Towards Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0

Marco Bodrato

Centro “Vito Volterra” – Università di Roma Tor Vergata – Italy

Submitted to WAIFI on February 14, accepted[**] on March 23, 2007

Abstract. Toom-Cook strategy is a well-known method for building algorithms to efficiently multiply dense univariate polynomials. Efficiency of the algorithm depends on the choice of interpolation points and on the exact sequence of operations for evaluation and interpolation. If carefully tuned, it gives the fastest algorithm for a wide range of inputs.
This work smoothly extends the Toom strategy to polynomial rings, with a focus on $\mathrm{GF}_2[x]$. Moreover a method is proposed to find the faster Toom multiplication algorithm for any given splitting order. New results found with it, for polynomials in characteristic 2, are presented.
A new extension for multivariate polynomials is also introduced; through a new definition of density leading Toom strategy to be efficient.

Keywords: Polynomial multiplication, multivariate, finite fields, GF2x, Toom-Cook, Karatsuba, binary polynomials, squaring, convolution.

1 Introduction

Starting with the works of Karatsuba [9] and Toom [13], who found methods to lower asymptotic complexity for polynomial multiplication from $\mathrm{O}(n^2)$ to $\mathrm{O}(n^{1+\epsilon})$ with $0 < \epsilon < 1$, many efforts have been done in finding optimised implementations in arithmetic software packages [5,6,12].

The family of so-called Toom-Cook methods is an infinite set of algorithms. Each of them requires polynomial evaluation of the two operands and a polynomial interpolation problem, with base points not specified *a priori*, giving rise to many possible Toom-k algorithms, even for a fixed size of the operands.

Moreover, to implement one of them, we will need a sequence of many basic operations, which typically are additions and subtractions of arbitrary long operands, multiplication and exact division of a long operand by a *small* one, optimised by bitshifts where possible.

The exact sequence is important because it determines the real efficiency of the algorithm. It is well known [10] that the recursive application of a single Toom-k algorithm to multiply two polynomials of degree n gives an asymptotic complexity of $\mathrm{O}(n^{log_k(2k-1)})$. There is even the well known Schönhage-Strassen

[**] To appear in C. Carlet and B. Sunar (editors), *WAIFI 2007 proceedings*, Madrid, España, June 21-22, 2007; pages 116–133, LNCS 4547. © Springer-Verlag

method [14,15], whose complexity is asymptotically better than any Toom-k: $O(n \log n \log\log n)$. But the O-notation hides a constant factor which is very important in practice.

All the advanced software libraries actually implement more than one method because the asymptotically better ones are not optimal for small operand sizess. So there can be a wide range of operand sizes where Toom-Cook methods can be the preferred ones. The widely known GMP library [5] uses Toom-2 from around 250 decimal digits, then Toom-3, and finally uses FFT based multiplication with more than 35,000 digits. Hence the interest for improvement in Toom-k.

On the multivariate side, the problem is much more complex. Even if the combination of Kronecker's trick [11] with FFT multiplication can give asymptotically fast methods, the overhead is often too big to have algorithms useful in practice. The constraint for the polynomials to be dense is most of the time false, for real world multivariate problems. A more flexible definition for density can help.

1.1 Representation of $\mathbf{GF_2}[x]$ and Notation

All the algorithms in this paper work smoothly with elements of $\mathrm{GF}_2[x]$ stored in compact dense binary form, where each bit represents a coefficient and any degree 7 polynomial fits in one byte.

For compactness and simpler reading, we will sometimes use hexadecimal notation. Every hexadecimal number h corresponds to the element $p \in \mathrm{GF}_2[x]$ such that $p(2) = h(over\mathbb{Z})$. For example $p \in \mathrm{GF}_2[x] \leftrightarrow \text{hex}, 1 \leftrightarrow 1, x \leftrightarrow 2, x+1 \leftrightarrow 3, \ldots, x^3+x^2+x+1 \leftrightarrow \text{F}, \ldots, x^8+x^7+x^6 \leftrightarrow 1\text{C0}, \ldots$.

We will also use the symbols $\ll$ and $\gg$ for bit-shifts. Meaning multiplication and division by power of x, in $\mathrm{GF}_2[x]$, or by power of 2 in $\mathbb{Z}[x]$.

2 Toom-Cook Algorithm for Polynomials, Revisited

A general description of the Toom algorithm follows. 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 degree $\forall i, \deg(u_i) < b, \deg(v_i) < b$.

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. $\overline{\mathfrak{u}(\alpha, \beta)} = E_{d,n}\overline{u} \Longrightarrow$

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

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 degree is paragonable 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. $\overline{\mathfrak{w}(\alpha, \beta)} = A_d\overline{w} \Longrightarrow$

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

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 $\mathbb{R}[x]$. The goal of this paper is to find some *optimal* Evaluation Sequences of operations (called ES from now on) as well as Interpolation Sequences (IS), leading to optimal algorithms.

2.1 References on Collected Ideas

After the first proposals [13,4], many small improvements where introduced for the Toom-Cook splitting schemes. Winograd [17] proposed ∞ and fractions for the evaluation points; same results are obtained here with homogenisation. Zimmermann and Quercia [18] proposed to evaluate also on positive and negative powers of $x \in \mathrm{GF}_2[x]$; this idea is extended here using any coprime couple $\alpha_i, \beta_i \in \mathbb{R}[x]$ in the polynomial ring. Bodrato and Zanoni [2], underlined the need to consider unbalanced operands; this idea was inherited by this paper.

3 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.

3.1 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:*

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

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:

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

3.2 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 $(1, 0, \ldots), (\ldots, 0, 1)$. An other 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, or 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. That's why in $\mathrm{GF}_2[x]$ we consider only the polynomials with degree at most 1. So we will have $\alpha, \beta \in \{0, 1, x, x+1\}$, and only 9 possible couples:

$$P_{\mathrm{GF}_2[x]} = \{(0,1), (1,0), (1,1), (x,1), (1,x), (x{+}1,1), (1,x{+}1), (x{+}1,x), (x,x{+}1)\}$$

With this restriction we will be able to analyse Toom-k algorithms up to Toom-5. For any choice of the points the following theorem tells us that any Toom-k in $\mathrm{GF}_2[x]$with $k > 2$ requires at least one division.

Theorem 2. *Suppose $d > 2$, and the two points $(0,1),(1,0) \in P_d$. Then, for any choice of the other points $d-1$ points in P_d, the determinant of the invertible matrix A_d for a Toom algorithm in $\mathrm{GF}_2[x]$ is not a power of x.*

Proof. From theorem 1 we have:

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

By contradiction, if the determinant is a power of x, then any factor of the above formula must be a power of x. Then all the α_i and β_i are power of x. Any factor $(\alpha_i\beta_j - \alpha_j\beta_i)$ is then a difference of powers of x, giving 0 (a non invertible matrix) or a non-power of x.

3.3 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 3. *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 the rank is n.

4 Optimising Through Graph Search

To study ES and IS, we need at first to fix the operations we admit. We consider 4 basic operations, giving a name for their cost in time:

- add or subtract two elements (cost: `add`)
- multiply an element by a small constant (cost: `Smul`)
- exact division by a small constant (cost: `Sdiv`)
- bit-shift by a small amount (cost: `shift`)

By *small constant*, we mean an element which fits in a few bytes, hopefully in a register of the target CPU. All the resulting algorithms in this paper use *small constants* needing at most two bytes.

We assume that additions and subtractions do cost the same, right and left shift do cost the same, multiplication cost and exact division cost do not depend on the constant. Moreover we require some relations on the costs:

- `shift` < `add`: it should be faster to compute $a = b \ll 1$ than $a = b + b$
- `shift` < `Smul`: it should be faster to compute $a = b \ll 3$ than $a = b \cdot x^3$

In the experiments we also used the empirical relations `shift` < `Smul` < `add` < `Sdiv`, but we did not assume those to be true in general.

We also assume that any linear combination $l_i \leftarrow \pm c_j \cdot l_j/d_j \pm c_k \cdot l_k/d_k$ is possible without using temporary variables, for any $c_j, d_j, c_k, d_k \in \mathbb{R}[x]$ small constants, even if $i = j$. The cost of this linear combination will be simply computed adding up the cost of single operations, converted to bit-shift whenever possible or skipped when the coefficient is trivial.

4.1 Searching for Evaluation Sequences

The sequences ES and IS will be searched working on their respective matrices. IS can be seen as a sequence of operations on the lines of a matrix, starting from the matrix A_d and reaching the identity matrix. A method to determine the optimal IS with no use of temporaries was already given in [2]; except theorem 2 already shown, all the results from that paper can be directly applied to $\mathrm{GF}_2[x]$; the same strategy was used to find optimal IS for this paper.

Here we focus on ES. Also ES can be searched working only on the matrix. Again we require the algorithm not to use any temporary variable.

Any evaluation $\mathtt{u}(\alpha_i, \beta_i) = \sum_{j=0}^{n-1} u_j \cdot (\alpha_i^{n-1-j} \cdot \beta_i^j)$ can be directly computed with a cost at most equal to $(n-1) \cdot \mathtt{sum} + n \cdot \mathtt{Smul}$, without any division. So we search for the best ES without divisions.

We will search a sequence of elementary operations on lines starting from the zero matrix and leading to the goal matrix $E_{d,n}$. Computing the evaluations $\mathtt{u}(\alpha_i, \beta_i)$ we can always use the coefficients of the polynomial $\mathtt{u}$, the vector $\overline{u} = (\ldots, u_j, \ldots)$. These values can not be modified. So we use a block matrix, where one block is the identity, and the other is the goal $E_{d,n}$. Moreover, since we always use the two points $(0,1), (1,0)$, and they give two rows already present in the identity matrix, we will cut off two lines and use a smaller $\widetilde{E}_{d,n}$ as the goal.

$$\begin{pmatrix} \overline{u} \\ \hline \overline{\mathtt{u}(\alpha_i,\beta_i)} \end{pmatrix} = \begin{pmatrix} \boxed{I} \\ \hline \boxed{E} \end{pmatrix} \cdot \overline{u}; \quad \begin{pmatrix} I_n \\ \hline E_{d,n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & 0 & 1 \\ \hline 1 & 0 & \cdots \\ \hline & \widetilde{E}_{d,n} & \\ \hline \cdots & 0 & 1 \end{pmatrix} ; \text{ examine } \begin{pmatrix} l_{-1}: & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ l_{-n}: & \cdots & 0 & 1 \\ \hline l_1: & & & \\ \vdots & & \widetilde{E}_{d,n} & \\ l_{d-2}: & & & \end{pmatrix}$$

Lines coming from the identity matrix must be left untouched, and are noted with a negative index. Allowed operations are

$$l_i \leftarrow c_j \cdot l_j + c_k \cdot l_k, \quad \text{where } i > 0, k \neq i, \quad c_j, c_k \text{ are null or small constants} \tag{3}$$

Then we look for a sequence starting from the empty matrix $M_0 = (\overline{0})$, reaching the goal matrix $\widetilde{E}_{d,n}$. Every single step changes only one line in the M matrix with a linear combination of lines as formula (3) shows.

$$\begin{pmatrix} l_{-1}: & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ l_{-n}: & \cdots & 0 & 1 \\ \hline l_1: & 0 & \cdots & 0 \\ \vdots & \vdots & 0 & \vdots \\ l_{d-2}: & 0 & \cdots & 0 \end{pmatrix} \overset{l_1 \leftarrow l_{-1} + l_{-2}}{\rightsquigarrow} \begin{pmatrix} 1 & 0 & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & 0 & 1 \\ \hline 1\,1 & 0 & \cdots \\ \vdots & 0 & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \cdots \begin{pmatrix} \boxed{I} \\ \hline \boxed{M} \end{pmatrix} \cdots \rightsquigarrow \begin{pmatrix} l_{-1}: & 1 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ l_{-n}: & \cdots & 0 & 1 \\ \hline l_1: & & & \\ \vdots & & \widetilde{E}_{d,n} & \\ l_{d-2}: & & & \end{pmatrix} \tag{4}$$

4.2 The Graph

Now we have all the ingredients to build a graph, and search for the *shortest path*. The nodes in the graph are all the possible matrices of the form $\left(\begin{array}{|c|}\hline I \\ \hline M \\ \hline\end{array}\right)$ as in (4), and will be labelled by M. From every node, directed arcs represent possible operations as in (3), we only need a limit for the possible coefficients. For the result in this paper we explored combinations with coefficients limited by the biggest coefficient in the goal matrix $E_{d,n}$. In $\mathbb{Z}[x]$ the limit being the absolute value, in $\mathrm{GF}_2[x]$ the degree. The weight of an arc $M \to \widetilde{M}$ is the minimal cost of the operations of the form (3) that lead M to $\widetilde{M}$.

The graphs mentioned above have an infinite number of nodes, so it's essential to use a clever algorithm for the shortest path search. Two possibilities were explored: the Travel Through algorithm described in the previous work [2] on IS and the more standard A* algorithm [7]. The first being slower, but with a smaller memory footprint. While the second is faster but needs too much memory for big matrices.

4.3 Estimate for Evaluation Sequences

Both A* and the Travel Through algorithm need a function to estimate the remaining cost of a path, the estimated cost for a given node M must be smaller or equal to the actual cost of the shortest path from M to the goal $G = \widetilde{E}_{d,n}$.

To build this function we need some preliminary definitions and observations.

Definition 1 (Insertion). *A given arc $l_i \leftarrow c_j \cdot l_j + c_k \cdot l_k$ is an* **insertion** *if and only if $j < 0 \wedge c_j \neq 0$ or $k < 0 \wedge c_k \neq 0$.*

Theorem 4. *If there exist a path of non-insertion arcs from node M to $\widetilde{M}$, then* $\mathrm{rank}(M) \geq \mathrm{rank}(\widetilde{M})$.

Proof. A non-insertion arc, operates inside the matrix M. The resulting line is a linear combination of lines in M, so the rank can not grow.

Theorem 5 (Rank estimate). *The cost of the path from any node M to the goal G is at least* $(\mathrm{rank}(G) - \mathrm{rank}(M)) \cdot \mathtt{add}$.

Proof. By theorem 3, the $\mathrm{rank}(G)$ is maximal and $\mathrm{rank}(G) - \mathrm{rank}(M) \geq 0$. Each step modifies only one line, so the rank will be increased one by one.

Definition 2 (Needed insertions). *Given a matrix M, and a line G_i of the goal matrix, we define $Ni(G_i, M)$ the minimal number of insertions needed to obtain the line G_i from the matrix M.*

If we fix a line M_j of the matrix M, and we note M_{jk}, G_{ik} the k-th elements of the two lines we can compute the minimal number of needed insertions for a path from M_j to G_i with

$$\widetilde{Ni}(G_i, M_j) = n - \max_{\lambda \in \mathrm{GF}_2[x] \setminus \{0\}} (\#\{k : G_{ik} = \lambda M_{jk}\}, 2).$$

Then we can compute the global $Ni(G_i, M) = \min_j(\widetilde{Ni}(G_i, M_j))$.

Theorem 6 (Line estimate). *The cost of the path from any node M to the goal G is at least* $(\#\{i : Ni(G_i, M) \neq 0\}) \cdot \texttt{add}$.

Proof. The function $Ni(G_i, M)$ for a given line G_i gives zero iff G_i is already in M. A combination is needed to change each line which is not yet in the goal.

Theorem 7 (Combined estimate). *To estimate the cost of the path from a node M to the goal G; let $r = \mathrm{rank}(G) - \mathrm{rank}(M)$, $a = \#\{i : Ni(G_i, M) = 1\}$ and $b = \#\{i : Ni(G_i, M) > 1\}$, then*
if $r \leq a$ the cost is at least $(a + b) \cdot \texttt{sum}$
if $r > a$ the cost is at least $(r + b - \lfloor (r - a)/2 \rfloor) \cdot \texttt{sum}$

Proof. If $r \leq a$ the cost is that of theorem 6.

If $r > a$ we proceed by induction. For the base case, we note that the formula $f(r, a, b) = (r + b - \lfloor (r - a)/2 \rfloor)$, when $r = a$, gives $f(a, a, b) = a + b$ which is correct.

Then we study how the values a, b, r are modified following an arc from M to an other matrix M'. We can have the new $r' < r$ only if the arc is an insertion, when this happens we have $r' = r - 1$. An insertion can decrease by one the lines counted by a, or move a line from the set counted by b to the set counted by a. If the first condition applies, $b' = b, a' = a - 1 \Rightarrow f(r', a', b') = r - 1 + b - \lfloor (r - 1 - a + 1)/2 \rfloor) = f(r, a, b) - 1$, if the second applies, $b' = b - 1, a' = a + 1 \Rightarrow f(r', a', b') = r - 1 + b - 1 - \lfloor (r - a - 2)/2 \rfloor) = f(r, a, b) - 1$. Otherwise, if $r' = r$, the arc can be a non-insertion, so it can change more than one element, but on a single line, and possibly decrease a or b by 1. In both cases $f(r', a', b') \geq f(r, a, b) - 1$.

The last combined estimate is stronger than the others and is good enough to allow the complete analysis for Toom-4 matrices in $\mathrm{GF}_2[x]$.

5 Results and Algorithms in Characteristic 2

The algorithm described in the following sections were studied to work in $\mathrm{GF}_2[x]$, but can be applied in general for characteristic 2. We skip Toom-2 because it coincides with the well known Karatsuba.

5.1 Toom-2.5 in $\mathbf{GF_2}[x]$

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.

There are many possible choices for the set of points P_3, once inserted the canonical points $(1, 0), (0, 1)$, there is a couple of free points left.

Many pairs of points give a total cost of both ES for $E_{3,2}$ and $E_{3,3}$ equal to $6 \cdot \texttt{add} + 3 \cdot \texttt{shift}$ and the evaluation always require 4 multiplications. But only two pairs[1], $(1, 1), (x, 1)$ and $(1, 1), (x + 1, 1)$, reach the minimum cost for the IS: $6 \cdot \texttt{add} + 2 \cdot \texttt{shift} + 1 \cdot \texttt{Sdiv}$.

[1] Also their reciprocal $(1, 1), (1, x)$ and $(1, 1), (1, x + 1)$.

We show here both algorithms, they are very similar. The author prefers the first one, involving $x+1$, because of a slightly better locality.
$P_3^{x+1} = \{(0,1),(1,1),(x+1,1),(1,0)\}$ $\quad P_3^{x} = \{(0,1),(1,1),(x,1),(1,0)\}$

$$E^{x+1}_{3,2}=\begin{pmatrix}1&0\\ \hline 1&1\\1&3\\ \hline 0&1\end{pmatrix};E^{x+1}_{3,3}=\begin{pmatrix}1&0&0\\ \hline 1&1&1\\1&3&5\\ \hline 0&0&1\end{pmatrix};A^{x+1}_3=\begin{pmatrix}1&0&0&0\\1&1&1&1\\1&3&5&F\\0&0&0&1\end{pmatrix}E^{x}_{3,2}=\begin{pmatrix}1&0\\ \hline 1&1\\1&2\\ \hline 0&1\end{pmatrix};E^{x}_{3,3}=\begin{pmatrix}1&0&0\\ \hline 1&1&1\\1&2&4\\ \hline 0&0&1\end{pmatrix};A^{x}_3=\begin{pmatrix}1&0&0&0\\1&1&1&1\\1&2&4&8\\0&0&0&1\end{pmatrix}$$

```
U  = U2*Y^2 + U1*Y + U0
V  = V1*Y   + V0
\\ Evaluation:6 add,3 shift;4 mul
W3 = U2 + U1 + U0; W0 = V1 + V0
W1 = W3 * W0
W3 = W3 +(U1 + U2*(x))*(x)
                    W0 = W0 + V1*(x)
W2 = W3 * W0
W3 = U2 * V1     ; W0 = U0 * V0
\\ Interpolate:6 add,2 shift,1 Sdiv
W2 =(W2 + W1)/(x)
W1 = W1 + W0
W2 =(W2 + W1)/(x+1)
W2 = W2 + W3*(x)
W1 = W1 + W2 + W3

\\ Recomposition
W = W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V
```

```
U  = U2*Y^2 + U1*Y + U0
V  = V1*Y   + V0
\\ Evaluation:6 add,3 shift,4 mul
W3 = U2 + U1 + U0; W0 = V1 + V0
W1 = W3 * W0
W3 = U0 +(U1 + U2*(x))*(x)
                    W0 = V0 + V1*(x)
W2 = W3 * W0
W3 = U2 * V1     ; W0 = U0 * V0
\\ Interpolate:6 add,2 shift,1 Sdiv
W2 =(W2 + W1)/(x+1)
W1 = W1 + W0
W2 =(W2 + W1)/(x)
W2 = W2 + W3*(x)
W1 = W1 + W2
W2 = W2 + W3
\\ Recomposition
W = W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V
```

5.2 Toom-3 in $\mathbf{GF_2}[x]$

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, only one implementation of balanced Toom-3 in $\mathrm{GF}_2[x]$ was found on the net, by Zimmermann [18], based on the NTL library [12] and carefully optimised. It uses the points $P_4^Z = \{(0,1),(1,x),(1,1),(x,1),(1,0)\}$, an ES requiring $6 \cdot$ `add` $+ 4 \cdot$ `shift` for each operand, and an IS with cost $11 \cdot$ `add` $+ 5 \cdot$ `shift` $+ 2 \cdot$ `Sdiv`.

We tested all the triplets of points in $P_{\mathrm{GF}_2[x]} \setminus (1,0),(0,1)$, and the combination $(1,1),(1,x),(1,x+1)$ together with its reciprocal $(1,1),(x,1),(x+1,1)$ gave the best results.

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 uses $E_{4,2}$ for the smallest operand and $E_{4,4}$ for the bigger one.

The set of points used is $P_4 = \{(0,1),(1,1),(1,x),(1,x+1),(1,0)\}$.

$$E_{4,3} = \left(\begin{array}{ccc} 1&0&0\\ \hline 1&1&1\\ 1&2&4\\ 1&3&5\\ \hline 0&0&1 \end{array}\right);\ A_4 = \begin{pmatrix} 1&0&0&0&0\\ 1&1&1&1&1\\ 1&2&4&8&10\\ 1&3&5&F&11\\ 0&0&0&0&1 \end{pmatrix} \qquad E_{4,2} = \left(\begin{array}{cc} 1&0\\ \hline 1&1\\ 1&2\\ 1&3\\ \hline 0&1 \end{array}\right);\ E_{4,4} = \left(\begin{array}{cccc} 1&0&0&0\\ \hline 1&1&1&1\\ 1&2&4&8\\ 1&3&5&F\\ \hline 0&0&0&1 \end{array}\right)$$

```
U = U2*Y^2 + U1*Y + U0                U = U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V2*Y^2 + V1*Y + V0                V = V1*Y   + V0
\\ Evaluation:10 add,4 shift;5 mul    \\Eval:11 add,4 shift,1 Smul;5 mul
W3 = U2+U1+U0    ; W2 = V2+V1+V0      W3 = U3+U2+U1+U0 ;W2 = V1 + V0
W1 = W3 * W2                          W1 = W2 * W3
W0 = U2*x^2+U1*x ; W4 = V2*x^2+V1*x   W0 = U3*(x^3)+U2*(x^2)+U1*(x)
W3 = W3 + W0     ; W2 = W2 + W4       W3 = W3 + W0 + (x^2+x)*U3
                                                        W2 = W2 + V1*(x)
W0 = W0 + U0     ; W4 = W4 + V0       W0 = W0 + U0     ;W4 = W2 + V1
W3 = W3 * W2     ; W2 = W0 * W4       W3 = W3 * W2     ;W2 = W0 * W4
W4 = U2 * V2     ; W0 = U0 * V0       W4 = U3 * V1     ;W0 = U0 * V0

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

The IS needs two exact divisions, one by the small constant element $x+1$ and one by $x\cdot(x+1)$. Since we know these divisions are exact by very small constant, they can be performed in linear time [8]. For a test implementation in NTL on a 32-bit CPU, the following C code for exact division by $x+1$ was implemented. It is inspired by an analogous function by Michel Quercia [18]. Division by x^2+x was actually implemented by one shift and the same function.

```
static void ExactDivOnePlusX (_ntl_ulong *c, long n) {
  _ntl_ulong t = 0; long i;
  for (i = 0; i < n; i++) {
    t ^= c[i]  ; t ^= t << 1; t ^= t << 2;
    t ^= t << 4; t ^= t << 8; t ^= t << 16;
    c[i] = t; t >>= 32-1;
}}
```

The main idea for this function is to multiply each word by the inverse of $x+1$ modulo x^{2^b}, where 2^b is the number of coefficients stored in one word. This requires $b+1$ shifts and sums for each word. Similar functions can be developed for any exact division needed in Toom-3,4,5 IS.

With this function, the new algorithm is about 5% faster than Zimmermann's, and beats the NTL mul starting from 8 words, meaning degree 256. It is also faster than Karatsuba for operands above 11 words, or degree 352.

5.3 Toom-4 in $\mathbf{GF_2}[x]$

The complete analysis of the Toom-4 candidate algorithms requires too much resources. So here we tested only the most promising choice for the 7 points: $P_6 = \{(0,1),(1,x+1),(1,x),(1,1),(x,1),(x+1,1),(1,0)\}$. Here we show only the balanced algorithm, used when the two operands have about the same size, and the matrices.

```
U = U3*Y^3 + U2*Y^2 + U1*Y + U0
V = V3*Y^3 + V2*Y^2 + V1*Y + V0
\\ Evaluation: 13*2 add, 7*2 shift, 2*2 Smul, 7 mul
W1 = U3 + U2 + U1 + U0    ; W2 = V3 + V2 + V1 + V0
W3 = W1 * W2
W0 = U1 + x*(U2 + x*U3)   ; W6 = V1+ x*(V2 + x*V3)
W4 =(W0 + U3*(x+1))*x+W1  ; W5 =(W6 + V3*(x+1))*x+W2
W0 = W0*x + U0            ; W6 = W6*x + V0
W5 = W5 * W4              ; W4 = W0 * W6
W0 = U0*x^3+U1*x^2+U2*x   ; W6 = V0*x^3+V1*x^2+V2*x
W1 = W1 + W0 + U0*(x^2+x) ; W2 = W2 + W6 + V0*(x^2+x)
W0 = W0 + U3              ; W6 = W6 + V3
W1 = W1 * W2              ; W2 = W0 * W6
W6 = U3 * V3              ; W0 = U0 * V0
\\ Interpolation: 22 add, 4 shift, 4 Sdiv, 5mul
W1 = W1 + W2 + W0*(x^4+x^2+1)
W5 =(W5 + W4 + W6*(x^4+x^2+1) + W1)/(x^4+x)
W2 = W2 + W6 + W0*x^6
W4 = W4 + W2 + W6*x^6 + W0
W4 =(W4 + W5*(x^5+x))/(x^4+x^2)
W3 = W3 + W0 + W6
W1 = W1 + W3
W2 = W2 + W1*x + W3*x^2
W3 = W3 + W4 + W5
W1 =(W1 + W3*(x^2+x))/(x^4+x)
W5 = W5 + W1
W2 =(W2 + W5*(x^2+x))/(x^4+x^2)
W4 = W4 + W2
\\ Recomposition:
W = W6*Y^6 + W5*Y^5 + W4*Y^4+ W3*Y^3+ W2*Y^2+ W1*Y + W0
W == U*V \\ check
```

$$E_{6,4} = \left(\begin{array}{|cccc|} \multicolumn{4}{c}{1\ 0\ 0\ 0} \\ \hline F & 5 & 3 & 1 \\ 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 5 & F \\ \hline \multicolumn{4}{c}{0\ 0\ 0\ 1} \end{array}\right)$$

$$A_6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 55 & 33 & 11 & \mathrm{F} & 5 & 3 & 1 \\ 40 & 20 & 10 & 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 10 & 20 & 40 \\ 1 & 3 & 5 & \mathrm{F} & 11 & 33 & 55 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

5.4 Toom-5 in $\mathbf{GF_2}[x]$

The complete analysis of Toom-5 is even harder. There is only one possible choice for the evaluating points with minimal degree:

$$P_{\mathrm{GF}_2[x]} = \{(0,1),(x{+}1,x),(x{+}1,1),(x,1),(1,1),(1,x),(1,x{+}1),(x,x{+}1),(1,0)\}.$$

The resulting algorithms are too big to be transcribed here, the found cost being:

ES: $2 \times (19 \cdot \texttt{add} + 6 \cdot \texttt{shift} + 4 \cdot \texttt{Smul})$

IS: $36 \cdot \texttt{add} + 9 \cdot \texttt{shift} + 5 \cdot \texttt{Smul} + 6 \cdot \texttt{Sdiv}$

Only 3 different denominators for exact division are needed: $x^3 \cdot (x+1)^3$, $x \cdot (x+1) \cdot (x^2+x+1)^2$ and $x^2 \cdot (x+1)^2 \cdot (x^2+x+1)$. The two matrices are

$$E_{8,5} = \left(\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ \hline 11 & 1E & 14 & 18 & 10 \\ 11 & F & 5 & 3 & 1 \\ 10 & 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 10 \\ 1 & 3 & 5 & F & 11 \\ 10 & 18 & 14 & 1E & 11 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right) \quad A_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 101 & 1FE & 154 & 198 & 110 & 1E0 & 140 & 180 & 100 \\ 101 & FF & 55 & 33 & 11 & F & 5 & 3 & 1 \\ 100 & 80 & 40 & 20 & 10 & 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 10 & 20 & 40 & 80 & 100 \\ 1 & 3 & 5 & F & 11 & 33 & 55 & FF & 101 \\ 100 & 180 & 140 & 1E0 & 110 & 198 & 154 & 1FE & 101 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

6 Bivariate and Multivariate

The same strategy described in section 2 can be extended to multivariate polynomials. In particular it fits perfectly for those polynomials whose homogenisation is dense: polynomials dense with respect to *total degree*.

On the opposite side there is the Kronecker substitution [11] which is very efficient for polynomials dense with respect to *maximal degree*.

Definition 3. *We call* triangular polynomial *a polynomial dense with respect to total degree. We mean a polynomial where coefficients for all the possible terms with* sum of exponents *limited by a constant d are mostly non-zero. We will call* square polynomial*, those which are dense with respect to* maximal degree*.*

A couple of examples, $1+x+y+x^2+y^2+xy$ *will be called* triangular*, while* $1+x+y+x^2+y^2+xy+x^2y+xy^2+x^2y^2$ *is a* square polynomial*.*

6.1 Multivariate Toom-2

Karatsuba's idea was generalised in many ways, one of them can be the extension to multivariate polynomials. If we start from two *triangular polynomials*, u, v, and we want to compute the product $w = u \cdot v$, we can proceed as in section 2.

If we call X_0 the homogenising variable and X_i the other ones, we will have the canonical splitting $\mathfrak{u} = \sum_i u_i \cdot X_i$. Then we have many smaller polynomials u_i, where u_0 is a *square*, and the others are *triangular*.

All the evaluations and interpolations can be condensed in a one-line formula, valid in any characteristic:$\mathfrak{u} = \sum_i u_i X_i \wedge \mathfrak{v} = \sum_i v_i X_i \Rightarrow$

$$\mathfrak{w} = \mathfrak{u} \cdot \mathfrak{v} = \sum_i (u_i \cdot v_i) X_i^2 + \sum_{i<j} \left((u_i - u_j) \cdot (v_j - v_i) + u_i v_i + u_j v_j \right) X_i X_j$$

where any product $u_i v_i$ is computed only once, and recycled for all the $X_i X_j$ coefficients.

Recurrence is not very easy in this algorithm, because all the products involving u_0 and v_0 are *square* product, where the same algorithm can not be used. On squares we can fall back to the Kronecker's trick or use univariate algorithms recursively on any variable.

Another possible formula for the product, is the nearly equivalent

$$\mathfrak{w} = \mathfrak{u} \cdot \mathfrak{v} = \sum_i (u_i \cdot v_i) X_i^2 + \sum_{i<j} \left((u_i + u_j) \cdot (v_i + v_j) - u_i v_i - u_j v_j \right) X_i X_j$$

which is interesting for one reason: if we use this formula for an univariate polynomial, with the identification $X_i = x^i$, we obtain the Karatsuba generalisation by Weimerskirch and Paar [16].

6.2 Bivariate Toom-2.5 in $\mathbf{GF_2}[x]$

The smallest interesting example of multivariate Toom, which is not a generalisation of Karatsuba, is the algorithm to multiply a polynomial of degree 2 by one of degree 1. Both with 2 variables. The product has degree 3, so it will have $\binom{3+2}{2} = 10$ coefficients, and we need 10 points.

After homogenising, we have 3 variables and evaluation points need 3 values. With the points $P_3^2 = \{(1,0,0),(1,1,0),(1,x+1,0),(0,1,0),(0,1,1),(0,1,x+1),(0,0,1),(1,0,1),(x+1,0,1),(1,1,1)\}$ we obtain the block-like matrices:

$$A_3^2 = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&0&0&0&0&0&0\\ F&5&3&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0\\ 0&0&0&1&1&1&1&0&0&0\\ 0&0&0&F&5&3&1&0&0&0\\ 0&0&0&0&0&0&1&0&0&0\\ 1&0&0&0&0&0&1&1&1&0\\ 1&0&0&0&0&0&F&5&3&0\\ 1&1&1&1&1&1&1&1&1&1 \end{pmatrix}; \quad E_{3,2}^2 = \begin{pmatrix} 1&0&0\\ 1&1&0\\ 3&1&0\\ 0&1&0\\ 0&1&1\\ 0&3&1\\ 0&0&1\\ 1&0&1\\ 1&0&3\\ 1&1&1 \end{pmatrix}; E_{3,3}^2 = \begin{pmatrix} 1&0&0&0&0&0\\ 1&1&1&0&0&0\\ 5&3&1&0&0&0\\ 0&0&1&0&0&0\\ 0&0&1&1&1&0\\ 0&0&5&3&1&0\\ 0&0&0&0&1&0\\ 1&0&0&0&1&1\\ 1&0&0&0&5&3\\ 1&1&1&1&1&1 \end{pmatrix}$$

We can observe that the square invertible matrix A_{2k-2}^n used for the n-variate Toom-k, is not a Vandermonde matrix when $n > 1$, and it has $\binom{2k-2+n}{n}$ lines (and columns). Also the $E_{2k-2,d}^n$ are somehow sparse $\binom{2k-2+n}{n} \times \binom{d-1+n}{n}$ matrices.

Theorems proved in this paper can not be directly extended to those new Vandermonde-blocks matrices, anyway algorithms developed for the univariate case still works. Sparse matrix give graphs much smaller than expected and can be fully analysed. The best bivariate triangular Toom-2.5 found by our software follows.

```
U =  U00*Z^2 + U10*Z*X + U20*X^2 \
   + U01*Z*Y + U11*Y*X \
   + U02*Y^2
V =  V00*Z   + V10*X   \
   + V01*Y
\\ Evaluation: 22 add, 9 shift; 10 mul
W3 = U20+ U10+ U00          ; W0 = V10+ V00     ; W1 = W0 * W3
W3 = W3 +(U10+U20*(x))*(x)  ; W0 = W0 + V10*(x) ; W2 = W0 * W3
W3 = U20+ U11+ U02          ; W0 = V10+ V01     ; W4 = W0 * W3
```

```
W3 = W3 +(U11+U02*(x))*(x) ; W0 = W0 + V01*(x) ; W5 = W0 * W3
W3 = U02+ U01+ U00         ; W0 = V00+ V01     ; W7 = W0 * W3
W9 = W3 +(U01+U00*(x))*(x) ; W6 = W0 + V00*(x) ; W8 = W6 * W9
W3 = W3 + U20+ U11+ U10    ; W0 = W0 + V10
W9 = W3 * W0; W6 = U02* V01; W0 = U00* V00; W3 = U20* V10
\\ Interpolation: 21 add, 6 shift; 3 Sdiv
W2 =(W2 + W1)/(x)   ; W5 =(W5 + W4)/(x)   ; W8 =(W8 + W7)/(x)
W1 = W1 + W0        ; W4 = W4 + W3        ; W7 = W7 + W6
                      W9 = W9 + W7 + W4 + W1
W2 =(W2 + W1)/(x+1) ; W5 =(W5 + W4)/(x+1) ; W8 =(W8 + W7)/(x+1)
W2 = W2 + W3*(x)    ; W5 = W5 + W6*(x)    ; W8 = W8 + W0*(x)
W1 = W1 + W2 + W3   ; W4 = W4 + W5 + W6   ; W7 = W7 + W8 + W0
\\ Recomposition
W = W0*Z^3   + W1*Z^2*X + W2*Z*X^2 + W3*X^3 \
  + W8*Z^2*Y + W9*Z*Y*X + W4*Y*X^2 \
  + W7*Z*Y^2 + W5*Y^2*X \
  + W6*  Y^3
W==U*V
```

Three instances of the $x+1$ version of univariate Toom-2.5 can be recognised in the code, the same trick could be applied using the x version. Only the point $(1,1,1)$ requires some extra operations.

6.3 Bivariate Toom-3 in $\mathbf{GF_2}[x]$

The first non-Karatsuba multivariate Toom which can be used for recursion is the bivariate triangular Toom-3, with this algorithm we can multiply two *triangular* bivariate polynomials with degree 2 to obtain a triangular result with degree 4. This time we need $\binom{4+2}{2} = 15$ interpolation points. The choice $P_4^2 = \{ (1,0,0), (1,x+1,0), (1,x,0), (1,1,0), (0,1,0), (0,1,1), (0,1,x), (0,1,x+1), (0,0,1), (x,0,1), (1,0,1), (1,0,x), (1,1,x), (1,1,1), (x,1,1)\}$ gives again block-like matrices. The algorithm requires 15 smaller multiplication, 5 involving *triangular* polynomials, and 10 requiring some *squared polynomial* algorithm. We choosed three different sub-matrices, so it's more difficult to recover sub-IS.

$$A_4^2 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & F & 5 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 8 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 3 & 5 & F & 11 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 8 & 4 & 2 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 0 & 0 & 0 \\
10 & 8 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 4 & 8 & 4 & 2 & 2 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 4 & 8 & 10 & 8 & 4 & 2 & 2 & 2 & 4
\end{pmatrix}$$

```
U = U00*Z^2+U10*Z*X+U20*X^2+U01*Z*Y+U11*X*Y+U02*Y^2
V = V00*Z^2+V10*Z*X+V20*X^2+V01*Z*Y+V11*X*Y+V02*Y^2
\\ Evaluation: 23*2 add, 6*2 shift; 15 mul
W0 = U00+ U10+ U20     ; W4 = V00+ V10+ V20
```

```
W12=(U10+ U00*(x))*(x) ; W10=(V10+ V00*(x))*(x)
W2 = W0 + W12          ; W8 = W4 + W10
W3 = W12+ U20          ; W5 = W10+ V20
W1 = W2 * W8 ; W2 = W3 * W5 ; W3 = W0 * W4
W6 = U20+ U11+ U02     ; W7 = V20+ V11+ V02
W8 = U01*(x)           ; W13= V01*(x)
W11= W6 + W12+ W8      ; W10= W10+ W7 + W13
W12= W11* W10; W5 = W6 * W7
W10= U02*(x^2)         ; W11= V02*(x^2)
W9 = W10+ U11*(x)      ; W14= W11 +V11*(x)
W6 = W6 + W9           ; W7 = W7  +W14
W0 = W0 + W9 +W8       ; W4 = W4  +W14+W13
W9 = W9 + U20          ; W14= W14 +V20
W10= W10+ W8 +U00      ; W11= W11 + W13+V00
W8 = W8 + U02+U00*(x^2); W13= W13 + V02+V00*(x^2)
W7 = W6 * W7 ; W6 = W9 * W14; W14= W0 * W4
W0 = U02+U01+U00       ; W4 = V02+V01+V00
W9 = W10* W11; W11= W8 * W13; W10= W0 * W4
W0 = W0+U20+U11+U10    ; W4 = W4+V20+V11+V10
W13= W0 * W4 ; W8 = U02*V02 ; W0 = U00*V00 ; W4 = U20*V20
```

$$E^2_{4,3} = \begin{pmatrix} 1&0&0&0&0&0\\ 5&3&1&0&0&0\\ 4&2&1&0&0&0\\ 1&1&1&0&0&0\\ 0&0&1&0&0&0\\ 0&0&1&1&1&0\\ 0&0&1&2&4&0\\ 0&0&1&3&5&0\\ 0&0&0&0&1&0\\ 1&0&0&0&4&2\\ 1&0&0&0&1&1\\ 4&0&0&0&1&2\\ 4&2&1&1&1&2\\ 1&1&1&1&1&1\\ 1&1&1&2&4&2 \end{pmatrix}$$

```
\\ Interpolation: 42 add, 8 shift, 2 Smul, 8 Sdiv
W12= W12+W2             ;W14= W14+W6             ;W13= W13+ W5
W1 = W1 +W2             ;W7 = W7 +W6             ;W9 = W9 + W0 + W8*(x^4)
W2 =(W2 +W4 )/(x) + W1  ;W6 =(W6 +W4 )/(x) + W7  ;W11= W11+ W8 + W0*(x^4)
W2=(W2+W0*(x^3+1))/(x+1);W6=(W6+W8*(x^3+1))/(x+1);W10= W10+ W8 + W0
W3 = W3 +W4             ;W5 = W5 +W4             ; W13= W13+ W10
     W13 = W13+W3       ; W12 = W12+W5    ;  W12=((W12+W11)/x+W13)/(x+1)
     W14 = W14+W3       ;                    W14=((W14+W9 )/x+W13)/(x+1)
W1 =(W1 +W3)/(x*(x+1))  ;W7 =(W7 +W5)/(x*(x+1)) ;W9 =(W9+W11)/(x*(x^2+1))
W3 = W3 +W0 + W2        ;W5 = W5 +W8 + W6       ;W10= W10+ W9
W2 = W2 +W1             ;W6 = W6 +W7            ;W11=(W11/x+W9+W10*x)/(x^2+1)
   W13 = W13+W12+W14                            ;W9 = W9 +W11
\\ Recomposition
W = W0 *Z^4    + W1 *Z^3  *X+ W2 *Z^2*X^2 + W3*Z*X^3 + W4*X^4 \
  + W11*Z^3*Y  + W12*Z^2*Y*X+ W13*Z*Y*X^2 + W5*Y*X^3 \
  + W10*Z^2*Y^2+ W14*Z*Y^2*X+ W6 *Y^2*X^2 \
  + W9 *Z  *Y^3+ W7   *Y^3*X \
  + W8     *Y^4
W ==U*V
```

7 Conclusions

The paper presented a method to determine an optimal evaluation sequence of basic operations to be used in Toom multiplications. Joined with the previous work on inversion sequences [2], this gives a complete framework for the search of optimal Toom-Cook algorithms. This method shows his immediate effectiveness giving new algorithms to be used in $\mathrm{GF}_2[x]$, and the best known Toom-3 algorithm for $\mathbb{Z}[x]$ and $\mathbb{Z}$.

New generalisation of Toom described in section 2 open the possibility to easily generate simple Toom multiplication algorithms for polynomials on other integral domains [1]. Moreover section 6 generalise to multivariate polynomials, with a natural definition of density. Further work is needed to find general implementations working with any number of variables and any degree.

Note on Algorithms

Algorithms in this paper uses PARI/GP syntax [6], which should be simple enough to translate to any other language, and allow a fast checking within a GP shell. Some more definitions should be typed to have correct results for algorithms in characteristic 2.

```
U0 = u0 * Mod(1,2) ; U1 = u1 * Mod(1,2) ; U2 = u2 * Mod(1,2)
U3 = u3 * Mod(1,2) ; U4 = u4 * Mod(1,2) ; U5 = u5 * Mod(1,2)
V0 = v0 * Mod(1,2) ; V1 = v1 * Mod(1,2) ; V2 = v2 * Mod(1,2)
V3 = v3 * Mod(1,2) ; V4 = v4 * Mod(1,2) ; V5 = v5 * Mod(1,2)
```

Acknowledgements

The author wants to thank Paul Zimmermann for the many stimulating ideas and beautiful code which gave the start to this paper.

References

1. M. Bodrato: *Notes on Low Degree Toom-Cook Multiplication with Small Characteristic.* Preprint, Centro "V.Volterra", Università di Roma "Tor Vergata" (2007) http://bodrato.it/papers/#CIVV2007
2. M. Bodrato, A. Zanoni: *Integer and Polynomial Multiplication: Towards Optimal Toom-Cook Matrices.* Proceedings of the ISSAC 2007 Conference, ACM press, New York (2007) http://bodrato.it/papers/#ISSAC2007
3. J. Chung, M. Anwar Hasan: *Asymmetric squaring formulae.* Technical Report, CACR 2006-24, University of Waterloo (2006)
4. S.A. Cook: *On the Minimum Computation Time of Functions.* Thesis, Harvard University, 51–77 (1966)
5. The GNU Multi-Precision Library (GMP) http://gmplib.org/
6. The GP-Pari Computer Algebra System http://pari.math.u-bordeaux.fr/
7. P. E. Hart, N. J. Nilsson, B. Raphael: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics SSC4, 100–107 (1968).
8. T. Jebelean: *An algorithm for exact division.* Journal of Symbolic Computation, volume 15, 169–180 (1993).
9. А.Карацуба, Ю.Офман: *Multiplication of multidigit numbers on automata.* Soviet Physics-Doklady, 7, 595–596, (1963); transl. of "Умножение многозначных чисел на автоматах", in Доклады Академии Наук СССР, 145:2, 293–294, (1962)
10. D.E. Knuth: *The Art of Computer Programming, Vol. 2, Second Edition.* Addison-Wesley, Reading Mass., Chapter 4, Section 3.3, 278–301 (1981)
11. L. Kronecker: *Grundzüge einer arithmetischen Theorie der algebraischen Grössen.* Journal Für die reine und angewandte Mathematik, 92, 1–122 (1882)

12. The Number Theory Library (NTL) `http://www.shoup.net/ntl/`
13. А. Л. Тоом: *The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers.* Soviet Mathematics, Vol. 3, 714–716 (1963); transl. from "О сложности схемы из функциональных злементов, реализирующей умножение целых чисел" in Доклады Акад. Наук СССР, 150:3, 496–498, (1963)
14. A. Schönhage, V. Strassen: *Schnelle Multiplikation großer Zahlen.* Computing 7, 281–292 (1971)
15. A. Schönhage: *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2.* Acta Informatica 7, 395–398 (1977)
16. A. Weimerskirch, C. Paar: *Generalizations of the Karatsuba Algorithm for Efficient Implementations.* Cryptology ePrint Archive, Report 224 (2006)
17. S. Winograd: *Arithmetic Complexity of Computations.* CBMS-NSF Regional Conference Series in Mathematics, 33 (1980)
18. P. Zimmermann, M. Quercia: *Personal communication, October 2006.* `irred-ntl` source code (2003) `http://www.loria.fr/~zimmerma/irred/`

Z Appendix Z: Results in $\mathbb{Z}[x]$

Z.1 Toom-3 in $\mathbb{Z}[x]$

The IS for Toom-3 in $\mathbb{Z}$ was fully examined in the previous work with Zanoni. We give here the ES for the balanced and unbalanced (4x2) flavour. Both saves at least one `shift` if compared to the currently used ES. Both were tested against the GMP library, giving a small speedup.

```
U = U2*x^2 + U1*x + U0                   U = U3*x^3 + U2*x^2 + U1*x + U0
V = V2*x^2 + V1*x + V0                   V = V1*x + V0
\\ Evaluation: 5*2 add, 2 shift; 5mul    \\ Eval: 7+3 add, 3 shift; 5mul
W0 = U2 + U0      ; W4 = V2 + V0         W0 = U1 + U3 ; W4 = U0 + U2
W2 = W0 - U1      ; W1 = W4 - V1         W3 = W4 + W0 ; W4 = W4 - W0
W0 = W0 + U1      ; W4 = W4 + V1         W0 = V0 + V1 ; W2 = V0 - V1
W3 = W2 * W1      ; W1 = W0 * W4         W1 = W3 * W0 ; W3 = W4 * W2
W0 =(W0 + U2)<<1-U0; W4 =(W4 + V2)<<1-V0  W4 =((U3<<1+U2)<<1+U1)<<1+U0
W2 = W0 * W4                             W0 = W0 + V1 ; W2 = W4 * W0
W0 = U0 * V0      ; W4 = U2 * V2         W0 = U0 * V0 ; W4 = U3 * V1
```

```
\\ Interpolation: 8 add, 3 shift, 1 Sdiv
W2 =(W2 - W3)/3
W3 =(W1 - W3)>>1
W1 = W1 - W0
W2 =(W2 - W1)>>1 - W4<<1
W1 = W1 - W3 - W4
W3 = W3 - W2
\\ Recomposition:
W  = W4*x^4+ W2*x^3+ W1*x^2+ W3*x + W0
W == U*V
```

Z.2 Asymmetric Squaring in $\mathbb{Z}[x]$

Chung and Anwar Hasan proposed new linear systems useful for integer squaring; refer to their report [3] for the details. Here we only show results found by our software starting from their matrices, although not optimised for this case.

The report proposed an evaluation sequence and an inversion algorithm for the 5-way squaring using temporary variables, with cost respectively $14\cdot\texttt{add}+4\cdot\texttt{shift}$ and $18\cdot\texttt{add}+7\cdot\texttt{shift}$. We where able to find shorter sequences, whitout temporaries, reaching $12\cdot\texttt{add}+5\cdot\texttt{shift}$ and $16\cdot\texttt{add}+3\cdot\texttt{shift}$.

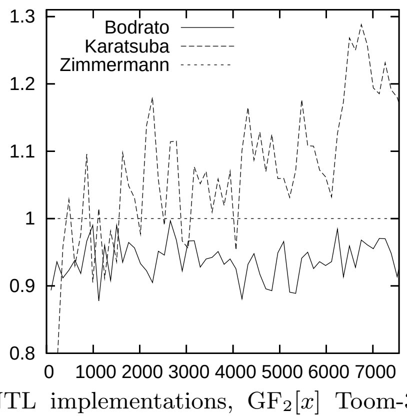
```
U = U4*Y^4 + U3*Y^3 + U2*Y^2 + U1*Y + U0
\\ Evaluation: 12 add, 5 shift; 5 mul, 4 sqr
W0 = U0 - U3 ; W1 = U3 - U1   ; W6 = U1 - U2
W4 = U1 + U2 ; W5 = W6 - U4   ; W3 = W5 + W0<<1
W0 = W0 - W5 ; W6 = W0 + W6<<1; W7 = W6 + W1
W5 = W7 + W1 ; W8 = W5 + W4<<1; W4 = W4 - U4
  W2 = W4 * W3; W4 = W6 * W5; W3 = W7 * W1
W1 = U0 * U1 * 2      ; W7 = U3 * U4 * 2
W5 = W8^2 ; W6 = W0^2 ; W0 = U0^2  ; W8 = U4^2
\\ Interpolation: 16 add, 3 shift.
W6 =(W6 + W5)>>1 ; W5 = W5 - W6
W4 =(W4 + W6)>>1 ; W6 = W6 - W4; W3 = W3 +W5>>1
W5 = W5 - W3 - W1; W4 = W4 - W8 - W0
W2 = W2 - W8 - W1 - W7 + W4 + W5
W3 = W3 - W7     ; W6 = W6 - W2
\\ Recomposition:
W  = W8*Y^8 + W7*Y^7 + W6*Y^6 + W5*Y^5    \
   + W4*Y^4 + W3*Y^3 + W2*Y^2 + W1*Y + W0
W == U^2
```

$$\widetilde{E} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 0 & -1 \\ 2 & 1 & -1 & -2 & -1 \end{pmatrix}$$

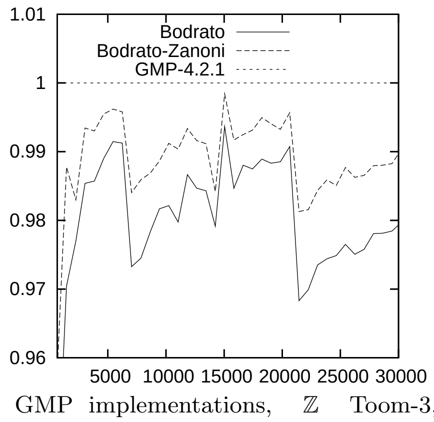
$$A_s = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 & 1 \\ 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

T Appendix T: Toom Three Timing Tests

We include at last some raw graphs of multiplication timings for balanced n bits long univariate operands. Both show relative timings for different operand sizes.



NTL implementations, $GF_2[x]$ Toom-3, normalised to Zimmermann's `irred-ntl` code. Up to degree 7,500.



GMP implementations, $\mathbb{Z}$ Toom-3, normalised to GMP-4.2.1 timings. Up to 30,000 bits operands.

THIS PAGE IS NOT PART OF THE ARTICLE[2]

BibTeX entry

```
@InProceedings{Bodrato:WAIFI2007,
  author =    {Marco Bodrato},
  title =     {Towards Optimal {Toom-Cook} Multiplication for
               Univariate and Multivariate Polynomials in
               Characteristic 2 and 0},
  pages =     {116--133},
  note =      {\url{http://bodrato.it/papers/\#WAIFI2007}},
  crossref =  {WAIFI2007},

  year =      {2007},
  booktitle = {{WAIFI 2007} proceedings},
  editor =    {Claude Carlet and Berk Sunar},
  volume =    {4547},
  series =    {LNCS},
  month =     {June},
  publisher = {Springer}
}

@Proceedings{WAIFI2007,
  title =     {{WAIFI} 2007 - {I}nternational {W}orkshop
               on the {A}rithmetic of {F}inite Fields,
               {M}adrid, {S}pain, {J}une 21-22, 2007},
  year =      {2007},
  location =  {Madrid, Spain},
  booktitle = {{WAIFI 2007} proceedings},
  editor =    {Claude Carlet and Berk Sunar},
  volume =    {4547},
  series =    {Lecture {N}otes in {C}omputer {S}cience},
  month =     {June},
  publisher = {Springer}
}
```

Thanks

Jörg Arndt[3] helped the author in proofreading this on-line version of the paper.

[2] Paper edited with Emacs-21 and LaTeX-3.0 on a Debian GNU/Linux box.
[3] `http://jjj.de/`