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
Via Columbia 2 – 00133 Rome, Italy
waifi2007@bodrato.it

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 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, Toom-Cook, Karatsuba, GF2x, 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 O($n^2$) to 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 sums and subtractions of arbitrary long operands, multiplication and exact division of long operand by small one, optimised, when possible, by bit-shifts.

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 $O(n^\log_k(2k-1))$. There is even the well known Schönhage-Strassen method[14,15], which 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 practical for small operands. So there can be a wide range of operands where Toom-Cook methods can be the preferred ones. The widely known GMP library\cite{5} uses Toom-2 from around 250 decimal digits, then Toom-3, and finally uses FFT based multiplication over 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\cite{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 GF\(_2[x]\) and Notation

All the algorithms in this paper work smoothly with elements of 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 somewhere use hexadecimal notation. Every hexadecimal number \(h\) corresponds to the element \(p \in \text{GF}_2[x]\) such that \(p(2) = h\). For example \(p \in \text{GF}_2[x] \leftrightarrow \text{hex}, 1 \leftrightarrow 1, x \leftrightarrow 2, x + 1 \leftrightarrow 3, \ldots, x^3 + x^2 + x + 1 \leftrightarrow F, \ldots, x^8 + x^7 + x^6 \leftrightarrow 1C0, \ldots\)

We will also use the symbols \(\ll\) and \(\gg\) for bit-shifts. Meaning multiplication and division by power of \(x\), in 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 \(u(y, z) = \sum_{i=0}^{n-1} u_i z^{n-1-i} y^i, v(y, z) = \sum_{i=0}^{m-1} v_i z^{m-1-i} y^i\), both homogeneous, with respectively \(n\) and \(m\) coefficients and degrees \(\deg(u) = n - 1, \deg(v) = m - 1\). Such that \(u(x^b, 1) = u, v(x^b, 1) = v\). 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 \(w = u \cdot v\) which 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 \(u\)) on the points \((\alpha_i, \beta_i)\), can be computed with a matrix by vector multiplication. The matrix \(E_{d,n}\) is a \((d + 1) \times n\) Vandermonde-like matrix. \(\bar{u}(\alpha, \beta) = E_{d,n} \bar{u} \implies\)