Optimization of the MPQS-factoring algorithm on the Cyber 205 and the NEC SX-2

Walter Lioen, Herman te Riele, Dik Winter
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT
This paper describes the optimization of a program for the factorization of large integers on two large vector processors: a CDC Cyber 205 and a NEC SX-2. The factoring method used is the so-called multiple polynomial version of the quadratic sieve algorithm.
Several large integers in the 48–92 decimal digits range have actually been factorized with these two programs. The largest number, the 92-digit composite $(6^{131}-1)/(5\cdot 263\cdot 3931\cdot 6551)$, was factorized in about 95 CPU-hours on the NEC SX-2. This result means a new absolute record for general purpose factoring methods.

1980 Mathematics Subject Classification (1985) revision: 11Y05, 11A51, 65F05, 65V05, 65W05.
Keywords and Phrases: quadratic sieve factorization, vector computer, vectorization.

1. Introduction

Factoring large numbers has long been considered a nice, but useless, activity in number theory. However, the discovery, about ten years ago, by Rivest et al [7], that the difficulty of breaking certain cryptographic codes depends on the difficulty of factoring large integers, has considerably stimulated the interest in this problem. In particular, in order to be able to use safe cryptosystems based on factoring, it is of continuous interest to know what can be achieved in factoring with the best method and with the fastest available (super)computer. At present, at least six groups of researchers in the USA, Australia, Japan and The Netherlands are heavily involved in factoring large numbers, some of them aided by very powerful vector an parallel computers. Although many factoring methods are known [6], only two of them are still feasible for numbers of, say, more than 65 decimal digits, viz., the Elliptic Curve Method (ECM, [2]) and the Multiple Polynomial Quadratic Sieve (MPQS, [4]). ECM and MPQS are complementary in the sense that if N is the number to be factorized, then the computing time of ECM depends on the size of the second largest prime divisor of N, whereas the computing time of MPQS depends on the size of N itself. Usually, ECM is tried first and when it has failed after some time, then MPQS is invoked.

In this paper we describe how we have optimized the MPQS-algorithm for the Cyber 205 and the NEC SX-2 vector computers. We assume the reader to be familiar with the basic characteristics of the two machines. We shall restrict our attention to the two most time-critical loops: the so-called sieve loop and the selection loop. These loops consume more than 75% of the total CPU-time. Moreover, we devote a few words to a memory-critical part of the algorithm: the Gaussian elimination step.

We have applied our programs to various large composite numbers in the 48–92 decimal digits range. The results obtained are described in [5]. The largest number we have split into prime factors was a 92-digit composite from the so-called Cunningham project [1]. It took the NEC SX-2 about 95 AP-hours to factorize this number, and at the moment of writing this is the largest difficult composite number ever factorized by means of a general purpose factoring method.

2. The MPQS-factorization method

It is beyond the scope of this paper to fully describe the Multiple Polynomial Quadratic Sieve algorithm. We describe it here in a simplified form in order to be able to explain the place of the two time-critical loops and the Gaussian elimination step. A complete description of the algorithm may be found in [4] and practical experiences with MPQS are described in [8] and [5].

Suppose that we want to factorize the (large) integer N, which is known to be composite and whose smallest prime divisor is known to be reasonably large (e.g. by Pollard's $p-1$ method [6], or by ECM). The idea of the Quadratic Sieve Factoring algorithm is to find two integers X and Y satisfying the congruence $X^2 \equiv Y^2 \pmod{N}$, from congruences of the form $U_i^2 \equiv V_i^2 W_i \pmod{N}$, where the latter congruences are generated by means of a quadratic polynomial $W(x)$, and where the numbers W_i are easy to factor, or at least much easier than N. If sufficiently many such triples (U_i, V_i, W_i) are known, where the W_i's are completely factorized, then indeed such an (X, Y)-congruence can be found. Next we compute $d := \gcd(X - Y, N)$ by Euclid's algorithm and if $1 < d < N$ then d is a proper divisor of N. If not sufficiently many congruences could be generated with one quadratic polynomial, then another is constructed, and so on.

The simplified version of the Quadratic Sieve (with one polynomial) now looks as follows. Let $U(x) := x + m$, where $m = \lfloor N^{1/2} \rfloor$, $V := 1$ and $W(x) := (U(x))^2 - N$, for $x = 0, \pm 1, \pm 2, \ldots$. Then we have

$$(U(x))^2 \equiv W(x) \pmod{N}$$

and

$$W(x) \approx 2xN^{1/2} \ll N \quad \text{if} \quad x \ll N^{1/2}.$$

Hence $W(x)$ is easier to factorize than N. Moreover, $W(x)$ has the nice property that if $p \mid W(x_0)$ for some x_0 then also $p \mid W(x_0 + kp)$, for all $k \in \mathbf{Z}$. Such an x_0 may be found for given p as follows:

$$W(x) \equiv 0 \pmod{p} \quad \text{implies that} \quad (x + m)^2 \equiv N \pmod{p};$$

this equation has two solutions if N is a so-called quadratic residue of p (shortly denoted by the Legendre symbol notation $(\frac{N}{p}) = 1$). These solutions can be computed quite easily (cf. [6, pp. 287–288]).

We now describe the *Quadratic Sieve Algorithm*:

1. Choose a factor base $FB := \{p \le B \mid p \text{ prime and } (\frac{N}{p}) = 1\}$ for some suitable B (these are the primes allowed in the W's that we want to factorize completely); let $F := |FB|$.
2. $\forall p \in FB$ solve $W(x) \equiv 0 \pmod{p} \longrightarrow$ two solutions r_1^p and r_2^p.
3. Initialize a sieving array $SI(-M : M - 1)$ to 0, where M is suitably chosen.
4. (Sieving) $\forall p \in FB, \forall j \in [-M, M - 1]$ such that $j \equiv r_1^p \pmod{p}$ or $j \equiv r_2^p \pmod{p}$: $SI(j) := SI(j) + \log p$.
5. (Selection) Select those $x \in [-M, M-1]$ for which $|SI(x)| \approx \log |W(x)|$ and store these numbers into $x_1, x_2, \ldots$. Note that $\log |W(x)|$ is very slowly varying and for those x which are selected in this step, there is a very good chance that $W(x)$ is only composed of primes belonging to the factor base! Associate with x_i and $W(x_i)$ the vector of exponents of $W(x) \pmod{2} : \vec{\alpha}^T = (\alpha_0, \alpha_1, \ldots, \alpha_F)$ where

   $$W(x_i) = (-1)^{\alpha_0} \prod_{j=1}^{F} p_j^{\alpha_j},$$

   and $p_1, \ldots, p_F$ are the primes in FB.

6. (Gaussian elimination) Collect at least $F+2$ completely factorized W's (assume this is possible for the choice of B and M made) and find linear combinations of vectors $\vec{\alpha}$ which, added (mod 2), yield $\vec{0}$. This is carried out by Gaussian elimination (mod 2).
7. Multiply those $W(x)$-values whose linear combination of exponent-vectors yield the $\vec{0}$-vector. This implies that we have found a congruence of the form $X^2 \equiv Y^2 \pmod{N}$; compute these X and Y, and $\gcd(X-Y, N)$ which should be a factor of N. (Usually, in step 6 more than one suitable linear combination of exponent-vectors are found and this may help if the first gcd found is 1 or N.)

The most time-consuming part in this algorithm is step 4 because in order to factor a very large number N, the parameters B and M have to be chosen very large (and a large M implies a long sieving array and a large B implies many primes in the sieving step 4). Step 5 may also consume a non-trivial portion of the total CPU-time. Finally, the Gaussian elimination step 6 deserves attention, not because of the time, but because of the memory it needs. In the next Section we describe how we have optimized steps 4, 5 and 6 on the Cyber 205 and on the NEC SX-2.

3. Optimization

3.1 The sieve loop

The loop in the sieving step 4 may be given in Fortran as

```
      DO 30 I = 1, F
         P = FB(I)
         LP = ALOG(P)
         DO 20 J = 1, 2
            RJ = SOL(I, J)
            KF = FIRST(M, P, RJ)
            KL = LAST (M, P, RJ)
            DO 10 K = KF, KL, P
               SI(K) = SI(K) + LP
   10       CONTINUE
   20    CONTINUE
   30 CONTINUE
```

Here, `FB(I)` is the `I`-th prime in the factor base, `SOL(I,1)` and `SOL(I,2)` are the two precomputed solutions of the equation $W(x) \equiv 0 \pmod{\texttt{P}}$, and `FIRST` and `LAST` are functions which determine the first and the last places in array `SI(-M:M-1)` to which $\log \texttt{P}$ has to be added.

Of course, only the `10`-loop is vectorizable. On the NEC SX-2 this was done automatically by the compiler, but on the Cyber 205 we had to invoke periodic gather and scatter calls to obtain vector speed. This piece of code looks as follows (`LSI` is the length of the sieving array `SI`, i.e., `2*M`):

```
      LEN = (KL - KF)/P + 1
      HELP(1;LEN) = Q8VGATHP(SI(KF;LSI),P,LEN;HELP(1;LEN))
      HELP(1;LEN) = HELP(1;LEN)+LP
      SI(KF;LSI)  = Q8VSCATP(HELP(1;LEN),P,LEN;SI(KF;LSI))
```

The length (`LEN`) of the `10`-loop is about `2*M/P` and this may vary, for the values of B and M involved, roughly between 5 and 170,000 as `P` runs through the primes in the factor base. Therefore, on the NEC SX-2, this loop is processed much more efficiently than on the Cyber 205, since the NEC SX-2 attains vector speed for much smaller vectors than the Cyber 205. For the complete `30`-loop the NEC SX-2 reached an average speed of about 90 million `LP`-additions per second, against 13 million reached by the Cyber 205.

3.2 The selection loop

The loop in the selection step 5 may be described in Fortran as follows (THRES represents the value of $\log |W(x)|$, approximately constant for $x \in [-M, M-1]$):

```
      COUNT = 0
      DO 10 I = -M, M-1
         IF (SI(I) .LT. THRES) GOTO 10
         COUNT = COUNT + 1
         X(COUNT) = I
   10 CONTINUE
```

Neither the Cyber 205 nor the NEC SX-2 compiler are able to vectorize this loop automatically. On the Cyber 205 we could use (non standard Fortran) bit vectors and obtain reasonable vector speed with the following code:

```
      BITV(-M; LSI) = SI(-M; LSI) .GE. THRES
      COUNT = Q8SCNT(BITV(-M; LSI))
      X(1;COUNT) = Q8VCMPRS(SI(-M;LSI),BITV(-M;LSI);X(1;COUNT))
```

The function Q8SCNT counts the number of 1's in a bit vector. We have not measured precisely the performance of this piece of code (on the Cyber 205) because its CPU-time was dominated by the time needed to perform the sieve step.

On the NEC SX-2 we could obtain vector speed (because the number of times that SI(I) $\geq$ THRES was always extremely small compared with the length of the sieving array SI) with the following piece of code:

```
      COUNT = 0
      IPOINT = -M
*VDIR LOOPCNT=256
   10 DO 20 I = IPOINT, IPOINT+255
         IF (SI(I) .GE. THRES) GOTO 30
   20 CONTINUE
      IPOINT = IPOINT + 256
      IF (IPOINT+255 .LE. M-1) THEN
         GOTO 10
      ELSE
         GOTO 40
      END IF
   30 COUNT = COUNT + 1
      X(COUNT) = I
      IPOINT = I + 1
      IF (IPOINT+255 .LE. M-1) GOTO 10
   40 CONTINUE
*VDIR LOOPCNT=256
      DO 50 I = IPOINT, M-1
         IF (SI(I) .GE. THRES) THEN
            COUNT = COUNT + 1
            X(COUNT) = I
         END IF
   50 CONTINUE
```

The time to run this piece of code on the SX-2 was about equal to the total sieving time in step 4. The speed was about 90 million comparison per second.

3.3 The Gaussian elimination

The algorithm we used to perform Gaussian elimination (mod 2) on the matrix of exponent vectors of $W(x_i)$ is described in [3]. The binary elements of this matrix could be packed in words of 64 bits on the Cyber 205, and in words of 32 bits on the NEC SX-2; the elimination process could be carried out very efficiently by means of the `XOR`-operation. On both machines, the time to carry out the Gaussian elimination was completely negligible compared to the time needed for the sieving and the selection steps. However, for our program, the available central memory was crucial for the maximum size of the Gaussian elimination matrix and this, in turn, dictated the maximal size of the number we could factorize.

For example, the largest number we have factorized on the Cyber 205 has 82 decimal digits, and the size of the factor base was about 7,400. The Gaussian elimination on the corresponding $7{,}400 \times 7{,}400$ matrix consumed about 4 minutes CPU-time (the total CPU-time needed was about 70 hours!) and this matrix almost completely occupied the available (at that time) central memory of about 1 Mwords of 64 bits. However, on the NEC SX-2 which has a central memory of 32 Mwords of 32 bits, we could process a much larger matrix, and, therefore, also factorize larger numbers. This larger memory also allowed us to work with a larger sieving array and this reduced the factoring time to some extent, compared to that for the same number on the Cyber 205. Our NEC SX-2 champion has 92 decimal digits, the size of the factor base was about 24,300 and the size of the Gaussian elimination matrix $24{,}300\times24{,}300$. The Gaussian elimination time was about 4 minutes (and the total amount of factoring time about 95 AP-hours!).

4. Conclusions

We have described how we optimized the time-critical loops in our (multiple polynomial) quadratic sieve factoring program. On the NEC SX-2 these loops could be vectorized using standard Fortran 77, combined with vector directives. On the Cyber 205 we had to invoke FORTRAN 200 vector syntax in order to vectorize these loops. Our NEC SX-2 program runs about 5–10 times as fast as our Cyber 205 program. Moreover, due to the much larger central memory on the NEC SX-2 we could also factorize larger numbers with our NEC SX-2 program than with our Cyber 205 program.

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*, volume 22 of *Contemporary Mathematics Series*. American Mathematical Society, Providence, second edition, 1988.
2. H.W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics (Second Series)*, 126(3):649–673, November 1987.
3. D. Parkinson and M. Wunderlich. A compact algorithm for Gaussian elimination over GF(2) implemented on highly parallel computers. *Parallel Computing*, 1(1):65–73, August 1984.
4. C. Pomerance, J.W. Smith, and R. Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM Journal on Computing*, 17(2):387–403, April 1988.
5. H.J.J. te Riele, W.M. Lioen, and D.T. Winter. Factoring with the quadratic sieve on large vector computers. *Journal of Computational and Applied Mathematics*, 27(1&2):267–278, September 1989.
6. H. Riesel. *Prime Numbers and Computer Methods for Factorization.* Birkhäuser, Boston, 1985.
7. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.
8. R.D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48(177):329–339, January 1987.