On Polynomial Multiplication Complexity in Chebyshev Basis

Pascal Giorgi

INRIA Rocquencourt, Algorithms Project’s Seminar
November 29, 2010.
Outline

1. Polynomial multiplication in monomial basis
2. Polynomial multiplication in Chebyshev basis
3. A more direct reduction from Chebyshev to monomial basis
4. Implementations and experimentations
5. Conclusion
Outline

1. Polynomial multiplication in monomial basis
2. Polynomial multiplication in Chebyshev basis
3. A more direct reduction from Chebyshev to monomial basis
4. Implementations and experimentations
5. Conclusion
Arithmetic of polynomials has been widely studied, and its complexity is well established:

Let \( f, g \in K[x] \) two polynomials of degree \( d < n = 2^k \) and \( K \) a field. One can compute the product \( f \cdot g \in K[x] \) in

- \( O(n^2) \) op. in \( K \) [schoolbook method]
- \( O(n^{1.58}) \) op. in \( K \) [Karatsuba's method]
- \( O(n^{\log_{r+1}(2r+1)}) \) op. in \( K \), \( \forall r > 0 \) [Toom-Cook method]
- \( O(n \log n) \) op. in \( K \) [DFT-based method] assuming \( K \) has a \( n^{th} \) primitive root of unity.
Polynomial multiplication

Arithmetic of polynomials has been widely studied, and its complexity is well established:

Let \( f, g \in \mathbb{K}[x] \) two polynomials of degree \( d < n = 2^k \) and \( \mathbb{K} \) a field. One can compute the product \( f \cdot g \in \mathbb{K}[x] \) in

- \( O(n^2) \) op. in \( \mathbb{K} \) [schoolbook method]
- \( O(n^{1.58}) \) op. in \( \mathbb{K} \) [Karatsuba's method]
- \( O(n^{\log_{r+1}(2r+1)}) \) op. in \( \mathbb{K} \), \( \forall r > 0 \) [Toom-Cook method]
- \( O(n \log n) \) op. in \( \mathbb{K} \) [DFT-based method] assuming \( \mathbb{K} \) has a \( n^{th} \) primitive root of unity.

Remark

this result assumes that \( f, g \) are given in the monomial basis \((1, x, x^2, \ldots)\)
Polynomial multiplication in $\mathbb{R}[x]$

All these methods work on polynomials over $\mathbb{R}[x]$.

- Most of them use evaluation/interpolation technique.
- $O(n \log n)$ method needs to perform the computation in the algebraic closure of $\mathbb{R}$, which is $\mathbb{C}$, to have primitive roots of unity.
Polynomial multiplication in $\mathbb{R}[x]$

All these methods work on polynomials over $\mathbb{R}[x]$.

- most of them use evaluation/interpolation technique
- $O(n \log n)$ method needs to perform the computation in the algebraic closure of $\mathbb{R}$, which is $\mathbb{C}$, to have primitive roots of unity.

Let $f, g \in \mathbb{R}[x]$ with degree $d < n = 2^k$, the sketch to compute $h = fg$ is:

Set $\omega = e^{-\frac{2i\pi}{2n}} \in \mathbb{C}$ and calculate

- $[f(1), f(\omega), \ldots, f(\omega^{2n-1})] \in \mathbb{C}^n$ using DFT on $f$ and $\omega$
- $[g(1), g(\omega), \ldots, g(\omega^{2n-1})] \in \mathbb{C}^n$ using DFT on $g$ and $\omega$
- $fg = f(0)g(0) + f(\omega)g(\omega)x + \ldots + f(\omega^{2n-1})g(\omega^{2n-1})x^{2n-1} \in \mathbb{C}[x]$
- $[h_0, h_1, \ldots, h_{2n-1}] \in \mathbb{R}^n$ using DFT on $fg$ and $\omega^{-1}$ plus scaling
Let \( f(x) = f_0 + f_1 x + \ldots + f_{n-1} x^{n-1} \in \mathbb{R}[x] \).

The \( n \)-point Discrete Fourier Transform (DFT\(_n\)) can be defined as follow:

\[
\text{DFT}_n(f) = (F_k)_{k=0}^{n-1} \quad \text{such that} \quad F_k = \sum_{j=0}^{n-1} f_j e^{-\frac{2\pi i}{n} kj} \in \mathbb{C}
\]

which is equivalent to say

\[
\text{DFT}_n(f) = (f(0), f(\omega), \ldots, f(\omega^{n-1}) \quad \text{with} \quad \omega = e^{-\frac{2\pi i}{n}}.
\]
The Fast Fourier Transform (FFT) is a fast method based on a divide and conquer approach [Gauss 19th, Cooley-Tuckey 1965] to compute the DFT\(_n\). It uses the following property:

Let \( f = q_0(x^{n/2} - 1) + r_0 \) and \( f = q_1(x^{n/2} + 1) + r_1 \) s.t. \( \deg r_0, \deg r_1 < \frac{n}{2} \) then \( \forall k \in \mathbb{N} \) s.t. \( k < \frac{n}{2} \)

\[
\begin{align*}
  f(\omega^{2k}) &= r_0(\omega^{2k}), \\
  f(\omega^{2k+1}) &= r_1(\omega^{2k+1}) = \bar{r}_1(\omega^{2k}).
\end{align*}
\]

This means \( \text{DFT}_n(f) \) can be computed using \( \text{DFT}_{n/2}(r_0) \) and \( \text{DFT}_{n/2}(\bar{r}_1) \), yielding a recursive complexity of \( H(n) = 2H(n/2) + O(n) = \tilde{O}(n \log n) \).
Polynomial multiplication in $\mathbb{R}[x]$

Asymptotic complexity is not reflecting real behaviour of algorithms in practice, constant and lower terms in complexity does really matter !!!

Exact number of operations in $\mathbb{R}$ to multiply two polynomials of $\mathbb{R}[x]$ with degree $d < n = 2^k$ in monomial basis

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>nbr of multiplication</th>
<th>nbr of addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook</td>
<td>$n^2$</td>
<td>$(n - 1)^2$</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>$n^{\log_3 \log_3}$</td>
<td>$7n^{\log_3 \log_3} - 7n + 2$</td>
</tr>
<tr>
<td>DFT-based(*)</td>
<td>$3n \log 2n - n + 6$</td>
<td>$9n \log 2n - 12n + 12$</td>
</tr>
</tbody>
</table>

Polynomial multiplication in $\mathbb{R}[x]$

Polynomial Multiplication in monomial basis (theoretical count)

- Schoolbook method
- Karatsuba method
- DFT-based method

1 FP mul = 1 FP add

10 0
10 1
10 2
10 3
10 4
10 5
10 6
10 7
10 8
2 4 8 16 32 64 128 256 512 1024 2048 4096

nbr of operations

Polynomials size (equiv. to degree+1)
Polynomial multiplication in $\mathbb{R}[x]$

Polynomial Multiplication in monomial basis (theoretical count)

1 FP mul = 2 FP add

Schoolbook method
Karatsuba method
DFT–based method
Polynomial multiplication in $\mathbb{R}[x]$

Polynomial Multiplication in monomial basis (theoretical count)

- Schoolbook method
- Karatsuba method
- DFT-based method

1 FP mul = 4 FP add

nbr of operations vs. Polynomials size (equiv. to degree+1)
Monomial basis for polynomial are not the only ones !!!
Monomial basis for polynomial are not the only ones !!!

Chebyshev basis seems to be very useful
In this presentation, I will assume that

- polynomials have degree \( d = n - 1 \) where \( n = 2^k \).
- formula will be using the degree \( d \).
- complexity estimate will be using the number of terms \( n \).
Outline

1. Polynomial multiplication in monomial basis
2. Polynomial multiplication in Chebyshev basis
3. A more direct reduction from Chebyshev to monomial basis
4. Implementations and experimentations
5. Conclusion
Chebyshev Polynomials: a short definition

Chebyshev polynomials of the first kind on $[-1, 1]$ are defined as $T_k(x)$ s.t.

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

These polynomials are orthogonal polynomials defined by the following recurrence relation:

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

Therefore, they can be used to form a base of the $\mathbb{R}$-vector space of $\mathbb{R}[x]$. 
Polynomials in Chebyshev basis

Every \( a \in \mathbb{R}[x] \) can be expressed as a linear combination of \( T_k \):

\[
a(x) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(x)
\]
Every $a \in \mathbb{R}[x]$ can be expressed as a linear combination of $T_k$:

$$a(x) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(x)$$

**Question:**

How fast can we multiply polynomials in Chebyshev basis?
Polynomials in Chebyshev basis

Every \( a \in \mathbb{R}[x] \) can be expressed as a linear combination of \( T_k \):

\[
a(x) = \frac{a_0}{2} + \sum_{k=1}^{d} a_k T_k(x)
\]

**Question:**

How fast can we multiply polynomials in Chebyshev basis?

**Main difficulty**

\[
T_i(x) \cdot T_j(x) = \frac{T_{i+j}(x) + T_{|i-j|}(x)}{2}, \quad \forall i, j \in \mathbb{N}
\]
Multiplication of polynomials in Chebyshev basis

Complexity results:

Let $a, b \in \mathbb{R}[x]$ two polynomials of degree $d = n - 1$ given in Chebyshev basis, one can compute the product $a \cdot b \in \mathbb{R}[x]$ in Chebyshev basis with

- $O(n^2)$ op. in $\mathbb{R}$ [direct method]
- $O(n \log n)$ op. in $\mathbb{R}$ [Baszenski, Tasche 1997]
- $O(M(n))$ op. in $\mathbb{R}$ [Bostan, Salvy, Schost 2010] where $M(n)$ is the cost of the multiplication in monomial basis.

Remark: $M(n)$ method is using basis conversions which increase constant term in the complexity estimate and probably introduce numerical errors. My goal is to get a more direct reduction to the monomial case!!!
Multiplication of polynomials in Chebyshev basis

Complexity results:

Let $a, b \in \mathbb{R}[x]$ two polynomials of degree $d = n - 1$ given in Chebyshev basis, one can compute the product $a \cdot b \in \mathbb{R}[x]$ in Chebyshev basis with

- $O(n^2)$ op. in $\mathbb{R}$ [direct method]
- $O(n \log n)$ op. in $\mathbb{R}$ [Baszenski, Tasche 1997]
- $O(M(n))$ op. in $\mathbb{R}$ [Bostan, Salvy, Schost 2010] where $M(n)$ is the cost of the multiplication in monomial basis.

Remark:

$M(n)$ method is using basis conversions which increase constant term in the complexity estimate and probably introduce numerical errors.
Multiplication of polynomials in Chebyshev basis

Complexity results:
Let \( a, b \in \mathbb{R}[x] \) two polynomials of degree \( d = n - 1 \) given in Chebyshev basis, one can compute the product \( a \cdot b \in \mathbb{R}[x] \) in Chebyshev basis with

- \( O(n^2) \) op. in \( \mathbb{R} \) [direct method]
- \( O(n \log n) \) op. in \( \mathbb{R} \) [Baszenski, Tasche 1997]
- \( O(M(n)) \) op. in \( \mathbb{R} \) [Bostan, Salvy, Schost 2010] where \( M(n) \) is the cost of the multiplication in monomial basis.

Remark:
\( M(n) \) method is using basis conversions which increase constant term in the complexity estimate and probably introduce numerical errors.

My goal is to get a more direct reduction to the monomial case !!!
Let's first recall existing algorithms ...
Multiplication in Chebyshev basis: the direct method

Derived from \( T_i(x). T_j(x) = \frac{T_{i+j}(x)+T|i-j|}(x)}{2}, \quad \forall i, j \in \mathbb{N} \)
Multiplication in Chebyshev basis: the direct method

Derived from \( T_i(x) \cdot T_j(x) = \frac{T_{i+j}(x) + T_{|i-j|}(x)}{2} \), \( \forall i, j \in \mathbb{N} \)

Let \( a, b \in \mathbb{R}[x] \) of degree \( d \), given in Chebyshev basis, then

\[
c(x) = a(x) b(x) = \frac{c_0}{2} + \sum_{k=1}^{2d} c_k T_k(x)
\]

is computed according to the following equation:

\[
2c_k = \begin{cases} 
    a_0 b_0 + 2 \sum_{l=1}^{d} a_l b_l, & \text{for } k = 0, \\
    \sum_{l=0}^{k} a_{k-l} b_l + \sum_{l=1}^{d-k} (a_l b_{k+l} + a_{k+l} b_l), & \text{for } k = 1, \ldots, d - 1, \\
    \sum_{l=k-d}^{d} a_{k-l} b_l, & \text{for } k = d, \ldots, 2d.
\end{cases}
\]
Multiplication in Chebyshev basis: the direct method

Derived from \( T_i(x) \cdot T_j(x) = \frac{T_{i+j}(x) + T_{|i-j|}(x)}{2}, \quad \forall i, j \in \mathbb{N} \)

Let \( a, b \in \mathbb{R}[x] \) of degree \( d \), given in Chebyshev basis, then

\[
c(x) = a(x) b(x) = \frac{c_0}{2} + \sum_{k=1}^{2d} c_k T_k(x)
\]

is computed according to the following equation:

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

using \( n^2 + 2n - 1 \) multiplications and \( \frac{(n-1)(3n-2)}{2} \) additions in \( \mathbb{R} \).
Yet another quadratic method

Lima, Panario and Wang [IEEE TC 2010] proposed another approach yielding a quadratic complexity which lowers down the number of multiplications

\[
\frac{n^2 + 5n - 2}{2} \text{ multiplications in } \mathbb{R},
\]
\[
3n^2 + n^{\log 3} - 6n + 2 \text{ additions in } \mathbb{R}.
\]

Main idea

perform the multiplication on polynomials as if they were in monomial basis and correct the result.
Yet another quadratic method

Lima, Panario and Wang [IEEE TC 2010] proposed another approach yielding a quadratic complexity which lowers down the number of multiplications

- \( \frac{n^2 + 5n - 2}{2} \) multiplications in \( \mathbb{R} \),
- \( 3n^2 + n^{\log 3} - 6n + 2 \) additions in \( \mathbb{R} \).

**Main idea**

perform the multiplication on polynomials as if they were in monomial basis and correct the result.

**Interesting:** this approach is almost halfway from a direct reduction to monomial basis case !!!
Yet another quadratic method [Lima, Panario, Wang 2010]

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

Compute convolutions \( f_k = \sum_{l=0}^{k} a_{k-l} b_l \) using Karatsuba algorithm.

Simplifying direct method to

\[
2c_k = \begin{cases} 
  f_0 + 2 \sum_{l=1}^{d} a_l b_l & \text{for } k = 0, \\
  f_k + \sum_{l=1}^{d-k} (a_l b_{k+l} + a_{k+l} b_l) & \text{for } k = 1, \ldots, d - 1, \\
  f_k & \text{for } k = d, \ldots, 2d.
\end{cases}
\]
Yet another quadratic method [Lima, Panario, Wang 2010]

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

Compute convolutions \( f_k = \sum_{l=0}^{k} a_{k-l} b_l \) using Karatsuba algorithm.

Simplifying direct method to

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

still need to compute remaining part to get the result
Yet another quadratic method [Lima, Panario, Wang 2010]

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

All the terms have been already computed together in convolutions \( f_k \).

- could be recomputed but at an expensive cost
- could use separation technique to retrieve them [Lima, Panario, Wang 2010]

Main idea

exploit the recursive three terms structure \( < a_0 b_0, a_0 b_1 + a_1 b_0, a_1 b_1 > \) of Karatsuba’s algorithm to separate all the \( a_{i_k} b_{j_k} + a_{j_k} b_{i_k} \).
Yet another quadratic method [Lima, Panario, Wang 2010]

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

All the terms have been already computed together in convolutions \( f_k \).

- could be recomputed but at an expensive cost
- could use separation technique to retrieve them [Lima, Panario, Wang 2010]

Main idea

exploit the recursive three terms structure \(< a_0 b_0, a_0 b_1 + a_1 b_0, a_1 b_1 >\) of Karatsuba’s algorithm to separate all the \(a_{i_k} b_{j_k} + a_{j_k} b_{i_k}\)

still needs \(O(n^2)\) operations and \(O(n^{\log 3})\) extra memory !!!

Let \( f(x) = f_0 + f_1 x + \ldots + f_d x^d \in \mathbb{R}[x] \), then

\[
\text{DFT}_n(f) = (F_k)_{k=0..n-1} \quad \text{such that} \quad F_k = \sum_{j=0}^{n-1} f_j e^{-\frac{2i\pi}{n} kj}
\]

\[
\text{DCT}_n(f) = (F_k)_{k=0..n-1} \quad \text{such that} \quad F_k = 2 \sum_{j=0}^{n-1} f_j \cos \left[ \frac{\pi k}{2n} (2j + 1) \right]
\]

\( \text{DCT}_n \) is almost the real part of a \( \text{DFT}_{2n} \) of the even symmetrized input

Main idea

Use same approach as for monomial basis but with DCT instead of DFT.

Use evaluation/interpolation on the points \( \mu_j = \cos\left(\frac{j\pi}{n}\right), j = 0, \ldots, n - 1 \) allows the use of DCT and its inverse.

**Principle**

Let \( a(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k T_k(x) \).

Using Chebyshev polynomials definition we have

\[
T_k(\mu_j) = \cos\left(\frac{k\pi j}{n}\right)
\]

which gives

\[
a(\mu_j) = \sum_{k=0}^{n-1} a_k \cos\left(\frac{k\pi j}{n}\right).
\]

This is basically a DCT\(_n\) on coefficients of \( a(x) \).
Let $a, b \in \mathbb{R}[x]$ given in Chebyshev basis with degree $d < n = 2^k$

The method to compute $c = ab$ is:

- set $\mu_j = \cos\left(\frac{j\pi}{n}\right), j = 0, \ldots, 2n$
- $[a(\mu_0), a(\mu_1), \ldots, a(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on $a$
- $[b(\mu_0), b(\mu_1), \ldots, b(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on $b$
- $ab = a(\mu_0)b(\mu_0) + a(\mu_1)b(\mu_1)x + \ldots + a(\mu_{2n})b(\mu_{2n})x^{2n} \in \mathbb{R}[x]$
- $[c_0, c_1, \ldots, c_{2n}] \in \mathbb{R}^{2n}$ using DCT on $ab$ plus scaling
Let $a, b \in \mathbb{R}[x]$ given in Chebyshev basis with degree $d < n = 2^k$

The method to compute $c = ab$ is:

- set $\mu_j = \cos(j\frac{\pi}{n}), j = 0, \ldots, 2n$
  - $[a(\mu_0), a(\mu_1), \ldots, a(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on $a$
  - $[b(\mu_0), b(\mu_1), \ldots, b(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on $b$
  - $ab = a(\mu_0)b(\mu_0) + a(\mu_1)b(\mu_1)x + \ldots + a(\mu_{2n})b(\mu_{2n})x^{2n} \in \mathbb{R}[x]$
  - $[c_0, c_1, \ldots, c_{2n}] \in \mathbb{R}^{2n}$ using DCT on $ab$ plus scaling

This method requires:

- $3n \log 2n - 2n + 3$ multiplications in $\mathbb{R}$,
- $(9n + 3) \log 2n - 12n + 12$ additions in $\mathbb{R}$. 
Polynomial multiplication in \( \mathbb{R}[x] \) (Chebyshev basis)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>nbr of multiplication</th>
<th>nbr of addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct method</td>
<td>( n^2 + 2n - 1 )</td>
<td>( 1.5n^2 - 2.5n + 1 )</td>
</tr>
<tr>
<td>Lima et al.</td>
<td>( 0.5n^2 + 2.5n - 1 )</td>
<td>( 3n^2 + n^{\log 3} - 6n + 2 )</td>
</tr>
<tr>
<td>DCT-based</td>
<td>( 3n \log 2n - 2n + 3 )</td>
<td>( (9n + 3) \log 2n - 12n + 12 )</td>
</tr>
</tbody>
</table>
Polynomial multiplication in $\mathbb{R}[x]$ (Chebyshev basis)

Polynomial Multiplication in Chebyshev basis (theoretical count)

- Direct method
- Lima et. al method
- DCT-based method

1 FP mul = 1 FP add

nbr of operations vs Polynomials size (equiv. to degree+1)
Polynomial multiplication in $\mathbb{R}[x]$ (Chebyshev basis)

Polynomial Multiplication in Chebyshev basis (theoretical count)

- Direct method
- Lima et. al method
- DCT-based method

1 FP mul = 2 FP add

Polynomials size (equiv. to degree+1)
Outline

1. Polynomial multiplication in monomial basis
2. Polynomial multiplication in Chebyshev basis
3. A more direct reduction from Chebyshev to monomial basis
4. Implementations and experimentations
5. Conclusion
Use same approach as in [Lima, Panario, Wang 2010]

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

Using any polynomial multiplication algorithms (monomial basis) to compute convolutions:

\[
f_k = \sum_{l=0}^{k} a_{k-l} b_l
\]

\[
2c_k = \begin{cases} 
  f_0 + 2 \sum_{l=1}^{d} a_l b_l & \text{for } k = 0, \\
  f_k + \sum_{l=1}^{d-k} (a_l b_{k+l} + a_{k+l} b_l) & \text{for } k = 1, \ldots, d - 1, \\
  f_k & \text{for } k = d, \ldots, 2d.
\end{cases}
\]
Use same approach as in [Lima, Panario, Wang 2010]

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

Using any polynomial multiplication algorithms (monomial basis) to compute convolutions:

\[
f_k = \sum_{l=0}^{k} a_{k-l} b_l \]

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

so we do for the remaining part !!!
Correcting by monomial basis multiplication

We need to compute:

\[
\begin{align*}
\sum_{l=1}^{d} a_l b_l \\
\sum_{l=1}^{d-k} a_l b_{k+l} \\
\sum_{l=1}^{d-k} a_{k+l} b_l
\end{align*}
\]

for \( k = 1, \ldots, d - 1 \)
Correcting by monomial basis multiplication

We need to compute:

\[
\sum_{l=1}^{d} a_l b_l
\]

\[
\sum_{l=1}^{d-k} a_l b_{k+l}
\]

\[
\sum_{l=1}^{d-k} a_{k+l} b_l
\]

for \( k = 1, \ldots, d - 1 \)

which can be deduced from convolutions

\[
g_d = \sum_{l=0}^{d} r_{d-l} b_l
\]

\[
g_{d+k} = \sum_{l=0}^{d-k} r_{d-l} b_{k+l}
\]

\[
g_{d-k} = \sum_{l=0}^{d-k} r_{d-k-l} b_l
\]

with an extra cost of \( O(n) \) operations, where \( a_l = r_{d-l} \)
A direct reduction: PM-Chebyshev

Let \( a(x), b(x) \in \mathbb{R}[x] \) of degree \( d = n - 1 \) given in Chebyshev basis and \( r(x) \) the reverse polynomial of \( a(x) \).

Use any monomial basis algorithms to compute convolutions:

\[
f_k = \sum_{l=0}^{k} a_{k-l} b_l \quad \text{and} \quad g_k = \sum_{l=0}^{k} r_{k-l} b_l
\]
A direct reduction : PM–Chebyshev

Let \( a(x), b(x) \in \mathbb{R}[x] \) of degree \( d = n - 1 \) given in Chebyshev basis and \( r(x) \) the reverse polynomial of \( a(x) \).

Use any monomial basis algorithms to compute convolutions:

\[
\begin{align*}
f_k &= \sum_{l=0}^{k} a_{k-l} b_l \\
g_k &= \sum_{l=0}^{k} r_{k-l} b_l
\end{align*}
\]

Computation of \( c(x) = a(x)b(x) \) in Chebyshev basis is deduced from :

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

at a cost of \( 2M(n) + O(n) \) operations in \( \mathbb{R} \).
Analysis of the complexity

Let \( M(n) \) be the cost of polynomial multiplication in monomial basis with polynomials of degree \( d < n \).

PM–Chebyshev algorithm costs exactly \( 2M(n) + 8n - 10 \) op. in \( \mathbb{R} \).
Analysis of the complexity

Let $M(n)$ be the cost of polynomial multiplication in monomial basis with polynomials of degree $d < n$.

PM–Chebyshev algorithm costs exactly $2M(n) + 8n - 10$ op. in $\mathbb{R}$.

### Optimization trick

Setting $a_0 = b_0 = 0$ juste before computation of coefficients $g_k$ gives

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

reducing the cost to $2M(n) + 4n - 3$ op. in $\mathbb{R}$.
Analysis of the complexity

<table>
<thead>
<tr>
<th>M(n)</th>
<th>nb. of multiplication</th>
<th>nb. of addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schoolbook</td>
<td>$2n^2 + 2n - 1$</td>
<td>$2n^2 - 2n$</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>$2n \log^3 + 2n - 1$</td>
<td>$14n \log^3 - 12n + 2$</td>
</tr>
<tr>
<td>DFT-based(*)</td>
<td>$6n \log 2n - 6n + 11$</td>
<td>$18n \log 2n - 22n + 22$</td>
</tr>
</tbody>
</table>

Special case of DFT-based multiplication

PM-Chebyshev algorithm can be degenerated to reduce the constant term. It involves 2 multiplications with only 3 different operands. One DFT is computed twice.

\[ \frac{1}{6} \text{th of the computation can be saved giving a complexity in this case of} \]

\[ 5n \log_2 n - 3n + 9 \text{ multiplications in } R \]

\[ 15n \log_2 n - 17n + 18 \text{ additions in } R. \]
Special case of DFT-based multiplication

PM–Chebyshev algorithm can be degenerated to reduce the constant term. It involves 2 multiplications with only 3 different operands. One DFT is computed twice.

1/6th of the computation can be saved, giving a complexity in this case of

- $5n \log 2n - 3n + 9$ multiplications in $\mathbb{R}$,
- $15n \log 2n - 17n + 18$ additions in $\mathbb{R}$. 
Special case of DFT-based multiplication

We can trade another DFT for few linear operations. Indeed, we need to compute:

\[ \text{DFT}_{2n}(\bar{a}(x)) \quad \text{and} \quad \text{DFT}_{2n}(\bar{r}(x)) \]

where

\[ \bar{a}(x) = a_0 + a_1 x + \ldots + a_d x^d \quad \text{and} \]
\[ \bar{r}(x) = a_d + a_{d-1} x + \ldots + a_0 x^d = \bar{a}(x^{-1}) x^d. \]

Assuming \( \omega = e^{\frac{-2i\pi}{2n}} \), we know that

\[ \text{DFT}_{2n}(\bar{a}) = [ \bar{a}(\omega^k) ]_{k=0\ldots2n-1}, \quad (1) \]
\[ \text{DFT}_{2n}(\bar{r}) = [ \bar{a}(\omega^{2n-k}) \omega^{kd} ]_{k=0\ldots2n-1}. \quad (2) \]

(1) and (2) are equivalent modulo \( 4n - 2 \) multiplications in \( \mathbb{C} \).
Almost $1/3$th of the computation can be saved giving a complexity in this case of

- $4n \log 2n + 12n + 1$ multiplications,
- $12n \log 2n + 8$ additions.
Almost 1/3th of the computation can be saved
giving a complexity in this case of
- $4n \log 2n + 12n + 1$ multiplications,
- $12n \log 2n + 8$ additions.

We will now refer to our algorithm as PM–Chebyshev(XXX),
where XXX represents underlying monomial basis algorithm.
Polynomial multiplication in Chebyshev basis (theoretical)

Method of [Lima et. al 2010]
DCT-based method
PM-Chebyshev (Karatsuba)
PM-Chebyshev (DFT-based)

1 FP mul = 1 FP add
Polynomial multiplication in Chebyshev basis (theoretical)

Method of [Lima et al. 2010]
DCT-based method
PM-Chebyshev (Karatsuba)
PM-Chebyshev (DFT-based)

1 FP mul = 2 FP add

Polynomials size (equiv. to degree+1)

Speedup

Direct method
Method of [Lima et al. 2010]
DCT-based method
PM-Chebyshev (Karatsuba)
PM-Chebyshev (DFT-based)
Polynomial multiplication in Chebyshev basis (theoretical)

- Direct method
- Method of [Lima et. al 2010]
- DCT-based method
- PM-Chebyshev (Karatsuba)
- PM-Chebyshev (DFT-based)

![Graph showing speedup comparison](image-url)
Outline

1. Polynomial multiplication in monomial basis
2. Polynomial multiplication in Chebyshev basis
3. A more direct reduction from Chebyshev to monomial basis
4. Implementations and experimentations
5. Conclusion
Software Implementation

My goal

- provide efficient code for multiplication in Chebyshev basis
- evaluate performances of existing algorithms

My method

- use C++ for generic, easy, efficient code
- re-use as much as possible existing efficient code
  → especially for DFT/DCT based code [Spiral project, FFTW library]
Implementing PM–Chebyshev algorithm

Easy as simple calls to monomial multiplication

```cpp
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]);
}
Implementation of multiplication in monomial basis

Remark
no standard library available for $\mathbb{R}[x]$

My codes
- naive implementations of Schoolbook and Karatsuba (recursive)
- highly optimized DFT-based method using FFTW library\(^a\)
  \(\leftrightarrow\) use hermitian symmetry of DFT on real inputs

\(^a\)http://www.fftw.org/
## Implementation of multiplication in Chebshev basis

**C++ based code**
- naive implementation of direct method

## Remark
No implementation of [Lima, Panario, Wang 2010] method, since
- needs almost as many operations as direct method
- requires $O(n^{\log 3})$ extra memory
- no explicit algorithm given, sounds quite tricky to implement
Implementation of multiplication in Chebyshev basis

C++ based code

- naive implementation of direct method

Remark

No implementation of [Lima, Panario, Wang 2010] method, since

- needs almost as many operations as direct method
- requires $O(n^{\log 3})$ extra memory
- no explicit algorithm given, sounds quite tricky to implement

My feelings

it would not be efficient in practice !!!
A note on DCT in FFTW

Numerical accuracy

Faster algorithms using pre/post processed read DFT suffer from instability issues. In practice, prefer to use:

- smaller optimized DCT-I codelet
- doubled size DFT

According to this, our PM–Chebyshev(DFT–based) should really be competitive.
A note on DCT in FFTW

Numerical accuracy

Faster algorithms using pre/post processed read DFT suffer from instability issues. In practice, prefer to use:

- smaller optimized DCT-I codelet
- doubled size DFT

According to this, our PM–Chebyshev (DFT–based) should really be competitive. But, is our code numerically correct???
Insight on the numerical accuracy

We experimentally check the relative error of each method.

Relative error on polynomial multiplication

Let $a, b \in \mathbb{F}_l[x]$ given in Chebyshev basis.

Consider $\hat{c} \approx a \cdot b \in \mathbb{F}_l[x]$ and $c = a \cdot b \in \mathbb{R}[x]$, then the relative error is

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

---

1gmplib.org
Insight on the numerical accuracy

We experimentally check the relative error of each methods.

Relative error on polynomial multiplication

Let $a, b \in \mathbb{F}[x]$ given in Chebyshev basis.

Consider $\hat{c} \approx a \cdot b \in \mathbb{F}[x]$ and $c = a \cdot b \in \mathbb{R}[x]$, then the relative error is

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

We use GMP library\(^1\):

- to get exact result as a rational number,
- to almost compute the relative error as a rational number.

\(^1\)gmplib.org
Experimental relative error

Average error on random polynomials with entries lying in \([-50, 50]\).
Experimental relative error

Average error on random polynomials with entries lying in $[0, 50]$.
Lets now see practical performances ...

based on average time estimation
Polynomial multiplication in Chebyshev basis: experimental performance on Intel Xeon 2GHz

![Graph showing polynomial multiplication performance in Chebyshev basis. The graph compares different methods: Direct method, DCT-based method, DCT-based (via FFT) method, PM-Chebyshev (Karatsuba), PM-Chebyshev (DFT-based), PM-Chebyshev (DFT-based accur). The x-axis represents polynomial size (equivalent to degree+1), and the y-axis represents speedup. The graph visually demonstrates the performance differences among the methods.]
Polynomial multiplication in Chebyshev basis: experimental performance on Intel Xeon 2GHz
Polynomial multiplication in Chebyshev basis: experimental performance on Intel Xeon 2GHz

![Graph showing speedup comparison between different methods](image-url)
Outline

1. Polynomial multiplication in monomial basis
2. Polynomial multiplication in Chebyshev basis
3. A more direct reduction from Chebyshev to monomial basis
4. Implementations and experimentations
5. Conclusion
Conclusion

Main contribution

Provide a direct method to multiply polynomials given in Chebyshev basis which reduces to monomial basis multiplication:

- better complexity than using basis conversions,
- easy implementation offering good performances,
- probably as accurate as any other direct methods,
- offer the use of FFT instead of DCT-I.

Would be interesting to compare with optimized Karatsuba's implementation, further investigation on stability issues, discover similarity with finite fields (Dickson Polynomials) and see applications in cryptography [Hasan, Negre 2008].
Conclusion

Main contribution
Provide a direct method to multiply polynomials given in Chebyshev basis which reduces to monomial basis multiplication:

- better complexity than using basis conversions,
- easy implementation offering good performances,
- probably as accurate as any other direct methods,
- offer the use of FFT instead of DCT-I.

Would be interesting to

- compare with optimized Karatsuba’s implementation,
- further investigation on stability issues,
- discover similarity with finite fields (Dickson Polynomials) and see applications in cryptography [Hasan, Negre 2008].
Times of polynomial multiplication in Chebyshev basis (given in $\mu$s) on Intel Xeon 2GHz platform.

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

PM-Cheby stands for PM-Chebyshev algorithm.