Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Factoring integers with large prime variations of the quadratic sieve

H. Boender and H.J.J. te Riele

Department of Numerical Mathematics

NM-R9513 1995

Report NM-R9513
ISSN 0169-0388

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Factoring Integers with Large Prime Variations of the Quadratic Sieve

Henk Boender and Herman J.J. te Riele

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

E–mail: henkb@cwi.nl, herman@cwi.nl

Abstract

We present the results of many factorization runs with the single and double large prime variations (PMPQS, and PPMPQS, respectively) of the quadratic sieve factorization method on SGI workstations, and on a Cray C90 vectorcomputer. Experiments with 71–, 87–, and 99–digit numbers show that for our Cray C90 implementations PPMPQS beats PMPQS for numbers of more than 80 digits, and this cross–over point goes down with the amount of available central memory.

For PMPQS a known theoretical formula is worked out and tested that helps to predict the total running time on the basis of a short test run. The accuracy of the prediction is within 10% of the actual running time. For PPMPQS such a prediction formula is not known and the determination of an optimal choice of the parameters for a given number would require many full runs with that given number, and the use of an inadmissible amount of CPU–time. In order yet to provide measurements that can help to determine a good choice of the parameters in PPMPQS, we have factored **many** numbers in the 66 – 88 decimal digits range, where each number was run once with a specific choice of the parameters. In addition, an experimental prediction formula is given that has a restricted scope in the sense that it only applies to numbers of a given size, for a fixed choice of the parameters of PPMPQS. So such a formula may be useful if one wishes to factor many different large numbers of about the same size with PPMPQS.

AMS Subject Classification (1991): 11A51, 11Y05
CR Subject Classification (1991): F.2.1
Keywords & Phrases: Factorization, Multiple Polynomial Quadratic Sieve, Vector supercomputer, Cluster of workstations

1. Introduction

Let n be an odd positive integer to be factored and suppose that n is not a prime power. If we can find two integers X and Y such that

$$X^2 \equiv Y^2 \bmod n, \tag{1.1}$$

then the greatest common divisor of $X-Y$ and n is a non–trivial factor of n if $X \not\equiv \pm Y \pmod{n}$. If X and Y are randomly chosen subject to (1.1), then this yields a proper factor of n in at least 50% of the tries. This principle is the basis for the best known

general factorization methods, namely, the multi–polynomial quadratic sieve (MPQS [Bre89, Pom85, PST88, Sil87, RLW89]) and the number field sieve (NFS [LL93]).

In this paper we discuss and compare the *single large prime variation* (PMPQS) and the *double large prime variation* (PPMPQS) of MPQS, and we factor many numbers in the 66–88 decimal digits range, mainly with PPMPQS, both on SGI workstations, and on a Cray C90 vectorcomputer.

PPMPQS is known to be faster than PMPQS "by approximately a factor of 2.5 for sufficiently large n" [LM94], but the cross–over point depends heavily on the choice of the parameters in the two methods, on the computer, on the available memory, and on the implementation. It is stated further in [LM94] that PPMPQS was found to be faster than PMPQS for numbers of at least 75 decimal digits, and that the speed–up factor of 2.5 was obtained for numbers of more than 90 digits. As a comparison, a 106–digit number was factored with PMPQS in about 140 mips years, and a 107–digit number with PPMPQS in about 60 mips years, both with a factor base size of 65,500. A 116–digit number was factored with PPMPQS in about 400 mips years, with a factor base size of 120,000. No actual results for smaller numbers were given. In Thomas Denny's Master Thesis [Den93] various experiments with PPMPQS are reported for numbers in the 75 – 95 decimal digits range. From these experiments it is not clear where the cross–over point for Denny's implementation lies. The largest numbers presently factored with PPMPQS are a 120–digit number done in about 825 mips years [DDLM94], and the 129–digit RSA challenge described by Martin Gardner, done in about 5000 mips years with a factor base size of 524,339 [AGLL].

A theoretical and practical problem with PPMPQS is the determination of the optimal parameters for a number of a given size. Since it only pays to use PPMPQS for rather large numbers, and since it is difficult to accurately predict the total running time of PPMPQS on the basis of a short test run (as contrasted with PMPQS), the precise effect of one specific choice of the parameters can only be measured accurately by carrying out the complete sieve part of the job. So in order to find the *optimal* parameter choice for a given number, that would minimize the CPU–time, one would have to *repeat* the complete sieve job for several (10, say) different choices of the parameters. Of course, this does not make much sense since *one* sieve job will do to factor the number, so we decided to adopt the strategy to factor as many as possible *different* numbers in a not too wide decimal digits range, thus providing extensive experience with PPMPQS for many different numbers on the one hand, and contributing to a table of unfactored numbers [BR92] on the other hand. The price to pay for this strategy is that we can only give an *indication* of the optimal parameter choice for PPMPQS for numbers in the 65 – 90 decimal digits range.

We have implemented PPMPQS on an SGI workstation, and on a Cray vectorcomputer. Some comparative experiments with PMPQS and PPMPQS on a Cray C90 indicated that for our implementation on that machine the cross–over point lies around numbers having 80 – 85 decimal digits. For several different choices of the parameters

in PPMPQS, we have factored eight numbers in the 66 – 83 digit range on an SGI workstation, and more than 70 numbers in the 67 – 88 digit range on a Cray C90 vectorcomputer, as a contribution to the table [BR92]. Most of these numbers were already tried before with the elliptic curve method (ECM), but without success.

In Section 2 Dixon's algorithm is discussed. MPQS is described in Section 3. In Section 4 we treat the efficient generation of the polynomials in MPQS.

In Section 5 the single large prime variation of MPQS (PMPQS) is described. A known theoretical formula is worked out that helps to predict the total sieve time on the basis of a short test run. In this test run (of a few minutes CPU–time, say) the speed is determined by which so–called complete and partial relations are generated during the sieve step of the algorithm; this speed is approximately constant during the whole sieve step. The accuracy of the prediction formula is within 10% of the actual sieve time. In Section 6 the double large prime variation of MPQS is described, and an experimental prediction formula is given that has a restricted scope in the sense that it only applies to numbers of roughly the same size, and for a fixed choice of the parameters of the algorithm. In addition, for one particular number of 80 decimal digits, we have determined the optimal choice of one (of the four) parameters in PPMPQS as an illustration of the fact that this optimum is attained for a rather wide range of this parameter. Section 7 covers implementation aspects and discusses our experiments, including a comparison of our PMPQS– and PPMPQS–implementations for 71–, 87, and 99–digit numbers. An Appendix gives tables of the numbers we have factored.

2. Basic Idea

The algorithm described in this section is due to Dixon who based it on the continued fraction method of Morrison and Brillhart [MB75]. It is not efficient in practice compared to almost any other method, but it shows clearly the idea behind finding X and Y. So we mention it mainly for didactical reasons.

For $x \in \mathbf{Z}$, $|x| > \sqrt{n}$, define

$$g(x) :\equiv x^2 \bmod n.$$

(If we write $\sqrt{n}$, we always mean the positive square root of n.) Suppose that we have a finite subset $J \subset \mathbf{Z}$ such that $\prod_{x\in J} g(x)$ is a square. Then we can take

$$X = \sqrt{\prod_{x\in J} g(x)}\,,\ Y = \prod_{x\in J} x.$$

A problem is how to determine J.

Choose a positive integer B_1, let $\pi = \pi(B_1)$ be the number of primes $\leq B_1$, and $\{p_1, p_2, \ldots, p_\pi\}$ be the set of primes $\leq B_1$. Suppose that we have a set T of $t > \pi$ numbers $g(x)$ only composed of primes $\leq B_1$, i.e.,

$$g(x) = p_1^{e_1(x)} \cdot p_2^{e_2(x)} \cdots p_\pi^{e_\pi(x)},$$

where $e_i(x)$ is the exponent of p_i in $g(x)$. Then

$$\prod_{x \in J} g(x) = \prod_{i=1}^{\pi} p_i^{\sum_{x \in J} e_i(x)}.$$

This is a square if and only if

$$\sum_{x \in J} e_i(x) \equiv 0 \bmod 2 \quad (i = 1, 2, \ldots, \pi). \tag{2.2}$$

Since $|T| = t > \pi$, there exists a non–trivial solution of the linear system (2.2) of equations over GF(2). A solution can be found using Gaussian elimination. This yields at least $t - \pi$ useful subsets J.

3. The Multi-Polynomial Quadratic Sieve

Dixon's algorithm does not tell us how to find efficiently the set T. Building on previous work of Kraitchik [Kra29], Lehmer and Powers [LP31], Morrison and Brillhart [MB75], and Schroeppel, C. Pomerance [Pom82] introduced the *quadratic sieve algorithm.* This works with the quadratic polynomial $g(x) = (x + \lfloor \sqrt{n} \rfloor)^2 - n$, where x runs over the integers in $(-n^\epsilon, n^\epsilon)$, so that $g(x) = \mathcal{O}(n^{1/2+\epsilon})$. With this $g(x)$ the set T may be built up, where some of the numbers $g(x)$ can be factored completely by a cheap sieve process because $g(x)$ is a polynomial (this is much more efficient than trial division or any other factoring method). We could also use a sieve process in Dixon's algorithm if we choose random numbers x in an arithmetic progression like e.g. $x, x+1, x+2, \ldots$ However, in practice this single polynomial $g(x)$ (or an arithmetic progression in Dixon's algorithm) does not give rise to a sufficiently large set T (with $t > \pi$ elements) in a reasonable amount of time. The reason for this is that the interval $(-n^\epsilon, n^\epsilon)$ is large when n is large and since $g(x) = \mathcal{O}(n^{1/2+\epsilon})$ (which is large), many numbers $g(x)$ are not likely to factor over a set of small primes. P.L. Montgomery found an efficient way to use *several* polynomials (thus introducing a simple way to run the algorithm *in parallel*). The numbers x can be taken from much smaller intervals rather than from one single very large interval. The average polynomial values then are smaller than the average value of g and are thus more likely to factor over small primes than the $g(x)$–values. If all the numbers in a small interval have been considered, then we can pass to a next polynomial and try again. We describe here the resulting *multi–polynomial quadratic sieve* method. (We remark that Davis and Holdridge [DH83] had a multi–polynomial version before Montgomery came up with his idea of how to choose the polynomials. In fact, Montgomery's method is based on that of Davis and Holdridge.)

Suppose that we have integer numbers x, $U(x)$, $V(x)$, and $W(x)$ such that

$$U^2(x) \equiv V^2(x) W(x) \bmod n \text{ for all integers } x \in \mathbf{Z}. \tag{3.3}$$

If $J \subset \mathbf{Z}$ is a finite subset such that $\prod_{x \in J} W(x)$ is a square, then we can take

$$X = \prod_{x \in J} V(x) \sqrt{\prod_{x \in J} W(x)}, Y = \prod_{x \in J} U(x).$$

In practice we choose

$$\begin{aligned} U(x) &= a^2 x + b, \\ V(x) &= a, \\ W(x) &= a^2 x^2 + 2bx + c, \end{aligned}$$

with $|x| \leq M$ (where M is a parameter we choose beforehand) and where a, b and c are integers satisfying the following conditions [Bre89, p. 117]:

$$a^2 \approx \sqrt{2n}/M, \tag{3.4}$$

$$b^2 - n = a^2 c, \tag{3.5}$$

$$|\, b \,| < a^2/2. \tag{3.6}$$

In the next section we describe how a, b and c are to be calculated.

$W(x)$ plays the role of $g(x)$ in Dixon's algorithm. In order to determine the subset J, we choose an upper bound B_1 for the primes. We want to have many $W(x)$–values that consist of primes $\leq B_1$. However, only roughly half of the primes below B_1 can occur as a prime divisor of $W(x)$. Namely, if a prime p divides $W(x)$, then $p \mid a^2 W(x)$ and thus $p \mid (a^2 x + b)^2 - n$ which means that n is a quadratic residue modulo p. This leads to the definition of the **factor base** $\mathcal{F}$:

$$\mathcal{F} = \{\text{prime powers } q = p^k \leq B_1 \mid (\tfrac{n}{p}) = 1\}.$$

(Of course, a prime can divide $W(x)$ more than once, so we also have to account for prime powers.) Note that $\mathcal{F}$ is independent of the choices of a, b and c, so we can use the same factor base for every proper choice of a, b and c.

Since $W(x)$ is more likely to be divisible by small primes than by large primes, it is advantageous that the factor base contains many small primes. We can construct such a factor base to multiply the number n to be factored by a suitable small number, the so–called **multiplier**, and factor the product rather than n [PST88, p. 391].

For a given $q \in \mathcal{F}$ the values of x for which q divides $W(x)$ can be found as follows. Compute the solution $t = t_q$ of the congruence equation

$$t^2 \equiv n \bmod q,\ 0 < t \leq q/2$$

(see [Rie85, pp. 212 and 287–288]). This has to be done only once during the algorithm. Now, if $q \mid W(x_0)$, then $q \mid (a^2 x_0 + b)^2 - n$ and thus

$$x_0 \equiv a^{-2}(\pm t_q - b) \bmod q, \tag{3.7}$$

provided that $\gcd(a,q)=1$. This is guaranteed by the choice of a (see Section 4). For each proper choice of a we compute $a^{-2} \bmod q$ for all $q \in \mathcal{F}$. In the next section we describe how these computations can be done. Furthermore, since $W(x)$ is a quadratic polynomial, $q \mid W(x_0 + lq)$, $l \in \mathbf{Z}$. So we can calculate efficiently the places where an element of $\mathcal{F}$ divides the $W--$values. This idea originated from Schroeppel.

Define the **report threshold** RT as the average of $\log|W(x)|$ on the interval $[-M,M]$, which is approximately $\log(\frac{M}{2}\sqrt{n/2})$. Initialize a **sieve array** $SI(-M,M)$ to zero and **sieve** with each $q=p^k \in \mathcal{F}$, i.e., add $\log p$ to $SI(x_0+lq)$ for all $l \in \mathbf{Z}$ such that x_0+lq is in the interval $[-M,M]$. For those numbers x for which $SI(x) \geq RT$, $W(x)$ is a good candidate for fully factoring over the factor base. In general, the time spent on sieving takes more than 85% of the total computing time.

Since sieving with small primes is expensive, it is customary *not* to sieve with the primes and prime powers $\leq QTHRES$, where $QTHRES$ is some suitably chosen threshold value. In order not to loose $W(x)$–values divisible by such small primes, the report threshold RT will be lowered by the amount $\sum_{p^k \leq QTHRES} \log p$. After the sieve step and the selection of those x for which $SI(x) \geq RT$, the prime factors of the corresponding $W(x)$ are found by comparing, for all $q \in \mathcal{F}$, x with the two values of x_0 in (3.7) (which are computed and stored after the factor base has been computed). In this way, $W(x)$–values divisible by one or more of the small primes by which we have "forgotten" to sieve, are not lost. If $QTHRES$ is suitably chosen, this can save a considerable amount of sieve time. This refinement of MPQS is known as the **small prime variation**.

4. Efficient Calculation of The Polynomials

Choose integers r and k such that $1 < k < r$ (typical choices are e.g. $r=30$ and $k=3$). Generate primes $g_1, g_2, \ldots, g_r$, the so–called ***g*–primes**, such that

(i) $g_i \approx (\frac{\sqrt{2n}}{M})^{1/(2k)}$,
(ii) $(\frac{n}{g_i}) = 1$,
(iii) $\gcd(g_i, q) = 1$

for $i = 1,2,\ldots,r$ and for all $q \in \mathcal{F}$. Let a be the product of k g–primes:

$$a = g_{i_1} \cdot g_{i_2} \cdots g_{i_k},$$

with $1 \leq i_1 < i_2 < \cdots < i_k \leq r$. Because of (i), this a satisfies condition (3.4).

Let b_i be a solution of the congruence equation

$$t^2 \equiv n \bmod g_i^2,$$

$(i = 1, 2, \ldots, r)$. Solve the system of congruence equations (for a specific choice of the signs)

$$\begin{aligned} x &\equiv b_{i_1} \bmod g_{i_1}^2 \\ x &\equiv \pm b_{i_2} \bmod g_{i_2}^2 \\ &\vdots \\ x &\equiv \pm b_{i_k} \bmod g_{i_k}^2 \end{aligned} \tag{4.8}$$

by means of the Chinese Remainder Theorem. Let b be the solution of this system of equations. Then it follows that $b^2 \equiv n \bmod a^2$ so that condition (3.5) holds with $c = (b^2 - n)/a^2$. If $b \geq a^2/2$, then we replace b by $b - a^2$ in order to satisfy condition (3.6). Since there are 2^{k-1} possible combinations of signs in (4.8), the number of polynomials that can be calculated with one set of r g–primes and a fixed k is $2^{k-1}\binom{r}{k}$.

If a new a has to be chosen then new sieve numbers x_0 subject to (3.7) have to be computed. Since $a = g_{i_1} g_{i_2} \ldots g_{i_k}$, we can use

$$a^{-2} \bmod q = g_{i_1}^{-2} g_{i_2}^{-2} \ldots g_{i_k}^{-2} \bmod q.$$

Therefore, with the generation of the g–primes we also compute and store the numbers $g_i^{-2} \bmod q$, $(i = 1, 2, \ldots, r)$, for all the prime powers q in the factor base.

For a fixed a, Alford and Pomerance [AP] developed a method to compute iteratively all the other values of b (and thus c) from a given initial value of b (see also the work of Peralta [Per]). They also pointed out how the two solutions in the interval $[0, q)$ of the congruence equation $W(x) \equiv 0 \bmod q$ can be calculated from the zeros mod q of a "previous" polynomial. MPQS provided with this improvement is called the **self initializing variation** of MPQS. This variation has the advantage that it can change polynomials rapidly and hence a shorter sieve interval can be used. We implemented this variation and it appeared that we gained a speed–up of a few percent of the total computing time. This is not surprising since the initializing part takes no more than five percent of the total time (in our implementation). Furthermore, for larger numbers (of more than 100 decimal digits, say) the self initializing version takes too much memory.

5. The Large Prime Variation of MPQS

The following idea to improve the quadratic sieve algorithm is based on a step in the continued fraction algorithm introduced in 1975 by Morrison and Brillhart [MB75]. This improvement is called the *large prime variation* of MPQS: $W(x)$ is allowed to have a factor $R > B_1$ that is not composed of primes from the factor base. If the cofactor R (after dividing out all factor base primes in $W(x)$) is less than or equal to B_1^2, it must be a prime. In order to restrict the amount of disk space needed for storage of the relations (3.3), we only accept factors $R \leq B_2$, where B_2 is a parameter we choose beforehand. In practice we choose B_2 in such a way that B_2/B_1 is a number

between 10 and 100. We have to lower the report threshold by $\log(B_2)$ in order to find these $W(x)$–values after sieving.

If we have found two $W(x)$–values with the same R, multiplication of the corresponding relations (3.3) yields a relation of the form (3.3) where $W(x)$ only consists of prime powers $q \in \mathcal{F}$ (and R is moved to $V(x)$).

A relation of the form (3.3), where $W(x)$ only consists of primes $q \in \mathcal{F}$, is called a **complete relation**. If $W(x)$ has one prime factor $R \leq B_2$ (and the others are in $\mathcal{F}$), then the relation is called a **partial relation**.

We wish to compute E, the expected number of complete relations coming from a given number of r partial relations. Let $\mathcal{Q} = \{\text{primes } q \mid B_1 < q \leq B_2, (\frac{n}{q}) = 1\}$. The elements of $\mathcal{Q}$ are called **large primes**. Let P_q be the probability that a large prime q occurs in a partial relation. Lenstra and Manasse [LM94] assume that

$$P_q \approx q^{-\alpha} \Big/ \sum_{p \in \mathcal{Q}} p^{-\alpha} \tag{5.9}$$

for some positive constant $\alpha < 1$ that should be determined experimentally. They report that $\alpha \in [\frac{2}{3}, \frac{3}{4}]$ gives a reasonable fit with their experimental results. Denny [Den93, pp. 44–49] takes $\alpha = 0.775$.

From [LM94] it follows that

$$E = r - \#\mathcal{Q} + \sum_{q \in \mathcal{Q}} (1 - P_q)^r.$$

We apply the binomial formula of Newton and use approximation (5.9) to find:

$$E \approx \sum_{i=2}^{r} (-1)^i \binom{r}{i} \left(\sum_{q \in \mathcal{Q}} q^{-\alpha} \right)^{-i} \sum_{q \in \mathcal{Q}} q^{-\alpha i}. \tag{5.10}$$

Since $\pi(t) \sim t/\log t$ as $t \to \infty$, we have

$$\sum_{p \leq x} p^{-u} \approx \int_2^x t^{-u} d(t/\log t)$$

(p prime, $x \in \mathbf{R}_{\geq 2}$, $u \in \mathbf{R}_{>0}$). Hence for $u > 0$ we have

$$\sum_{q \in \mathcal{Q}} q^{-u} \approx \frac{1}{2} \int_{B_1}^{B_2} t^{-u} d(t/\log t). \tag{5.11}$$

To compute the last integral we first use partial integration and then use the substitution $s = (1-u)\log t$. We get

$$\begin{aligned} \int_{B_1}^{B_2} t^{-u} d(t/\log t) &= B_2^{1-u}/\log B_2 - B_1^{1-u}/\log B_1 \\ &\quad + u\{\mathrm{Ei}((1-u)\log B_2) - \mathrm{Ei}((1-u)\log B_1)\} \end{aligned} \tag{5.12}$$

where Ei is the exponential integral defined by $\mathrm{Ei}(x) = \int_{-\infty}^{x}(e^s/s)ds$. Now combine (5.10), (5.11), and (5.12) for the appropriate choices of u to get an approximation for E. In approximation (5.10) we sum from $i = 2$ to $i = 5$ and forget about the higher order terms to get a formula for an approximation of E that we can use in practice (given B_1, B_2, r, and α).

The following examples show that our approximation works well for our experiments if $\alpha = 0.73$. The table below contains a column for the number r of partial relations, the actual number of complete relations derived from these partial relations, and the estimated number of complete relations. By Cx (first column) we denote a composite number with x decimal digits. An approximation of E can be used to *predict* the computing time. We determined α as follows. We wrote a program in Maple that (given α) computes the absolute value of the difference of the actual number of complete relations and the estimated number of complete relations for each of fifteen test numbers we took. Then we summed the fifteen absolute values of the differences. Thus for each α we took, we got a sum of absolute values. It appeared that $\alpha = 0.73$ gave rise to the smallest sum. In the table below we list the results for ten test numbers.

n	$B_1/10^5$	B_2/B_1	r	actual # comp. rel.	estimated # comp. rel.
C75	3	20.0	37472	4790	4966
C80	1	60.0	15918	1121	1209
C80	3	167	68195	4113	4150
C84	8	25.0	96138	10894	11148
C88	5	100	94651	6605	6736
C88	7.5	100	148403	11455	11211
C88	7.5	100	158214	12830	12657
C88	7.5	100	146983	11051	11008
C88	7.5	100	150327	11498	11488
C88	7	100	148016	12116	11827

6. The Double Large Prime Variation

In the large prime variation of MPQS we allow $W(x)$ in (3.3) to have a prime factor R with $B_1 < R \leq B_2$. In the double large prime variation of MPQS we also accept $W(x)$ to have a factor $R \leq B_2^2$ composed of *two* primes $> B_1$. In this case we call such a relation a **partial–partial** relation (pp–relation for short). Now the problem of finding combinations of partial and partial–partial relations that yield a *complete* relation can be formulated as finding cycles in an undirected graph: the vertices are the large primes and two vertices (primes) are connected by an edge if there is a pp–relation in which both primes occur. A partial relation is represented by adding 1 as a vertex to the graph. We consider this partial relation as a pp–relation where one of the large primes is 1. So an edge in the graph corresponds to a partial or partial–partial relation and a cycle corresponds to a set of relations with the following property: if we multiply these relations, then all the large primes in the product occur to an even

power. Hence, for the linear algebra step this set can be viewed as a complete relation. To avoid dependent relations one only has to find the *basic* cycles of the graph.

The number of complete relations coming from the pp–relations is much more difficult to predict than that coming from the partial relations. One has to know how the number of basic cycles in a graph with given vertices varies when edges are added more or less randomly. Having a basic cycle is a monotone increasing property [Bol85, p. 33] that can appear rather suddenly [ER59], [ER60], [ER61]. An algorithm for finding the basic cycles in a graph can be found in [Pat69].

If R is prime then we require $R < B_2$ in order to restrict the total number of relations (in our experience partial relations with $B_2 \leq R < B_1^2$ do not contribute much to the total number of complete relations). If R is composite, its large prime factors can be found, e.g., by using Shanks' SQUFOF algorithm [Rie85, pp. 191–199]. This algorithm has the advantage that most numbers that occur during its execution are in absolute value not larger than $2\sqrt{R}$.

We want to estimate the time that PPMPQS spends on the sieve step for numbers n of about d decimal digits, given B_1, M, B_2, and $QTHRES$. To that end, let

$$\begin{aligned} n_f &= \text{number of elements in the factor base,} \\ n_c &= \text{number of complete relations,} \\ f_1 &= n_c/n_f, \\ n_1 &= \text{number of partial relations,} \\ n_2 &= \text{number of pp–relations,} \\ f_2 &= n_2/n_1, \\ T_s &= \text{sieve time.} \end{aligned}$$

During the sieve step, the numbers n_c, n_1 and n_2 grow (more or less) linearly with the time, so that also the fraction f_1 grows linearly, and f_2 stays more or less constant (after the sieve step has been running for a short time). We observed that the values of the fractions f_1 and f_2, measured after completion of the sieve step, seem to be connected. For example, the following tables give the values of f_1 and f_2 measured for 16 numbers factored on the Cray C90 with $d = 86$, $B_1 = 5 \times 10^5$, $M = 1.5 \times 10^6$, $B_2/B_1 = 20$, and $QTHRES = 40$. For each of the 16 numbers n we computed a multiplier m and factored $m \cdot n$ instead of n. More information about the 16 numbers can be found in the tables in the Appendix.

f_1	0.243	0.244	0.255	0.269	0.275	0.297	0.301	0.310
f_2	5.98	5.79	4.04	3.68	2.75	2.37	2.13	2.29
multiplier	109	37	1	109	1	109	5	29
Table	10(1)	10(3)	10(5)	10(12)	10(10)	10(6)	12(3)	10(8)

f_1	0.320	0.325	0.331	0.346	0.348	0.349	0.352	0.363
f_2	1.70	1.64	1.14	0.906	0.961	0.862	0.760	0.798
multiplier	1	1	1	7	43	1	1	41
Table	12(2)	10(2)	10(7)	10(4)	12(1)	10(9)	8(13)	10(11)

The tables suggest that f_2 is an exponential function of f_1:

$$f_2 = ae^{bf_1}$$

for some constants a and b. Based on the table, we estimated $a = 315$ and $b = -16.5$. Since $\log f_2 = \log a + bf_1$, it follows that $n_c = \frac{1}{b}(\log f_2 - \log a) \cdot n_f$. If u is the time needed to generate one complete relation, we obtain the following approximation for the sieve time T_s:

$$T_s \approx (0.349 - 0.061 \log f_2) \cdot u \cdot n_f. \tag{6.13}$$

We can estimate u and f_2 by letting the program run for a short while, five minutes say. We tested our formula with several 85 and 86 digit numbers on the Cray C90 and it appeared that the estimate works well (actually, the numbers are composite factors of the numbers in the column below "n"; 98 91+ means $98^{91}+1$, 47 67 – means $47^{67}-1$ etc.):

n	mult.	Table	u (sec)	f_2	n_f	actual T_s (h)	appr. (6.13)
98 91+	19	8(2)	5.140	1.1945	20741	9.8	10.0
56 96+	1	8(12)	4.226	1.0866	24641	10.0	9.94
80 58+	1	8(3)	4.518	0.7646	20744	9.8	9.50
39 111+	1	8(5)	3.357	1.4378	20930	6.0	6.37
47 67 –	5	12(3)	8.785	2.1364	20911	15.4	15.4

Consequently, the approximation (6.13) can be used to obtain a good estimate of T_s in the PPMPQS–algorithm for numbers of about the same size, and fixed parameters B_1, M, B_2, and $QTHRES$. For numbers in another range, or if we wish to change the parameters, some experiments have to be done to determine the total sieve time under these new conditions, by which the coefficients in (6.13) can be estimated.

In order to test the dependency of T_s on B_2, we carried out the *complete* sieve step of PPMPQS for the 80–digit number:

$$\text{C80} = (75^{64}+1)/(2 \cdot 224914177 \cdot 151113908786421917036806943723393) =$$

$$1484463729\ 7924826822\ 3924402812\ 7205475762$$
$$2335589237\ 4279886592\ 8124925295\ 6234072833$$

(having the two prime factors 687990387865123193888213509255 69
and 215768091527974049646247615957101365677594246657)

on the Cray C90, with $B_1 = 10^5$, $M = 3 \times 10^6$, $QTHRES = 50$ fixed, and for various different values of B_2. In the partial relations we accepted the large prime R to be $< B_1^2$. (We get these for free because $R < B_1^2$ implies that R is prime.) For $B_1 = 10^5$ the number of elements in the factor base is 4806. The sieving was continued until the total number of complete relations (including those generated by the partial relations and the partial–partial relations) surpassed this number (measuring the total number of complete relations obtained so far is done only at selected points in our program, so the *actual* total number of complete relations usually is somewhat larger than the number of elements in the factor base). The column $n_{c,1}$ gives the number of complete relations generated by the partial relations (counted in the previous column n_1), and the column $n_{c,2}$ gives the number of complete relations generated by combining the partial relations (with different large primes) and the pp–relations (counted in the previous column n_2).

B_2/B_1	T_s (h)	n_c+ $+n_{c,1}+n_{c,2}$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$
30	8.64	4818	1036	129318	1661	29143	2121
60	7.06	4859	871	117532	1249	51929	2739
100	6.49	4870	775	109506	1025	76324	3070
200	6.02	4819	685	99474	795	123001	3339
400	5.67	4850	618	91332	634	193278	3598
600	5.71	4844	578	87265	568	243015	3698
800	5.62	4869	563	84926	531	291177	3766
1000	5.75	4843	546	83082	501	333726	3796
1600	6.19	4845	521	79960	464	445526	3860

As we increase B_2/B_1, the program generates more partial–partial relations and less complete and less partial relations in a given, fixed amount of sieve time. For $30 \leq B_2/B_1 \leq 400$, the gain in complete relations ($n_{c,2}$) generated by the pp–relations (n_2) more than sufficiently compensates for the loss of complete relations directly found by the sieve (n_c) and the loss of complete relations ($n_{c,1}$) generated by the partial relations (n_1). As a result, the total sieve time T_s *goes down*. For $B_2/B_1 > 1000$, however, the increase in size of the large primes in the partial and partial–partial relations is responsible for a decrease in the number of complete relations derived from these relations, and also the time that SQUFOF needs to find the two large primes in a pp–relation increases, so now the resulting total sieve time *increases*. Consequently, the minimal sieve time is reached if we choose B_2/B_1 in the interval $400 < B_2/B_1 < 1000$. In that interval the total sieve time is only slightly varying. We conclude that, in order also to minimize the amount of *memory* for storage of the relations, the optimal choice of B_2/B_1 is about 400.

7. Implementation and experiments

For our PMPQS–experiments we used the implementation described in [RLW89]. Almost all our subroutines are written in Fortran.

We have originally implemented the PPMPQS–algorithm on a supercomputer like the Cray C90 vectorcomputer. We used the same implementation on Silicon Graphics workstations (although we now have written a program especially designed for workstations). The sieve operations (i.e., additions of $\log p$ to an element of the sieve array) are done in 64–bits floating–point arithmetic on Cray and in 32–bits on SGI. The maximum speed we obtained (in millions of sieve operations per second) was 3.3 on the Silicon Graphics, 110 on the Cray Y–MP [RLW91] and 270 on the Cray C90. The maximum speed was 5.7 when we used the workstation version of our program. We used a package of Winter in order to carry out multi–precision integer arithmetic. For the large prime R occurring in the partial relations we accepted R with $B_1 < R < B_2$ and those with $B_2 \leq R < B_1^2$ were rejected. We have implemented Paton's cycle finding algorithm [Pat69] and used it as a preprocessing step for the Gaussian elimination step in PPMPQS. An algorithm for just *counting* (but not finding) the basic cycles ([LM94, pp. 789–790] and [Den93, pp. 61–64]) was implemented by us as a tool to check during the sieve part of PPMPQS whether sufficiently many relations (complete, partial, and partial–partial) were collected. The method used to do the Gaussian elimination (mod 2) is described in [PW84]. The elements of the bit–array are packed in words of 64 bits (on the Cray computers) or 32 bits (on the Silicon Graphics). This allows the use of XOR–ing (exclusive or) with the column vectors of the array, which is very efficient. The total Gaussian elimination step (including finding basic cycles) takes less than 0.6% of the total work of the PPMPQS–algorithm.

In order to compare PMPQS with PPMPQS we have run our implementations of these algorithms on the Cray C90 for the 71–digit number

$$\mathrm{C71} = (10^{71} - 1)/9$$

and for the 87–digit cofactor

$$\mathrm{C87} = 1360245\ 9257583786\ 3939661047\ 9463908049\ 3042354284$$
$$1197990430\ 2204441489\ 2390146207\ 9070640121$$

of $72^{99} + 1$. For C71, four experiments with different combinations of B_1, B_2/B_1, and M were carried out where in the second, third and fourth experiment only one of the three parameters was changed compared with the previous experiment. The value of $QTHRES$ was kept fixed on 40. For C80 treated in the previous section with PPMPQS, we made a comparing run with PMPQS for $B_1 = 10^5$, $M = 3 \times 10^6$, $QTHRES = 50$, and $B_2/B_1 = 400$ (the optimal choice for PPMPQS). The results are given in Table 1.

For C71, the parameter choice $B_1 = 3 \times 10^5$, $B_2/B_1 = 20$, and $M = 5 \times 10^5$ yields a somewhat smaller sieve time for PPMPQS (0.55 CPU hours) than for PMPQS (0.58),

but if we allow more memory use by choosing $B_1 = 6 \times 10^5$ and $M = 2.5 \times 10^6$ (and $B_2/B_1 = 40$), then PMPQS beats PPMPQS (0.29 vs. 1.21). Increasing the length of the sieve interval (M from 5×10^5 to 2.5×10^6) particularly improves the efficiency of PMPQS (and, to a lesser extent, of PPMPQS). For C87, with the parameter choice $B_1 = 5 \times 10^5$, $B_2/B_1 = 20$, and $M = 2.5 \times 10^6$, PPMPQS is faster than PMPQS (11.9 vs. 16.4).

We conclude that for our implementations PPMPQS can beat PMPQS for numbers of more than 80 (say) decimal digits, but the cross–over point strongly depends on the amount of available central memory. For practical reasons (like throughput) it can be profitable to reduce the size of a sieve job on the Cray C90, so even though such a computer has a very large central memory, it is still worthwhile to restrict the size of the upper bound on the primes in the factor base and to have an efficient implementation of a memory–economic method like PPMPQS. This aspect is even more important on workstations, particularly when there are primary and secondary cache memories (as is usual on workstations).

	PMPQS–results	**PPMPQS–results**
C71		
$B_1 = 3 \times 10^5$	$n_c = 10204$	$n_c = 5063$
$n_f = 12979$	$n_1 = 17993 \rightarrow n_{c,1} = 2784$	$n_1 = 36468 \rightarrow n_{c,1} = 4709$
$B_2/B_1 = 20$		$n_2 = 42617 \rightarrow n_{c,2} = 3400$
$M = 5 \times 10^5$	$\boldsymbol{T_s} = \mathbf{0.58}$ hrs.	$\boldsymbol{T_s} = \mathbf{0.55}$ hrs.
C71		
$B_1 = 6 \times 10^5$	$n_c = 20827$	$n_c = 10868$
$n_f = 24510$	$n_1 = 23794 \rightarrow n_{c,1} = 3703$	$n_1 = 68019 \rightarrow n_{c,1} = 8383$
$B_2/B_1 = 20$		$n_2 = 70395 \rightarrow n_{c,2} = 5389$
$M = 5 \times 10^5$	$\boldsymbol{T_s} = \mathbf{0.56}$ hrs.	$\boldsymbol{T_s} = \mathbf{0.96}$ hrs.
C71		
$B_1 = 6 \times 10^5$	$n_c = 20312$	$n_c = 9817$
$n_f = 24510$	$n_1 = 30399 \rightarrow n_{c,1} = 4209$	$n_1 = 80017 \rightarrow n_{c,1} = 7390$
$B_2/B_1 = 40$		$n_2 = 132290 \rightarrow n_{c,2} = 7412$
$M = 5 \times 10^5$	$\boldsymbol{T_s} = \mathbf{0.55}$ hrs.	$\boldsymbol{T_s} = \mathbf{1.28}$ hrs.
C71		
$B_1 = 6 \times 10^5$	$n_c = 20196$	$n_c = 9803$
$n_f = 24510$	$n_1 = 31034 \rightarrow n_{c,1} = 4359$	$n_1 = 81612 \rightarrow n_{c,1} = 7499$
$B_2/B_1 = 40$		$n_2 = 138147 \rightarrow n_{c,2} = 7969$
$M = 2.5 \times 10^6$	$\boldsymbol{T_s} = \mathbf{0.29}$ hrs.	$\boldsymbol{T_s} = \mathbf{1.21}$ hrs.
C80		
$B_1 = 10^5$	$n_c = 1580$	$n_c = 618$
$n_f = 4806$	$n_1 = 49143 \rightarrow n_{c,1} = 3229$	$n_1 = 91332 \rightarrow n_{c,1} = 634$
$B_2/B_1 = 400$		$n_2 = 193278 \rightarrow n_{c,2} = 3598$
$M = 3 \times 10^6$	$\boldsymbol{T_s} = \mathbf{13.4}$ hrs.	$\boldsymbol{T_s} = \mathbf{5.67}$ hrs.
C87		
$B_1 = 5 \times 10^5$	$n_c = 9902$	$n_c = 7009$
$n_f = 20838$	$n_1 = 70029 \rightarrow n_{c,1} = 10940$	$n_1 = 63089 \rightarrow n_{c,1} = 8220$
$B_2/B_1 = 20$		$n_2 = 57513 \rightarrow n_{c,2} = 5620$
$M = 2.5 \times 10^6$	$\boldsymbol{T_s} = \mathbf{16.4}$ hrs.	$\boldsymbol{T_s} = \mathbf{11.9}$ hrs.

Table 1: Comparison of PMPQS with PPMPQS for C71, C80, and C87

Furthermore, with our PMPQS–program we have factored the 99–digit cofactor

168483084 9783397621 1530436039 9726602530 8430041776
9257490404 3633682183 8963842217 5595211200 8347771913

of the *more wanted number* from the Cunningham table with code "2,914M C133". This "C133" is the composite number of 133 decimal digits $(2^{457} + 2^{229} + 1)/(5 \times 71293)$; Peter Montgomery had found the 34–digit prime factor

6196333979234679466021864314534473

of this number with ECM. and left the 99–digit composite cofactor. We decomposed it into the product of the 49– and a 50–digit primes:

5845296257595668545524969937697507923682374822769
and
28823703291241135239378075616078003806433692452377,

with the help of an eight processor IBM 9076 SP1, and 69 Silicon Graphics workstations (63 at CWI and 6 at Leiden University). The factor base size was 56976 with $B_1 = 1.5 \times 10^6$, $B_2/B_1 = 50$, $M = 2 \times 10^6$, and $QTHRES = 30$. Parallel processing with good load balancing was effectuated by assigning different polynomials to different workstations. The total amount of sieve time was about 19,500 workstation CPU–hours. The physical time for this factorization was about four weeks. This means that we consumed about 40% of the total CPU–capacity of these workstations during that period (assuming that they all are equally fast: in fact, an RS 6000 processor of the IBM SP1 sieved about twice as fast as an SGI workstation). The Gaussian elimination step was carried out on a Cray C90; it required about 0.5 Gbytes of central memory, and one hour CPU–time.
As a comparison with a vectorcomputer [RLW91], on a Cray Y–MP we factored a 101–digit more wanted Cunningham number with PMPQS in 475 CPU–hours, using $B_1 = 1300000$, with 50179 primes in the factor base, $B_2/B_1 = 50$, $M = 4.5 \times 10^6$, and $QTHRES = 40$ (our PMPQS–implementation runs about twice as fast on the Cray C90 as on the Cray Y–MP).
As a comparison with PPMPQS, from the PPMPQS results listed in Table 1 we estimate (based on the assumption that the computing time of PPMPQS approximately doubles if the size of the number increases by three decimal digits) that we would roughly need 10,000 CPU–hours of an SGI workstation to factor the 99–digit cofactor of 2,914M C133, yielding a speed–up factor of about 2 compared to PMPQS. If we would take a factor 1.64 (see the next paragraph) instead of 2, then the time would be less than 4000 CPU–hours.

In the Appendix (Tables 2 – 15) we list the results of our experiments with PPMPQS on eight numbers in the 66 – 83 digit range on an SGI workstation, and 73 numbers

in the 67 – 88 digit range on a Cray C90 vectorcomputer. Most of these numbers fill gaps in the table [BR92], and are difficult to factor (they were tried before with ECM without success). We have varied the different parameters B_1, B_2/B_1, and M on different numbers (but not in a very systematic way) and kept $QTHRES = 40$ fixed. We observe that the average CPU–time for numbers in the 67 – 88 digit range varies between 0.4 and 12 CPU–hours, so that increasing the number of digits by three gives an increase of the sieve time by a factor of about 1.64. This is smaller than the factor of 2 that is usually observed for PMPQS.

ACKNOWLEDGEMENTS

We thank Arjen Lenstra, Walter Lioen and Rob Tijdeman for reading the paper and for suggesting several improvements. Walter Lioen helped us with the implementation of our programs on SGI workstations and on Cray vectorcomputers. We gratefully acknowledge the Dutch National Computing Facilities Foundation NCF for the provision of computer time on Cray Y–MP and Cray C90 vector processors. Finally, we acknowledge the help of IBM and the Academic Computing Center Amsterdam (SARA) for providing access to and CPU–time on the IBM SP1 at SARA.

REFERENCES

[AGLL] Derek Atkins, Michael Graff, Arjen K. Lenstra, and Paul C. Leyland. THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE. In *Proceedings of Asiacrypt '94*, Lecture Notes in Computer Science, Berlin. Springer–Verlag. To appear.

[AP] W.R. Alford and C. Pomerance. Implementing the self initializing quadratic sieve on a distributed network. To appear.

[Bol85] B. Bollobás. *Random Graphs.* Academic Press, 1985.

[BR92] R.P. Brent and H.J.J. te Riele. Factorizations of $a^n \pm 1$, $13 \leq a < 100$. Technical Report NM–R9212, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, June 1992.
Update 1 to this report has appeared as CWI Report NM–R9419, September 1994, with P.L. Montgomery as third author.

[Bre89] David M. Bressoud. *Factorization and Primality Testing.* Springer–Verlag, New York, NY, 1989. Undergraduate Texts in Mathematics.

[DDLM94] T.F. Denny, B. Dodson, A. K. Lenstra, and M. S. Manasse. On the factorization of RSA-120. In D.R. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 166–174, Berlin, 1994. Springer–Verlag.

[Den93] T.F. Denny. Faktorisieren mit dem Quadratischen Sieb. Master's thesis, Universität des Saarlandes, Saarbrücken, 1993.

[DH83] J.A. Davis and D.B. Holdridge. Factorization using the quadratic sieve

algorithm. Technical Report Sandia Report Sand 83–1346, Sandia National Laboratories, Albuquerque, New Mexico, 1983.

[ER59] P. Erdős and A. Rényi. On random graphs I. *Publ. Math. Debrecen*, 6:290–297, 1959.

[ER60] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, 5:17–61, 1960.

[ER61] P. Erdős and A. Rényi. On the evolution of random graphs. *Bull. Inst. Int. Statist. Tokyo*, 38:343–347, 1961.

[Kra29] M. Kraitchik. *Recherches sur la théorie des nombres*, volume II. Gauthier–Villar, Paris, 1929.

[LL93] A.K. Lenstra and H.W. Lenstra, Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer–Verlag, Berlin, 1993.

[LM94] A.K. Lenstra and M.S. Manasse. Factoring with Two Large Primes. *Mathematics of Computation*, 63:785–798, 1994.

[LP31] D.H. Lehmer and R.E. Powers. On factoring large numbers. *Bull. Amer. Math. Soc.*, 37:770–776, 1931.

[MB75] M.A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Mathematics of Computation*, 29:183–205, 1975.

[Pat69] K. Paton. An Algorithm for Finding a Fundamental Set of Cycles of a Graph. *Comm. ACM*, 12:514–518, 1969.

[Per] René Peralta. Implementation of the Hypercube Variation of the Multiple Polynomial Quadratic Sieve. Submitted for publication.

[Pom82] Carl Pomerance. Analysis and comparison of some integer factoring algorithms. In H.W. Lenstra, Jr. and R. Tijdeman, editors, *Computational methods in number theory*, Mathematical Centre Tracts 154/155, pages 89–139. Mathematisch Centrum, Amsterdam, 1982.

[Pom85] Carl Pomerance. The Quadratic Sieve Factoring Algorithm. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology, Proceedings of EUROCRYPT 84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, Springer–Verlag, New York, 1985.

[PST88] Carl Pomerance, J.W. Smith, and Randy Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM J. Comput.*, 17:387–403, 1988.

[PW84] D. Parkinson and W. Wunderlich. A compact algorithm for Gaussian elimination over GF(2) implemented on highly parallel computers. *Parallel Comput.*, 1:65–73, 1984.

[Rie85] Hans Riesel. *Prime Numbers and Computer Methods for Factorization.* Birkhäuser, Boston, etc., 1985.

[RLW89] H.J.J. te Riele, W.M. Lioen, and D.T. Winter. Factoring with the quadratic sieve on large vector computers. *J. Comp. Appl. Math.*, 27:267–278, 1989.

[RLW91] Herman te Riele, Walter Lioen, and Dik Winter. Factorization beyond the googol with MPQS on a single computer. *CWI Quarterly*, 4:69–72, March 1991.

[Sil87] R.D. Silverman. The Multiple Polynomial Quadratic Sieve. *Math. Comp.*, 48:329–339, 1987.

Appendix

We factored many numbers (with PPMPQS) in order to update the table of Richard Brent and Herman te Riele [BR92]; we also factored some numbers of the form $a^n \pm 1$ that are *outside* the range covered by [BR92]. We give two tables with information about the most important quantities involved. For the sake of clarity we explain these quantities (again):

d = $^{10}\log$(number to be factored)
B_1 = upper bound for the primes in the factor base,
B_2 : B_2^2 is the upper bound for the input R to SQUFOF (yielding a pp–relation),
n_f = number of primes in the factor base,
M : $[-M, M]$ is the sieve interval,
n_c = number of complete relations we find immediately,
n_1 = number of partial relations,
$n_{c,1}$ = number of complete relations coming from partial relations,
n_2 = number of pp–relations,
$n_{c,2}$ = number of complete relations coming from pp–relations,
T_s = sieve time in CPU–hours.

The small prime variation parameter $QTHRES$ was kept fixed on 40.

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^5$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	65.56	0.8	3911	11.25	2	1493	9753	1715	4102	710	5.8
2	66.17	0.8	3908	10	1.5	1452	9433	1766	3697	693	4.8
3	66.83	0.8	3984	10	2	1214	9952	2139	4238	637	14.2
4	74.15	3	13045	20	6	4840	37472	4790	26391	3424	55.4
5	78.76	3	12898	30	5	4444	44583	5104	29653	3355	123
6	81.54	5	20812	20	5	7992	63176	8471	33614	4351	173
7	81.70	4.5	18961	20	4.5	6796	55435	7229	38950	4942	198
8	82.89	5	20861	20	8	7387	62346	8229	40035	5250	273

Table 2: Parameter choices and timings for numbers in the 66–83 decimal digits range, factored with PPMPQS on a SGI workstation

nr.	number	prime factor
1	C66 77 53+ = P31 * P35	P31 = 8508101816450689975658227843439
2	C67 58 88+ = P26 * P41	P26 = 62057338333442627487392257
3	C67 62 89– = P31 * P37	P31 = 3916898265747514256035560079891
4	C75 70 87+ = P29 * P46	P29 = 56476537654063551106920429541
5	C79 72 118+ = P38 * P42	P38 = 160594909070093212254803474806878324 41
6	C82 84 71+ = P33 * P50	P33 = 133184106044570646620234096956423
7	C82 80 99+ = P32 * P51	P32 = 11935171798229644025656192643827
8	C83 92 87+ = P23 * P61	P23 = 10127992394070979564027

Table 3: Factorizations of the numbers in Table 2

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^5$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	66.80	2	8881	30	25	2945	27673	2762	31855	3347	0.36
2	68.74	2.5	11086	20	5	3988	30107	3631	27746	3476	0.46
3	74.20	3.16	13623	20	6.31	4921	38371	4889	29855	3822	1.22
4	74.51	3.16	13625	20	6.31	5503	42284	5844	17604	2297	1.16
5	74.69	1	4790	60	5	1005	17630	1320	29502	2465	2.42
6	74.83	3	12892	17	25	4697	37137	5388	19447	2820	1.20
7	74.92	2.5	11086	36	25	3339	35899	3335	43531	4382	1.91
8	77.37	5	20972	20	25	7152	54706	6444	60361	7393	1.84
9	77.56	5	20888	30	30	7518	65930	6980	60042	6453	1.41
10	79.04	5	20597	30	30	6596	61563	6201	76295	7828	2.43

Table 4: Parameter choices and timings for numbers in the 67–80 decimal digits range, factored with PPMPQS on a Cray C90 vectorcomputer

nr.	number	prime factor
1	C67 89 64+ = P24 * P44	P24 = 153316525308739316934017
2	C69 50 122+ = P30 * P40	P30 = 276832194921994230575098974137
3	C75 101 41+ = P32 * P43	P32 = 215872277033288219520305273145O7
4	C75 110 41+ = P16 * P25 * P35	P16 = 3850561614882023
		P25 = 7797598239853074057655219
5	C75 110 47+ = P24 * P51	P24 = 728424414211828929294823
6	C75 35 147+ = P35 * P40	P35 = 860524394110991401680708629331438O1
7	C75 53 59– = P24 * P51	P24 = 943970114867362247759443
8	C78 19 165+ = P28 * P50	P28 = 2481953419044452308291386601
9	C78 51 102+ = P30 * P48	P30 = 459028910227193494771112394289
10	C80 86 58+ = P33 * P47	P33 = 129094951090723152084884804969621

Table 5: Factorizations of the numbers in Table 4

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^6$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	79.17	4	16927	20	1	5619	45717	5584	48399	6279	3.29
2	79.17	5	20895	20	3	6457	72272	11650	37114	2802	2.68
3	79.39	3	13001	166.7	3	3739	68195	4113	72708	5157	2.27
4	79.87	3	13011	166.7	3	3323	64308	3624	91150	6084	3.41
5	80.86	5	20819	20	6	6925	57619	7050	55281	6877	3.36
6	82.82	6	24598	20	2.5	8522	68723	8378	59901	7713	4.82
7	82.91	7	28413	20	2.5	11451	87010	11694	40636	5271	4.38
8	83.66	8	32104	25	2.5	11419	96138	10894	85260	9807	5.46
9	83.98	7	27980	25.7	2.5	10594	93766	11327	51233	6070	6.59

Table 6: Parameter choices and timings for numbers in the 80–84 decimal digits range, factored with PPMPQS on a Cray C90 vectorcomputer

nr.	number	prime factor
1	C80 75 64+ = P32 * P48	P32 = 687990387865123193888821350925569
2	C80 59 85– = P36 * P44	P36 = 192052183634195717382812875959337681
3	C80 76 123+ = P28 * P53	P28 = 1602475801546350975094860307
4	C80 84 87– = P40 * P41	P40 = 2904043752413366850400636076474517615769
5	C81 18 103– = P35 * P47	P35 = 15936754604932361311519937275763087
6	C83 82 68+ = P40 * P43	P40 = 9241855378580566956862595601843404638609
7	C83 93 71+ = P34 * P50	P34 = 18715988916952079528029392484745 57
8	C84 89 67– = P41 * P44	P41 = 1734546038685607265716888388635135765 1503
9	C84 74 91– = P31 * P54	P31 = 6300454649733691099786120178647

Table 7: Factorizations of the numbers in Table 6

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^6$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	84.10	5	20713	20	2.5	6175	51592	5808	76377	8732	5.6
2	84.35	5	20741	20	2.5	6865	60444	7638	72201	6256	9.8
3	84.80	5	20744	20	2.5	7809	57576	7457	44022	5481	9.8
4	84.87	5	20790	20	2.5	7153	61546	7923	43044	5721	8.4
	84.87	5	20790	40	2.5	6412	73385	6960	75133	7427	8.4
5	84.92	5	20930	20	2.5	6434	52315	5865	75217	8614	6.0
6	84.99	5	20749	20	2.5	7106	58607	7259	53507	6389	6.8
7	85.02	5	20675	20	2.5	6982	61080	7920	64746	5774	9.8
8	85.02	5	20792	20	2.5	6679	58782	7268	81258	6853	11.
9	85.05	5	20887	20	2.5	7754	65228	8990	46265	4178	8.4
10	85.11	5	20810	20	2.5	4923	43182	4064	280566	11857	8.4
11	85.11	5	20841	20	2.5	5615	50651	5434	182705	9822	10.7
12	85.12	6	24641	20	2.5	8518	67320	9253	73153	6953	10.0
13	85.12	5	20651	20	1.5	7269	64239	8799	48843	4625	9.5

Table 8: Parameter choices and timings for numbers in the 85–86 decimal digits range, factored with PPMPQS on a Cray C90 vectorcomputer

nr.	number	prime factor
1	C85 69 117+ = P42 * P43	P42 = 553775456930001686459646662784000439421893
2	C85 98 91+ = P39 * P47	P39 = 150856027763097994901861400756223948651
3	C85 80 58+ = P42 * P44	P42 = 587407531780545617292693056474932755332969
4	C85 56 64+ = P43 * P43	P43 = 1120971223480359091305712645673434758493441
5	C85 39 111– = P32 * P54	P32 = 38661901037861787717347412050407
6	C85 77 95– = P34 * P52	P34 = 1254200040785197567017611121581711
7	C86 18 111+ = P35 * P51	P35 = 57095169829153516132919139336069139
8	C86 76 59+ = P39 * P47	P39 = 471586815074704431240140019672222092489
9	C86 20 97+ = P34 * P52	P34 = 26453329120142876693394950899515 67
10	C86 93 99– = P31 * P55	P31 = 3466732593888008254791613360081
11	C86 58 93– = P32 * P54	P32 = 75701865042739143157590250368211
12	C86 56 96+ = P39 * P47	P39 = 232559086557407467762901333407938321409
13	C86 92 84+ = P43 * P43	P43 = 24651527156587484288308809948243436390198 33

Table 9: Factorizations of the numbers in Table 8

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^6$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	85.14	5	20812	20	1.5	5064	43981	4223	263194	11614	10.7
2	85.21	5	20709	20	1.5	6722	56788	6924	92891	7136	8.27
3	85.26	5	20859	20	1.5	5101	44412	4378	256996	11412	11.0
4	85.26	5	20768	20	1.5	7186	63721	8449	57739	5154	12.4
5	85.31	5	20812	20	1.5	5297	45852	4584	185169	10967	7.45
6	85.31	5	20576	20	1.5	6107	55044	6362	130553	8115	13.1
7	85.33	5	20709	20	1.5	6859	60552	7686	68840	6177	11.5
8	85.35	5	20923	20	1.5	6480	55476	6546	127090	7903	10.7
9	85.37	5	20672	20	1.5	7221	62980	8435	54345	5029	9.65
10	85.42	5	20672	20	1.5	5695	50790	5707	139604	9308	10.4
11	85.49	5	20772	20	1.5	7530	63927	8600	51034	4656	11.9
12	85.52	5	20634	20	1.5	5556	50383	5456	185347	9653	13.7
13	85.53	5	20711	20	2.5	5054	43759	4078	270308	11587	11.4

Table 10: Parameter choices and timings for numbers in the 86 decimal digits range, factored with PPMPQS on a Cray C90 vectorcomputer

nr.	number	prime factor
1	C86 67 99– = P34 * P52	P34 = 251520821420628512125495193264146 9
2	C86 13 138+ = P29 * P57	P29 = 54836637716450236990971812089
3	C86 59 89– = P31 * P55	P31 = 2689941424488348023848649808389
4	C86 21 123+ = P39 * P47	P39 = 380770063539669474313312691529545132713
5	C86 38 81– = P36 * P50	P36 = 511662075163970762060417538436484323
6	C86 31 117– = P39 * P47	P39 = 250630033376957433234617073114910871767
7	C86 50 96+ = P35 * P51	P35 = 36774112300765382067961168652800897
8	C86 96 95+ = P28 * P58	P28 = 2418476990688796014581890831
9	C86 24 130+ = P36 * P50	P36 = 684989928644194001785075922656446841
10	C86 93 53+ = P38 * P49	P38 = 19192699869550253389095978550167828173
11	C86 98 59+ = P32 * P55	P32 = 29037047448209810589475647292291
12	C86 80 65+ = P31 * P55	P31 = 3416871674919158699528742801241
13	C86 $8^{2^7} + 7^{2^7}$ = P42 * P43	P42 = 5199759350603466607839860527609770251368 97 (P43 = 65757674240355835167624181741955409969833473)

Table 11: Factorizations of the numbers in Table 10

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^6$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	85.59	5	20797	20	1.5	7244	63668	8524	61191	5044	12.6
2	85.70	5	20712	20	1.5	6637	56910	6862	96694	7226	10.3
3	85.72	5	20911	20	1.5	6311	56349	6710	120384	7895	15.4
4	85.73	6	26392	2	3	16159	24514	9487	3153	749	13.8
5	85.92	6	26363	2	3	16376	24473	9358	7417	631	12.1
6	85.93	3	13041	20	1.5	4117	39602	5212	39395	3713	17.4
7	85.95	3	13011	20	2.5	4255	40517	5390	24478	3366	20.6
8	85.98	5	20756	2.4	2.5	10516	22044	7450	6610	2795	15.7
9	86.04	5	20840	20	2.5	7153	62231	8139	63273	5557	14.0
10	86.13	5	20838	20	2.5	7009	63089	8220	57513	5620	11.9
11	86.16	5	20787	22	2.5	7367	63987	8559	54708	4900	10.8
12	86.18	5	20688	20	2.5	7447	64778	8836	47154	4419	10.7
13	86.22	5	20852	20	2.5	6202	54180	6282	144069	8376	11.6
14	86.22	5	20947	20	2.5	7522	63620	8412	52191	5091	9.35

Table 12: Parameter choices and timings for numbers in the 86–87 decimal digits range, factored with PPMPQS on a Cray C90 vectorcomputer

nr.	number	prime factor
1	C86 23 83– = P38 * P49	P38 = 27736074503263071062950778805992164759
2	C86 76 56+ = P40 * P47	P40 = 4868699568817220592890920460964327586529
3	C86 47 67– = P32 * P55	P32 = 21270964162538089013014983761851
4	C86 67 76+ = P42 * P45	P42 = 315618216027848486834301078445774290254513
5	C86 39 81+ = P37 * P50	P37 = 24430036165666630699892784411335180 59
6	C86 22 95– = P34 * P52	P34 = 962435791906840355509151236741426 1
7	C86 76 117– = P42 * P45	P42 = 606202897105850025527074421945484005533987
8	C86 95 80+ = P38 * P49	P38 = 450897580997918678316374862447596670 41
9	C87 62 65+ = P34 * P53	P34 = 1439106922902522842484110155444391
10	C87 72 99+ = P28 * P59	P28 = 80975407891689909106865888 41
11	C87 92 85– = P32 * P56	P32 = 14285278844357974752432939513571
12	C87 30 95+ = P35 * p52	P35 = 80451911996934444483653727156040931
13	C87 50 100+ = P41 * P46	P41 = 589514788785130719305008867620773920776 01
14	C87 66 96+ = P42 * P46	P42 = 153055732248039041786999207837459270270017

Table 13: Factorizations of the numbers in Table 12

nr.	d	$B_1/10^5$	n_f	B_2/B_1	$M/10^6$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	86.27	5	20978	40	2.5	6773	79489	8184	75416	6035	14.2
2	86.29	5	20797	40	2.5	6387	72868	6909	116000	7520	11.1
3	86.38	5	20754	40	2.5	6881	76861	7638	86915	6253	6.72
4	86.43	5	20920	40	2.5	7177	76854	7706	82085	6054	8.81
5	86.45	5	20631	40	2.5	6329	74485	7262	92362	7046	14.6
6	86.63	5	20902	80	2.5	5806	85167	6249	148784	8876	13.8
7	86.64	6	24404	100	3	7564	124510	9691	89594	7355	13.2
8	86.69	6	24573	100	3	7803	122935	9614	89375	7369	14.1
9	86.70	6	24495	100	3	6571	105037	7010	149698	11272	12.1
10	86.73	6	24538	100	3	7635	120888	9389	99930	7811	11.5
11	86.75	6	24374	100	3	7827	126178	9899	80444	6862	14.7
12	86.82	6	24615	100	3	6532	121187	9864	167590	8507	11.8
13	86.96	6	24658	100	3	7762	116023	8546	126334	8798	7.66
14	87.54	5	20604	100	1	6101	94651	6605	108893	8048	12.1

Table 14: Parameter choices and timings for numbers in the 87–88 decimal digits range, factored with PPMPQS on a Cray C90 vectorcomputer

nr.	number	prime factor
1	C87 19 101– = P25 * P62	P25 = 52456476443168631828545 71
2	C87 33 85+ = P33 * P54	P33 = 249536921989169261065035112257901
3	C87 63 65+ = P42 * P46	P42 = 10841088997442568505957564739184105515 5451
4	C87 42 99– = P33 * P55	P33 = 234373090934137193434426100841739
5	C87 77 67– = P41 * P46	P41 = 75024943244844149373705126243013155715853
6	C87 84 59– = P35 * P53	P35 = 11779548019122302808328920808327631
7	C87 26 129+ = P31 * P57	P31 = 3076814278757622588317626405309
8	C87 33 111– = P38 * P50	P50 = 21457939605898871224437297672972660829
9	C87 86 84+ = P40 * P48	P40 = 1039512269081394539159468072656199331337
10	C87 85 65+ = P40 * P48	P40 = 4645176624103101144238593467706089788481
11	C87 45 85+ = P36 * P52	P36 = 2181360904850689209750606257400202 21
12	C87 87 93+ = P35 * P53	P35 = 65234702723152738657728499902597613
13	C87 45 71+ = P27 * P61	P27 = 69229816187403473081388 1603
14	C88 19 168+ = P42 * P47	P42 = 26168871234858167232514678609739331349 7473

Table 15: Factorizations of the numbers in Table 14