ell_{\max}$  quadwords during polynomial evaluations. We might try convolution length  $2^{21}$  (another choice is  $5 \cdot 2^{19}$ ). With the above  $P$ , we want  $s_2 \approx \phi(P)/(\ell_{\max}/2) \approx 34.8$ , say  $s_2 = 36$  and  $s_1 = 1013760$ . With  $c = 36/2 = 18$  convolutions each of length  $\ell_{\max} = 2^{21}$ , we search all  $m$  with  $|m| < (\ell_{\max} - 1013760)/2 \approx 542000$ . This corresponds to  $B_2 \approx 542000(2P) \approx 1.2 \cdot 10^{14}$ . Previous efforts <http://www.mersenne.org/ecmf.htm> have done this through  $B_1 = 2.4e10$  and  $B_2 = 5e14$ .

Alex reports (14 Feb 2007) that the pre-2007 GMP-ECM  $P - 1$  stage 2 for  $2^{4096} + 1$  takes exactly 2 hours with  $B_2 = 25e12$  on a 2.4 GHz Opteron. Memory use was  $\approx 5$ GB (not quite sure) with -treefile. He estimates we could reach  $B_2 = 10^{15}$  in about 3 days.

## 10 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 [11] which used product trees of cost  $O(n \log n)$  modular multiplications for building polynomials of degree  $n$  and a variant of Montgomery's POLYEVAL [8] algorithm for multipoint evaluation which has cost  $O(n(\log n)^2)$  modular multiplications and  $O(n \log n)$  memory. The practical limit for  $B_2$  was about  $10^{14} - 10^{15}$ .

GMP-ECM includes routines for modular arithmetic using the functions for computing remainders in GMP, or using Montgomery's REDC [5] algorithm, either implemented on GMP's low level arithmetic functions or as low-overhead routines coded in assembly for x86 and x86-64 architectures. It also includes routines for polynomial arithmetic, in particular convolution products. One algorithm already available for this purpose is a small prime NTT/CRT (but without the DCT variant), another is Kronecker-Schönhage's segmentation method. The latter converts a polynomial multiplication into a multiplication of huge integers, which can use GMP's fast arithmetic functions. This is advantageous if the modulus is large and the degree is comparatively small since in this case the time for splitting and recombining the coefficients modulo small primes dominates in the NTT/CRT.

**Re. checkpointing: We'll test for a factor after each multipoint evaluation so nothing really needs to be stored. We can recompute  $f(X)$  easily. We should allow the user to skip the first couple of elements of  $s_2$ , so in case of an aborted run, he can compute only the still missing multipoint evaluations.**

## 11 Asymptotic analysis

## 12 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 8, each core can process a contiguous block of subscripts. Use the explicit formulas to compute  $r^{-j^2}$  or  $g_i$  at the start 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 9 can have separate subbuffers for each prime.

## 13 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 and  $13 \leq b \leq 99$ ;
- (b) Some Fibonacci and Lucas numbers;
- (c) Cunningham cofactors of  $12^n \pm 1$  with  $n < 300$ .

The  $B_1$  values varied, with  $10^{11}$  being typical. Table 2 (resp. 3) has new factors  $p$  of 50 digits or more found by  $P - 1$  (resp  $P + 1$ ). The second line of each entry has the largest factors of  $p - 1$  (resp.  $p + 1$ ). The 52-digit factor of  $68^{118} + 1$  was found during stage 1.

**Table 2.** Large  $P - 1$  factors found

Input	Factor $p$ found Largest factors of $p - 1$
$68^{118} + 1$	7506686348037740621097710183200476580505073749325089 22807 · 480587 · 14334767 · 89294369 · 4649376803 · 5380282339

**Table 3.** Large  $P + 1$  factors found

Input	Factor $p$ found Largest factors of $p + 1$
$47^{146} + 1$	7986478866035822988220162978874631335274957495008401 20540953 · 56417663 · 1231471331 · 1632221953 · 843497917739

## 14 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. Günter Baszenski and Manfred Tasche, Fast polynomial multiplication and convolutions related to the discrete cosine transform. *Linear Algebra and its Applications*. Volume 252, Issues 1–3, February, 1997, pp. 1–25.
3. Pierrick Gaudry, Alexander Kruppa and Paul Zimmermann, A GMP-Based Implementation of Schönhage-Strassen’s Large Integer Multiplication Algorithm, *ISSAC 2007*, ACM, pp. 167–174
4. J.G. Liu, Y.Z. Liu, and G.Y. Wang, Fast DCT-I, DCT-III, and DCT-IV via Moments, *EURASIP Journal on Applied Signal Processing* 2005:12, 1902–1909.
5. Peter L. Montgomery, Modular Multiplication without Trial Division, *Math. Comp.*, v. **44**(1985), pp. 519–521
6. Peter L. Montgomery, Speeding the Pollard and Elliptic Curve Methods of Factorization, *Math. Comp.*, v. **48**(1987), pp. 243–264.
7. Peter L. Montgomery and Robert D. Silverman, An FFT Extension to the  $P - 1$  Factoring Algorithm, *Math. Comp.*, v. **54**(1990), pp. 839–854.
8. Peter L. Montgomery. An FFT Extension to the Elliptic Curve Method of Factorization. UCLA dissertation, 1992. Available at <ftp://ftp.cwi.nl/pub/pmontgom>.
9. John M. Pollard, Theorems on Factorization and Primality Testing, *Proc. Cambridge Philosophical Society*, volume 76(1974), pp. 521–528.
10. H. C. Williams, A  $p + 1$  Method of Factoring, *Math. Comp.*, v. **39**(1982), pp. 225–234.
11. 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.