2011 20th IEEE Symposium on Computer Arithmetic

High degree Toom'n'half for balanced and unbalanced multiplication

Marco Bodrato
mambaSoft
Torino, Italy
Email: bodrato@mail.dm.unipi.it

***Abstract*—Some hints and tricks to automatically obtain high degree Toom-Cook implementations, i.e. functions for integer or polynomial multiplication with a reduced complexity. The described method generates quite an efficient sequence of operations and the memory footprint is kept low by using a new strategy: mixing evaluation, interpolation and recomposition phases. It is possible to automatise the whole procedure obtaining a general Toom-n function, and to extend the method to polynomials in any characteristic except two.**

***Keywords*-integer multiplication; polynomial product; Toom-Cook; unbalanced; code generation.**

I. Introduction

The family of so-called Toom-Cook [1], [2] methods is an infinite set of algorithms for polynomial multiplication, all of them are sub-quadratic and can be applied to long integer multiplication too. Many algebraic libraries and programs implement some Toom-methods [3], [4], even if asymptotically better algorithms exist [5], [6]. There usually is a wide range where Toom-strategy is the fastest, and recent papers by Zanoni moved the interest towards higher degree [7] methods and strong unbalancement [8].

In two previous papers by the author [10] with Zanoni [9], some techniques are described in order to find optimal sequences for the linear computations needed in the Toom's algorithms. The model was good enough to give new sequences, using fewer operations than the ones implemented before, but the model was also very restricted and rigid, and it didn't cover additional tricks that can speed up computations.

A new strategy is proposed, to achieve a better performance and a smaller memory footprint by mixing the traditional phases: evaluation, interpolation and recomposition. A similar approach was independently proposed by Bernstein for polynomial multiplication in $\mathbb{F}_2$ [11], but it is in some sense too general, and does not give a concrete help in actually generating the sequence of operations.

The goal of this paper is giving a general method to generate code for integer or polynomial multiplication using very high degree Toom-Cook methods. The method gives reasonably efficient code and is really straightforward, so that it can be made fully automatic. It is even possible to implement a generic Toom-n function, with only a few restrictions on n. Moreover the method can be generalised to work with any characteristic, except 2.

Applications

When this paper was written, a couple of functions for long integer multiplication using the technique described in this paper where already implemented within the version 5.0 of the GMP library [3], namely `toom6h` and `toom8h`, they work with a wide range of operand size, because of their efficiency, and with many possible unbalancement ratios using the technique described in §VI. Next step could be to write a code generator to implement `toom10h`, `toom12h` and so on, or a generic function choosing the splitting order and computing the sequence on the fly.

The generalisation in §VII allows to extend the use of this technique to polynomials too.

II. The Toom-Cook algorithm

The first implementation of the algorithm described in this paper was coded by the author for integer multiplication within the Gnu Multi-Precision library. Nevertheless, we will recall how the Toom's algorithm works for polynomials, because the description is simpler.

A. The general multiplication method for polynomials

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 $m = n$. We generalise to unbalanced ones, we assume $n \geq m > 1$, and we call the method Toom-$\frac{n+m}{2}$.

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 $\alpha_i, \beta_i \in \mathbb{R}[x]$.

The evaluation of a single polynomial (for example $\mathfrak{u}$) on the points (α_i, β_i), can be computed with a matrix by

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.12

IEEE computer society

vector multiplication. The matrix $E_{d,n}$ is a $(d+1) \times n$ Vandermonde-like matrix. $\overline{\mathfrak{u}(\alpha,\beta)} = E_{d,n}\overline{u} \Longrightarrow$

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

In the following, we will sometimes specify points as a couple (α,β), some other times we will indicate them with a single number, being the ratio $\frac{\alpha}{\beta}$, as a special case, $(1,0)$ is sometimes called ∞.

Recursive multiplication : We compute $\forall i, \mathfrak{w}(\alpha_i,\beta_i) = \mathfrak{u}(\alpha_i,\beta_i)\mathfrak{v}(\alpha_i,\beta_i)$, with $d+1$ multiplications of polynomials whose degree is less than or equal to that of $Y = x^b$.

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

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

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 additions.

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 require many additions and subtractions, shifts, and even small multiplications or exact divisions (interpolation only) by small elements in $\mathbb{R}[x]$.

Splitting apart, because it is only a *virtual* step without any actual computation, we will propose to mix all the phases, interlacing operations, with the goal of reducing both memory usage and the total number of operations in $\mathbb{R}$.

B. Available operations, and costs

In the following descriptions of the algorithms we will assume that a wide range of in-place operations are available on any couple of operands $A, B \in \mathbb{R}[x]$: $A \leftarrow A+B, A \leftarrow A-B, A \leftarrow B-A, A \leftarrow A+\beta B \ldots$. Basically we may need any general linear combination

$$A \leftarrow \alpha A + \beta B \tag{3}$$

with $\alpha, \beta \in \mathbb{R}[x]$ and $\alpha = \pm 1$ or $\beta = \pm 1$. We also need exact divisions, that may be shifts, depending on the divisor δ:

$$A \leftarrow A/\delta. \tag{4}$$

With exact division we mean that we know in advance that there will be no reminder. The only exception to exactness may be found in §IV-E where we propose the use of shifts clearing a non-zero remainder.

To simplify the analysis we will only count the number of operations, i.e. the number of times we need to write results.

Moreover we will use some kind of simultaneous operations:

$$(A,B) \leftarrow (A \pm B, B \mp A)\ , (A,B) \leftarrow (\frac{A \pm B}{2}, \frac{A \mp B}{2})\ . \tag{5}$$

We will count them as two operations.

III. Symmetries: a reason to prefer Toom'n'half

The first and main symmetry we can use is the one involving positive and negative evaluation points, i.e. opposite points. The second one consists in using a point (α,β) and its inverse (β,α).

A. Opposite points

The evaluation of a polynomial $\mathfrak{p}$ with degree d in a point α costs d additions and the same number of multiplications by (powers of) α. The contextual evaluation of $\mathfrak{p}$ in two points $\pm\alpha$ costs $d+1$ additions and d multiplications, because we can separately add the even and the odd parts of the polynomial, then obtain the required evaluations by computing both the sum and the difference.

This means that the evaluation cost of adding a new evaluation point α when we already have $-\alpha$ is really small.

We can obtain the interpolation by a sequence of operation on lines of the matrix seen in equation (2): the more entries we nullify in each step, the shorter the sequence [9]. The lines generated by two opposite evaluations share the same absolute values, all positive on one line, alternated on the other one:

$$\begin{pmatrix} L_p \\ L_n \end{pmatrix} = \begin{pmatrix} \beta^d & \alpha\beta^{d-1} & \alpha^2\beta^{d-2} & \cdots & \alpha^d \\ \beta^d & -\alpha\beta^{d-1} & \alpha^2\beta^{d-2} & \cdots & (-\alpha)^d \end{pmatrix}\ . \tag{6}$$

By adding and subtracting the two lines (and divide by two) we obtain the following lines, where half the entries are zeroed:

$$\begin{pmatrix} 0 & \alpha\beta^{d-1} & 0 & \alpha^3\beta^{d-3} & \cdots \\ \beta^d & 0 & \alpha^2\beta^{d-2} & 0 & \cdots \end{pmatrix}\ . \tag{7}$$

The great advantage given by the presence of opposite points suggests us to always use couples of points. This means that, with the two exceptional points $(1,0)$ and $(0,1)$, often called zero and infinity, we want an even number of points. For the usual balanced Toom-n, the number of points required is $d+1 = 2n-1$, always odd; that's the first reason why we will study the unbalanced Toom'n'half: Toom-$n+\frac{1}{2}$, requiring $2n$ points. In §VI we will explain how the results in this paper can be extended also to the widely used balanced product.

B. Inverses

The use of inverses is not as effective as the use of opposites, but can give some advantage anyway.

If we evaluate in (α, β) and its inverse (β, α), we will find in the matrix of equation (2) two symmetric lines with exactly the same values and signs, but with the opposite order:

$$\begin{pmatrix} \beta^d & \alpha\beta^{d-1} & \cdots & \alpha^{d-1}\beta & \alpha^d \\ \alpha^d & \alpha^{d-1}\beta & \cdots & \alpha\beta^{d-1} & \beta^d \end{pmatrix} . \tag{8}$$

By adding and subtracting the two lines we obtain the following lines, one is symmetric, the other one is antisymmetric::

$$\begin{pmatrix} \alpha^d + \beta^d & \alpha^{d-1}\beta + \alpha\beta^{d-1} & \cdots & \beta^d + \alpha^d \\ \alpha^d - \beta^d & \alpha^{d-1}\beta - \alpha\beta^{d-1} & \cdots & \beta^d - \alpha^d \end{pmatrix} .$$

If all the lines in the matrix of equation (2) do have their own inverse, we can obtain a new matrix that we can block-wise represent as:

$$\left(\begin{array}{c|c} A & A' \\ \hline -B & B' \end{array}\right) ,$$

where A' and B' are the mirrored matrices of A and B, respectively. We ignored the two exceptional points $(1, 0)$ and $(0, 1)$, that can be handled separately. The two regular ones $(1, 1)$ and $(-1, 1)$ give symmetric lines and should be included in A, A' with no need for pre-computations.

In the following steps, when we operate with lines in A, every zeroed entry in A has a mirrored entry zeroed with the same operation in A'. The same happens with B and B'. This way, the zeroing effect of the following operations is doubled.

C. An estimate of Toom'n'half cost exploiting symmetries

To have all the symmetries, we need quartets of points: $\pm\alpha, \pm\alpha^{-1}$, plus the four "standard" $0, \pm 1, \infty$; that's why we will analyse in full details the Toom-$2k + \frac{1}{2}$ methods: $4k$ evaluation points.

Here we give a first estimate, we count only linear combinations, regardless of their complexity. We start from evaluation.

The two extremal points $0, \infty$ do not require any operation for evaluation. The two operands must be represented by two polynomials with degree n and m so that $n + m - 1 = 4k$, this means that the evaluation on every couple $\pm\alpha$ costs $(n+1)+(m+1) = 4k+1$. There are $2k-1$ couples. Therefore the evaluation require $(4k + 1) \cdot (2k - 1)$ combinations.

We will support the explanation of the interpolation phase with an example: the matrix used by Toom-$4 + \frac{1}{2}$ with the four "standard" evaluation points and $\pm 2, \pm\frac{1}{2}$. The initial matrix is:

$$M_4 = \left(\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 \\ 1 & -2 & 4 & -8 & 16 & -32 & 64 & -128 \\ \hdashline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ \hdashline 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ 128 & -64 & 32 & -16 & 8 & -4 & 2 & -1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) .$$

The interpolation phase will start with exploiting opposite points, this means computing the sum and the difference for each couple of opposite lines: one operation for each one of the $4k - 2$ lines. In the example we performed some more shifts (divisions by 2^e) to remove any common factor from entries in a line:

$$M_4' = \left(\begin{array}{c|c|c|c|c|c|c|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 4 & 0 & 16 & 0 & 64 \\ 1 & 0 & 4 & 0 & 16 & 0 & 64 & 0 \\ \hdashline 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hdashline 0 & 64 & 0 & 16 & 0 & 4 & 0 & 1 \\ 64 & 0 & 16 & 0 & 4 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) .$$

Now, half the lines have a non-zero first entry and half the lines have a non-zero last one, we can clear them with a combination with the first or the last line respectively: $4k-2$ operations. In the example we obtain (with implied shifts):

$$M_4'' = \left(\begin{array}{c|cc:cc:cc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \mathbf{1} & 0 & \mathbf{4} & 0 & \mathbf{16} & 0 & 0 \\ 0 & 0 & 1 & 0 & 4 & 0 & 16 & 0 \\ \hdashline 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hdashline 0 & \mathbf{16} & 0 & \mathbf{4} & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 16 & 0 & 4 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \rightsquigarrow \begin{pmatrix} 1 & 4 & 16 \\ 1 & 1 & 1 \\ 16 & 4 & 1 \end{pmatrix} .$$

The example shows one more structure we can use, a smaller $(2k - 1) \times (2k - 1)$ Vandermonde's matrix, representing both sub-matrices of even and odd rows. In §IV we will explain how to exploit it, now we simply use this compact representation to jump to a bigger example, the matrix coming from Toom-$6 + \frac{1}{2}$, using the values 2 and 4:

$$\widetilde{M_6''} = \begin{pmatrix} 1 & 16 & 256 & 4096 & 65536 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 1 & 1 & 1 & 1 \\ 256 & 64 & 16 & 4 & 1 \\ 65536 & 4096 & 256 & 16 & 1 \end{pmatrix} .$$

We started with an interpolation problem with points $\pm\alpha_0, \pm\alpha_1, \ldots$, and we reduced it to a smaller one for half the values: $\alpha_0^2, \alpha_1^2, \ldots$. We can solve it the way we prefer.

Next step we propose consists in exploiting inverses, again a sum and a difference for each couple (remember, each line

in the example represents two rows), skipping ± 1, $2(2k-2)$ operations more.

$$\widetilde{M}_6''' = \left(\begin{array}{ccc|cc} 65537 & 4112 & \mathbf{512} & 4112 & 65537 \\ 257 & 68 & \mathbf{32} & 68 & 257 \\ 1 & 1 & \mathbf{1} & 1 & 1 \\ \hline -255 & -60 & \mathbf{0} & 60 & 255 \\ -65535 & -4080 & \mathbf{0} & 4080 & 65535 \end{array}\right).$$

We obtained two sub-matrices, a bigger $k \times k$ one and a smaller $(k-1)\times(k-1)$ one. None of them is Vandermonde's, but we can still use the estimate given in the work with Zanoni[9] to compute a maximal number of operations needed to finish interpolation.

$$\widetilde{M}_6'''' = \left(\begin{array}{ccc|cc} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{array}\right).$$

To obtain the (almost) final matrix we had to nullify $k \cdot (k-1) + (k-1) \cdot (k-2)$ entries, each of them representing two values in the original matrix, thus requiring two operations (see §V for full details on the sequence). We reach the identity with a final $(L_u, L_d) \leftarrow (\frac{L_u - L_d}{2}, \frac{L_u + L_d}{2})$ on couples of lines, $2(2k-2)$ operations again.

If we add up all the costs we obtain, for Toom-$2k+\frac{1}{2}$:
- evaluation: $(4k+1)\cdot(2k-1)$ linear combinations;
- interpolation: $2((4k-2)+2(2k-2)+k(k-1)+(k-1)(k-2))$
$= (4k^2+8k-8)$ combinations (and $(4k-4)$ divisions).

D. Estimate adapted to Toom-n

When $n = 2k$ we can use exactly the sequence proposed above, removing one of the two points 0 or ∞.

The evaluation is basically the same, but we don't have one operand that is longer than the other one: n operations times $n-1$ couples for each of the two operands (only once for squaring).

During interpolation we only save half the operations from M'_{2k} to M''_{2k}, due to the removal of one of the external columns: we save $\frac{4k-2}{2} = n-1$.

This estimates gives, for Toom-n with $n = 2k$:
- evaluation: $2 \cdot n(n-1)$ linear combinations;
- interpolation: n^2+3n-7 combinations, $(2n-4)$ divisions.

IV. Mixing all Phases

We propose to mix-up the phases with the goal of reducing both memory needs and the number of required operations.

We start by observing the matrix M'_4 used for the estimation in §III-C. For each Toom'n'half we will obtain a similar one.

Theorem 1: Two lines in the interpolation matrix of a Toom'n'half generated by opposite points $(\pm\alpha, \beta)$ can generate, by linear combinations with the first and last line, two lines with zeroed first and last column, and the same entries, shifted by one position.

Proof: From (6), by using $L_n \leftarrow \frac{L_p+L_n}{2}; L_p \leftarrow L_p - L_n$, we obtain (7). With $L_n \leftarrow \frac{L_n - \beta^d(1,0\cdots0)}{\alpha^2\beta}, L_p \leftarrow \frac{L_p - \alpha^d(0\cdots0,1)}{\alpha\beta^2}$ we obtain the two linearly independent lines:

$$\begin{pmatrix} 0 & \beta^{d-3} & 0 & \alpha^2\beta^{d-5} & 0 & \cdots & 0 \\ 0 & 0 & \beta^{d-3} & 0 & \alpha^2\beta^{d-5} & \cdots & 0 \end{pmatrix}. \quad (9)$$

■

A. Early recomposition

The last phase described in §II is recomposition, one of its effects is to reduce memory usage. This is true both when Toom is used for integers, and when it is used for polynomials and the base $Y = x^b$ used has an exponent $b > 1$.

Using the same notation as in §II, we can say that the original coefficients u_i, v_i after splitting are smaller than the base Y, while the values we handle during the interpolation phase approximatively have the size of Y^2. In practice recomposition consists in adding halves of the obtained values, as shown in fig. 1, to recompose the final product.

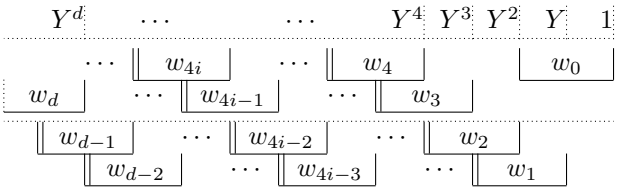

Figure 1. Recomposition for Toom-$2k+\frac{1}{2}$, degree $d = 4k-1$

The trick we propose here consists in anticipating parts of the recomposition during the interpolation phase. In particular, as soon as all pairs of lines w_j, w_{j+1} are in the form described by equation (9), we can recompose all pairs at once obtaining the new values $W_j = Yw_{j+1} + w_j$. Their size approximatively is the size of Y^3.

After partial recomposition an operation between the new values W will be equivalent to the same operation on couples of lines; e.g. $W_i + \beta W_j = Y(w_{i+1} + \beta w_{j+1}) + (w_i + \beta w_j)$.

B. Even better estimate for Toom'n'half

Thanks to the early recomposition, we can rerun the estimate in §III-C taking into account the fact that from matrix M'' on we can use the new partially-recomposed values W_i.

For a correct estimate we must consider the size of the operands. Without specifying the "unit", we use b as the size of initial coefficients u_i, v_i; the size of multiplied values, used during interpolation, is $2b$; finally we count $3b$ for the

size of partially recomposed W_i. The costs of the operations used during evaluation and interpolation are linear in the size of the operands. We will ignore the constants, and we will count only one thing: how many times we operate on "atomic" portions of operands. We won't even take care of the extra length needed by carries and other border effects, not relevant with big sizes.

When we obtain the matrix M'', we perform early recomposition. We do not count the cost of it, because it is perfectly compensated by a correspondent reduction in the final recomposition (compare fig. 1 and fig. 2). After it, the operations on the reduced $\widetilde{M}$ matrix will not operate on couples of rows, but on the new values W.

If we add again the numbers computed in §III-C we obtain:

- evaluation: $b \cdot (4k+1) \cdot (2k-1)$ atomic operations;
- interpolation up to M'': $2b \cdot 2(4k-2)$
- from M'': $3b \cdot (2(2k-2) + k(k-1) + (k-1)(k-2))$.

The total for interpolation is:

$$2b(3k^2+8k-7) \text{ combinations } 2b(3k-3) \text{ divisions.} \quad (10)$$

C. Comparison with previous results

We can adjust equation (10) the same way we did in §III-D. We obtain, with the use of early recomposition, a cost for the interpolation required by Toom-n of $2b(3k^2 + 6k - 6) = 2b(\frac{3}{4}n^2 + 3n - 6)$ combinations and $2b(\frac{3}{2}n - 3)$ divisions.

The table I compares the number of operations needed by interpolation. The sequences generated by the present work are compared with the best results published so far. We count the number of operations on $2b$ long operands.

	this paper		previous result		
n	comb.	div.	comb.	div.	reference
2	3	0	2	0	Karatsuba[12]
4	18	3	18	3	Bodrato and Zanoni[9]
4.5	21	3	22	4	"
6	39	6	46	7	Bodrato, using[9]
8	66	9	87	11	Zanoni[7]

Table I
COMPARING NEW RESULTS WITH PREVIOUS STATE OF THE ART

Even if we did not yet specify the step by step sequences to handle the two sub-matrices of $\widetilde{M}'''$, we already know that, starting from Toom-$4 + \frac{1}{2}$, the new automatically generated sequences will need fewer steps than the ones published so far, even if some of them where carefully designed. The winning strategy is the use of symmetries and early recomposition.

One more side effect of early recomposition is reducing the impact of divisions on running time.

D. Memory optimisation

We will not get deeply into details, but we can claim that an even stronger mix of phases allows some memory saving.

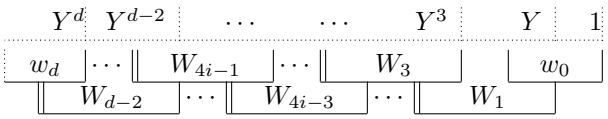

Figure 2. Memory usage with early recomposition, $W_j = Yw_{j+1} + w_j$

Theorem 2: The extra memory needed by Toom-$2k + \frac{1}{2}$ does not exceed the size of the result plus some $O(b)$.

Proof: Fig. 2 gives some hint about how to store some of the early recomposed values in the result area. Values $w_0, W_3, \cdots, W_{4i-1}, \cdots, w_d$ can be stored in the result area of size $b(d+2)$, without any risk of overlapping.

The remaining $\frac{d+1}{4}$ values $W_1, \cdots, W_{4i-3}, \cdots$ can be stored apart, each using $3b + O(1)$ memory units (carries included). Total size $\frac{d+1}{4}(3b + O(1)) < b(d+2)$.

This means that we need a strategy to avoid the generation of the full list of values required by interpolation (2). It consists in handling quartets of points $\pm\alpha, \pm\alpha^{-1}$ at once, from evaluation, passing by recursive multiplication, up to the first steps of inversion and early recomposition.

Each quartet can be handled separately (even in parallel but we do not reduce memory usage that way). If all quartets are computed as a sequence, the same $O(b)$ memory can be used for each one, storing the results as shown in fig. 2. ■

E. Some more hints for implementation

To work with early recomposition, we do not really need to completely clear the first and the last column, we can delay this step.

There are two cases, we can delay the removal of an integer entry, or a fraction. The first one is easier to follow and we start describing it.

Assume we are working with the two lines:

$$\begin{pmatrix} L_u \\ L_d \end{pmatrix} = \left(\begin{array}{c|cc:cc:cc|c} 0 & x_1 & 0 & x_2 & 0 & x_3 & 0 & \beta \\ \alpha & 0 & x_1 & 0 & x_2 & 0 & x_3 & 0 \end{array}\right)$$

The procedure described so far would suggest to perform

$$L_u \leftarrow L_u - \beta(\cdots, 0, 1); L_d \leftarrow L_d - \alpha(1, 0, \cdots)$$

then the recomposition $W = L_u Y + L_d$. But we can swap the order, by computing W from the original lines, then removing the spurious values with:

$$W \leftarrow W - Y\beta(\cdots, 0, 1) - \alpha(1, 0, \cdots).$$

It is only an offset issue, not changing the total number of operations nor the result.

Surprisingly this works also if α or β are positive fractions and all x_i are positive integers. For example, we may shift the lines generated by ± 2 obtaining:

$$\begin{pmatrix} L_u \\ L_d \end{pmatrix} = \left(\begin{array}{c|cc:cc:cc|c} 0 & 1 & 0 & 4 & 0 & 16 & 0 & 64 \\ \frac{1}{4} & 0 & 1 & 0 & 4 & 0 & 16 & 0 \end{array}\right).$$

The division performed on L_d generating the $\frac{1}{4}$ entry could be non-exact, i.e. some non-zero bits could be removed. Luckily we know in advance that the same bits would have been cleared by the subtraction $L_d - (1, 0, \cdots)$, without generating a borrow. After the early recomposition we can clear both the fraction and the 64 with:

$$W \leftarrow W - 64Y(\cdots, 0, 1) - \frac{1}{4}(1, 0, \cdots).$$

GMP version 5.0 [3] uses this trick in both Toom-6.5 and Toom-8.5 implementations, to delay the two trivial multiplications in $0, \infty$, and use for evaluation purposes the memory area reserved for their product.

V. Generating the sequence

We still have to characterise the matrix $\widetilde{M}'''$ obtained exploiting evaluation on inverses in the matrix of early recomposed coefficients, and to give a sequence to clear its values.

Note that we focus on Toom-$2k+\frac{1}{2}$, i.e. $\widetilde{M}''_{2k}$ is a square Vandermonde-like $(2k-1)\times(2k-1)$ matrix. We use quartets of points $\pm\alpha, \pm\alpha^{-1}$, always using the smaller powers of 2 we can use (or powers of x for the extension explained in §VII), thus we obtain powers of $4 = 2^2$ in the recomposed matrix.

The step from $\widetilde{M}''_{2k}$ to $\widetilde{M}'''_{2k}$ described in §III-C generates k symmetric rows and $k-1$ antisymmetric rows. There are an odd number $2k-1$ of columns, this means that the antisymmetric rows have a zero entry in the middle. On the other side, the middle entry of symmetric rows is a power of 4 (resp. x), doubled.

Generating the sequence

If we want to generate the sequence with a program, the first step is to generate the two sub-matrices of $\widetilde{M}'''_{2k}$. We do not write here the general formulas, whose complexity is limited to the mess of indexes. The two matrices are completely independent, and the two inversion sequences can run simultaneously.

We show, as an example, the two sub-matrices generated by Toom-8.5:

$$A = \begin{pmatrix} 68719476737 & 1073741888 & 16781312 & \mathbf{524288} \\ 16777217 & 1048592 & 65792 & \mathbf{8192} \\ 4097 & 1028 & 272 & \mathbf{128} \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$B' = \begin{pmatrix} 240 & 1020 & 4095 \\ 65280 & 1048560 & 16777215 \\ 16773120 & 1073741760 & 68719476735 \end{pmatrix}$$

At first we will show how to transform the matrix A to lower triangular, with a line by line process, starting from the lower right corner. For each line the first entry (on the right) will be removed with a linear combination involving a power of 2 (i.e. a shift), all the other combinations, with lines below the one we are working on, require a linear combination in the form $W_i \leftarrow W_i - \alpha W_j$ with $\alpha \in \mathbb{N} \setminus \{2^n\}$.

In the example, we can start modifying two rows and obtain,

$$\begin{pmatrix} L_2 \\ L_3 \\ L_4 \end{pmatrix} = \begin{pmatrix} 15181425 & \mathbf{680400} & 0 & 0 \\ 3969 & 900 & \mathbf{144} & 0 \\ 1 & 1 & 1 & \mathbf{1} \end{pmatrix}.$$

Next step is the linear combination for the first line:

$$L_1 \leftarrow (L_1 - 524288L_4 - 112896L_3 - 1428L_2)/46591793325.$$

All involved numbers are integers, i.e. we do never need to use a more general combination like $W_i \leftarrow \beta W_i - \alpha W_j$, because all the time we have a value to be cleared, we find that it is a multiple of the value we have on the diagonal to clear it. This is not obvious, it depends on the order we use, and it can be a surprise. The first cleared entry is divisible by 1, this is obvious; after the first subtraction, the second one is divisible by 144; then the third one by 680400. This property is not trivial to prove, but it is easy to generate sequences and test if it is true. The author generated all the inversion sequences for Toom-$2k+\frac{1}{2}$, up to Toom-$160+\frac{1}{2}$, and no counterexample was found. It would be more elegant to give a general proof, true for all values of k, but it would not be more useful, because we can conjecture that asymptotically faster methods[5] make oversized Toom methods completely useless.

The division we inserted in the operation above starts the second part, removing elements under the diagonal and divide, to obtain all ones on the main diagonal. We proceed line by line, starting from the upper left corner.

Every line, except the first and last one, need to be processed only twice, if writing to memory is slow and general linear combinations are available, this can be a good point.

The clearing of B' works the same way, upside-down: it is transformed to upper triangular, starting from the upper left corner, then the way back starts from the lower right one.

VI. Cover a wide range of unbalancements

Once we have implemented an inversion sequence for a Toom-$2k+\frac{1}{2}$ method, we can use it for many different multiplication functions. It can be used for almost balanced $(2k+1)\times 2k$ multiplications, or for slightly unbalanced $(2k+2)\times(2k-1)$ or strongly unbalanced $(3k+1)\times(k)$ multiplications. We don not consider stronger unbalancements than $(4k-1)\times(2)$, because very unbalanced multiplications need a different approach [8].

Unfortunately, if the multiplication is exactly balanced (e.g. for squaring) or the sizes are $(2k+1)\times(2k-1)$ or $(2k+j)\times(2k-j)$ for any integer $0 \le j \le 2k$ the method does not work, because in the splitting phase we can not split the operands in n and m parts so that $n+m = 4k+1$.

The best we can do is to have an empty part for one of the operands i.e. n or m can be decreased by one, and we have $n + m = 4k$.

The operand ratios that Toom-$2k + \frac{1}{2}$ can not handle, are covered by Toom-$2k$. We have already described in §III-D the small adjustment to obtain the balanced version from the unbalanced one. The simpler solution is to ignore the evaluation at ∞ replacing it with a zero. This moves the implementation detail of correctly handle the length of the product to other portions of the code.

A slightly more efficient way would be to ignore the evaluation in 0, because the product required by the evaluation in ∞ can be shorter, but this trick has never been implemented.

Once we implement an interpolation function capable of both $2k$ and $2k + \frac{1}{2}$, we can write a single function that can handle a wide range of unbalancements, from balanced multiplications to products whit one operand $2k$ times longer than the other one.

VII. Extension to polynomials

The technique described so far can be extended to polynomials too. Not only for polynomials with rational coefficients, but also for polynomials over different fields where we may not have all the powers of 2 we need for evaluating, with the exception of characteristic two (because we need positive and negative points).

The simple trick to extend all the methods described in this paper to polynomials is to replace the points $\pm 2^n$ with $\pm x^n$, where x is the variable. Evaluation points $(\pm x^n, 1), (1, \pm x^n) \in \mathbb{R}[x]^2$ are perfectly compatible with the method described in §II-A.

Consistency of the method, and the property that all generated entries are divisible by the values on the diagonal, has been tested by the author up to Toom-$50 + \frac{1}{2}$.

VIII. Conclusions

We described a straightforward way to obtain full sequences of operations to obtain memory efficient, high degree, unbalanced Toom'n'half-methods. Even if automatically generated in a direct way, the obtained sequences are comparable to the previously published ones for low degree, and far better for degrees above Toom-6.

The method proved to be effective, the `toom8h` function implementing Toom-8.5 multiplication in the current version of GMP [3] is used for a wide range of values. For example, for the Core-i CPU, GMP uses `toom8h` for operands of more than 18,000 bits and less than 200,000; above that range FFT is used, below it we have Toom-2, Toom-3, Toom-4, Toom-6.5. Those values suggest that other higher degree Toom-$n + \frac{1}{2}$ can find their range.

Future work

We plan to write a program generating Toom functions in the Transalpyne language, in order to automatically obtain a wide range of functions for the middle product[13]. It would also be interesting to generate, with the same language, functions for the multi-point evaluation, then transpose them to obtain interpolation, and finally compare the code and its speed with the one obtained with the method described in this paper.

Acknowledgements

The author thanks Törbjorn Grandlund, Niels Möller and all the participants to the GMP development list.

Moreover the author thanks the anonymous reviewers for their valuable suggestions and corrections.

References

[1] А. Л. Тоом, "О сложности схемы из функциональных элементов, реализирующеуи умножение целых чисел," Доклады Академии Наук СССР, vol. 150, no. 3, pp. 496–498, October 1963, english translation in [14].

[2] S. A. Cook, "On the minimum computation time of functions," Ph.D. dissertation, Dept. of Mathematics, Harvard University, 1966.

[3] T. Grandlund *et al.*, "GNU MP: The GNU multiple precision arithmetic library, v5.0.1," 2010, http://gmplib.org/#DOC.

[4] V. Shoup *et al.*, "The NTL library," http://www.shoup.net/ntl/.

[5] A. Schönhage and V. Strassen, "Schnelle multiplikation großer zahlen," *Computing*, vol. 7, no. 3–4, pp. 281–292, 1971.

[6] A. De, P. P. Kurur, C. Saha, and R. Saptharishi, "Fast integer multiplication using modular arithmetic," *CoRR*, vol. abs/0801.1416, 2008.

[7] A. Zanoni, "Toom-Cook 8-way for long integers multiplication," in *SYNASC '09: Proceedings of the 2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Los Alamitos, CA, USA: IEEE Computer Society, September 2009, pp. 54–57.

[8] ——, "Iterative Toom-Cook methods for very unbalanced long integers multiplication," in *Proceedings of the ISSAC 2010 Conference*, S. M. Watt, Ed. ACM, July 2010, pp. 319–323.

[9] M. Bodrato and A. Zanoni, "Integer and polynomial multiplication: Towards optimal Toom-Cook matrices," in *Proceedings of the ISSAC 2007 Conference*, C. W. Brown, Ed. ACM press, July 2007.

[10] M. Bodrato, "Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0," in *WAIFI'07 proceedings*, ser. LNCS, C. Carlet and B. Sunar, Eds., vol. 4547. Springer, June 2007, pp. 116–133.

[11] D. J. Bernstein, "Batch binary edwards," in *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 317–336.

[12] А. А. Карацуба and Ы. Офман, "Умножение многозначных чисел на автоматах," *Доклады Академии Наук СССР*, vol. 145, no. 2, pp. 293–294, 1962, english translation in [15].

[13] L. De Feo and E. Schost, "Transalpyne: a language for automatic transposition," *SIGSAM Bull.*, vol. 44, no. 1/2, pp. 59–71, 2010.

[14] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Mathematics Doklady*, vol. 3, pp. 714–716, 1963.

[15] A. A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, vol. 7, pp. 595–596, 1963.