Using the Parallel Karatsuba Algorithm for Long Integer Multiplication and Division

Tudor Jebelean

RISC-Linz, A-4040 Linz, Austria
Jebelean@RISC.Uni-Linz.ac.at

Abstract. We experiment with the sequential and parallel Karatsuba algorithm under the `paclib` system on a Sequent Symmetry shared-memory architecture, obtaining efficiency 88% on 9 processors for the controlled-depth version, and 71% on 18 processors for the scalable version. Moreover, we use the Karatsuba algorithm within *long integer division*, by a recent divide-and-conquer technique. The parallel version exhibits modest speed-ups (2.3 on 3 processors and 3.4 on 9 processors), however the combined speed-up over the classical sequential method is more than 10 at 500 words.

Introduction

Fast methods for operations over long integers are very important for applications like cryptography and arbitrary precision arithmetic. Since FFT is only useful for very long operands [12, 1], the practical method of choice is Karatsuba multiplication. Moreover, this algorithm is easy to parallelize: experiments have been reported in [12] on shared memory, [3] on workstation network and [1, 2] on distributed memory.

Under the `paclib` [4] computer algebra system on a Sequent-Symmetry architecture with 20 processors (Intel 386, 16 MHz), we experiment with the "controlled depth" parallel Karatsuba algorithm. The speed-up ranges from 2 (at 30 words) to almost 3 (at 100 words) on 3 processors, while on 9 processors it ranges from 2 (20 words) to 7.4 (250 words) to 8 (500 words). This is better than the speed-up on 9 processors of [2] (6.5 at 250 words and 7.5 at 500 words), and almost 2 times better than the results reported in [3].

We also experiment with a scalable algorithm using up to 18 processors. For 500 words the efficiency ranges between 93% (3 processors) and 71% (18 processors). The speed-up is significantly higher than in [12] on 12 processors (100 words: 6.8 vs. 3.0, 200 words: 8.8 vs. 3.9, 300 words: 9.5 vs. 4.7 on 300).

Long integer division has the same theoretical complexity as multiplication ([10], p. 275), however using Karatsuba method involves a loss of 15 to 30 times in speed, which leads to a break-even point of several thousand words (see [11, 8]). Also, parallelization of integer division is difficult: theoretical parallel algorithms have been designed (see [13] for a survey), but practical implementations are realized mostly for VLSI design [14] and on systolic architectures [6]. *Exact* division has been parallelized on 2 processors [11], and the method presented below can be used for further increasing the speed [5].

Here we use a novel technique from [8], which allows to use Karatsuba multiplication for division with a slow-down of only a factor of two. Embedding the parallel "fork-join" Karatsuba method into division does not yield good speed-ups, because most of the calls to the multiplication are for small-length operands. For 500 words, the speed-up on 3 processors is 2.3 (efficiency 77%) and on 9 processors is 3.44 (efficiency 38%). However, the combined speed-up over the classical sequential algorithm is quite significant.

1 Multiplication

The Karatsuba multiplication algorithm [9], [10], p. 258, is a recursive divide-and-conquer technique which replaces a multiplication by three multiplications of half-length operands. In practice the threshold for stopping the recursion is determined experimentally (6 words of 29 bits in our implementation). Table 1 shows the timings (averaged over 100 runs) of the classical algorithm (column 2), the Karatsuba algorithm (column 3) and the speed-up (column 4).

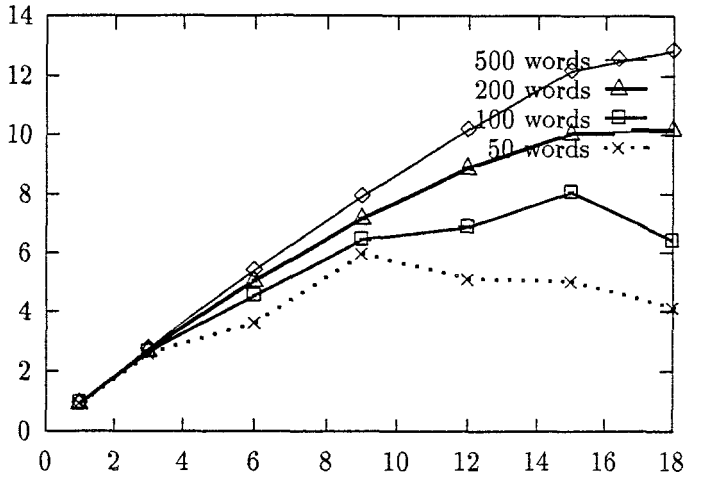
As the previous authors, we use the straight-forward "fork-join" parallelization of the Karatsuba scheme: the three multiplications involved are performed in parallel. For the "controlled-depth" approach we obtained the timings shown in Table 1. Columns 5, 8 show the absolute time, columns 7, 10 show the speed-up over the sequential Karatsuba algorithm, and columns 6, 9 show the combined speed-up over the sequential classical algorithm. The efficiency surpasses 75% on 3 processors already at 40 words, and on 9 processors at 150 words. At 500 words the efficiency is about 90%.

In order to obtain a **scalable algorithm**, we give up the control over the recursion depth and we use instead a parallelization threshold of 24 words (experimentally determined). The algorithm scales well (efficiency over 72%) until 18 processors only for the length of 500 words - see Fig. 1.

Table 1. Multiplication (up) and division (bottom) in milliseconds.

length (words of 29 bits)	classic ($\mathcal{C}$)	seq. Karatsuba ($\mathcal{K}$)		parallel Karatsuba					
				on 3 processors			on 9 processors		
		abs.	vs.$\mathcal{C}$	abs.	vs.$\mathcal{C}$	vs.$\mathcal{K}$	abs.	vs.$\mathcal{C}$	vs.$\mathcal{K}$
40	65	37	*1.76*	15	*4.33*	*2.47*	8	*8.13*	*4.63*
100	398	158	*2.52*	57	*6.98*	*2.77*	25	*15.92*	*6.32*
200	1,585	480	*3.30*	168	*9.43*	*2.86*	66	*24.02*	*7.27*
500	9,862	2,034	*4.85*	704	*14.01*	*2.89*	255	*38.67*	*7.98*
76/40	70	55	*1.27*	45	*1.56*	*1.22*	47	*1.49*	*1.17*
196/100	447	278	*1.61*	175	*2.55*	*1.59*	161	*2.78*	*1.73*
396/200	1,790	881	*2.03*	454	*3.94*	*1.94*	365	*4.90*	*2.41*
996/500	11,255	3,825	*2.94*	1,659	*6.78*	*2.31*	1,111	*10.13*	*3.44*

Fig. 1. Speed-up of the scalable multiplication algorithm.



2 Division

The classical division algorithm (see [10] p. 237) consists of a series of successive updates of the dividend A by subtracting the divisor B multiplied by the current digit of the quotient Q, and right-shifted. This process is represented pictorially by the parallelogram in Fig. 2. The final value of A will be the remainder R. The scheme introduced in [8] starts by splitting the quotient Q into its high-order part Q_H of length q_H and its low-order part Q_L of length q_L such that $q_L \leq q_H \leq q_L + 1$, and also the divisor B into its high order part B_H of length b_H and its low-order part B_L such that $q_H + 3 \leq b_H$. Computing the high-order part Q_H of the quotient would normally require to perform the updates in the upper half of the parallelogram of Fig. 2 (areas 1 and 2). However, Krandick [11] proves that in most cases it is enough to update only 3 words below the lowest digit of A needed for the lowest quotient digit to be computed (i.e. down to the vertical line crossing area 1). This allows to split the updates of the dividend into 4 parts as shown by 1, 2, 3, 4 in Fig.2, and to perform them in the order of

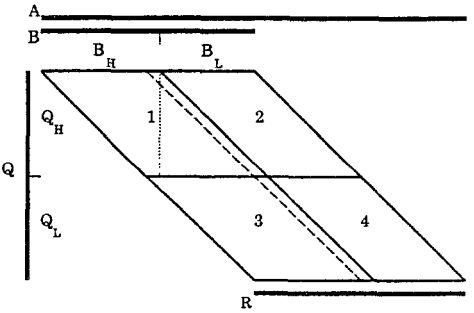


Fig. 2. Organization of the dividend updates $(Q, R \longleftarrow A : B)$.

the numbering: parts 2 and 4 by Karatsuba multiplication, and parts 1 and 3 by the same recursive technique. Under a certain threshold (15 words in our implementation), the recursion is replaced by classical division. This method has Karatsuba-like complexity, and the number of digit-multiplications is only 2 times higher than in Karatsuba multiplication (for details see [8]).

The speed-up can be increased by using the parallel "fork-join" version of the Karatsuba multiplication algorithm presented in the previous section. Table 1 shows the results of the experiments using the "controlled depth" algorithms on 3 and 9 processors. As expected, the efficiency of the parallel division is much lower than the one of multiplication, because the Karatsuba multiplication is called many times, and mostly with short-length operands. However, the combined speed-up over the classical sequential algorithm is quite significant: on 9 processors it ranges from almost 3 at 100 words, to more than 10 at 500 words.

Further details on the algorithms, timings, and the behavior of the parallel implementation. are given [7], which is available on the internet.

References

1. G. Cesari and R. Maeder. Parallel 3-primes FFT algorithm. In J. Calmet and C. Limongelli, editors, *DISCO'96*, pages 174–182. Springer LNCS 1128, 1996.
2. G. Cesari and R. Maeder. Performance analysis of the parallel Karatsuba multiplication algorithm for distributed memory architectures. *J. of Symbolic Computation*, 21:467–473, 1996.
3. B. Char, J. Johnson, D. Saunders, and A. P. Wack. Some experiments with bignum arithmetic. In H. Hong, editor, *PASCO'94*, pages 94–103. World Scientific, Singapore, 1994.
4. H. Hong, W. Schreiner, and A. Neubacher. The design of the SACLIB/PACLIB kernel. *J. of Symbolic Computation*, 19(1–3):111–132, 1995.
5. T. Jebelean. Exact division with Karatsuba complexity. Technical Report 96-31, RISC-Linz, December 1996.
6. T. Jebelean. Integer and rational arithmetic on MasPar. In J. Calmet and C. Limongelli, editors, *DISCO'96*, pages 162–173. Springer LNCS 1128, 1996.
7. T. Jebelean. Applications of the parallel Karatsuba algorithm to long integer multiplication and division. Technical Report 97-08, RISC-Linz, `http://www.risc.uni-linz.ac.at/library`, February 1997.
8. T. Jebelean. Practical integer division with Karatsuba complexity. In W. Kuechlin, editor, *ISSAC'97*. ACM Press, 1997.
9. A. Karatsuba and Yu Ofman. Multiplication of multidigit numbers on automata. *Sov. Phys. Dokl.*, 7:595–596, 1962.
10. D. E. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, 1981.
11. W. Krandick and T. Jebelean. Bidirectional exact integer division. *Journal of Symbolic Computation*, 21:441–455, 1996.
12. W. Kuechlin, D. Lutz, and N. Nevin. Integer multiplication on PARSAC-2 on stock microprocessors. In *AAECC-9*, pages 216–217. Springer LNCS 539, 1991.
13. S. Lakshmivarahan and S. K. Dhall. *Analysis and design of parallel algorithms: Arithmetic and matrix problems*. McGraw-Hill, 1990.
14. E. E. Swartzlander, editor. *Computer Arithmetic*, volume 2. IEEE Computer Society Press, 1990.