Iterative Toom-Cook Methods For Very Unbalanced Long Integers Multiplication

Alberto Zanoni

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

Abstract. We consider the multiplication of long integers when one factor is much bigger than the other one. We describe an iterative approach using Toom-Cook unbalanced methods, by evaluating the smaller factor just once. The particular case of Toom-2.5 is considered in full detail, and a further optimization depending on the parity of shortest operand evaluation in 1 is described. A comparison with GMP library is also presented.

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

Keywords and phrases: Long integers, multiplication, Karatsuba, Toom-Cook

1 Introduction

Starting with the works of Karatsuba [6], Toom [9] and Cook [3], who found methods to lower asymptotic complexity for polynomial multiplication from $O(n^2)$ to $O(n^e)$, where $1 < e \leqslant \log_2 3$, many efforts have been done to find optimized implementations in arithmetic software packages [7], [4], [5].

The family of Toom-Cook (Toom, for short) methods is an infinite set of algorithms (Toom-3, Toom-4, etc. – Karatsuba may be identified with Toom-2). The original family was generalized by Bodrato and Zanoni in [2] by considering unbalanced operands as well – that is, polynomials with different degrees – with the so-called Toom-$(k+1/2)$ methods (Toom-2.5, Toom-3.5, etc.) and with the unbalanced use of classic methods.

Each of them may be viewed as solving a polynomial interpolation problem, with base points not specified *a priori*, from which a matrix to be inverted is born. In a software implementation, a set O of basic operations (typically sums, subtractions, bit shiftings, multiplications and divisions by small numbers, etc.) is given. Practically, this is a set of very efficiently implemented basic functions in a certain computer language, and the idea is to use them both to evaluate factors in the base points and invert the resulting matrix step by step, by using row elementary operations.

When Toom methods are applied to long integers multiplication, carries and borrows enter the game, and code implementing each method must also take care of this aspects. The latest release of GMP library [5], our reference, implements many Toom methods, for balanced and limitedly unbalanced factors. For *very* unbalanced factors, either an iterative approach is used or, if factors are very long, FFT (see [8]) is used.

In this work we describe the general use of unbalanced Toom methods for iterative multiplication applied to very unbalanced factors, with an *ad hoc* optimization. In particular, we explain in full detail the basic and iterative Toom-2.5 method (the smallest unbalanced case), introducing a further optimization that can be exploited.

2 Toom-Cook method general description

We briefly summarize Toom-k multiplication algorithm for natural numbers. Let $u, v \in \mathbb{N}$: to compute the product $u \cdot v = w \in \mathbb{N}$, follow the five below indicated steps.

Splitting : Fix an appropriate basis $B \in \mathbb{N}$ and represent the two operands by two homogeneous polynomials $a, b \in \mathbb{N}[x, h]$ with degree d_1, d_2 respectively and $0 \leqslant a_i, b_i < B$ (base B expansion). In computer science, usually B is a power of two.

$$a(x,h) = \sum_{i=0}^{d_1} a_i x^i h^{d_1 - i} \quad ; \quad b(x,h) = \sum_{i=0}^{d_2} b_i x^i h^{d_2 - i}$$

One has $u = a(B, 1)$ and $v = b(B, 1)$. Let $c(x, h) = a(x, h)b(x, h)$, with $\deg(c) = d_1 + d_2$. For the classical Toom-k method one has $d_1 = d_2 = k - 1$, while in general, for possibly unbalanced operands, it must be $d_1 + d_2 = 2(k - 1)$ – in this case k can be whatever multiple of $1/2$.

Evaluation : Choose $2k-1$ values $v_i = (v_i', v_i'') \in \mathbb{Z}^2$ with v_i' and v_i'' coprime and such that $v_i \neq \pm v_j$ for $i \neq j$: evaluate both operands on all of them, obtaining $a(v_i)$, $b(v_i)$.
Recursion : Compute $w_i = a(v_i) \cdot b(v_i)$ recursively. Let $\mathbf{w} = (w_i)$ be the so obtained values vector.
Interpolation : Solve the interpolation problem $c(v_i) = w_i$ inverting the obtained pseudo–Vandermonde matrix A_k generated by the v_i values, computing $\mathbf{c} = A_k^{-1}\mathbf{w}$, where $\mathbf{c} = (c_i)$ is the vector of $c(x,h)$ coefficients.
Recomposition : Once all coefficients are computed, it's enough to evaluate back $w = c(B,1)$.

Standard complexity analysis shows that Toom-k's is $O(n^{log_k(2k-1)})$. The multiplicative constant hidden by the $O(\cdot)$ notation absorbs the complexity of Splitting, Evaluation, Interpolation and Recomposition phases. In order to minimize it, an accurate choice of v_i values and of the operations sequence for Evaluation and Interpolation phases helps in reducing the extra overhead. Details about this can be found in [2], [1], [12].

Example 1 (Toom-2.5, Toom-3). The matrices $A_{2.5}$ and A_3, obtained by the interpolation values $\{(1,0),(-1,1),(1,1),(0,1)\}$ and $\{(1,0),(2,1),(-1,1),(1,1),(0,1)\}$, respectively, are

$$A_{2.5} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad ; \quad A_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 16 & 8 & 4 & 2 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

3 Toom-2.5 method description

While in the following section we'll describe the iterative use of the smallest unbalanced Toom method, we report here all details of the basic version; for the sake of simplicity, in this and in the following section we use non-homogeneous notation. In this case we set $\nu' = \lfloor \log_2 v \rfloor + 1$, $\nu = \lceil \nu'/2 \rceil$ and $B = 2^\nu$, having thus

$$a(x) = a_2x^2 + a_1x + a_0 \quad ; \quad b(x) = b_1x + b_0 \quad ; \quad c(x) = c_3x^3 + c_2x^2 + c_1x + c_0$$

We recall that, because of the recursive implicit use, the coefficients a_0, a_1, a_2, b_0 and b_1 are themselves long integers with bit length $\leqslant \nu$ (a_2 and b_1 bit lengths may be strictly smaller than ν). There are only two non trivial evaluations to compute, corresponding in practice to set $x = \pm 1$.

3.1 Classical version

Evaluation phase is completed with 5 algebraic additions, as follows (t is a temporary variable):

$$\begin{array}{lll} t = a_2 + a_0 & & \\ a^+ = t + a_1 & ; & b^+ = b_0 + b_1 \\ a^- = t - a_1 & & b^- = b_0 - b_1 \end{array}$$

Computing the 4 needed values by multiplying corresponding evaluations we may set up the interpolation problem:

$$w_i = a(v_i)b(v_i) = c(v_i) \quad ; \quad i = 0,\ldots,3 \quad \Longrightarrow \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} w_3 \\ w_2 \\ w_1 \\ w_0 \end{pmatrix} = \begin{pmatrix} a_2b_1 \\ a^-b^- \\ a^+b^+ \\ a_0b_0 \end{pmatrix}$$

As the bit length of w_i is 2ν (it may be smaller for w_3), we indicate with the prefixes H and L their high (most meaningful) and low (less meaningful) parts: $w_i = H_ix + L_i$. Note moreover that $c_3 = w_3$ and $c_0 = w_0$. Beginning the interpolation, with two algebraic sums and a shift (indicated with $\gg$) we destructively have what we call the "decoupling" step:

$$\begin{array}{lll} w_2 = w_2 + w_1 & \Longrightarrow & (0\ 2\ 0\ 2)\ \text{Addition} \\ w_2 = w_2 \gg 1 & \Longrightarrow & (0\ 1\ 0\ 1)\ \text{Shift (division by 2)} \\ w_1 = w_1 - w_2 & \Longrightarrow & (1\ 0\ 1\ 0)\ \text{Subtraction} \end{array}$$

If at this point we "mix" interpolation and recomposition phases, it is possible to save a $O(\nu)$-addition. Consider infact the final recomposition at this point: formally, with new values of w_1 and w_2 *after* decoupling, we have

$$c(x) = (w_3)x^3 + (w_2 - w_0)x^2 + (w_1 - w_3)x + w_0$$

but we must consider overlaps: we may infact rewrite $c(x)$ as follows.

$$\begin{aligned} c(x) &= (H_3)x^4 + (H_2 + (L_3 - H_0))x^3 + (L_2 - H_3 + H_1 - L_0)x^2 + (L_1 - (L_3 - H_0))x + L_0 \\ &= c'_4x^4 + c'_3x^3 + c'_2x^2 + c'_1x + c'_0 \end{aligned}$$

Note that the value $L_3 - H_0$ occurs two times, and it can then be computed just once. Supposing that an operation in the interpolation phase has double cost, we need in total 15 additions and 2 shifts of complexity $O(\nu)$ each. In the case of long integers, one should of course take care of carries and borrows as well, but we do not describe all technical details here.

3.2 Even Toom-2.5

Sometimes it is also possible to use another version of Toom-2.5, by applying the so-called "divide by 2" technique introduced in [11]. The technique is based on the observation that

$$\frac{c(v_i)}{C} = \frac{a(v_i)b(v_i)}{C} = \frac{a(v_i)}{C}b(v_i) = a(v_i)\frac{b(v_i)}{C}$$

for whatever constant $C \in \mathbb{N}$ dividing exactly both members of whatever pair of above equalities. Now, take $C = 2$ for Toom-2.5 and test if $b_0 \equiv b_1 \pmod 2$ – just compare their least meaningful bits. If it is the case, then $b_0 + b_1$ (and $b_0 - b_1$ as well) is even, and we can compute the "divided by 2" products

$$w'_1 = \frac{w_1}{2} = a^+\frac{b^+}{2} \quad ; \quad w'_2 = \frac{w_2}{2} = a^-\frac{b^-}{2}$$

realizing the decoupling step with just two algebraic additions. This can also be done if $a_0 + a_2 \equiv a_1 \pmod 2$ (a^+ and a^- as well are even). In the below pseudo-code, t_1, t_2 are temporary variables:

Even evaluations : a		**Even evaluations** : b	
$a^+ = a_2 + a_0$	$b^+ = b_0 + b_1$	$t_1 = a_2 + a_0$	$b^- = b_0 - b_1$
$a^+ = a^+ + a_1$	$b^- = b_0 - b_1$	$a^+ = t_1 + a_1$	$b^- = b^- \gg 1$
$a^+ = a^+ \gg 1$	$a^- = a^+ - a_1$	$a^- = t_1 - a_1$	$b^+ = b^- + b_1$

$$\begin{aligned} t_2 &= a^-b^- \\ w_1 &= a^+b^+ \\ w_2 &= w_1 + t_2 \\ w_1 &= w_1 - t_2 \end{aligned}$$

Note that the shift operation moved in the evaluation phase, so that its cost is halved. With a probability of 75 %, the number of $O(\nu)$ shifts is 1, not 2, so that the average linear complexity of Toom-2.5 is (A = addition, S = shift)

$$\mathcal{L}_{2.5} = 15A + \frac{1}{4}(2S) + \frac{3}{4}S = 15A + \frac{5}{4}S$$

4 Iterative Toom-Cook methods

In [10] the first approach to an iterative use of Karatsuba (i.e. Toom-2) method was proposed. Inspired by those ideas, we present here an iterative approach for generic very unbalanced Toom methods. We first present the particular case of Toom-2.5, and then the general one. Let $d_1 = d$ be much bigger than $d_2 = 1$: we have

$$a(x) = \sum_{i=0}^{d} a_i x^i \quad ; \quad b(x) = b_1x + b_0$$

so that

$$c(x) = \sum_{i=0}^{d+1} c_i x^i = (a_db_1)x^{d+1} + (a_db_0 + a_{d-1}b_1)x^d + \cdots + (a_1b_0 + a_0b_1)x + (a_0b_0)$$

with

$$c_{d+1} = a_db_1 \quad ; \quad c_0 = a_0b_0 \quad ; \quad c_i = a_ib_0 + a_{i-1}b_1 \ : \ i = 1, \ldots, d \tag{1}$$

4.1 Iterative Toom-2.5

Let $m = \left\lfloor \frac{d+1}{3} \right\rfloor$ and split $a(x)$ into sub-polynomials ("sections") of degree 2 each until possible, as follows – mathematically speaking, set $y = x^3$ and consider $a(x)$ opportunely as $\bar{a}(x,y) = \sum_j a_j(x)y^j = \sum_j (a_{3j+2}x^2 + a_{3j+1}x + a_{3j})y^j$:

$$a(x) = a'(x)x^{3m} + (a_{3m-1}x^2 + a_{3m-2}x + a_{3m-3})x^{3(m-1)} + \cdots + (a_5x^2 + a_4x + a_3)x^3 + (a_2x^2 + a_1x + a_0)$$

Note that we may have a "border" effect when the most meaningful part of $a(x)$, indicated here with $a'(x)$, has degree smaller than 2 (it may be $a'(x) = 0$ as well). Applying Toom-2.5 method to $a_j(x)$ and $b(x)$, we may reconstruct sectionwise the final result.

Note that for each product $a_j(x)b(x)$ the second factor $b(x)$ is fixed, and therefore we can compute b^+ and b^- just once, using them for every product. At step j, Toom-2.5 gives the four values

$$a_{3j+2}b_1 \quad , \quad c_{3j+2} \quad , \quad c_{3j+1} \quad , \quad a_{3j}b_0$$

The coefficients c_{3j}, with $j > 0$, can be obtained simply by adding the products $a_{3(j-1)+2}b_1$, obtained at step $j-1$, and $a_{3j}b_0$, obtained at step j (when $j = 0$, we directly have $c_0 = a_0b_0$, with no extra addition, and similarly for c_{d+1}, if $\deg(a'(x)) = 2$). If $\deg(a'(x)) < 2$, we recursively compute $c'(x) = a'(x)b(x)$, using the most appropriate multiplication method, finally obtaining the remaining highest part of $c(x)$, whose less meaningful part must be combined with the most meaningful part of the precedent section.

For what concerns additions, all sections need 15 of them in total: the first one ($j = 0$) works as the (classical or even) Toom-2.5 case, while for all the other ones ($j = 1, \ldots m-1$) we have two less, as we don't need to evaluate b any more, but the recomposition asks for two more. For the number of shifts, there are three cases to consider:

1. $\boxed{b^+ \equiv b^- \equiv 0 \pmod 2}$: we can use even Toom-2.5 for all m sections, and just *one* single shift is needed.
2. $\boxed{b^+ \equiv b^- \equiv 1 \pmod 2,\ a_j^+ \equiv a_j^- \equiv 0 \pmod 2}$: we can use even Toom-2.5 for this section.
3. $\boxed{b^+ \equiv b^- \equiv 1 \pmod 2,\ a_j^+ \equiv a_j^- \equiv 1 \pmod 2}$: we must use classical Toom-2.5 for this section.

The probability of case 1 is $50\,\% = 1/2$. To compute the average shift complexity, we have to consider the event $E(m,h)$ related to cases 2 and 3, described as follows: $b^+ \equiv 1 \pmod 2$ and for $0 \leqslant h \leqslant m$ sections, even Toom-2.5 cannot be applied, while for the remaining $m-h$ ones it can. The probability of $E(m,h)$ and the associated number of $O(\nu)$ shifts $S(m,h)$ are

$$p(E(m,h)) = \frac{1}{2} \cdot \frac{1}{2^m}\binom{m}{h} \quad ; \quad S(m,h) = 2h + (m-h) = m + h$$

so that the average number $S(m)$ of $O(\nu)$ shifts is given by the following expression:

$$\begin{aligned}
S(m) &= \frac{1}{2}\left[1 + \sum_{h=0}^{m} \frac{1}{2^m}\binom{m}{h}(m+h)\right] = \frac{1}{2}\left[1 + \frac{1}{2^m}\sum_{h=0}^{m}\binom{m}{h}m + \sum_{h=0}^{m}\frac{1}{2^m}\binom{m}{h}h\right] \\
&= \frac{1}{2}\left[1 + \frac{m}{2^m}\sum_{h=1}^{m}\binom{m}{h} + \frac{1}{2^m}\sum_{h=0}^{m} h\frac{m!}{h!(m-h)!}\right] \\
&= \frac{1}{2}\left[1 + \frac{m}{2^m}\sum_{h=0}^{m}\binom{m}{h}1^h 1^{m-h} + \frac{1}{2^m}\sum_{h=1}^{m} h\frac{m(m-1)!}{h(h-1)!(m-1-h+1)!}\right] \\
&= \frac{1}{2}\left[1 + \frac{m}{2^m}(1+1)^m + \frac{m}{2^m}\sum_{h=1}^{m}\frac{(m-1)!}{(h-1)!(m-1-(h-1))!}\right] \\
&= \frac{1}{2}\left[1 + \frac{m}{2^m}2^m + \frac{m}{2^m}\sum_{h=0}^{m-1}\frac{(m-1)!}{h!(m-1-h))!}\right] = \frac{1}{2}\left[1 + m + \frac{m}{2^m}2^{m-1}\right] = \\
&= \frac{1}{2}\left[1 + m + \frac{m}{2}\right] = \frac{1}{2}\left[1 + \frac{3}{2}m\right] = \frac{3m+2}{4}
\end{aligned}$$

The average linear complexity $\mathcal{L}_{2.5}^{(m)}$ for all sections and the total complexity $\mathcal{L}_{2.5}(d)$ are then

$$\mathcal{L}_{2.5}^{(m)} = (15m)A + \frac{3m+2}{4}S \qquad ; \qquad \mathcal{L}_{2.5}(d) = \mathcal{L}_{2.5}^{(\lfloor (d+1)/3 \rfloor)} + \mathcal{L}'_{2.5}(d)$$

where $\mathcal{L}'_{2.5}(d)$ is the residual complexity due to the "border" extra component $a'(x)b(x)$, depending on $a'(x)$.

For what concern the number of multiplications (see equation 1) with "classic" method $\mathcal{M}_c(d)$, we have one product for c_0 and c_{d+1} each, and two for all other c_j, $1 \leqslant j \leqslant d$. With Toom-2.5 iterative method, we have four multiplications every three coefficients, so that the total number $\mathcal{M}_{2.5}(d)$ is lowered (similarly as before, we indicate with $\mathcal{M}'(d)$ the number of "remaining" multiplications due to $a'(x)$, of order $O(1)$).

$$\mathcal{M}_c(d) = 2(d+1) \qquad ; \qquad \mathcal{M}_{2.5}(d) = 4\left\lfloor \frac{d+1}{3} \right\rfloor + \mathcal{M}'(d)$$

The ratio of these two numbers and the corresponding gain are then

$$R_{2.5}(d) = \frac{\mathcal{M}_{2.5}(d)}{\mathcal{M}_c(d)} \simeq \frac{4}{3} \cdot \frac{d+1}{2(d+1)} \xrightarrow[d\to\infty]{} \frac{2}{3} \qquad ; \qquad G_{2.5}(d) = 1 - R_{2.5}(d) \xrightarrow[d\to\infty]{} \frac{1}{3} \simeq 33\,\%$$

4.2 Iterative Toom-*k*

For the generic Toom-k method (k can be whatever multiple of $1/2$), we consider its most unbalanced version, giving us sections of $a(x)$ with degree equal to $2k-3$. The schema is similar to the one used in the iterative Toom-2.5 case: the product of section $a_j(x)$ and $b(x)$ gives now

$$a_{(2k-2)j+2k-3}b_1 \quad , \quad c_{(2k-2)j+2k-3} \quad , \quad \cdots \quad , \quad c_{(2k-2)j+1} \quad , \quad a_{(2k-2)j}b_0$$

where the first and last product have to be recombined with (last product of) precedent and (first product of) following section to obtain the two extremal coefficients.

We have $m = \left\lfloor \frac{d+1}{2k-2} \right\rfloor$ (set $y = x^{2k-2}$), and the number of multiplications is now

$$\mathcal{M}_k = (2k-1)\left\lfloor \frac{d+1}{2(k-1)} \right\rfloor + \mathcal{M}'(d)$$

with a gain $G_k(d) \simeq 1 - \frac{2k-1}{2(k-1)}\frac{d+1}{2(d+1)} \xrightarrow[d\to\infty]{} 1 - \frac{2k-1}{4(k-1)} \xrightarrow[k\to\infty]{} 50\,\%$

For small values of k, *ad hoc* versions of Toom-k methods are considered in GMP: each one has a different specific and specialized evaluation and interpolation phase. For asymptotically big k values, Toom-k method are simply not used in practice, because of the more effective FFT method. It is therefore not possible for us to present a general treatment of linear complexity of their iterative counterparts here.

5 Iterative Toom-2.5 method: Implementation

We implemented `C` code for iterative Toom-2.5 method, available on request, by using GMP library. GMP library packs bits in blocks, called *limbs* (a classical case is when a limb has 32 bits). Our implementation needs a $(4\nu+2)$-limbs scratch space. We report a scheme of the needed operations only for the generic section for classical Toom-2.5, for simplicity. For even Toom-2.5 cases and all technical details concerning carries and borrows treatment we invite the reader to refer directly to the code itself. The characteristics of our architecture are the following:

Processor	Intel Core 2 Duo : 3 GHz
RAM	4 GB
Operating System	Linux Kubuntu 8.10
C Compiler	gcc 4.3.2

We indicate with S the scratch space and with R the area in which the current part of the result should "appear". Actually 5 ν-parts should be present in R, but we split them: the less meaningful three parts in R and the two most meaningful ones in S, so that the possible final recomposition if $\deg(a') < 2$ can be straightforwardly done. At the beginning, we indicate with H_p and L_p the values H_3 and $H_2 + L_3 - H_0$ (the most meaningful ones), respectively, of

the result given by Toom-2.5 applied to the precedent section. We save the evaluation values b^+ and b^- in the highest part of the memory area that will contain the result.

	R			S			
			$a_0 + a_2$			H_p	L_p
		$a_0 + a_1 + a_2$	$a_0 + a_2$			H_p	L_p
			$a_0 + a_2$	H_1	L_1	H_p	L_p
			$a_0 - a_1 + a_2$	H_1	L_1	H_p	L_p
	H_2	L_2		H_1	L_1	H_p	L_p
	$H_2 + H_1$	$L_2 + L_1$		H_1	L_1	H_p	L_p
	$H_2 = \frac{H_2 + H_1}{2}$	$L_2 = \frac{L_2 + L_1}{2}$		H_1	L_1	H_p	L_p
	H_2	L_2		$H_1 = H_1 - H_2$	$L_1 = L_1 - L_2$	H_p	L_p
$\cdots$	H_2			$H_1 + L_2$	L_1	H_p	L_p
	H_2	H_0	L_0	$H_1 + L_2$	L_1	H_p	L_p
	H_2	H_0	L_0	$H_1 + L_2 - L_0$	L_1	H_p	L_p
	H_2	H_0	$L_0 + L_p$	$H_1 + L_2 - L_0$	L_1	H_p	
	H_2	H_0	$L_0 + L_p$	$H_1 + L_2 - L_0$	$L_1 + H_p$		
	H_2	H_0	$L_0 + L_p$	$H_1 + L_2 - L_0$	$L_1 + H_p$	H_3	L_3
	H_2	$L_3 - H_0$	$L_0 + L_p$	$H_1 + L_2 - L_0$	$L_1 + H_p$	H_3	
		$L_3 - H_0$	$L_0 + L_p$	$H_1 + L_2 - L_0$	$L_1 + H_p$	H_3	$H_2 + L_3 - L_0$
		$L_1 + H_p - L_3 + H_0$	$L_0 + L_p$	$H_1 + L_2 - L_0$		H_3	$H_2 + L_3 - L_0$
	$H_1 + L_2 - L_0 - H_3$	$L_1 + H_p - L_3 + H_0$	$L_0 + L_p$			H_3	$H_2 + L_3 - L_0$

At the end, (the first three parts of) R and (the first two parts of) S contain the correct values for the current section, and the whole process can be iterated.

Figure 1 shows the behavior of iterative Toom-2.5 with respect to GMP 4.3.1, the latest official release. On x and y axes we report the number of limbs of u and v: from 1 to 5000 and from 1 to 1000, respectively. Inside the region of applicability of iterative Toom-2.5 method – the infinite triangle between x-axis and the line $y = 2x/3$ – the darker the point, the bigger the gain with respect to GMP: white indicates that GMP is faster. On our architecture, in the shown region we obtained an average percentage gain (counting only cases in which iterative Toom-2.5 is faster than GMP) of 5.01 % and a maximum gain of 29.05 % (the extra linear complexity "consumes" part of the possible gain).

In figure 2 and 3 we report other two graphics comparing iterative Toom-2.5 with a development version (pre-4.4) of GMP library (beginning of December 2009). The last graphic was obtained considering a small code modification concerning thresholds related to different multiplication methods choosing in the general low-level function `mpn_mul`. The average and maximum gains in this case were 5.07 % and 32.09 %. Note that in this very last case iterative Toom-2.5 method is particularly effective in a well localized bottom horizontal strip, immediately above a thin white one, given by high-school multiplication method, used when one factor is sensibly small (below a certain threshold).

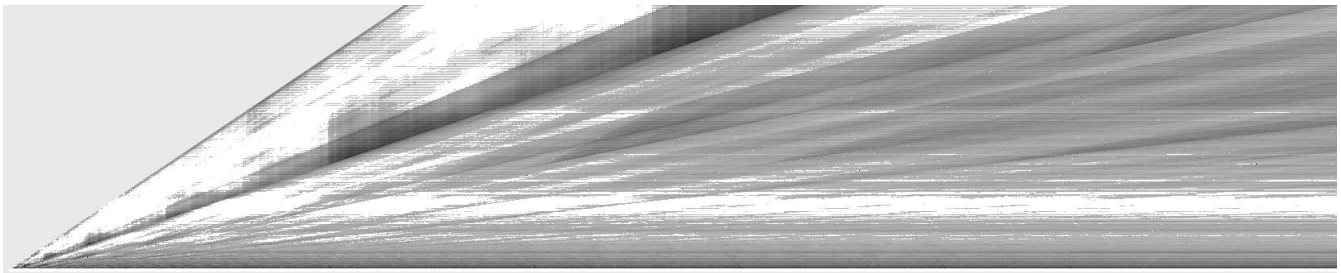

Fig. 1. Iterative Toom-2.5 vs. GMP 4.3.1. Limbs: $[1, 5000] \times [1, 1000]$. Gains: Average 5.01 % ; Max 29.05 %

We observed that exploiting just the parity of b^+ and b^- our code is on average slightly faster than taking into consideration also the parity of a^+ and a^-. This seems to mean that the extra operations needed for each section to distinguish between classic and even Toom-2.5 are not always worth the gain. Anyway, more tests on many different hardware and software architecture are needed to have a clear idea of the general behavior of the code.

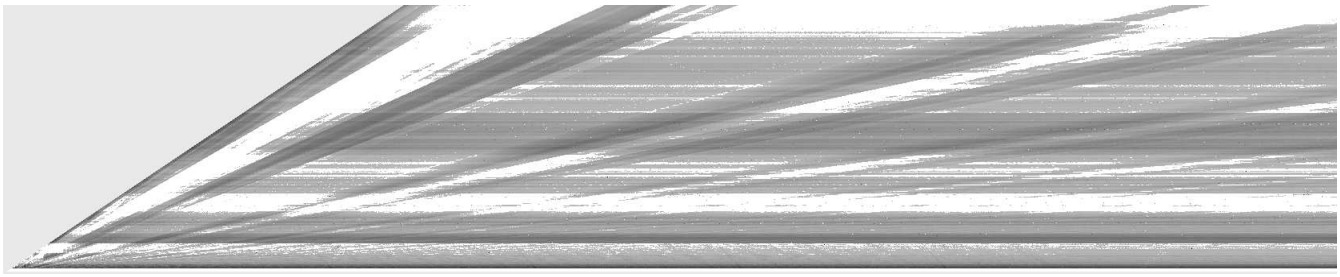

Fig. 2. Iterative Toom-2.5 vs. GMP pre-4.4.0. Limbs: $[1, 5000] \times [1, 1000]$. Gains: Average 5.07 % ; Max 32.09 %

Fig. 3. Iterative Toom-2.5 vs. GMP pre-4.4.0, modified. Limbs: $[1, 5000] \times [1, 1000]$. Gains: Average 3.80 % ; Max 33.64 %

6 Acknowledgements

A grateful thank goes to Marco Bodrato, for his precious advice and constant encouragement and help in preparing the `C` code and the paper organization. This work is dedicated to him.

7 Conclusion

We described an approach to very unbalanced long integers multiplication by means of unbalanced Toom-Cook methods. We detailed the Toom-2.5 case comparing the performance of our code with GMP library. The approach revealed to be quite effective in a well-defined region of the product space, and more unbalanced Toom-k methods promise to behave effectively as well.

References

1. Marco Bodrato. Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In Claude Carlet and Berk Sunar, editors, *WAIFI '07 proceedings*, volume 4547 of *Lecture Notes in Computer Science*. Springer, June 2007.
2. Marco Bodrato and Alberto Zanoni. Integer and polynomial multiplication: Towards optimal Toom-Cook matrices. In Christopher W. Brown, editor, *Proceedings of the ISSAC 2007 Conference*. ACM press, July 2007.
3. Stephen A. Cook. *On the minimum computation time of functions*. PhD thesis, Department of Mathematics, Harvard University, 1966.
4. Tom St Denis, Mads Rasmussen, and Greg Rose. Multi-precision math (tommath library documentation). URL: `http://math.libtomcrypt.com/files/tommath.pdf`.
5. The GNU multiple precision (GMP) library documentation. URL: `http://gmplib.org/#DOC`.
6. Anatolii Alexeevich Karatsuba and Yuri Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7(7):595–596, 1963.
7. Donald E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1981.
8. A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3–4):281–292, 1971.
9. Andrei L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963. URL: `http://www.de.ufpe.br/~toom/articles/engmat/MULT-E.PDF`.
10. André Weimerskirch and Christof Paar. Generalizations of the Karatsuba algorithm for polynomial multiplication. Technical report, Ruhr-Universität-Bochum, 2003.
11. Alberto Zanoni. Some Toom-Cook methods for even long integers. In Dorin Wainberg Daniel Breaz, Nicoleta Breaz, editor, *Proceedings of the International Conference on Theory and Applications of Mathematics and Informatics, ICTAMI 2009*, pages 807–828. Aeternitas Publishing House, September 2009.
12. Dan Zuras. More on squaring and multiplying large integers. *IEEE Transactions on Computers*, 43(8):899–908, August 1994. URL: `http://ieeexplore.ieee.org/iel1/12/7318/00295852.pdf?arnumber=295852`.