Notes on Low Degree Toom-Cook Multiplication with Small Characteristic

Marco Bodrato

Centro Interdipartimentle "Vito Volterra" – Università di Roma Tor Vergata
Via Columbia 2 – 00133 Rome, Italy

Abstract. The use of Toom-Cook sub-quadratic polynomial multiplication was recently shown to be possible also when the coefficient field does not have elements enough, particularly for $\mathbb{F}_2[x]$. This paper focus on how Toom's strategies can be adapted to polynomials on non-binary small fields. In particular we describe the Toom-3 algorithm for $\mathbb{F}_p[x]$ with $p \in \{3, 5, 7\}$, and some other unbalanced algorithms.
Algorithms are described with full details, not only the asymptotic complexity is given, but the exact sequence of operations for possibly optimal implementations.
Algorithms given here for $\mathbb{F}_p[x]$ perfectly works, as well, for $\mathbb{F}_{p^n}[x]$, and may be useful for computations inside $\mathbb{F}_{p^n}$ for large enough n.
Moreover some connections with FFT-based multiplication are found, slightly improving the worst cases of FFT.

Keywords: Polynomial multiplication, finite fields, Toom-Cook, Karatsuba, optimal, convolution, squaring.

1 Introduction

In a previous paper [Bod07] the author faced the problem of writing fast multiplication algorithms, using Toom-Cook splitting strategies [Too63a,Coo66], for polynomials on a field with very few elements: $\mathbb{F}_2[x]$. A task that some authors believe impossible, because "there are not enough evaluation points".

The main idea comes from the fact that Toom-Cook algorithms are efficient only for quite big polynomials and should be used recursively. That's why we rewrite the operands as polynomials in $\mathbb{F}_p[x][Y]$ with a small degree in Y. After this rewriting we can choose elements to evaluate at; not only in the field $\mathbb{F}_p$, but in the bigger ring $\mathbb{F}_p[x]$.

In this work we will focus on fields with characteristic 3, 5 and 7, because binary fields already got enough attention and for bigger primes we can usually adapt algorithms optimised for $\mathbb{Z}[x]$.

2 Toom-Cook Algorithm for Polynomials

We start recalling Toom's algorithm, as generalised in [Bod07]. Starting from two polynomials $u, v \in \mathbb{R}[x]$, on some integral domain $\mathbb{R}$, we want to compute the product $\mathbb{R}[x] \ni w = u \cdot v$. The whole algorithm can be described in five steps.

Splitting : Choose some base $Y = x^b$, and represent u and v by means of two polynomials $\mathfrak{u}(y, z) = \sum_{i=0}^{n-1} u_i z^{n-1-i} y^i$, $\mathfrak{v}(y, z) = \sum_{i=0}^{m-1} v_i z^{m-1-i} y^i$, both homogeneous, with respectively n and m coefficients and degrees $\deg(\mathfrak{u}) = n - 1, \deg(\mathfrak{v}) = m - 1$. Such that $\mathfrak{u}(x^b, 1) = u, \mathfrak{v}(x^b, 1) = v$. The coefficients $u_i, v_i \in \mathbb{R}[x]$ are themselves polynomials and can be chosen to have degrees $\forall\, i, \deg(u_i) < b, \deg(v_i) < b$.

Traditionally the Toom-n algorithm requires balanced operands so that $m = n$, but we can easily generalise to unbalanced ones. We assume commutativity, hence we also assume $n \geq m > 1$.

Evaluation : We want to compute $\mathfrak{w} = \mathfrak{u} \cdot \mathfrak{v}$ whose degree is $d = n + m - 2$, so we need $d + 1 = n + m - 1$ evaluation points $P_d = \{(\alpha_0, \beta_0), \ldots, (\alpha_d, \beta_d)\}$ where $\alpha_i, \beta_i \in \mathbb{R}[x]$ can be polynomials. We define $c = \max_i\{\deg(\alpha_i), \deg(\beta_i)\}$.
The evaluation of a single polynomial (for example $\mathfrak{u}$) on the points (α_i, β_i), can be computed with a matrix by vector multiplication. The matrix $E_{d,n}$ is a $(d + 1) \times n$ Vandermonde-like matrix. $\overline{\mathfrak{u}(\alpha, \beta)} = E_{d,n}\overline{u} \Longrightarrow$

$$\begin{pmatrix} \mathfrak{u}(\alpha_0, \beta_0) \\ \mathfrak{u}(\alpha_1, \beta_1) \\ \vdots \\ \mathfrak{u}(\alpha_d, \beta_d) \end{pmatrix} = \begin{pmatrix} \beta_0^{n-1} & \alpha_0 \cdot \beta_0^{n-2} & \cdots & \alpha_0^{n-2} \cdot \beta_0 & \alpha_1^{n-1} \\ \beta_1^{n-1} & \alpha_1 \cdot \beta_1^{n-2} & \cdots & \alpha_1^{n-2} \cdot \beta_1 & \alpha_2^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ \beta_d^{n-1} & \alpha_d \cdot \beta_d^{n-2} & \cdots & \alpha_d^{n-2} \cdot \beta_d & \alpha_d^{n-1} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} \tag{1}$$

Recursive multiplication : We compute $\forall\, i, \mathfrak{w}(\alpha_i, \beta_i) = \mathfrak{u}(\alpha_i, \beta_i) \cdot \mathfrak{v}(\alpha_i, \beta_i)$, with $d + 1$ multiplications of polynomials whose degrees are comparable to that of $Y = x^b$. We have $\deg(\mathfrak{u}(\alpha_i, \beta_i)) \leq c(n - 1) + b$, $\deg(\mathfrak{v}(\alpha_i, \beta_i)) \leq c(m - 1) + b$, and the results $\deg(\mathfrak{w}(\alpha_i, \beta_i)) \leq c(n + m - 2) + 2b = cd + 2b$. We note that c, d, m, n are fixed numbers for a chosen implementation, b instead will grow as the operands grow.

Interpolation : This step depends only on the expected degree of the result d, and on the $d+1$ chosen points (α_i, β_i), no more on n and m separately. We now need the coefficients of the polynomial $\mathfrak{w}(y, z) = \sum_{i=0}^{d} w_i z^{d-i} y^i$. We know the values of $\mathfrak{w}$ evaluated at $d+1$ points, so we face a classical interpolation problem. We need to apply the inverse of A_d, a $(d+1) \times (d+1)$ Vandermonde-like matrix. $\overline{\mathfrak{w}(\alpha, \beta)} = A_d\overline{w} \Longrightarrow$

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} \beta_0^d & \alpha_0 \cdot \beta_0^{d-1} & \cdots & \alpha_0^{d-1} \cdot \beta_0 & \alpha_0^d \\ \beta_1^d & \alpha_1 \cdot \beta_1^{d-1} & \cdots & \alpha_1^{d-1} \cdot \beta_1 & \alpha_1^d \\ \vdots & \vdots & & \vdots & \vdots \\ \beta_d^d & \alpha_d \cdot \beta_d^{d-1} & \cdots & \alpha_k^{d-1} \cdot \beta_d & \alpha_d^d \end{pmatrix}^{-1} \begin{pmatrix} \mathfrak{w}(\alpha_0, \beta_0) \\ \mathfrak{w}(\alpha_1, \beta_1) \\ \vdots \\ \mathfrak{w}(\alpha_d, \beta_d) \end{pmatrix} \tag{2}$$

Recomposition : The desired result can be simply computed with one more evaluation: $w = \mathfrak{w}(x^b, 1)$. This step requires at most d *shifts* and sums.

The two critical phases are **evaluation** and **interpolation**. As stated by formulas (1) and (2), both require a matrix by vector multiplication. This two phases can require many sums and subtractions, shifts, and even small multiplications or exact divisions (interpolation only) by small elements in $\mathbb{R}[x]$. In this paper we will give explicit Evaluation Sequences of operations (called ES from now on) as well as Interpolation Sequences (IS) for described Toom algorithms paying much attention in giving sequences with the lowest possible computational cost.

3 Searching for Optimality

The two previous papers [BZ07,Bod07] described techniques for computer aided search of optimal Toom-n algorithms. Those techniques are valid for $\mathbb{Z}, \mathbb{Z}[x]$ and $\mathbb{F}_2[x]$, with very small adaptations. Here we will not explain them from start, but only highlight differences to take care of for different polynomial rings.

In both papers we basically assume we have the four operations: addition and subtraction, together with multiplication and exact division both by *small* fixed[1] constants. We also silently assume that the costs of this operations are linear with respect to operand length; this assumption is valid for any $\mathbb{F}_{p^n}[x]$. For readers not used with exact division, detailed explanation can be found in §A. Moreover we assumed some relations between costs of the various operations or some combinations, and this relations can be very different for different characteristics.

3.1 Linear Combinations

The basic operation we really need is linear combination $l_i \leftarrow \pm c_j \cdot l_j/d_j \pm c_k \cdot l_k/d_k$. We assume it is possible, without temporary variables, for any $c_j, d_j, c_k, d_k \in \mathbb{R}[x]$ small constants, even if $i = j$ or $i = k$. The cost of various types of linear combinations are carefully classified in [BZ07], but here we will simply compute them by adding up the cost of single operations, converted to bit-shift whenever possible or skipped when the coefficient is trivial.

3.2 Cost of Basic Operations

We keep on assuming that addition and subtraction do cost the same, and we will call this cost `add`.

We assumed exact division by a small constant to cost always the same regardless of the constant, but this is not true any more, because if the constant is an invertible element $\delta \in \mathbb{F}_{p^n}$ of the field, than the division reduce to a multiplication times the inverse $\delta^{-1} \in \mathbb{F}_{p^n}$, which should cost far less than the exact division by a generic element of $\mathbb{F}_{p^n}[x]$. Vice-versa in $\mathbb{Z}[x]$ we could count on a cost-less multiplication (or division) by the invertible element -1: now we should no more count on a simple sign-changing. Depending on representation, multiplication by -1 can have very different cost. That's why we admit operations like $l_1 \leftarrow l_0 - l_1$ whose result equals $-(l_1 - l_0)$.

Doable shifts We also assumed some particular multiplications and divisions could be replaced by fast shifting, by powers of 2 in $\mathbb{Z}[x]$ and by powers of x in $\mathbb{F}_2[x]$. Powers of x should be simple multipliers or divisors for any polynomial ring, the case being very different for powers of 2.

Multiplication by 2 can not cost more than a sum, because we can compute $a \leftarrow 2 \cdot b$ by $a \leftarrow b + b$, and this can be even faster than a generic sum, because we have to read only one operand. Other powers of 2, instead, can not be easily optimised, because of the needed reduction.

Division by 2 can be obtained with a multiplication by $2^{-1} \in \mathbb{F}_p$, but if elements in $\mathbb{F}_p$ are represented in memory as integers in the range $[0..p-1]$ we can also divide even numbers with a simple shift, and odd numbers with a shift followed by a correction: addition of $2^{-1} \in \mathbb{F}_p$. This procedure strategy gives the correct result because $\frac{a}{2} \equiv \frac{a-1}{2} + \frac{1}{2} \equiv (a-1) \gg 1 + 2^{-1} \pmod{p}$. This trick also works with Montgomery's representation [Mon85], with only one difference: the 1 in the above formula is not the neutral element for multiplication, but the residue represented by all zeroes but the least significant bit.

4 The Matrices

Two kind of matrices are involved in any Toom-k algorithm, the square invertible matrix A_d and the two, possibly equal, matrices $E_{d,n}, E_{d,m}$ with the same number $d + 1 = 2k - 1$ of rows, but fewer (respectively. $n \leq d$ and $m \leq d$) columns.

[1] Where *small* means that we can store those constants in a CPU word, but is not as an important attribute as "fixed", meaning *asymptotically small*; thus operations are asymptotically linear.

4.1 Matrices for the Interpolation Sequence

Since the matrices from equation 2 must be invertible, we are interested in the determinant. Which can be computed from the points in P_d.

Theorem 1. *For the Vandermonde-like matrix A_d generated from the $d+1$ points in $P_d = \{(\alpha_0, \beta_0), \ldots, (\alpha_d, \beta_d)\}$, the determinant can be computed with:*

$$\det(A_d) = \prod_{0 \le i < j \le d} (\alpha_i \beta_j - \alpha_j \beta_i)$$

Proof. It can be easily seen that a matrix with two points with $\beta_i = \beta_j = 0$ is not invertible, and the above formula correctly gives 0.

If one point, suppose (α_0, β_0), has $\beta_0 = 0$, the matrix will start with the line $(0, \ldots, 0, \alpha_0^d)$. Computing the determinant starting from this row, we will have $\alpha_i^d \det(\widetilde{A}_d)$ where $\widetilde{A}_d$ is the complementary minor. $\widetilde{A}_d$ is a Vandermonde $d \times d$ matrix for the points α_i/β_i, where the i-th line was multiplied by β_i^d.

Using the classical formula for Vandermonde matrices, we obtain:

$$\det(A_d) = \alpha_0^d \det(\widetilde{A}_d) = \alpha_0^d \prod_{0 < i \le d} \beta_i^d \prod_{0 < i < j \le d} (\alpha_i/\beta_i - \alpha_j/\beta_j) = \prod_{0 \le i < j \le d} (\alpha_i \beta_j - \alpha_j \beta_i)$$

4.2 The Choice of Evaluation Points

The choice of the evaluation points P_d is one of the most important steps to reach an optimal implementation, and completely determines the matrices $A_d, E_{d,2}, \ldots, E_{d,d}$.

We will consider two of them as being automatically chosen $(0,1), (1,0)$, representing respectively 0 and ∞, and immediately giving $w_0 = u_0 \cdot v_0, w_d = u_{n-1} \cdot v_{m-1}$, and the rows $(1, 0, \ldots), (\ldots, 0, 1)$. Another good choice is the point $(1,1)$, and (if characteristic $\neq 2$) $(-1, 1)$. We need an invertible matrix A_d, so if we use any point (α_i, β_i), no other multiple point $(\lambda\alpha_i, \lambda\beta_i)$ can be added, otherwise the factor $(\alpha_i \lambda \beta_i - \lambda \alpha_i \beta_i)$ will nullify the determinant.

Since the dimension of the extra space needed for the *carries* depends on the parameter $c = \max_i\{\deg(\alpha_i), \deg(\beta_i)\}$, we try to keep it as small as possible. In $\mathbb{F}_2[x]$ the use of degree-one polynomials allows 9 possible couples, so that algorithms up to Toom-5 can be analysed. Even for higher characteristic we will restrict to degree one, because we are not seeking higher Toom instances anyway.

In $\mathbb{Z}$ and in $\mathbb{F}_2$ we could prove that any Toom-k with $k > 2$ requires at least one division, because of the determinant value. Working in $\mathbb{F}_{p^n}[x]$ we have many invertible elements, and the determinant of the matrices A_d, even for higher Toom, can be 1 or the inverse of a power of two.

On the other hand, when we use any non-constant polynomial in $\mathbb{F}_{p^n}[x]/\mathbb{F}_{p^n}$, theorem 2 in [Bod07, p.120] can be used to prove the need of a polynomial division.

4.3 Matrices for the Evaluation Sequence

The matrices $E_{d,n}$ are non-square, so we can not compute the determinant. But we can compute the rank.

Theorem 2. *If the points P_d give an invertible A_d, then the rank of any $E_{d,n}$ matrix is n.*

Proof. Since the $E_{d,n}$ are sub-matrices of the matrix A_d, modulo some multiplication of rows by non-zero constants, all the n columns are linearly independent, so that the rank is n.

4.4 Unusual Evaluations

Dealing with general finite fields one could consider some more general evaluation/interpolation matrices.

In $\mathbb{Z}[x]$ one can consider basically two possible evaluations of a polynomial $p(x)$ of degree d: $p(a), a \in \mathbb{Z}$ or $b^d p(\frac{a}{b})$. Thus if we have $p(x) = p_0 + p_1 x + p_2 x^2$ we usually evaluate $p(a) = p_0 + p_1 a + p_2 a^2$ or $b^2 p(1/b) = b^2 p_0 + p_1 b + p_2$.

But in $\mathbb{F}_{p^n}[x]$ one can also consider a general $\beta p(\alpha)$, with the quadratic example above we can obtain also, for example, $ap(a^{-1}) = ap_0 + p_1 + p_2 a^{-1}$; giving a line $[a\ 1\ a^{-1}]$ in the matrix $E_{d,3}$ and a fairly easy evaluation sequence.

For squaring purposes we would better evaluate "both" operands with the same matrix, but for general products we can even mix evaluations, for example:

$$\mathfrak{v}(\alpha,\beta)\mathfrak{u}(\beta^{-1},\alpha^{-1}) = (\alpha\beta)^{-n+1}\mathfrak{w}(\alpha,\beta)$$

or more generally:

$$\forall(\alpha_v,\beta_v,\alpha_u,\beta_u,\gamma,\delta)\in\mathbb{R}^6 \quad \alpha_v\beta_u = \alpha_u\beta_v \Rightarrow \exists\lambda : \gamma\mathfrak{v}(\alpha_v,\beta_v)\cdot\delta\mathfrak{u}(\alpha_u,\beta_u) = \lambda\mathfrak{w}(\alpha_v,\beta_v)$$

This new kind of mixed evaluation does not affect theorem 2, but will change the determinant. Anyway the author did not find any optimal sequence with this kind of asymmetrical evaluations, further investigation can be done in this direction.

From now on we will consider only plain evaluations, so that the algorithms will be easily adapted from product to squaring.

5 Results and Algorithms in Characteristic 3

The algorithms described in the following sections were studied to work in $\mathbb{F}_3[x]$, but can be applied in general for characteristic 3. A recent work [AHM07] suggest that arithmetic in $\mathbb{F}_{3^m}$ works very fast with polynomial basis representation: this fact opens the way to experimentation for multiplication of elements in those fields with our algorithms. We skip Toom-2 because it coincides with the well known Karatsuba [KO62].

To verify algorithms proposed in this section with GP/PARI, some lines defining coefficients in $\mathbb{F}_3$ should be pre-inserted:

```
U0=u0*Mod(1,3) ; U1=u1*Mod(1,3) ; U2=u2*Mod(1,3) ; U3=u3*Mod(1,3)
V0=v0*Mod(1,3) ; V1=v1*Mod(1,3) ; V2=v2*Mod(1,3)
```

5.1 Choosing Evaluation/Interpolation Points

Possible values for α, β up to degree 1 are:$\{0, \pm 1, \pm x, \pm x + 1, \pm x - 1\}$. Contrary to what happen in characteristic 2, the square of a binomial is a trinomial. Even simple evaluation using $x \pm 1$ will require lots of computations; therefor, when possible, it is better to use $\pm x$

5.2 Toom-2.5 in $\mathbb{F}_3[x]$

The Toom-2.5 algorithm can be used to multiply two operands whose size is not the same. In particular, one will be divided in 3 parts, the other in 2 parts.

Toom-2.5 requires four evaluations, so we need four "values". With $\mathbb{F}_2[x]$ we could not just use elements of the field $\mathbb{F}_2$, so we added also the variable x. Here we use the three values we have in $\mathbb{F}_3$ and ∞, named in our homogenized notation $\{(1,0),(1,1),(-1,1),(0,1)\}$. This choice

is possible for all greater characteristic, and the algorithm described in [BZ07] for $\mathbb{Z}[x]$ can be used. The unique small trick possible in $\mathbb{F}_3$ is to avoid the division. Playing with signs we can have just one division by $-2 \equiv 1 \pmod 3$; the code is shown in figure 1.

Matrices are the same for any characteristic $\neq 2$. Swapping lines we can obtain $\det(A_3) = \pm 2$, as needed.

```
U = U2*X^2 + U1*X*Y + U0*Y^2
V = V1*X   + V0*  Y
\\ Evaluation: 5 add; 4 mul (n)
W3 = U0 + U2 ; W0 = V0 + V1
W2 = W3 + U1
W1 = W2 * W0
W3 = W3 - U1 ; W0 = V1 - V0
W2 = W3 * W0
W3 = U2 * V1 ; W0 = U0 * V0
\\ Interpolation: 4 add (2n)
W2 -= W1
W1 -= W2 + W3
W2 -= W0
\\ Recomposition
W  = W3*X^3+ W2*X^2*Y+ W1*X*Y^2 + W0*Y^3
W == U*V
```

$$E_{3,3} = \begin{pmatrix} 1 & 0 & 0 \\ \boxed{\begin{matrix} 1 & 1 & 1 \\ 1 & -1 & 1 \end{matrix}} \\ 0 & 0 & 1 \end{pmatrix};\ E_{3,2} = \begin{pmatrix} 1 & 0 \\ \boxed{\begin{matrix} 1 & 1 \\ -1 & 1 \end{matrix}} \\ 0 & 1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \boxed{\begin{matrix} 1 & 1 & 1 \\ -1 & 1 & -1 \end{matrix}} & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 1. Toom-2.5 in $\mathbb{F}_3$

5.3 Toom-3 in $\mathbb{F}_3[x]$

Toom-3 is by far the best known and widely used variant of Toom-Cook algorithms. But usually only in characteristic 0, for the multiplication in $\mathbb{Z}$ or $\mathbb{Z}[x]$. While writing this paper, no implementations of balanced Toom-3 in $\mathbb{F}_3[x]$ by other authors was found on the net, and no detailed description in the literature. Here we propose one.

Toom-3 has two variants, the balanced one, which is the most interesting, because it can be used recursively; and the unbalanced, good when one operand is about twice as big as the other. The two variants share the same IS but have different evaluation matrices. The balanced version uses twice $E_{4,3}$, while the unbalanced one uses $E_{4,2}$ for the smallest operand and $E_{4,4}$ for the bigger one.

The set of points used is $P_4 = \{(0,1), (1,-1), (1,1), (1,x), (1,0)\}$; code is shown in figure 2.

The IS needs one exact division, by the small constant element $x^3 - x$. This division can be obtained by a shift and a division by $x^2 - 1$, eventually changing the two central lines of interpolation with:

```
W3  = (W0 - W3)/x + W1 + W2*x + W4*x^3
W3 /= (1-x^2)
```

with a new sequence requiring the same number of operations.

Moreover the point $(1, x)$ can be changed to some different $(1, x^w)$, with a suitable w, may be the number of elements of the field stored in each CPU word, so that shift should cost far less. The drawback is that operands get bigger after evaluation, and recursive multiplication has to manage longer values. In that case division will be by $x^{3w} - x^w = x^w(x^{2w} - 1)$, and the algorithm in §A.3 can be useful.

When working on some $\mathbb{F}_{3^n}$, or other extension, an element $a \in \mathbb{F}_{3^n}$ can be used instead of x. In this case, $(a^3 - a)$ needs to be invertible.

$$E_{4,3} = \begin{pmatrix} 1 & 0 & 0 \\ \boxed{\begin{matrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & x & x^2 \end{matrix}} \\ 0 & 0 & 1 \end{pmatrix};\ A_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & x & x^2 & x^3 & x^4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad E_{4,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \boxed{\begin{matrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & x & x^2 & x^3 \end{matrix}} \\ 0 & 0 & 0 & 1 \end{pmatrix};\ E_{4,2} = \begin{pmatrix} 1 & 0 \\ \boxed{\begin{matrix} 1 & -1 \\ 1 & 1 \\ 1 & x \end{matrix}} \\ 0 & 1 \end{pmatrix}$$

```
U = U2*Y^2 + U1*Y + U0
V = V2*Y^2 + V1*Y + V0

\\ Evaluation:10 add,4 shift;5 mul (n)
W0 = U0 + U2 ; W2 = V0 + V2
W3 = W0 - U1 ; W4 = W2 + V1
W0 = W0 + U1 ; W2 = W2 - V1

W1 = W3 * W2 ; W2 = W0 * W4

W0 = U0 +(U1 + U2*x)*x
                W4 = V0 +(V1 + V2*x)*x
W3 = W0 * W4
W0 = U0 * V0 ; W4 = U2 * V2
```

```
U = U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V1*Y   + V0

\\ Evaluation:10 add,4 shift;5 mul
W0 = U0 + U2
W1 = U1 + U3
W3 = W0 - W1 ; W4 = V0 + V1
W0 = W0 + W1 ; W2 = V0 - V1
W1 = W3 * W2 ; W2 = W0 * W4

W0 = U0 +(U1 +(U2 +U3*x)*x)*x
                  W4 = V0 + V1*x
W3 = W0 * W4
W0 = U0 * V0 ; W4 = U3 * V1
```

```
\\ Interpolation: 9 add, 3 shift, 1 Sdiv
W1 -= W2
W2 -= W1 + W0 + W4
W3 -= W0 + W1*x + W2*x^2 + W4*x^4
W3 /=(x^3-x)
W1 -= W3
\\ Recomposition
W  = W4*Y^4+ W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V
```

Fig. 2. Toom-3 in $\mathbb{F}_3[x]$

6 Results and Algorithms in Characteristic 5

The algorithms described in the following sections were studied to work in $\mathbb{F}_5[x]$, but can be applied in general for characteristic 5. Again we skip Toom-2; we could skip Toom-2.5 too, because it works as in $\mathbb{Z}[x]$, but a small consideration is given here to introduce Toom-3.5, which is described in full details. The classical and most important Toom-3 is then given. The fact that for all the elements $x \in \mathbb{F}_5$ we have $x^4 \equiv 1$, gives many cancellation and the shortest sequences both for the 3-way and for the unbalanced Toom-3.5 .

To verify algorithms proposed in this section with GP/PARI, some lines defining coefficients in $\mathbb{F}_5$ should be pre-inserted:

```
U0=u0*Mod(1,5); U1=u1*Mod(1,5); U2=u2*Mod(1,5); U3=u3*Mod(1,5); U4=u4*Mod(1,5)
V0=v0*Mod(1,5); V1=v1*Mod(1,5); V2=v2*Mod(1,5)
```

6.1 Toom-2.5 in $\mathbb{F}_5[x]$, FFT-like

The Toom-2.5 algorithm has been completely eviscerated in §5.2, in $\mathbb{F}_5[x]$ the algorithm described in [BZ07] can be used. The only possible trick, is to play with signs obtaining $\det(A_3) = -2$, so that the division is replaced with a product by $2 = -2^{-1}$.

With different sets of evaluation points, we can even obtain $\det(\widetilde{A}_3) = 1$ and sequences with neither multiplications nor divisions of any kind, but they require a greater number of operation, so we discarded them; after all, a product by two is nothing but one more addition.

6.2 Toom-3 in $\mathbb{F}_5[x]$

With the same choice of points as in $\mathbb{Z}$, the determinant can be 2 or $-2 = 2^{-1}$, so that only one product (or division) by two is needed.

Again we will give the two variants for Toom-3: the balanced and the unbalanced one. In the code shown for the balanced product, we give two different ES, one for each operands: they are equivalent with respect to operation count. The preferred one can be chosen during implementation.

The set of points used is $P_4 = \{(0,1), (1,-1), (1,1), (1,2), (1,0)\}$; code is shown in figure 3

$$E_{4,3} = \begin{pmatrix} 1 & 0 & 0 \\ \boxed{\begin{matrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & -1 \end{matrix}} \\ 0 & 0 & 1 \end{pmatrix};\ A_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & -1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad E_{4,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \boxed{\begin{matrix} 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & -1 & -2 \end{matrix}} \\ 0 & 0 & 0 & 1 \end{pmatrix};\ E_{4,2} = \begin{pmatrix} 1 & 0 \\ \boxed{\begin{matrix} 1 & -1 \\ 1 & 1 \\ 1 & 2 \end{matrix}} \\ 0 & 1 \end{pmatrix}$$

```
U = U2*Y^2 + U1*Y + U0
V = V2*Y^2 + V1*Y + V0

\\ Evaluation:10 add,2 shift;5 mul (n)
W3 = U2 + U0     ; W1 = V2 + V0
                   W4 = W1 + V1
W0 = W3 + U1
W2 = W0 * W4
W3 = W3 - U1     ; W1 = W1 - V1
W0 =(W0 + U2)*2  ; W4 = V0 + V1*2
W0 = W0 - U0     ; W4 = W4 - V2

W1 = W3 * W1     ; W3 = W0 * W4
W4 = U2 * V2     ; W0 = U0 * V0
```

```
U = U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V1*Y   + V0

\\ Evaluation:10 add,1 shift;5 mul
W3 = U2 + U0      ; W4 = V0 + V1
W1 = U3 + U1
W0 = W3 + W1
W2 = W0 * W4
W3 = W3 - W1      ; W1 = V0 - V1
W0 =(U1 - U3)*2   ; W4 = W4 + V1
W0 = W0 + U0 - U2

W1 = W3 * W1      ; W3 = W0 * W4
W4 = U3 * V1      ; W0 = U0 * V0
```

```
\\ Interpolation: 7 add, 2 shift (2n)
W3 -= W1
W1 -= W2 ; W1 *= 2
W2 -= W0 + W4
W3 += W2 * 2
W2 -= W1
W1 -= W3
\\ Recomposition
W  = W4*Y^4+ W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V
```

Fig. 3. Toom-3 in $\mathbb{F}_5[x]$

6.3 Toom-3.5 in $\mathbb{F}_5[x]$

The unbalanced Toom-3.5 in $\mathbb{F}_5[x]$ is interesting for two reasons. First of all because it can be done using only products and divisions by 2, with a sequence very similar to FFT; this relationship

will be further investigated in §8. Moreover it has two versions, the slightly unbalanced (4 parts versus 3) and the strongly one (5 parts versus 2), with quite interesting evaluation sequences: in particular the 5 parts evaluation is only a slight modification of the 4 parts one, because the matrix repeats.

The set of points used is $P_4 = \{(0,1),(1,2),(1,-2),(1,-1),(1,1),(1,0)\}$. The code in figure 4 uses those points and the resulting matrices.

$$E_{5,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \boxed{\begin{matrix} 1 & 2 & -1 & -2 \\ 1 & -2 & -1 & 2 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{matrix}} \\ 0 & 0 & 0 & 1 \end{pmatrix} ; \; E_{5,3} = \begin{pmatrix} 1 & 0 & 0 \\ \boxed{\begin{matrix} 1 & 2 & -1 \\ 1 & -2 & -1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{matrix}} \\ 0 & 0 & 1 \end{pmatrix}$$

```
U = U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V2*Y^2 + V1*Y   + V0
\\ Evaluation: 14 add, 2 shift; 6 mul

W1 = U0 + U2   ; W5 = V0 + V2
W3 = U1 + U3
W2 = W1 - W3   ; W4 = W5 - V1
W1 = W1 + W3   ; W5 = W5 + V1
W3 = W2 * W4   ; W4 = W1 * W5

W2 =(U1 - U3)*2; W0 = V0 + V1*2
W1 = U0 - U2   ; W0 = W0 - V2
W5 = W1 - W2
W2 = W2 + W1
W1 = W2 * W0
                ; W0 = W0 + V1
W2 = W5 * W0
W5 = U3 * V2   ; W0 = U0 * V0
```

$$E_{5,4} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \boxed{\begin{matrix} 1 & 2 & -1 & -2 & 1 \\ 1 & -2 & -1 & 2 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} ; \; E_{5,2} = \begin{pmatrix} 1 & 0 \\ \boxed{\begin{matrix} 1 & 2 \\ 1 & -2 \\ 1 & -1 \\ 1 & 1 \end{matrix}} \\ 0 & 1 \end{pmatrix}$$

```
U = U4*Y^4 + U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V1*Y   + V0
\\ Evaluation: 13 add, 1 shift; 6 mul (n)
W0 = U0 + U4
W1 = W0 + U2
W3 = U1 + U3
W2 = W1 - W3   ; W4 = V0 - V1
W1 = W1 + W3   ; W5 = V0 + V1
W3 = W2 * W4   ; W4 = W1 * W5

W2 =(U1 - U3)*2
W1 = W0 - U2   ; W0 = W5 + V1
W5 = W1 - W2
W2 = W2 + W1
W1 = W2 * W0
                ; W0 = W0 + V1
W2 = W5 * W0
W5 = U4 * V1   ; W0 = U0 * V0
```

```
\\ Interpolation: 10 add, 4 shift (2n)
W4 =(W4 + W3)/2
W3 = W4 - W3
W1 = W1 - W2
W2 = W2 - W1*2
W4 =(W4 + W2)/2
W2 = W4 - W2
W3 =(W3 + W1)/2
W1 = W3 - W1
W1 = W1 - W5
W4 = W4 - W0
\\ Recomposition
W  = W5*Y^5 +W4*Y^4+ W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V
```

$$A_5 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \boxed{\begin{matrix} 1 & 2 & -1 & -2 & 1 & 2 \\ 1 & -2 & -1 & 2 & 1 & -2 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Fig. 4. Toom-3.5 in $\mathbb{F}_5[x]$

7 Results and Algorithms in Characteristic 7

The algorithms described in the following sections were studied to work in $\mathbb{F}_7[x]$, but can be applied in general for characteristic 7. Again we skip Toom-2; and Toom-2.5 too, because they

work as in $\mathbb{Z}[x]$. Only some specification for Toom-3 are given. It would be interesting to analyse even the 4-way and the unbalanced Toom-4.5, where cancellations due to the relation $x^6 \equiv 1$ should show up, but this work is left for future implementations.

To verify algorithms proposed in this section with GP/PARI, some lines defining coefficients in $\mathbb{F}_7$ should be pre-inserted:

```
U0=u0*Mod(1,7) ; U1=u1*Mod(1,7) ; U2=u2*Mod(1,7) ; U3=u3*Mod(1,7)
V0=v0*Mod(1,7) ; V1=v1*Mod(1,7) ; V2=v2*Mod(1,7)
```

7.1 Toom-3 in $\mathbb{F}_7[x]$

With the same choice of points as in $\mathbb{Z}$, the determinant can be ± 2, requiring one shift. Other choices are possible, even with determinant 1, but no shorter ES where found.

We propose here the sequences for balanced and unbalanced operands with the usual set of points: $P_4 = \{(0,1),(1,2),(1,1),(1,-1),(1,0)\}$. The code and the matrices are given in figure 5.

$$E_{4,3} = \begin{pmatrix} 1 & 0 & 0 \\ \hline 1 & 2 & 4 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ \hline 0 & 0 & 1 \end{pmatrix};\ A_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 1 & 2 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
U = U2*Y^2 + U1*Y + U0
V = V2*Y^2 + V1*Y + V0

\\ Evaluation: 10 add, 2 shift; 5 mul (n)

W1 = U2 + U0   ; W3 = V2 + V0
W0 = W1 + U1   ; W4 = W3 + V1
W2 = W0 * W4

W0 =(W0 + U2)*2 ; W4 =(W4 + V2)*2
W0 = W0 - U0    ; W4 = W4 - V0
W1 = W1 - U1    ; W3 = W3 - V1

W3 = W1 * W3    ; W1 = W0 * W4
W4 = U2 * V2    ; W0 = U0 * V0
```

$$E_{4,2} = \begin{pmatrix} 1 & 0 \\ \hline 1 & 2 \\ 1 & 1 \\ 1 & -1 \\ \hline 0 & 1 \end{pmatrix};\ E_{4,4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \hline 1 & 2 & 4 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \hline 0 & 0 & 0 & 1 \end{pmatrix}$$

```
U = U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V1*Y   + V0

\\ Evaluation: 10 add, 1 shift; 5 mul (n)
W1 = U2 + U0
W3 = U3 + U1
W0 = W1 + W3    ; W4 = V0 + V1
W2 = W0 * W4

W1 = W1 - W3    ; W3 = V0 - V1
W0 =(W0 + U2)*2 ; W4 = W4 + V1
W0 = W0 - U0 -U3

W3 = W1 * W3    ; W1 = W0 * W4
W4 = U3 * V1    ; W0 = U0 * V0
```

```
\\ Interpolation: 8 add, 2 shift (2n)
W1 -= W2
W3  =(W2 - W3)/2
W2  = W2 - W0 - W3
W1 -= W2
W2 -= W4
W1 -= W2*2
W3 -= W1
\\ Recomposition
W  = W4*Y^4+ W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V
```

Fig. 5. Toom-3 in $\mathbb{F}_7[x]$

8 Toom and FFT interactions

Toom's ideas for multiplication and the use of FFT for convolution [SS71] are tightly related. Both are based on the "evaluate, multiply, interpolate" paradigm. When we work on finite fields, the difference became even smaller. In the correct ring, Toom-2.5 and FFT-4 coincide, or, more correctly, NTT-4 is one of the possible Toom-2.5 algorithms. The main difference is that to multiply two polynomials in $\mathbb{F}_{p^n}[x]$, Toom uses evaluations on the ring itself, while the FFT approach always maps into some ring extension.

8.1 Toom-3.5 for Any Ring with Imaginary Unit i.

The inversion sequence described for Toom-3.5 in $\mathbb{F}_5[x]$, can be generalised to any characteristic $p \equiv 1 \pmod 4$, or more generally to any integral domain containing $\pm i$, the square roots of $-1 = (\pm i)^2$.

With the six evaluation points $(0, 1, -1, i, -i, \infty)$, we obtain evaluation and interpolation matrices whose central lines requires exactly those operations needed to compute the product modulo $(y^4 - 1)$ via FFT.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i & 1 & i \\ 1 & -i & -1 & i & 1 & -i \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \overset{FFT-4}{\underset{(\mathrm{mod}\ y^4-1)}{\rightsquigarrow}} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The same can be done if we have $\omega^4 = -1, (\omega^2) = \pm i$ and we use *negacyclic* convolution.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & \omega & \omega^2 & \omega^3 & -1 & -\omega \\ 1 & -\omega & \omega^2 & -\omega^3 & -1 & \omega \\ 1 & \omega^3 & -\omega^2 & \omega & -1 & \omega^3 \\ 1 & -\omega^3 & -\omega^2 & -\omega & -1 & -\omega^3 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \overset{FFT-4}{\underset{(\mathrm{mod}\ y^4+1)}{\rightsquigarrow}} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Finally the two 1 (respectively -1) can be easily cleared with two subtractions (additions).

The argument can be reversed, using Toom strategies to handle FFT worst cases. Assume we need to compute the product $c = ab$ where $a, b \in R[x]$. If $\deg(ab) = K - 1$ and K is a good value for FFT, we can compute $c = ab = ab \pmod{x^K \pm 1}$ very fast, using K point-wise multiplication. On the other hand if $\deg(ab) = K + 1$ we have two possible choices: use FFT with a bigger K', or compute $C = ab(\mod x^K \pm 1)$ as above, then $c_0 = a(0)b(0) = a_0 \cdot b_0$ and $c_{K+1} = a(\infty)b(\infty) = a_n \cdot b_m$; finally correct the coefficients and obtain $c_K = c_0 \mp C_0$, $c_1 = C_1 \pm c_{K+1}$, $c_i = C_i$. Thus we need $K + 2$ point-wise multiplication to obtain $K + 2$ coefficients, overhead is negligible, while plain FFT would have required $2K$ or more.

This approach is quite similar to the Granlund's trick described in [GKZ07], but generalises it using polynomials. This allow us to rebuild correct coefficient from both (low and high) ends, and not only from lowest bits. It be used only for the first step, not inside the recursions.

In general the FFT strategy [SS71] can be used to compute a total number of coefficients that is a power of two, thus we can compare Toom-4.5 with FFT-8, and Toom-$(2^k + 1)/2$ with FFT-(2^k). The argument above allows also FFT-(2^k+1) and FFT-(2^k+2), so that Toom-3.5 is an exemplified FFT-(4+2), and the algorithm in §5.2 implements FFT-(2+2), which is not FFT-4.

The generalization in $\mathbb{F}_2[x]$ uses power of three [Sch77], but again che parallelism works comparing Toom-5 with FFT-9, Toom-6 with FFT-(9+2), and so on.

9 Conclusions

We gave several hints on how to search for optimal evaluation and interpolation sequences for Toom-Cook algorithms, also in the event that we need more evaluation points than we have in the base-field.

The 3-way Toom-Cook algorithm was given in full details for all significant characteristics; for bigger ones (eleven and more) the sequences described for integers in [BZ07] can be used.

A final trick to enhance the worst cases of the FFT multiplication strategies was suggested, this too can be applied to every characteristic with a very small ovrhead.

Acknowledgements

The author wants to thank Marica Tarantino for her patience which made this work possible; Alberto Zanoni, Paul Zimmermann, Jörg Arndt and Richard Brent for helpful discussions.

References

AHM07. Omran Ahmadi, Darrel Hankerson, and Alfred Menezes, *Software implementation of arithmetic in $\mathbb{F}_{3^m}$*, in Carlet and Sunar [CS07].

Arn07. Jörg Arndt, *Algorithms for programmers (working title)*, `http://www.jjj.de/fxt/`, June 2007, Draft version.

Bod07. Marco Bodrato, *Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0*, in Carlet and Sunar [CS07], `http://bodrato.it/papers/#WAIFI2007`, pp. 116–133.

Bro07. Christopher W. Brown (ed.), *ISSAC 2007 - International Symposium on Symbolic and Algebraic Computation, Waterloo, Ontario, Canada, July 29-August 1, 2007*, ACM press, July 2007.

BZ07. Marco Bodrato and Alberto Zanoni, *Integer and polynomial multiplication: Towards optimal Toom-Cook matrices*, in Brown [Bro07], `http://bodrato.it/papers/#ISSAC2007`.

Coo66. Stephen A. Cook, *On the minimum computation time of functions*, Ph.D. thesis, Dept. of Mathematics, Harvard University, 1966.

CS07. Claude Carlet and Berk Sunar (eds.), *WAIFI 2007 - International Workshop on the Arithmetic of Finite fields, Madrid, Spain, June 21-22, 2007*, Lecture Notes in Computer Science, vol. 4547, Springer, June 2007.

GKZ07. Pierrick Gaudry, Alexander Kruppa, and Paul Zimmermann, *A GMP-based implementation of Schnhage-Strassen's large integer multiplication algorithm*, in Brown [Bro07].

Jeb93. Tudor Jebelean, *An algorithm for exact division*, Journal of Symbolic Computation **15** (1993), 169–180.

KO62. Анатолии Алешеевич Карацуба анд Юри Офман, *Умножение многозначных чисел на автоматах*, Доклады Академии Наук СССР **145** (1962), no. 2, 293–294, english translation in [KO63].

KO63. Anatoly Alexeevich Karatsuba and Yuri Ofman, *Multiplication of multidigit numbers on automata*, Soviet Physics Doklady **7** (1963), 595–596.

Mon85. Peter L. Montgomery, *Modular multiplication without trial division*, Mathematics of Computation **44** (1985), no. 170, 519–521.

SS71. Arnold Schönhage, Volker Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281–292.

Sch77. Arnold Schönhage, *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2*, Acta Informatica **7** (1977), 395–398.

Too63a. Андре уи Леонович Тоом, *О сложности схемы из функциональных элементов, реализирующе уи умножение целых чисел*, Доклады Академии Наук СССР **150** (1963), no. 3, 496–498 (russian), english translation in [Too63b].

Too63b. Andrei Leonovich Toom, *The complexity of a scheme of functional elements realizing the multiplication of integers*, Soviet Mathematics Doklady **3** (1963), 714–716.

ZQ03. Paul Zimmermann and Michael Quercia, *irred-ntl source code*, 2003, `http://www.loria.fr/~zimmerma/irred/`.

A Exact Division

The cost for exact division by a fixed element of the ring field is linear on degree of the dividend. The first reference the author found for a proof of this fact is the paper by Jebelean [Jeb93]. Code by Quercia [ZQ03] for exact division by $1+x^2$ in $\mathbb{F}_2[x]$ gave the author the ideas for generalisation to polynomial rings. Finally Arndt description of inversion for series [Arn07] gave hints for a clean explanation of the procedure.

A.1 Exact Division by $1 \pm x^n$

The divisor we most frequently have to divide for is $D = 1 \pm x^n$, so in this section we assume we have an element $A \in \mathbb{A}[x]$ of any polynomial ring which is known to be a multiple of our divisor D. Then we know the quotient $Q \in \mathbb{A}[x]$ exist so that $A = Q \cdot D$. We want to compute the element Q.

We can at first compute the degree $d = \deg(A)$ of the polynomial A, this immediately gives us the degree of Q, $\deg(Q) = d - n$.

To compute Q we start computing A_0, then we recourse:

$$\begin{aligned} A_0 &\leftarrow A(1 \mp x^n) = Q(1 \pm x^n)(1 \mp x^n) = Q(1 - x^{2n}) \\ A_1 &\leftarrow A_0(1 + x^{2n}) = Q(1 - x^{2n})(1 + x^{2n}) = Q(1 - x^{4n}) \\ &\dots \\ A_k &\leftarrow A_{k-1}(1 + x^{n \cdot 2^k}) = Q(1 + x^{n \cdot 2^{k+1}}) \end{aligned} \tag{3}$$

until $n \cdot 2^{k+1} > d - n$, then $A_k \equiv Q \pmod{x^{d-n}}$ and we can easily extract Q from the lowest coefficient of A_k.

Since we need the result modulo x^{d-n}, all the computations can be performed modulo that polynomial. Each step requires then a shift and an addition of at most d elements in $\mathbb{A}$. The number of necessary steps is $1 + \lfloor \log_2 \frac{d-n}{n} \rfloor = \mathrm{O}(\log_2 d)$.

Another possible way to clear the factor $1 \pm x^n$ from $A = B(1 \pm x^n)$ is to compute $\widetilde{A}_0 = A(1 \mp x^n + x^{2n}) = Q(1 \pm x^n)(1 \mp x^n + x^{2n}) = Q(1 \pm x^{3n})$, and recursively $\widetilde{A}_k$ until $n \cdot 3^{k+1} > d - n$. Which is usually slower, because one step requires two additions and gives only a factor 3 for the exponent. One or two steps of this computation can have some use anyway, whenever a factor 2 is not enough while a 3 is, or 8 is not and 9 is. As an example:

$$(1 - x^3)^{-1} \equiv (1 + x^3)(1 + x^6)(1 + x^{12})(1 + x^{24}) \equiv (1 + x^3 + x^6)(1 + x^9)(1 + x^{18}) \pmod{x^{32}}$$

in both cases an additional factor, resp. $(1 + x^{48})$ or $(1 + x^{36})$, gives the inverse modulo x^{64}.

A.2 Exact Division with Linear Complexity

Assume we have a representation for the finite field where each CPU word stores w coefficients, and $a_0, \dots, a_n$ are the words so that $A = \sum_{i=0}^{n} a_i x^{iw}$ and $\forall\, i\ ,\ \deg(a_i) < w$. Moreover we assume that the divisor D is invertible modulo x^w. Then we can implement exact division with linear complexity; the algorithm follows.

for $i = 0 \dots n$
$\quad a_i \leftarrow a_i \cdot D^{-1} \pmod{x^w}$
$\quad a_{i+1} \leftarrow a_{i+1} - \frac{a_i \cdot D}{x^w}$

When $D = 1 \pm x^n$, multiplication by the inverse D^{-1} can be optimised as above, and

$$\frac{a_i \cdot D}{x^w} = \frac{a_i}{x^w} \pm \frac{a_i \cdot x^n}{x^w} = \pm \frac{a_i}{x^{w-n}} = \pm a_i \gg (w - n)$$

reduces to one shift operation by $(w-n)$ positions.

The very last step $a_{n+1} \leftarrow a_{n+1} - a_n \gg (w-n)$ should be avoided. Anyway it must not have any effect at all. If division is exact we must have $a_n \gg (w-n) = 0$.

A.3 Exact Division with Same Cost of an Addition

The above algorithm greatly simplifies if $D = 1 \pm x^{kw} \Rightarrow D^{-1} \equiv 1 \pmod{x^w}$; the loop reduces to:

for $i = 0 \dots (n-k)$
| $a_{i+k} \leftarrow a_{i+k} \mp a_i$

This requires 1 word addition for each word. The last k steps of the previous loop should all give zero, they can be used for check or simply skipped, because we started from a multiple A of D. With $(n+1)$ words, only $(n+1-2k)$ steps are necessary; $2k$ less than those needed for an addition, and here we just read/write one operand. The cost of this division can be even less than the cost of an addition.

B Toom-k Complexity

Asymptotic complexity of Toom-k algorithm to multiply two degree-d polynomials has been analysed many times. Here we recall and reorganise the results, with the aim of computing the number of operations tightly, showing the influence of the optimisations in the linear part on the usually implied constant.

We will call $M(d)$ the number of operations needed to multiply two polynomials with d coefficients. All Toom-k algorithms require some linear algebra, for evaluation and interpolation phases; all operations used in those steps are linear in the number of coefficients d, so we will condense the cost of all them in $c_k d$, with a constant c_k depending on the specific Toom-k method.

One step of a multiplication method splitting operands in k parts, and requiring m sub-products and c_k linear operation, will cost

$$M_k(d) = m \cdot M\left(\frac{d}{k}\right) + c_k \cdot d$$

operations. We will use the well known relations

$$\alpha^{\log_b \beta} = \beta^{\log_b \alpha} \quad , \quad \sum_{i=0}^{n} \alpha^i = \frac{\alpha^{n+1}-1}{\alpha-1} \quad ,$$

to compute the cost of r recursions of the same algorithm:

$$\begin{aligned} M_k^r(d) &= m^r \cdot M\left(\frac{d}{k^r}\right) + d \cdot c_k \sum_{i=0}^{r-1} \left(\frac{m}{k}\right)^i = m^r \cdot M\left(\frac{d}{k^r}\right) + d \cdot c_k \frac{(m/k)^r - 1}{m/k - 1} \\ &= m^r \cdot M\left(\tfrac{d}{k^r}\right) + d \cdot c_k \tfrac{k}{m-k}\left(\left(\tfrac{m}{k}\right)^r - 1\right) = m^r \cdot M\left(\tfrac{d}{k^r}\right) + c_k \tfrac{k}{m-k}\tfrac{d \cdot m^r}{k^r} - d \cdot c_k \tfrac{k}{m-k} \\ &= m^r \left(M\left(\frac{d}{k^r}\right) + \frac{d}{k^r} c_k \frac{k}{m-k}\right) - d \cdot c_k \frac{k}{m-k} \quad . \end{aligned}$$

Then we analyse the asymptotic behaviour of Toom-k algorithm. Let $m = 2k-1$, and assume to use all possible recursions $r = \log_k d$, then the in the above formula $m^r = d^{\log_k m}$, $k^r = d$, $k/(m-k) = k/(k-1)$:

$$M_k(d) = d^{\log_k m}\left(M(1) + c_k \frac{k}{k-1}\right) - d \cdot c_k \frac{k}{k-1} = \Theta(d^{\log_k(2k-1)}) \quad .$$

THIS PAGE IS NOT PART OF THE ARTICLE[2]

BibTEX entry

```
@TechReport{Bodrato:CIVV2007,
  author =  {Marco Bodrato},
  title =  {Notes on Low Degree {Toom-Cook} Multiplication with Small Characteristic},
  institution ={Centro "Vito Volterra", Universit\`a di Roma "Tor Vergata"},
  year =  {2007},
  number =  {621},
  pages =  {14},
  month =  {December},
  note =  {\url{http://bodrato.it/papers/\#CIVV2007}},
}
```

[2] Paper edited with Emacs-21 and LaTeX-3.0 on a Debian GNU/Linux box.