Multiplying numbers efficiently

MG Ferreira

August 2014

Abstract

We propose a method for multiplying numbers that requires no recursion and that outperforms the schoolbook method by a wide margin. This method performs extremely well for numbers that are too small to benefit from Karatsuba's method. Karatsuba's method can also be used to speed up this method for larger numbers.

1 Background

Effecient multiplication algorithms are of great interest to mathematicians, who utilise it to calculate millions of digits of π for example. Less interesting but perhaps more practical uses can be found amongst hardware and software vendors and cryptographers.

One well known and often used algorithm is that of Karatsuba, discovered in 1960, that reduces multiplication of two n digit numbers to $n^{log_2 3}$ multiplications when n is a power of 2, an ideal situation for this algorithm. It does so by recursively splitting the digits of say a in half, say $a = a_0 + a_1 S$, with S an appropriate scaling factor, so that ab becomes

$$ab = (a_0 + a_1 S)(b_0 + b_1 S) = a_0 b_0 + (a_0 b_1 + a_1 b_0)S + a_1 b_1 S^2. \quad (1)$$

Karatsuba's contribution is the discovery that rewriting the middle term as

$$a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1 \quad (2)$$

reduces the number of multiplications required to calculate ab by 1. Henceforth we'll refer to this as *Karatsuba's trick*.

The overhead that Karatsuba's trick introduces is not trivial and in practice performance only improves for fairly large n. The GMP (Gnu multiple precision) library, for example, applies Karatsuba only when the number of bits exceeds 800 - a rather large number. Numbers made up of 800 or fewer bits are still multiplied, presumably faster, using the *schoolbook* method at n^2 multiplications for an n digit number.

The US currently requires 192 or 256 bits to encrypt sensitive data and it seems an algorithm that outperforms the schoolbook method in this range will be a welcome addition.

Table 1: ab

a_0b_0	a_1b_1	a_2b_2	...	$a_{n-1}b_{n-1}$
	$a_0b_1 + a_1b_0$	$a_1b_2 + a_2b_1$	...	$a_{n-2}b_{n-1} + a_{n-1}b_{n-2}$
		...		
		$a_0b_{n-2} + a_1b_{n-1}$	$a_1b_{n-1} + a_0b_{n-2}$	
		$a_0b_n + a_1b_{n-1}$		

2 Building pyramids

At the time of Karatsuba's discovery, Kolmogorov suggested that, since multiplication dates back to at least Ancient Egypt, it is optimal and any multiplication algorithm would require steps in the order of n^2. To improve multiplication we thus start by constructing some pyramids, albeit upside down ones. Say we have two arbitrary large numbers, a and b, with n digits each. Thus

$$a = a_0 + a_1S + a_2S^2 + ... + a_{n-1}S^{n-1} \tag{3}$$

and the same for b.

To calculate the product ab, we need to multiply each digit of a with each digit of b and scale and add the result appropriately to obtain the final product. One way of representing this process is given in table 1.

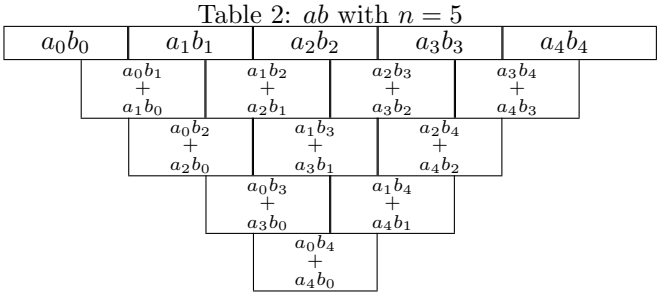
Note that each cell or block in the pyramid represents the sum of two two digit numbers, each with a value of magnitude S^2. If a number is too *big* to fit into a single cell, the overflow or *carry* is carried over into the next cell and so forth. Since ab will have $2n$ digits at most, we do not need to worry about a final carry increasing the size of the pyramid's base. Once the pyramid has been constructed, the cells are added (vertically) to arrive at the final answer, with any overflow likewise carried over to the next cell. Note that, although each cell represents two digits, the horizontal addition proceeds on a single digit basis.

Note that *digits* here have a rather loose definition. Thus a_0 could be a single decimal (or binary) digit, it could be a byte or a word or even 10,000 decimal digits. In what follows, we'll continue to use the word digit but, with an appropriate scaling factor S, it could have any base or size. This makes it possible to apply Karatsuba's trick in each cell in the pyramid, without having to resort to recursion.

If we have a threshold t, so that numbers larger than S^t will benefit from Karatsuba's trick, we use this to define the digits of a and b and the scheme above to apply it without any recursion.

We also align a and b's least significant digit at the least significant digit of the pyramid. The result is then scaled after the multiplication to start at $l_a + l_b$, where l_a is the position or scale of a's least significant digit. Thus if we multiply 12 000 and 0.00034 for example, we use the numbers 12 and 34 to construct the pyramid and we have $l_a = 3$ and $l_b = -4$

Table 2: ab with $n = 5$



so that we align the result at position -1 for decimal numbers with scale $S = 10$.

On a computer the underlying implementation of arbitrary precision numbers will determine the appropriate scale to use, but it is easy to envisage each digit being a *byte* and each cell in the pyramid being a *word* and calculations being performed using *double words* in order to allow for overflow.

3 Traversing the pyramid

For this section we will make use of a concrete example of calculating ab where a and b have an equal amount of digits, say 5 each, and where the result thus requires 10 digits (or 5 words) to store. Note that we can apply the same algorithm to numbers with unequal digit lengths, albeit resulting in unbalanced, ragged edged pyramids. The height of such a pyramid is determined by the number of digits in the number with most digits.

As mentioned before, one of the biggest advantages in traversing this pyramid, relative to Karatsuba, is that recursion is not equired. A number of alternatives also present themselves - we could for example traverse the pyramid from left to right, writing out the result sequentially as we do so, in a way that requires no interim storage. Thus we start at the least significant edge by calculating a_0b_0 and store the lower digit as the first digit into the answer's storage space. We use the higher digit as carry and proceed to the next digit, where we find all cells that align with it on the least significant side (only $a_0b_1 + a_1b_0$ in this case) and calculate and add them to the carry. The lower digit is stored as the result's next digit and the higher digits used as carry as we proceed to the next digit.

Note that although cells in the pyramid contain double digit results, we store the result as a single digit at a time from left to right, or from least to most significant digit. We also require multiple digits to accommodate the interim calculations, with the actual number of digits depending on the maximum possible carry, which in turn depends on the height of the pyramid. In practise we may define a digit as a byte and use double words (four bytes) for interim calculations for example.

This method is highly efficient and requires the minimum storage. It

Table 3: $12\,345 \cdot 98\,765 = 1\,219\,253\,925$

1	2	3	4	5
9	8	7	6	5
$1\cdot9=9$	$2\cdot8=16$	$3\cdot7=21$	$4\cdot6=24$	$5\cdot5=25$
	$1\cdot8+2\cdot9=26$	$2\cdot7+3\cdot8=38$	$3\cdot6+4\cdot7=46$	$4\cdot5+5\cdot6=50$
		$1\cdot7+3\cdot9=34$	$2\cdot6+4\cdot8=44$	$3\cdot5+5\cdot7=50$
			$1\cdot6+4\cdot9=42$	$2\cdot5+5\cdot8=50$
				$1\cdot5+5\cdot9=50$
carry 1 $= 1$	carry 1+9 $= 2$ carry 1	carry 5+26 $= 1$ carry 3	carry 9+16+34 $= 9$ carry 5	carry 12+38+42 $= 2$ carry 9	carry 10+21+44+50 $= 5$ carry 12	carry 7+46+50 $= 3$ carry 10	carry 5+24+50 $= 9$ carry 7	carry 2+50 $= 2$ carry 5	^{25} $= 5$ carry 2

is particularly well suited if we need to produce the result sequentially to some storage device as we calculate the result in a single step from left to right.

Another way in which we could traverse the pyramid requires double the storage space, but allows us to apply Karatsuba's trick. We start by separately calculating $a_i b_i$ and storing it in $2n$ digit additional storage space. These $a_i b_i$ values are then used to apply Karatsuba's trick to speed up the calculations in each cell, albeit only if a digit is large enough to realise a performance improvement from Karatsuba's trick.

Yet another alternative is to traverse the pyramid in diagonals, so that we can cache the previously retrieved values of a_i and b_i. This is possible as, along diagonals, the product in each cell can be written as $a_i b_j + a_j b_i$ with either i or j fixed along the diagonal, depending on whether the diagonal is to the left or to the right.

Finally we can simply traverse the pyramid row by row, updating the result with carry as we go along.

4 Example

The construction of the pyramid has until now been computer friendly, with the least sigificant digit on the left. Since humans, at least in English speaking countries, usually write numbers with the least significant digit on the right, we will construct yet another example, this time a human friendly, right to left one, to calculate the product $12345 \cdot 98765$ using this technique. This time, we also start by writing out $a = 12345$ and $b = 98765$ on top of the pyramid, and in such a way that each digit occupies two digits' worth of space of the result. This makes it easy to fill in the cells and the result is given in 3.

Note that for each total we add the carry to the result of cells that align on the right or least significant digit side with the result's digit. Also note that constructing the pyramid as in 3, with a and b written down above the base, makes it very clear which digits of a and b contribute to which digits of the product. This aspect adds a certain scholastic element to it that makes it attractive as a method of teaching children to do

multiplication.

This method is similar to the *lattice method* of multiplication, but, since numbers are arranged in order of significance, rather than in a matrix, this approach is more intuitive.

5 Performance

We now have four ways of traversing the pyramid.

- First calculating $a_i b_i$ in separate storage and then proceeding on a row by row basis through the pyramid, updating the result as we go along.
  - We can also use this to apply Karatsuba's trick.
- Diagonally, caching previous digits of a and b and continuously updating the result.
- Left to right, where we calculate and add up all cells that align with a given digit in the result and where we write the answer out sequentially.

These four techniques were implemented and used to multiply random numbers containing from 1 to 250 decimal digits a thousand times. The schoolbook method was used to verify the answers and also, for the same numbers, to establish a reference calculation time. Temporary storage was allocated before any calculations and reused so that no method was penalised for memory allocation or freeing overhead. The calculations were done using both 32 and 64 bit architectures.

The performance of the four alternatives, on a 64 bit architecture, is discussed below.

- The row by row alternative fared worst but still needed only 6.4% the time of the schoolbook algorithm to calculate the product of 140 digit numbers.
  - Karatsuba's trick was applied but caused performance to deteriorate. No attempt was made to increase the digit size in order to realise a Karatsuba related performance improvement, although it is suspected that for sufficiently large digits this will improve performance notably.
- The diagonal method was roughly twice as fast as the row by row alternative and registered a best performance, relative to the schoolbook method, of 3.1% for numbers with around 230 digits each.
- The left to right method was the fastest, albeit not much faster than the diagonal method. Its best time, relative to the schoolbook algorith, was 3% at around 225 digit numbers.

The performance of all four alternatives are much better than the schoolbook method. The schoolbook method performs better for small numbers, but even for four digit numbers the performance of any of the alternatives already surpasses that of the schoolbook method. The outperformance is also notably large. To calculate products in less than 5% of the time it takes for the schoolbook method is by no means a trivial

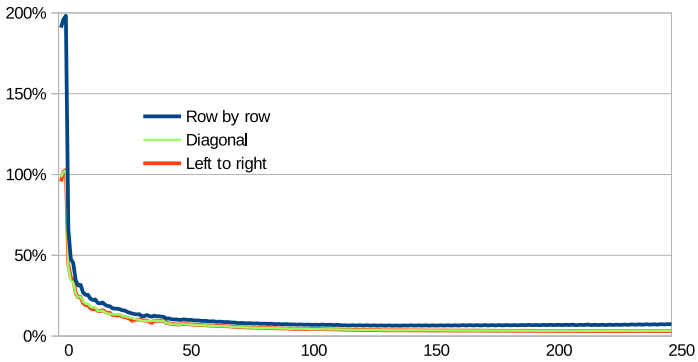


Figure 1: Performance relative to schoolbook

improvement. The performance, relative to the schoolbook method, does appear to deteriorate slowly as the numbers continue to increase, but even at 150 000 digits the diagonal and left to right alternatives still perform at less than 15% than that of the schoolbook algorithm.

The performance on a 32 bit architecture was similar albeit somewhat worse than that of the 64 bit architecture.

This huge performance gain is hard to explain at first, given that this approach clearly uses just as many multiplications as the schoolbook method. If we multiply two n digit numbers, the schoolbook method will multiply say the first with each of the n digits of the second number, and so forth, for a total of n^2 multiplications. These numbers, however, are added to a running total after each multiplication, yielding roughly n single digit additions for each product or, if we also include the carry, roughly $2n$ additions. Since this happens for all n digits, the schoolbook method uses around $2n^2$ additions in total to compute its answer. [1]

In the pyramid the number of cells that have additions in them is $\frac{n^2-n}{2}$ which is all cells excluding the base. Since we need to add the base as well as all of the cells' results, including any carry, to get to the final result, the total number of additions becomes n^2 or half that of the schoolbook method.

The schoolbook method's running sum is an arbitrary precision number while in the pyramid we can use machine precision in each cell and for each digit, so that the fewer pyramid additions are also executed faster than the schoolbook additions.

[1] The $2n^2$ additions is roughly correct - while we could save n additions by initialising the running sum with the first product, we also need to take into account the possible carry at the n-th digit of each product.

6 Other uses

This approach can also help when factoring numbers. Note that the least significant digit is completely determined by the product of the two least significant digits of the two numbers. The next digit is determined by the most significant digit of this product and by the least significant digit of $a_0b_1 + a_1b_0$ and so on. The pyramid makes it very clear which digits of the two numbers are involved in determining the outcome of a certain digit in the product.

7 Conclusion

The method for multiplication described in this paper outperforms the schoolbook method by a wide margin. This outperformance is registered in the area where Karatsuba's method still has too much overhead to be successfully applied, but which is relevant to for example cryptographers given current key lengths. While this approach may not be novel, the way in which it is presented here makes it easy to index the digits of the numbers being multiplied in a very efficient way.