Karatsuba Algorithm

André Weimerskirch
escrypt Inc.

Related concepts and keywords

polynomial multiplication, schoolbook multiplication method, finite field multiplication

Definition

A method for multiplying two polynomials that saves coefficient multiplications at the cost of extra additions compared to the schoolbook multiplication method.

Theory

The Karatsuba Algorithm (KA) for multiplying two polynomials was introduced in 1962 [3]. It saves coefficient multiplications at the cost of extra additions compared to the schoolbook or ordinary multiplication method. The basic KA is performed as follows. Consider two degree-1 polynomials $A(x)$ and $B(x)$ with $n = 2$ coefficients.

$$A(x) = a_1 x + a_0$$

$$B(x) = b_1 x + b_0$$

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

$$D_0 = a_0 b_0$$

$$D_1 = a_1 b_1$$

$$D_{0,1} = (a_0 + a_1)(b_0 + b_1)$$

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

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

This method requires three multiplications and four additions. The schoolbook method requires n^2 multiplications and $(n-1)^2$ additions, i.e., four multiplications and one addition. Clearly, the KA can also be used to multiply integer numbers.

The KA can be generalized for polynomials of arbitrary degree [6]. The following algorithm describes a method to multiply two arbitrary polynomials with n coefficients using the *one-iteration KA*.

Algorithm 1 ***Generalized One-Iteration KA***
Consider two degree-d polynomials with $n = d + 1$ *coefficients*

$$A(x) = \sum_{i=0}^{d} a_i x^i, \;\; B(x) = \sum_{i=0}^{d} b_i x^i$$

Compute for each $i = 0, \ldots, n-1$

$$D_i := a_i b_i$$

Calculate for each $i = 1, \ldots, 2n-3$ *and for all* s *and* t *with* $s + t = i$ *and* $t > s \geq 0$

$$D_{s,t} := (a_s + a_t)\ (b_s + b_t)$$

Then $C(x) = A(x)\ B(x) = \sum_{i=0}^{2n-2} c_i x^i$ *can be computed as*

$$\begin{aligned} c_0 &= D_0 \\ c_{2n-2} &= D_{n-1} \\ c_i &= \begin{cases} \sum_{s+t=i;t>s\geq 0} D_{s,t} \\ \quad - \sum_{s+t=i;n-1\geq t>s\geq 0} (D_s + D_t) \\ \quad \textit{for odd } i,\ 0 < i < 2n-2 \\ \sum_{s+t=i;t>s\geq 0} D_{s,t} \\ \quad - \sum_{s+t=i;n-1\geq t>s\geq 0} (D_s + D_t) \\ \quad + D_{i/2} \\ \quad \textit{for even } i,\ 0 < i < 2n-2 \end{cases} \end{aligned}$$

The number of auxiliary variables is given as:

$$\#D_i = n$$

$$\#D_{s,t} = n^2/2 - n/2$$

$$\#D = \#D_i + \#D_{s,t} = n^2/2 + n/2$$

The operational complexity is as follows:

$$\#\text{MUL} = n^2/2 + n/2$$

$$\#\text{ADD} = 5/2\ n^2 - 7/2\ n + 1$$

For example, consider the KA for three coefficients. Let $A(x)$ and $B(x)$ be two degree-2 polynomials:

$$A(x) = a_2x^2 + a_1x + a_0$$

$$B(x) = b_2x^2 + b_1x + b_0$$

with auxiliary variables

$$\begin{aligned} D_0 &= a_0b_0 \\ D_1 &= a_1b_1 \\ D_2 &= a_2b_2 \\ D_{0,1} &= (a_0 + a_1)(b_0 + b_1) \\ D_{0,2} &= (a_0 + a_2)(b_0 + b_2) \\ D_{1,2} &= (a_1 + a_2)(b_1 + b_2) \end{aligned}$$

Then $C(x) = A(x)\ B(x)$ is computed by an extended version of the KA using 6 multiplications and 13 additions. The schoolbook method requires in this case 9 multiplications and 4 additions.

$$\begin{aligned} C(x) &= D_2x^4 + (D_{1,2} - D_1 - D_2)\ x^3 \\ &\quad +(D_{0,2} - D_2 - D_0 + D_1)\ x^2 \\ &\quad +(D_{0,1} - D_1 - D_0)\ x + D_0 \end{aligned}$$

The KA can be performed recursively to multiply two polynomials as shown in Algorithm 2. Let the number of coefficients be a power of 2, $n = 2^i$. 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, i.e., as if they were coefficients. The polynomials A_u, A_l, B_u, and B_l are split again in half in the next iteration step. In the final step of the recursion, the polynomials degenerate into single coefficients. Since every step exactly halves the number of coefficients, the algorithm terminates after $t = \log_2 n$ steps. Note that there are overlaps

of the coefficient positions when combining the result that lead to further additions. For example, consider Algorithm 2 where N is the number of coefficients of $A(x)$ and $B(x)$. The polynomial D_0 has $N-1$ coefficients such that the upper $N/2-1$ coefficients of D_0 are added to the lower $N/2-1$ coefficients of $(D_{0,1} - D_0 - D_1)$.

Let #MUL and #ADD be the number of multiplications and additions, respectively, in the underlying coefficient ring. Then the complexity to multiply two polynomials with $n = 2^i$ coefficients is as follows [5]:

$$\#\text{MUL} = n^{\log_2 3}$$

$$\#\text{ADD} \leq 6n^{\log_2 3} - 8n + 2$$

Algorithm 2 ***Recursive KA, C = KA(A, B)***
`INPUT`*: Polynomials A(x) and B(x)*
`OUTPUT`*: C(x) = A(x) B(x)*
$N \leftarrow max(degree(A),\ degree(B)) + 1$
`if` $N = 1$ `return` $A \cdot B$
`Let` $A(x) = A_u(x)\ x^{N/2} + A_l(x)$
`and` $B(x) = B_u(x)\ x^{N/2} + B_l(x)$
$D_0 \leftarrow KA(A_l, B_l)$
$D_1 \leftarrow KA(A_u, B_u)$
$D_{0,1} \leftarrow KA(A_l + A_u, B_l + B_u)$
`return` $D_1 x^N + (D_{0,1} - D_0 - D_1)x^{N/2} + D_0$

The recursive KA can be generalized in various ways. If the number of coefficients n is no power of 2 Algorithm 2 is slightly altered by splitting the polynomials into a lower part of $\lceil N/2 \rceil$ coefficients and and upper part of $\lfloor N/2 \rfloor$ coefficients. This variant is called the *simple recursive KA*. In this case the KA is less efficient than for powers of 2. A lower bound for the number of multiplications is given by $\#\text{MUL}_{low} = n^{\log_2 3}$ whereas an upper bound is $\#\text{MUL}_{up} = 1.39\ n^{\log_2 3}$. When applying the one-iteration KA for two and three coefficients as basis of the recursion, i.e., when applying the KA for two and three coefficients as final recursion step, the upper bound improves to $\#\text{MUL}_{up} = 1.24\ n^{\log_2 3}$. When applying the one-iteration KA for two, three, and nine coefficients as basis of the recursion, the upper bound further improves to $\#\text{MUL}_{up} = 1.20\ n^{\log_2 3}$.

In the same manner bounds for the number of additions are obtained $\#\text{ADD}_{low} = 6n^{\log_2 3} - 8n + 2$ and $\#\text{ADD}_{up} = 7.30\ n^{\log_2 3}$ which improves

for a basis of two and three coefficients to $\#\text{ADD}_{up} = 6.85\ n^{\log_2 3}$. For a basis of two, three, and nine coefficients, it further improves to $\#\text{ADD}_{up} = 6.74\ n^{\log_2 3}$.

When using the recursive KA for $n = p^j$ coefficients, i.e., applying the KA for p coefficients for j recursion levels, the number of multiplications is given as $\#\text{MUL} = n^{\log_p(1/2p^2+1/2p)}$. For $p = 2$ the number of multiplications is given by $\#\text{MUL} = n^{\log_2 3}$ whereas for large p this converges to $(1/2)^j\ n^2$.

Now two polynomials $A(x)$ and $B(x)$ with $n = \prod_{i=1}^{j} n_i$ coefficients are considered that are multiplied with a variant of the KA, called *generic recursive KA*. First the term $A(x) = \sum_{s=0}^{n_j-1} A_s x^{s\cdot\prod_{i=1}^{j-1} n_i}$ is written as polynomial with n_j coefficients A_i (and the same for $B(x)$). Each of these "coefficients" is itself a polynomial with $\prod_{i=1}^{j-1} n_i$ coefficients. Then the recursive KA for n_j coefficients is used until the recursion eventually terminates. The number of needed multiplications is as follows:

$$\#\text{MUL}_{\prod_{i=1}^{j} n_i} = (1/2)^j \prod_{i=1}^{j} n_i(n_i+1)$$

The number of additions is a complex expression [6]. There is a simple rule for the generic recursive KA to be most efficient: use the factorization of a number n with multiple prime factors combined with an increasing sequence of steps, i.e., KA for $n = \prod_{i=1}^{j} n_i$ with $n_i \geq n_{i+1}$, e.g., $2\cdot2\cdot3\cdot5$ for polynomials with 60 coefficients. In this case the polynomials with 60 coefficients are first split into an upper and lower half of 30 coefficients each, and the KA for two coefficients is applied to these halves. These polynomials of 30 coefficients are again split in half, and the KA for two coefficients is applied to the two halves. In the next recursion step the polynomials of 15 coefficients are split into 3 parts of 5 coefficients each, and so on. Note that in most cases the simple recursive KA that splits the operands in two halves of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ is more efficient than the general recursive KA, especially when the number of coefficients n has large prime factors.

The recursive KA versions are more efficient than the one-iteration KA. For example, instead of using the one-iteration KA for $n = 31$ coefficients one can use the recursive KA and split the 31 coefficient polynomial into two polynomials of 15 and 16 coefficients, respectively. Alternatively you 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 has to be stored due to the recursive nature. This might reduce the efficiency of the recursive KA variants for small-sized polynomials.

The simple recursive KA is the easiest and most efficient way to implement the KA when the number of coefficients is unknown at implementation time or if it often changes. It is especially efficient if special cases are implemented as basis of the recursion. Providing special cases for $n = 2$ and $n = 3$ coefficients using the one-iteration KA as well as for $n = 9$ using the KA that splits the operands into three parts recursively yield efficient running times.

Further improvements are possible due to the use of dummy coefficients. For example, to multiply two polynomials of $n = 15$ coefficients it might be useful to append a zero coefficient and use a recursive KA for $k = 16$ coefficients. Some operations can be saved whenever the leading zero coefficient is involved. However, this gains only little improvement to the simple recursive KA without using dummy coefficients.

The time-ratio between a multiplication and an addition on a given platform $r' = t_m/t_a$ where t_m and t_a denotes the cost for a multiplication and an addition, respectively. Let #MUL and #ADD be the number of multiplications and additions that the KA needs to multiply two polynomials of n coefficients, and $r = \frac{\#\mathrm{ADD}-(n-1)^2}{n^2-\#\mathrm{MUL}}$. If the actual ratio r' on a given hardware platform is larger than r than it is more efficient to use the KA instead of the ordinary method. One can show that the KA always outperforms the ordinary multiplication method if $r' > 3$, i.e., if one multiplication takes longer than three additions the KA performs faster than the ordinary schoolbook method. For some cases the KA is always faster than the schoolbook method, i.e., it needs less multiplications and additions. However, there are also cases where it is more efficient to use a combination of KA and the schoolbook method. For example, consider the case $n = 8$ and a platform with $r' = 2$. When applying the elementary recursive KA it needs 27 multiplications and 100 additions, resulting in $r = 1.38$. Since $r' > r$ it is more efficient to use the KA. However, for $n = 4$ the ratio is $r = 2.14$. Thus it is more efficient to use the ordinary method to multiply polynomials with four coefficients. Therefore it is efficient to apply one recursive KA step, and then use the schoolbook method to multiply the polynomial halves of degree three.

For some applications it is wise to use efficient underlying macros to multiply two polynomials with w coefficients. For example, there might be a macro to multiply two polynomials with four coefficients. Then two polynomials with $n = 20$ coefficients can be multiplied by using the KA for $20/4 = 5$

coefficients. In most cases it is efficient to use a mixture of different recursive steps combined with different underlying macros. Note that there might be optimized versions of the KA for special underlying coefficient rings like binary or prime fields. Also note that the KA can be applied to squaring polynomials by simply replacing all the coefficient multiplications by coefficient squarings. Although there is no special form of a squaring KA there still might be a performance gain compared to the ordinary squaring method which requires n squarings, $n(n-1)/2$ multiplications and $(n-1)^2$ additions. However, this varies for different platforms and depends on the ratio in time between a squaring and a multiplication. In most cases, i.e., if n is not very small, the squaring KA outperforms the ordinary squaring method [6].

Further information about the Karatsuba and similar algorithms can be found in [1] and [4]. Further aspects of efficient implementations are presented in [6] and [2].

Applications

Multiplying two polynomials efficiently is an important issue in a variety of applications, including signal processing, cryptography and coding theory. In particular, the KA is used to multiply elements of a finite field $GF(p^m)$, followed by a modular reduction afterwards. The KA can also be used for squaring in finite fields. The KA can be found in the modular multiplication of elliptic curve cryptography and sometimes RSA. Note that for large operands, other algorithms such as fast Fourier transformation offer advantages over the KA.

References

[1] D. J. Bernstein. Multidigit Multiplication for Mathematicians. *Preprint*, 1998.

[2] S. S. Erdem. Improving the Karatsuba-Ofman Multiplication Algorithm for Special Applications. *Ph.D. Thesis*, Department of Electrical & Computer Engineering, Oregon State University, November 8, 2001.

[3] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics - Doklady*, 7, 1963, 595-596.

[4] D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Massachusetts, 3rd edition, 1997.

[5] C. Paar. *Efficient VLSI Architecture for Bit Parallel Computation in Galois Fields*. PhD Thesis, Institute for Experimental Mathematics, University of Essen, Germany, 1994.

[6] A. Weimerskirch and C. Paar. Generalizations of the Karatsuba Algorithm for Efficient Implementations. *Technical Report, Ruhr-University Bochum*, 2002. Available at http://www.crypto.rub.de.