INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Karatsuba Square Root

Paul Zimmermann

N°3805

THÈME 2

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--3805--FR+ENG

Karatsuba Square Root

Paul Zimmermann

Thème 2 — Génie logiciel
et calcul symbolique
Projet PolKA

Rapport de recherche n° 3805 — — 8 pages

Abstract: We exhibit an algorithm to compute the square-root with remainder of a n-word number in $\frac{3}{2}K(n)$ word operations, where $K(n)$ is the number of words operations to multiply two n-word numbers using Karatsuba's algorithm. If the remainder is not needed, the cost can be reduced to $K(n)$ on average. This algorithm can be used for floating-point or polynomial computations too; although not optimal asymptotically, its simplicity gives a wide range of use, from about 50 to $1,000,000$ digits, as shown by computer experiments.

Key-words: square root, Karatsuba division, GNU MP, MPFR library

(Résumé : tsvp)

This work has been partly supported by the "Action de recherche coopérative" entitled "Outils pour un calcul numérique fiable" (`http://www-sop.inria.fr/prisme/fiable`). A previous version of this paper has been submitted to IEEE Transactions on Computers.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : 03 83 59 30 30 - International : +33 3 3 83 59 30 30
Télécopie : 03 83 27 83 19 - International : +33 3 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : 03 87 20 35 00 - Internatională: +33 3 87 20 35 00
Télécopie : 03 87 76 39 77 - International : +33 3 87 76 39 77

Racine carrée à la Karatsuba

Résumé : Nous présentons un algorithme calculant la racine carrée avec reste d'un entier de n mots machine en $\frac{3}{2}K(n)$ opérations sur des mots, où $K(n)$ désigne le nombre de telles opérations utilisées par l'algorithme de Karatsuba pour multiplier deux entiers de n mots. Si seul le quotient est recherché, le coût peut être abaissé à $K(n)$ opérations en moyenne. Cet algorithme s'applique également aux nombres flottants ou aux polynômes ; quoique non optimal asymptotiquement, l'expérience montre qu'il est efficace en pratique sur une large plage, allant environ de 50 à un million de chiffres.

Mots-clé : racine carrée, division à la Karatsuba, GNU MP, bibliothèque MPFR

Introduction. The current asymptotically fastest known method to compute the square-root of a n-word number is using Fast Fourier Transform (FFT) multiplication and Newton's method, with a complexity of $5M(n)$[1] [1, p. 155].[2] The algorithm presented here is based on Burnikel-Ziegler Karatsuba division [2]. Given an integer n, our algorithm computes simultaneously its integer square root $s = \lfloor\sqrt{n}\rfloor$ and the corresponding remainder $r = n - s^2$. It is not asymptotically optimal but very efficient in practice, with a complexity of about $\frac{3}{2}K(n)$ word operations, where $K(n)$ is the number of words operations to multiply two n-word numbers using Karatsuba's algorithm. This low complexity comes both from the beautiful Karatsuba division recently found by Burnikel and Ziegler, and from the careful use of remainders, which avoids unnecessary computations.

The paper is organized as follows: we first present the algorithm, then prove its correctness, analyze its complexity, discuss some variants, and give timings of an implementation in the GNU MP multiprecision library.

Algorithm $\texttt{SqrtRem}(n = a_3b^3 + a_2b^2 + a_1b + a_0)$
Input: $0 \le a_i < b$ with $a_3 \ge b/4$
Output: (s, r) such that $s^2 \le n = s^2 + r < (s+1)^2$
  $(s', r') \leftarrow \texttt{SqrtRem}(a_3b + a_2)$
  $(q, u) \leftarrow \texttt{DivRem}(r'b + a_1, 2s')$
  $s \leftarrow s'b + q$
  $r \leftarrow ub + a_0 - q^2$
  if $r < 0$ **then**
    $r \leftarrow r + 2s - 1$
    $s \leftarrow s - 1$
  return (s, r)

Proof of the algorithm. We have to prove three facts: firstly that $n = s^2 + r$ at the end of the algorithm, then that $n \le s^2$ and finally that $n < (s+1)^2$. After the call $\texttt{SqrtRem}(a_3b + a_2)$, we have $n = s'^2b^2 + r'b^2 + a_1b + a_0$. After $\texttt{DivRem}(r'b + a_1, 2s')$, we have $n = s'^2b^2 + 2s'qb + ub + a_0 = (s'b + q)^2 + ub + a_0 - q^2$. This proves that $n = s^2 + r$ at the end of the algorithm, since $s^2 + r$ is an invariant from the if-statement.

Let us prove now $n < (s+1)^2$. If $n \ge (s'b + t)^2$, then we have $2s'bt \le 2s'bt + t^2 \le r'b^2 + a_1b + a_0$. Since $a_0 < b$, this also implies $2s't \le r'b + a_1$ and thus $t \le \lfloor\frac{r'b+a_1}{2s'}\rfloor = q$. This proves that $s'b + q$ is an upper bound for $\lfloor\sqrt{n}\rfloor$.

Finally, let us prove that $n \ge s^2$. Since $n = s^2 + r$, this is equivalent to prove that $r \ge 0$ at the end of the algorithm. Since $a_3 \ge b/4$, we have $s' \ge b/2$, and therefore

$$q = \lfloor\frac{r'b + a_1}{2s'}\rfloor \le \lfloor\frac{r'b + a_1}{b}\rfloor \le r' \le 2s'$$

[1] See the postscript on page 7 where a new and better result is presented.

[2] Brent uses a different model where $M(n)$ stands for the cost of multiplying two n-bit floating-point numbers with a n-bit result, where the best method first computes the inverse square root using a third-order iteration with a complexity of $\frac{11}{2}M(n)$. If $M(n)$ is the cost of multiplying two n-bit integers with a $2n$-bit result, then the best method is the iteration (2.2) given by Brent, namely $x_{i+1} = x_i + (a - x_i^2)/(2x_i)$, with a cost of $5M(n)$.

since $a_1 < b$. We also have $q \leq \lfloor \frac{2s'b+a_1}{2s'} \rfloor \leq b$. Therefore $q^2 \leq 2s'b$, which implies $q^2 < 2(s'b+q) = 2s$ by just treating apart the case $q = 0$ (s' cannot be zero). This proves that if $r < 0$ before the if-statement, it becomes positive or zero afterwards. □

Complexity analysis.

Theorem 1 *The number of word operations used by algorithm* `SqrtRem` *for an input of* $2n$ *words is bounded by* $\frac{3}{2}K(n) + O(n \log n)$ *where* $K(n)$ *denotes the number of word operations to multiply two* n*-word numbers using Karatsuba's algorithm [4].*

PROOF. Let $S(n)$ the number of word operations used by `SqrtRem` for an input of $2n$ words (then b corresponds to $n/2$ words). We have $S(n) \leq S(n/2) + D(n/2) + M(n/2)$. Now $D(n/2) \leq 2K(n/2) + O(n \log n)$ using Karatsuba division as in [2], $M(n/2) \leq K(n/2)$, which gives $S(n) \leq \frac{9}{2}K(n/2) + O(n \log n)$ by induction. The theorem follows from $3K(n/2) \leq K(n)$. □

Algorithm `SqrtRem` can also be used efficiently together with fast FFT multiplication. Indeed, division with remainder can be performed in $5M(n) + O(n)$ operations [10, Theorem 9.6], which gives $S(n) \leq 6M(n)$. This is the best known complexity,[3] since Brent's value of $5M(n)$ — see the footnote on page 3 — needs an additional multiplication to get the remainder.

Square root without remainder. If the remainder is not needed, for example in a floating-point computation, then one can avoid a multiplication of size $n/2$ in the main call to `SqrtRem`, and replace the corresponding `DivRem` call by a division without remainder. The latter can be performed in $\frac{3}{2}K(n/2)$ operations [2, Section 3] or in $4M(n/2)$ using FFT [1]. Whence the total gain with respect to the square root with remainder is either $\frac{3}{2}K(n/2) \sim \frac{1}{2}K(n)$ or $2M(n/2) \sim M(n)$, giving a cost of at most $K(n)$ with Karatsuba's algorithm and $5M(n)$ with FFT. Both cases achieve the best-known complexities in this case too.

However if the remainder is not computed, one cannot decide whether it is negative in the if-statement, and the output value of s may be too large by one. There are two workarounds: either compute a few high bits from the remainder, which can surely be done in negligible time on average; or add a few zero bits to the input number. The latter is probably the simpler. Suppose we multiply the input n by 2^{2k} (i.e. add $2k$ zero bits to the right). Then after truncating the k least significant bits from the result s of the algorithm without remainder, we will get the exact result, except when those k bits are all zero, in which case subtracting 1 also affects the non-truncated part. The probability of such an event is 2^{-k}. Taking $k = \lceil \log_2 n \rceil$, we get a probability of 'failure' at most $1/n$, which gives a multiplicative factor of $1 + O(1/n)$ on average. When those k bits are zero, we multiply q and $2s'$ to get the remainder u of the division, and then proceed as usual. This requires two additional multiplications of size $n/2$, i.e. either $\frac{2}{3}K(n)$ or $M(n)$ operations. The worst

[3] See however the postscript on page 7, where a reference is given, which enables one to compute a square root in $\frac{7}{2}M(n)$ operations, and therefore a square root with remainder in $\frac{9}{2}M(n)$ operations.

case is thus $\frac{5}{3}K(n)$ or $6M(n)$. If the worst case complexity is more important, one should directly compute the square root with remainder in the Karatsuba case.

Implementation details. We implemented algorithm `SqrtRem` in the GNU MP (or GMP for short) multiprecision library [3]. For sake of clarity, we explained the algorithm in the simple case where the input number can be written exactly as $n = a_3b^3 + a_2b^2 + a_1b + a_0$ with $0 \leq a_i < b$ and $a_3 \geq b/4$. If one looks at the proof, the only assumptions used are $a_0 < b$, $a_1 < b$ and $a_3b + a_2 \geq b^2/4$, the latter ensuring that $s' \geq b/2$. Therefore the algorithm is valid in the general case too, taking as base b the largest power of the machine word base such that the input number n is greater or equal to $b^4/4$. The only remaining constraints are that n uses an even number of machine words, and the most significant one has one of its two most significant bits set. This ensures that all divisors s' are normalized, i.e. their most significant bit is set.

Computer experiments. Our experiments were made on a 366Mhz Pentium II laptop under Linux. All timings are given in milliseconds. We compared our implementation to the built-in GMP function `mpn_sqrtrem` which uses a 9-bit initial approximation, then extends it to a full-word approximation using Newton's method, and finally to the desired number of words still with Newton's method. Our implementation of algorithm `SqrtRem` wins over `mpn_sqrtrem` up from 6 words of 32-bits, i.e. up from 50 digits. We believe this can be improved further by a non-recursive version with no memory allocation.

For large operands, it was not possible to find the break-even point with Newton's method and FFT since GMP currently only multiplies by the naïve $O(n^2)$ method or Karatsuba's algorithm. We had implemented in GMP Schönhage's asymptotically optimal algorithm with complexity $O(n \log n \log \log n)$ to multiply two residues modulo $2^n + 1$ using FFT [7, 8, 9]. This algorithm is perhaps not the best one to use in practice, however our implementation is carefully optimized — much more than that of `SqrtRem`. The table in the left of Fig. 1 shows in the column `fft_mul` the time in milliseconds — multiplied by 5 — to perform an integer (or floating-point) multiplication between two numbers of w 32-bit words using Schönhage's algorithm (the modulo $2^n + 1$ needs to be twice as large as the input numbers). Therefore it gives an idea of $5M(n)$ whereas column `mpn_mul` gives the value of $\frac{3}{2}K(n)$. One can see the break-even point lies around one million digits.

Conclusion. The algorithm presented here can be viewed as a variant of the schoolboy method for square root extraction — although it is no longer taught at school — where instead of taking two digits at a time, one takes half of the whole digits at a time. In such a sense, it could be called a divide and conquer square root algorithm. On the other hand, if one looks carefully at the operations performed in `SqrtRem`, one sees that it is also a discrete variant of Newton's iteration. However, one important difference is that the multiplication appears *after* the division in each step, or in other words that multiplications have operands of half the size of those in the usual floating-point iteration, if we take apart the last multiplication which only computes the remainder.

w	digits	$\frac{3}{2}$ mpn_mul	5 fft_mul
512	4932	2.7	20
1024	9864	8.1	44
2048	19728	26	100
4096	39456	75	220
8192	78913	230	500
16384	157826	680	1200
32768	315652	2100	2900
65536	631305	6200	6500
131072	1262611	18600	15500
262144	2525222	56000	33000

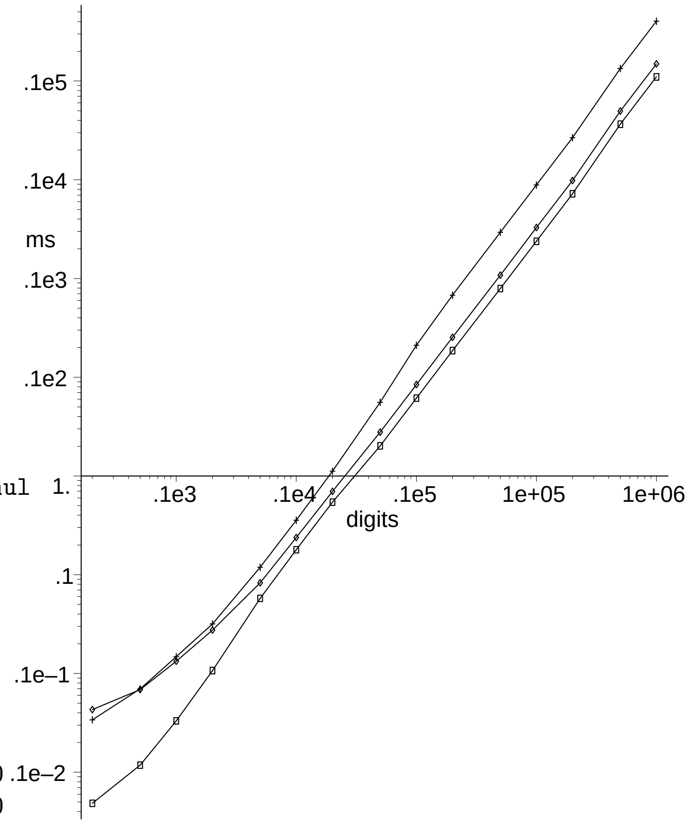

Figure 1: To the left: Table giving the number of milliseconds needed to multiply two numbers of w 32-bit words in GMP using either Karatsuba's algorithm (`mpn_mul`) or Schönhage's algorithm (`fft_mul`). To the right: graph comparing the multiplication time (`mpn_mul`, boxes) to the built-in GMP function `mpn_sqrtrem` (crosses) and to the implementation of algorithm `SqrtRem` (diamonds); both scales are logarithmic.

Apart from the computation of integer square roots, algorithm `SqrtRem` can be used to perform floating-point computations with exact rounding. Suppose one wants to compute $\sqrt{n}$ rounded to nearest. This is s iff $r \leq s$, and $s+1$ otherwise. (Here no special rule is needed for the middle case, since $(s+1/2)^2$ is not an integer.) The MPFR (Multiple Precision Floating-Point Reliable) library is precisely a library that provides exact rounding of arbitrary precision floating-point numbers. This library is currently being developed at LORIA and INRIA-Lorraine. Algorithm `SqrtRem` will most certainly be incorporated into MPFR. It can also be used for polynomial or Taylor series computations: just replace the base b by a variable x. Finally, the idea of algorithm `SqrtRem` may also be extended to cubic or higher roots.

Acknowledgements. This paper would not have been written without Henri Cohen who told me about Karatsuba division, and without Bruno Haible who pointed out the beautiful paper of Burnikel and Ziegler. Many thanks also to Richard Brent for fruitful discussions about the optimal asymptotic way to compute a square root, to Torbjörn Granlund for the great GMP library, and to my colleagues from the MPFR team for their help in that titanic project.

Postscript. This postscript was written on November 9, 1999, while the rest of the paper was written on September 1st, 1999. Christoph Burnikel pointed out the paper [5] of Karp and Markstein, which presents a new algorithm for high precision square root. The main idea is to use the classical Newton iteration for the inverse square root $x_{k+1} = x_k + \frac{x_k}{2}(1 - Ax_k^2)$, but to replace the last step by $y_k = Ax_k$ followed by

$$y_{k+1} = y_k + \frac{x_k}{2}(A - y_k^2). \tag{1}$$

The cost of Karp and Markstein's algorithm — which does not compute the square root remainder — is $\frac{11}{6}K(n)$ in the Karatsuba case, so the algorithm presented here is still better. However, in the FFT case, Karp and Markstein's algorithm costs only $\frac{7}{2}M(n)$, and is therefore better than the $5M(n)$ from the introduction. To our knowledge this value of $\frac{7}{2}M(n)$ was not known before (Karp and Markstein did no complexity analysis).

Another interesting reference is Mulders' paper [6] communicated by Mark Sofroniou. Thom Mulders found a clever way to compute the n most significant coefficients of an $n \times n$ product faster than the full product, even in the Karatsuba case. The resulting "short product" algorithm needs about $\sim 0.808\,K(n)$ operations. Mulders also presents a "short division" algorithm in $\sim 1.397\,K(n)$. This enables one to improve the bound for the square root without remainder: in the last step, the division without remainder costs only $\sim 1.397\,K(n/2)$ instead of $\frac{3}{2}K(n/2)$ using Burnikel and Ziegler's algorithm. This gives an algorithm in $\sim 0.966\,K(n)$ for the square root without remainder.

References

[1] Brent, R. P. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic Computational Complexity* (New York, 1975), J. F. Traub, Ed., Academic Press, pp. 151–176.

[2] Burnikel, C., and Ziegler, J. Fast recursive division. Research Report MPI-I-98-1-022, MPI Saarbrücken, Oct. 1998.

[3] Granlund, T. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 2.0.2 ed., June 1996.

[4] Karatsuba, A. A., and Ofman, Y. P. Multiplication of multiplace numbers by automata. *Dokl. Akad. Nauk SSSR 145*, 2 (1962), 293–294.

[5] Karp, A. H., and Markstein, P. High precision division and square root. HP Labs Report 93-93-42, Hewlett Packard, June 1993. Revised October 1994.

[6] Mulders, T. On short multiplications and divisions. Preprint, March 1998. Available via URL http://www.inf.ethz.ch/personal/mulders/papers/Shortprodquo/shortprodquo.ps.gz.

[7] Schönhage, A. Schnelle multiplikation groβer zahlen. *Computing*, 7 (1971), 281–292.

[8] Schönhage, A. Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients. In *Computer Algebra, EUROCAM'82* (1982), no. 144 in Lecture Notes in Computer Science, pp. 3–15.

[9] Schönhage, A. Tapes versus pointers, a study in implementing fast algorithms. *Bulletin of the EATCS*, 30 (1986), 23–32.

[10] von zur Gathen, J., and Gerhard, J. *Modern Computer Algebra*. Cambridge University Press, 1999.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399