On Polynomial Multiplication Complexity in Chebyshev Basis

Pascal Giorgi

INRIA Rocquencourt, Algorithms Project's Seminar
November 29, 2010.

Outline

Outline

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 $f.g \in \mathbb{K}[x]$ 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.

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 $f.g \in \mathbb{K}[x]$ 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 $(1, x, x^2, ...)$

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

All these methods works on polynomials over $R[x]$.

- 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 works on polynomials over $R[x]$.

- 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

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

Discrete Fourier Transform DFT

Let $f(x) = f_0 + f_1 x + ... + f_{n-1} x^{n-1} \in \mathbb{R}[x]$.

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

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

which is equivalent to say

$$\mathtt{DFT}_n(f) = (f(0), f(\omega), ..., f(\omega^{n-1}) \text{ with } \omega = e^{\frac{-2i\pi}{n}}.$$

Fast Fourier Transform FFT

The Fast Fourier Transform (FFT) is a fast method based on a divide and conquer approach [Gauss 19th, Cooley-Tuckey 1965] to compute the $\mathtt{DFT}_n$. It uses the following property:

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

$$\begin{aligned} f(\omega^{2k}) &= r_0(\omega^{2k}), \\ f(\omega^{2k+1}) &= r_1(\omega^{2k+1}) = \bar{r_1}(\omega^{2k}). \end{aligned}$$

This means $\mathtt{DFT}_n(f)$ can be computed using $\mathtt{DFT}_{\frac{n}{2}}(r_0)$ and $\mathtt{DFT}_{\frac{n}{2}}(\bar{r_1})$, yielding a recursive complexity of $H(n) = 2H(n/2) + O(n) = O(n \log n)$.

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

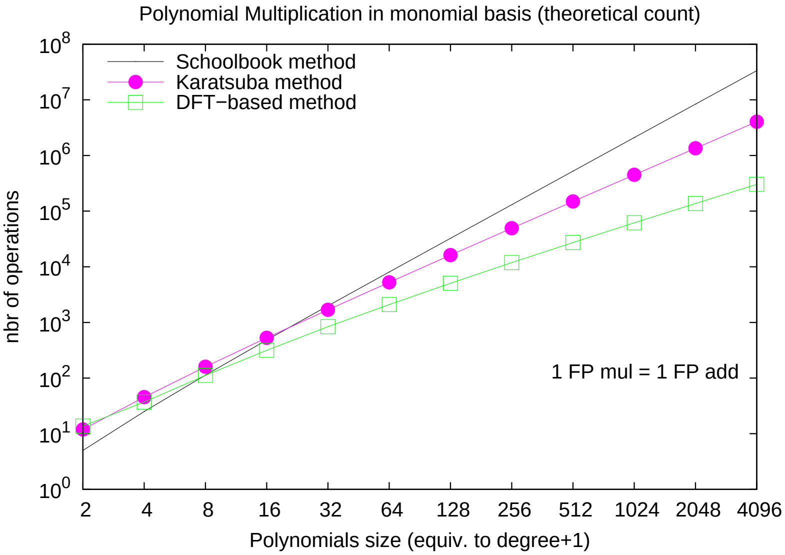
Asymptotic complexity is not reflecting real behaviour of algorithms in practice, constant and lower terms in complexity does really matter !!!

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

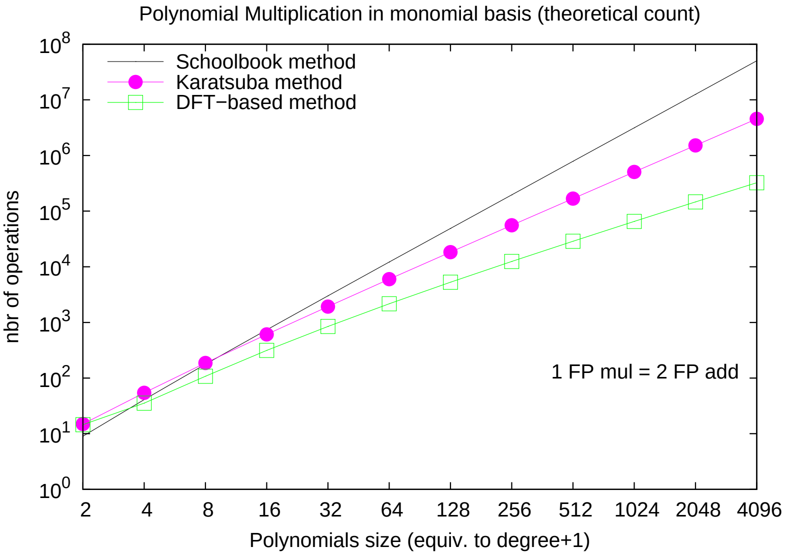
Algorithm	**nbr of multiplication**	**nbr of addition**
Schoolbook	n^2	$(n-1)^2$
Karatsuba	$n^{\log 3}$	$7n^{\log 3} - 7n + 2$
DFT-based$^{(*)}$	$3n \log 2n - n + 6$	$9n \log 2n - 12n + 12$

(*) using real-valued FFT [Sorensen, Jones, Heaideman 1987] with 3/3 complex mult.

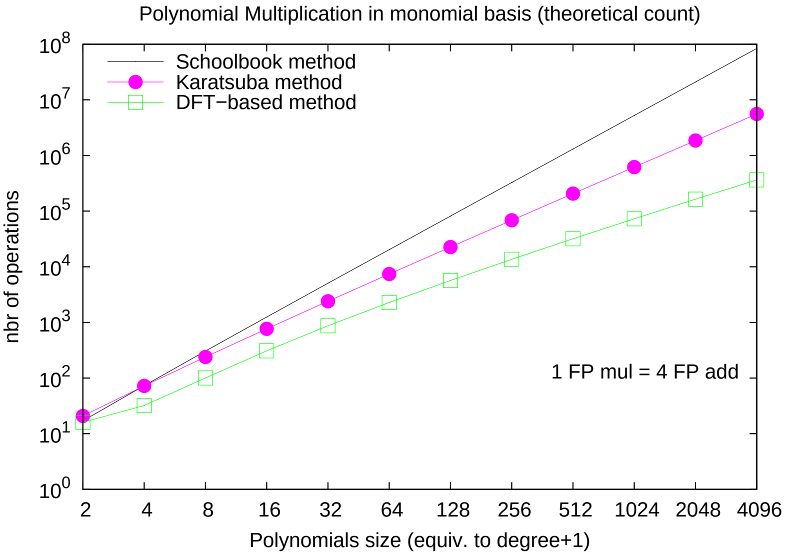
Polynomial multiplication in $\mathbb{R}[x]$



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



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



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

Chebyshev Polynomials: a short definition

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

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

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:

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

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 ?

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

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

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 $a.b \in \mathbb{R}[x]$ 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.

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 $a.b \in \mathbb{R}[x]$ 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 $a.b \in \mathbb{R}[x]$ 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 !!!

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

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:

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

Multiplication in Chebyshev basis: the direct method

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

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:

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

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

- $\dfrac{n^2+5n-2}{2}$ 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.

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

- $\dfrac{n^2 + 5n - 2}{2}$ 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

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

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]

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

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 !!!

Fast method - $\mathtt{DCT}$-based [Baszenski, Tasche 1997]

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

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

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

$\mathtt{DCT}_n$ is almost the real part of a $\mathtt{DFT}_{2n}$ of the even symmetrized input

Main idea

Use same approach as for monomial basis but with $\mathtt{DCT}$ instead of $\mathtt{DFT}$.

Fast method - DCT-based [Baszenski, Tasche 1997]

Use evaluation/interpolation on the points $\mu_j = \cos(\frac{j\pi}{n}), j = 0, \ldots, n-1$ allows the use of $\mathtt{DCT}$ 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

$$T_k(\mu_j) = \cos(\frac{k\pi j}{n})$$

which gives

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

This is basically a $\mathtt{DCT}_n$ on coefficients of $a(x)$.

Fast method - DCT-based [Baszenski, Tasche 1997]

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 $\mu_j = \cos(\frac{j\pi}{n}), j = 0, \ldots, 2n$

- $[a(\mu_0), a(\mu_1), ..., a(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on a
- $[b(\mu_0), b(\mu_1), ..., b(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on b
- $ab = a(\mu_0)b(\mu_0) + a(\mu_1)b(\mu_1)x + ... + a(\mu_{2n})b(\mu_{2n})x^{2n} \in \mathbb{R}[x]$
- $[c_0, c_1, ..., c_{2n}] \in \mathbb{R}^{2n}$ using DCT on ab plus scaling

Fast method - DCT-based [Baszenski, Tasche 1997]

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 $\mu_j = \cos(\frac{j\pi}{n}), j = 0, \ldots, 2n$

- $[a(\mu_0), a(\mu_1), ..., a(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on a
- $[b(\mu_0), b(\mu_1), ..., b(\mu_{2n})] \in \mathbb{R}^{2n+1}$ using DCT on b
- $ab = a(\mu_0)b(\mu_0) + a(\mu_1)b(\mu_1)x + ... + a(\mu_{2n})b(\mu_{2n})x^{2n} \in \mathbb{R}[x]$
- $[c_0, c_1, ..., c_{2n}] \in \mathbb{R}^{2n}$ using DCT on ab plus scaling

This method requires:

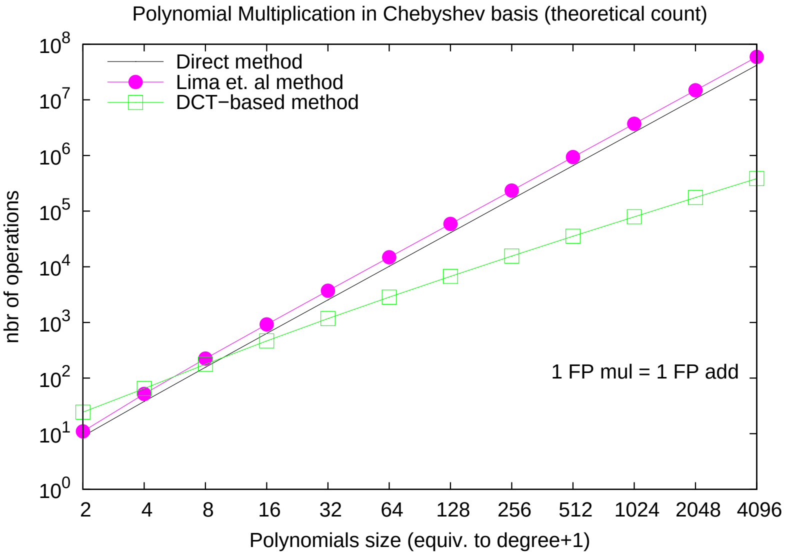
- $3n \log 2n - 2n + 3$ multiplications in $\mathbb{R}$,
- $(9n + 3) \log 2n - 12n + 12$ additions in $\mathbb{R}$.

Polynomial multiplication in $\mathbb{R}[x]$ (Chebyshev basis)

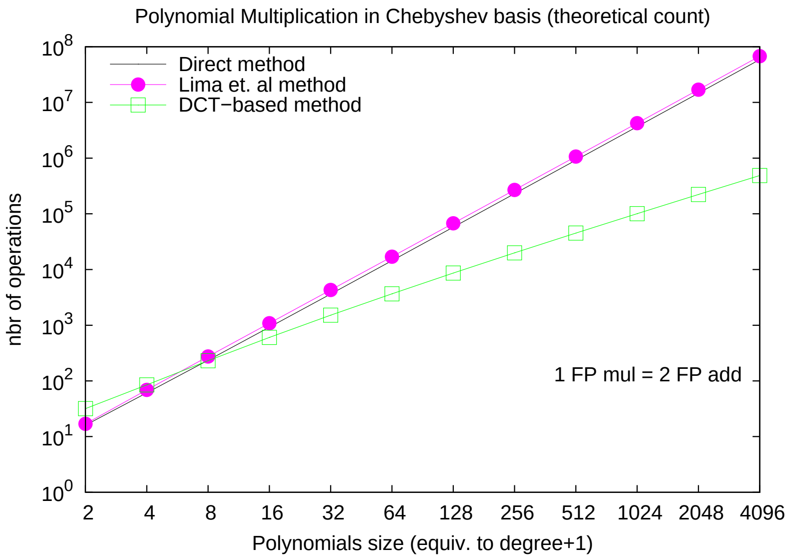
Exact number of operations in $\mathbb{R}$ to multiply two polynomials of $\mathbb{R}[x]$ with degree $d < n = 2^k$ in Chebyshev basis

Algorithm	nbr of multiplication	nbr of addition
Direct method	$n^2 + 2n - 1$	$1.5n^2 - 2.5n + 1$
Lima et al.	$0.5n^2 + 2.5n - 1$	$3n^2 + n^{log 3} - 6n + 2$
`DCT`-based	$3n \log 2n - 2n + 3$	$(9n + 3) \log 2n - 12n + 12$

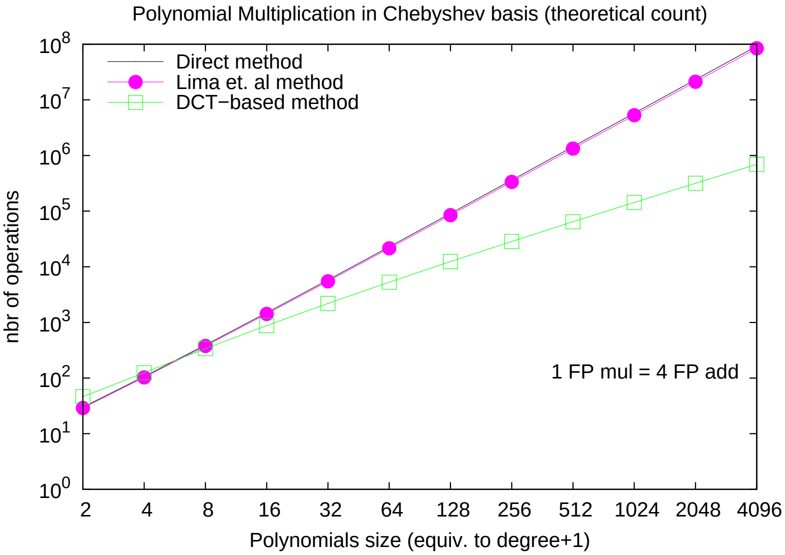
Polynomial multiplication in $\mathbb{R}[x]$ (Chebyshev basis)



Polynomial multiplication in $\mathbb{R}[x]$ (Chebyshev basis)



Polynomial multiplication in $\mathbb{R}[x]$ (Chebyshev basis)



Outline

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:

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

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

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:

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

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

so we do for the remaining part !!!

Correcting by monomial basis multiplication

We need to compute:

- $\displaystyle\sum_{l=1}^{d} a_l b_l$
- $\displaystyle\sum_{l=1}^{d-k} a_l b_{k+l}$
- $\displaystyle\sum_{l=1}^{d-k} a_{k+l} b_l$

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

- $g_d \quad = \sum_{l=0}^{d} r_{d-l} b_l$
- $g_{d+k} = \sum_{l=0}^{d-k} r_{d-l} b_{k+l}$
- $g_{d-k} = \sum_{l=0}^{d-k} r_{d-k-l} b_l$

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:

$$f_k = \sum_{l=0}^{k} a_{k-l} b_l \text{ and } g_k = \sum_{l=0}^{k} r_{k-l} b_l$$

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:

$$f_k = \sum_{l=0}^{k} a_{k-l} b_l \text{ and } g_k = \sum_{l=0}^{k} r_{k-l} b_l$$

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

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

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

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

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

Analysis of the complexity

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

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` *[Sorensen, Jones, Heideman 1987] with 3/3 complex mult.*

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.

$\hookrightarrow$ one DFT is computed twice

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.

$\hookrightarrow$ 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 :

$$\mathtt{DFT}_{2n}(\bar{a}(x)) \text{ and } \mathtt{DFT}_{2n}(\bar{r}(x)) \text{ where}$$

$\bar{a}(x) = a_0 + a_1 x + \ldots + a_d x^d$ 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

$$\mathtt{DFT}_{2n}(\bar{a}) = [\ \bar{a}(\omega^k)\ ]_{k=0\ldots 2n-1}, \tag{1}$$

$$\mathtt{DFT}_{2n}(\bar{r}) = [\ \bar{a}(\omega^{2n-k})\ \omega^{kd}\ ]_{k=0\ldots 2n-1}. \tag{2}$$

(1) and (2) are equivalent modulo $4n-2$ multiplications in $\mathbb{C}$.

Special case of DFT-based multiplication

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.

Special case of DFT-based multiplication

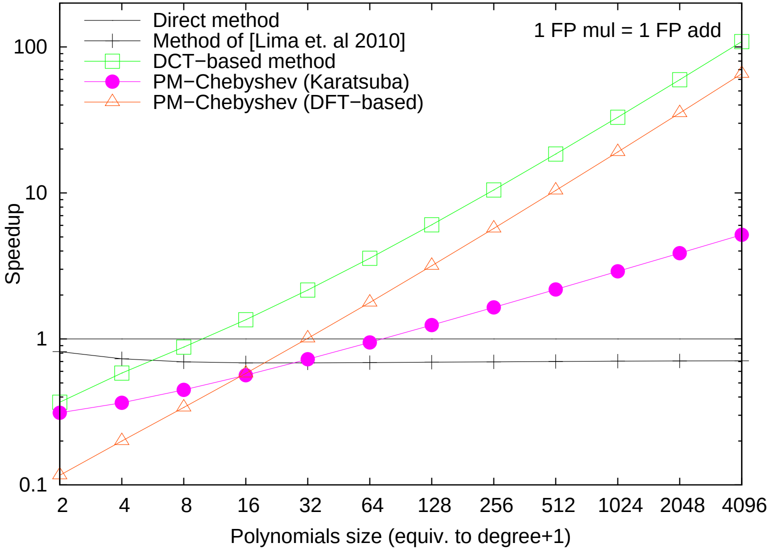
Almost $1/3th$ of the computation can be saved

giving a complexity in this case of

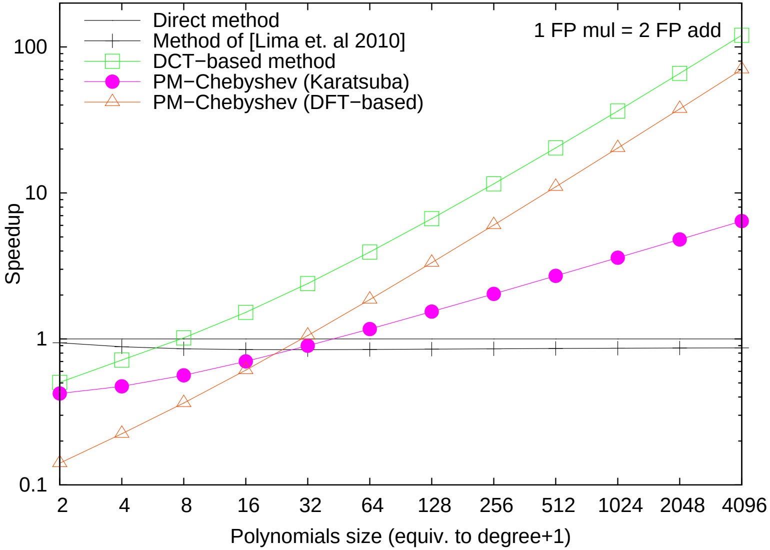
- $4n \log 2n + 12n + 1$ multiplications,
- $12n \log 2n + 8$ additions.

We will now refer to our algorithm as `PM-Chebyshev(XXX)`, where `XXX` represents underlying monomial basis algorithm.

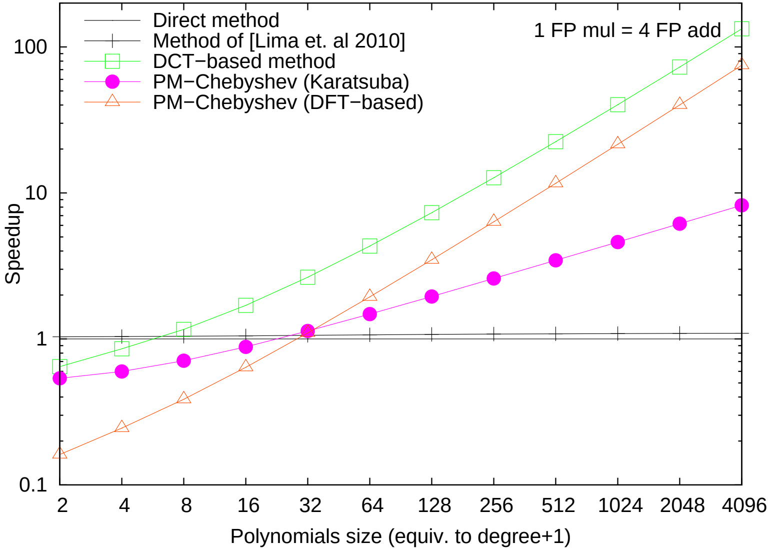
Polynomial multiplication in Chebyshev basis (theoretical)



Polynomial multiplication in Chebyshev basis (theoretical)



Polynomial multiplication in Chebyshev basis (theoretical)



Outline

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
  $\hookrightarrow$ especially for `DFT`/`DCT` based code [Spiral project, FFTW library]

Implementing PM-Chebyshev algorithm

Easy as simple calls to monomial 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]);
}
```

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]
  $\hookrightarrow$ 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
- optimized DCT-based method of [Tasche, Baszenski 1997] using FFTW

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 Chebshev basis

C++ based code

- naive implementation of direct method
- optimized DCT-based method of [Tasche, Baszenski 1997] using FFTW

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 methods.

Relative error on polynomial multiplication

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

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

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

[1] `gmplib.org`

Insight on the numerical accuracy

We experimentally check the relative error of each methods.

Relative error on polynomial multiplication

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

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

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

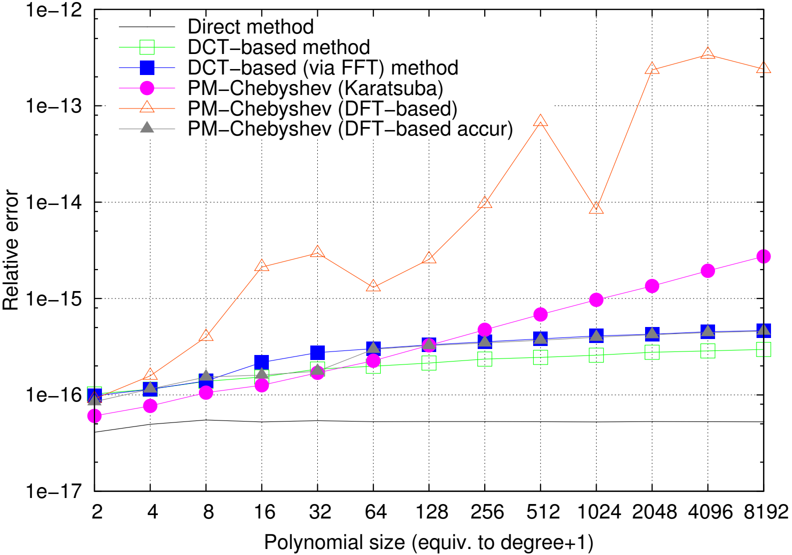
We use GMP library[1] :

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

[1] `gmplib.org`

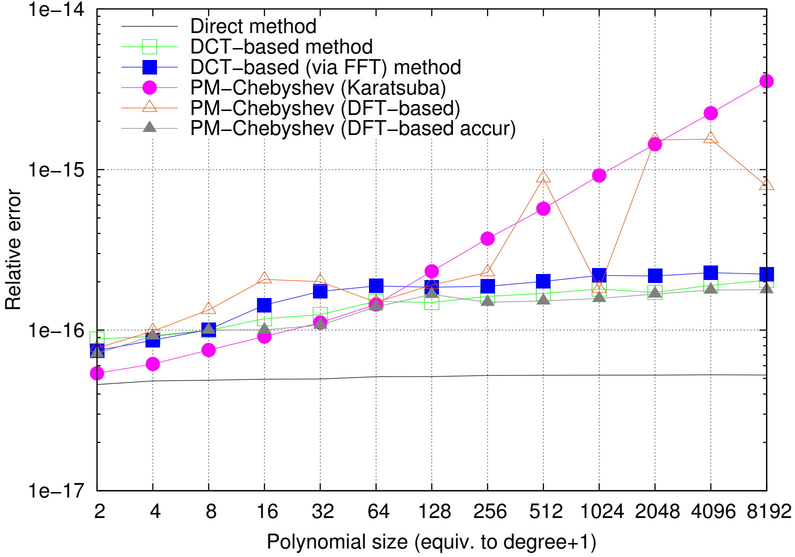
Experimental relative error

Average error on random polynomials with entries lying in $[-50, 50]$.



Experimental relative error

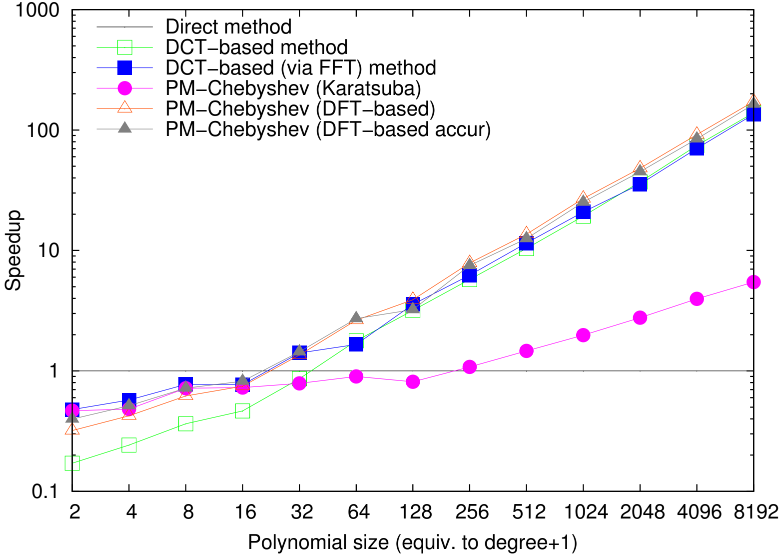
Average error on random polynomials with entries lying in $[0, 50]$.



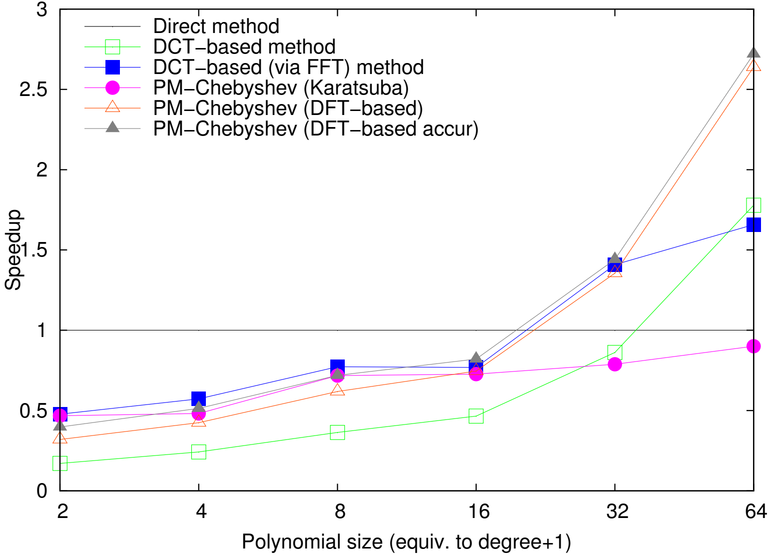
Lets now see practical performances ...

based on average time estimation

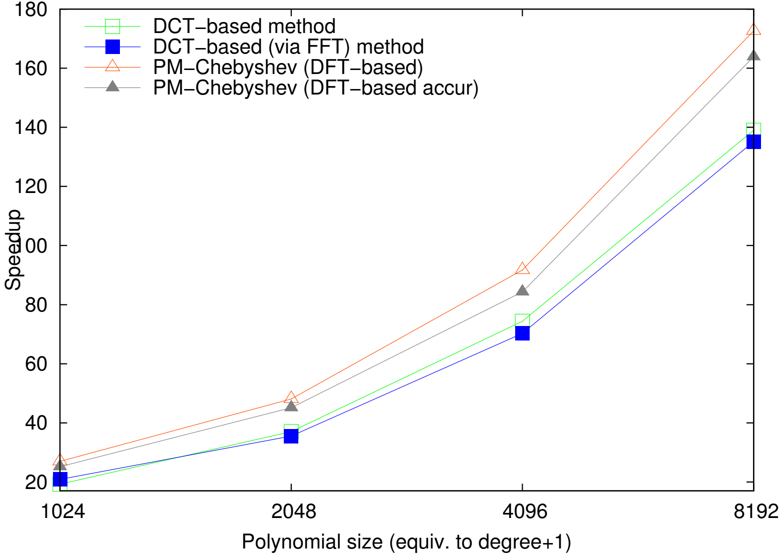
Polynomial multiplication in Chebyshev basis: experimental performance on Intel Xeon 2GHz



Polynomial multiplication in Chebyshev basis: experimental performance on Intel Xeon 2GHz



Polynomial multiplication in Chebyshev basis: experimental performance on Intel Xeon 2GHz



Outline

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`.

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 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.