NUMBER THEORETIC and ALGEBRAIC METHODS in COMPUTER SCIENCE

MOSCOW 1993

Editors

Alf van der Poorten
Department of Mathematics, School of MPCE
Macquarie University, Sydney, Australia

Igor Shparlinski
Department of Computing, School of MPCE
Macquarie University, Sydney, Australia

Horst G Zimmer
Fachbereich 9 Mathematik
Universität des Saarlandes, Saarbrücken, Germany

Conference Committee 1992–1993

Horst G Zimmer, Chairman (Germany)
Gerard Cohen (France)
Stephen D Cohen (Scotland)
Joachim von zur Gathen (Canada)
Dima Grigorev (Russia)
Michael Kaminski (Hong Kong)
Gary L Mullen (USA)
Harald Niederreiter (Austria)
Andrew Odlyzko (USA)
Michael Pohst (Germany)
Carl Pomerance (USA)
Alf van der Poorten (Australia)
Claus P Schnorr (Germany)
Igor Shparlinski (Russia)
Sergey Vladuts (Russia)

PREFACE

NTAMCS '93 was an unusual Conference. When the meeting was first conceived, a driving motive was to bring foreign mathematicians and computer scientists to the USSR, thereby to assist in beginning to normalise scientific relationships between the USSR and the so-called Western world. By the time the meeting took place the USSR was no more. The originator of the notion of the meeting, Igor Shparlinski, had succeeded in finding a position at Macquarie University, Sydney, Australia. Thus most of his organisational effort was done remotely, by e-mail. Ultimately, passport problems meant he could not attend the Conference. The present remarks give us an opportunity to give special thanks to Nadja and Alexander Polupanov and to Vjacheslav Rykov who, amongst others, acted for Igor in Moscow and made the meeting possible.

Equally, of course, we are grateful to the mathematicians and computer scientists who agreed to serve on the Organising Committee for the meeting. All the more, we must thank all those who attended and whose participation made the meeting the success it became.

A primary motive of the Conference whose proceedings are represented herein was that it should bring together researchers from areas of computer science and of mathematics that traditionally have been apart, but which nevertheless use similar number theoretic and algebraic methods. An incomplete list of such areas includes cryptography, coding theory, computational algebra and number theory, and numerical analysis. One can readily add to that list of mathematical fields.

Of course we remembered that researchers attack similar problems, sometimes the very same problems, from different points of view. We use different techniques. We use different jargon; one might well say we employ different languages. We emphasise different aspects. Often we do not know each other's results.

We felt certain that a joint meeting would help to acquaint all of us with new and fresh ideas and would stimulate progress in the areas of common interest to us. The contents of this book confirm that, generally, our aims were achieved.

If one wishes, one can separate the papers herein into various categories.

One such category consists of papers which contain new results or give a survey of previously known number theoretic or algebraic results which provide the infrastructure for applications of their field to computer science. An example is the papers of Gary Mullen and Steve Cohen who deal, from different points of view, with the theory of Dickson polynomials over finite fields. These polynomials are especially important for applications to cryptography.

Another category is of papers which contain new ingenious applications of known results. Examples include the papers of 'Red' Alford and Carl Pomerance and of

Johannes Buchmann and Sachar Paulus. These papers deal with two of the most celebrated and important problems of modern computational number theory — integer factorisation and the discrete logarithm problem, respectively.

Yet another category, represented for example by the paper of Vladimir Anashin and that of Isobel McFarlane and Stuart Hoggar, demonstrates the fundamental mathematical nature of principles and methods of computer science.

Various editorial exigencies have not made it possible to include in this volume all the material recommended to us by the referees. We apologise to the authors and even more to you, the readers, for those omissions. The language of NTAMCS '93 was English, today's scientific lingua franca. That was not easy for all our participants* and contributors. We have endeavoured to aid the comprehensibility of the contributions herein with some judicious editing without, however, attempting to change the flavour and rhythm of the original manuscripts.

The other two editors wish to thank the one of us who performed the reTeXing and other final editorial work on this volume. That work was assisted by Dr Ross Moore of the Department of Mathematics, School of MPCE at Macquarie University. Ross is a co-author of the XY-pic macro package used herein to create or change the size of various diagrams and tables. XY-pic is a package for typesetting graphs and diagrams using any one of the various flavours of TeX.

We believe that the present collection of papers displays, on the one hand, the variety of different deep mathematical methods and tools which are potentially or actually applicable to computational problems. On the other hand, it illustrates current trends in modern computer science.

Alfred J van der Poorten
Department of Mathematics
School of Mathematics, Physics, Computing and Electronics
Macquarie University NSW 2109 Australia
alf@mpce.mq.edu.au

Igor Shparlinski
Department of Computing
School of Mathematics, Physics, Computing and Electronics
Macquarie University NSW 2109 Australia
igor@mpce.mq.edu.au

Horst Gunter Zimmer
Fachbereich 9 Mathematik
Universität des Saarlandes
Postfach 15 11 50
D-66041 Saarbrücken, Germany
zimmer@arabella.math.uni-sb.de

* Particularly the Scots.

CONTENTS

IMPLEMENTING THE SELF INITIALIZING QUADRATIC SIEVE ON A DISTRIBUTED NETWORK

W. R. Alford and Carl Pomerance

Department of Mathematics, The University of Georgia, Athens, Georgia 30602, USA
E-mail: red@math.uga.edu and carl@math.uga.edu

1. Introduction

While the number field sieve now stands ready to give some competition (see [4]), the quadratic sieve factoring algorithm is still the method of choice for factoring large "hard" numbers. The record with the quadratic sieve is a 120 digit 'RSA modulus' (see [3]). It is expected that a similar attack on the famous RSA challenge number of 129 digits will be successful in 1994, though this time many people around the world are helping out.

This paper will discuss a variation of the quadratic sieve algorithm, which we call *self initialization*, that is well suited either for a single processor implementation or distributing the algorithm on a network of computers. We believe this variation can perhaps take as much as 1/2 off the running time, though actual speed-up factors will depend on the architecture employed. We describe an implementation of the self initializing quadratic sieve on an array of about 125 personal computers. With such an array we were able to factor a 95 digit number using about three weeks CPU time per unit and a 100 digit number using about six weeks CPU time per unit.

2. The basic quadratic sieve factoring algorithm

Building on previous work of Kraitchik, Lehmer and Powers, Brillhart and Morrison, and Schroeppel, the quadratic sieve algorithm was introduced by the second author in [6]. Let $f(x) = (x + [\sqrt{n}])^2 - n$ where n is the number to be factored. The basic idea is that as x runs over the integers in $(-n^\epsilon, n^\epsilon)$, $f(x)$ may occasionally have the property that it is composed of only the primes up to some relatively small point, say F. Further, these rare and special locations where $f(x)$ is so factorable may be found efficiently using a sieve procedure not unlike the sieve of Erastosthenes. If there are t primes up to F involved in $t+2$ special values $f(x_1), \dots, f(x_{t+2})$,

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

then by using a mod 2 matrix and Gaussian elimination, a subset $f(x_{k_1}), \ldots, f(x_{k_s})$ may be found with product a square:

$$f(x_{k_1}) \cdots f(x_{k_s}) = u^2. \tag{2.1}$$

If we let $v = (x_{k_1} + [\sqrt{n}]) \cdots (x_{k_s} + [\sqrt{n}])$, we evidently have

$$v^2 \equiv f(x_{k_1}) \cdots f(x_{k_s}) = u^2 \bmod n.$$

If n is composite, there should be at least a 50-50 chance that the greatest common divisor $(v - u, n)$, found by Euclid's algorithm, is a non-trivial factor of n. If it is trivial, a longer list of special values $f(x_i)$ may be found and another solution of (2.1) produced so as to get a new chance at factoring n.

The function $f(x)$ has several important properties for the above scheme to work. First, for each integer x we have some integer A (it is $x + [\sqrt{n}]$), such that

$$A^2 \equiv f(x) \bmod n, \quad A^2 \neq f(x).$$

Second, since $f(x)$ is a polynomial with integer coefficients, we have $f(x+k) \equiv f(x) \bmod k$ for any k, so that the multiples of any fixed number k appear in a regular pattern in the sequence of consecutive values of $f(x)$. This is the property that allows a sieve procedure to search for the special values of $f(x)$. Third, if $|x| < n^\epsilon$, then $|f(x)| = O(n^{1/2+\epsilon})$, for small ϵ. That is , the values of $f(x)$ are not much longer than half the length of n. Since they are relatively small, the property of being wholly factorable with the primes up to F should not be as rare as it would be for larger numbers.

3. The multiple polynomial version

Note that with $f(x)$ as above, $f(x) = (2x + O(1))\sqrt{n}$ for $|x| < n^\epsilon$ and ϵ small. Thus as x moves away from 0, the values $f(x)$ grow approximately linearly. In their implementation [2], Davis and Holdridge noticed that as $|x|$ grew, the success rate of finding completely factored values $f(x)$ declined significantly. They reasoned that this was due to the growing size of $f(x)$ and that if they could find some way to switch to a new polynomial and start over again, they would again be in a situation with relatively frequent "hits". Ultimately Davis did find such a means of switching polynomials, calling it the "special q's variation." They found about an order of magnitude improvement in speed with this idea. Around the same time, P. Montgomery independently suggested another method of changing polynomials that was somewhat better than the Davis scheme. Both of these methods are described in [7].

Suppose we are willing to sieve each polynomial over the interval $[-m, m)$ of arguments, where m is a parameter to be chosen. In P. Montgomery's method, we first find integers a, b, c which satisfy

$$b^2 - ac = n, \ |b| < a, \ a \approx \sqrt{2n}/m \tag{3.1}$$

and then let

$$f_{a,b}(x) = ax^2 - 2bx + c. \tag{3.2}$$

Then from (3.1)

$$af_{a,b}(x) = (ax - b)^2 - n. \tag{3.3}$$

The choice of $a \approx \sqrt{2n}/m$ in (3.1) is so that

$$f_{a,b}(m) \approx f_{a,b}(-m) \approx |f_{a,b}(0)| \approx \frac{m}{\sqrt{2}}\sqrt{n}.$$

How are the numbers a, b, c in (3.1) to be found? P. Montgomery suggested choosing a as a prime for which $(n/a) = 1$ and then solving $b^2 \equiv n \bmod a$ for b. In [7] it was suggested choosing a as the square of a prime q with $(u/q) = 1$. We can still solve $b^2 = n \bmod a$, but now fewer special values of the polynomials $f_{a,b}$ are needed to produce the congruent squares mod n as described in §2. In fact if we choose prime values of a, if t primes are involved in the factorizations of the special values $f_{a,b}(x)$ and if s values of a are involved, then from (3.3), we would need $s + t + 2$ special values to produce congruent squares. However, if a is always a square, we need only $t + 2$ special values. This is the multiple polynomial version of the quadratic sieve generally used today.

4. Initialization strategies

To sieve a polynomial $g(x)$ with the primes p up to F, one has to know which residues of $x \bmod p$ have $g(x)$ divisible by p. That is, we have to solve the polynomial congruence

$$g(x) \equiv 0 \bmod p.$$

This is called the initialization problem. If we are only sieving with one polynomial as in §2, then the initialization problem need only be solved once and so is not important to the performance of the algorithm. However in the multiple polynomial versions described in §3, the faster we can change polynomials, the better will be the performance of the algorithm. It is thus important to solve the initialization problem as efficiently as possible.

Suppose p is an odd prime with $p \nmid a$. Then from (3.3), $f_{a,b}(x) \equiv 0 \bmod p$ has solutions if and only if $(n/p) = 1$. Suppose this condition is satisfied and t_p is such that $t_p^2 \equiv n \bmod p$. Then to solve the initialization problem for $f_{a,b}$ we must compute the numbers

$$(b \pm t_p)a^{-1} \bmod p \tag{4.1}$$

for each such prime p. The set of primes $2 < p \le F$ for which $(n/p) = 1$ is called the *factor base*. If the factor base is large, the task of computing all of the numbers (4.1) can be non-negligible, especially computing the inverses $a^{-1} \bmod p$. Note however, that the numbers t_p stay fixed for one choice of $f_{a,b}$ to the next, so this need not be an important part of the computation.

In 1985, P. Montgomery suggested to the second author an idea that greatly mitigated the problem of computing the inverses $a^{-1} \bmod p$. Instead of choosing $a = q^2$ for some prime q, say one chooses $a = q_i^2 q_j^2$ where q_i, q_j are chosen from a pre-computed set $q_1, \ldots, q_r$ of primes with $(n/q_l) = 1$ and $q_l \approx (\sqrt{2n}/m)^{1/4}$ for each $l = 1, \ldots, r$. If in addition we pre-compute the files $q_l^{-2} \bmod p$ for each l and each factor base prime p then $a^{-1} \bmod p$ may be computed by a single multiplication mod p:

$$a^{-1} \equiv q_i^{-2} q_j^{-2} \bmod p.$$

Further, with $a = q_i^2 q_j^2$ there are two intrinsically different solutions b_1, b_2 to $b^2 \equiv n \bmod p$ (that is, $b_1 \not\equiv \pm\, b_2 \bmod p$), so there are two polynomials f_{a,b_1}, f_{a,b_2} with the same value of a. Thus with r inversions for each factor base prime p to compute the $q_l^{-2} \bmod p$ files, we are ready to quickly initialize $2\binom{r}{2} = r(r-1)$ polynomials. This idea was described in [9] with the further twist that the primes q_l be taken k at a time rather than two at a time. That is, if each $q_l \approx (\sqrt{2n}/m)^{1/(2k)}$, we can choose $a = q_{i_1}^2 \cdots q_{i_k}^2$. For each value of a there would be 2^{k-1} intrinsically different values of b and so with r primes $q_1, \ldots, q_r$, one could form $2^{k-1}\binom{r}{k}$ polynomials. It was suggested in [9] that $k = 3$ might be a good value. This is the version of the quadratic sieve recently implemented by te Riele on the NEC SX-2.

Others were aware of this idea but purposely shunned it in their implementations. The reason for this is that it does not easily lend itself to distribution over a network. Of course each node might have available a file server to store the large files $q_l^{-2} \bmod p$ (perhaps 50,000 or more 3-byte integers for each l), but in practice most computers in the network were not so endowed. Thus for the scheme to work, a central computer would have to be continually sending these files out to various nodes in the network. This daunting problem completely sank the idea.

5. The self initializing quadratic sieve

We saw in the previous section that if a has k distinct prime factors, then there are 2^{k-1} intrinsically different choices for b associated to this value of a. Say we choose $k = 10$. There are then 512 different polynomials $f_{a,b}$ with the same value for a. Thus the computation of $a^{-1} \bmod p$ for each factor base prime p need be done only once for each of the 512 polynomials. However, if we choose a as the square of the product of 10 roughly equal primes q, then each q is about $(\sqrt{2n}/m)^{1/20}$. If $n \approx 10^{100}$, $m \approx 10^7$, then each q is about 400 or 500. There may be only about 10 primes q in this range with $(n/q) = 1$ and so this procedure can be done only once.

This problem can be solved by taking a as the product of 10 distinct primes, rather than the square of such a product. Then with the same n, m as above, the primes q can be chosen near 200 000, so there are more to choose from. For example, between 206 000 and 206 750 there should be about 30 primes q with $(n/q) = 1$. The largest product of 10 of these 30 primes would only be about 3% larger than the smallest product. We thus could generate $2^9\binom{30}{10} > 1.5 \times 10^{10}$ acceptable polynomials.

Since the primes in a are in the factor base range, there is no additional work to eliminate them in the matrix stage of the algorithm. However, care must be taken to avoid duplications from one polynomial to another. An easy way to do this is to discard any report $f_{a,b}(x)$ divisible by a prime among the 30 used to construct leading coefficients a. In the above example, fewer than 1% of the reports would need to be discarded. With a little more effort, this discard pile can be culled for repeats, and then used.

How are the 512 values of b for each choice of a related to each other? Suppose $a = q_1 q_2 \cdots q_{10}$ where $q_1, q_2 \ldots, q_{10}$ are distinct primes with $(n/q_l) = 1$. Then if $b^2 \equiv n \bmod a$, we may solve for b via the Chinese Remainder Theorem. If q_l is one of the prime factors of a, let $t(a, q_l)$ denote the least non-negative integer in the residue class $t_{q_l}(a/q_l)^{-1} \bmod q_l$, where $t_{q_l}^2 \equiv n \bmod q_l$. Then the various solutions of $b^2 \equiv n \bmod a$ are given by the formula

$$b \equiv \sum_{l=1}^{10} \pm(a/q_l)t(a, q_l) \bmod a.$$

There are 2^{10} choices of signs in this expression. However we do not wish to use both b and $-b$, so we fix one of these signs, leaving 2^9 choices. Let

$$B_l = (a/q_l)t(a, q_l) \quad \text{for } l = 1, \ldots, 10.$$

Thus the 512 choices of b are

(5.1) $$B_{10} \pm B_9 \pm \cdots \pm B_1.$$

The numbers given by (5.1) may be traversed with a Gray code. Thus if we let $b_1 = B_{10} + B_9 + \cdots + B_1$ we have

(5.2) $$b_{i+1} = b_i + 2(-1)^{\lceil i/2^\nu \rceil} B_\nu$$

where $2^\nu \| 2i$ for $i = 1, 2, \ldots, 511$. Note that we skip the reduction mod a in (5.1) and (5.2), so we may no longer have $|b_i| < a$ as required by (3.1). However, each B_l satisfies $0 \le B_l < a$, so we have instead $|b_i| < 10a$. This does not pose a problem.

This choice of the sequence of b's has the following pleasant consequence for initialization. Suppose z_1, z_2 are the two solutions of $f_{a,b_i}(z) \equiv 0 \bmod p$ where p is a factor base prime not in the set of q's used to form the initial coefficients a. Then if z_1', z_2' are the corresponding solutions to $f_{a,b_{i+1}}(z) \equiv 0 \bmod p$, we have from (4.1) and (5.2) that

$$z_j' \equiv z_j + 2(-1)^{\lceil i/2^\nu \rceil} B_\nu a^{-1} \bmod p \text{ for } j = 1, 2. \tag{5.3}$$

Thus the initial values z_1', z_2' for polynomial $f_{a,b_{i+1}}$ may be computed from the initial values z_1, z_2 for polynomial f_{a,b_i} via (5.3). In addition, the numbers z_1, z_2 need not be saved for this to work, since when we are finished sieving f_{a,b_i} we still have numbers $\hat{z}_1, \hat{z}_2$ respectively congruent mod p to z_1, z_2. Thus we may use $\hat{z}_1, \hat{z}_2$ in (5.3) in place of z_1, z_2. Thus the *terminal* numbers for f_{a,b_i} can be used to compute the *initial* numbers for $f_{a,b_{i+1}}$.

To summarize, the work needed to initialize $f_{a,b_{i+1}}$ is to compute z_1', z_2' for each factor base prime p using (5.3) with z_1, z_2 the terminal numbers mod p of f_{a,b_i}. It is advantageous to save the 9 multiprecision integers $B_1, \ldots, B_9$ (they are less than $\sqrt{n}$) and the numbers $2a^{-1} \bmod p$ for each factor base prime p. Thus to compute z_1', z_2' we must

(i) reduce $B_\nu \bmod p$
(ii) multiply $2a^{-1}B_\nu \bmod p$
(iii) for $j = 1, 2$, do the modular additions $z_j \pm 2a^{-1}B_\nu \bmod p$, for the appropriate choice of sign.

In comparison, if z_1', z_2' were found via (4.1), essentially the same tasks would be performed, but in addition we would have to compute $a^{-1} \bmod p$.

If there is more memory available, the above scheme for self initialization may be significantly improved. For example, suppose we can save the numbers $2a^{-1}B_1 \bmod p$ for each factor base prime p. Since $\nu = 1$ whenever i is odd, we may use this pre-computed value for initialization in half of the 512 polynomials. That is, half of the time, we do not have to do (i) and (ii) above, but can go directly to (iii). If there is also room for $2a^{-1}B_2 \bmod p$ for each factor base prime p, we can skip (i) and (ii) for half of the remaining cases, and so on. In fact, if there is enough memory for $2a^{-1}B_l \bmod p$ for each p and each $l = 1, \ldots, 9$, we need never do (i) and (ii) after an initial computation. (In this case we would not need to store the numbers $2a^{-1} \bmod p$.) Moreover, if we also remember the <u>initial</u> values for z_1, z_2 rather than use the terminal values, step (iii) becomes the totally trivial problem of two addition or subtraction problems mod p with inputs already reduced mod p.

We thus see there is a classic time-space trade-off on the initialization problem, with the more space available to remember previous calculations, the less time is necessary for initialization. However, even with the minimal space required for just storing $2a^{-1} \bmod p$ for each factor base prime p, we still gain significantly using self-initialization.

Finally we remark how self initialization may be implemented on a distributed network. Suppose, for example, there are about 450 (or fewer) nodes. We have the problem of each node choosing various subsets of size 10 from the 30 primes used to form initial coefficients, but without duplication between nodes. To do this, we fix some 11 of these primes as "markers" and to each node we assign a different 5-tuple of these markers. This allows for $\binom{11}{5} = 462$ nodes. Then each node chooses 5 primes from the remaining 19 in every possible way — with each such choice a 10-tuple is formed by using the 5 fixed marker primes for that node. Thus each node can chose $\binom{19}{5} = 11\,628$ 10-tuples and so can form nearly 6 million polynomials. Even if polynomials are changed as frequently as once per minute, each node still has more than an 11 year supply, and no node will duplicate the work of another.

6. The ratio of sieving time to initialization time

If each node in a network is running the quadratic sieve as a sequential processor, it makes sense to ask what fraction of the time is spent initializing and what fraction of the time is spent sieving. Neglecting pre-processing and the final steps of the algorithm, these two fractions add to 1. If we spend a long time sieving each polynomial, then we will spend only a small fraction of the time initializing. But it is good to change polynomials frequently, since the more time that is spent on a polynomial, the lower the yield per unit time in factored values. On the other hand, the yield per unit time is zero while we are initializing, so this should not comprise too great a fraction of the running time.

There is thus a question of finding the optimal ratio r of sieving time to initialization time for a given number n to be factored and a given size factor base. Of course, r may be approximated empirically by trying various choices, but it may also be of interest to attempt a theoretical analysis.

Suppose we are trying to factor n using a factor base consisting of the primes up to F. The length, $2m$, of the interval we sieve for each polynomial, is directly proportional to our choice of r, say $m = c_1 r$ (where c_1 depends on the implementation). The largest polynomial values on this interval are about $m\sqrt{n/2}$, so say the typical values are about $\frac{1}{2}m\sqrt{n} = \frac{1}{2}c_1 r\sqrt{n}$. Let

$$u = u(r) = \frac{\log(\frac{1}{2}c_1 r\sqrt{n})}{\log F}.$$

Then the expected number N of values of one of our polynomials on $[-m, m]$ which factor completely over the factor base satisfies

$$N \approx 2m\rho(u) = 2c_1 r\rho(u(r)) \tag{6.1}$$

where ρ is the Dickman–de Bruijn function. This function is continuous, 1 on $[0, 1]$, and satisfies $u\rho'(u) = -\rho(u-1)$ for $u > 1$. The connection between this function and the distribution of integers which factor completely over a set of small primes is well known though it is still not proved for polynomial values. Here, the estimate (6.1) is only conjectured.

Assuming (6.1), the yield rate of completely factored polynomial values per unit time is given by

$$Y(r) = \frac{2c_1 r\rho(u(r))}{1+r}.$$

Thus to find the optimal choice for r we equate $Y'(r)$ to 0, getting

$$(\rho(u) + r\rho'(u)u'(r))(1+r) = r\rho(u),$$

so that using $\rho'(r) = 1/(r\log F)$,

$$1 + \frac{\rho'(u)}{\rho(u)\log F} = \frac{r}{1+r} = 1 - \frac{1}{1+r}.$$

Thus, using the differential equation for ρ,

$$r = \frac{u\rho(u)\log F}{\rho(u-1)} - 1. \tag{6.2}$$

Strictly speaking, (6.2) is not the solution for r since u depends on r. But note that $u\rho(u)/\rho(u-1) \approx 1/\log u$, so the dependence of the right side on r is slight. To see how one might use (6.2) to compute r, suppose $n \approx 10^{100}$, $F \approx 1,300,000$ (corresponding to a factor base of about 50,000 primes). We assume $\frac{1}{2}m\sqrt{n} \approx 10^{57}$. (If we later repeat the computation with 10^{57} replaced with 10^{60} or 10^{54}, we get about the same answer.) Then

$$u = \frac{\log(10^{57})}{\log(1.3 \times 10^6)} \approx 9.323,$$

which when put in (6.2) gives $r \approx 2.95$.

This argument neglects the effect of the large prime variation (see [6]). Since this variation becomes more effective the larger the value of u, (6.2) might then be slightly understating the optimal value of r. Again, this should not produce a big difference in the answer. Further, a smooth function is flat near an extreme point. Thus in the above example, choosing r anywhere in the interval [2.5, 3.5] should give about the same performance.

7. Experience with the method

In 1989 we factored four numbers from the Cunningham project [1]: a 92 digit composite factor of $2^{128}+1$, composite factors of $2^{332}+1$ and $7^{128}+1$ each of 95 digits, and a 100 digit composite factor of $12^{119}+1$.

The computations were done primarily on 125 PC's found in public laboratories of the Departments of Mathematics and of Computer Science at the University of Georgia. These computers were used when not occupied by students, namely nights, holidays and weekends. (We wish to thank these departments for the use of their computers on this project.)

These 125 PC's are not advanced computers. The CPU of each is an Intel 8088 operating at 8 Mhz, and with RAM of either 256k, 512k or 640k. The computers each have a 360k floppy drive and are not networked. That is, the only communication these computers have is via floppy disks.

The PC's were divided into two groups: the sievers and the factorers. The sievers initialize the sieve from data supplied by floppy disk, do the sieving, and report sieve locations (to the floppy disk) that correspond to numbers that potentially factor over the factor base. Due to using both the large prime and small moduli variations (see [7]), the majority of these reports are false. The factorers later factor the reports (via trial division), keeping only the true reports and reserving these for the final matrix stage of the algorithm.

The PC's with at least 512k RAM were used as the sievers, while the PC's with less RAM were used as factorers. The larger a number is that one tries to factor, the less frequent in general are the reports from sieving. Thus with the 100 digit number we factored, the ten PC's with 256k RAM reserved for factoring were more than enough to keep up with the sievers; in fact, the reports were so infrequent that we took the opportunity to push the small moduli variation to 100 without overloading the factorers with too many false reports. With the 92 digit number that we factored, we had to take some sievers and convert them to factorers despite the fact that we had the small moduli cutoff set at 32.

With less than one Meg of total memory (640k or 512k of RAM and 360k floppy) for the sievers, a description of some aspects of the program may be in order. The floppy disks had to be bootable; however, after the boot sequence, the three files (BIOS, DOS and COMMAMD.COM) that make a disk bootable were erased from the floppy. After the sieving program was initiated, all of the programs including the sieving program were erased leaving the full 360k on the disk for data. The sieving program treated the memory space on the disk as a virtual extension of RAM with the least used data residing on the floppy. The factor base was "dynamic" and was, of course, smaller for the 512k machines. As reports to the floppy accumulated, the factor base was truncated on the large prime end to accommodate the increased space needed for reports. Once sieving commenced, each and every byte in RAM and on the disk was used for the entire run. As always in combination of congruences factoring algorithms space is at a premium and these remarks are given to aid others and to explain how we were able to factor such large numbers with such modest machines.

For our 100 digit factorization, our initial factor base consisted of the primes to 1170599, or about 45300 primes. As mentioned above, this was truncated somewhat for the smaller sievers and was also truncated during runs as the disk filled with data. We used 20 special primes around 18300, taking 10 at a time to form the polynomials (512 polynomials for each choice of 10 primes). Our sieve interval length (namely, the number of values sieved for each polynomial) was $272 \cdot 2^{16}$, which is close to 18 million. Each siever spent about 70% of the time sieving and about 30% of the time initializing. Our large prime cut-off was 2^{28} (that is, we only kept those reports that factored completely over the factor base, except possibly for one larger prime that was below the cut-off). For some of our earlier factorizations we had used a cut-off of 2^{32}. Diminishing the cut-off cost us a little in the running time, but saved a large amount of storage space.

Coordinating all of these PC's was one 80286 AT with 640k of RAM, a 40 Mb hard disk drive and a $5\frac{1}{4}''$ high density floppy disk drive. This machine was used to do all of the precalculations needed such as calculating a multiplier and finding the factor base. It also did the final factorization, passed out polynomials to the sievers, collected reports from the sievers, made the sieving disks bootable again, passed out reports to the factorers, received true reports from the factorers and save these true reports to floppies. Since much more data than 40M was collected (many of the large primes in the large prime variation did not have a match so they were not used in the final factoring), it had to be stored on removable media. The data needed for the final factoring amounted to about 17M so the hard disk drive was sufficiently large to accommodate it.

There was, in addition, one Sun 3/60 used for the matrix step of the algorithm. Communication between the AT and the Sun was early on via $\frac{1}{4}''$ tape and later via a high speed network. The primary reason for the use of the Sun was that the linear algebra step for matrix reduction was already written for the Sun in connection with the project described in [8]. (We would like to thank J. W. Smith for the use of that program.)

8. Conclusion

In the multiple polynomial version of the quadratic sieve, the less time spent sieving a particular polynomial, the higher the yield of factored residues reported. The principal overhead involved with switching polynomials is the sieve initialization computation. In a sequential implementation, while the computer initializes, the yield of factored residues is of course zero. Thus one wants to have initialization time be at most a small fraction of the total time spent factoring. Any strategy that can speed initialization is best used by not altering the fraction of time spent initializing, but rather used to shorten the time spent sieving per polynomial, and thus to increase the yield.

In this paper we have described a method, 'self initialization', that essentially performs the initialization computation for many polynomials at once. Thus the amortized time per polynomial can be greatly reduced. This idea can be equally well used for a single processor implementation or for a distributed implementation. We have described our experience with a 'network' of about 125 PC's.

The principal problem with self initialization is an increased demand on memory. However, it is not an 'all or nothing' strategy, and with limited memory resources, one can use some of the ideas presented and still make a gain over traditional initialization strategies.

Since our experience is with PC's, it might be expected that other issues enter when one uses a workstation, a mainframe, or a supercomputer. We have heard from Herman te Riele that he and a student are trying out some of these ideas on such computers; we await the report of their experience. The self initialization idea seems ideally suited for a hypercube architecture, since the array of polynomials used in the process naturally arranges itself in a high dimensional hypercube. It would be interesting to see if this is in fact a profitable idea. In this regard, see [5].

Acknowledgements

The second author is partially supported by an NSF grant.

References

[1] 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*, second edition, Contemp. Math. **22** Amer. Math. Soc. Providence 1988.

[2] J. A. Davis and D. B. Holdridge, 'Factorization using the quadratic sieve algorithm', *Sandia Report* Sand 83-1346, Sandia National Laboratories, Albuquerque, NM 1983.

[3] T. Denny, B. Dodson, A. K. Lenstra and M. S. Manasse, 'On the factorization of RSA-120', Extended Abstract, Crypto 93.

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

[5] R. Peralta, 'A quadratic sieve on the n-dimensional hypercube', in *Advances in Cryptology* – Crypto '92, E. Brickell, ed., Lecture Notes in Computer Science 740, Springer Verlag, Berlin, 1993, 324–332.

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

[7] C. Pomerance, 'The quadratic sieve factoring algorithm' in *Advances in Cryptology* – Proceedings of EUROCRYPT 84, T. Beth, et al., eds., Lecture Notes in Computer Science 209 Springer-Verlag, Berlin 1985, 169–182.

[8] C. Pomerance and J. W. Smith, ' Reduction of huge, sparse matrices over finite fields via created catastrophes', *Experimental Math.*, **1** (1992), 89–94.

[9] C. Pomerance, J. W. Smith and R. Tuler, 'A pipeline architecture for factoring large integers with the quadratic sieve algorithm', *SIAM J. Comput.*, **17** (1988), 387–403.