On Polynomial Multiplication in Chebyshev Basis

Pascal Giorgi

On Polynomial Multiplication in Chebyshev Basis

Pascal Giorgi

September 23, 2010

Abstract

In a recent paper Lima, Panario and Wang have provided a new method to multiply polynomials in Chebyshev basis which aims at reducing the total number of multiplication when polynomials have small degree. Their idea is to use Karatsuba's multiplication scheme to improve upon the naive method but without being able to get rid of its quadratic complexity. In this paper, we extend their result by providing a reduction scheme which allows to multiply polynomial in Chebyshev basis by using algorithms from the monomial basis case and therefore get the same asymptotic complexity estimate. Our reduction allows to use any of these algorithms without converting polynomials input to monomial basis which therefore provide a more direct reduction scheme then the one using conversions. We also demonstrate that our reduction is efficient in practice, and even outperform the performance of the best known algorithm for Chebyshev basis when polynomials have large degree. Finally, we demonstrate a linear time equivalence between the polynomial multiplication problem under monomial basis and under Chebyshev basis.

1 Introduction

Polynomials are a fundamental tool in mathematics and especially in approximation theory where mathematical functions are approximated using truncated series. One can think of the truncated Taylor series to approximate a function as a polynomial expressed in monomial basis. In general, many other series are preferred to the classical Taylor series in order to have better convergence properties. For instance, one would prefer to use the Chebyshev series in order to have a rapid decreasing in the expansion coefficients which implies a better accuracy when using truncation [18, 5]. One can also use other series such as Legendre or Hermite to achieve similar properties. It is therefore important to have efficient algorithms to handle arithmetic on polynomials in such basis and especially for the multiplication problem [2, 7].

Polynomial arithmetic has been intensively studied in the past decades, in particular following the work in 1962 of Karatsuba and Ofmann [16] who have shown that one can multiply polynomials in a subquadratic number of operations. Let two polynomials of degree d over a field $\mathbb{K}$ be given in monomial basis, one can compute their product using Karatsuba's algorithm in $O(n^{\log_2 3})$ operations in $\mathbb{K}$. Since this seminal work, many other algorithms have been invented in order to asymptotically reduce the cost of the multiplication. In particular, one can go down to $O(n^{\log_{r+1}(2r+1)})$ operations in $\mathbb{K}$ with the generalized Toom-Cook method [23, 10] for any integer $r > 0$. Finally, one can even achieve a quasi-linear time complexity using the so-called FFT [11] assuming the field $\mathbb{K}$ have some nice properties (see [13, 6] for a good introduction). One of the main concern of this work is that all these algorithms have been designed for polynomials given in monomial basis, and they do not directly fit the other basis, such as the Chebyshev one.

In this work, we extend the result of Lima, Panario and Wang [17] which is to directly use Karatsuba's algorithm [16] within the multiplication of polynomials given in Chebyshev basis. In [17] the authors partially succeeded in such a task but without being able to reach Karatsuba's asymptotic complexity. Our approach here is more general and it endeavors to completely reduce the multiplication in Chebyshev basis to the one in monomial basis. Of course, one can already achieve such a reduction by using back and forth conversions between the Chebyshev and monomial basis using methods presented in [3, 4]. However, this reduction scheme is not direct and it implies at least three calls to multiplication in monomial basis: two for the back and forth conversions and one for the multiplication of the polynomials. Note that it is not even clear from [3, 4] that basis conversions are equivalent to only one multiplication in monomial basis. In this work, we present a new reduction scheme which does not rely on basis conversion and therefore reduces the number of multiplications in monomial basis to only two. We also demonstrate that degenerating this reduction for the case of DFT-based multiplication algorithm reduces the number of operations. Considering practical efficiency, we will see that our degenerated reduction scheme will definitively compete with implementations of the most efficient algorithms available in the literature.

Organization of the paper. Section 2 recalls some complexity results on polynomial multiplication in monomial basis and provides a detailed study on arithmetic operation count in the case of polynomials in $\mathbb{R}[x]$. In Section 3 we give a short review on the available methods in the literature to multiply polynomials given in Chebyshev basis. Then, in Section 4 we propose our new method to perform such multiplication by re-using multiplication in monomial basis. We analyze the complexity of this reduction and compare it to other existing methods. We perform some practical experimentations of such a reduction scheme in Section 5, and then compare its efficiency and give a small insight on its numerical reliability. Finally, we exhibit in Section 6 the linear equivalence between the polynomial multiplication problem in Chebyshev basis and in monomial basis with only a constant factor of two.

2 Classical Polynomial Multiplication

It is well-known that polynomial multiplication of two polynomials of $\mathbb{K}[x]$ with degree $d = n-1$ can be achieved with less than $O(n^2)$ operations in $\mathbb{K}$, for any field $\mathbb{K}$ (see [13, 8]), if polynomials are given in monomial basis. Table 1 exhibits the arithmetic complexity of two well known algorithms in the case of polynomials of $\mathbb{R}[x]$. One is due to Karatsuba and Ofman [16] and has an asymptotic complexity of $O(n^{\log_2 3})$ operations in $\mathbb{K}$; the other one is based on DFT computation using complex FFT and it has an asymptotic complexity of $O(n \log n)$ operations in $\mathbb{K}$, see [13, algorithm 8.16] and [21, 8] for further details. One can see [15, 22] for more details on complex FFT. We also give in Table 1 the exact number of operations in $\mathbb{R}$ for the schoolbook method. From now on, we will use $\log n$ notation to refer to $\log_2 n$.

Table 1: Exact number of operations to multiply two polynomials of $\mathbb{R}[x]$ with degree $n-1$ in monomial basis s.t. $n = 2^k$

Algorithm	**nb. of multiplications**	**nb. of additions**
Schoolbook	n^2	$(n-1)^2$
Karatsuba	$n^{\log 3}$	$7n^{\log 3} - 7n + 2$
DFT-based$^{(*)}$	$3n \log 2n - 4n + 6$	$9n \log 2n - 12n + 12$

(*) using real-valued FFT of [22] with 3/3 strategy for complex multiplication

To perform fast polynomial multiplication using DFT-based method on real inputs, one need to compute 3 DFT with $2n$ points, n pointwise multiplications with complex numbers and $2n$ multiplications with the real constant $\frac{1}{2n}$. Note that we do not need to perform $2n$ pointwise multiplications since the DFT on real inputs has an hermitian symmetry property. Using Split-Radix FFT of [22] with 3/3 strategy for complex multiplication (3 real additions and 3 real multiplications), one can calculate the DFT with n points of a real polynomial with $\frac{n}{2}\log n - \frac{3n}{2} + 2$ real multiplications and $\frac{3n}{2}\log n - \frac{5n}{2} + 4$ additions. Adding all the involved operations gives the arithmetic operation count given in Table 1. Note that one can even decrease the number of operations by using the modified split-radix FFT of [15], yielding an overall asymptotic complexity of $\frac{34}{3}n\log 2n$ instead of $12n\log 2n$.

In the following, we will use the function $\mathsf{M}(n)$ to denote the number of operations in $\mathbb{R}$ to multiply polynomials of degree less than n when using the monomial basis. For instance, $\mathsf{M}(n) = O(n^{\log_2 3})$ with Karatsuba's algorithm. In order to simplify the notations, we assume throughout the rest of the paper that polynomials are of degree $d = n - 1$ with $n = 2^k$.

3 Polynomial Multiplication in Chebyshev Basis

Chebyshev polynomials of the first kind on the interval $[-1, 1]$ are defined by

$$T_k(x) = \cos(k\ \arccos(x)), \quad k \in \mathbb{N}^* \text{ and } x \in [-1, 1].$$

According to this definition, one can remark that these polynomials are orthogonal polynomials. The following recurrence relation holds:

$$\begin{cases} T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \\ T_0(x) = 1 \\ T_1(x) = x \end{cases}$$

It is obvious from this relation that the i-th Chebyshev polynomial $T_i(x)$ has degree i in x. Therefore, it is easy to show that $(T_i(x))_{i\geq 0}$ form a basis of the $\mathbb{R}$-vector space of $\mathbb{R}[x]$. Hence, every polynomial $f \in \mathbb{R}[x]$ can be expressed as a linear combination of $T_i(x)$. This representation is called the Chebyshev expansion. In the rest of this paper we will refer to this representation as the Chebyshev basis.

Arithmetic operations in Chebyshev basis are not as easy as in the classical monomial basis, in particular for the multiplication. Indeed, the main difficulty comes from the fact that the product of two basis elements spans over two other basis elements. The following relation illustrates this property:

$$T_i(x)\ T_j(x) = \frac{T_{i+j}(x) + T_{|i-j|}(x)}{2}, \quad \forall i, j \in \mathbb{N} \tag{1}$$

3.1 Quadratic Algorithms

According to (1), one can derive an algorithm to perform the product of two polynomials given in Chebyshev basis using a quadratic number of operations in $\mathbb{R}$. This method is often called the "direct method". Let two polynomials $a, b \in \mathbb{R}[x]$ of degree $d = n - 1$ given in Chebyshev basis :

$$a(x) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(x) \text{ and } b(x) = \frac{b_0}{2} + \sum_{k=1}^{d} b_k T_k(x).$$

The $2d$ degree polynomial $c(x) = a(x)\ b(x) \in \mathbb{R}[x]$ expressed in Chebyshev basis can computed using the following formula [1]:

$$c(x) = \frac{c_0}{2} + \sum_{k=1}^{2d} c_k T_k(x)$$

such that

$$2c_k = \begin{cases} a_0 b_0 + 2\sum_{l=1}^{d} a_l b_l & \text{for } k = 0, \\ \sum_{l=0}^{k} a_{k-l} b_l + \sum_{l=1}^{d-k} (a_l b_{k+l} + a_{k+l} b_l) & \text{for } k = 1, ..., d-1, \\ \sum_{l=k-d}^{d} a_{k-l} b_l & \text{for } k = d, ..., 2d. \end{cases} \quad (2)$$

The number of operations in $\mathbb{R}$ to compute all the coefficients of $c(x)$ is exactly [1, 17]:

- $n^2 + 2n - 1$ multiplications,
- $\frac{(n-1)(3n-2)}{2}$ additions.

Lima *et. al* recently proposed in [17] a novel approach to compute the coefficient of $c(x)$ which reduces the number of multiplications. The total number of operations in $\mathbb{R}$ is then:

- $\frac{n^2 + 5n - 2}{2}$ multiplications,
- $3n^2 + n^{\log 3} - 6n + 2$ additions.

The approach in [17] is to compute the terms $\sum a_{k-l} b_l$ using Karatsuba's algorithm [16] on polynomial $a(x)$ and $b(x)$ as if they were in monomial basis.

Of course, this does not give all the terms needed in (2). However, by re-using all partial results appearing along the recursive structure of Karatsuba's algorithm, the authors are able to compute all the terms $a_l b_{k+l} + a_{k+l} b_l$ with less multiplication than the direct method. Even if the overall number of operations in $\mathbb{R}$ is higher than the direct method, the balance between multiplication and addition is different. The author claims this may have an influence on architectures where multiplier's delay is much more expensive than adder's one.

3.2 Quasi-linear Algorithms

One approach to get quasi-linear time complexity is to use the discrete cosine transform (DCT-I). The idea is to transform the input polynomials by using forward DCT-I, then perform a pointwise multiplication and finally transform the result back using backward DCT-I. An algorithm using such a technique has been proposed in [1] and achieves a complexity of $O(n \log n)$ operations in $\mathbb{R}$. As mentioned in [17], by using the cost of the fast DCT-I algorithm of [9] one can deduce the exact number of operations in $\mathbb{R}$. However, arithmetic operation count in [17] is partially incorrect, the value should be corrected to:

- $3n \log 2n - 2n + 3$ multiplications,
- $(9n + 3) \log 2n - 12n + 12$ additions.

DCT-I algorithm of [9] costs $\frac{n}{2}\log n - n + 1$ multiplications and $\frac{3n}{2}\log n - 2n + \log n + 4$ additions when using n sample points. To perform the complete polynomial multiplication, one needs to perform 3 DCT-I with $2n$ points, $2n$ pointwise multiplications and $2n$ multiplications by the constant $\frac{1}{2n}$. Adding all the operations count gives the arithmetic cost given above.

4 Reduction To Monomial Basis Case

4.1 Using Basis Conversions

One can achieve a reduction to the monomial basis case by converting the input polynomials given in Chebyshev basis to the monomial basis, then perform the multiplication in the latter basis and finally convert the product back. Hence, the complexity directly relies on the ability to perform the conversions between the Chebyshev and the monomial basis. In [4], authors have proved that conversion between these two basis can be achieved in $O(\mathsf{M}(n))$ operations. Assuming such reductions have a constant factor greater than or equal to one, which is the case to our knowledge, the complete multiplication in Chebyshev basis would requires an amount of operation larger than $3\mathsf{M}(n)$. In the next section, we provide a new reduction scheme which decrease the constant factor of the reduction to exactly two.

4.2 Our Direct Approach

As seen in Section 3.1, Lima *et. al*'s approach [17] is interesting since it introduces the use of monomial basis algorithms (i.e. Karatsuba's one) into Chebyshev basis algorithm. The main idea in [17] is to remark that the terms $\sum a_{k-l}b_l$ in (2) are convolutions of order k. Hence, they are directly calculated in the product of the two polynomials

$$\begin{aligned} \bar{a}(x) &= a_0 + a_1x + a_2x^2 + ... + a_dx^d, \\ \bar{b}(x) &= b_0 + b_1x + a_2x^2 + ... + b_dx^d. \end{aligned} \quad (3)$$

This product gives the polynomials

$$\bar{f}(x) = \bar{a}(x)\ \bar{b}(x) = f_0 + f_1x + f_2x^2 + ... + f_{2d}x^{2d}.$$

Each coefficient f_k of the polynomial $\bar{f}(x)$ corresponds to the convolution of order k. Of course, this polynomial product can be calculated by any of the existing monomial basis algorithms (e.g. those of Section 2). Unfortunately, this gives only a partial reduction to monomial basis multiplication. We now extend this approach to get a complete reduction.

Using coefficients $\bar{f}(x)$ defined above one can simplify (2) to

$$2c_k = \begin{cases} f_0 + 2\sum_{l=1}^{d} a_l b_l & \text{for } k = 0, \\ f_k + \sum_{l=1}^{d-k}(a_l b_{k+l} + a_{k+l} b_l) & \text{for } k = 1, ..., d-1, \\ f_k & \text{for } k = d, ..., 2d. \end{cases} \quad (4)$$

In order to achieve the complete multiplication, we need to compute the three following summation terms for $k = 1 \dots d-1$:

$$\sum_{l=1}^{d} a_l b_l\ ,\ \sum_{l=1}^{d-k} a_l b_{k+l} \text{ and } \sum_{l=1}^{d-k} a_{k+l} b_l. \quad (5)$$

Let us define the polynomial $\bar{r}(x)$ as the reverse polynomial of $\bar{a}(x)$:

$$\bar{r}(x) = \bar{a}(x^{-1})x^d = r_0 + r_1 x + r_2 x^2 + \ldots + r_d x^d.$$

This polynomial satisfies $r_i = a_{d-i}$ for $i = 0 \ldots d$. Let the polynomial $\bar{g}(x)$ be the product of the polynomials $\bar{r}(x)$ and $\bar{b}(x)$. Thus, we have

$$\bar{g}(x) = \bar{r}(x)\ \bar{b}(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{2d} x^{2d}.$$

The coefficient of this polynomials satisfies the following relation for $k = 0 \ldots d$:

$$g_{d+k} = \sum_{l=0}^{d-k} r_{d-l} b_{k+l} \text{ and } g_{d-k} = \sum_{l=0}^{d-k} r_{d-k-l} b_l.$$

According to the definition of $\bar{r}(x)$ we have:

$$g_{d+k} = \sum_{l=0}^{d-k} a_l b_{k+l} \text{ and } g_{d-k} = \sum_{l=0}^{d-k} a_{k+l} b_l. \tag{6}$$

All the terms defined in (5) can be easily deduced from the coefficients g_{d+k} and g_{d-k} of the polynomial $\bar{g}(x)$. This gives the following simplification for (4)

$$2c_k = \begin{cases} f_0 + 2(g_d - a_0 b_0) & \text{for } k = 0, \\ f_k + g_{d-k} + g_{d+k} - a_0 b_k - a_k b_0 & \text{for } k = 1, ..., d-1, \\ f_k & \text{for } k = d, ..., 2d. \end{cases} \tag{7}$$

Applying (7), one can derive an algorithm which satisfies an algorithmic reduction to polynomial multiplication in monomial basis. This algorithm is identified as `PM-Chebyshev` below.

4.3 Complexity Analysis

Algorithm `PM-Chebyshev` is exactly an algorithmic translation of (7). Its correctness is thus immediate from (4) and (6).

Its complexity is $O(\mathsf{M}(n)) + O(n)$ operations in $\mathbb{R}$. It is easy to see that coefficients f_k and g_k are computed by two products of polynomials of degree $d = n - 1$ given in monomial basis. This exactly needs $2M(n)$ operations in $\mathbb{R}$. Note that defining polynomials $\bar{a}(x), \bar{b}(x)$ and $\bar{r}(x)$ does not need any operations in $\mathbb{R}$. The complexity of the algorithm is therefore deduced from the number of operations in (7) and the fact that $d = n - 1$.

The exact number of operations in $\mathbb{R}$ of Algorithm `PM-Chebyshev` is $2\mathsf{M}(n) + 8n - 10$. The extra linear operations are divided into $4n-4$ multiplications and $4n-6$ additions. It is possible to decrease these numbers by setting to zero the constant coefficient of the polynomials $\bar{a}(x)$ and $\bar{b}(x)$ (i.e. $a_0 = b_0 = 0$) just before the computation of $\bar{g}(x)$. Indeed, this removes all the occurrences of a_0 and b_0 in (6) which gives the following relation:

$$g_{d+k} = \sum_{l=1}^{d-k} a_l b_{k+l} \text{ and } g_{d-k} = \sum_{l=1}^{d-k} a_{k+l} b_l, \tag{8}$$

Algorithm 1: `PM-Chebyshev`

Input : $a(x), b(x) \in \mathbb{R}[x]$ of degree $d = n-1$ s.t. $a(x) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(x)$ and

$b(x) = \frac{b_0}{2} + \sum_{k=1}^{d} b_k T_k(x)$.

Output: $c(x) \in \mathbb{R}[x]$ of degree $2d$ s.t. $c(x) = a(x)\ b(x) = \frac{c_0}{2} + \sum_{k=1}^{2d} c_k T_k(x)$.

begin

let $\bar{a}(x)$ and $\bar{b}(x)$ as in (3)

$\bar{f}(x) := \bar{a}(x)\ \bar{b}(x)$

$\bar{g}(x) := \bar{a}(x)\ \bar{b}(x^{-1})\ x^d$

$c_0 := \frac{f_0}{2} + g_d - a_0 b_0$

for $k = 1$ **to** $d-1$ **do**

$c_k := \frac{1}{2}(f_k + g_{d-k} + g_{d+k} - a_0 b_k - a_k b_0)$

for $k = d$ **to** $2d$ **do**

$c_k := \frac{1}{2} f_k$

return $h(x)$

end

and therefore simplifies (7) to

$$2c_k = \begin{cases} f_0 + 2g_d & \text{for } k = 0, \\ f_k + g_{d-k} + g_{d+k} & \text{for } k = 1, ..., d-1, \\ f_k & \text{for } k = d, ..., 2d. \end{cases} \tag{9}$$

Embedding this tricks into Algorithm `PM-Chebyshev` leads to an exact complexity of $2\mathsf{M}(n) + 4n - 3$ operations in $\mathbb{R}$, where extra linear operations are divided into $2n-1$ multiplications and $2n-2$ additions.

Table 2 exhibits the exact number of arithmetic operation needed by Algorithm `PM-Chebyshev` depending on the underlying algorithm chosen to perform monomial basis multiplication. We separate multiplications from additions in order to offer a fair comparison to [17] and we use results in Table 1 for $\mathsf{M}(n)$ costs.

4.4 Special Case of DFT-based Multiplication

When using DFT-based multiplication, we can optimize the Algorithm `PM-Chebyshev` in order to further reduce the number of operations. In particular, we can remark that Algorithm `PM-Chebyshev` needs two multiplications in monomial basis using operands $\bar{a}(x), \bar{b}(x)$ and

Table 2: Arithmetic operation count in Algorithm `PM-Chebyshev`

M(n)	**nb. of multiplication**	**nb. of addition**
Schoolbook	$2n^2 + 2n - 1$	$2n^2 - 2n$
Karatsuba	$2n^{\log 3} + 2n - 1$	$14n^{\log 3} - 12n + 2$
DFT-based$^{(*)}$	$6n \log 2n - 6n + 11$	$18n \log 2n - 22n + 22$

(*) using real-valued FFT of [22] with 3/3 strategy for complex arithmetic

$\bar{a}(x), \bar{b}(x^{-1})x^d$. Therefore, applying the generic scheme of Algorithm `PM-Chebyshev`, we compute twice the DFT transform of $\bar{a}(x)$ on $2n$ points. The same remark applies to the DFT transform of $\bar{b}(x)$ and $\bar{r}(x) = \bar{b}(x^{-1})x^d$ which can be deduced one from the other at a cost of a permutation plus $O(n)$ operations in $\mathbb{R}$.

Indeed, we have

$$\texttt{DFT}_{2n}(\bar{b}) = [\ \bar{b}(w^k)\ ]_{k=0\ldots 2n-1},$$
$$\texttt{DFT}_{2n}(\bar{r}) = [\ \bar{b}(w^{-k})\ \omega^{kd}\ ]_{k=0\ldots 2n-1}.$$

Since $\omega = e^{\frac{-2i\pi}{2n}}$ by definition of the DFT, we have $\omega^{2n} = 1$ and therefore :

$$\omega^k = \omega^{k-2n} \text{ and } \omega^{-k} = \omega^{2n-k} \text{ for } k \in \mathbb{N}.$$

This gives :

$$\texttt{DFT}_{2n}(\bar{r}) = [\ \bar{b}(w^{2n-k})\ \omega^{dk}\ ]_{k=0\ldots 2n-1}.$$

Considering the DFT as an evaluation process, we have

$$\bar{r}(w^k) = (\omega_d)^k\ \bar{b}(w^{2n-k}) \text{ for } k = 0 \ldots 2n-1$$

where $\omega_d = \omega^d = e^{\frac{-2i\pi d}{2n}}$. We can easily see that computing $\texttt{DFT}_{2n}(\bar{r})$ is equivalent to reverse the values of $\texttt{DFT}_{2n}(\bar{b})$ and multiply them by the adequate power of ω_d. This process needs exactly a permutation plus $4n-2$ multiplications in $\mathbb{R}$, which is much less than the $O(n \log n)$ cost of the FFT.

Remark 1. *Instead of applying a permutation, one can use the hermitian symmetry property of real input DFT. In other words, it is equivalent to say that $\bar{b}(\omega^{2n-k})$ is equal to the complex conjugate of $\bar{b}(\omega^k)$.*

This remark has no influences on the complexity analysis but for real implementation it replaces memory swaps by modifications of the sign in the complex numbers structure. If data does not fit in cache, this might reduce the number of memory access and cache misses, and therefore provide better performances.

Using these considerations, one can modify Algorithm `PM-Chebyshev` in order to save almost the computation of 2 DFTs. Hence, we obtain an arithmetic cost in this case of:

- $4n \log 2n + 4n + 5$ multiplications,
- $12n \log 2n - 12n + 14$ additions.

Table 3: Exact complexity for polynomial multiplication in Chebyshev basis, with degree $n-1$

Algorithm	**nb. of operations in** $\mathbb{R}$
Direct method	$2.5n^2 - 0.5n$
Lima *et. al* [17]	$3.5n^2 + n^{\log 3} - 3.5n + 1$
DCT-based	$(12n+3)\log 2n - 14n + 15$
`PM-Chebyshev` (Schoolbook)	$4n^2 - 1$
`PM-Chebyshev` (Karatsuba)	$16n^{\log 3} - 10n + 1$
`PM-Chebyshev` (DFT-based)	$16n \log 2n - 8n + 19$

4.5 Comparisons With Previous Methods

We now compare the theoretical complexity of our new method with existing algorithms presented in Section 3.

In Table 3, we report the exact number of operations in $\mathbb{R}$ for each methods. One can conclude from this table that the asymptotically fastest multiplication is the one using DCT [1]. However, according to the constants and the non-leading terms in each cost function, the DCT-based method is not always the most efficient, especially when polynomial degrees tend to be very small. Furthermore, we do not differentiate the cost of additions and multiplications which does not reflect the reality of computer architecture.

In Figure 1, one can find the speedup of each methods compared to the Direct method. We provide different cost models to capture a little bit more the reality of nowadays computers where the delays of floating point addition and multiplication may differ by a factor of 4 at large. Note that both axis use a logarithmic scale.

First, we can remark that changing cost model only affect the trade-off between methods for small polynomials (i.e. size less than 16). As expected for large degrees, the DCT-based method is always the fastest and our Algorithm `PM-Chebyshev` (DFT-based) is catching up with it since they mostly differ by a constant factor. However, when polynomial degrees tend to be small (less than 10) the Direct method is becoming the most efficient even if it has a quadratic complexity.

As already mentioned in [17], the method of *Lima et. al* tends to become more efficient than the direct method for small polynomials when the cost model assumes that one floating point multiplication cost more than three floating point additions. However, practical constraint such as recursivity, data read/write or cache access have an impact on performance, as we will see, in Section 5 and need to be considered.

5 Implementation and Experimentations

In order to compare our theoretical conclusions with practical computations, we develop a software implementation of our Algorithm `PM-Chebyshev` and we report here its practical performances. As a matter of comparison, we provide implementations for previous know methods: namely the Direct method and the DCT-based method. For the Direct method, a naive implementation with double loop has been done, while for the DCT-one we re-use existing software to achieve best possible performances.

Figure 1: Theoretical speedup of polynomial multiplication in Chebyshev basis with different cost models.

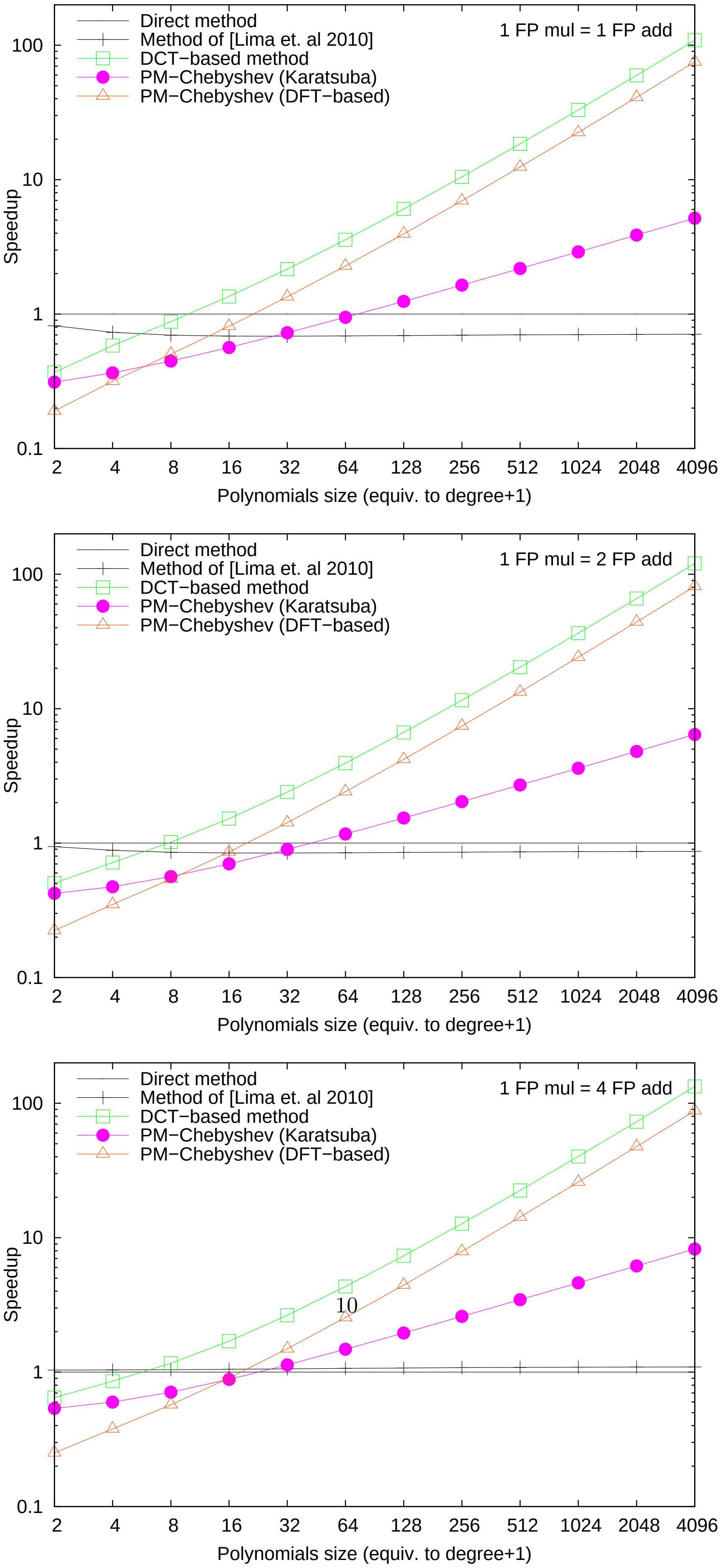


Figure 2: Generic C++ code achieving the reduction to monomial basis multiplication.

```
template<class T, void mulM(vector<T>&,
                            const vector<T>&,
                            const vector<T>&)>
void mulC(       vector<T>& c,
           const vector<T>& a,
           const vector<T>& b){
  size_t da,db,dc,i;
  da=a.size(); db=b.size(); dc=c.size();

  vector<T> r(db),g(dc);

  for (i=0;i<db;i++)
     r[i]=b[db-1-i];

  mulM(c,a,b);
  mulM(g,a,r);

  for (i=0;i<dc;++i)
     c[i]*=0.5;

  c[0]+=c2[da-1]-a[0]*b[0];

  for (i=1;i<da-1;i++)
     c[i]+= 0.5*(g[da-1+i]+g[da-1-i]-a[0]*b[i] -a[i]*b[0]);
}
```

5.1 A Generic Code

We design a C++ code to implement Algorithm `PM-Chebyshev` in a generic fashion. The idea is to take the polynomial multiplication in monomial basis as a template parameter in order to provide a generic function. We decided to manipulate polynomials as vectors to benefit from the C++ Standard Template Library [19], and thus benefit from genericity on coefficients, allowing the use of either double or single precision floating point numbers. Polynomial coefficients are ordered in the vector by increasing degree. The code given in Figure 2 emphasis the simplicity of our implementation:

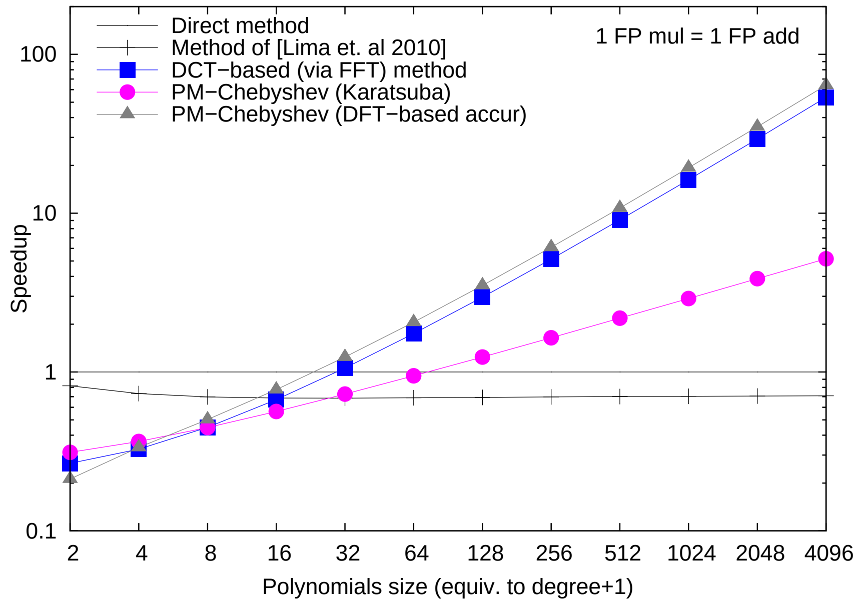
The function `mulM` corresponds to the implementation of the multiplication in monomial basis while the function `mulC` corresponds to the one in Chebyshev basis. The vectors `a` and `b` represents the input polynomials and `c` is the output product. As expected, this code achieves a complete reduction to any implementation of polynomial multiplication in monomial basis, assuming the prototype of the function is compliant. In our benchmarks, we will use this code to reduce to a homemade code implementing the recursive Karatsuba's multiplication algorithm.

5.2 Optimized Code Using DCT and DFT

Many groups and projects have been already involved in designing efficient implementations of discrete transforms such as DCT and DFT. We can cite for instance the Spiral project [20] and the FFTW library effort [12]. In order to benefit from the high efficiency of these works, we build our DCT/DFT based codes on top of the FFTW routines. For both DCT and DFT computations we use FFTW plans with FFTW_MEASURE planning option, which offer optimized code using runtime measurement of several transforms.

As explained in the documentation of the FFTW library, the DCT-I transform using a pre/-post processed real DFT suffers from numerical instability. Therefore, the DCT-I implementation in FFTW is using either a recursive decomposition in smaller optimized DCT-I codelets or a real DFT of twice the size plus some scalings. For the latter case, this means that the complexity

Figure 3: Theoretical speedup of polynomial multiplication in Chebyshev basis.



of the DCT-I code is not reflecting the one of [9] we used in our complexity analysis. Taking this into account, one should replace the $2n$ points DCT-I transforms of Section 3.2 by $4n$ points DFT transforms plus $2n$ multiplications by the real constant 2. This increases the complexity of the DCT-based method to :

- $6n \log 4n - 12n + 6$ multiplications,
- $18n \log 4n - 30n + 12$ additions.

In the following, we will denote this modification as the DCT-based (via FFT) method. In order to make a fair comparison with this DCT method which cares of stability, we shall mention that the Algorithm `PM-chebyshev` (DFT-based) as explained in section 4.4 may suffer in practice from instability issues. Indeed, the trick to compute $\texttt{DFT}_{2n}(\bar{r})$ from $\texttt{DFT}_{2n}(\bar{b})$ needs multiplications by some powers of ω_d and therefore introduces errors in the computed DFT. As we will see in the next section these errors will have an influence on the accuracy of the final product. An alternative to not introduce these numerical errors is therefore to really compute $\texttt{DFT}_{2n}(\bar{r})$ by an FFT calculation, but at a price of more operations. This consideration will increase the theoretical complexity to:

- $5n \log 2n - 3n + 9$ multiplications,
- $15n \log 2n - 17n + 18$ additions.

In the following, we will denote this modification as the `PM-chebyshev` (DFT-based accur).

Considering practical stability issues and its impact on the complexity of the DCT-based and our DFT-based method, we can see in Figure 3 the effect on the theoretical speedups. In particular, our DFT-based reduction is becoming more efficient than the DCT-based when polynomial's degrees are getting larger. This is of course explained by the difference of the constant term in the complexity: $20n \log 2n$ for our method and $24n \log 2n$ for the DCT-based (via FFT).

5.3 Code Validation

As a matter of reliability, we check the validity of all our implementations. First, we check their correctness by verifying the results of their implementations done in a symbolic way using Maple[1] software.

Since we want to perform numerical computations, it is clear that the accuracy of the results may differ from one method to another. It is therefore crucial to investigate their stability to give good statement on the accuracy. It is not the intend of this work to give statements on the accuracy and this task would definitively require a dedicated work. However, in order to give a small insight we did some experiments to emphasis the relative error of every methods. Let us now give the definition of the relative error on polynomials as given in [14].

Definition 5.1. *Let $a(x), b(x)$ be polynomials given in Chebyshev basis with double precision floating point numbers coefficients. We define $\hat{c}(x)$ to be the approximation of the product $a(x)b(x)$ using double precision computation (53 bits of mantissa) and $c(x)$ to be the exact product computed over rational numbers. Using this notation, the relative error $E(\hat{c}(x))$ is defined as*

$$E(\hat{c}(x)) = \frac{\|c(x) - \hat{c}(x)\|_2}{\|c(x)\|_2}$$

where $\|\ldots\|_2$ represents the Euclidean norm of polynomials, i.e. $\|a(x)\|_2 = (\sum_{k=0}^{d} a_k^2)^{\frac{1}{2}}$ where the a_k correspond to the coefficients of $a(x)$.

Following this definition, we have computed the relative error on polynomial products using polynomial inputs having random floating point entries. While the numerical results are computed in double precision floating point numbers, the exact product is computed using the arbitrary precision rational numbers of the GMP[2] library. The relative error is almost computed exactly since only the square root is using floating point approximations, the remaining parts being computed over the rationals. We propose in Figure 4 the measure of the relative error in our experiments. The ordinates axis gives the average relative error of 50 products with different random double precision floating point entries lying between -50 and 50.

As explained in Section 5.2, we can see in this figure that Algorithm `PM-chebyshev` (DFT-based) is clearly suffering from instability issues. In these settings, the replacement of one DFT by few multiplications of complex number powers introduces too many errors, which causes a loss in the accuracy of the final result (e.g. up to three decimal digits can be erroneous). Hopefully, using Algorithm `PM-chebyshev` (DFT-based accur) completely avoids this issue and we get similar accuracy as for the DCT-based methods.

If we change a little bit the settings of our experiment, taking only positive floating point random entries (e.g. in $[0, 50]$), the instability issue of Algorithm `PM-chebyshev` (DFT-based) seems to be less dramatic. This is illustrated in Figure 5 where we can see that relative error of Algorithm `PM-chebyshev` (DFT-based) only differ by at most one decimal digit.

Undoubtedly, these experiments exhibit some stability issues in Algorithm `PM-chebyshev` (DFT-based) while it does not seem to be the case for the other methods based on discrete

[1] www.maplesoft.com

[2] http://gmplib.org/

Figure 4: Experimental measure of the relative error in double precision (Intel Xeon 2GHz). Entries lying in $[-50, 50]$

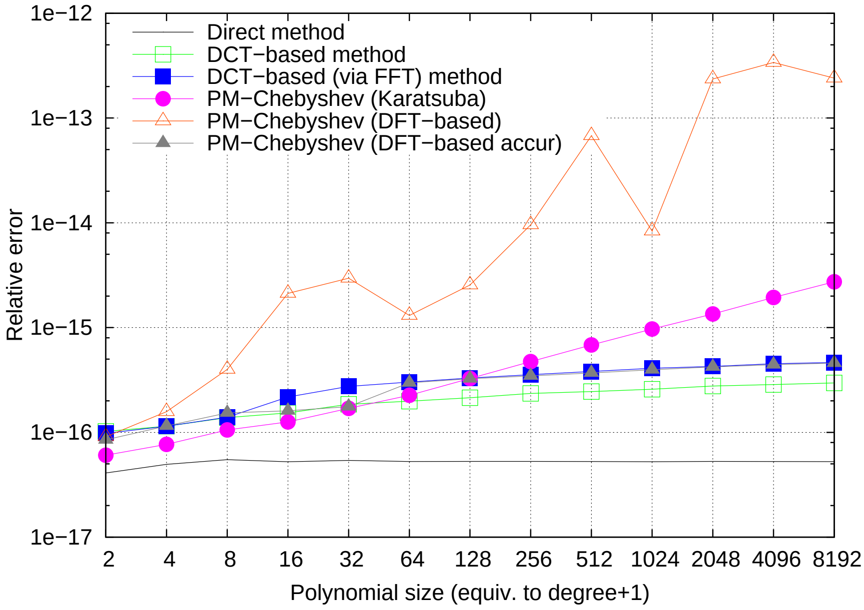
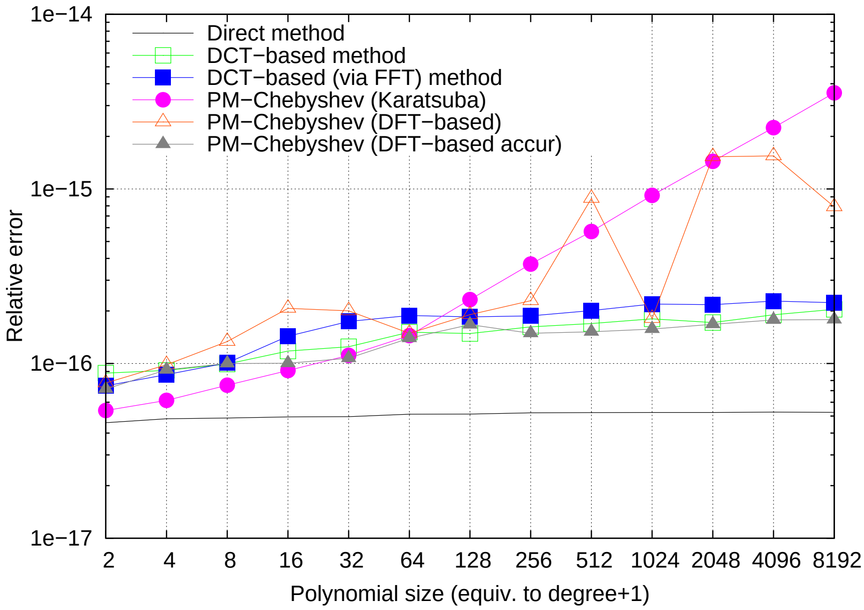


Figure 5: Experimental measure of the relative error in double precision (Intel Xeon 2GHz). Entries lying in $[0, 50]$



transforms. Algorithm PM-chebyshev (Karatsuba) also seems to have some numerical issues. This can be motivated by the nature of Karatsuba method which replaces one multiplication by few additions.

From these experiments we can conclude few thoughts. Algorithm PM-chebyshev (DFT-based accur) seems to offer the same numerical behaviour as the DCT based method, and thus offer a concrete alternative in practice. If accuracy is not a problem, Algorithm PM-chebyshev (DFT-based) will provide an interesting option as it will increase efficiency, see section 5.4. Finally, a theoretical study of the numerical stability of all these methods need to be done to give precise statement on their reliability.

5.4 Benchmarks

We now compare the practical efficiency of the different methods. We performed our benchmarks on an architecture which represents nowadays processors: an Intel Xeon processor 5130 running at 2GHz with 2×4MB of L2 cache. We use the gcc compiler version 4.4.4 with O3 optimization. Even if the platform is multi-core, we did not use any parallel computations and the FFTW library has been built sequential. For each method, we measure the average running time of several polynomial multiplications. All the computations have been done with double precision floating point numbers and with the same data set.

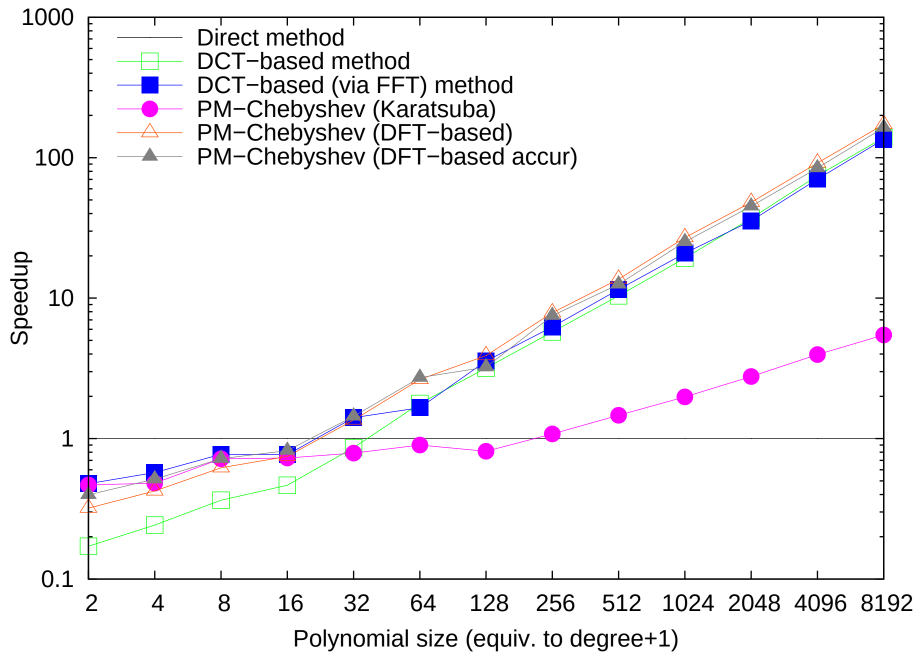
Remark 2. *We only offer an average running time estimate of each algorithms since it is not realistic on nowadays processor to estimate precise running time of computation taking few milliseconds.*

We report in Figure 6 the relative performances to the Direct method implementation for polynomial sizes ranging from 2 to 8192. Both axis use logarithmic scale, and the ordinates axis represents the speedup against Direct method. All times used in this figure are given in the appendix. One can also find in Figure 7 of the appendix more detailed views of the Figure 6.

As expected, one can see on these Figures that the Direct method reveals the most efficient for very small polynomials (i.e. polynomial degrees less than 16). This is explained by the low number of operations required by this method and its simplicity which makes possible several optimizations by the compiler (e.g. loop unrolling). When polynomial sizes are getting larger, the methods based on discrete transforms become the most efficients. In particular, we can see that DCT-based method is catching up with its version based on FFT, which clearly illustrates that DCT-I implementation of FFTW is using a double length FFT, as explained in Section 5.2. Therefore, as expected, our Algorithm PM-Chebyshev (DFT-based) is the most efficient with polynomial sizes greater than 32. In particular, our PM-Chebyshev (DFT-based) implementation is gaining 20% to 30% of efficiency over the DCT-based implementation. Of course, if a more accurate result is needed one may prefer to use the PM-Chebyshev (DFT-based accur) version but at a price of less efficiency : only 15% of gain against DCT-based implementation for polynomials of size 8096. Surprisingly, for polynomials of size 32 and 64 this method reveals to be the most efficient in practice. It seems that the code of the DFT in FFTW library is more efficient for these sizes than our implementation of the algebraic trick described in section 4.4.

Remark 3. *One could have been interested to see the practical behavior of the method of Lima et. al [17]. However, our feelings on the efficiency of such a method lead us to be pessimistic. Even if this method decreases the number of multiplications, it increases the overall number of operations. Moreover, this method needs an important amount of extra memory (i.e.* $O(n^{\log_2 3})$*)*

Figure 6: Practical performances of polynomial multiplication in Chebyshev basis against direct method - Intel Xeon 2GHz (global view).



which definitively increases data access and then should considerably penalize performances. Furthermore, the method is quite complex, especially for the indices management in the separation procedure. Since no detailed algorithm is given in [17] it is not easy to make an implementation and then offer a fair comparison.

Finally, from our benchmarks we observe that the performance of the Karatsuba multiplication does not compete with the Direct method for small polynomials (e.g. size less than 16). Adding the storage of intermediate value within Karatsuba procedure plus the extra quadratic operations needed by the method of Lima et. al [17] will probably make its implementation not competitive with other existing methods.

6 A Note on Problems Equivalence

Let us consider the problem of multiplying two polynomials given by their coefficients in a given basis of the $\mathbb{R}$-vector space of $\mathbb{R}[x]$. We denote this problem in monomial basis as M_{mon} and the one in Chebyshev basis as M_{che}. Under this consideration, one can demonstrate the following theorem:

Theorem 6.1. *Problem M_{mon} and M_{che} are equivalent under a linear time transform, $M_{mon} \equiv_L M_{che}$ and the constant of both transforms is equal to two.*

Proof. As we have shown in Section 4 the problem of multiplying polynomials in Chebyshev basis linearly reduces to the multiplication in monomial basis, and the constant in the reduction is two. Thus we have already demonstrate $M_{che} \leq_L M_{mon}$.

We can show that $M_{mon} \leq_L M_{che}$ by using (4). Indeed, we can see from (4) that the $d+1$ leading coefficients of the product in Chebyshev basis exactly match with the ones in monomial basis on the same input coefficients. It is easy to show that the remaining d coefficients can be read from the product in Chebyshev basis of the reversed inputs.

Let us denote $\times_c$ the multiplication in Chebyshev basis and $\times$ the one in monomial basis. Consider the two polynomials $\bar{a}, \bar{b} \in \mathbb{R}[x]$ given in monomial basis as

$$\bar{a}(x) = \sum_{k=0}^{d} a_k x^k \text{ and } \bar{b}(x) = \sum_{k=0}^{d} b_k x^k.$$

Consider the polynomials $a(x), b(x), \alpha(x)$ and $\beta(x)$ sharing the same coefficients as $\bar{a}(x)$ and $\bar{b}(x)$ but expressed in Chebyshev basis:

$$a(x) = \sum_{k=0}^{d} a_k T_k(x) \ , \ b(x) = \sum_{k=0}^{d} b_k T_k(x),$$

$$\alpha(x) = \sum_{k=0}^{d} a_{d-k} T_k(x) \ , \ \beta(x) = \sum_{k=0}^{d} b_{d-k} T_k(x).$$

The coefficients c_k of the polynomial $\bar{c}(x) = \bar{a}(x) \times \bar{b}(x)$ expressed in monomial basis can be read from the coefficients of the polynomials

$$f(x) = a(x) \times_c b(x) \text{ and } g(x) = \alpha(x) \times_c \beta(x)$$

using the relation

$$c_k = \begin{cases} g_{d-1-k} & \text{for } k = 0 \dots d-1, \\ f_k & \text{for } k = d \dots 2d. \end{cases}$$

This clearly demonstrates that $M_{mon} \leq_L M_{che}$ and thus complete the proof. □

7 Conclusion

We described yet another method to reduce the multiplication of polynomials given in Chebyshev basis to the multiplication in the monomial basis. Our method decreases the constant of the problem reduction and therefore offer a better complexity than the ones using basis conversions. Moreover, since our method does not rely on basis conversions, it might offer more numerical stability as it could be when converting coefficients to other basis. As we already mention, the problem of numerical stability is of great interest and should be treated as a dedicated article.

Our `PM-Chebyshev` algorithm offers an efficient alternative to any existing quasi-linear algorithms. In particular, it allows to use Fast Fourier Transform of half length of the one needed by the specialized DCT-based method, which is an alternative when DCT codes are not available or sufficiently efficient. In such a case, our method achieves the best performances among all the available method for large degree polynomials.

Although our reduction scheme using Karatsuba's method is not as efficient as one could have expected for polynomial of medium size, further work to optimize its implementation should be investigated.

Finally, our attention in this work has been focused only on polynomials of $\mathbb{R}[x]$ but our approach is still valid for Chebyshev polynomials defined over other domains, as Dickson polynomials over finite fields for example.

References

[1] G. Baszenski and M. Tasche. Fast polynomial multiplication and convolution related to the discrete cosine transform. *Linear Algebra and its Application*, 252(1-3):1–25, 1997.

[2] Z. Battles and L. Trefethen. An extension of matlab to continuous fractions and operators. *SIAM J. Sci. Comp*, 2004.

[3] A. Bostan, B. Salvy, and E. Schost. Power series composition and change of basis. In *ISSAC '08: Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 269–276, New York, NY, USA, 2008. ACM.

[4] A. Bostan, B. Salvy, and E. Schost. Fast conversion algorithms for orthogonal polynomials. *Linear Algebra and its Applications*, 432(1):249–258, January 2010.

[5] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover N.Y., New York, 2001.

[6] R. Brent and P. Zimmermann. *Modern Computer Arithmetic*. August 2010. version 0.5.3,http://www.loria.fr/ zimmerma/mca/mca-cup-0.5.3.pdf.

[7] N. Brisebarre and M. Joldes. Chebyshev interpolation polynomial-based tools for rigorous computing. In *ISSAC 2010: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*. ACM Press, 2006.

[8] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[9] S. C. Chan and K. L. Ho. Direct methods for computing discrete sinusoidal transforms. *Radar and Signal Processing, IEE Proceedings F*, 137(6):433–442, 1990.

[10] S. A. Cook. On the minimum computation time of functions. Master's thesis, Harvard University, May 1966.

[11] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[12] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[13] J. v. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[14] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[15] S. Johnson and M. Frigo. A modified split-radix FFT with fewer arithmetic operations. *IEEE Transactions on Signal Processing*, 55(1):111–119, January 2007.

[16] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Doklady Akademii Nauk SSSR*, 145(2):293–294, 1962.

[17] J. B. Lima, D. Panario, and Q. Wang. A Karatsuba-based algorithm for polynomial multiplication in Chebyshev form. *IEEE Transactions on Computers*, 59:835–841, 2010.

[18] J. C. Mason and D. C. Handscomb. *Chebyshev polynomials.* Chapman and Hall/CRC, Boca Raton, FL, 2002.

[19] D. R. Musser, G. J. Derge, and A. Saini. *The STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

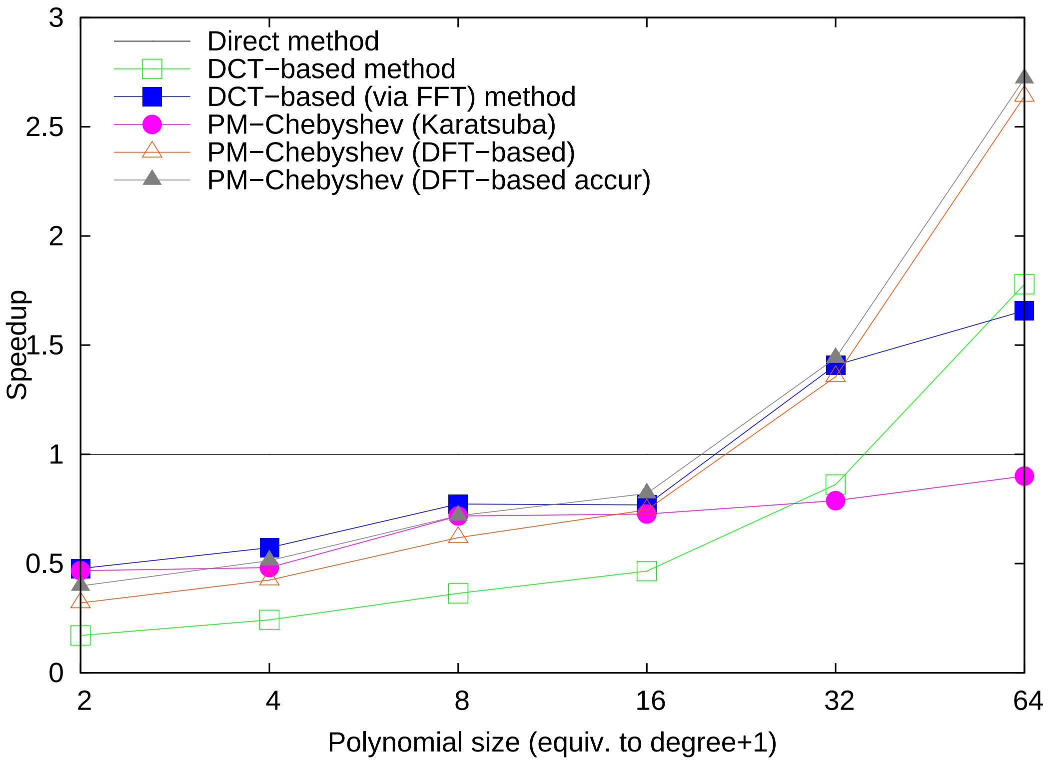
[20] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE*, 93(2):232–275, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[21] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[22] H. Sorensen, D. Jones, M. Heideman, and C. Burrus. Real-valued fast fourier transform algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(6):849–863, 1987.

[23] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Math*, 3:714–716, 1963.

Figure 7: Practical performances of polynomial multiplication in Chebyshev basis against direct method - Intel Xeon 2GHz (partial view).



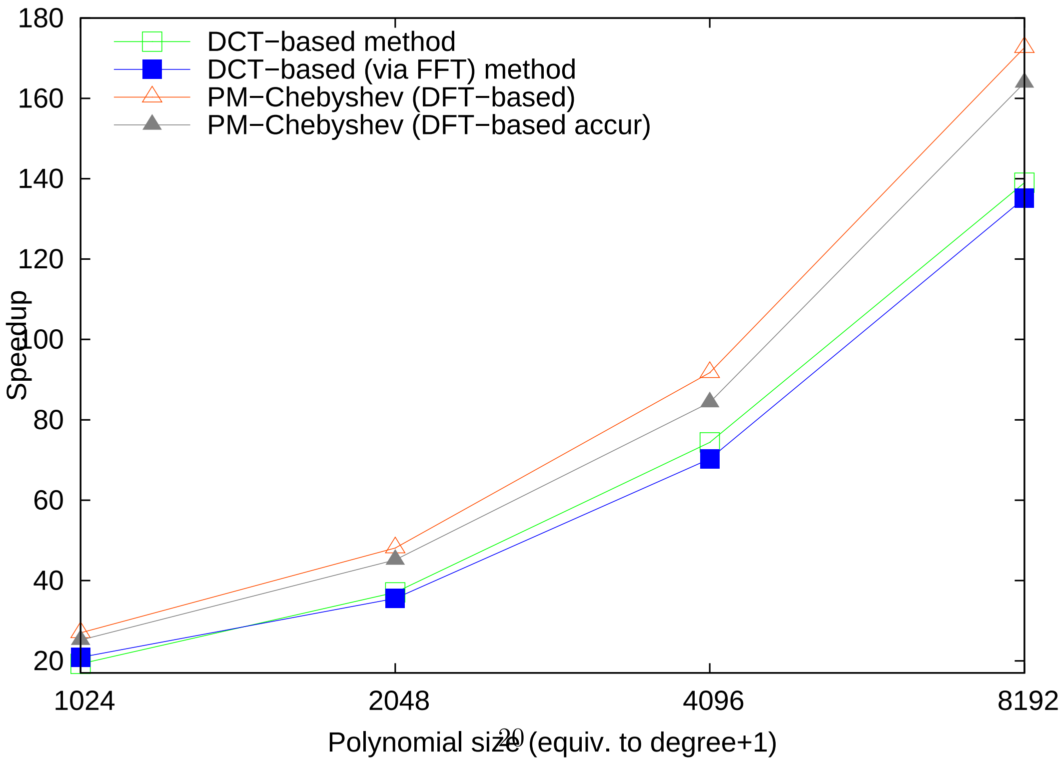


Table 4: Times of polynomial multiplication in Chebyshev basis (given in μs) on Intel Xeon 2GHz platform.

n	**Direct**	**DCT-based**	**DCT-based (FFT)**	**PM-Cheby (Kara)**	**PM-Cheby (DFT)**	**PM-Cheby (DFT accur)**
2	0.18	1.08	0.38	0.39	0.57	0.46
4	0.28	1.15	0.48	0.58	0.66	0.54
8	0.57	1.58	0.74	0.80	0.93	0.80
16	1.13	2.43	1.47	1.56	1.52	1.38
32	3.73	4.33	2.65	4.74	2.75	2.59
64	13.44	7.56	8.11	14.93	5.09	4.94
128	50.06	15.76	14.04	61.68	12.84	15.52
256	185.48	32.29	29.69	171.78	23.58	24.70
512	716.51	69.00	62.13	489.29	52.46	57.07
1024	2829.78	146.94	135.47	1427.82	104.94	112.40
2048	11273.20	304.55	317.35	4075.72	234.41	249.88
4096	47753.40	642.17	679.50	12036.00	520.56	566.43
8192	194277.00	1397.42	1437.42	35559.60	1125.40	1185.41

PM-Cheby stands for `PM-Chebyshev` algorithm.