Iterative Karatsuba For Multivariate Polynomial Multiplication

Alberto Zanoni

Centro "Vito Volterra" – Università di Roma "Tor Vergata"
Via Columbia 2 – 00133 Roma, Italy
zanoni@volterra.uniroma2.it

Abstract. This work deals with Karatsuba method for multivariate polynomials, not recursing on variables number, but using an iterative scheme, with an eye to a better parallelism exploitation. Integers base 2 and 3 expansions are used in order to access the needed data.

AMS Subject Classification: 11A05, 11A25, 11K65, 11Y70

Keywords and phrases: Multivariate polynomials multiplication, Karatsuba

1 Introduction

The first subquadratic multiplication method for univariate polynomials was introduced by Karatsuba in [2]. The product of two linear polynomials $a(x) = a_1x + a_0$, $b(x) = b_1x + b_0$ is computed as $a_0b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)x + (a_1b_1)x^2$, so that just three coefficients multiplications are sufficient, instead of the four ones needed by the high school method $(a_0b_0, a_1b_0, a_0b_1, a_1b_1)$. It is a now classical evaluate-multiply-interpolate (EMI) scheme whose recursive use results in an algorithm with asymptotic complexity $O(n^{\log_2 3})$.

The (degree) recursion cost is a critical point of Karatsuba method, which makes it difficult to take advantage of parallelism at hardware level. Univariate approaches towards recursion reduction have been considered e.g. in Weimerskirch and Paar [7], Montgomery [5], Erdem and Koç [1] and Kounavis [3].

For multivariate polynomials, fast multiplication was considered e.g. by Moenck [4] by either:

1. recursing on every variable, with an approach having as basic case the univariate one.
2. using the Kronecker trick to reduce the problem to the univariate case, and then applying univariate algorithms (typically, the FFT approach of Schönhage and Strassen [6]).

Using approach 1, one has therefore to face a *double* recursion: in addition to the factors degree, the variables number enters the game. We'll show how Karatsuba method for multivariate polynomials can be realized iteratively on the number of variables, removing thus one "level" of recursion.

2 Multivariate "square" polynomials

We consider polynomials $a(X), b(X) \in \mathbf{R}[X]$, where $\mathbf{R}$ is a commutative ring with identity and $X = (x_1, \ldots, x_n)$ is a set of variables. We use the max-degree definition for multindexes $\alpha = [\alpha_1, \ldots, \alpha_n] \in A \subset \mathbb{N}^n$:

$$a(X) = \sum_{\alpha \in A} a_\alpha X^\alpha \in \mathbf{R}[X] \implies deg_\infty(a) = \|a\| = \max_{\substack{\alpha \in A \\ a_\alpha \neq 0}} \left\{ \max_{i=1,\ldots,n} \{\alpha_i\} \right\}$$

In order to have a "geometrical" view of the problem, it is useful to consider multindexes as points with integer coordinates in the n-dimensional space $\mathbb{N}^n$. In the rest of the paper we assume that $\alpha \in A \implies a_\alpha \neq 0$, that is, the multindex set coincides with the polynomial support.

2.1 Support issues

Considering max-degree definition, dense polynomials have a multindex support A which is nothing but a discrete n-dimensional hypercube (we speak about "square" polynomials). In the following, we will use the word "cube" as a shorthand for hypercube and "face" as a shorthand for hyperface.

Vertexes, sides, d-dimensional faces of these cubes are particular subsets of A. Let $s = \ell + 1$ be the number of points forming a side of the cube, so that the cube is composed by s^n points. A (support) face with dimension $0 \leqslant d \leqslant n$ can be described by choosing indexes $1 \leqslant i_1 < \cdots < i_{n-d} \leqslant n$ and values $v_1, \ldots v_{n-d} \in \{0, \ell\}$, and is defined as

$$F(\ell)^{v_1,\ldots,v_{n-d}}_{i_1,\ldots,i_{n-d}} = \{\alpha \in A \mid \|\alpha\| \leqslant \ell, \quad \alpha_{i_j} = v_j \quad : \quad j = 1, \ldots, n-d\}$$

2.2 Polynomial faces

Consider a polynomial $a(X) \in \mathbf{R}[X]$ and a face $F = F(\|a\|)^{v_1,\ldots,v_{n-d}}_{i_1,\ldots,i_{n-d}}$: we define

Definition 1. *a* polynomial face $a_F(X)$ *of* $a(X)$ *as* $a_F(X) = \sum_{\alpha \in F} a_\alpha X^\alpha$

We indicate with $\mathcal{F}_d$ the set of faces with dimension d (each having $\|a\|^d$ points) and with $\partial(F)$ the dimension of a face F. As we can choose $n-d$ among n indexes and for each of them we have two possibilities for the corresponding v value, we have $^{\#}\mathcal{F}_d = 2^{n-d}\binom{n}{n-d} = 2^{n-d}\binom{n}{d}$. The extreme cases with $d = 0$ and $d = n$ correspond to vertexes and the whole cube, respectively. The total number $\mathbf{F}_n$ of faces is then

$$\mathbf{F}_n = \sum_{d=0}^{n} {}^{\#}\mathcal{F}_d = \sum_{d=0}^{n} 2^{n-d}\binom{n}{d} = \sum_{d=0}^{n} 1^d \cdot 2^{n-d}\binom{n}{d} = (1+2)^n = 3^n$$

Proposition 1. *Polynomial multiplication maps corresponding faces to faces.*
If $a(X) \cdot b(X) = c(X)$, $F_a = F(\|a\|)^{v_1,\ldots,v_{n-d}}_{i_1,\ldots,i_{n-d}}$, $F_b = F(\|b\|)^{v_1,\ldots,v_{n-d}}_{i_1,\ldots,i_{n-d}}$ *and* $F(\|c\|)^{v_1,\ldots,v_{n-d}}_{i_1,\ldots,i_{n-d}}$ *we have*

$$F_a + F_b = F_c \qquad ; \qquad a_{F_a}(X) \cdot b_{F_b}(X) = c_{F_c}(X)$$

Faces are related in a natural way to homogenization based on max-degree definition, that we call multihomogenization.

Definition 2. *Let* $a(X) \in \mathbf{R}[X]$, $H = (h_1, \ldots, h_n)$ *a set of new variables and* $u = (1, \ldots, 1) \in \mathbb{N}^n$. *Let* $Z = (z_1, \ldots, z_{2n})$ *the vector formed by* X *and* H *variables, so that* $x_i = z_i$ *if* $i \leqslant n$ *and* $h_i = z_{n+i}$ *otherwise. The* multihomogenization *of* a *is the below defined polynomial* $a^h(X, H) \in \mathbf{R}[X, H] = \mathbf{R}[Z]$.

$$\mathbf{a}(Z) = a^h(X, H) = \sum_{\alpha \in A} a_\alpha X^\alpha H^{(\|a\|u - \alpha)} = H^{\|a\|u} \sum_{\alpha \in A} a_\alpha \frac{X^\alpha}{H^\alpha}$$

Multidehomogenization corresponds to an evaluation where n *variables, whose indexes* $\{i_1, \ldots, i_n\}$ *have to be specified, are set equal to* 1*:*

$${}^h\mathbf{a}(Z) = \mathbf{a}(z_1, \ldots, z_{i_1-1}, 1, z_{i_1+1}, \ldots, z_{i_j-1}, 1, z_{i_j+1}, \ldots, z_{i_n-1}, 1, z_{i_n+1}, \ldots, z_{2n})$$

Obviously, if $i_j = n + j$ *we have* ${}^h a^h(X, H) = a^h(X, u) = a(X)$.

We note that if we consider $F = F(\|a\|)^{v_1,\ldots,v_{n-d}}_{i_1,\ldots,i_{n-d}}$ and partially evaluate $a^h(X, H)$ setting

$$x_{i_j} = 0 \text{ for all } j \text{ such that } v_j = 0 \qquad ; \qquad h_{i_j} = \begin{cases} 0 & \text{for all } j \text{ such that } v_j = \|a\| \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

we obtain exactly $a_F(X)$. Theoretically speaking, considering polynomial faces is therefore equivalent to consider particular evaluations of (previously multihomogenized) polynomials.

3 Multivariate Karatsuba

Karatsuba multiplication method appeared first in [2] for univariate polynomials and is based on a evaluate-multiply-interpolate (EMI) scheme, which is the core of the process described below in steps. Its extension to the multivariate case is immediate: just consider the isomorphism

$$\mathbf{R}[x_1, \ldots, x_n] \ni a(X) \to \hat{a}(x_n) \in \mathbf{R}[x_1, \ldots, x_{n-1}][x_n]$$

mapping n-variate polynomials to univariate polynomials with $(n-1)$-variate polynomial coefficients.

Recursion on the number of variables is the simplest way to generalize Karatsuba method to the multivariate case, but it weaks the possible use of parallelism. We therefore look for an *iterative* version on the number of variables, so that parallelism can be possibly exploited in a more efficient way.

Before starting the EMI scheme description, we note that one could theoretically compute the product first multihomogenizing factors, then computing the product and finally multidehomogenizing the result:

$$c(X) = (a^h(X, H) \cdot b^h(X, H))\Big|_{H=u}$$

By considering faces, we can "simulate" evaluations on factors multihomogenizations and the final multidehomogenising process, and work in practice with just the original variables X.

Splitting : let $D' = \max\{\|a\|, \|b\|\}$ and $D = \lceil D'/2 \rceil$. Introduce new variables $Y = (y_1, \ldots, y_n)$ with $y_i = x_i^D$ and rewrite the factors as polynomials in y variables with coefficients $\overline{a}_\alpha, \overline{b}_\beta \in \mathbf{R}[X]$, so that $\|\overline{a}_\alpha\| < D$ and $\|\overline{b}_\beta\| < D$:

$$a(X) = \overline{a}(Y) = \sum_{\alpha \in A'} \overline{a}_\alpha Y^\alpha, \qquad ; \qquad b(X) = \overline{b}(Y) = \sum_{\beta \in A'} \overline{b}_\beta Y^\beta,$$

We denote with $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ the multindex with a single 1 entry in the i^{th} position. Now, $\overline{a}$ and $\overline{b}$ are polynomials with max-degree (in Y variables) equal to 1, so that the problem is to consider the product $\overline{c}(Y) = \overline{a}(Y)\overline{b}(Y)$ when $A' = \{\|\alpha\| \leqslant 1\} = \{\alpha \in \{0,1\}^n\}$. Detailing by degree, the first factor becomes then

$$\overline{a}(Y) = \sum_{\alpha \in A'} \overline{a}_\alpha Y^\alpha = \overline{a}_{(0,\ldots,0)} + \sum_{i=1}^{n} \overline{a}_{e_i} y_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \overline{a}_{e_i+e_j} y_i y_j + \cdots + \overline{a}_u \prod_{i=1}^{n} y_i$$

and similarly for $\overline{b}$. Here all multindexes entries for $\overline{a}$ and $\overline{b}$ are either 0 or 1, while for $\overline{c}$ they can be 0, 1 or 2 ($\|\,\overline{c}\,\| = 2$). This means to consider, respectively, the vertexes of n-dimensional hypercubes with side length $s = 2$ and $s = 3$. Note that factors $\overline{a}, \overline{b}$, as they have now the same max-degree, have equal faces.

Evaluation : The product to be computed has 3^n coefficients, so we need to evaluate $\overline{a}$ and $\overline{b}$ in 3^n points. As some particular evaluations for multihomogenized polynomials are in practice nothing but evaluations on original polynomial faces and there are exactly 3^n of them, we may then realize (multihomogenization) evaluation in practice by simply choosing a single evaluation point for each face – e.g. setting to 1 all "surviving" face variables (x_k with $k \neq i_j$ in equation 1).

Recursive multiplication : Compute recursively the 3^n products: $\overline{c}_F(\delta_1, \ldots, \delta_n) = \overline{a}_F(\delta_1, \ldots, \delta_n)\overline{b}_F(\delta_1, \ldots, \delta_n)$ for all faces – where δ_i is 0 or 1, depending on the face.

Interpolation : We now know the values of $\overline{c}$ evaluated in 3^n points. To recover coefficients, we have to solve a classical interpolation problem. Compute them by inverting the matrix corresponding to the evaluation.

Recomposition : Reconstruct explicitly $\overline{c}$ by using its just obtained coefficients.

Many complexity issues rely on evaluation and interpolation phases: the following sections will detail them.

4 Evaluation

The main aim of this and the following section is to minimize the number of needed operations. First of all, note that vertexes $\overline{c}_{(v_1,\ldots,v_n)} = \overline{a}_{(v_1/2,\ldots,v_n/2)}\overline{b}_{(v_1/2,\ldots,v_n/2)}$ – with $v_i \in \{0, 2\}$ for $i = 1, \ldots, n$ – are directly computed, and no evaluation at all is needed. There are 2^n vertexes, so that $E = 2(3^n - 2^n)$ values remain to be computed (the multiplying constant 2 is due to the fact that there are two factors).

It is possible to have just a single sum per factor per face, recycling already computed values for faces with smaller dimension. The idea is very simple: as every evaluated polynomial face $\tilde{a}_F = a_F(1,\dots,1)$ with $\partial(F) = d$ is a sum of 2^d coefficients, it is sufficient to sum two (already evaluated) polynomial sub-faces with dimension $d-1$ forming $\tilde{a}_F$. This implies that faces with smaller dimension have to be evaluated *before* faces with greater dimension. We then need a way to fastly identify faces and their dimensions: in order to do this we'll use base 2 and 3 integer expansions.

Definition 3. *Let $m \in \mathbb{N}$: we indicate the ordered vector of the coefficients of m expansion in base $B \in \mathbb{N}$ with n digits, where $n > \ell = \lfloor \log_B m \rfloor$, as follows*

$$m = \sum_{i=0}^{\ell} m_i B^i \quad \Longrightarrow \quad V_B^n(m) = [m_0, \dots, m_\ell, \underbrace{0, \dots, 0}_{n-1-l}]$$

Vice versa, for a vector $v = [m_0, \dots, m_{n-1}]$ with $0 \leqslant m_i < B$ we define

$$I_B(v) = I_B([m_0, \dots, m_{n-1}]) = \sum_{i=0}^{n-1} m_i B^i$$

so that $\forall\, m,\ \forall\, B$ one has $I_B(V_B^n(m)) = m$.

In the following, everything concerning $\overline{a}$ applies similarly for $\overline{b}$. By using definition 3, we'll use integers as indexes, meaning $\overline{a}_{I_2(\alpha)} = \overline{a}_\alpha$ and $\overline{c}_{I_3(\gamma)} = \overline{c}_\gamma$ every time it makes sense. We suppose that $\overline{a}_\alpha$ ($\overline{c}_\gamma$) coefficients are (will be) memorized in an array $\mathcal{A}$ ($\mathcal{C}$) with 2^n (3^n) entries, such that the entry in position i – starting from 0 – is $\overline{a}_i$ (c_i).

Example 1. For $n = 3$, we have

$$\mathcal{A} = [\,\overline{a}_{000}, \overline{a}_{001}, \overline{a}_{010}, \overline{a}_{011}, \overline{a}_{100}, \overline{a}_{101}, \overline{a}_{110}, \overline{a}_{111}\,] \qquad \mathcal{B} = [\,\overline{b}_{000}, \overline{b}_{001}, \overline{b}_{010}, \overline{b}_{011}, \overline{b}_{100}, \overline{b}_{101}, \overline{b}_{110}, \overline{b}_{111}\,]$$

$$\mathcal{C} = [\,\overline{c}_{000}, \overline{c}_{001}, \overline{c}_{002}, \overline{c}_{010}, \overline{c}_{011}, \overline{c}_{012}, \overline{c}_{020}, \overline{c}_{021}, \overline{c}_{022}, \overline{c}_{100}, \overline{c}_{101}, \overline{c}_{102}, \overline{c}_{110}, \overline{c}_{111}, \overline{c}_{112}, \overline{c}_{120}, \overline{c}_{121}, \overline{c}_{122}, \overline{c}_{200}, \overline{c}_{201}, \overline{c}_{202}, \overline{c}_{210}, \overline{c}_{211}, \overline{c}_{212}, \overline{c}_{220}, \overline{c}_{221}, \overline{c}_{222}\,]$$

As we said, we must evaluate the 3^n faces working by increasing dimension: apart from the 2^n vertexes which need no operation, we have the $n2^{n-1}$ sides, the $n(n-1)2^{n-3}$ bidimensional faces, and so on. One has that for every index $0 \leqslant i < 3^n$ the corresponding face F_i dimension is given by the number of "1" in i expansion in base 3: if $o_{i,j} = V_3^n(i)[j]$ we have

$$\partial(F_i) = \sum_{j=0}^{n-1} o_{i,j}(2 - o_{i,j})$$

To specify an order to analyze faces, we use a multindexes "order" vector ord_n on $\mathcal{C}$. There are many possible orderings, as the only essential requirement is that the order is strict on blocks: for $d_1 < d_2$, all indexes corresponding to faces in $\mathcal{F}_{d_1}$ should be smaller than the ones corresponding to $\mathcal{F}_{d_2}$. Ordering inside each single block $\mathcal{F}_d$ does not matter.

Example 2. For $n = 3$, one possible good ordering for faces is

$ord_3 = [$ [0,0,0], [0,0,2], [0,2,0], [2,0,0], [0,2,2], [2,0,2], [2,2,0], [2,2,2], ⟨vertexes⟩
[0,0,1], [0,2,1], [2,0,1], [2,2,1], [0,1,0], [0,1,2], [2,1,0], [2,1,2], [1,0,0], [1,0,2], [1,2,0], [1,2,2], ⟨sides⟩
[0,1,1], [2,1,1], [1,0,1], [1,2,1], [1,1,0], [1,1,2], ⟨faces⟩
[1,1,1]] ⟨whole cube⟩

We suppose that two vectors $\mathcal{A}'$, $\mathcal{B}'$ with 3^n elements each are available (indexed from 0 to $3^n - 1$). They will contain the evaluations for the two factors. After having computed the vector ord_n, evaluation can begin.

- For the first 2^n entries (vertexes) $\gamma = ord_n[0], \dots, ord_n[2^n - 1]$, simply set

  $$\mathcal{A}'[I_3(\gamma)] = \mathcal{A}[I_2(\gamma/2)] \qquad ; \qquad \mathcal{B}'[I_3(\gamma)] = \mathcal{B}[I_2(\gamma/2)]$$

  Note that here $\gamma/2 \in \mathbb{N}^n$, because γ entries are all 0 or 2, so that we obtain valid multindexes.

- For every other index $i = 2^n, 2^n + 1, \dots, 3^n - 1$, consider $ord_n[i]$ and look for an entry with value 1 in it (as the corresponding face has dimension greater than 0, there is surely at least one). Let's suppose the j^{th} entry is such[1]. We just have to sum the two (already computed) values in positions $I_3(ord_n[i] - e_j)$

[1] As finding base expansion digits is usually done by successive divisions by B and remainders computation, it is convenient to consider the position of the first remainder equal to 1 that comes out.

and $I_3(ord_n[i] + e_j)$. If $h = I_3(ord_n[i])$, this means

$$\mathcal{A}'[h] = \mathcal{A}'[h - 3^j] + \mathcal{A}'[h + 3^j] \qquad ; \qquad \mathcal{B}'[h] = \mathcal{B}'[h - 3^j] + \mathcal{B}'[h + 3^j]$$

We thus have exactly E additions in total. Once evaluation is completed, to obtain the interpolating values it is now sufficient to recursively compute the products

$$\mathcal{C}[i] = \mathcal{A}'[i] \cdot \mathcal{B}'[i] \qquad \text{for all } i = 0, \ldots, 3^n - 1$$

4.1 Temporary variables number reduction

We note that it is not difficult to avoid one temporary vector, say $\mathcal{B}'$. One can save the second factor evaluations directly in $\mathcal{C}$, with indexes shifted by 1, so that $\mathcal{C}[i]$ will contain the value $\mathcal{B}'[i+1]$ for $i = 0, \ldots, 3^n - 2$. Evaluation should of course take care of this indexes shift and refer directly to $\mathcal{B}$ for $\bar{b}_0$ accessing. The recursive multiplications should finally be executed in a top-down style:

$$\begin{aligned} &\mathcal{C}[i] = \mathcal{A}'[i] \cdot \mathcal{C}[i-1] \qquad \text{for all } i = 3^n - 1, \ldots, 1 \\ &\mathcal{C}[0] = \mathcal{A}[0] \cdot \mathcal{B}[0] \end{aligned}$$

Working this way, the extra needed space is halved to just 3^n extra temporaries. For small values of n one can write explicit code taking direct benefit of *all* entries corresponding to vertexes, which can be the last ones to be filled, and be therefore used in intermediate computations as temporary variables.

5 Interpolation

In this phase the product coefficients are recovered by working on the just computed values in $\mathcal{C}$. The idea is someway dual with respect to evaluation phase: as we there had to start from lower-dimensional faces values in order to obtain all other ones, we start here from values corresponding to higher-dimensional faces. What is done here is essentially a clever application of inclusion-exclusion principle, taking benefit of already computed intermediate values.

The plain inclusion-exclusion principle requires $3^d - 1$ operations for every face with dimension d. Summing over all faces, we have that one would in total need

$$I_{incl.-excl.} = \sum_{d=0}^{n} {}^{\#}\mathcal{F}_d \cdot (3^d - 1) = \sum_{d=0}^{n} \binom{n}{d} 2^{n-d} 3^d - \sum_{d=0}^{n} \binom{n}{d} 2^{n-d} = (2+3)^n - (2+1)^n = 5^n - 3^n$$

operations (additions/subtractions). We instead reason by recycling faces values obtained along the way.

A face F with $\partial(F) = d > 1$ has $2d$ subfaces with dimension $d-1$. Loosely speaking, if we "remove" all $(d-1)$-subfaces, only the corresponding correct value will remain. This removing process corresponds in practice to successive subtractions of "intermediate" values as soon as they are available. The total number of needed subtractions is therefore much smaller than before:

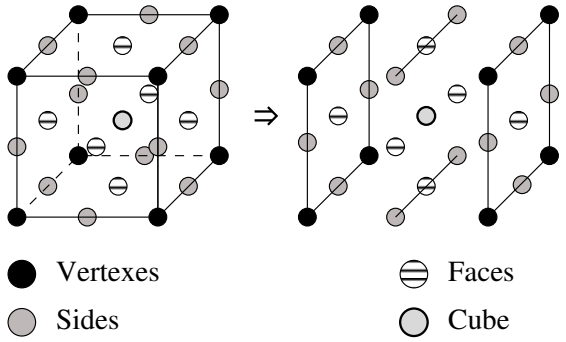
$$I = \sum_{d=1}^{n} 2d \cdot {}^{\#}\mathcal{F}_d = 2\sum_{d=1}^{n} d 2^{n-d} \binom{n}{d} = 2\sum_{d=0}^{n-1} 2^d (n-d) \frac{n!}{d!(n-d)!} =$$

$$= 2\sum_{d=0}^{n-1} 2^d \frac{n(n-1)!}{d!(n-d-1)!} = 2n \sum_{d=0}^{n-1} \binom{n-1}{d} 2^d = 2n(1+2)^{n-1} = 2n \cdot 3^{n-1}$$

The idea to organize subtractions is the following: starting from dimension $d = n-1$ down to 1, let $F \in \mathcal{F}_{d'}$ a face having higher dimension $d' = d+1, \ldots, n$, and let i be its index position in $\mathcal{C}$. We consider the "distance" $D = d' - d$: we must subtract from $\mathcal{C}[i]$ the values corresponding to the two (sub-)faces of F characterized by having 0 and 2, respectively, as entries in the position corresponding to the D^{th} entry of $V_3^n(i)$ (starting to count always from the same side) whose value is 1, and all other entries equal.

Example 3. For $n = 3$, we obtain the reported sequence of interpolation operations (for brevity, we don't indicate I_3 function to convert multindexes into integers indexing $\mathcal{C}$). "Blocks" of operations are separated by double lines.

1. The first block only consists in removing two faces from the whole cube.
2. In the second block, sides are appropriately subtracted from all faces (first section) and two "partial" faces are subtracted from the cube (second section).
3. In the last block, vertexes are subtracted from all sides (first section) "partial" sides are subtracted from all faces (second section) and the last two "partial" faces from the whole cube (very last line).



1
$$\mathcal{C}[1,1,1] \leftarrow \mathcal{C}[1,1,1] - \mathcal{C}[0,1,1] - \mathcal{C}[2,1,1]$$

2
$$\mathcal{C}[0,1,1] \leftarrow \mathcal{C}[0,1,1] - \mathcal{C}[0,0,1] - \mathcal{C}[0,2,1]$$
$$\mathcal{C}[2,1,1] \leftarrow \mathcal{C}[2,1,1] - \mathcal{C}[2,0,1] - \mathcal{C}[2,2,1]$$
$$\mathcal{C}[1,0,1] \leftarrow \mathcal{C}[1,0,1] - \mathcal{C}[0,0,1] - \mathcal{C}[2,0,1]$$
$$\mathcal{C}[1,2,1] \leftarrow \mathcal{C}[1,2,1] - \mathcal{C}[0,2,1] - \mathcal{C}[2,2,1]$$
$$\mathcal{C}[1,1,0] \leftarrow \mathcal{C}[1,1,0] - \mathcal{C}[0,1,0] - \mathcal{C}[2,1,0]$$
$$\mathcal{C}[1,1,2] \leftarrow \mathcal{C}[1,1,2] - \mathcal{C}[0,1,2] - \mathcal{C}[2,1,2]$$
$$\mathcal{C}[1,1,1] \leftarrow \mathcal{C}[1,1,1] - \mathcal{C}[1,0,1] - \mathcal{C}[1,2,1]$$

3
$$\mathcal{C}[0,0,1] \leftarrow \mathcal{C}[0,0,1] - \mathcal{C}[0,0,0] - \mathcal{C}[0,0,2]$$
$$\mathcal{C}[2,0,1] \leftarrow \mathcal{C}[2,0,1] - \mathcal{C}[2,0,0] - \mathcal{C}[2,0,2]$$
$$\mathcal{C}[0,2,1] \leftarrow \mathcal{C}[0,2,1] - \mathcal{C}[0,2,0] - \mathcal{C}[0,2,2]$$
$$\mathcal{C}[2,2,1] \leftarrow \mathcal{C}[2,2,1] - \mathcal{C}[2,2,0] - \mathcal{C}[2,2,2]$$
$$\mathcal{C}[0,1,0] \leftarrow \mathcal{C}[0,1,0] - \mathcal{C}[0,0,0] - \mathcal{C}[0,2,0]$$
$$\mathcal{C}[2,1,0] \leftarrow \mathcal{C}[2,1,0] - \mathcal{C}[2,0,0] - \mathcal{C}[2,2,0]$$
$$\mathcal{C}[0,1,2] \leftarrow \mathcal{C}[0,1,2] - \mathcal{C}[0,0,2] - \mathcal{C}[0,2,2]$$
$$\mathcal{C}[2,1,2] \leftarrow \mathcal{C}[2,1,2] - \mathcal{C}[2,0,2] - \mathcal{C}[2,2,2]$$
$$\mathcal{C}[1,0,0] \leftarrow \mathcal{C}[1,0,0] - \mathcal{C}[0,0,0] - \mathcal{C}[2,0,0]$$
$$\mathcal{C}[1,2,0] \leftarrow \mathcal{C}[1,2,0] - \mathcal{C}[0,2,0] - \mathcal{C}[2,2,0]$$
$$\mathcal{C}[1,0,2] \leftarrow \mathcal{C}[1,0,2] - \mathcal{C}[0,0,2] - \mathcal{C}[2,0,2]$$
$$\mathcal{C}[1,2,2] \leftarrow \mathcal{C}[1,2,2] - \mathcal{C}[0,2,2] - \mathcal{C}[2,2,2]$$
$$\mathcal{C}[0,1,1] \leftarrow \mathcal{C}[0,1,1] - \mathcal{C}[0,1,0] - \mathcal{C}[0,1,2]$$
$$\mathcal{C}[2,1,1] \leftarrow \mathcal{C}[2,1,1] - \mathcal{C}[2,1,0] - \mathcal{C}[2,1,2]$$
$$\mathcal{C}[1,0,1] \leftarrow \mathcal{C}[1,0,1] - \mathcal{C}[1,0,0] - \mathcal{C}[1,0,2]$$
$$\mathcal{C}[1,2,1] \leftarrow \mathcal{C}[1,2,1] - \mathcal{C}[1,2,0] - \mathcal{C}[1,2,2]$$
$$\mathcal{C}[1,1,0] \leftarrow \mathcal{C}[1,1,0] - \mathcal{C}[1,0,0] - \mathcal{C}[1,0,0]$$
$$\mathcal{C}[1,1,2] \leftarrow \mathcal{C}[1,1,2] - \mathcal{C}[1,0,2] - \mathcal{C}[1,0,2]$$
$$\mathcal{C}[1,1,1] \leftarrow \mathcal{C}[1,1,1] - \mathcal{C}[1,1,0] - \mathcal{C}[1,1,2]$$

In appendix A a (very naive) implementation in `gp-pari` is provided.

6 Conclusions

We showed how Karatsuba method for multivariate polynomials can be implemented iteratively on the number of variables, by using integers base 2 and 3 expansions in order to access the needed data.

7 Acknowledgements

This work is dedicated to the memory of the author's primary school teacher, Giampiero Carlini.

References

1. Serdar S. Erdem and Çetin Kaya Koç. A less recursive variant of Karatsuba-Ofman algorithm for multiplying operands of size a power of two. In *IEEE Symposium on Computer Arithmetic*, pages 28–35. IEEE Computer Society, 2003.
2. Anatolii Alexeevich Karatsuba and Yuri Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7(7):595–596, 1963.
3. M. Kounavis. A new method for fast integer multiplication and its application to cryptography. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2007.
4. Robert T. Moenck. Practical fast polynomial multiplication. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 136–148, New York, NY, USA, 1976. ACM Press.
5. Peter L. Montgomery. Five, six, and seven-term Karatsuba-like formulae. *IEEE Trans. Comput.*, 54(3):362–369, 2005.
6. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3–4):281–292, 1971.
7. André Weimerskirch and Christof Paar. Generalizations of the Karatsuba algorithm for polynomial multiplication. Technical report, Ruhr-Universität-Bochum, 2003.

A Multivariate Karatsuba code

We provide some functions written in `gp-pari`, implementing multivariate iterative Karatsuba algorithm.

This is a "service" function : it outputs a vector of symbolic coefficients representing a polynomial factor with max-degree equal to 1. The second argument indicates which symbolic letter is used for coefficients.

```
coeffSquarePoly(n,a = "a") =
{
  local (tmp = vector(2^n,i, concat(vector(n - matsize(binary(i-1))[2],i,0), binary(i-1))) );

  for (i=1,2^n, str = "";
                for (j = 1,n,  str = Str(str,tmp[i][j]));
                tmp[i] = eval(Str(Str(a),str));
  );
 return (tmp);
}
```

Indexes conversion: to base 2 and from base 3.

```
intToBase2(n, vectorLength) =
{
 local(res = vector(vectorLength),
       j = vectorLength + 1, tmp = 1);

 while(j>1, res[j--] = if(bitand(n,tmp), 1, 0);
            tmp <<= 1;);
 return (res);
}
```

```
base3toInt(v) =
{
 local(i, res = 0);

 forstep(i = matsize(v)[2],1,-1, res *= 3;
                                 res += v[i];
 );
 return (res);
}
```

Position of the i^{th} rightmost 1 in vector `v`.

```
iThRightmostOnePos(v, i = 1) =
{
  local(res = matsize(v)[2], position = i+1);
  while( position--,  while( (res >= 1) && (v[res] != 1), res--; );
                      res--;
  );
  return(res);
}
```

Computation of vectors with n elements with exactly $k \leqslant n$ entries equal to 1 and the others equal to 0 or 2.

```
vecSizeNwithK1(n,k) =
{
 local(res, tmp1,tmp2, dim1, dim2, i, j = 0, h);
 if ( k == 0,   res = matrix(2^n,n);
                forvec(x=vector(n,i,[0,1]), res[j++,] = vector(n,h,2*x[h]));
 , \\ else
   if ( n == k, res = matrix(1,n,i,j,1);
   , \\ else
    tmp1 = vecSizeNwithK1(n-1,k);   \\ First element: 0
    tmp2 = vecSizeNwithK1(n-1,k-1); \\ First element: 1
    dim1 = matsize(tmp1)[1]; dim2 = matsize(tmp2)[1];
    res = matrix(2*dim1 + dim2, n);
    for(i=1,dim2,  for(j=1,n-1, res[i,j]      = tmp2[i,j]; res[i,n]      = 1;););
    for(i=1,dim1,  for(j=1,n-1, res[i+dim2,j] = tmp1[i,j]; res[i+dim2,n] = 0;););
    dim2 += dim1;
    for(i=1,dim1,  for(j=1,n-1, res[i+dim2,j] = tmp1[i,j]; res[i+dim2,n] = 2;););
  ));
  return(res);
}
```

Compute the vector ord_n. Vertexes are not considered: they are treated apart in the multiplication function.

```
ordIndexes(n) =
{
  local(res = matrix(3^n, n), i, j, k = 0, tmp);
  for(i=1,n,  tmp = vecSizeNwithK1(n,i);
              for(j=1,matsize(tmp)[1],   res[k++,] = tmp[j,];
  ));
  return(res);
}
```

Data for $\mathcal{F}_d$'s management.

```
initBlocks(n) =
{
 local(res = vector(n,i,vector(2)), k);
 res[1] = [1,n<<(n-1)];
 for(k = 2, n,  res[k] = [res[k-1][1] + res[k-1][2], binomial(n,k)<<(n-k)]; );
 return(res);
}
```

Subtract from faces their subfaces having 0 and 2 in correspondence with the `posOne`th entry whose value is 1 starting from the right end.

```
subFaces(v, indexes, base, limit, posOne) =
{
  local(i, index, offset);
  for(i = base, base + limit - 1,   index  = base3toInt(indexes[i,]) + 1;
                                    offset = 3^(iThRightmostOnePos(indexes[i,], posOne));
                                    v[index] -= (v[index-offset] + v[index+offset]););
  return(v);
}
```

This is the complete function. The arguments `a` and `b` are two vectors with the coefficients of the two factors.

```
multivariateIterativeKaratsuba(a, b) =
{
  local(n   = ceil(log(matsize(a)[2])/log(2)), resA = vector(3^n), resB = vector(3^n),
        res = vector(3^n), indexes = ordIndexes(n), i, sections = initBlocks(n) );

  \\\\\\\\\\\\\\\\\\ Evaluation
  \\ Vertexes first (no additions, just copies)...
  for(i = 1, 2^n, index = base3toInt(intToBase2(i-1,n))<<1 + 1;
                  resA[index] = a[i]; resB[index] = b[i]; );
  \\ ...then all the rest.
  for(i = 1, 3^n-2^n,  index  = base3toInt(indexes[i,]) + 1;
                       offset = 3^iThRightmostOnePos(indexes[i,],1);
                       resA[index] = resA[index-offset] + resA[index+offset];
                       resB[index] = resB[index-offset] + resB[index+offset];
  );
  \\\\\\\\\\\\\\\\\\ Products : here with simple *, but they should be recursive.
  for(i = 1, 3^n, res[i] = resA[i]*resB[i]);
  \\\\\\\\\\\\\\\\\\ Interpolation.
  forstep(i = n, 1, -1, for( j = i, n,
    res = subFaces(res, indexes, sections[j][1], sections[j][2], j-i+1 );
  ));
  return(res);
}
```

Here is an example code:

```
n = 3;  a = coeffSquarePoly(n,"a");  b = coeffSquarePoly(n,"b");
print(multivariateIterativeKaratsuba(a,b));
```