ZOT-M_K: A NEW ALGORITHM FOR BIG INTEGER MULTIPLICATION

BY

SHAHRAM JAHANI

**Thesis submitted in fulfillment of the requirements
for the degree of
Master of Science**

June 2009

Acknowledgements

There is no word to express my thankful respecting to god that always has mercy on me.

This is a great opportunity to express my deep appreciation to my supervisor, (Assoc. Prof.) Dr. Azman, who spares his invaluable patience in guiding me in my research work. I am short in word to express his contribution to this thesis, with criticism, suggestions and discussions. I'd like to thank him because of having trust in me and support me to carry out my research with encouragement. Also I'd like to thank him for his support on my research through his grant. I'm pleased to make grateful acknowledgement to the Dean and all staff members of the School of Computer Science, USM, for making all this possible.

I would like to express my deep feeling to my family members especially to my wife and my son for their kindness and encouragement.

TABLE OF CONTENTS

CHAPTER TWO: LITERATURE REVIEW

CHAPTER THREE: ZOT-M_K

CHAPTER FOUR: IMPLEMENTATION

CHAPTER FIVE: RESULTS AND DISSCUTIONS

CHAPTER SIX: SUMMARY AND CONCLUSION

LIST OF TABLES

LIST OF FIGURES

Page

ZOT-M_K: SATU ALGORITMA BARU UNTUK PENDARABAN INTEGER BESAR

Abstrak

Pendaraban nombor besar banyak digunakan dalam pengkomputeran saintifik. Walau bagaimanapun, terdapat hanya beberapa alogritma yang ada kini, memperoleh keefisienan mereka melalui pendaraban integer besar. Oleh sebab pendaraban integer tidak natif terhadap struktur penomboran arkitektur komputer bagi bit dan bait, maka pelaksanaan algoritma tersebut akan menjadi agak lambat

Penyelidikan ini menekankan algoritma pendaraban nombor besar berdasarkan simbol yang terekstrak daripada sistem nombor perduaan. Kami namakan struktur penomboran baru ini sebagai "ZOT". Algoritma baru bagi pendaraban nombor besar, ZOT-M_K, dibina daripada gabungan algoritma Karatsuba dan struktur ZOT.

Bagi tujuan penilaian, kami merangsang suatu persekitaran yang mampu mengendalikan nombor yang besar untuk membandingkan prestasi algoritma yang dicadangkan terhadap algoritma Karatsuba, yang sudah dikenali ramai. Keputusan berjulat di antara julat nombor 25 hingga 5000 bit, menunjukkan bahawa kadar pemampatan nombor tersebut yang diwakili oleh struktur ZOT terhadap perwakilan normal perduaan adalah 41 peratus. Oleh itu, secara teorinya, dalam purata halaju perlakuan ZOT-M_K, ia sepatutnya dua kali lebih laju daripada algoritma Karatsuba.

Walau bagaimanapun, disebabkan penggunaan memori ZOT-M_K yang efisien, yang menghilangkan memori kelui, keputusan eksperimen menunjukkan bahawa masa perlakuan purata daripada ZOT-M_K dalam nombor berjulat rendah (25 bit hingga 1 Kbit) adalah lebih kurang 35 peratus daripada alogritma Karatsuba. Nilai purata ini akan berkurangan bagi nombor berjulat tinggi (1 Kbit hingga 5 Kbit) sehingga 25 peratus.

Kesimpulannya, keputusan yang diperoleh mengesahkan keefisienan algoritma pendaraban ZOT-M_K terhadap algoritma Karatsuba, yang merupakan "de-facto standard" bagi alogritma pendaraban nombor besar.

ZOT-M_K: A NEW ALGORITHM FOR BIG INTEGER MULTIPLICATION

Abstract

Multiplication of big numbers is being used heavily in scientific computation. However, there are only a few existing algorithms today that gain their efficiency through the multiplication of the big integer characteristic. Since the multiplication on integers is not native to the computer architecture numbering structure of bits and bytes, such algorithms are bound to be a bit slower on the implementation.

This research focuses on big number multiplication algorithm that is based on the symbols extracted from the binary numbering system. We named the new numbering structure as "ZOT". The new algorithm for big numbers multiplication, ZOT-M_K, is constructed from the combination of Karatsuba algorithm and the ZOT structure.

For evaluation purposes, we simulate an environment capable of handling big numbers to compare the performance of the propose algorithm against the well known Karatsuba algorithm. Over the range of 25 to 5000 bits numbers, results show that the compression rate of those numbers represented by the ZOT structure against the normal binary representation is 41 percent. Therefore, theoretically, in average the execution speed of ZOT-M_K should be about double of the Karatsuba

algorithm. However because of efficient memory utilization of ZOT-M_K that eliminates extensive memory paging, the experimental result shows the average execution time of ZOT-M_K in lower range numbers (25 bits to 1Kbits) is about 35 percent of the Karatsuba algorithm. This average value will decrease for higher range numbers (1Kbits to 5Kbits) to 25 percent.

In conclusion, the available results validate the efficiency of the ZOT-M_K multiplication algorithm against Karatsuba algorithm, which is currently the de-facto standard for big number multiplication algorithm.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Before the widespread use of data processing equipment, the security of information was primarily provided by physical and administrative means. An example of this is the former use of rugged filing cabinets with a combination lock for storing sensitive documents. As the computer was introduced, a need for an automated lock for protecting sensitive documents became evident [1]. As society's dependence upon digital computing and telecommunication increases, the need for quantitative computer security increases proportionally. In this study, we will scrutinize the different models of numerical systems, and institute a new method to increase the time efficiency and decreases the memory usage to facilitate the increased overall efficiency of the security (cryptographic) computational algorithm.

As stated by McLennan, Cryptography is a discipline of Mathematics and Computer Science concerned with information security using encryption and authentication techniques [2]. The discipline of writing messages as ciphertext, with the aim of protecting a secret from adversaries, interceptors, intruders, interlopers, eavesdroppers, opponents or simply attackers [3]. The application of information security must be instituted into the software programs, through mathematical algorithms with the proper procedural techniques to create an efficient system of data integrity. In Computer

Science, a numeral system is a mathematical notation of number that represents a positional notation allowing computational applications. This allows internal representation of arbitrarily large integers or arbitrarily precise rational numbers and the arithmetic operations on such numbers. Numbers are typically stored as (ratios of) digit lists which can grow using dynamically allocated memory. The most prevalent need for multiple precision arithmetic, often referred to as “bignum” or “big number” math is within the implementation domain of Cryptography, Mathematics, Cosmology, Statistical Mechanics and others [4]. Yet, the most widespread usage of bignum arithmetic is probably for Cryptography. In this study, we will focus on two aspects of the bignum system, the application and the computation.

The modern field of Cryptography can be divided into main two areas of study, the symmetric-key and asymmetric-key cryptography. Symmetric-key cryptosystems use the same key for encryption and decryption of a message. A significant disadvantage of symmetric ciphers is the issue of key management. To maintain the integrity of the sensitive data, it is necessary for each distinct pair of communicating parties to ideally share a different key, and perhaps for each cipher text exchanged as well. Asymmetric-key (also, more generally, known as public-key) cryptography is conceptualized with two different but mathematically related keys: a public key and a private key. Asymmetric-key cryptosystem is constructed such that calculation of one key (the 'private key') is computationally infeasible from the other key (the 'public key'), even though the keys are related. In asymmetric-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret, known only to the

receiving party. Hence, the public key is typically used for encryption, while the private key is used for decryption [3].

RSA [5] is a type of asymmetric-key cryptosystem, which uses modular exponential module on the product of two large prime numbers. RSA encryption is defined by the exponential notation of $(g^e)^d \bmod m = g$. The modulus which is based on $n = pq$, where p and q are big prime numbers, with an exponent e, which is relatively prime to $(p-1)(q-1)$. To produce a valid private key, the values of two large primes must be found. Yet, due to the size of the value of p and q, the length of time to find the values may be lengthy. Hence, the time efficiency is low, resulting in the decrease in the overall effectiveness of the cryptosystem.

Due to the difficulty of factoring large numbers using the RSA cryptosystem, EIGamal [6] cryptosystem based on the discrete logarithm was theorized. The discrete logarithm uses number raised to large exponentials, defined by modulo p that relies on the difficulty of the inverse computation of the discrete logarithm. Hence, even though the EIGamal cryptosystem is efficient in comparison to the application of the RSA cryptosystem, due to the complexity of the arithmetic module, its simulation will cause a need for high memory capacitance, as well as low time efficiency.

In most of the cryptosystems, there is a need of big number calculation, especially for the multiplication of large numbers. The three most popular big number multiplication algorithms are the Karatsuba-Ofman, Toom-Cook, and FFT[1].

[1] -Fast Fourier Transform

The Karatsuba-Ofman [7] method is an efficient procedure for performing the multiplication of large numbers. It reduces the multiplication of two n-digit numbers to at most single-digit multiplications. It is therefore faster than the classical algorithm, which requires n^2 single-digit products. In other words, the multiplication of a 2 n-digit integer is reduced to two n-integers multiplications, one (n+1) digits multiplication, two n-digit subtractions, and two 2 n-digit additions. This form of binary splitting can be used to analyze the efficiency of the program, using the basic steps of the recursive algorithm, which is n^c , where $c = log_2 n$. Hence, if a computer has a 32-bit multiplier, one could choose $B = 2^{31} = 2{,}147{,}483{,}648$ or $B = 10^9 = 1{,}000{,}000{,}000$, and store each digit as a separate 32-bit binary word. Then the sums $x_1 + x_0$ and $y_1 + y_0$ will not need an extra carry-over digit and the Karatsuba recursion can be applied until the numbers are only 1 digit long. Therefore, the Karatsuba algorithm allows hardware implementation performance with fewer shifts. Hence, it increases the time efficiency of the application.

Toom-Cook algorithm is a well known convolution algorithm that was first proposed in 1963 [8]. If we construct two sequences using the polynomial coefficients of two polynomials, when their linear convolution is computed, the elements of the resulting sequence give the coefficients of the product of the two polynomials. Hence, any linear convolution algorithm may easily be adapted to compute polynomial multiplications.

The Toom–Cook algorithm works by treating the two operands as polynomials of maximum degree $k - 1$. This is done by partitioning the integer representations into k

digits. Both polynomials are evaluated at $2k - 1$ points and multiplied together. Solving the linear system of equations gives the coefficients of the product [3].

The usual convolution of polynomial multiplication is less efficient than the Fast Fourrier Transform (FFT) [9]. Convolution is a bilinear form associated to the syntactical encoding of polynomials as a coefficient list, whereas FFT recalls that a polynomial is a function and hence, it takes advantage of the semantical features of the polynomials to multiply (FFT evaluates polynomials at sufficiently many points and then interpolates to get the coefficients of the polynomial product) [9]. Shonhage and Strassen [10] introduced a Discrete Fourrier Transformer (DFT)-based integer multiplication method that achieves an exciting asymptotic complexity of $O(n \log n \log\log n)$.

1.2 Problem Statement

It is apparent that the multiplication of big numbers is being heavily used in area such as Cryptography. However there are only a few existing big number multiplication algorithms that gain their efficiency through the manipulation of the big integer characteristics. Since the manipulation on integers is not native to the computer architecture numbering structure of bits and bytes, such algorithms are bound to be a bit slower on the implementation. Therefore, in this research we are working on a new big number multiplication algorithm that is based on the symbols originated from the binary numbering system.

1.3 Research Objectives and Scope

The main objectives of this research are listed as below:

Table 1.1: Main objectives of the research

Main Objectives	Activities	Details
Proposing new model for representation of big numbers	Introducing big-digits	Definitions, rules and different types
	Introducing ZOT structure	Converting binary numbers to ZOT representation
Proposing new model for big numbers multiplication algorithm	Introducing big-digits multiplication algorithm	Definitions, rules and different types
	Introducing big number multiplication algorithm	Mixing ZOT and Karatsuba
Evaluation of the proposed algorithms	Simulation	Simulation of ZOT, ZOT-M_k and Karatsuba multiplication for big numbers
	Analyzing and evaluation	Evaluate the efficiency of ZOT and ZOT-M_k algorithms

As we can see from Table (1.1) the main objectives of this research are:

- To propose a new representation of big number system that we call ZOT. A tabulated system will be created for the conversion of binary system to the ZOT structure.
- To propose a new algorithm for big number multiplication, this is a combinatory method of the ZOT structure and the Karatsuba algorithm.
- To analyze and evaluate the new proposed algorithms in comparison with similar algorithms.

1.4 Research Methodology

As mentioned in Table (1.1), the research methodology, is divided into three main parts:

- Designing the proposed model
- Implementation and evaluation the proposed model
- Analyzing the efficiency of the proposed model

1.4.1 Designing and Proposed Model

We first propose a new structure for representation of Bignum, driven from binary representation of numbers, called ZOT. The main differences between our multiplication algorithm and others are derived from the structure and related properties of ZOT.

The first step is to introduce and define the main elements and concepts of ZOT. Big-digits is a general name in the ZOT structure. Big-digits are categorized to three types:

- Big-zero: we use the symbol of A_{ZL} to represent this type of "Big-digits". Any group (in binary representation) of "0" is a big-zero. '*L*' is the length of "0" elements in A_{ZL} .

  Example: A_{Z5}*="00000"* and A_{Z7}*="0000000"*

- Big-one: we use the symbol of A_{OL} to represent this type of "Big-digits". Any group (in binary representation) of "1" is a big-one. '*L*' is the length of "1" elements in A_{OL} .

  Example: A_{O5}*="11111"* and A_{O7}*="1111111"*

- Big-two: we use the symbol of A_{TL} to represent this type of "Big-digit". Any group (in binary representation) of "10" that ends to "1" is a big-two. '*L*' is the length of "10" elements in A_{TL} .

  Example: A_{T5}*="10101010101"* and A_{T7}*="101010101010101"*

Concepts and methods for converting binary to ZOT representation, and vice versa is the main contribution of this section.

The second step is extending our discussion to define new operations for Big-digits. Multiplication of Big-digits is the main operation in which we will detail our work in this section. Big-digit multiplication is categorized into four types of multiplication as shown below:

- ZOT-M_0: Big - zero – Big - digits multiplication
- ZOT-M_{11}: Big - one – Big - one multiplication
- ZOT-M_{12}: Big - one – Big - two multiplication
- ZOT-M_{22}: Big - two – Big - two multiplication

We will proceed with our work by presenting new algorithms for all types of Big-digits multiplication as mentioned above.

Extending the concept of multiplication from Big-digits multiplication to big number multiplication (with the help from literature) is the next conceptual step in our research. Integrating ZOT and Karatsuba, the well-known multiplication algorithm for bignum multiplication is the last step in this phase. According to our literature review, Karatsuba multiplication algorithm is a good candidate to be coded with the ZOT structure. Firstly, Karatsuba algorithm is more efficient than school-book method for bignum multiplication and secondly, Karatsuba algorithm is a special case of the Toom-Cook algorithm. Consequently our result can be extended to a wide range of big numbers multiplication algorithms such as the Toom-Cook algorithm as well.

1.4.2 Implementation and Evaluation of the Proposed Model

To provide with a proper evaluation tool is the purpose of this section. This part of work is divided and implemented as below:

- Recognizing the specification of the software library
- Providing big number requirements in the software library
- Simulating of ZOT , ZOT-M_K and Karatsuba algorithms
- Evaluating of Karatsuba and ZOT-M_k algorithms

C++ is as a powerful language with the ability to simulate low level operation, therefore C++ is our choice of computer language for the implementation and evaluation. For compatibility with our requirements, we need a complementary library. We create the library especially to handle the followings:

- Managing bignum as input/output parameter
- Computational functions for bignum
- Logical functions for bignum
- Special algorithms
- Evaluation algorithms

1.4.3 Analysis Results

We performed a detailed analysis on the results obtained from simulation scenarios. We analyzed the effectiveness of the proposed algorithms over our simulated range of numbers.

1.5 Research Contributions

Contributions of this thesis are as follows:

- Propose a new structure representing big numbers
- Propose a new multiplication algorithm for big-digits
- Propose a new multiplication algorithm for big numbers

1.6 Thesis Outline

The thesis is organized into six chapters:

Chapter One: Provides an overview of the thesis content and directions of the thesis

Chapter Two: Reviews in brief the number theory used in Cryptography. Then the chapter continues with related works on big number multiplication as one of the main operations in cryptography computation. Finally, well known algorithms in this area are studied and compared against each other.

Chapter Three: ZOT-M_K as the new algorithm is being introduced in this chapter. New related concepts to ZOT structure such as Big-digit's, Big-zero's, Big-one's and Big-two's will be defined. The main algorithm for Big-digits multiplication is discussed in detail as well.

Chapter Four: Detailed implementation of ZOT-M_K with the most important functions is presented. This chapter will focus on the main parts of the algorithm that

has been designed in C++. Single ZOT and Double ZOT are two variations of ZOT algorithms that we will explain in detail, as the fundamental algorithms for fast multiplication algorithms that we are proposing.

Chapter Five: Research results and analyses are presented in this chapter. General discussion about the results as a whole is being presented in this chapter as well.

Chapter Six: Concludes the thesis and presents the future direction of the research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Many modern scientific computations require the manipulation of numbers that are too big to fit in a word of memory [12]. Big numbers are a compound representation for integers, created for scientific computation. Big number multiplication is one of the fundamental algorithms in this representation, that many mathematicians and computer scientists are continuously working to come up with lower complexity algorithms [7, 8, 10]. Knuth [11] has dedicated a significant part of the second volume in his Art of Computer Programming book to such algorithms, where he also discusses the School-book arbitrary precision algorithms, and various advanced multiplication algorithms. In this chapter we will proceed to the most popular and best known algorithm of big numbers multiplication, specifically Schoolbook, Karatsuba [7], Toom-Cook [8] and Shonang-Strassen [10] multiplication algorithms. We will study these algorithms and their variants. Algorithm complexity, as an analysis tool will be discussed as well.

In this chapter, Karatsuba algorithm has been given more attention than the other algorithms, specifically because our proposed method for improving the efficiency of multiplication algorithm has been implemented based on this algorithm. On top of that, as we will discover in this chapter, some variants of Karatsuba [13, 14] are more

efficient than the other well-known big numbers multiplication algorithms (especially for the use in the cryptography).

2.2 Multiplication Algorithm

Multiplication is the process of repeating additions of the same number [15]. On the other hand a multiplication algorithm is a method of multiplying two numbers. Depending on the size of the numbers, different algorithms will be used. Consequently many efficient multiplication algorithms had been proposed.

Introducing important related concepts such as *complexity of computation* is necessary when analyzing such algorithms. Employing this concept will help us to compare and evaluate different algorithms in a well-structured manner.

2.2.1 Computational Complexity

Any given algorithm has a certain complexity, which is essentially a measure for how long it takes to run the algorithm [17]. Complexity theory analyzes the difficulty of computational problems against many different computational resources such as time, space, randomness, alternation, etc. [18, 19, 20, 21].

One of the main objectives of computational complexity theory is to categorize algorithms. Thus for designing the cryptosystems, computational complexity can help us to design algorithms that are easy to use but hard to break [3].

Running time is the most important computational resources and it depends on the basic steps taken by an algorithm. Typically, the running time is measured as a function of the input length. For numerical problems, we assume the input is represented in binary, so the length of an integer n is roughly $\log_2 n$ [11]. For example, adding two n-bit numbers has running time proportional to n. [3]

The common way for showing the complexity of algorithm is, the symbol of Big O notation (is also known as "Big Oh" notation or asymptotic notation). Big O notation usually describes only upper bound on the growth rate of a function. There are several related notations, using the symbols Ω, ω, and Θ, to describe other kinds of bounds on asymptotic growth rates [22, 23, 24].

Definition 2.1: Big O is a theoretical measure of the execution of an algorithm, usually the time or memory utilization, given the problem size n, which is usually the number of items being processed. Informally, saying some equation $f(n) = O(g(n))$ means it is less than some constant multiple of $g(n)$ (see Figure 2.1). The notation is read, "f of n is big oh of g of n" [24, 25].

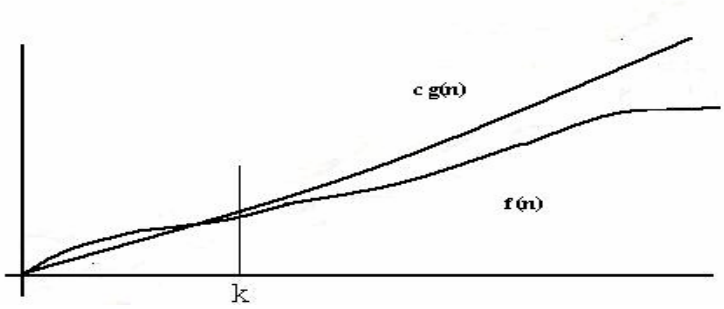

Figure 2.1: Complexity definition graph

There are three types of complexity (for more detail, see [3]):

- Best-case complexity
- Average-case complexity
- Worst-case complexity

As can be seen in Section 2.3 the running time of schoolbook multiplication algorithm or its complexity is $O(n^2)$. In this example, the running time is measured in the worst case. Typically, in much of complexity theory, we measure the maximum running time over all inputs of length n.

Another important point in the study of algorithm complexity is the behavior of the algorithm for very large inputs. Differences between two algorithms with complexity of, $2n^3$ and n^3, or n^2 and $100n^2$, are irrelevant by a many fold increase in the speed of computers. On the other hand, for large enough n, slowing growing terms in a function will be affected by faster terms (like the $5n$ term in comparison with n^2 in the $(n^2 + 5n)$) [17].

2.3 Schoolbook Multiplication

The integer multiplication operation lies at the many cryptographic algorithms [3]. Naturally, remarkable effort went into developing efficient multiplication algorithms. The simplest of such algorithms is the schoolbook multiplication.

In positional numeral system [11], the natural way of multiplying numbers known as schoolbook multiplication is multiplying the multiplicand by each digit of the multiplier and then add up all the properly shifted results. It requires the multiplication table for single digits. Humans do this algorithm usually in base 10 on paper, by writing down all the products and then add them together. Computers usually use a very similar algorithm in base 2 and will sum the products as soon as each one is computed.

Following example uses symbolic schoolbook multiplication to multiply $A = (a_3 a_2 a_1 a_0)_b$ (multiplicand) by $C = (c_2 c_1 c_0)_b$ (multiplier) in arbitrary base b.

$$
\begin{array}{rccccccc}
 & & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & \times & & c_2 & c_1 & c_0 \\
\hline
 & & & a_3c_0 & a_2c_0 & a_1c_0 & a_0c_0 & & \\
+ & & a_3c_1 & a_2c_1 & a_1c_1 & a_0c_1 & & & \\
+ & a_3c_2 & a_2c_2 & a_1c_2 & a_0c_2 & & & & \\
\hline
s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 & &
\end{array}
$$

Figure 2.2: Schoolbook multiplication example

As can be seen from the above example, schoolbook multiplication method can be stated as follows:

Multi precision schoolbook Multiplication Algorithm

Input: positive integers $u = (u_{m-1}u_{m\ 2} \ldots u_1u_0)_b$
and $v = (v_{n-1}v_{n\ 2} \ldots v_1v_0)_b$
Output: The integer product $t=u.v$
For $i=0$ to $m+n-1$ do $t_{i=}\ 0;$
End For
$C_{=}\ 0;$
For $i=0$ to $m-1$ do
For $j=0$ to $n-1$ do
$(cs)b_{=}t_{i+j}+u_i.v_j+c;$
$T_{i+j=}\ s;$
End For
$T_{i+n+1=}\ c;$
End For
Return (t)

Figure 2.3: Multi precision schoolbook multiplication algorithm

The algorithm proceeds in a *row-wise* manner. That is, it takes one digit of one operand and multiplies it with the digits of the other operand. Shifting and accumulating the product is the next step for computation. The subscript b indicates that the digits are represented in radix-b. As can be easily seen the number of digit multiplications performed in the overall execution of the algorithm is $m \times n$.

2.3.1 Time Complexity of the Schoolbook Multiplication Algorithm

Based on Definition 2.1, time complexity is a theoretical measure of the execution time that is needed for an algorithm to run. The schoolbook multiplication algorithm has $O(n^2)$ complexity [11], meaning that the amount of time needed to multiply two n-digit numbers needs about n^2 operations. This means for large values of n the computation time basically explodes (i.e. asymptotic).

2.3.2 Space Complexity of the Schoolbook Multiplication Algorithm

According to the Definition 2.1, space complexity is a theoretical measure of the memory usage for an algorithm. Let n be the total number of bits in the two input numbers. Schoolbook multiplication has the advantage that it can easily be formulated as a *log* space algorithm; that is, an algorithm that only needs working space proportional to the logarithm of the number of digits in the input $O(\log n)$ [11]. In Schoolbook multiplication (see Figure 2.2) the process that is needed is only adding the columns right-to-left and keeping the carry of each column. The maximum number of total sum can never exceed 2n and the carry is at most n. Thus both of these values can be stored in O ($log\ n$) bits.

2.4 Karatsuba Multiplication Algorithm

As we mentioned before, in some applications such as computer algebra systems and big number libraries, there is a need to multiply numbers in the range of several thousand digits. In these cases schoolbook multiplication will be too slow.

Karatsuba's algorithm is an efficient procedure for multiplying two large numbers or two polynomials. It was introduced by Anatolii Alexeevitch Karatsuba in 1960 and published in 1962 [26, 27]. This algorithm is a notable example of the divide and conquer [28, 29,30] paradigm, specifically of binary splitting [31, 32]. Hence before entering to the main discussion about Karatsuba algorithms and its different variants, we will review briefly, the structure and specifications of the processing paradigm, which is the divide and conquer algorithm.

2.4.1 Divide and Conquer Algorithm

Divide and conquer algorithm is a method for solving a problem by dividing it into two or more smaller sub-problems. Each of these smaller sub-problems is recursively solved, and the solutions to the sub-problems are then combined to give a solution to the original problem [28].

The name "divide and conquer" is occasionally applied also to algorithms that reduce each problem to only one sub-problem, the binary search algorithm for finding a record in a sorted list [28] is an example for this application. Every algorithm that uses recursion or loops can be expressed as a "divide and conquer algorithm". Hence, some authors use the name "decrease and conquer" instead [29]. This technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quick sort, merge sort), multiplying large numbers (e.g. Karatsuba), and computing the discrete Fourier transform (DFT) [28].

Improvement in the asymptotic cost of the solution is one of the important advantages of using the divide and conquer algorithm. For example, if the base cases have constant-bounded size, the work of splitting the problem and combining the partial solutions is proportional to the problem's size n, and there are a bounded number p of sub-problems of size n/p at each stage, then the cost of the divide and conquer algorithm will be O(n log n) [28].

For execution in multi-processor machines, especially shared-memory systems, because distinct sub-problems can be executed on different processors, the divide and conquer

algorithms performance will improve. Therefore parallelism techniques can be easily applied for the algorithms, in which constructed based on divide and conquer algorithm.

2.4.2 Karatsuba Multiplication Algorithm Construction and Specifications

We stated previously that the Karatsuba algorithm is an efficient method for large numbers multiplication. It reduces the multiplication of two n-digit numbers to $n^{\log_2 3} = n^{1.585}$ (i.e. asymptotic) single-digit multiplications and saves coefficient multiplications at the cost of extra additions compared to the School-book multiplication algorithm. The lower exponent of Karatsuba algorithm shows; it is faster than the School-book algorithm, which requires n^2 single-digit products. The Karatsuba algorithm is a notable example of the divide and conquer algorithm. The basic Karatsuba algorithm is performed as follows [11]:

Consider two degree-1 polynomials $A(x)$ and $B(x)$ with $n = 2$ coefficients.

$$A(x) = a_1x + a_0$$
$$B(x) = b_1x + b_0.$$

Let D_0, D_1, $D_{0,1}$ be auxiliary variables with

$$D_0 = a_0b_0$$
$$D_1 = a_1b_1$$
$$D_{0,1} = (a_1 + a_0)(b_1 + b_0)$$

Then the polynomial $C(x) = A(x) \times B(x)$ can be calculated in the following way:

$$C(x) = D_1x^2 + (D_{0,1} - D_0 - D_1)x + D_0.$$

Example: assume we want to compute the product of 1234 and 5678.

$12\ 34 = 12 \times 10^2 + 34$

$56\ 78 = 56 \times 10^2 + 78$

$D_0 = 34 \times 78 = 2652$

$D_1 = 12 \times 56 = 672$

$D_{0,1} = (12 + 34)(56 + 78) - D_1 - D_0 = 46 \times 134 - 672 - 2652 = 2840$

$C(x) = D_1 \times 10^{2\times2} + D_{0,1} \times 10^2 + D_0 = 672 \times 10000 + 2840 \times 100 + 2652 = 7006652$

This method requires three multiplications and four additions. The School-book method requires n^2 multiplications and $(n-1)^2$ additions, i.e., four multiplications and one addition for $n=2$. Clearly, the Karatsuba algorithm can also be used to multiply integer numbers and can be generalized for polynomials of arbitrary degree. Let the number of coefficients be even. To apply the algorithm both polynomials are split into a lower and an upper half:

$$A(x) = A_u(x)\, x^{n/2} + A_l(x)$$

$$B(x) = B_u(x)\, x^{n/2} + B_l(x).$$

These halves are used as before. The polynomials A_u, A_l, B_u, and B_l are split again in half in the next iteration step. Since every step exactly halves the number of coefficients, the algorithm terminates after $t = \log_2 n$ steps. Figure 2.4 shows this algorithm, where n is the number of coefficients of $A(x)$ and $B(x)$.

Recursive Karatsuba Algorithm, C = KA(A, B)

INPUT: Polynomials $A(x)$ and $B(x)$

OUTPUT: $C(x) = A(x)\,B(x)$

$N = max(degree(A), degree(B)) + 1$

IF $n = 1$ Return $A \cdot B$

$A(x) = A_u(x)\,x^{n/2} + A_l(x)$

$B(x) = B_u(x)\,x^{n/2} + B_l(x)$

$D0 = KA(A_l, B_l)$

$D1 = KA(A_u, B_u)$

$D0,1 = KA(A_l + A_u, B_l + B_u)$

Return $D_1 x^n + (D_{0,1} - D_0 - D_1)x^{n/2} + D_0$

Figure 2.4: Recursive Karatsuba algorithm

Let N_{MUL} and N_{ADD} be the number of multiplications and additions, respectively. Then the complexity to multiply two polynomials with $n = 2^i$ coefficients are as follows [33]:

$$N_{MUL} = n^{\log_2 3} = n^{1.585}$$
$$N_{ADD} \leqslant 6n^{\log_2 3} - 8n + 2$$

The recursive Karatsuba algorithm versions are more efficient than the one-iteration Karatsuba algorithm. For example, instead of using the one-iteration Karatsuba algorithm for $n = 31$ coefficients, Karatsuba algorithm can use the recursive Karatsuba algorithm and split the 31- coefficient polynomial into two polynomials of 15 and 16 coefficients, respectively. Alternatively one could split the 31-coefficient polynomial into three parts of 10, 10, and 11 coefficients, respectively. In the next recursion step, the polynomials are again split in two or three parts, and so on. However, a number of intermediate results have to be stored due to the recursive nature. This might reduce the efficiency of the recursive Karatsuba algorithm variants for small size polynomials.

When the number of coefficients is unknown at implementation time, the simple recursive Karatsuba algorithm is the most efficient way to implement the Karatsuba algorithm [33]. Using the one-iteration Karatsuba algorithm for special cases, $n = 2$ and $n = 3$ as well as for $n = 9$ that splits the operands into three parts recursively yield efficient running times.

Using the dummy coefficients is another technique for improving the performance. For example, to multiply two polynomials of $n = 15$ coefficients it might be useful to append a zero coefficient and use a recursive Karatsuba algorithm for k=16 coefficients. Some operations can be saved whenever the leading zero coefficient is involved. However, this is only little improvement. For some cases the Karatsuba algorithm is always faster than the School-book method, it needs less multiplications and additions. However, there are also cases where it is more efficient to use a combination of Karatsuba algorithm and the School-book method [33].

In most cases, if n is not very small, the squaring Karatsuba algorithm outperforms the ordinary squaring method [33]. Further information about the Karatsuba and similar algorithms can be found in [34, 35, 36].

2.4.3 Related Work to Karatsuba Algorithm

Work on combination multiplication algorithms with Karatsuba algorithm is another technique that researchers have been engaged on. J. V. Gathen, et al. [37] worked on the combination of the School-book and the Karatsuba methods, and reported