An FFT Extension to the $P - 1$ Factoring Algorithm

Peter L. Montgomery; Robert D. Silverman

Mathematics of Computation, Volume 54, Issue 190 (Apr., 1990), 839-854.

Stable URL:
http://links.jstor.org/sici?sici=0025-5718%28199004%2954%3A190%3C839%3AAFETTF%3E2.0.CO%3B2-Y

MATHEMATICS OF COMPUTATION
VOLUME 54, NUMBER 190
APRIL 1990, PAGES 839-854

AN FFT EXTENSION TO THE $P-1$ FACTORING ALGORITHM

PETER L. MONTGOMERY AND ROBERT D. SILVERMAN

ABSTRACT. J. M. Pollard, in 1974, presented the $P-1$ integer factoring algorithm. His paper couched the algorithm in theoretical terms based upon use of Fast Fourier Transform techniques, but he was unable to say whether the method could be made practical. We discuss the mathematical basis of the algorithm and show how it can work in practice. The practical implementation depends, for its success, upon the use of Residue Number Systems. We also present an open problem as to how the method could be made to work for the Elliptic Curve factoring algorithm.

1. INTRODUCTION

Let N be an odd composite integer whose factorization is sought. A key problem in one variant of the second phase of the $P-1$ and Elliptic Curve (ECM) factoring algorithms [11] is constructing two sequences

$$x_0, x_1, \cdots, x_{m-1},$$
$$y_0, y_1, \cdots, y_{n-1}$$

over $\mathbb{Z}/N\mathbb{Z}$ and checking whether $\mathrm{GCD}(x_i - y_j, N) > 1$ for some i and j. For example, in $P-1$ the x_i and y_j might represent selected powers of an element $H \bmod N$ [11, p. 251]. In ECM the x_i and y_j might represent the x-coordinates of selected points on an elliptic curve [11, p. 256]. A nontrivial GCD will usually yield a factorization of N, and the chance of finding a nontrivial GCD increases with mn.

Let lg denote logarithm to the base 2 and ln denote natural logarithm. Previous methods required $O(mn)$ operations mod N to do the above tests; this comes to $O(mn \lg^2(N))$ bit operations, using classical algorithms for modular arithmetic. If instead we select m and n such that $m \mid n$ and m is a power of 2, then we can do it in $O((n+m)\lg(m)\lg(N)\,[\lg(m)+\lg(N)])$ operations and $O(m\,[\lg(m)+\lg(N)])$ space by using fast polynomial convolution algorithms. The time is even less for the $P-1$ algorithm if we require that the y-sequence be a geometric progression mod N. The constants are sufficiently small that

Received March 24, 1989.
1980 *Mathematics Subject Classification* (1985 *Revision*). Primary 11Y05; Secondary 11A07, 11T06, 11Y16, 65W05.
Key words and phrases. Convolutions, FFT, residue number systems, smooth groups, factorization.
This work was done while the first author was at Unisys.

the method is practical. For example, if m is 16384, n is 131072, and N is a 100-digit number, then an Alliant FX/8 with four computational elements can complete the approximately $2 \cdot 10^9$ tests in 16 minutes. In this way we found the factors in Table 2 (see §8).

2. Classical $P-1$ algorithm

The $P-1$ factoring algorithm [14] depends, for its success, upon the concept of a group of smooth order. An integer is said to be y-smooth if all of its prime factors are less than y. Suppose that we have an arbitrary composite odd integer N that we wish to factor, and that p is a prime dividing N. Preselect a limit B_1 (positive, real), and put

$$M = \prod_{\substack{p_i^{\alpha_i} < B_1 \le p_i^{\alpha_i+1} \\ p_i \text{ prime}}} p_i^{\alpha_i}, \tag{2.1}$$

so that M is the product of prime powers less than B_1. Then, choose any small integer $a \neq \pm 1$ which is coprime to N, and compute

$$H = a^M \bmod N. \tag{2.2}$$

If $p \mid N$ and $p-1 \mid M$, then Fermat's Little Theorem implies that for some integer k,

$$a^M \bmod N = a^{k(p-1)} \bmod N \quad \text{and} \quad a^{k(p-1)} - 1 \equiv 0 \pmod{p}.$$

Hence,

$$p \mid \mathrm{GCD}(a^M - 1, N) = \mathrm{GCD}(H - 1, N). \tag{2.3}$$

Relation (2.3) will be true whenever $p-1$ has all of its prime power divisors less than B_1, that is to say, the order $|(\mathbb{Z}/p\mathbb{Z})^*|$ of the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ is B_1-smooth. It is a possible but rare occurrence for large N that N has additional factors q of the same form as p. In that case all will appear in the GCD of (2.3). To rectify this problem, one reduces the value of M. Otherwise, p will equal the GCD, and not just divide it, in (2.3).

A second stage of the algorithm succeeds whenever $|(\mathbb{Z}/p\mathbb{Z})^*|$ is smooth up to B_1 and has a *single* prime factor between B_1 and B_2, where $B_2 \gg B_1$. This part of the algorithm proceeds as follows. Precompute the quantities

$$H^2, H^4, \ldots, H^R \bmod N, \tag{2.4}$$

where the maximum exponent R equals or exceeds the largest gap between successive primes in the interval $(B_1, B_2]$. It seems not to exceed $\ln^2(B_2)$ [3, 17]. For $B_1 < s \le B_2$, define

$$Q_s = H^s \bmod N = a^{Ms} \bmod N.$$

If p_j is the jth prime in the interval $(B_1, B_2]$, then

$$Q_{p_1} \equiv H^{p_1} \bmod N \quad \text{and} \quad Q_{p_{j+1}} \equiv H^{p_{j+1}-p_j} Q_{p_j} \pmod{N}, \tag{2.5}$$

where $H^{p_{j+1}-p_j} \bmod N$ is found by table look-up. Accumulate the product

$$P = \prod_{j=1}^{\pi(B_2)-\pi(B_1)} (Q_{p_j} - 1) \bmod N, \tag{2.6}$$

and periodically compute $\mathrm{GCD}(P, N)$. Step 2 succeeds if $p-1 \mid Ms$ for some prime s with $B_1 < s \leq B_2$. When the algorithm is implemented in this fashion, each prime in the interval $(B_1, B_2]$ requires one multiplication mod N from (2.5) and one from (2.6). Add the setup cost in (2.4) to get a total of $2(\pi(B_2) - \pi(B_1)) + O(\ln^2(B_2))$ modular multiplications. Montgomery [11] suggests methods for reducing the number of multiplications.

3. Convolution Theorem

Our algorithms require considerable manipulation of polynomials over $\mathbb{Z}/N\mathbb{Z}$, esp. multiplication and evaluation. One may use Fast Fourier Transforms (FFT) to perform the polynomial multiplications via the Convolution Theorem [1, Chapter 7].

Theorem 1 (Convolution theorem). *Let* $f(x) = \sum_{i=0}^{n-1} f_i x^i$ *and* $g(x) = \sum_{i=0}^{n-1} g_i x^i$ *be polynomials of degree at most* $n-1$. *Let*

$$s(x) = f(x)g(x) \bmod (x^n - 1) = \sum_{i=0}^{n-1} s_i x^i$$

(*so that* $s_i = \sum_{j+k \equiv i \pmod n} f_j g_k$). *Form the following vectors of length* n:

$$\vec{f} = [f_0, f_1, \ldots, f_{n-1}],$$

$$\vec{g} = [g_0, g_1, \ldots, g_{n-1}].$$

Then the circular convolution $f \otimes g$ *is*

$$\vec{s} = [s_0, s_1, \ldots, s_{n-1}] = \mathrm{FFT}^{-1}(\mathrm{FFT}(\vec{f}) * \mathrm{FFT}(\vec{g})),$$

where $*$ *indicates a pointwise product. The quantity* $\mathrm{FFT}(\vec{x})$ *is the Fast Fourier Transform of the vector* $\vec{x}$, *and* FFT^{-1} *is the inverse transform.*

The FFT in Theorem 1 is performed over the ring $\mathbb{Z}/N\mathbb{Z}$ and is exactly analogous to the more familiar FFT's over $\mathbb{C}$. If ω is a primitive root of order n [1], then the FFT of the vector $\vec{x}$ of length n is

$$\mathrm{FFT}(\vec{x}) = [y_0, y_1, \ldots y_{n-1}] \quad \text{where } y_i = \sum_{j=0}^{n-1} x_j \omega^{ij}.$$

By choosing n larger than the degree of the product and by padding f and g with leading zeros, one can compute an exact polynomial product via a circular convolution.

Assume $n > 1$ is a power of 2 and N is odd. If we know a primitive nth root of unity mod N, then a convolution algorithm based upon straightforward

implementation of the Fast Fourier Transform will take $O(n \lg(n))$ arithmetic operations mod N. Such a primitive root will not exist unless all prime factors of N are congruent to 1 mod n; however the factorization of N is unknown. Nussbaumer's convolution algorithm [13; 8, Exercise 4.6.4–59] avoids this requirement, and can perform a circular convolution of length n in time $O(n \lg(n))$ multiplications and $O(\lg^2(n))$ additions mod N. We do not use, and hence do not describe, Nussbaumer's method in detail. The convolutions can be performed more efficiently with the use of Residue Number Systems, because multiplication mod N is expensive.

4. Polynomial convolutions by residue numbers

One can multiply two polynomials in $\mathbb{Z}/N\mathbb{Z}[x]$ together mod N by multiplying them together over $\mathbb{Z}$ and then reducing the coefficients mod N. If the original coefficients are in the interval $[0, N-1]$, and both original polynomials have degree at most $n-1$, then the coefficients of the polynomial product will be in the interval $[0, n(N-1)^2]$. By reducing the original coefficients modulo enough primes p_i, performing separate convolutions over the finite fields $\mathbb{Z}/p_i\mathbb{Z}$, and then using the Chinese Remainder Theorem (henceforth abbreviated as CRT), the coefficients of the product mod N can be reconstructed. The key to the usefulness of this approach is that all of the computations may now be performed in single-precision arithmetic if the p_i's are chosen properly.

To compute a polynomial product $f(x)g(x)$ mod N mod $(x^n - 1)$, where f and g have degree at most $n-1$ with coefficients in the interval $[0, N-1]$, select K distinct primes p_i for $0 \le i \le K-1$ with $p_i \equiv 1 \pmod{n}$, such that

$$P = \prod_{i=0}^{K-1} p_i > nN^2/(1-\varepsilon), \qquad \varepsilon > 0. \tag{4.1}$$

Try to select the primes p_i as large as possible while remaining single precision, in order to minimize K. If the machine has too small a word size, then there may not be enough primes available, forcing the use of double precision. If all p_i are in an interval $[p_{\max}/2, p_{\max}]$, then the Prime Number Theorem says (approximately)

$$K \le \frac{p_{\max} - p_{\max}/2}{\phi(n)\ln(p_{\max})} = \frac{p_{\max}}{2\phi(n)\ln(p_{\max})}.$$

But (4.1) requires $K\ln(p_{\max}) > \ln(n) + 2\ln(N)$. Consequently,

$$p_{\max} \ge 2\phi(n)[\ln(n) + 2\ln(N)].$$

For example, if $n = 65536$ and $N \approx 10^{1000}$, then

$$p_{\max} \ge 2 \cdot 32768 \cdot 4620 \approx 1.1 \cdot 2^{28},$$

requiring the use of 29–bit primes. If one uses 32-bit primes, then

$$K \approx \frac{\lg(n) + 2\lg(N)}{32} \approx \frac{\lg(N)}{16}. \tag{4.2}$$

One now performs K single-precision convolutions mod p_i, rather than one convolution mod N. This construction provides an additional advantage in that the individual convolutions are independent of one another and may be performed in parallel or in vector mode. The congruence restrictions on p_i ensure that appropriate primitive roots of order n, required to do the FFT's, exist.

Let s_{ij}, $0 \le i \le K-1$, $0 \le j \le n-1$ be the coefficients obtained from the individual convolutions taken mod p_i. We use the CRT to reconstruct the coefficients of the product $f(x)g(x) \bmod (x^n-1) \bmod N$ from those convolutions. Precompute the quantities

$$v_i = P/p_i \quad \text{and} \quad r_i = v_i^{-1} \bmod p_i, \qquad 0 \le i \le K-1.$$

For each j, $0 \le j \le n-1$, compute

$$y_{ij} = r_i s_{ij} \bmod p_i, \qquad 0 \le i \le K-1, \tag{4.3}$$

and

$$S_j = \sum_{i=0}^{K-1} v_i y_{ij} - k_j P \quad \text{where } k_j = \left\lfloor \sum_{i=0}^{K-1} \frac{y_{ij}}{p_i} + \varepsilon/2 \right\rfloor. \tag{4.4}$$

Let the jth coefficient of $f(x)g(x) \bmod (x^n-1)$ be s_j. Then

$$s_j \equiv s_{ij} \equiv (v_i r_i) s_{ij} \equiv v_i y_{ij} \equiv S_j \pmod{p_i}$$

for all i, implying $s_j \equiv S_j \pmod{P}$. The choice of P in (4.1) ensures $0 \le s_j \le n(N-1)^2 < P$, and the choice of k_j in (4.4) ensures $0 \le S_j < P$. Consequently $s_j = S_j$, implying $s_j \bmod N = S_j \bmod N$.

When evaluating $S_j \bmod N$, one can use precomputed values of $v_i \bmod N$ and $P \bmod N$ in (4.4).

The value of k_j can be estimated by floating-point arithmetic if one selects ε in (4.1) and (4.4) so the accumulated error (due to round-off) in the second sum of (4.4) will not exceed $\varepsilon/2$. Indeed,

$$\begin{aligned}
\sum_{i=0}^{K-1} \frac{y_{ij}}{p_i} + \varepsilon/2 &= \sum_{i=0}^{K-1} \frac{v_i y_{ij}}{P} + \varepsilon/2 \\
&= (S_j + k_j P)/P + \varepsilon/2 \\
&\in [k_j + \varepsilon/2,\ k_j + \varepsilon/2 + n(N-1)^2/P] \\
&\subseteq [k_j + \varepsilon/2,\ k_j + \varepsilon/2 + (1-\varepsilon)] \\
&= [k_j + \varepsilon/2,\ k_j + 1 - \varepsilon/2]
\end{aligned}$$

differs by at least $\varepsilon/2$ from the nearest integer.

The usual FFT algorithm divides by n at the end of each convolution mod p_i. These divisions may be incorporated into (4.3) by replacing r_i by $n^{-1} r_i \bmod p_i$.

5. FFT IMPLEMENTATION OF STEP 2

Select $\theta \approx \sqrt{B_2}$ and compute the coefficients a_i of

$$f(x) = \sum_{i=0}^{\phi(\theta)} a_i x^i = \prod_{\substack{0 \le u < \theta \\ \mathrm{GCD}(u,\theta)=1}} (x - H^u) \bmod N, \tag{5.1}$$

where ϕ denotes the Euler totient function. After computing the powers H^u, the coefficients a_i can be computed by writing the product in (5.1) as a product of polynomials of equal degree, and recursively multiplying them together in pairs, using the methods of §4.

Next, evaluate $f(x)$ along the geometric progression

$$H^{v\theta}, \qquad 1 \le v \le B_2/\theta. \tag{5.2}$$

The reason is that if $p \mid \mathrm{GCD}(H^s - 1, N)$ where $s = v\theta - u$, then

$$p \mid \mathrm{GCD}(H^{v\theta} - H^u, N), \tag{5.3}$$

so that

$$p \mid \mathrm{GCD}(f(H^{v\theta}), N). \tag{5.4}$$

These points of evaluation form a geometric progression mod N. A polynomial of degree n may be evaluated along m points of a geometric progression with a circular convolution of length $n + m$, followed by some simple post-multiplications. Given f defined by (5.1), suppose we wish to evaluate it at $x = e, er, er^2, \ldots, er^{m-1}$. Set $L = n + m - 1$, where $n = \phi(\theta)$, and form the vectors (all taken mod N)

$$\begin{aligned} y_k &= a_k r^{-k(k+1)/2}, && 0 \le k \le n; \\ y_k &= 0, && n+1 \le k \le L; \\ z_k &= e^{L-k} r^{(L-k)(L-k+1)/2}, && 0 \le k \le L. \end{aligned} \tag{5.5}$$

Compute the circular convolution, $y \otimes z$, of the vectors y and z, yielding another vector u of length $L + 1$. We then apply the following result.

Theorem 2 (Polynomial evaluation). *If the vectors $\vec{y}$ and $\vec{z}$ are given as in* (5.5), *and $u = y \otimes z$, then*

$$f(er^k) = e^{-k} r^{-k(k+1)/2} u_{L-k}, \qquad 0 \le k \le m - 1. \tag{5.6}$$

Proof. By the definition of a circular convolution, the elements of $y \otimes z$ are the coefficients of a polynomial u which is the product mod $(x^{L+1} - 1)$ of two

polynomials whose coefficients are given by y_j and z_k. If $i \geq n$, then

$$u_i = \sum_{\substack{j+k \equiv i \pmod{L+1} \\ 0 \leq j \leq L \\ 0 \leq k \leq L}} y_j z_k = \sum_{\substack{j+k \equiv i \pmod{L+1} \\ 0 \leq j \leq n \\ 0 \leq k \leq L}} y_j z_k$$

$$= \sum_{\substack{j+k=i \\ 0 \leq j \leq n \\ 0 \leq k \leq L}} a_j r^{-j(j+1)/2} e^{L-k} r^{(L-k)(L-k+1)/2}.$$

Let $i = L - s$, where $0 \leq s \leq m-1$. This implies that $k = L - s - j$, and

$$u_{L-s} = \sum_{j=0}^{n} a_j r^{-j(j+1)/2} e^{s+j} r^{(s+j)(s+j+1)/2}$$

$$= e^s r^{s(s+1)/2} \sum_{j=0}^{n} a_j e^j r^{sj} = e^s r^{s(s+1)/2} f(er^s). \quad \square$$

As with the coefficient evaluation, these convolutions may be performed modulo a fixed set of primes and the CRT used to reassemble the results mod N. The CRT need be used only for those coefficients required by (5.6).

Our application uses (5.5) and (5.6) with $r = H^\theta \bmod N$, while e varies. Since $\mathrm{GCD}(r^{k(k+1)/2} e^k, N) = 1$, we can ignore the multiplications from (5.6) and take greatest common divisors directly from the final convolution vector in (5.4). The vector y is fixed, so it and its forward transforms mod p_i need to be computed only once. The powers of r and $1/r$ in (5.5) may be precomputed, so that computation of z_k requires only computing the relevant powers of e and multiplying them by the appropriate power of r.

If $(p-1) \mid Ms$, then the algorithm will succeed from (5.6) at $k = \lfloor \frac{s+\theta}{\theta} \rfloor$. In essence, evaluating $f(r^k)$ tests all possible primes s in the range $[\theta(k-1), \theta k]$. The reason we need powers of e in (5.5) and (5.6) is because we need to evaluate $f(r^x)$, $x = 0, \ldots, B_2/\theta$ and we evaluate it at only m points at a time. The first progression runs from 1 to r^{m-1}, the second from r^m to r^{2m-1}, etc. We thus set e successively to $1, r^m, r^{2m}, \ldots$ and perform multiple convolutions of length $\phi(\theta) + m$.

It is not necessary to compute a GCD for every k in (5.6). Instead, one can compute $\prod_{k=0}^{m-1} u_{L-k} \bmod N$ and then take a single GCD. If multiple factors appear, then one can go back and compute the GCD's individually.

6. Complexity

In this section we compare the complexity of performing the convolutions mod N, using Nussbaumer's algorithm, versus using the method in §4. We also suggest how to select the algorithm parameters to yield good performance. We take, as our fundamental unit of work, a single-precision modular multiplication.

Consider the cost of a circular convolution of length n with coefficients in $\mathbb{Z}/N\mathbb{Z}$. Nussbaumer's algorithm uses $O(n \lg(n))$ multiplications mod N and $O(n \lg^2(n))$ additions mod N. If one assumes classical algorithms for arithmetic, then the cost of a modular multiplication mod N is $O(\lg^2(N))$ and the cost of a modular addition is $O(\lg(N))$, for a total cost of

(6.1) $O(n \lg(n) \lg(N) [\lg(n) + \lg(N)])$ units (Nussbaumer's algorithm).

Using the CRT requires $O(n \lg(n)K)$ units to do a convolution of length n mod all p_i, plus $O(n \lg(N)K)$ to reduce the original coefficients mod p_i, and $O(n \lg(N)K)$ more to perform the CRT. Upon substituting K from (4.2), we find that the work per convolution is

(6.2) $O(n \lg(N) [\lg(n) + \lg(N)])$ units (CRT).

A. **Computation of coefficients.** Let $d = \phi(\theta)$. To compute the polynomial of degree d, we perform one convolution of length d, two of length $d/2$, four of length $d/8$, ... The total work by Nussbaumer's algorithm sums to

(6.3) $O(d \lg^2(d) \lg(N) [\lg(d) + \lg(N)])$ units.

If we use the CRT, then the total work is

(6.4) $O(d \lg(d) \lg(N) [\lg(d) + \lg(N)])$ units.

We observe that (6.4) is a factor of $O(\lg(d))$ faster than (6.3). We also note that the factor of $1/16$, from (4.2), does not affect the asymptotic complexity, but has great practical value. A naive computation of the coefficients, by direct long multiplication, would take

(6.5) $O(d^2 \lg^2(N))$ units.

B. **Evaluation of the Polynomial.** Performing a convolution of length $L + 1$ to evaluate the polynomial mod N via Nussbaumer's method will take

(6.6) $O((L + 1) \lg(L + 1) \lg(N) [\lg(L + 1) + \lg(N)])$ units.

Performing the convolutions via the CRT will take

(6.7) $O((L + 1) \lg(N) [\lg(L + 1) + \lg(N)])$ units.

We observe that (6.7) is a factor of $O(\lg(L + 1))$ faster than (6.6).

C. **Example.** To illustrate how much of a typical speedup is obtained by using convolutions over (2.5) and (2.3), let us take $B_2 = 10^{10}$. Consider using Nussbaumer's algorithm and select $\theta = 72930 = 2 \cdot 3 \cdot 5 \cdot 11 \cdot 13 \cdot 17 \approx 10^5$. We then have $d = \phi(\theta) = 15360$. The advantage of selecting θ in this fashion is twofold. The first is that $\phi(\theta)$ is small, relative to θ, and hence the degree of the polynomial is small. The second is that it allows the sizes of the convolutions to be very close to powers of 2, simplifying computation of the FFT's.

Nussbaumer's algorithm takes exactly $3 \cdot 2^{n-1+\lceil \lg n \rceil}$ multiplications mod N for a convolution of length 2^n [8, Exercise 4.6.4–59], so with Nussbaumer's

algorithm, for our chosen θ, computation of the coefficients in (5.1) will take $\sum_{n=5}^{14} 2^{14-n} \cdot 3 \cdot 2^{n-1+\lceil \lg(n) \rceil} = 3 \cdot 2^{20} \approx 3.1 \cdot 10^6$ multiplications mod N. The time to perform the necessary additions mod N will be small relative to this quantity. To evaluate the polynomial, we will need to compute $f(H^{v\theta})$ for $v = 1, \ldots, (10^{10}/72930) \approx 138000$. This will be done in eight convolutions of length 32768, each convolution computing $32768 - 15360 = 17408$ points. The convolutions will then take $8 \cdot 3 \cdot 2^{18} \approx 6.3 \cdot 10^6$ multiplications mod N, and the total work will be about $9.4 \cdot 10^6$ modular multiplications. Since there are approximately $4.5 \cdot 10^8$ primes less than 10^{10}, performance of step 2 via convolutions will be about 95 times better than the original method using (2.5).

Use of the CRT will result in an even greater speedup over Nussbaumer's method. For a 300-bit N represented in radix 2^{30} it takes 210 single-precision multiplication instructions to multiply two numbers mod N. It also takes twenty 31-bit primes from (4.1) for the CRT. A convolution of length 2^n will therefore take $210 \cdot 3 \cdot 2^{n-1+\lceil \lg(n) \rceil} = 315 \cdot 2^{n+\lceil \lg(n) \rceil}$ operations using Nussbaumer's algorithm. Assuming that the necessary powers of the primitive roots of unity have been precomputed, careful counting of the total operations involved in the CRT reveals a total of $(60n + 670)2^n$ multiplications/divisions to perform the convolutions and reconstruct the coefficients. For $n = 15$, the CRT is therefore 3.4 times as fast. This will increase asymptotically to n times as fast as $N \to \infty$.

7. Algorithmic description and coding considerations

For the example given above we have $\phi(\theta) = 15360 = 2^{10} \cdot 3 \cdot 5$. In practice, before beginning the convolutions, it is convenient to break the product in (5.1) into smaller products, each a polynomial of degree equal to the odd part of $\phi(\theta)$. Our code therefore starts by forming 1024 polynomials mod N of degree 15. Multiply these in pairs to get 512 polynomials of degree 30, then 256 of degree 60, ..., 1 of degree 15360. At each stage, all convolution lengths are equal, so it suffices to select the p_i and do the precomputations for the CRT only once per stage. Each polynomial mod N is converted to K polynomials mod p_i, the convolutions are performed, and the polynomials mod p_i are then converted back to a single polynomial mod N. The full algorithm is as follows:

(1) Select B_1 and perform step 1 of the $P-1$ algorithm. Let H be the output.
(2) Verify $\mathrm{GCD}(H-1, N) = 1$.
(3) Given H from step 1, compute H^u for $1 \le u \le \theta$, $\mathrm{GCD}(u, \theta) = 1$.
(4) Form, via long multiplication, $\phi(\theta)/q$ polynomials mod N of degree q, where q is the odd part of $\phi(\theta)$. Let *conlen* be the least power of 2 exceeding $2q$.
(5) Repeat the following procedure until one has a single polynomial $f(x)$ of degree $\phi(\theta)$, as in (5.1).
5a. Select an appropriate set of primes. These primes must be congruent

to 1 (mod *conlen*). Find appropriate primitive roots of these primes for the FFT's. The primes and their roots can be precomputed and placed in a table.

5b. Reduce the polynomials mod each of these primes.

5c. Multiply the reduced polynomials in pairs, via the Convolution Theorem, yielding $\phi(\theta)/(2q)$ polynomials of degree $2q$ for each p_i. Replace q by $2q$ and *conlen* by $2conlen$.

5d. Reconstruct products mod N from the separate polynomials mod p_i via the CRT.

(6) Compute or look up in a table the necessary primes for (5.5) and (5.6).

(7) Reduce $f(x)$ modulo these primes and perform the convolutions to evaluate $f(x)$ mod p_i.

(8) Reconstruct $f(x)$ mod N via the CRT and compute the GCD's in (5.4).

Remark 1. One can effectively find a primitive root of order r mod p by successively trying $a^{(p-1)/r}$ for $a = 2, 3, \ldots$. If r is a power of 2, it suffices to make a a quadratic nonresidue mod p.

Remark 2. There are variations to our scheme for the calculation of the coefficients. Rather than treat the coefficients as integers in $[0, N-1]$, they can instead be treated as in the interval $[(1-N)/2, (N-1)/2]$. In this case the bound in (4.1) for P can be halved and the computation of k_j in (4.4) should round to the nearest integer. However, this scheme will usually not reduce K, and it requires signed arithmetic during the conversion to and from residue numbers. It is easier, when doing multiple-precision arithmetic, to use only unsigned quantities.

Remark 3. The vectors in (5.5) can be computed without modular exponentiations (which are expensive). To compute $r_k = r^{k(k+1)/2}$ for successive values of k, it suffices to do the following:

(1) Set $r_0 = 1$ and let $R_0 = r$.

(2) For $i = 1, \ldots, n$, compute $r_i = r_{i-1}R_{i-1}$ and $R_i = R_0 R_{i-1}$.

This replaces each modular exponentiation with just two modular multiplications (but does not parallelize well). Likewise, one can replace the last line of (5.5) by

(1) Set $z_L = 1$ and let $Z_L = e$.

(2) For $i = L-1, \ldots, 0$, compute $Z_i = rZ_{i+1}$ and $z_i = z_{i+1}Z_i$.

Remark 4. Instead of using $0 \le u < \theta$ and $\mathrm{GCD}(u, \theta) = 1$ in (5.1), one can use arbitrary representatives of $(\mathbb{Z}/\theta\mathbb{Z})^*$ for u provided each $|u| \ll B_2$. The advantage is that one can use the multiplicative relations amongst the H^u to shorten the computation of f by a factor of $O(\lg(\phi(\theta)))$. We illustrate the procedure assuming $\theta = 2q_1q_2q_3$ for distinct odd primes q_i, with each $q_i \equiv 1$ (mod 2^{α_i}).

(1) Initialize

$$F(x, y) = \prod_{\substack{1 \le j < \theta \\ j \text{ odd} \\ 1 \le j \bmod q_i \le (q_i - 1)/2^{\alpha_i} \forall i}} (x - H^j y) \bmod N$$

(homogeneous of degree $(q_1 - 1)(q_2 - 1)(q_3 - 1)/2^{\alpha_1+\alpha_2+\alpha_3} = \phi(\theta)/2^{\alpha_1+\alpha_2+\alpha_3}$).

(2) For $1 \le i \le 3$ and $1 \le k \le \alpha_i$, replace

$$F(x, y) \leftarrow F(x, y)F(x, H^m y) \bmod N \quad \text{where} \begin{cases} m \equiv (q_i - 1)/2^k \pmod{q_i} \\ m \equiv 0 \pmod{\theta/q_i}. \end{cases}$$

The iterations over i and k can occur in any convenient order. Each iteration doubles the degree of F, and uses approximately $2\lg(m) + 2\deg(F)$ multiplications mod N to compute the coefficients of $F(x, H^m y)$ from those of $F(x, y)$, plus one convolution of length at least $2\deg(F) + 1$ to do the polynomial multiplication.

(3) Set $f(x) = F(x, 1)$.

Remark 5. Our implementation uses 42 Mbytes of data storage and can accommodate $N \approx 250$ decimal digits. Storage requirements grow linearly with $\lg N$. We did not optimize the program for space and can cut storage at least in half by using some statically defined arrays for different things in different places. For example, we declared separate arrays for the coefficient and polynomial evaluation calculations. A minimal storage implementation should be able to handle 100-digit composites in 6–8 megabytes of storage without difficulty.

8. Results

In this section we present timing information for various stages of the algorithm on an Alliant FX/8 computer with four computational elements (CE's). Each one has vector instructions which make performing FFT's and dot products as in (4.4) very efficient. The Alliant computer is a tightly coupled parallel processor with shared memory. Many stages of our algorithm can be computed in parallel with 100% efficiency. For example, the convolutions mod each of the separate primes from (4.1), the reconstruction of the coefficients from $S_j \bmod N$, and the GCD's from (5.4) may all be conducted in parallel. Furthermore, since there are K primes from (4.1), T processors ($T \le K$) will run approximately T times as fast as one processor. The step-2 parts of the computations that are not parallelizable form only a small percentage of the total work. Table 1 gives timing information in seconds for a step-2 limit of 10^{10} for 25-, 50- and 100-digit composite integers. The times given that are relevant to (5.5) and (5.6) are those for one iteration only, using the parameters given in §6.C. Eight separate iterations of (5.5) and (5.6) are required to cover the entire interval from B_1 to B_2. The time given to set up the vectors in (5.5) is split into two parts. The first number is the time to compute the powers of r,

TABLE 1. Algorithm performance in seconds (4 computational elements)

Computation	25 Digits	50 Digits	100 Digits
Compute roots of (5.1)	12	48	195
Form 1024 polynomials of degree 15	27	110	440
Convolution time for (5.1)	105	212	425
Coefficient reduction & CRT time for (5.1)	96	253	700
Construct vectors for (5.5)	28/5	105/18	380/70
Convolutions for (5.5)	83	165	330
CRT time for (5.5)	13	50	200

and occurs only once. The second number is the time to multiply by the powers of e for z_k, and occurs at each iteration.

Using naive modular multiplication, step 1 takes $O(B_1 \lg^2(N))$ time. It does not parallelize at all, short of programming the underlying modular multiplication in parallel. A single computational element of the Alliant takes 5500 seconds to perform step 1 for $B_1 = 3 \cdot 10^6$ and $N \approx 10^{100}$.

A step-2 implementation based upon (2.5) and (2.6) takes approximately 55000 seconds for $B_2 = 10^8$ and $N = 10^{100}$ on a single CE. It takes about 8500 seconds when speeded by the methods of [11]. We have not taken step 2 to 10^{10} by the older methods, but since $\pi(10^{10})/\pi(10^8) \approx 79$, we project that for $B_2 = 10^{10}$ step 2 will take about 4,300,000 seconds using (2.5) and (2.6) directly, and 660,000 seconds by the methods of [11] for a single CE. Furthermore, computation of step 2 via (2.5) and (2.6) does not parallelize well. The newer methods presented herein take 30700 seconds for a single CE; a factor of 139 over (2.5) and (2.6) and a factor of 22 over the methods of [11]. We observe that these factors will increase as N increases.

Table 2 lists some new factorizations found by the method described herein, using a step-1 limit of $3 \cdot 10^6$ and a step-2 limit of 10^{10}. With the exception of U575 they are all taken from the 'Cunningham Project' [6]. The designations $y, xxx\pm$ indicate a cofactor of $y^{xxx} \pm 1$ respectively. The number U575 was a cofactor of the 575th Fibonacci number and its 34-digit factor is the largest factor yet found by $P-1$. We note that on average the Elliptic Curve algorithm

will still be more effective than the $P-1$ algorithm, even with our enhancements. While step 1 of ECM runs about 8 times slower than step 1 of $P-1$, step 2 takes about the same time for both, using non-FFT techniques. A comparison of the run times and effectiveness of the two methods was presented in [5]. It is difficult to directly compare the run time of the two algorithms since ECM is a random method while $P-1$ is deterministic. Many factors which would have otherwise been found by our method had already been found by ECM.

TABLE 2. Factorizations

N	Factor (P)	$P-1$
2,584+	32871186029052837857	$2^5 \cdot 13 \cdot 73 \cdot 163 \cdot 209333 \cdot 31722973$
2,656+	591100350038949953	$2^6 \cdot 41 \cdot 71 \cdot 5737 \cdot 553036999$
2,664+	3508428556083287152033	$2^5 \cdot 3 \cdot 71 \cdot 83 \cdot 151 \cdot 2007359 \cdot 15751691$
2,784+	66308056470365249	$2^6 \cdot 7^2 \cdot 11 \cdot 19 \cdot 433 \cdot 233644769$
2,1008+	4940773915639703903354880001	$2^{16} \cdot 3^2 \cdot 5^3 \cdot 7 \cdot 31 \cdot 67 \cdot 168677 \cdot 2732575201$
2,1032+	242193739165634585377	$2^5 \cdot 3^2 \cdot 43 \cdot 1861 \cdot 24371 \cdot 431203469$
2,535+	10155021917057853331411	$2 \cdot 3^5 \cdot 5 \cdot 107 \cdot 5347 \cdot 17093 \cdot 427328971$
2,591+	8588408897489780521	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 167 \cdot 197 \cdot 367 \cdot 282269147$
2,661+	366407098838775691153027 31	$2 \cdot 5 \cdot 457 \cdot 661 \cdot 131543 \cdot 163337 \cdot 564538939$
2,733+	60321002146908466556237317 69	$2^3 \cdot 3 \cdot 7^2 \cdot 17^2 \cdot 67 \cdot 173 \cdot 733 \cdot 1634681 \cdot$ $\cdot 1277925359$
2,813+	897186516077633497	$2^3 \cdot 3 \cdot 11 \cdot 13 \cdot 271 \cdot 3559564581$
2,873+	921446320166308603	$2 \cdot 3^2 \cdot 29 \cdot 97 \cdot 309107 \cdot 58873379$
2,895+	5070463106922154841	$2^3 \cdot 5 \cdot 19 \cdot 179 \cdot 5737 \cdot 6496749983$
2,1017+	28876407885213594067	$2 \cdot 3^2 \cdot 41 \cdot 113 \cdot 1092173 \cdot 317042093$
U575	7146831801094929757704917464134401	$2^8 \cdot 5^2 \cdot 11^2 \cdot 23^2 \cdot 821 \cdot 3371 \cdot 39209 \cdot$ $\cdot 3394739 \cdot 47358559$

9. APPLICATION TO $P+1$ ALGORITHM

For an integer n, the nth Lucas (Chebyshev) polynomial, V_n, is defined by the formal identity $V_n(x+x^{-1}) = x^n + x^{-n}$. If $n > 0$, then V_n is a monic polynomial of degree n over $\mathbb{Z}$. It satisfies

$$\begin{aligned} &V_0(x) = 2, && V_{mn}(x) = V_m(V_n(x)), \\ &V_1(x) = x, && V_m(x)V_n(x) = V_{m+n}(x) + V_{m-n}(x), \\ &V_2(x) = x^2 - 2, && V_{2n}(x) = V_n(x)^2 - 2. \end{aligned} \tag{9.1}$$

The $P+1$ factoring algorithm [16] starts with an integer a, chooses B_1 and M as in (2.1), and computes

$$H = V_M(a) \bmod N \tag{9.2}$$

instead of (2.2). Suppose $p \mid N$, $(p+1) \mid M$, and $a^2 - 4$ is a quadratic nonresidue mod p. Choose $\alpha \in \mathrm{GF}(p^2)$ satisfying $a = \alpha + \alpha^{-1}$. Then $a =$

$a^p = \alpha^p + \alpha^{-p}$, which implies that $\alpha^p = \alpha$ or $\alpha^p = \alpha^{-1}$. The former is impossible since $a^2 - 4$ is a quadratic nonresidue. Hence $\alpha^{p+1} = 1$ and

$$V_M(a) = \alpha^M + \alpha^{-M} \equiv 2 \pmod{p}. \tag{9.3}$$

This implies that $p \mid \mathrm{GCD}(H-2, N)$ (cf. (2.3)). The $P+1$ algorithm operates in the multiplicative subgroup of elements of $\mathrm{GF}(p^2)^*$ having norm 1 (i.e., algebraic conjugate equal to multiplicative inverse).

As in $P-1$, if the first step of $P+1$ is unsuccessful, we can still succeed (providing $a^2 - 4$ and hence $H^2 - 4$ is a quadratic nonresidue) if $P+1$ is smooth up to B_1 and has only one prime factor between B_1 and B_2. Assume that $\theta \equiv 2 \pmod 4$. Set $\alpha = (H + \sqrt{H^2 - 4})/2 \bmod N$ so that $H \equiv \alpha + \alpha^{-1} \pmod N$ (in the machine, α would have real and imaginary parts mod N). An FFT version of step 2 can replace (5.1) by

$$f(x) = \prod_{\substack{|u|<\theta/4 \\ \mathrm{GCD}(u,\theta/2)=1}} (x - \alpha^{2u}) = \prod_{\substack{0<u<\theta/4 \\ \mathrm{GCD}(u,\theta/2)=1}} (x^2 - V_{2u}(H)x + 1). \tag{9.4}$$

Instead of (5.2), evaluate f along the geometric progression

$$\alpha^{v\theta/2} \quad \text{where } 1 \le v \le 2B_2/\theta,\ v \equiv 1 \pmod 2. \tag{9.5}$$

As in (5.3) and (5.4), if $s = v\theta/2 - 2u$ and $p \mid \alpha^s - 1 \mid V_s(H) - 2$, then

$$p \mid (\alpha^{v\theta/2} - \alpha^{2u}) \mid f(\alpha^{v\theta/2}).$$

The degree of f is $\phi(\theta/2) = \phi(\theta)$, as in the $P-1$ algorithm. Since this f is a reciprocal polynomial (i.e., $f(x) = x^{\deg(f)} f(1/x)$), it needs only half the storage. This f will be evaluated at about B_2/θ points, as in $P-1$, but the points of evaluation (though not the coefficients of f) have both real and imaginary parts when expressed mod N. This means that the methods of §5 cannot be directly applied to (5.6). The problem is easily circumvented by doing four convolutions: one with the real parts of both sequences, one with both imaginary parts (result multiplied pointwise by $H^2 - 4$), and two with a real part and an imaginary part. This can be reduced to three convolutions (and one multiplication by $H^2 - 4$) by emulating the trick

$$\begin{aligned}(a + b\sqrt{H^2 - 4})&(c + d\sqrt{H^2 - 4}) \\ &= ac + bd(H^2 - 4) + ((a+b)(c+d) - ac - bd)\sqrt{H^2 - 4}.\end{aligned}$$

Since $H^2 - 4$ is assumed to be a quadratic nonresidue mod p, we can do even better. If $p \mid f(\alpha^{v\theta/2})$, then both the real and imaginary parts of $f(\alpha^{v\theta/2})$ must be divisible by p. It suffices to compute only one of these parts before taking GCD's; that can be done with *two* convolutions mod N.

10. Relationship to elliptic curves

An elliptic curve E over a finite field whose characteristic is not 2 or 3 consists of the solutions (x, y) of a cubic equation $y^2 = x^3 + Ax + B$ where

$4A^3 + 27B^2 \neq 0$. These solutions, together with a point at infinity, form an additive, commutative group known as the Mordell-Weil group. When defined over a field of order p, the order of the Mordell-Weil group is bounded by $[p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1]$, by a theorem of Hasse [15, p. 131]. By changing the pair (A, B), one changes curves and can obtain groups of different orders. Whereas the $P-1$ algorithm provides only one group whose order must be smooth for success, the elliptic curve algorithm (ECM) [9, 11] provides many such groups, only one of which must be smooth. In order to implement an FFT version of ECM via the methods of §5, it is necessary to construct a ring $R[E, p]$ whose group of units R^* has a subgroup isomorphic to the additive Mordell-Weil group. One can do this by decomposing the Mordell-Weil group into its cyclic subgroups, but the decomposition requires one to know the group order in the first place! We would like to be able to find an explicit injection which takes the result, H, from step 1 of ECM and maps it into the ring R. If we could do this, the algorithm could be made much more effective.

Another approach uses Brent's birthday paradox algorithm [4]. This variant of step 2 of ECM generates many random points (x, y) on the curve, all of which are multiples of the point generated by step 1. If two of these points happen to be equal or to be negatives of each other, then they will have identical x-coordinates. Construct a polynomial like (5.1) with several of these x-coordinates as roots, and evaluate that polynomial at the remaining x-coordinates. The methods of §5 do not apply, since the points of evaluation will not lie in a geometric progression, but [1, §8.5] gives a way for evaluating a polynomial of degree at most $m-1$ at m points using convolutions of total length $O(m \lg(m))$. By repeating this n/m times, we can evaluate it at n points in time $O(n \lg(m) \lg(N) [\lg(m) + \lg(N)])$. This is $O(\lg(m))$ worse than the $P-1$ extension in §5, and may not be practical.

Acknowledgments

The authors are grateful to Mr. Tony Davis for providing programming support for this project and to the MITRE Corporation for having provided the computer time for this project.

Bibliography

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, MA, 1974.
2. A. Borodin and I. Munro, *The computational complexity of algebraic and numeric problems*, American Elsevier, New York, 1975.
3. R. P. Brent, *The first occurrence of certain large prime gaps*, Math. Comp. **35** (1980), 1435–1436.
4. R. P. Brent, *Some integer factorization algorithms using elliptic curves*, Research Report CMA-
R32-85, The Center for Mathematical Analysis, The Australian National University, 1985.
5. J. Brillhart, P. L. Montgomery, and R. D. Silverman, *Tables of Fibonacci and Lucas factorizations*, Math. Comp. **50** (1988), 251–259.

6. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to high powers*, (2nd ed.) Contemporary Mathematics, vol. 22, Amer. Math. Soc., Providence, RI, 1988.
7. A. M. Despain, A. M. Peterson, O. S. Rothaus, and E. H. Wold, *Fast Fourier transform processors using Gaussian residue arithmetic*, J. Parallel & Distributed Comp. **2** (1985), 219–237.
8. D. E. Knuth, *The art of computer programming, Vol.* II, *Seminumerical algorithms*, Addison-Wesley, Reading, MA, 1981.
9. H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, Ann. of Math. **126** (1987), 649–673.
10. J. H. McClellan and C. M. Rader, *Number theory in digital signal processing*, Prentice-Hall, Englewood Cliffs, NJ, 1979.
11. P. L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp. **48** (1987), 243–264.
12. A. Norton and A. J. Silberger, *Parallelization and performance analysis of the Cooley–Tukey FFT algorithm for shared memory architectures*, IEEE Trans. Comp. **C-36** (1987), 581–591.
13. H. J. Nussbaumer, *Fast Fourier transforms and convolution algorithms*, Springer-Verlag, New York, 1982.
14. J. M. Pollard, *Theorems on factorization and primality testing*, Proc. Cambridge Philos. Soc. **5** (1974), 521–528.
15. J. H. Silverman, *The arithmetic of elliptic curves*, Graduate Texts in Mathematics, vol. 106, Springer–Verlag, New York, 1986.
16. H. C. Williams, *A* $p + 1$ *method of factoring*, Math. Comp. **39** (1982), 225–234.
17. J. Young and A. Potler, *First occurrence prime gaps*, Math. Comp. **52** (1989), 221–224.

Department of Mathematics, University of California, Los Angeles, California 90024. *E-mail*: pmontgom@math.ucla.edu

The MITRE Corporation, Burlington Road, Bedford, Massachusetts 01730. *E-mail*: bs@mitre.org