Numerical evaluation of complexity of integer multiplication algorithms

Grigory Stepanov[1]

Institute for Information Transmission Problems
http://www.iitp.ru

Abstract. Modern coding and cryptography methods require fast multiplication algorithms over integers or polynomials. Two significant theoretical evaluation were stated by authors of Karatsuba, Toom-Cook and Schonhage-Strassen algorithms, which reqiure $O(n^{\log_2 3})$, $O(n^{\log_3 5})$ and $O(n \cdot \log n \cdot \log\log n)$ respectively. In this paper Karatsuba execution was examined and a multiplication algorithm via FFT was proposed and investigated. Approximate ranges of "best" performance and of Multiple Precision, Karatsuba and FFT multiplication were evaluated.

Keywords: Integer multiplication algorithm · Karatsuba · FFT · Schonhage Strassen

1 Introduction

The problem of fast integer multiplication occurs in cryptography methods because they requires operating with long integers or polynomials, which are almost equal in complexity. There some widely known theoretical asymptotic evaluations, which were implied from correlating algorithms. The most frequently used are Multiple Precision[5], Karatsuba[3], Toom-Cook[1] and Schonhage-Strassen[2], which have decreasing asymptotics and increasing constants in their evaluation. Hence each algorithm has its own operating range. In this work Multiple Precision , Karatsuba and long FFT multiplication (was proposed by the author of the paper) algorithms were implemented and compared in order to learn their operating ranges.

2 Background

2.1 Multiple-precision multiplication

The most famous pencil and paper algorithm that is taught in grade school, called multiple-precision multiplication in [5] requires $O(n^2)$ operations, where n is both integers' length.

2.2 Karatsuba algorithm

The first algorithm that appeared to be lower $O(n^2)$ was provided by Anatoly Karatsuba in his work [3]. The idea is the following:

$$A \cdot B = (A_1 \cdot 10^m + A_2)(B_1 \cdot 10^m + B_2) = A_1 B_1 \cdot 10^{2m} + (A_1 B_2 + A_2 B_1)10^m + A_2 B_2 \tag{1}$$

where

$$A_1 B_2 + A_2 B_1 = (A_1 + A_2)(B_1 + B_2) - A_1 B_1 - A_2 B_2 \tag{2}$$

$$2m = \log_{10} A = \log_{10} B$$

which gives us "divide and conqueror" recursive algorithm, with the following complexity:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 3}) \tag{3}$$

2.3 Toom-Cook algorithm

Toom-Cook [1] algorithm is a generalization of Karatsuba. In Toom-k the given numbers are divided into k parts (i.e. Karatsuba is Toom-2). Let's define $m(k)$ as number of multiplications, $s(k)$ – number of additions (i.e. $m(2) = 3, s(k) = 8$ for Karatsuba (Toom-2) multiplication algorithm), assuming all low level operations are equal in complexity. Let be $T_k(n)$ – complexity of Toom-k algorithm, $T_1(1) = 1$ due to previous assumption. Then

$$T_k(n) = m(k)T_k(\tfrac{n}{k}) + s(k) = m(k)(m(k)T_k(\tfrac{n}{k^2}) + s(k)) + s(k) = \dots$$

$$\dots = m^{\log_k n}(k) + s(k)\tfrac{m^{\log_k n}(k)-1}{m(k)-1} \leq c(k)n^{\log_k m(k)}$$

.

If $m(k) = 2k-1,\ T_k(n) \approx c(k)n^{1+\frac{1}{\log k}},\ c(k) = 1 + \frac{s(k)}{2k}$

2.4 Schonhage-Strassen algorithm

Basic idea The idea of the algorithm is based on DFT [10] and Convolution Theorem [11]. According to this theorem:

$DFT^-1(DFT(P) * DFT(Q)) = PQ$, where $P,\ Q$ - polynomials, $*$ - convolution.

Which allows to multiply polynomials with complexity equal to the complexity of certain DFT algorithm, which can be lower than $O(n^{1+\varepsilon}),\ \forall\varepsilon > 0$.

Choice of the ring The DFT can be perform over any cyclic group. Most commonly it is performed over the group of complex roots of unity or $\mathbb{Z}/n\mathbb{Z}$. The key idea is to choose $n = 2^m + 1$, where $m \in \mathbb{N}$. Such approach gives the following benefits: any value can be easily transformed into $2^m + 1$ using binary operations, certain group of $2^m + 1$ roots of unity can be easily described in such ring with a controllable accuracy of calculations.

$\sqrt{n}$ optimization Schonhage-Strassen works as follows: divide input numbers into m and performs multiplication via FFT with a special accumulation procedure for overflow digits. The main optimization, that giving $O(n \cdot \log n \cdot \log\log n)$ complexity is division input into approximately $m = \sqrt{n}$ parts.

3 Multiplication via long FFT (LFFTM)

3.1 Basic idea

In this work a multiplication algorithm was proposed and investigated. This algorithm works as follows: performs FFT of size equal to $2n$ over input vectors, that represent multipliers, then makes a convolution and applies inverse FFT to the result of convolution, and after that makes a digit shifting.

3.2 Motivation

The motivation of such approach is to simplify Schonhage-Strassen, because Schonhage-Strassen has not very much levels of recursion in practice. That gives a consideration that may be the only first level of recursion is enough for good performance.

3.3 Estimation

FFT is executed with the size equal to number length $N = \frac{n}{m}$, where m is a minimal section of a long number defined by CPU/GPU low-level arithmetic. FFT provides a transform of multipliers and their product and requires $4 \log n$ low-level arithmetic to preserve accuracy. The complexity of this algorithm can be estimated as $O(n \log n \log^2 n)$, where $\log^2 n$ occurs due to low-level arithmetic operations.

4 Soft- and hardware

4.1 BigInt lib

In process of investigating Karatsuba and MPM performance a C++ open-source library "BigInt" (see source code on [8]) was made by the author of the paper. Its kernel is class BigInt with basic MP addition and subtraction (also see [5]), implemented Karatsuba multiplication (with all optimizations described below), and basic MP division, that uses Karatsuba multiplication. The class also includes most of the basic overloaded operators, so the library can be easily used for any reasons.

4.2 FFT implementation

To make an experiment on simplified SS algorithm a state of the art FFT GPU library cuFFT [7] was used, algorithm was performed on "NVIDIA GeForce GT 1030" GPU. The cuFFT Library uses the Cooley-Tukey [9] algorithm to reduce the number of required operations to optimize the performance of particular transform sizes. This algorithm expresses the DFT matrix as a product of sparse building block matrices. The cuFFT Library implements the following building blocks: radix-2, radix-3, radix-5, and radix-7. Hence the performance of any smooth transform size that can be factored as $2^a \cdot 3^b \cdot 5^c \cdot 7^d$ (a, b, c, and d – non-negative integers) is optimized in the cuFFT library. There are also radix-m building blocks for other primes, m, whose value is ¡ 128. When the length cannot be decomposed as multiples of powers of primes from 2 to 127, Bluestein's algorithm is used. cuFFT has only real and complex value FFT implementation, hence the algorithm has an logarithmic error which increases with the number of operations, but still has the same complexity as if it was executed over $\mathbb{Z}/n\mathbb{Z}$.

Correctness All source code that was used for experiments is on the GitHub [8] and can be checked on correctness.

5 Numerical experiment

5.1 Karatsuba vs. MPM

The primal interest is to compare MP multiplication approach and Karatsuba method. But due to right transition to MP multiplication in the end of recursion

Karatsuba is always not slower. To achieve such performance the following problems should have been solved. The first one is the moment when to switch to MP algorithm depending on length of integer in current recursive call. According to numerical experiment (see Fig. **??**, Fig. **??**, comparison between different length of multipliers to switch to MP method) the first constant is ~ 50 digits in 10-digit arithmetics. Which makes this Karatsuba performance not worse that MP (see Fig. 1, comparison between optimized Karatsuba and MP).

The second problem is whether to chose Karatsuba or MP when the length of the multipliers are different. This constant can be counted from the following considerations:

$$T_{primal}(n,m) = O(n \cdot m), T_{Kar}(n) = O(max(n,m)^{\log_2 3})$$

hence

$$T_{Kar}(n,m) > T_{primal} \Leftrightarrow m < C_1 n^{\log_2 3 - 1} + C_2$$

numerical experiment shows that (see Fig. 2)

$$C_1 = 6,6, C_2 = 60.7$$

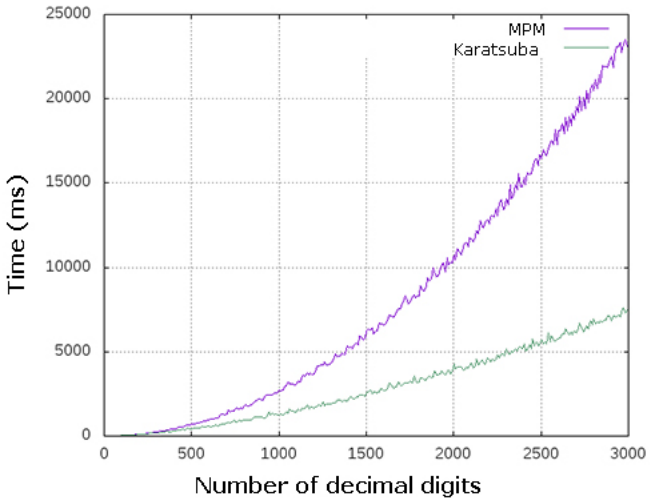


Fig. 1: Time complexity comparison between primal and Karatsuba method with switching constant equals 50.

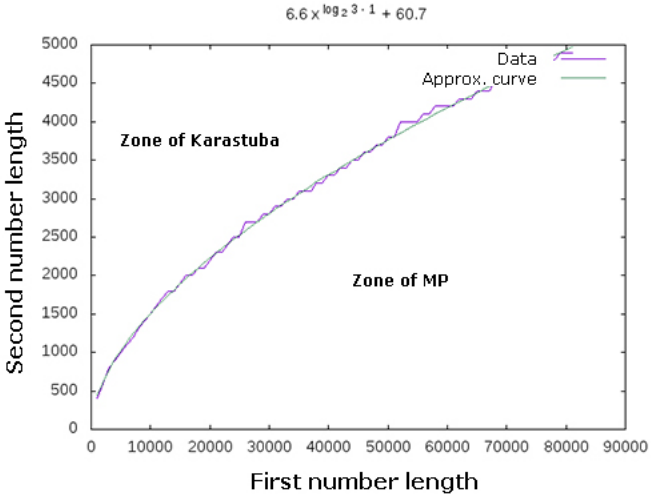
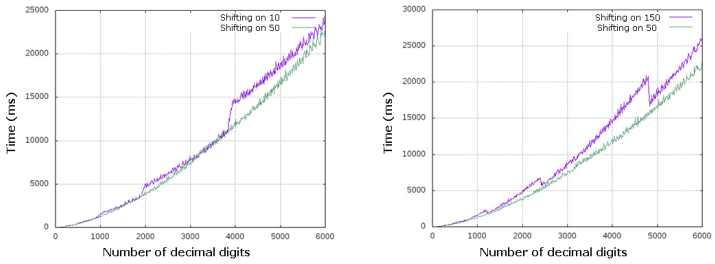


Fig. 2: Researching n, m ratio to choose MP multiplication instead of Karatsuba, when length of multipliers is unequal.



10 vs 50 50 vs 150

Fig. 4: Comparison between switching to MP different amount of digits in order to exit recursion when performing Karatsuba multiplication.

5.2 Karatsuba vs. GPU LFFTM

Due to desire to reach faster results an attempt to FFT multiplication on GPU using cuFFT library on CUDA architecture for NVDIA GPU [7]. As we see on plots it starts to perform faster on amount of digits from 1000 to 2000. The result has such variance (see Fig. 6) because of the specifics of FFT implementation which execution speed depends on polynomyal's length factorisation. Another issue that have an impact on FFT complexity is time that graphic card requires for initializing memory and prepare for execution, which according to Fig. 6) is 600 ms. That means that the complexity of several performances would be lower, because it would take only one time for all preprocessing. According to this thought we can assume that GPU long FFT multiplication algorithm implementation working range starts from $\sim 750 - 1750$ digits.

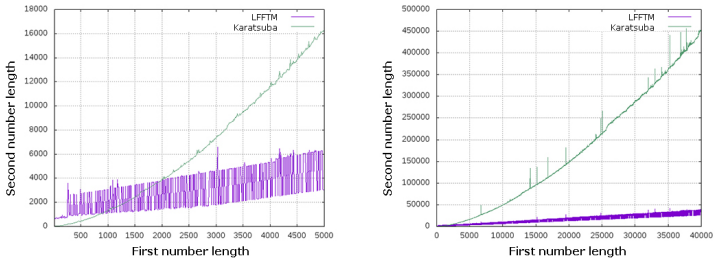


Fig. 6: Karatsuba and GPU LFFTM comparison.

6 Conclusion

6.1 What do we have

In this research hyperparameters for better Karatsuba performance were achieved. Due to this, Karatsuba method can be performed not slower than regular MP independently from choice of number length. Also a multiplication algorithm via FFT was proposed and investigated, operating range on GPU was checked, which appeared to be $\sim 750-1750$ digits (in comparison to Karatsuba on CPU). And a simple and useful library to operate with big numbers was created.

6.2 Perspectives

According to the article [4], the Karatsuba tree can be flattened, which means, this algorithm can be performed in parallel. But this needs further research, because parallel processing requires heavy memory manipulations, which might lower the complexity on working ranges. Though FFT multiplication is still performs better on very big numbers, as it was shown on 750 – 1750 decimal digits,

better results could be reached by using better implementations of FFT algorithm and by using state of the art GPU. Long FFT multiplication could be also optimized by choosing appropriate radix to perform multiplication over. Another research interest is to optimize Schonhage-Strassen with the same approach that Karatsuba was, by switching to simpler algorithm on the lower layers of recursion.

Acknowledgements I would like to express my appreciation to my scientific director and consultant Valentin Afanasyev from IITP RAS for help and support in writing this paper.

References

1. D. Knuth. The Art of Computer Programming, Volume 2. Third Edition, Addison-Wesley, 1997. Section 4.3.3.A: Digital methods, pg.294.
2. A. Schnhage and V. Strassen, "Schnelle Multiplikation grosser Zahlen", Computing 7 (1971), pp. 281292.
3. A. Karatsuba and Yu. Ofman (1962). "Multiplication of Many-Digital Numbers by Automatic Computers". Proceedings of the USSR Academy of Sciences. 145: 293294.
4. Th. Baruche, Flattening Karatsubas recursion tree into a single summation, https://arxiv.org/pdf/1902.08982.pdf (2019)
5. A. Menezes, P. van Oorschot, S. Vanstone Handbook of Applied Cryptography (1996)
6. Overview of Magma V2.9 Features, arithmetic section
7. CUDA Toolkit 4.2 CUFFT Library, March 2012
8. https://github.com/higheroplane/GaloisField/tree/BigInt
9. Cooley, James W.; Tukey, John W. (1965). "An algorithm for the machine calculation of complex Fourier series".
10. Brigham, E. Oran (1988). The fast Fourier transform and its applications. Englewood Cliffs, N.J.: Prentice Hall.
11. Katznelson, Yitzhak (1976), An introduction to Harmonic Analysis
12. D. Knuth. The Art of Computer Programming, Volume 2. Third Edition, Addison-Wesley, 1997. Section 4.3.3.A: Digital methods, pg.294.