Generalizations of the Karatsuba Algorithm for Efficient Implementations

André Weimerskirch and Christof Paar

Communication Security Group
Department of Electrical Engineering & Information Sciences
Ruhr-Universität Bochum, Germany
email: {weika, cpaar}@crypto.rub.de

Abstract

In this work we generalize the classical Karatsuba Algorithm (KA) for polynomial multiplication to (i) polynomials of arbitrary degree and (ii) recursive use. We determine exact complexity expressions for the KA and focus on how to use it with the least number of operations. We develop a rule for the optimum order of steps if the KA is used recursively. We show how the usage of dummy coefficients may improve performance. Finally we provide detailed information on how to use the KA with least cost, and also provide tables that describe the best possible usage of the KA for polynomials up to a degree of 127. Our results are especially useful for efficient implementations of cryptographic and coding schemes over fixed-size fields like $GF(p^m)$.

Keywords: `polynomial multiplication, Karatsuba Algorithm, finite fields, cryptography, coding theory`

1 Introduction

Multiplying two polynomials efficiently is an important issue in a variety of applications, including signal processing, cryptography and coding theory. The present paper provides a generalization and detailed analysis of the algorithm by Karatsuba [2] to multiply two polynomials which was introduced in 1962. The Karatsuba Algorithm (KA) saves coefficient multiplications at the cost of extra additions compared to the schoolbook or ordinary multiplication method. We consider the KA to be efficient if the total cost of using it is less than the cost of the ordinary method. If we assume that we know the cost ratio between one multiplication and one addition we can decide which method is more efficient.

In order to simplify the problem we assume that the maximum degree of the two polynomials which are multiplied is identical. Knuth [3] gives a brief introduction on how to multiply polynomials in a fast way. He demonstrates an algorithm very similar to the KA which he calls "digital method", and achieves a complexity of $O(n2^{\sqrt{2\log n}}\log n)$ for very large polynomials. Another well known fast approach for polynomial multiplication is the Fast Fourier Transform (FFT). A theoretical upper bound for very large numbers can be shown as $O(n\log n\log\log n)$. A comprehensive survey of different methods to multiply polynomials was given by Bernstein [1]. Assuming that the polynomials represent elements of a Galois Field $GF(p^m)$ we could use the KA to multiply two elements of the finite field and reduce the resulting polynomial afterwards. Another approach is shown by Lempel, Seroussi and Winograd in [4] and by Winograd in [7]. They demonstrate algorithms which perform a modular multiplication and derive asymptotical lower bounds for these.

While many algorithms have lower asymptotic complexity than the KA the later one shows better performance for polynomials of small degree as they are used in many applications. In this paper we show in detail how to use the KA in an efficient way, both iteratively and recursively. We provide methods and tables to ease this task, and give a detailed count of the numbers of elementary additions and multiplications needed. We also show that for many polynomials using the KA needs less multiplications *and* additions than the schoolbook method. The work is organized as follows. Section 2 introduces the KA. Section 3 extends the KA for polynomials of arbitrary degree in one iteration, and Section 4 enhances the KA to recursive use. Section 5 describes the complexity of the KA when using it for squaring. Section 6 improves the KA by using dummy coefficients, and Section 7 provides the conclusion.

2 Preliminaries: Karatsuba Algorithm

Let R be a ring. Let $A(x)$ and $B(x)$ be degree-d polynomials over R. The Karatsuba Algorithm (KA) describes a method to multiply two polynomials with coefficients in R. There are two ways to derive the KA: the Chinese Remainder Theorem [1] and simple algebraic transformations. The KA can easily be applied recursively for polynomials which have 2^i coefficients. But first we show an example of the schoolbook method.

2.1 Schoolbook Method

The usual way to multiply two polynomials is often called the schoolbook method. Consider two degree-d polynomials with $n = d + 1$ coefficients:

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

Then the product $C(x) = A(x)\ B(x)$ is calculated as

$$C(x) = \sum_{i=0}^{d} x^i \cdot \left(\sum_{s+t=i; s,t \geq 0} a_s b_t \right) = \sum_{i=0}^{d} \sum_{j=0}^{d} a_i b_j x^{i+j} \tag{1}$$

The polynomial $C(x)$ can be obtained with n^2 multiplications and $(n-1)^2$ additions.

2.2 KA for Degree-1 Polynomials

The KA for degree-1 polynomials was introduced by Karatsuba in [2]. We will now develop the KA through simple algebraic manipulations. Consider two degree-1 polynomials $A(x)$ and $B(x)$.

$$A(x) = a_1 x + a_0, \ B(x) = b_1 x + b_0$$

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:

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

We need four additions and three multiplications to compute $C(x)$. Using the schoolbook method we need four multiplications and one addition, thus we save one multiplication and need three extra additions.

For application in practice, e.g., multi-precision multiplication we are interested in the particular value of the ratio between the cost of one multiplication and one addition for which the KA is efficient. Let r be the ratio between the cost of one multiplication and one addition on a specific implementation platform. Then $r = t_m/t_a$ where t_m and t_a denote the cost of one multiplication and one addition, respectively. The cost for the schoolbook method c_s can be calculated as $c_s = 1t_a + 4t_m$. The cost of the KA can be similarly obtained as $c_k = 4t_a + 3t_m$. We want to know the ratio r when the cost of the KA is less than for the schoolbook method. Therefore we obtain $c_k < c_s \Leftrightarrow 4t_a + 3t_m < 1t_a + 4t_m \Leftrightarrow 3 < r$. If the ratio between the cost of one multiplication and one addition is greater than three it is more efficient to use the KA.

2.3 Recursive KA for Polynomials of Degree $2^i - 1$

The KA can be applied in a recursive way as shown in Algorithm 1. This is straightforward for polynomials whose number of coefficients n is a power of 2. 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 halfs are used as before, i.e., as if they were coefficients. The algorithm becomes recursive if it is applied again to multiply these polynomial halfs. The next iteration step splits these polynomials again in half. The algorithm eventually terminates after t steps. In the final step the polynomials degenerate into single coefficients. Since every step exactly halves the number of coefficients, the algorithm terminates after $t = \log_2 n$ steps. Let #MUL and #ADD be the number of multiplications and additions in the underlying ring. Then the complexity to multiply two polynomials with n coefficients is as follows [6]:

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

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

Algorithm 1 ***Recursive KA, 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` $A \cdot B$

`Let` $A(x) = A_u(x)\ x^{N/2} + A_l(x)$

`and` $B(x) = B_u(x)\ x^{N/2} + B_l(x)$

$D_0 = KA(A_l, B_l)$

$D_1 = KA(A_u, B_u)$

$D_{0,1} = KA(A_l + A_u, B_l + B_u)$

`return` $D_1 x^N + (D_{0,1} - D_0 - D_1)x^{N/2} + D_0$

3 One-Iteration KA for Polynomials of Arbitrary Degree

As mentioned above it is straightforward to apply the KA to polynomials which have 2^i, i positive integer, coefficients (if $i > 1$, we apply the KA recursively). However, it is not obvious how to apply the KA to polynomials with a number of coefficients which has the form $2^j\ n$, with j a non-negative integer and n an odd integer, $n > 1$. Even though the original trick can be applied j times, the problem of multiplying polynomials with n coefficients remains. In particular if $j = 0$, i.e., if the number of coefficients is odd, the classical KA cannot be applied in a straight forward manner. We start by giving a simple example. Then a general algorithm is provided, followed by a complexity analysis.

3.1 KA for Degree-2 Polynomials

Consider two degree-2 polynomials:

$$A(x) = a_2x^2 + a_1x + a_0,\ B(x) = b_2x^2 + b_1x + b_0$$

with the auxiliary variables

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

$C(x) = A(x)\ B(x)$ is computed with an extended version of the KA

$$C(x) = D_2x^4 + (D_{1,2} - D_1 - D_2)x^3 + (D_{0,2} - D_2 - D_0 + D_1)x^2 + (D_{0,1} - D_1 - D_0)x + D_0$$

We need 13 additions and 6 multiplications. Using the schoolbook method we needs 4 additions and 9 multiplications. Let us take a look at the ratio r. We obtain $c_k < c_s \Leftrightarrow 13t_a + 6t_m < 4t_a + 9t_m \Leftrightarrow 3 < r$. If $r > 3$ it is more efficient to use the KA for degree-2 polynomials.

3.2 KA for Polynomials of Arbitrary Degree

The following algorithm describes a method to multiply two arbitrary polynomials with n coefficients using a one-iteration KA.

Algorithm 2 *Consider two degree-d polynomials with $n = d + 1$ coefficients*

$$A(x) = \sum_{i=0}^{d} a_ix^i,\ B(x) = \sum_{i=0}^{d} b_ix^i$$

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

$$D_i := a_ib_i \tag{2}$$

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

$$D_{s,t} := (a_s + a_t)\ (b_s + b_t) \tag{3}$$

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

$$c_0 = D_0 \tag{4}$$

$$c_{2n-2} = D_{n-1} \tag{5}$$

$$c_i = \begin{cases} \sum_{s+t=i;t>s\geq 0} D_{s,t} - \sum_{s+t=i;n>t>s\geq 0} (D_s + D_t) & \text{for odd } i,\ 0 < i < 2n-2 \\ \sum_{s+t=i;t>s\geq 0} D_{s,t} - \sum_{s+t=i;n>t>s\geq 0} (D_s + D_t) + D_{i/2} & \text{for even } i,\ 0 < i < 2n-2 \end{cases} \tag{6}$$

Correctness of the algorithm First we prove (6) for odd i and then for even i. Using (1) we obtain

$$c_i = \sum_{s+t=i;s,t\geq 0} a_s b_t$$

Now consider some $D_{s,t}$. Each $D_{s,t}$ is calculated as $(a_s + a_t)\ (b_s + b_t) = a_s b_s + a_s b_t + a_t b_s + a_t b_t$ with $s + t = i$ and $t > s \geq 0$. The sum $\sum_j D_{s,t}$ consists of all combinations of coefficients $a_s b_t$ with $s + t = i$ and $s \neq t$, and all $a_s b_s$ and $a_t b_t$ where s, t are summands of $i = s + t$. We must subtract all of the latter products, which were denoted by D_s.

$$\sum_j D_{s,t} = \sum_{s+t=i;t>s\geq 0} (a_s b_s + a_s b_t + a_t b_s + a_t b_t) = \sum_{s+t=i;t>s\geq 0} (a_s b_t + a_t b_s) + \sum_{s+t=i;t>s\geq 0} (a_s b_s + a_t b_t)$$

This can be re-written as

$$\sum_j D_{s,t} = \sum_{s+t=i;t>s\geq 0} a_s b_t + \sum_{s+t=i;t>s\geq 0} a_s b_s = c_i + \sum_{s+t=i;t>s\geq 0} D_s$$

with $s \neq t$ for odd i. For even i there are two products $a_s b_s$ for $s = t = i/2$ in the sum $D_{s,t}$, such that we have to take care of the extra product:

$$\sum_j D_{s,t} = \sum_{s+t=i;t>s\geq 0} a_s b_t + \sum_{s+t=i;t>s\geq 0} a_s b_s = (c_i - D_{i/2}) + \sum_{s+t=i;t>s\geq 0} D_s$$

These equations can easily be transformed to (6). Equations (4) and (5) are special cases of (6) for even i, which ends the proof. $\square$

3.3 Complexity of KA for Arbitrary Polynomials

In order to determine the number of additions and multiplications we first analyze the number of auxiliary variables D_i and $D_{s,t}$, denoted by $\#D_i$ and $\#D_{s,t}$.

Lemma 1 *Let $A(x), B(x)$ and $C(x)$ be defined as in Algorithm 1, and the variables D_i and $D_{s,t}$ as defined in (2) and (3), respectively. The numbers of auxiliary variables is then given as:*

$$\begin{aligned}
\#D_i &= n \\
\#D_{s,t} &= \frac{1}{2}n^2 - \frac{1}{2}n \\
\#D &= \#D_i + \#D_{s,t} = \frac{1}{2}n^2 + \frac{1}{2}n
\end{aligned}$$

Proof We calculate for each $i = 0, \ldots, n-1$ an auxiliary variable D_i, therefore we need n variables D_i. The number of variables $D_{s,t}$ can be determined as follows. Each $D_{s,t}$ describes one pair of coefficients a_s, a_t. Each possible pair occurs once. Therefore the number of possible pairs out of n elements is $\binom{n}{2}$ and we obtain $D_{s,t} = \binom{n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$. $\square$

Corollary 1 *Let $\#MUL$ and $\#ADD$ be the number of multiplications and additions, respectively, needed to multiply two polynomials with n coefficients using the KA. Then using the extended KA we need one multiplication for determining each variable D_i and $D_{s,t}$, which results in*

$$\#MUL = \frac{1}{2}n^2 + \frac{1}{2}n$$

	KA		schoolbook		
n	#MUL	#ADD	#MUL	#ADD	r
2	3	4	4	1	3
3	6	13	9	4	3
5	15	46	25	16	3
7	28	99	49	36	3
11	66	265	121	100	3

Table 1: Comparison of the KA and the schoolbook method for small primes.

multiplications.

We need two additions to obtain an auxiliary variable $D_{s,t}$ resulting in 2 $\#D_{s,t}$ additions. Furthermore, we need 2 additions to determine c_{n-2}, c_{n-3}, c_1 and c_2, 5 additions for c_{n-4}, c_{n-5}, c_3 and c_4 and so on. So we need $4 \cdot (3i-1)$ additions for determining all c_i (one term must be removed for $i = \frac{n}{2}$). For each even i we need a further addition. This results in

$$\#ADD = 2\#D_{s,t} + 4\sum_{i=1}^{(n-1)/2}(3i-1) - (3\frac{n-1}{2} - 1) + (n-1) - 1 = \frac{5}{2}n^2 - \frac{7}{2}n + 1$$

For large n, the one-iteration KA approaches 0.5 n^2 coefficient multiplications, which is about half as many as the schoolbook method. The number of additions approaches 2.5 n^2, which is more than the n^2 of the schoolbook method. Note that KA is efficient if the ratio between multiplication and addition on a given platform is larger than 3. This is due to the fact that

$$c_k < c_s \Leftrightarrow (1/2n^2 + 1/2n)t_m + (5/2n^2 - 7/2n + 1)t_a < n^2 t_m + (n-1)^2 t_a \Leftrightarrow r > 3$$

Since $r = 3$ for $n = 2$ this is a sharp bound.

For short polynomials, especially for those with a prime number of coefficients (where a recursive application of the basic KA is not straightforward), the method can yield complexities which are relevant in applications. Table 1 shows a few expected values.

4 Recursive Application of the KA

We can use the KA in a recursive way to decrease the number of operations. First we show the complexity of a simple extension to the basic recursive KA. Then we show how two polynomials with $n \cdot m$ coefficients can be multiplied using a simple one-step recursion of the KA. We will divide the polynomial into m polynomials each with n coefficients and use the KA for m polynomials, i.e., we consider the original polynomial to have m coefficients. To multiply these, the KA for n coefficients is used in the recursive step. We will write "KA for $n \cdot m$" coefficients to mean that the KA for m coefficients was used on polynomials with n coefficients. These n coefficient polynomials, in turn, are multiplied using the KA for n coefficients. Furthermore we will determine a method to multiply polynomials with $\prod_{i=1}^{j} n_i$ coefficients by recursion.

4.1 Recursive KA for Arbitrary Polynomials

If the number of coefficients n is no power of 2 Algorithm 1 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. We call this the *simple recursive KA*. In this case the KA is less efficient than for powers of 2. A lower bound for the number of operations is given by the complexity of the KA for $n = 2^i$ coefficients as described in Section 2.3. Thus the lower bound for the number of multiplications is $\#\text{MUL}_{low} = n^{\log_2 3}$ whereas the lower bound for the number of additions is $\#\text{ADD}_{low} = 6n^{log_2 3} - 8n + 2$. We obtain the upper bound by empirical tests as $\#\text{MUL}_{up} = 1.39\ n^{\log_2 3}$. 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 by adding the recursion basis $\#\text{MUL}_3 = 6$, 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

$$\#\text{MUL}_{up} = 1.20\ n^{\log_2 3}.$$

The number of additions in the worst case, i.e., the upper bound can be obtained in a similar fashion.

$$\begin{aligned}
\#\text{ADD}_1 &= 0,\ \#\text{ADD}_2 = 4 \\
\#\text{ADD}_n &= \underbrace{2\ \#\text{ADD}_{\lceil n/2 \rceil} + \#\text{ADD}_{\lfloor n/2 \rfloor}}_{\text{recursive application of KA}} + \underbrace{2\ \lfloor n/2 \rfloor}_{\text{calculation of } D_{s,t}} \\
&+ \underbrace{2\ \lfloor n/2 \rfloor + 2\lceil n/2 \rceil - 2}_{\text{addition of auxiliary variables}} + \underbrace{2\ \lceil n/2 \rceil - 2}_{\text{overlaps}} \\
&= 2\ \#\text{ADD}_{\lceil n/2 \rceil} + \#\text{ADD}_{\lfloor n/2 \rfloor} + 4(n-1)
\end{aligned}$$

When applying the one-iteration KA for two and three coefficients as basis of the recursion, the anchor $\#\text{ADD}_3 = 13$ is included. An upper bound of additions is then obtained as

$$\#\text{ADD}_{up} = 7\ n^{\log_2 3}.$$

4.2 KA for Degree-5 Polynomials

Consider the two polynomials

$$A(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

$$B(x) = b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

Then $A(x)$ and $B(x)$ can be written as

$$A(x) = A_1(x)x^3 + A_0,\ B(x) = B_1(x)x^3 + B_0$$

with

$$A_1(x) = a_5x^2 + a_4x + a_3,\ A_0(x) = a_2x^2 + a_1x + a_0$$

$$B_1(x) = b_5x^2 + b_4x + b_3,\ B_0(x) = b_2x^2 + b_1x + b_0$$

Now we apply the KA for degree-1 polynomials. Notice that the coefficients of the polynomials $A_i(x)$ and $B_i(x)$ are themselves polynomials and the multiplications of these coefficients result in further applications of the KA for degree-2 polynomials. In the following, we drop the notation $''(x)''$ for convenience.

$$D_0 = A_0B_0,\ D_1 = A_1B_1,\ D_{0,1} = (A_0 + A_1)\ (B_0 + B_1)$$

Method	Multiplications #MUL	Additions #ADD
KA for $3 \cdot 2$	18	59
KA for $2 \cdot 3$	18	61

Table 2: Costs for multiplying degree-5 polynomials.

Thus, we obtain

$$C(x) = D_1 x^6 + (D_{0,1} - D_0 - D_1)x^3 + D_0$$

The KA for degree-1 polynomials needs four additions and three multiplications of degree-2 polynomials. Each multiplication is solved by the KA for degree-2 polynomials for which we need 13 additions and 6 multiplications. The number of additions must be carefully analyzed. In order to determine the auxiliary variables $D_{s,t}$ we need two additions of degree-3 polynomials. Two degree-d polynomials can be added by adding all $n = d + 1$ coefficients. Furthermore two degree-5 polynomials have to be added. Notice that there are some overlaps resulting in four further additions. As an example for the overlaps look at D_0 and $(D_{0,1} - D_0 - D_1)x^3$. The first polynomial has degree-4, the second one degree-7. To determine c_3 and c_4 we have to add coefficients from the first and second polynomial. Overall we need 18 multiplications and 59 additions.

For polynomials with 6 coefficients the KA is not unique. Above, we first applied the KA for 2 and then for 3 coefficients. One can also first use the KA for 3 and then for 2 coefficients. The needed operations can be calculated with similar arguments as above. The number of operations for both possibilities are depicted in Table 2. The first example is for the KA for $3 \cdot 2$, meaning that first the KA for 2 polynomials where the coefficients themselves are polynomials with 3 coefficients is used. The second example is the KA for $2 \cdot 3$ in which case the KA is used for polynomials for 3 polynomials where the coefficients themselves are polynomials with 2 coefficients. Notice that the number of multiplications is the same for both approaches. We will see that this is always true. We will also show that it is always more efficient to apply the KA for $n \cdot m$ than for $m \cdot n$ if $n > m$.

4.3 KA for Polynomials with $n \cdot m$ Coefficients

This section analyzes the complexity of the KA with a single step recursion. Let $A(x) = \sum_{i=0}^{nm-1} a_i x^i$. This can be written as $A(x) = \sum_{s=0}^{m-1} A_s(x)\, x^{ns}$ where $A_s(x)$ are degree-$(n-1)$ polynomials. $B(x)$ can be written in the same way. The KA is applied to the polynomials $A(x)$ and $B(x)$ that are considered to have m coefficients. In the recursive step the KA is applied to the polynomials with n coefficients and merged at the end. The number of additions needed for the KA for polynomials with n coefficients is denoted by $\#\mathrm{ADD}_n$, the number of multiplications with $\#\mathrm{MUL}_n$. Let $\#(D_{s,t})_n$ be the number of auxiliary variables $D_{s,t}$ needed for the KA for n coefficients. When applying the KA for m coefficients, we need $\#\mathrm{MUL}_m$ multiplications. Each one of these m multiplications requires $\#\mathrm{MUL}_n$ multiplications. This results in

$$\#\mathrm{MUL}_{n \cdot m} = \#\mathrm{MUL}_m \cdot \#\mathrm{MUL}_n = \left(\frac{1}{2}m^2 + \frac{1}{2}m\right) \cdot \left(\frac{1}{2}n^2 + \frac{1}{2}n\right)$$

Notice that the order of the recursion does not make any difference thus far.

The number of additions is achieved from the recursive application of the KA: the number of additions to build the variables $D_{s,t}$, the additions of the variables D_i and $D_{s,t}$ and the number of overlaps. For each recursive application of the KA we need $\#\mathrm{ADD}_n$ additions, overall $\#\mathrm{MUL}_m \cdot \#\mathrm{ADD}_n$ additions. We need two additions for polynomials with n coefficients to build a variable $D_{s,t}$, altogether $2n \cdot (\#D_{s,t})_m$ additions. Furthermore $(\#\mathrm{ADD}_m - 2(\#D_{s,t})_m)$ additions of the variables $D_{s,t}$ are needed, each of it having $(2n-1)$ coefficients. Finally we have to consider the overlaps resulting in $((2m-1)\cdot(2n-1)-(2nm-1))$ additions. Overall we obtain

$$\begin{aligned}\#\mathrm{ADD}_{n\cdot m} \quad = \quad & \underbrace{\#\mathrm{MUL}_m \cdot \#\mathrm{ADD}_n}_{\text{recursive application of KA}} \quad + \quad \underbrace{2n\cdot(\#D_{s,t})_m}_{\text{calculation of } D_{s,t}} \\ & + \underbrace{(2n-1)\cdot(\#\mathrm{ADD}_m - 2(\#D_{s,t})_m))}_{\text{addition of auxiliary variables}} + \underbrace{((2m-1)\cdot(2n-1)-(2nm-1))}_{\text{overlaps}}\end{aligned}$$

This results in

$$\#\mathrm{ADD}_{n\cdot m} = \frac{5}{4}m^2n^2 + \frac{5}{4}mn^2 + \frac{9}{4}m^2n - \frac{23}{4}mn - m^2 + m + 1 \tag{7}$$

Now we will prove that it is more efficient to use the KA for $n\cdot m$ with $n \geq m$. Since the number of multiplications is identical we only have to prove that this holds for the number of additions.

Lemma 2 *Let $n \geq m$ and $n,m \geq 2$. The number of additions for the KA for $n\cdot m$ is not greater than the number of additions for the KA for $m\cdot n$, i.e., $\#ADD_{n\cdot m} \leq \#ADD_{m\cdot n}$.*

Proof Let $n \geq m \geq 2$. Assume that $\#\mathrm{ADD}_{m\cdot n} < \#\mathrm{ADD}_{n\cdot m}$. Then

$$\begin{aligned} & \frac{5}{4}m^2n^2 + \frac{5}{4}nm^2 + \frac{9}{4}n^2m - \frac{23}{4}nm - n^2 + n + 1 \\ & \quad < \frac{5}{4}m^2n^2 + \frac{5}{4}mn^2 + \frac{9}{4}m^2n - \frac{23}{4}nm - m^2 + m + 1 \\ \Rightarrow \quad & \frac{5}{4}nm^2 + \frac{9}{4}m^2m - n^2 + n + 1 < \frac{5}{4}mn^2 + \frac{9}{4}m^2n - m^2 + m + 1 \\ \Rightarrow \quad & n^2(m-1) - n(m-1)(m+1) < (m-1)(-m) \\ \Rightarrow \quad & (1-n)\cdot(m-n) < 0 \Rightarrow m-n > 0 \Rightarrow n < m \end{aligned}$$

This is a contradiction and hence $\#\mathrm{ADD}_{n\cdot m} \leq \#\mathrm{ADD}_{m\cdot n}$ for $n \geq m$. □

4.4 KA for Polynomials with $\prod_{i=1}^{j} n_i$ Coefficients

We will determine the number of additions and multiplications needed when using the KA recursively for arbitrary polynomials with $\prod_{i=1}^{j} n_i$ coefficients. We call this KA version the *general recursive KA*. To be exact, we consider polynomials with a maximum degree of $\prod_{i=1}^{j} n_i - 1$. Consider two such polynomials $A(x)$ and $B(x)$ with $\prod_{i=1}^{j} n_i$ coefficients. First we write $A(x) = \sum_{s=0}^{n_j-1} A_s x^{s\cdot\prod_{i=1}^{j-1} n_i}$ as polynomial with n_j coefficients A_i and do the same for $B(x)$. Each of these "coefficients" is itself a polynomial with $\prod_{i=1}^{j-1} n_i$ coefficients. Then we use the KA for n_j. To obtain the number of multiplications we repeatedly apply the KA. The method introduced in Section 4.3 for $n\cdot m$ coefficients is a special case of this one.

In each application of the KA there are $\#\mathrm{MUL}_{n_i}$ multiplications needed

$$\#\mathrm{MUL}_{\prod_{i=1}^{j} n_i} = \prod_{i=1}^{j}(\frac{1}{2}n_i^2 + \frac{1}{2}n_i) = (\frac{1}{2})^j \prod_{i=1}^{j} n_i(n_i+1) \tag{8}$$

Again, the number of multiplications is independent of the order of the recursion. The number of additions can be developed as follows. We introduce a term w_l, $l = 1 \ldots j$, where j is the recursion depth, which describes the number of additions for one recursive application. This includes the additions to build $D_{s,t}$, to add all auxiliary variables, and it takes into account the overlaps. We comprise the last three points to

$$\begin{aligned} w_l &= w_{(\prod_{i=1}^{l-1} n_i) n_l} = \underbrace{2\prod_{i=1}^{l-1} n_i \cdot (\#D_{s,t})_{n_l}}_{\text{calculation of } D_{s,t}} \\ &+ \underbrace{(2\prod_{i=1}^{l-1} n_i - 1)\cdot(\#\mathrm{ADD}_{n_l} - 2(\#D_{s,t})_{n_l}))}_{\text{addition of auxiliary variables}} + \underbrace{((2n_l - 1)\cdot(2\prod_{i=1}^{l-1} n_i - 1) - (2\prod_{i=1}^{l} n_i - 1))}_{\text{overlaps}} \end{aligned}$$

describing the number of additions for the KA for n_l polynomials with $\prod_{i=1}^{l-1} n_i$ coefficients without looking at the recursion. This can be simplified to

$$w_l = 4\cdot\prod_{i=1}^{l} n_i\cdot(n_l - 1) - \frac{3}{2}n_l^2 + \frac{1}{2}n_l + 1$$

Moreover we need to look at the additions required for the recursion and define the number of additions recursively

$$\#\mathrm{ADD}_{\prod_{i=1}^{j} n_i} = w_j + \#\mathrm{MUL}_{n_j}\cdot\#\mathrm{ADD}_{\prod_{i=1}^{j-1} n_i}$$

with initial condition

$$\#\mathrm{ADD}_{n_1} = \frac{5}{2}{n_1}^2 - \frac{7}{2}n_1 + 1.$$

Using (7) this recursion can be transformed to

$$\#\mathrm{ADD}_{\prod_{i=1}^{j} n_i} = \prod_{i=3}^{j}\#\mathrm{MUL}_{n_i}\cdot\#\mathrm{ADD}_{n_1\cdot n_2} + \sum_{l=3}^{j-1}(\prod_{i=l+1}^{j}\#\mathrm{MUL}_{n_i}\cdot w_l) + w_j$$

or

$$\#\mathrm{ADD}_{\prod_{i=1}^{j} n_i} = \prod_{i=2}^{j}\#\mathrm{MUL}_{n_i}\cdot\#\mathrm{ADD}_{n_1} + \sum_{l=2}^{j-1}(\prod_{i=l+1}^{j}\#\mathrm{MUL}_{n_i}\cdot w_l) + w_j \qquad (9)$$

Theorem 1 *Let n_i be integer values with $n_i \geq 2$. Then the application of the KA for $\prod_{i=1}^{j} n_i$ is most efficient (i.e., it needs the least number of additions and multiplications) for a permutation of $(n_i)_{i\in\{1,\dots,j\}}$ with $n_i \geq n_{i+1}$ for $1 \leq i \leq j-1$.*

Proof The number of multiplications is independent of the order of the recursion steps and needs not to be considered. Let $n_1,\dots,n_j$ be a sequence n with $n_i \leq n_{i+1}$ for $i = 1,\dots,j-1$. Let $t = s+1$ and $n'_1,\dots,n'_j$ be another sequence n' with $n'_s = n_t, n'_t = n_s$ and $n'_i = n_i$ otherwise. We will show that the recursive KA for the sequence n' needs at least the same number of additions as the recursive KA for the sequence n. Since each sequence can be determined by changing adjacent elements, i.e., elements which are next to each other, the recursive KA for the original sequence n is most efficient. We only look at $j \geq 3$. For $j = 2$ see Lemma 2.

Let w'_i be the value $w_{(\prod_{j=1}^{i-1} n'_j)n'_i}$. To compare the number of additions we only look at the values that differ in the two sequences. Then $w'_s \leq w_s$, $w_s \leq w_t$, $w_t \leq w'_t$, $w'_t \leq w'_s$ and $w_i = w'_i$ otherwise. Furthermore $n'_s = n_t$, $n'_t = n_s$, $n'_s \leq n'_t$ and $n_t \leq n_s$. We want to prove by applying (9) that

$$\#\mathrm{ADD}_{\prod_{i=1}^{j} n_i} \leq \#\mathrm{ADD}_{\prod_{i=1}^{j} n'_i}$$

Since most of the values in the sequence are identical we only have to prove that

$$\begin{aligned}
& w'_s \ \#\mathrm{MUL}_{n_j} \#\mathrm{MUL}_{n_{j-1}} \cdot \ldots \cdot \#\mathrm{MUL}_{n'_{s+1}} + w'_t \#\mathrm{MUL}_{n_j} \#\mathrm{MUL}_{n_{j-1}} \cdot \ldots \cdot \#\mathrm{MUL}_{n'_{t+1}} \\
\geq \ & w_s \ \#\mathrm{MUL}_{n_j} \#\mathrm{MUL}_{n_{j-1}} \cdot \ldots \cdot \#\mathrm{MUL}_{n_{s+1}} + w_t \#\mathrm{MUL}_{n_j} \#\mathrm{MUL}_{n_{j-1}} \cdot \ldots \cdot \#\mathrm{MUL}_{n_{t+1}}
\end{aligned}$$

We note that $n'_{s+1} = n'_t$ and $n'_{t+1} = n_{t+1}$. We simplify this to

$$w'_s \cdot \#\mathrm{MUL}_{n'_t} \geq w_s \cdot \#\mathrm{MUL}_{n_t}$$

and obtain

$$\begin{aligned}
& (4(n_1 \cdot n_2 \cdot \ldots \cdot n_{s-1} \cdot n'_s) \cdot (n'_s - 1) - \frac{3}{2}{n'_s}^2 + \frac{1}{2}n'_s + 1) \cdot (\frac{1}{2}{n'_t}^2 + \frac{1}{2}n_t) \\
\geq \ & (4(n_1 \cdot n_2 \cdot \ldots \cdot n_{s-1} \cdot n_s) \cdot (n_s - 1) - \frac{3}{2}{n_s}^2 + \frac{1}{2}n_s + 1) \cdot (\frac{1}{2}{n_t}^2 + \frac{1}{2}n_t) \\
\Leftrightarrow \ & \\
& (4(n_1 \cdot n_2 \cdot \ldots \cdot n_{s-1} \cdot n_t) \cdot (n_t - 1) - \frac{3}{2}{n_t}^2 + \frac{1}{2}n_t + 1) \cdot (\frac{1}{2}{n_s}^2 + \frac{1}{2}n_s) \\
\geq \ & (4(n_1 \cdot n_2 \cdot \ldots \cdot n_{s-1} \cdot n_s) \cdot (n_s - 1) - \frac{3}{2}{n_s}^2 + \frac{1}{2}n_s + 1) \cdot (\frac{1}{2}{n_t}^2 + \frac{1}{2}n_t)
\end{aligned}$$

This can be simplified to

$$\begin{aligned}
& n_s \cdot n_t \cdot (\frac{1}{4}n_s - \frac{3}{4}n_t) - {n_s}^2 + n_s \geq n_t \cdot n_s \cdot (\frac{1}{4}n_t - \frac{3}{4}n_s) - {n_t}^2 + n_t \\
\Leftrightarrow \ & {n_s}^2 n_t - {n_t}^2 n_s + {n_t}^2 - {n_s}^2 - n_s - n_t \geq 0 \\
\Leftrightarrow \ & (n_s - n_t) \cdot (n_s n_t + 1 - n_t - n_s) \geq 0
\end{aligned}$$

Since $n_s \geq n_t \geq 2$ and $n_s n_t > n_t + n_s$ for $n_s, n_t \geq 2$ this is always true. □

4.5 How to Apply KA with the Least Cost

It was shown above that for a sequence n_i the KA for $\prod_i n_i$ coefficients is most efficient when $n_i \geq n_{i+1}$. We will now show that it is more efficient to apply the KA recursively than only once. Let k be the non-prime number of coefficients. Since k is not prime it can be written as $k = n \cdot m$ and $n, m \neq k$. We compare using the KA for k and for $n \cdot m$ coefficients. Since we use the KA recursively proving that the one-step recursion is more efficient than no recursion results immediately in the law to use the KA for the factorization of k with multiple prime factors.

Theorem 2 *Let $k = \prod_i {p_i}^{e_i}$ with p_i prime. Then the general recursive KA for $\prod_i \prod_{j=1}^{e_i} p_i$ with $p_i \geq p_{i+1}$ results in the least number of operations.*

Proof Let $n \geq m \geq 2$. First we compare the number of multiplications of the KA for k and $n \cdot m = k$ coefficients.

$$\begin{aligned}
\#\mathrm{MUL}_k &= \frac{1}{2}(nm)^2 + \frac{1}{2}nm \\
\#\mathrm{MUL}_{n \cdot m} &= \frac{1}{4}(n^2 + n)(m^2 + m)
\end{aligned}$$

We will show that $\#\mathrm{MUL}_{n \cdot m} \leq \#\mathrm{MUL}_k$

$$\frac{1}{4}n^2m^2 + \frac{1}{4}nm^2 + \frac{1}{4}n^2m + \frac{1}{4}nm \leq \frac{1}{2}n^2m^2 + \frac{1}{2}nm$$

$$\Leftrightarrow \frac{1}{4}nm^2 + \frac{1}{4}n^2m \leq \frac{1}{4}n^2m^2 + \frac{1}{4}nm \Leftrightarrow m+n \leq nm+1$$

Since $m + n \leq mn$ for $n, m \geq 2$ this is true. Now we compare the number of additions.

$$\begin{aligned}
\#\mathrm{ADD}_k &= \frac{5}{2}n^2m^2 - \frac{7}{2}nm + 1 \\
\#\mathrm{ADD}_{n\cdot m} &= \frac{5}{4}n^2m^2 + \frac{5}{4}n^2m + \frac{9}{4}m^2n - \frac{23}{4}mn - m^2 + m + 1
\end{aligned}$$

Then

$$\#\mathrm{ADD}_{n\cdot m} \leq \#\mathrm{ADD}_k \Leftrightarrow 5n^2 + 9mn - 4m + 4 \leq 5n^2m + 9n$$

This can be proven by induction on n with arbitrary m.

Basis $n = 2, m \leq n$ implies $m = 2$:

$$5 \cdot 4 + 9 \cdot 2m - 4m + 4 = 24 + 14m = 52 \leq 58 = 20m + 18 = 5 \cdot 4m + 18$$

Assume the assertion holds for $k \leq n$. Then we prove it for $n + 1$

$$5(n+1)^2 + 9m(n+1) - 4m + 4 = 5n^2 + 9mn - 4m + 4 + 10n + 5 + 9m \leq 5n^2m + 9n + 10n + 5 + 9m$$

$$\leq 5n^2m + 9n + 9 + 10(n+m) + m \leq 5n^2m + 9n + 9 + 10nm + m \leq 5(n+1)^2m + 9(n+1)$$

□

This results in a simple rule on how to use the general recursive KA: use the factorization of a number k with multiple prime factors combined with an increasing sequence of steps, i.e., KA for $k = \prod_{i=1}^{j} n_i$ with $n_i \geq n_{i+1}$, e.g., $2 \cdot 2 \cdot 3 \cdot 5$ for polynomials with 60 coefficients. However, a number of intermediate results has to be stored due to the recursive nature. This might reduce the efficiency for small-sized polynomials. As we showed in Section 3.3 the threshold value r for the KA to be efficient is always 3 for the one-iteration KA. Since we proved that the recursive KA is more efficient than the one-iteration KA it is obvious that $r \leq 3$ always holds. Appendix A displays the least number of operations needed to multiply two polynomials with the KA for polynomials with $n \in [2, 128]$ coefficients. There are two sections. The first one displays numbers for the general recursive KA as described in Section 4.4 while the second one describes numbers for the simple recursive KA as introduced in Section 4.1. The first column "n" displays the number of coefficients of the polynomials. The second column "distribution" displays how to use the KA for n coefficients (as mentioned above these are the prime factors in declining order). The next columns denote the number of multiplications and additions. The following column gives the ratio

$$r = \frac{\#\mathrm{ADD} - (n-1)^2}{n^2 - \#\mathrm{MUL}}.$$

If the time ratio r' between a multiplication and an addition on a given platform is larger than this r, then the KA is efficient. Note that very small values occur for values of n which largest prime factor is still small, while negative values for r occur if the KA needs less multiplications and additions than the schoolbook method. The very last column "opt(r)" describes which KA version results in a better value r, i.e., has smaller r. We use this as an indicator for a better performance. One can see that the simple recursive KA often outperforms the general recursive KA, especially when n has not only small prime factors. When the number of coefficients is unknown at implementation time or changes permanently it is wise to use the simple recursive KA since it is more efficient in most cases and easier to implement. There are more variants of the KA that might be considered for a fixed-size polynomial multiplication. For example, instead of using the one-iteration KA for

p	#ADD
2	$6n^{\log_2 3} - 8n + 2$
3	$\frac{29}{5} n^{\log_3 6} - 8n + \frac{11}{5}$
5	$\frac{39}{7} n^{\log_5 15} - 8n + \frac{17}{7}$
7	$\frac{49}{9} n^{\log_7 28} - 8n + \frac{23}{9}$

Table 3: Values $\#\mathrm{MUL}_n$ for $n = p^j$.

$n = 31$ coefficients you 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.

Note that it might be more efficient to use a combination of the KA and the schoolbook method. For example, take a look at $n = 8$ with the distribution [2 2 2] in Appendix A. Let us consider a platform with $r' = 2$. Since $r' > r = 1.38$ it is more efficient to use the KA. However the threshold value for $n =$ [2 2] is $r = 2.14$. Therefore it is most efficient to first use the KA for polynomials with 2 coefficients where the coefficients themselves are polynomials with 4 coefficients. These polynomials are then multiplied by the schoolbook method, and not by the recursive KA.

4.6 Complexity of KA

In this section we will analyze the asymptotic complexity of the KA. To simplify this we assume that the number of coefficients n is a power of some integer number, i.e., $n = p^j$. The number of multiplications can be determined by

$$\#\mathrm{MUL}_n = (\frac{1}{2}p^2 + \frac{1}{2}p)^j = (\frac{1}{2}p^2 + \frac{1}{2}p)^{\log_p n} = n^{\log_p(\frac{1}{2}p^2+\frac{1}{2}p)} \tag{10}$$

For very large p this converges to $(1/2)^j\ n^2$, for small p, especially $p = 2$ we derive a complexity of $n^{\log_2 3}$. As shown in Section 4.1 an upper bound of the simple recursive KA for arbitrary n is $1.20^{\log_2 3}$.

The number of additions is not as easily obtained. The number can be obtained by (9). Table 3 shows the number of additions needed for some values of p. We obtain for #ADD a complex formula

$$\begin{aligned}\#\mathrm{ADD}_n =& \left(1/2\,p^2 + 1/2\,p\right)^{j-1}\left(5/2\,p^2 - 7/2\,p + 1\right) + 4\,p^j\,(p-1) - 3/2\,p^2 + 1/2\,p + 1 + 1/2\ (p+1) \\ &\cdot \left(1/2\,p^2 + 1/2\,p\right)^j \left(3\,p^2\left(2\,\frac{1}{p\,(p+1)}\right)^j + 2\ \left(2\,\frac{1}{p\,(p+1)}\right)^j p - 8\,p\left(2\ (p+1)^{-1}\right)^j - 16\ \left(2\ (p+1)^{-1}\right)^j\right) \\ &\cdot (p+2)^{-1} - 1/2\ (p+1)\left(1/2\,p^2 + 1/2\,p\right)^j\left(-52\ (p+1)^{-2} + 8\,\frac{1}{p\,(p+1)^2} - 32\,\frac{p}{(p+1)^2}\right)(p+2)^{-1}\end{aligned}$$

This expression can be written as

$$\#\mathrm{ADD}_n = a \cdot n^{\log_p(\frac{1}{2}p^2+\frac{1}{2}p)} - 8n + b \tag{11}$$

with a and b some positive number smaller than n (actually $a \leq 6$ and $b \leq 3$). For very large p (11) converges to

$$\lim_{p\to\infty} \#\mathrm{ADD}_n = \lim_{p\to\infty} 5n^{\log_p(\frac{1}{2}p^2+\frac{1}{2}p)} - 8n + 3 = 5n^2 - 8n + 3$$

Note that for small p and large j the number of multiplications and additions is smaller compared to the schoolbook method that requires $(n-1)^2$ additions. For large integer numbers of p (10) converges to a complexity of n^2.

5 Squaring with KA

The KA can be applied to squaring polynomials by simply replacing all the coefficient multiplications by coefficient squarings while keeping the additions. 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. Let t_a, t_s and t_m be the time for an addition, a squaring and a multiplication, respectively. Let $r = t_m/t_a$ be the ratio between a multiplication and an addition as before, and let c_k and c_s be the cost of the KA and schoolbook method, respectively. For the comparison we use the upper bound complexity of the KA as stated in Section 4.1. We obtain

$$c_k < c_s \Leftrightarrow 1.20n^{\log_2 3}t_s + 7n^{\log_2 3}t_a < nt_s + n(n-1)/2t_m + (n-1)^2 t_a$$

We want to present the two extreme scenarios. In the first one a squaring comes for free, e.g., as it almost is the case for binary fields. Thus $t_s = 0$ such that we obtain

$$r > \frac{7n^{\log_2 3} - n^2 + 2n - 1}{(n^2 - n)/2}$$

Let r' be the right side term. If $r > r'$ then it is efficient to use KA instead of the schoolbook squaring method. For $r = 10$, i.e. a multiplication takes as long as 10 additions, the Squaring KA outperforms the schoolbook squaring method for $n >= 4$ if a squaring is for free and a multiplication does not perform faster than an addition. For $r = 2$, the Squaring KA outperforms the schoolbook method for $n >= 24$.

The second scenario is the case when a squaring takes as long as a multiplication, i.e., $t_s = t_m$. Then we obtain

$$r > \frac{7n^{\log_2 3} - n^2 + 2n - 1}{n/2(n+1) - 1.20n^{\log_2 3}}$$

Again let r' be the right side term. In this case, for $n >= 3$ the KA Squaring outperforms the schoolbook method if $r = 10$, and for $n >= 21$ it outperforms the schoolbook method if $r = 2$. If $r = 1$, i.e. a multiplication takes as long as an addition, then the KA Squaring outperforms the schoolbook squaring method for $n >= 44$.

Clearly, if $t_s = a\ t_m$ with $0 < a < 1$ then the range where KA Squaring outperforms the schoolbook method are in a similar range as above. Hence, one needs first to consider the ratio r in order to estimate the superior squaring method.

6 Improvement by Using Dummy Coefficients

To improve the KA we can use dummy coefficients, i.e. prefix zero coefficients, to reduce the number of operations. In Appendix A we observe that the general recursive KA for 11 coefficients needs more operations than the one for 12. Assume $A(x)$ and $B(x)$ are polynomials with 11 coefficients, i.e., of degree 10. Just by adding a dummy coefficient a_{11} and b_{11} with $a_{11} = b_{11} = 0$ we can reduce the total number of operations from 331 to 275. Furthermore it can be observed that the added coefficients imply that there are some computations in the algorithm which do not need to be done. Whenever a_{11} or b_{11} occurs in the computation we do not need to compute the result. We will show this for an example for polynomials with 11 coefficients.

Let $A(x) = \sum_{i=0}^{10} a_i \cdot x^i$ and $B(x) = \sum_{i=0}^{10} b_i \cdot x^i$ be two degree-10 polynomials. Then $A'(x) = \sum_{i=0}^{11} a_i \cdot x^i$ and $B'(x) = \sum_{i=0}^{11} b_i \cdot x^i$ with $a_{11} = b_{11} = 0$ are two degree-11 polynomials with $A(x) \cdot B(x) = A'(x) \cdot B'(x)$. We will apply the KA recursively for polynomials with 12 coefficients (in the sequence 3 - 2 - 2). First we rewrite the polynomials:

$$A'(x) = {A_1}^{(1)} \cdot x^6 + {A_0}^{(1)},\ B'(x) = {B_1}^{(1)} \cdot x^6 + {B_0}^{(1)}$$

and use the KA for degree-1 polynomials:

$${D_0}^{(1)} = {A_0}^{(1)} \cdot {B_0}^{(1)},\ {D_1}^{(1)} = {A_1}^{(1)} \cdot {B_1}^{(1)},\ {D_{0,1}}^{(1)} = ({A_0}^{(1)} + {A_1}^{(1)}) \cdot ({B_0}^{(1)} + {B_1}^{(1)})$$

${D_0}^{(1)}$ and ${D_{0,1}}^{(1)}$ are computed as usual and only the computation of ${D_1}^{(1)}$ saves some operations (note that ${D_1}^{(1)}$ has 10 and not 12 coefficients). Now we compute ${D_1}^{(1)}$.

$${D_1}^{(1)} = {A_1}^{(1)} \cdot {B_1}^{(1)}$$

For the second iteration we divide the polynomials once more

$${A_1}^{(1)} = {A_1}^{(2)} \cdot x^3 + {A_0}^{(2)},\ {B_1}^{(1)} = {B_1}^{(2)} \cdot x^3 + {B_0}^{(2)}$$

and obtain the auxiliary variables:

$${D_0}^{(2)} = {A_0}^{(2)} \cdot {B_0}^{(2)},\ {D_1}^{(2)} = {A_1}^{(2)} \cdot {B_1}^{(2)},\ {D_{0,1}}^{(2)} = ({A_0}^{(2)} + {A_1}^{(2)}) \cdot ({B_0}^{(2)} + {B_1}^{(2)})$$

As above, ${D_0}^{(2)}$ and ${D_{0,1}}^{(2)}$ do not change compared to the usual computation. To compute ${D_1}^{(2)}$ we need the KA for 2 coefficients instead of 3. Furthermore we save 2 additions to compute each ${D_{0,1}}^{(1)}$ and ${D_{0,1}}^{(2)}$ because ${A_1}^{(1)}$ and ${B_1}^{(1)}$ have only 5 coefficients, ${A_1}^{(2)}$ and ${B_1}^{(2)}$ only 2. To obtain the result we have to compute

$${D_1}^{(1)} = {A_1}^{(1)} \cdot {B_1}^{(1)} = {D_1}^{(2)} \cdot x^6 + ({D_{0,1}}^{(2)} - {D_0}^{(2)} - {D_1}^{(2)}) \cdot x^3 + {D_0}^{(2)}$$

Since ${D_1}^{(2)}$ has only 4 coefficients instead of 6 we save another 2 additions. To obtain the desired result we compute

$$A'(x) \cdot B'(x) = {D_1}^{(1)} \cdot x^{12} + ({D_{0,1}}^{(1)} - {D_0}^{(1)} - {D_1}^{(1)}) \cdot x^6 + {D_0}^{(1)}$$

We save another 2 additions because ${D_1}^{(1)}$ has only 10 coefficients instead of 12. So altogether we derive

$$\begin{aligned}
\#\mathrm{MUL}'_{11} &= \#\mathrm{MUL}_{12} - \#\mathrm{MUL}_3 + \#\mathrm{MUL}_2 = 54 - 6 + 3 = 51 \\
\#\mathrm{ADD}'_{11} &= \#\mathrm{ADD}_{12} - \#\mathrm{ADD}_3 + \#\mathrm{ADD}_2 - 4 \cdot 2 = 221 - 13 + 4 - 8 = 204
\end{aligned}$$

and

$$\#\mathrm{MUL}'_{11} +\ \#\mathrm{ADD}'_{11} = 51 + 204 = 255$$

compared to

$$\#\mathrm{MUL}_{11} +\ \#\mathrm{ADD}_{11} = 331$$

for the general recursive KA.

Note that computing ${D_1}^{(1)}$ by using the KA for 5 coefficients requires the same amount of operations. This simple approach can be enhanced by adding one or more dummy coefficients and using this approach recursively.

Now observe that the number of multiplications is exactly the same as it is for the simple recursive KA for 11 coefficients while the number of additions is slightly less. Without a formal proof we can state that the usage of dummy coefficients combined with the general recursive KA only results in a slight performance gain compared to the simple recursive KA. This is due to the fact that the simple recursive KA for n coefficients always needs less operations than the KA for $n+1$ coefficients because of the algorithm's construction. Since the simple recursive KA is more efficient than the general recursive KA we can only expect a slight performance gain.

7 Concluding Remarks

In this article we demonstrated several recursive algorithms to multiply two arbitrary polynomials by means of the Karatsuba Algorithm. We analyzed the complexity of these algorithms and described how to apply them most efficiently. In most cases the simple recursive KA yields the most efficient computation. By adding dummy coefficients the complexity might be slightly decreased.

References

[1] D. J. Bernstein. Multidigit Multiplication for Mathematicians. *Advances in Applied Mathematics*, to appear.

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

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

[4] A. Lempel, G. Seroussi and S. Winograd. On the Complexity of Multiplication in Finite Fields. *Theoretical Computer Science*, 22 (1983), 285-296.

[5] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*, 2nd Edition. Springer-Verlag, Berlin, Heidelberg, New York, 1982.

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

[7] S. Winograd. Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants. *Mathematical Systems Theory*, 10 (1977), 169-180.

A Complexity of KA

n	*General Recursive KA*				*Simple Recursive KA*			opt(r)
	distribution	#MUL	#ADD	r	#MUL	#ADD	r	
2	[2]	3	4	3.00	3	4	3.00	equal
3	[3]	6	13	3.00	6	13	3.00	equal
4	[2 2]	9	24	2.14	9	24	2.14	equal
5	[5]	15	46	3.00	15	46	3.00	equal
6	[3 2]	18	59	1.89	18	59	1.89	equal
7	[7]	28	99	3.00	24	85	1.96	simple rec
8	[2 2 2]	27	100	1.38	27	100	1.38	equal
9	[3 3]	36	139	1.67	39	148	2.00	general rec
10	[5 2]	45	174	1.69	45	174	1.69	equal
11	[11]	66	265	3.00	51	204	1.49	simple rec
12	[3 2 2]	54	221	1.11	54	221	1.11	equal
13	[13]	91	378	3.00	66	277	1.29	simple rec
14	[7 2]	84	349	1.61	72	307	1.11	simple rec
15	[5 3]	90	385	1.40	78	341	0.99	simple rec
16	[2 2 2 2]	81	360	0.77	81	360	0.77	equal
17	[17]	153	664	3.00	105	460	1.11	simple rec
18	[3 3 2]	108	485	0.91	117	512	1.08	general rec
19	[19]	190	837	3.00	129	568	1.05	simple rec
20	[5 2 2]	135	598	0.89	135	598	0.89	equal
21	[7 3]	168	751	1.29	147	662	0.89	simple rec
22	[11 2]	198	879	1.53	153	696	0.77	simple rec
23	[23]	276	1243	3.00	159	734	0.68	simple rec
24	[3 2 2 2]	162	755	0.55	162	755	0.55	equal
25	[5 5]	225	1056	1.20	186	871	0.67	simple rec
26	[13 2]	273	1234	1.51	198	931	0.64	simple rec
27	[3 3 3]	216	1039	0.71	210	995	0.61	simple rec
28	[7 2 2]	252	1155	0.80	216	1029	0.53	simple rec
29	[29]	435	2002	3.00	228	1101	0.52	simple rec
30	[5 3 2]	270	1271	0.68	234	1139	0.45	simple rec
31	[31]	496	2295	3.00	240	1181	0.39	simple rec
32	[2 2 2 2 2]	243	1204	0.31	243	1204	0.31	equal
33	[11 3]	396	1843	1.18	291	1408	0.48	simple rec
34	[17 2]	459	2124	1.48	315	1512	0.50	simple rec
35	[7 5]	420	2011	1.06	339	1620	0.52	simple rec
36	[3 3 2 2]	324	1595	0.38	351	1676	0.48	general rec
37	[37]	703	3294	3.00	375	1792	0.50	simple rec
38	[19 2]	570	2659	1.48	387	1852	0.46	simple rec
39	[13 3]	546	2569	1.15	399	1916	0.42	simple rec
40	[5 2 2 2]	405	1950	0.36	405	1950	0.36	equal
41	[41]	861	4060	3.00	429	2082	0.38	simple rec
42	[7 3 2]	504	2417	0.58	441	2150	0.35	simple rec
43	[43]	946	4473	3.00	453	2222	0.33	simple rec
44	[11 2 2]	594	2809	0.72	459	2260	0.28	simple rec
45	[5 3 3]	540	2659	0.49	471	2340	0.26	simple rec
46	[23 2]	828	3909	1.46	477	2382	0.22	simple rec
47	[47]	1128	5359	3.00	483	2428	0.18	simple rec
48	[3 2 2 2 2]	486	2453	0.13	486	2453	0.13	equal
49	[7 7]	784	3879	0.97	534	2689	0.21	simple rec
50	[5 5 2]	675	3364	0.53	558	2809	0.21	simple rec
51	[17 3]	918	4381	1.12	582	2933	0.21	simple rec
52	[13 2 2]	819	3906	0.69	594	2997	0.19	simple rec
53	[53]	1431	6838	3.00	618	3129	0.19	simple rec
54	[3 3 3 2]	648	3329	0.23	630	3197	0.17	simple rec
55	[11 5]	990	4821	0.94	642	3269	0.15	simple rec
56	[7 2 2 2]	756	3685	0.28	648	3307	0.11	simple rec
57	[19 3]	1140	5467	1.11	672	3455	0.12	simple rec
58	[29 2]	1305	6234	1.45	684	3531	0.11	simple rec
59	[59]	1770	8497	3.00	696	3611	0.09	simple rec
60	[5 3 2 2]	810	4049	0.20	702	3653	0.06	simple rec
61	[61]	1891	9090	3.00	714	3741	0.05	simple rec
62	[31 2]	1488	7129	1.45	720	3787	0.02	simple rec
63	[7 3 3]	1008	4999	0.39	726	3837	0.00	simple rec
64	[2 2 2 2 2 2]	729	3864	-0.03	729	3864	-0.03	equal
65	[13 5]	1365	6676	0.90	825	4276	0.05	simple rec

n	*General Recursive KA*				*Simple Recursive KA*			opt(r)
	distribution	#MUL	#ADD	r	#MUL	#ADD	r	opt(r)
66	[11 3 2]	1188	5789	0.49	873	4484	0.07	simple rec
67	[67]	2278	10989	3.00	921	4696	0.10	simple rec
68	[17 2 2]	1377	6640	0.66	945	4804	0.09	simple rec
69	[23 3]	1656	7999	1.09	993	5024	0.11	simple rec
70	[7 5 2]	1260	6309	0.43	1017	5136	0.10	simple rec
71	[71]	2556	12355	3.00	1041	5252	0.09	simple rec
72	[3 3 2 2 2]	972	5069	0.01	1053	5312	0.07	general rec
73	[73]	2701	13068	3.00	1101	5548	0.09	simple rec
74	[37 2]	2109	10174	1.44	1125	5668	0.08	simple rec
75	[5 5 3]	1350	6925	0.34	1149	5792	0.07	simple rec
76	[19 2 2]	1710	8277	0.65	1161	5856	0.05	simple rec
77	[11 7]	1848	9199	0.84	1185	5988	0.04	simple rec
78	[13 3 2]	1638	8015	0.47	1197	6056	0.03	simple rec
79	[79]	3160	15327	3.00	1209	6128	0.01	simple rec
80	[5 2 2 2 2]	1215	6166	-0.01	1215	6166	-0.01	equal
81	[3 3 3 3]	1296	6871	0.09	1263	6434	0.01	simple rec
82	[41 2]	2583	12504	1.44	1287	6570	0.00	simple rec
83	[83]	3486	16933	3.00	1311	6710	0.00	simple rec
84	[7 3 2 2]	1512	7583	0.13	1323	6782	-0.02	simple rec
85	[17 5]	2295	11286	0.86	1347	6930	-0.02	simple rec
86	[43 2]	2838	13759	1.43	1359	7006	-0.04	simple rec
87	[29 3]	2610	12697	1.07	1371	7086	-0.05	simple rec
88	[11 2 2 2]	1782	8775	0.20	1377	7128	-0.07	simple rec
89	[89]	4005	19492	3.00	1401	7292	-0.07	simple rec
90	[5 3 3 2]	1620	8333	0.06	1413	7376	-0.08	simple rec
91	[13 7]	2548	12699	0.80	1425	7464	-0.09	simple rec
92	[23 2 2]	2484	12091	0.64	1431	7510	-0.11	simple rec
93	[31 3]	2976	14503	1.06	1443	7606	-0.12	simple rec
94	[47 2]	3384	16449	1.43	1449	7656	-0.13	simple rec
95	[19 5]	2850	14041	0.84	1455	7710	-0.15	simple rec
96	[3 2 2 2 2 2]	1458	7739	-0.17	1458	7739	-0.17	equal
97	[97]	4753	23184	3.00	1554	8215	-0.13	simple rec
98	[7 7 2]	2352	12025	0.36	1602	8455	-0.12	simple rec
99	[11 3 3]	2376	11839	0.30	1650	8699	-0.11	simple rec
100	[5 5 2 2]	2025	10488	0.09	1674	8823	-0.12	simple rec
101	[101]	5151	25150	3.00	1722	9075	-0.11	simple rec
102	[17 3 2]	2754	13547	0.44	1746	9203	-0.12	simple rec
103	[103]	5356	26163	3.00	1770	9335	-0.12	simple rec
104	[13 2 2 2]	2457	12130	0.18	1782	9403	-0.13	simple rec
105	[7 5 3]	2520	12895	0.24	1830	9671	-0.12	simple rec
106	[53 2]	4293	20934	1.43	1854	9807	-0.13	simple rec
107	[107]	5778	28249	3.00	1878	9947	-0.13	simple rec
108	[3 3 3 2 2]	1944	10415	-0.11	1890	10019	-0.15	simple rec
109	[109]	5995	29322	3.00	1914	10167	-0.15	simple rec
110	[11 5 2]	2970	14899	0.33	1926	10243	-0.16	simple rec
111	[37 3]	4218	20641	1.05	1938	10323	-0.17	simple rec
112	[7 2 2 2 2]	2268	11499	-0.08	1944	10365	-0.18	simple rec
113	[113]	6441	31528	3.00	1992	10665	-0.17	simple rec
114	[19 3 2]	3420	16853	0.43	2016	10817	-0.18	simple rec
115	[23 5]	4140	20451	0.82	2040	10973	-0.18	simple rec
116	[29 2 2]	3915	19162	0.62	2052	11053	-0.19	simple rec
117	[13 3 3]	3276	16339	0.28	2076	11217	-0.19	simple rec
118	[59 2]	5310	25959	1.42	2088	11301	-0.20	simple rec
119	[17 7]	4284	21379	0.75	2100	11389	-0.21	simple rec
120	[5 3 2 2 2]	2430	12623	-0.13	2106	11435	-0.22	simple rec
121	[11 11]	4356	22155	0.75	2130	11615	-0.22	simple rec
122	[61 2]	5673	27754	1.42	2142	11707	-0.23	simple rec
123	[41 3]	5166	25333	1.05	2154	11803	-0.24	simple rec
124	[31 2 2]	4464	21879	0.62	2160	11853	-0.25	simple rec
125	[5 5 5]	3375	17806	0.20	2172	11957	-0.25	simple rec
126	[7 3 3 2]	3024	15497	-0.01	2178	12011	-0.26	simple rec
127	[127]	8128	39879	3.00	2184	12069	-0.27	simple rec
128	[2 2 2 2 2 2 2]	2187	12100	-0.28	2187	12100	-0.28	equal