Tight upper bounds for polynomial multiplication

DAVIDE D'ANGELLA*
Technische Universität München
Fakultät für Informatik
Boltzmannstr. 3, Garching
Germany
davide.dangella@tum.de

CHIARA VALENTINA SCHIAVO
Università degli Studi di Milano
Dipartimento di Informatica
Via Comelico 39, Milano
Italy
chiara.schiavo@unimi.it

ANDREA VISCONTI
Università degli Studi di Milano
Dipartimento di Informatica
Via Comelico 39, Milano
Italy
andrea.visconti@unimi.it

Abstract: Efficient arithmetic over finite fields has high relevance both in hardware and software implementations. One of the most expensive operation over finite field is the multiplication. To our knowledge, the best known explicit upper bounds for the polynomial multiplication were obtained using the multiplication technique presented in [3]. In this paper, we improve such explicit upper bounds and show how this research allows us to reduce the number of bit operations needed to multiply m-bit polynomials.

Key–Words: Polynomial multiplication, Karatsuba algorithm, optimizations, AND/XOR gates

1 Introduction

Efficient implementation of finite field arithmetic is an important issue in a wide variety of applications, such as cryptography [12, 1, 7, 13], coding theory [2, 5] and digital signal processing [11, 6]. Specifically, binary fields — i.e., finite fields of characteristic 2 — are very attractive both for hardware and software implementations. One of the most critical operation over binary fields is the multiplication between two elements, which consists of an ordinary polynomial multiplication and a modular reduction by an irreducible polynomial. While the modular reduction is relatively inexpensive, the multiplication is a costly operation. For this reason, efficient implementations of polynomial multiplication are desired.

A great deal of effort has been spent on studying the polynomial multiplication problem. The classical method has complexity $\mathcal{O}(n^2)$, while the Karatsuba algorithm [10] improves the complexity to $\mathcal{O}(n^{log_2 3})$. Toom [17] and Cook [9] introduced an algorithm with complexity $\mathcal{O}(n^{1+\epsilon})$, for arbitrary small $\epsilon > 0$. Schönhage and Strassen [16] performed the multiplication using the Fast Fourier Transform (FFT) with complexity $\mathcal{O}(n \log n \log\log n)$. In the last decade, several works [14, 8, 15, 18] have dealt with hardware multiplier and reported explicit upper bounds on the minimum number of bit operations for polynomial multiplication.

As far as we know, the best explicit upper bounds appear in [3]. These bounds are obtained using a technique which employs recursively four multiplication algorithms: refined Karatsuba, five-way recursion, two-level seven-way recursion and schoolbook. However, as hinted in [3], the aforementioned bounds can be further reduced by skipping some redundant computations that show up during the polynomial multiplication.

In this paper, we investigate the possibility to improve the explicit upper bounds described in [3, 4]. The five-way recursion algorithm provides the minimum number of bit operations (AND/XOR gates) needed to multiply m-bit polynomials in more than 90% of bounds listed in [4]. Thus, we focus our attention on it. In particular, we suggest some optimizations for the five-way algorithm that allow to reduce the number of gates needed to multiply two polynomials of degree $m-1$.

In the sequel, we will use the term *straight-line program* (slp) to indicate a sequential program in which the instructions are of the form $x_i = x_j + x_k$ or $x_i = x_j * x_k$ where:

- x_i has not appeared before in the program;
- x_j and x_k are either inputs or have appeared before in the program;
- "+" and "*" denote the XOR and AND bit operations, respectively.

The remainder of the paper is organized as follows. In Section 2 we describe the polynomial multiplication problem, summarizing the technique presented in [3]. The main idea of this paper and the optimizations suggested for improving the Five-way recursion algorithm are presented in Section 3. In Section 4 we

*This work was done as part of a B.Sc. thesis at Cryptography and Coding Laboratory (CLUB), Department of Computer Science, Università degli Studi di Milano.

ISBN: 978-960-474-354-4

illustrate the results obtained. Finally, conclusions are described in Section 5.

2 Polynomial Multiplication

In this section we describe how to obtain explicit upper bounds for polynomial multiplication, summarizing the efficient hybrid approach adopted in [3]. Indeed, the author does not use a specific algorithm to multiply two m-bit polynomials but a family of algorithms by choosing the best at each stage.

2.1 Hybrid approach

Let $\mathcal{A}$ be a family of recursive algorithms used to multiply two m-bit polynomials. Each algorithm $a \in \mathcal{A}$ is composed of arithmetic operations and recursive calls. Let $T()$ be a vector of m elements. Each element of such a vector contains the minimum number of bit operation necessary to multiply two m-bit polynomials. We generate vector $T()$ as follows:

- for $m = 1$, set $T(m) = 1$;
- for each $m = 2, 3, \ldots, 1000$ and each algorithm $a \in \mathcal{A}$ compute the cost $T_a(m)$ required to multiply two m-bit polynomials using algorithm a;
- for each $m = 2, 3, \ldots, 1000$, set $T(m) = min_{a \in \mathcal{A}} \{T_a(m)\}$.

It is important to note that a slight improvement on a given element $T(m)$ will affect all the subsequent steps that recursive recall such $T(m)$.

In [3], the author described the hybrid approach mentioned above using four algorithms: schoolbook recursion, refined Karatsuba, five-way recursion and two-level seven-way recursion. Let us briefly recall these algorithms.

Schoolbook recursion The schoolbook recursion method multiplies two $n+1$ bits polynomials $f(t) = f_0 + \ldots + f_n t^n$ and $g(t) = g_0 + \ldots + g_n t^n$ as follows:

$$\begin{aligned}\{f_0 + \ldots + f_n t^n\}\{g_0 + \ldots + g_n t^n\} = \quad (1)\\ \{f_0 + \ldots + f_{n-1} t^{n-1}\}\{g_0 + \ldots + g_{n-1} t^{n-1}\} +\\ \{f_n g_0 + f_0 g_n\} t^n + \{f_n g_1 + f_1 g_n\} t^{n+1} + \cdots +\\ \{f_n g_{n-1} + f_{n-1} g_n\} t^{2n-1} + f_n g_n t^{2n}\end{aligned}$$

The total cost of schoolbook recursion algorithm [3] is $C(n+1) = C(n) + 4n$.

Refined Karatsuba Given two n-bit polynomials F_0, G_0 and two k-bit polynomials F_1, G_1, with $1 \leq k \leq n$, in [3] the multiplication between the polynomials $F = F_0 + F_1 t^n$ and $G = G_0 + G_1 t^n$ is performed by means of the projective Lagrange interpolation formula. Specifically, the obtained refined Karatsuba identity is as follows:

$$\begin{aligned}\{F_0 + F_1 t^n\}\{G_0 + G_1 t^n\} = \quad (2)\\ (1 - t^n)\{F_0 G_0 - t^n F_1 G_1\} +\\ t^n \{F_0 + F_1\}\{G_0 + G_1\}\end{aligned}$$

The total cost of the refined Karatsuba algorithm [3] is $C(n+k) = 2C(n) + C(k) + 3n + 4k - 3$.

Five-way recursion The five-way recursion algorithm represents a new way to multiply two generic polynomials $F(t) = F_0 + F_1 t^n + F_2 t^{2n}$ and $G(t) = G_0 + G_1 t^n + G_2 t^{2n}$, where F_0, F_1, G_0, G_1 are n-bit polynomials and F_2, G_2 are k-bit polynomials.

Consider $H(x) = H_0 + H_1 x + H_2 x^2 + H_3 x^3 + H_4 x^4$, a generic quartic polynomial over $GF(2)$. By using the Lagrange interpolation formula with the values $H(0)$, $H(1)$, $H(t)$, $H(t+1)$ e $H(\infty)$, in [3] the author obtains the formula:

$$\begin{aligned}H(x) = U(x) + H(\infty)(x^4 + x) + \quad (3)\\ \frac{(U(x) + V(x) + H(\infty)(t^4 + t))(x^2 + x)}{t^2 + t}\end{aligned}$$

where $U(x) = H(0) + (H(0) + H(1))x$, $V(x) = H(t) + (H(t) + H(t+1))(x + t)$, $H(0) = F_0 G_0$, $H(1) = (F_0 + F_1 + F_2)(G_0 + G_1 + G_2)$, $H(t) = \{F_0 + F_1 t + F_2 t^2\}\{G_0 + G_1 t + G_2 t^2\}$, $H(t+1) = \{(F_0 + F_1 + F_2) + F_1 t + F_2 t^2\}\{(G_0 + G_1 + G_2) + G_1 t + G_2 t^2\}$ and $H(\infty) = F_2 G_2$.

The formula (3) produces a new algorithm to multiply the polynomials $F(t)$ and $G(t)$. The cost of the five-way recursion algorithm [3] is

$$C(2n+k) = 2T(n) + T(k) + 24n + 6k - 12 + \begin{cases} 2T(n+2) + 5n, & \text{if } k = n \\ 2T(n+1) + n + 4k, & \text{if } \frac{n}{2} \leq k < n \\ 2T(n+1) + 6k, & \text{if } 1 \leq k < \frac{n}{2} \end{cases}$$

Two-level seven-way recursion The two-level seven way recursion algorithm consists mainly of a recursive application of the refined Karatsuba identity. In particular, applying three times the refined Karatsuba identity and factoring out $(1 - t^n)$, the multiplication of two 3-degree polynomials can be performed as:

$$\begin{aligned}\{F_0 + F_1 t^n + F_2 t^{2n} + F_3 t^{3n}\}\{G_0 + G_1 t^n + G_2 t^{2n} + G_3 t^{3n}\}\\ = (1 - t^{2n})\{(1 - t^n)(F_0 G_0 - t^n F_1 G_1 - t^{2n} F_2 G_2 + t^{3n} F_3 G_3)\\ + t^n (F_0 + F_1)(G_0 + G_1) - t^{3n}(F_2 + F_3)(G_2 + G_3)\}\\ + t^{2n}\{F_0 + F_2 + (F_1 + F_3)t^n\}\{G_0 + G_2 + (G_1 + G_3)t^n\} \quad (4)\end{aligned}$$

The cost of such multiplication [3] is $C(3n+k) = C(2n)+5C(n)+C(k)+19n+8k-8$ if $n/2 \leq k \leq n$.

3 Optimizations

3.1 A toy example

At first glance, one might think that removing the generation of the redundant instructions "a priori" or "a posteriori" have the same effect on the total number of bit operations necessary for multiplying two m-bit polynomials. This is wrong. In fact, vector $T()$ is computed by choosing the best algorithm which minimize the total number of bit operations. The presence of redundant instructions may negatively affect the choice of such algorithm. Therefore, our goal is to try to predict redundant instructions computed in [3] by the hybrid approach. In particular, we identify three cases in which such instructions can be avoided:

- **Zero**: instructions that are equal to zero — i.e., we apply the cancellation property in $GF(2)$;
- **Collision**: instructions that compute the same value in different ways;
- **Unused**: instructions that are used only to compute "zero" and/or "collision" instructions.

To better understand how and why the redundant instructions can be removed from the straight-line program, let us play with a toy example (see Figure 1).

First columns of Figure 1 describes a slp which has not been optimized. Second column stresses the redundant instructions of the straight-line program, while third presents an optimized slp. Let's look more closely at Figure 1.

Instruction $t2$ can be written as $t2 = g0 * t1 = g0*(f0+f1) = (f0*g0)+(f1*g0)$, while $t5$ as $t5 = t3+t4 = (f0*g0)+(f1*g0)$. It is straightforward to observe that $t2$ and $t5$ compute the same value, hence, we found a collision. Variable $t5$ can be removed from the straight-line program.

Similarly, instruction $t9$ can be written as $t9 = t6 + t8 = (t5 + t2) + (t5 + t7) = t2 + t7 = t5 + t7$, while $t8$ is $t8 = t5 + t7$. Hence, instruction $t9$ can be removed too.

Moreover, since $t2$ and $t5$ compute the same value, we can apply the cancellation property in $GF(2)$. Instruction $t6 = t5 + t2$ is equal to zero and can be avoided.

Finally, instructions $t3$ and $t4$ are used only to compute $t5$ — i.e., $t5 = t3+t4$, — but $t5$ has been removed. Therefore $t3$ and $t4$ are "unused instructions" and can be removed too.

3.2 Optimization 1

Our first optimization concerns the sum $H(t)+H(t+1)$ in $V(x)$ (see equation 3) in which some **Zero** instructions are generates. We know that for $k = n$ we have $deg(H(t)) = deg(H(t+1)) = 2n+2$, while for $k < n$ we have $deg(H(t)) = deg(H(t+1)) = 2n$.

However, as hinted in [3], when $k = n$ the coefficients of t^{2n+2} and t^{2n+1} in $H(t)$ are equal to the coefficients of t^{2n+2} and t^{2n+1} in $H(t+1)$. Consequently, the sum $H(t) + H(t+1)$ can be performed with $2n+1$ bit operations.

Moreover, we propose a further optimization for this step. Knowing that $H(x) = H_0 + H_1x + H_2x^2 + H_3x^3 + H_4x^4$, we can derive

$$H(t+1) \equiv H(0) + H(t) + H(1) + H_3\{t + t^2\} \quad (5)$$

and consequently

$$H(t) + H(t+1) \equiv H(0) + H(1) + H_3\{t + t^2\} \quad (6)$$

where $H_3 = F_1G_2 + F_2G_1$. Knowing that $deg(H(0)) = 2n - 2$, $deg(H(1)) = 2n - 2$ and $deg(H_3\{t+t^2\}) = n + k$ we can refine the degree of polynomial $H(t) + H(t+1)$ as follows:

- if $k = n$ then $deg(H(t) + H(t+1)) = 2n$
- if $k = n-1$ then $deg(H(t)+H(t+1)) = 2n-1$
- if $k \leq n-2$ then $deg(H(t)+H(t+1)) = 2n-2$

This optimization reduces the number of bits operations needed to compute $H(t) + H(t+1)$. In particular, for $k = n - 1$ and $k \leq n - 2$ the degree of $H(t) + H(t+1)$ is reduced by 1 and 2 units respectively. This means that we are able to compute the same polynomial with a reduced number of instructions — i.e., a shorter *straight-line program*.

Readers can note that such improvement — i.e., the reduction of 1 or 2 gates to compute $H(t)+H(t+1)$ — is not limited to this small value. In fact, all steps t of the Five-way recursion algorithm that involve such sum, can recursively exploit this optimization.

3.3 Optimization 2

It is possible to avoid the generation of **Zero** instructions in the computation of $U(t) + V(t)$. We know that $deg(U(t)) = 3n - 2$, $deg(V(t)) = 3n$, and $deg(U(t) + V(t)) = 3n$. Consequently, the sum $U(t) + V(t)$ can be implemented with $3n - 1$ XOR gates.

Nevertheless, this number of gates can be further improved by applying optimization 1 to $V(t)$ — recall

```
t1=f0+f1        t1=f0+f1
t2=g0*t1        t2=g0*t1
t3=f0*g0        t3=f0*g0 -> UNUSED
t4=g0*f1        t4=g0*f1 -> UNUSED                    t1=f0+f1
t5=t3+t4        t5=t3+t4 -> COLLISION (t5 and t2)     t2=g0*t1
t6=t5+t2        t6=t5+t2 -> ZERO                      t7=g1*f1
t7=g1*f1        t7=g1*f1                              t8=t2+t7
t8=t5+t7        t8=t5+t7                              t10=g2*f2
t9=t6+t8        t9=t6+t8 -> COLLISION (t9 and t8)     t11=t8+t10
t10=g2*f2       t10=g2*f2
t11=t9+t10      t11=t9+t10
```

Figure 1: SLP: a toy example

that the sum $H(t)+H(t+1)$ is in $V(t)$. Hence, the new degree of $V(t)$ is:

$$deg(V(t)) = \begin{cases} 3n & \text{if } k = n \\ 3n-1 & \text{if } k = n-1 \\ 3n-2 & \text{if } k \leq n-2 \end{cases} \quad (7)$$

It is straightforward to note that the result presented in equation (7) also affects the degree of $U(t)+V(t)$:

$$deg(U(t)+V(t)) = \begin{cases} 3n & \text{if } k = n \\ 3n-1 & \text{if } k = n-1 \\ 3n-2 & \text{if } k \leq n-2 \end{cases} \quad (8)$$

The bounds of equation 7 can be further optimized. In fact, equation (3) suggests that

$$U(t)+V(t) \equiv H(0)+H(t)+\{H(t)+H(t+1)\}t + \{H(0)+H(1)+H(t)+H(t+1)\}t^n \quad (9)$$

and applying equation (6) to (9) — recall that $H_3\{t+t^2\} \equiv H(t)+H(t+1)+H(0)+H(1)$, — we obtain:

$$U(t)+V(t) \equiv H(0)+H(t)+ \{H(t)+H(t+1)\}t + H_3\{t+t^2\}t^n \quad (10)$$

The degree of the polynomial $U(t)+V(t)$ is defined as

$$deg(U(t)+V(t)) = max\{\underbrace{deg(H_3\{t+t^2\}t^n)}_{2n+k}, \underbrace{deg(H(0))}_{2n\text{-}2}, \underbrace{deg(H(t))}_{\begin{cases} 2n+2 & \text{if } k=n \\ 2n & \text{if } k \leq n-1 \end{cases}}, \underbrace{deg(\{H(t)+H(t+1)\}t)}_{\begin{cases} 2n+1 & \text{if } k=n \\ 2n & \text{if } k=n-1 \\ 2n-1 & \text{if } k \leq n-2 \end{cases}}\}$$

For $k = n$, we have $deg(U(t)+V(t)) = max\{2n+k, 2n-2, 2n+2, 2n+1\} = 2n+k$, while for $k \leq n-1$, we have $deg(U(t)+V(t)) = max\{2n+k, 2n-2, 2n, 2n-1\} = 2n+k$. Therefore, $\forall n > 2$, $deg(U(t)+V(t)) = 2n+k$. This means that when $k < n-2$, $n > 2$, we can skip the computation of the last $3n-2-(2n+k) = n-k-2$ coefficients, that are equal to zero. Consequently, the total number of gates needed to compute $U(t)+V(t)$ is reduced by $n-k-2$ XOR gates.

3.4 Optimization 3

This optimization tries to avoid **Collision** instructions during the computation of $W(t) = U(t)+V(t)+H(\infty)\{t+t^4\}$. Considering the degrees of $U(t)$ and $V(t)$, we observe that $deg(W(t)) = 2n+k$ for $n \geq 4$. Let $W'(t) = \frac{W(t)}{t(t+1)}$. The coefficients $W'_0 \dots W'_{2n+k-2}$ of $W'(t)$ can be computed as:

$$\begin{cases} W'_0 = W_1 \\ W'_i = W'_{i-1} + W_{i+1} \quad i = 1 \dots 2n+k-2 \end{cases} \quad (11)$$

The computational cost of equation (11) is $2n+k-2$, while the one computed in [3] is $3n-2$. This means that we are avoiding $n-k$ instructions.

Moreover, in (11) it is possible to skip one more instruction. In fact, the equation $W(t) = U(t)+V(t)+H(\infty)\{t+t^4\}$ can be written as:

$$W(t) \equiv t\{t+1\}\{H_2 + H_3\{1+t^n\}\} \quad (12)$$

where $H_2 \equiv F_0G_2 + F_1G_1 + F_2G_0$ and $H_3 \equiv F_2G_1 + F_1G_2$.

Now consider the following theorem.

Theorem 1 *Let $P(t)$ be a n-degree polynomial with coefficients $P_0 \dots P_n$. Let $Q(t) = \{t+1\}P(t)$ be a polynomial with coefficients $Q_0 \dots Q_{n+1}$. Then $\sum_{i=0}^{h} Q_i \equiv \sum_{i=h+1}^{n+1} Q_i \equiv P_h \pmod 2$ $\forall h \in \mathbb{N} \mid 0 \leq h \leq n$.*

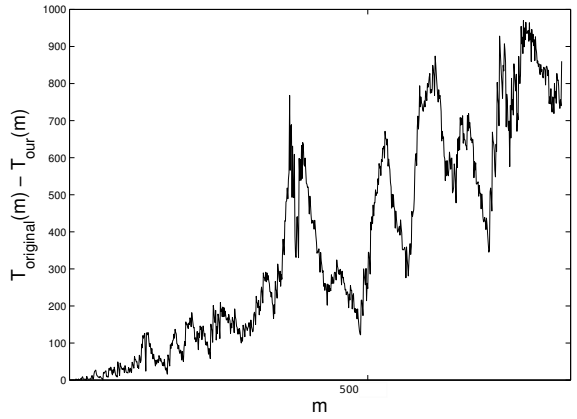

Figure 2: Number of instructions avoided

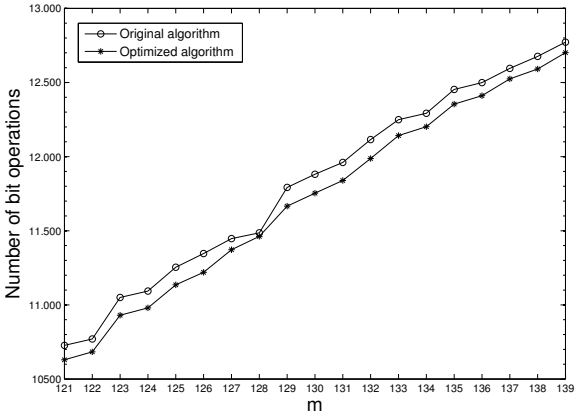

Figure 3: Improvements on [4]'s results for $m = 121, \ldots, 139$

From equation (12) we know that $W(t)$ is multiple of $t\{t+1\}$, hence we can apply Theorem 1:

$$W'_{2n+k-2} \equiv W_1 + \ldots + W_{2n+k-1} \equiv W_{2n+k} \qquad (13)$$

Since W'_{2n+k-2} is equal to W_{2n+k}, and W_{2n+k} is known, we have found a **Collision**. In order to avoid the computation described in (13), equation (11) can be modified as follows:

$$\begin{cases} W'_0 = W_1 \\ W'_i = W'_{i-1} + W_{i+1} \quad i = 1 \ldots 2n+k-3 \\ W'_{2n+k-2} = W_{2n+k} \end{cases} \qquad (14)$$

3.5 Optimization 4

Comparing equations (14) and (11), we can note that the coefficient W_{2n+k-1} is never used, hence we do not computed it. In addition, we can skip all instructions involved in this computation and never used elsewhere (**Unused** instructions).

From section 3.4, we know that for $n \geq 4$ the coefficient W_{2n+k-1} is determined only by $U(t) + V(t)$, — i.e., $W_{2n+k-1} = U_{2n+k-1} + V_{2n+k-1}$.

Let $U_0 \ldots U_{3n-2}$ be the coefficients of $U(t)$ and $V_0 \ldots V_{deg(V(t))}$ be the coefficients of $V(t)$. For $k = n$, we have $W_{2n+k-1} = U_{3n-1} + V_{3n-1}$. Since $deg(U(t)) \leq 3n-2$, the coefficient U_{3n-1} will be always equal to zero. Therefore, we can skip all the instructions involved for computing U_{3n-1}.

3.6 Optimization 5

This optimization concerns the generation of a **Zero** instruction in the computation of $W(t) = U(t) + V(t) + H(\infty)\{t + t^4\}$. Consider the computation of the coefficient of the constant term of $W(t)$. Such coefficient is determined by

$$\begin{aligned} W_0 &\equiv U_0 + V_0 \equiv \{H(0)\}_0 + \{H(t)\}_0 \qquad (15) \\ &\equiv \{F_0 G_0\}_0 + \{F_0 G_0\}_0 \equiv 0 \end{aligned}$$

where $\{P(t)\}_0$ is the coefficient of the term of degree zero of $P(t)$. Since $\{W(t)\}_0$ is determined by $\{H(0)\}_0 + \{H(t)\}_0$ and this sum is equal to zero, this computation can be skipped.

4 Results and Discussions

Applying the optimizations described in Section 3, we can compute a new cost function for the Five-way recursion algorithm:

$$\begin{aligned} M(2n+k) = 2M(n) + M(k) + 21n + 7k - 14 \\ + \begin{cases} 2M(n+2) + 7n, & \text{if } k = n \\ 2M(n+1) + 7n - 9, & \text{if } k = n-1 \\ 2M(n+1) + 7k - 3, & \text{if } k \leq n-2 \end{cases} \end{aligned} \qquad (16)$$

In particular, for $n \geq 4$ the number of instructions avoided is:

- 2, when $k = n$
- 8, when $k = n - 1$
- $4n - 4k + 5$, when $\frac{n}{2} \leq k \leq n - 2$
- $3n - 2k + 5$, when $1 \leq k < \frac{n}{2}$

We point out that additional redundant instructions can show up during the computation. The main

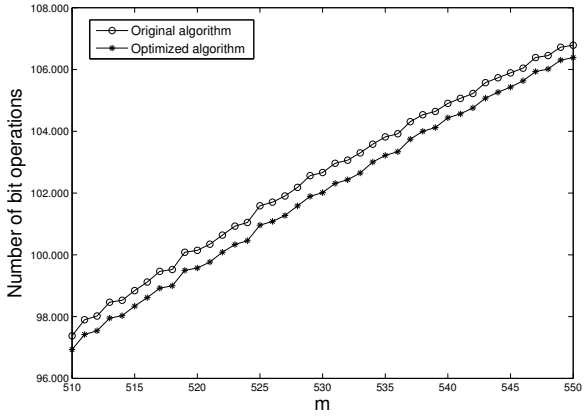

Figure 4: Improvements for $m = 510, \ldots, 550$

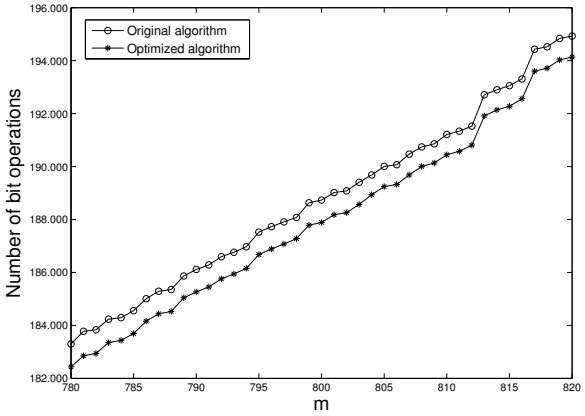

Figure 5: Improvements for $m = 780, \ldots, 820$

reason of this unwanted behavior lies in the recursive approach adopted in [3]. Indeed, a particular instruction x_i of the straight-line program involved in the computation of $T(m)$ can be computed as $x_i = x_j + x_k$ by algorithm a_1 — e.g., $T_{a_1}(m^{'}), m^{'} < m$ — and as $x_h = x_p + x_q = x_i$ by algorithm a_2 — e.g., $T_{a_2}(m^{''}), m^{'} < m^{''} < m$. Unfortunately, we are not able to predict the algorithm that will be chosen in advance, hence instructions x_h cannot be avoided a priori.

These unwanted set of instructions (a very small set) have to be deleted a posteriori from the *straight-line program*.

To evaluate the optimizations suggested in this paper, we develop an automatic tool for computing the polynomial multiplication. Following the approach adopted in [3], we measure the algorithm cost by counting the number of bit operations (XOR/AND gates) required. We tested our algorithm by multiplying m-bit polynomials, $m = 1, \ldots, 1000$, and compared the results obtained with those published in [4]. Figures 2, 3, 4 and 5 summarize these results.

We indicate with $T_{our}(m)$ and $T_{original}(m)$ the cost of algorithm [3] respectively with and without our optimizations. Figure 2 shows the number of instructions avoided using our algorithm — i.e., $T_{original}(m) - T_{our}(m)$, — while Figures 3, 4 and 5 show the improvements of the optimizations suggested for $m = 121, \ldots, 139$, $m = 510, \ldots, 550$, and $m = 780, \ldots, 810$ respectively. These values of m are not specific but have been randomly chosen. Figures 3–5 can help the reader to visualize how the optimizations suggested in this paper positively affect the number of operations required to multiply two m-bit polynomials.

Algorithm	Rate
Schoolbook recursion	1.6%
Refined Karatsuba	2.9%
Five-way recursion	92.7%
Two-level seven-way	2.8%

Table 1: Rate of use of the algorithms

5 Conclusions

In this paper, we dealt with the well-studied polynomial multiplication problem. Our research focused on improving the explicit upper bounds presented in [3] and [4]. Such bounds have been computed by means of a recursive approach used to multiply two m-bit polynomials. In particular, we presented some optimizations for the Five-way recursion algorithm which appears to be the most cost-effective solution. Indeeed, our experiments show that such algorithm provides the best solution over 92% of cases conducted (see Table 1). Testing results, with and without our optimizations, shows that new upper bounds on polynomial multiplication over GF(2) can be found, improving the number of bit operations required to multiply two m-bit polynomials.

References:

[1] G. B. Agnew, T. Beth, R. C. Mullin, and S. A. Vanstone. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6(1):3–13, 1993.

[2] E. R. Berlekamp. *Algebraic coding theory*, volume 111. McGraw-Hill New York, 1968.

[3] D. J. Bernstein. Batch binary edwards. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lec-*

ture Notes in Computer Science, pages 317–336. Springer, 2009.

[4] D. J. Bernstein. High-speed cryptography in characteristic 2, May 2009, http://binary.cr.yp.to/m.html.

[5] R. E. Blahut. *Theory and practice of error control codes*, volume 126. Addison-Wesley Reading Massachusetts, 1983.

[6] R. E. Blahut. *Fast algorithms for digital signal processing*. Addison-Wesley Longman Publishing Co., Inc., 1985.

[7] I. F. Blake, G. Seroussi, and N. Smart. *Elliptic curves in cryptography*, volume 265. Cambridge University Press, 1999.

[8] N. S. Chang, C. H. Kim, Y.-H. Park, and J. Lim. A non-redundant and efficient architecture for Karatsuba-Ofman algorithm. In *Information Security, 8th International Conference, ISC 2005, Singapore*, pages 288–299. Springer, 2005.

[9] S. A. Cook. On the minimum computation time of functions. *PhD thesis, Harvard University*, 1966.

[10] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, pages 595–596, 1963.

[11] J. H. McClellen and C. M. Rader. *Number theory in digital signal processing*. Prentice Hall Professional Technical Reference, 1979.

[12] R. J. McEliece. *Finite fields for computer scientists and engineers*, volume 23. Kluwer Academic Publishers Boston, 1987.

[13] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1997.

[14] S. Peter and P. Langendorfer. An efficient polynomial multiplier in $GF(2^m)$ and its application to ECC designs. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6. IEEE, 2007.

[15] F. Rodrıguez-Henrıquez and Ç. Koç. On fully parallel Karatsuba multipliers for $GF(2^m)$. In *International Conference on Computer Science and Technology (CST 2003), Cancun, Mexico*, pages 405–410, 2003.

[16] D. D. A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7(3-4):281–292, 1971.

[17] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics Doklady*, volume 3, pages 714–716, 1963.

[18] J. von zur Gathen and J. Shokrollahi. Fast arithmetic for polynomials over F_2 in hardware. In *Information Theory Workshop, 2006. ITW'06 Punta del Este. IEEE*, pages 107–111. IEEE, 2006.