B503 – Project Report

Arvind Dwarakanath

Introduction

As a part of B503 course, the project involves multiplication of two large integers and to multiply them in an optimal way.

The initial code was executed using the Algorithm 1.8 that considers the product of the words one by one using the two for loops. The algorithm is improved on by using the Karatsuba Algorithm that theoretically gives a better performance.

The algorithms used by this project are Algorithm 1.8, Algorithm 5.2 and Algorithm 1.5 described in reference book of Prof Paul Purdom Jr.

Logic for Product32

Consider 2 numbers 'a' and 'b'. We split the numbers 32 bit wise into many words. Consider that the total number of bits is <64 but > 32. Therefore we get the following:-

a(high) a(low)

b(high) b(low)

The value will be truncated as per low number of bits and high number of bits; so displaying the numbers would yield a decimal form of the two words. So we have to consider them for our calculation.

Imagine we have a number that is represented in decimal format as per the following.

000.....1101 and 000000...11 (32 bits each; 2 words)

So the value is 13 and 3 respectively. Hence we need to find the correct base by which the multiplication must take place as per Algorithm 1.8 to find the total value.

So calculating

1*2^35 + 1*2^34 + 1*2^32 + 1*2^1 + 1*2^0 = 13b+3

2^32 (8 + 4 +1) +3 = 13b +3

So b = 2^32 which is the base.

So if we multiply the above word by itself, the representation will be:-

(13*2^32 + 3)(13*2^32 +3)

13*13*2^64 + (13 *3 *2) 2^32 +9 to give the product.

Basically a form of (ah*bh* 2^64) + (ah*bl + al*bh)*2^32 + al*bl.

By extension of Algorithm of 5.2, each multiplication will **recursively** call the product function till the number is able to fit into the 32-bit slot. Also whenever we encounter the 2^2n multiplication, we must accordingly **shift the product to a higher word** rather than try to multiply the whole number.

Analysis of the 1.8 algorithm

The 1.8 algorithm is the basic algorithm that multiplies two digits word by word and adds the individual products to obtain the final result. This utilizes the fact that any word is a combination of the powers of the base value.

The algorithm is non recursive in nature and the code accepts the input as a list of 32 bit words and multiplies all the combination of the first variable int_a[i] and int_b[i]. In order to accomplish this, the code uses 2 'for' loops and then the multiplication takes place within the 'for' loops and the result is obtained. In the code, this is the multiply32 function.

Assume the size of the two inputs is 'n' bits. For simplicity sake, assume they are same (though the code is perfectly fine with unequal bits). So the two 'for' loops will run 'n' times each. Therefore the number of steps required will be $\mathbf{n^2}$.

Analysis of the 5.2 algorithm

The 5.2 algorithm shows a marked improvement on the modified 1.8 algorithm. In the modified 1.8 algorithm, we basically have two major for loops that will run wa and wb number of times; where wa and wb stand for the number of words in a and b respectively. So as the number of words increases, the steps within the two for loops will increase with wa*wb as a major factor. If we consider the number of bits then na*nb. If na = nb then n^2 would still dominate.

If we consider 5.2 analysis, then we consider the logic mentioned in the second section. The code will consider the larger number and divide that number by 2. The same split is utilized on both the numbers. Therefore we see that the final result after the first iteration would be.

$$\mathbf{u1v1*2^n + [t1t2-(u1v1+u2v2)]*2^{(n/2)} + u2v2}$$

Note there will be cases where the number of bits of one variable is less than half of the other variable. In such cases, the code simply runs the 1.8 Algorithm because the empirical performance of the 5.2 algorithm is very poor and quite erroneous.

So we have three major multiplications here U1V1, U2V2 and T1T2. These three multiplications are n/2 bits long each. The multiplication will be recursive each. The additions are n/2 each.

For simplicity sake, consider the number of bits to be n. Assume that $n = 2^k$.

So using the recursion equation, we get the following:-

T(n) = 3*T(n/2)+cn with n being the number of bits. T(n) is the amount of time required to compute the final product.

3*T(n/2) term is used because there are three major multiplications to be performed.

If the equation will be:-

T(n) <= 3T(n/2) + cn

$T(n) <= 3(3T(n/4) + c(n/2)) + cn$

$T(n) <= 3(3(3T(n/8) + c(n/4)) + c(n/2) + cn$

.

.

.

.

$T(n) <= 3^kT(n/2^k) + cn\ (3^{k-1}/2^{k-1} + ...3/2 + 1)$

$T(n) <= 3^kT(n/2^k) + cn\ [((3/2)^{k-1}-1) / 1/2]$

$T(n) <= 3^kT(1) + 2cn(3^k-2^k)$

OR

$T(n) <= 3c3^k$ (Because T(1) is considered 1)

By definition $n = 2^k$ So $k = \lg n$

So $T(n) <= 3c3^{\lg n}$

OR $T(n) <= 3cn^{\lg 3}$

So we get $O(n^{\lg 3})$ as the time complexity.

If we consider the number of words rather than the number of bits, then for equal words

$T_{(n)} = 3*T_{(n/2+1)} + 3_n$; the 3n following is the normal multiplication

Assuming the value of $n = n^2 + n$

So

$n^2 + n = 3[(n/2 + 1)^2 + n/2 + 1] + 3n$

Solving the above quadratic equation, we get

n is approx equal to 27.

So

$n^2 + n < 3[(n/2 + 1)^2 + n/2 + 1] + 3n$ $(0 < n < 27)$

Therefore the minimum number of words that can be done optimally using the Karatsuba is **27 words** (atleast theoretically speaking).

It would further enhance the code if we integrate Algorithm 1.8 into 5.2. When Karatsuba cuts-off, Algo 1.8 will commence with the lower level multiplication.

About the Code

The code is broken into many functions; each function designed to carry out certain operations. Listing the functions as below:-

Product32

product32 is the primary function that takes the input variables and breaks them down into a sequence of 32 bit words. This function invokes the Karatsuba function and passes to it all the required values and the return function.

Karatsuba32

karatuba32 function is the main portion of the entire algorithm. This is recursive by nature and calls itself recursively for all the major multiplications of U1V1, U2V2 and T1T2 as well. We need a base case so that Karatsuba can cease its recursion. The check is positioned at the start of the code. If the word size is below or equal to a word limit then the Karatsuba will cease its recursion and invoke multiply32(). Additionally, the condition that if one of the word sizes is less than or equal to half of the other word size then multiply32 will be invoked. This is keeping only timing consideration in mind.

Add32

Add32 function adds two 32 bit numbers and to provide the output. This is based on the Algorithm 1.4 provided in the reference notes.

Subtract32

Subtract32 subtracts the two 32 bit numbers and returns the value generated.

Multiply32

Executes the Algorithm 1.8 to multiply 32 bit numbers. This is the base algorithm on which Algorithm is built to improve the timings. It is effective but slow without Algorithm 5.2.

Performance of the Code and Empirical Observations

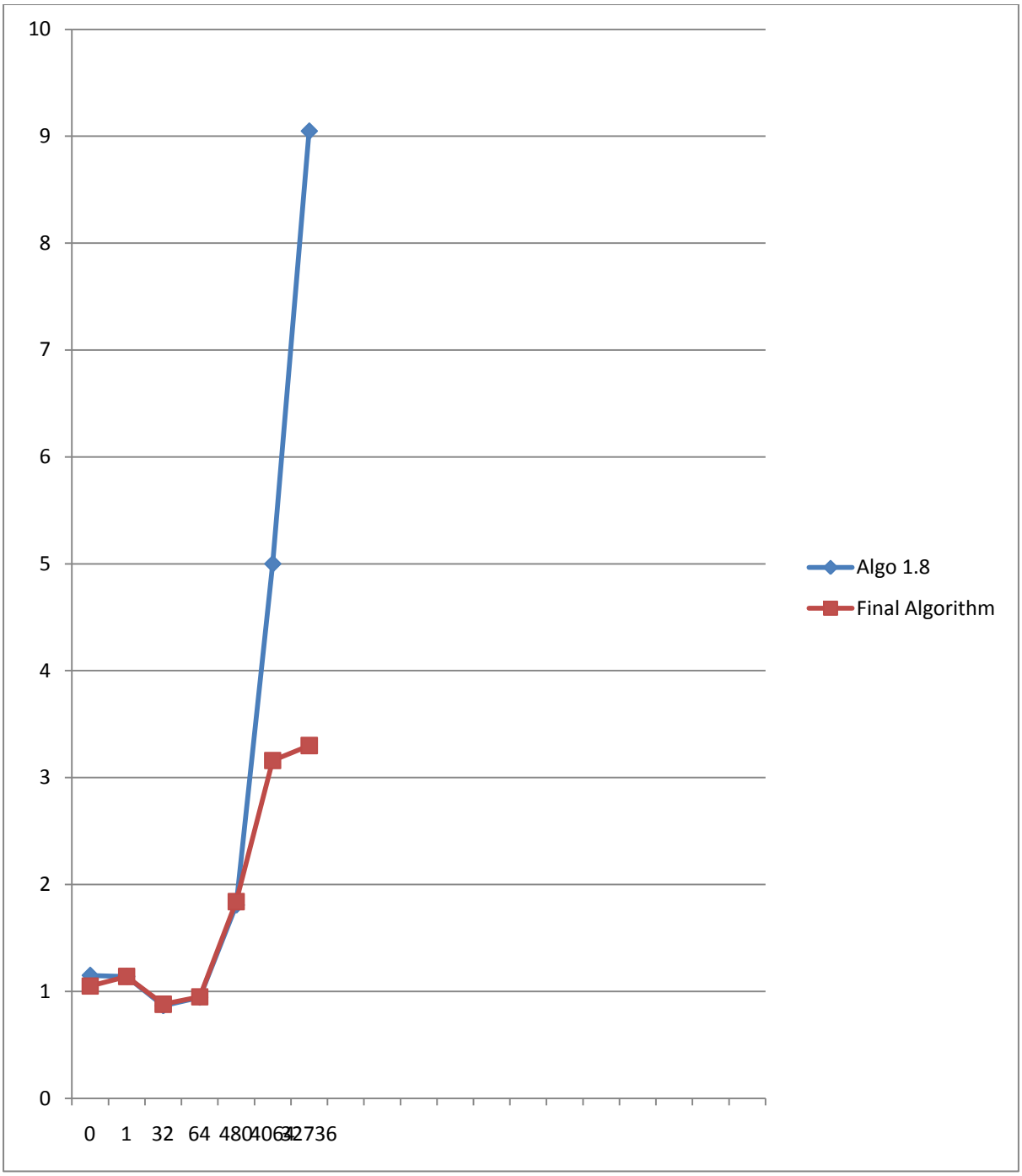
We compare the performance of the code based on the values that were obtained. The performance parameter considered is the ratio of code time to GMP time.

Theoretically, the cut-off for Karatsuba is **27** words. But on many experiments, the best cut-off was found to be around **35** words. This may be due to the time spent for the operations like calloc that have been used extensively in the code.

Note that the values obtained are for the test_32.sh test cases.

Number of Bits	Algorithm 1.8 Code to GMP Timings for 100 multiplications	Final Code to GMP Time ratios (Algorithm 1.8 backed by Algorithm 5.2) for 100 multiplications
1 1	1.14	1.14
0 0	1.15	1.05
0 1	1.408	1.112
1 0	1.12	1.15
32 32	0.866	0.88
64 64	0.944	0.95
480 480	1.807	1.84
4064 4064	5.00	3.16
32736 32736	9.0841	3.30
32 64	0.89	0.94
32 480	1.34	0.901
32 4064	1.11	1.20
32 32736	2.58	2.35
480 4064	3.14	3.09
480 32736	3.406	3.42
4064 32736	4.25	4.27

Graphs



The y-axis is the code/GMP ratio and the x-axis is the number of maximum bits multiplication. For unequal bits, see the performance table.

References

Project basic understanding is in collaboration with Anindya Lahiri and Vaibhav Nachankar.

Special thanks to Vaibhav Shankar for his help in clearing concepts.

References materials: Analysis of Algorithms by Paul W. Purdom Jr and Cynthia A. Brown – Algorithm 1.4 and 1.8 and Algorithm 5.2.

Also, inputs from A.I. Haipeng Zhang