**Implementation of a Fast Algorithm for Multipoint Evaluation of Univariate Polynomials**

*WA Gunarathna[1] and HM Nasir[2]*

*[1]Department of IT & Mathematics, Faculty of Engineering, General Sir John Kotelawala Defence University,SriLanka*

*[2]Department of Mathematics, Faculty of Science, University of Peradeniya, Sri Lanka*

**Abstract**

Let $f(z)=f_0+f_1z+\cdots+f_{n-1}z^{n-1}$ be a polynomial of degree $n-1$ with complex coefficients. Let $S=\{z_0,z_1,\cdots,z_{n-1}\}$ be a set of complex points. The straightforward method of evaluation of $f(z_j)$ for all points in requires arithmetic operations and also the method of synthetic division based on the Remainder theorem and Horner's method each requires arithmetic operations for the same evaluation. In this paper, we present a fast algorithm for evaluating simultaneously $f(z_j)$ for all $j=0,1,\cdots,n-1$ in arithmetic operations, in total. To examine the efficiency of the algorithm, both the standard straightforward and fast algorithms are implemented in MATLAB software to obtain numerical results for the evaluation of Chebyshev polynomials of the first kind at the Chebyshev nodes. The CPU times taken by both algorithms are compared. The numerical experiment demonstrates that the fast algorithm is much faster than the straightforward algorithm when $n\geq 128$.

**Key words**: Fast Fourier Transform (FFT), Circulant Matrix-vector multiplication, Toeplitz Matrix –Vector Multiplication, Vandermonde matrix Fast Polynomial Multiplication.

**1. Introduction:** The task of the evaluation of a polynomial at several arguments, referred in the literature as the multipoint polynomial evaluation problem (MPE) plays a significant role in a vast area of computational problems. These types of computations have been widely used in Engineering, Physics, Medicine, Weather forecasting, and so on. We denote a generic univariate polynomial of degree $n-1$ by $f(z)=f_0+f_1z+f_2z^2+\ldots+f_{n-1}z^{n-1}$. In the present paper our work is restricted to evaluate the polynomial $f(z)$ at $n$ arguments. To put it in another way, we have to compute the collection:

$$\left\{\sum_{i=0}^{n-1} f_i {z_k}^i\right\}_{k=0,\ldots n-1},$$

is equivalent to the computation of the following matrix-vector product:

$$\begin{pmatrix} f(z_0) \\ f(z_1) \\ \cdots \\ f(z_{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & z_0 & \cdots & {z_0}^{n-1} \\ 1 & z_1 & \cdots & {z_1}^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & z_{n-1} & \cdots & {z_{n-1}}^{n-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \cdots \\ f_{n-1} \end{pmatrix}, \quad \text{where}$$

the matrix $V = \begin{pmatrix} 1 & z_0 & \cdots & z_0^{n-1} \\ 1 & z_1 & \cdots & z_1^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & z_{n-1} & \cdots & z_{n-1}^{n-1} \end{pmatrix}^T$, called a Vandermonde matrix .

The naïve approach to solve this problem (MPE), the sequent computation of $f(z)$ at all augments $z_0, z_1, \ldots, z_{n-1}$, may require a total cost of $O(n^3)$ arithmetic operations. By Horner's rule or the method of synthetic division, any univariate polynomial $f$ of degree $n-1$ can be evaluated at a given evaluation argument in $O(n)$ arithmetic operations by expressing the polynomial in the form of:

$$f(z) = (\ldots(f_{n-1}z + f_{n-2})z + f_{n-3})z + f_0 \text{ (See [2]).}$$

In order to sequentially evaluate the polynomial $f$ at $n$ arguments, we may require $O(n^2)$ arithmetic operations, in total ([2)]). However, these both approaches are prohibitively expensive in terms of the number of arithmetic operations and hence computing time for problems of modest size.

Utilization of modern-day computers to solve problems including numerous applications has also demanded efficient algorithms to compute problems in the domain of multipoint polynomial evolution. The Discrete Fourier Transform (DFT) of a set of $n$ input complex numbers $\{f_j\}_{j=0,\ldots,n-1}^{n-1}$ yields the $n$ output collection:

$$\left\{ \sum_{j=0}^{n-1} f_j \, e^{-i2\pi jk/n} \right\}_{k=0,\ldots n-1}$$

in $O(n^2)$ arithmetic operations at most, where $i = \sqrt{-1}$ ( See [3],[6]).

This task which may be considered as a special case of the MPE problem can be performed more efficiently using the well-known fast Fourier transform (FFT) in $O(n \log_2 n)$ arithmetic operations at most in the case where $n$ is a power of 2. The materials found from the work of A Borodin and M Munro in 1975 ([2]) pointed out an exact algorithm of $O(n \log_2^2 n)$ for the computation of the MPE of size $n$. These authors used modular technique to develop this algorithm. J.R.Driscoll, D.M.Healy Jr., and D. Rockmore ([4]) formulated a sophisticated algorithm of the same polynomial computational complexity to compute an $n-1$ degree polynomial over the complex field at $n$ complex arguments in $8\, n \log_2^2 n + 3n \log_2 n$ optimal number of arithmetic operations. The basic idea of this algorithm is to factorize the matrix transpose $V^T$, of the Vandermonde matrix $V$ into product of sparse matrices having each Toeplitz blocks along its main diagonal so that the technique of Circulant matrix vector product and the FFT can be applied to speed up the computation.

The main objective of this paper is to examine the efficiency of the algorithm developed in ([4]). Our motivation behind studying this algorithm is to extend its idea to develop fast algorithms to

compute more efficiently discrete orthogonal polynomial transforms. For simplicity, we assume that $n$ is a power of 2 and we write $\log n$ instead of $\log_2 n$.

The present paper has the following structure: In section 2, we describe standard structured matrices and fast matrix -vector product algorithms for those matrices with computational complexities. Section 3 provides the fast algorithm of the present paper, providing details about its computational costs. Numerical results are presented in Section 4. Section 5 draws conclusions and discusses possible extensions.

**2. Fast Matrix –Vector Product:** The product of a matrix and a vector results in many problems in Engineering and Applied mathematics. For a matrix $A$ of order $n$ ,its matrix product $Ax$ with an arbitrary input vector $x$ requires $O(n^2)$ operation by the straight forward computation. For modest types of matrices, the computational cost of the matrix –vector product becomes prohibitive and thus yields computational inconvenience for practical purposes. We devote this section to present some known structured matrix –vector products, namely, the Fourier matrix-vector product, the Circulant matrix –vector product, and Toeplitz matrix-vector product which can be used to establish the fast algorithm for the MPE ([10],[1]).

**Definition 2.1**: An Fourier matrix, denoted by , is defined to be the square matrix given by

$$F_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega, & \cdots & \omega^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

where $\omega = e^{-i2\pi/n}$ is a primitive $n^{\text{th}}$ root of the unity.

It can be more easily seen that the product of the Fourier matrix with any arbitrary column vector of size yields the well-known discrete Fourier transform (DFT) ,which can be performed more efficiently using the fast Fourier transform (FFT) ([3],[6]). We summarize the materials found from [3] in the following theorem: It is worthwhile to notice that the Fourier matrix is a unitary matrix , that is $F_n F_n^* = I_n$ , where $F_n^*$ is the conjugate matrix of $F_n$, which is also a unitary matrix.

**Theorem 2.2.** The FFT and IFFT can each be done in $O(n \log n)$ arithmetic operations. A proof of this can be found in [ 3].

**Definition 2.3.** An square matrix has the following form is called a circulant matrix.

$$C_n = C(c_0, c_1, \ldots, c_{n-1}) = \begin{pmatrix} c_0 & c_1 & \cdots & c_{n-1} \\ c_{n-1} & c_0 & \cdots & c_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ c_1 & c_2 & \cdots & c_0 \end{pmatrix}$$

The circulant matrix processes constant element along the diagonals from top left to bottom right and in fact, it can be easily seen that the circulant matrix can be spanned by the entities in the first column.

**Theorem 2.4.** The circulant matrix can be diagonalzed by the Fourier matrix such that $C_n = F_n D_n F_n^{-1}$ .See the proof the theorem in [10].

**2.5 Ci rculant matrix -vector product.** Let $\underline{x} = (x_0, x_1, \cdots, x_{n-1})^T$ be a v ector of size .The standard s traightforward calculation of the matrix vector product $C\underline{x}$ requires arithmetic operations. The following FFT based technique needs at most arithmetic operations for the same.

$$\underline{y} = C\underline{x} = \left(F_n^{-1} D_{\ n}\right)\underline{x} = \left(\overline{F_n} D_{\ n}\right)\underline{x} = \overline{F_n} D\left(F_n \underline{y}\right).$$

Note that the Fourier matrix is a unitary matrix. To put it another way, where is the identity matrix of order and is the conjugate transpose of.

Here, $\overline{F}_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \overline{\omega} & \cdots & \overline{\omega}^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \overline{\omega}^{n-1} & \cdots & \overline{\omega}^{(n-1)(n-1)} \end{pmatrix}$

It should be noted that the circulant matrix vector product $C\underline{x}$ can be written as the cyclic convolution of the sequences, and .That is to show,

$$c * x_j = \sum_{m=0}^{n-1} x_m c_{j-m}$$

Now by the Convolution theorem,

$$DFT\left(c * x_j\right) = DFT\left(c_0, c_1, \cdots, c_{n-1}\right) DFT\left(x_0, x_1, \cdots, x_{n-1}\right) \text{ and this implies that}$$

$$\left(c * x_j\right) = IDFT\left[DFT\left(c_0, c_1, \cdots, c_{n-1}\right) DFT\left(x_0, x_1, \cdots, x_{n-1}\right)\right]$$

This concludes that the circulant matrix vector multiplication can be performed using only three FFTs (one IFFT and two FFTs). Since each FFT needs arithmetic operations, the circulant matrix vector multiplication can be done in arithmetic operations.

---

**Algorithm 2.6:** Circulant matrix vector multiplication

---

**Input:** $C = cir\left(c_0, c_1, \cdots, c_{n-1}\right), y = \left(y_0, y_1, \cdots, y_{n-1}\right)^T$ .

**Output:** $\underline{z} = C\underline{y}$

**Stages:** 1. Compute the FFT of the vector $\left(y_0, y_1, \cdots, y_{n-1}\right), Y_k = FFT\left(y_0, y_1, \cdots, y_{n-1}\right)$ .

2. Compute the FFT of the vector $C_k = FFT\left(c_0, c_1, \cdots, c_{n-1}\right)$.

3. for $k = 0$ to $n-1$

do $Z_k = C_k Y_k$ ; point wise multiplication

enddo

4. $z = IFFT(Z)$

---

**Definition 2.7.** A Toeplitz matrix of order is defined to be the following matrix:

$$T = T(t_{-n+1}, \cdots, t_0, \cdots t_{n-1}) := \begin{pmatrix} t_0 & t_1 & \cdots & t_{n-1} \\ t_{-1} & t_0 & \cdots & t_{n-2} \\ \cdots & \cdots & \cdots & \cdots \\ t_{-n+1} & t_{-n+2} & \cdots & t_0 \end{pmatrix}$$

, where t_(-n+1),...,t_0,...t_(n-1) are complex numbers.

A Toeplitz matrix can be spanned by it first column and its first row. Besides that, the entities of the Toeplitz matrix are constant along the sub diagonals parallel to the main diagonal.

**Toeplitz matrix vector multiplication.** The standard straightforward calculation of the matrix vector product $T\underline{x}$ requires arithmetic operations. The following theorem shows an FFT based technique that may compute $T\underline{x}$ in .

**Theorem 2.8.The** product of a Toeplitz matrix of order and a column vector of size can be performed in arithmetic operations ( [10]).

**Proof :**Let be a Toeplitz matrix of order and $\underline{x}$ be a column vector of size .

Define the circulant matrix of order 2n as follows:

$$C_{2n} = \begin{pmatrix} T_n & S_n \\ S_n & T_n \end{pmatrix},$$

where is a square matrix of order given by

$$S_n := \begin{pmatrix} 0 & t_{-n+1} & \cdots & t_{-1} \\ t_{n-1} & 0 & \cdots & t_{-2} \\ \cdots & \cdots & \cdots & \cdots \\ t_1 & t_2 & \cdots & 0 \end{pmatrix}, \quad T = \begin{pmatrix} t_0 & t_1 & \cdots & t_{n-1} \\ t_{-1} & t_0 & \cdots & t_{n-2} \\ \cdots & \cdots & \cdots & \cdots \\ t_{-n+1} & t_{-n+2} & \cdots & t_0 \end{pmatrix}.$$

Also let $\underline{y} = \begin{pmatrix} \underline{x} \\ 0 \end{pmatrix}$ be a column vector of size , where 0 denotes the zero matrix.

Now we get

$$C_{2n}\underline{y}=\begin{pmatrix}T_n & S_n\\ S_n & T_n\end{pmatrix}\begin{pmatrix}\underline{x}\\ 0\end{pmatrix}=\begin{pmatrix}T_n\underline{x}\\ S_n\underline{x}\end{pmatrix}.$$

Now this concludes that the toeplitz matrix -vector multiplication can be done in arithmetic operations.

It should be noticed only three FFTs of order each and one point-wise multiplications of sequence of length are needed to perform the Toeplitz matrix-vector multiplication and that the total number of arithmetic operations is equal to:

$$3\left(\frac{3(2n)}{2}\log 2n\right)+2n=9n\log 2n+2n.$$

Further this can be reduced to $9n\log n+1$ $n$ .That is, the computational complexity is $O(n\log n)$ .

**Algorithm 2.9:** Toeplitz matrix vector product

**Input :** $T=top(t_{-n+1},\cdots,t_0,\cdots,t_{n-1})$ ,
$y=(y_0,y_1,\cdots,y_{n-1})^T$ **Output:** $z=\mathbb{T}$

**Stages:** 1. Compute the FFT of $u=(0,0,\cdots 0,y_0,y_1,\cdots y_{n-1});\ Y_k=FFT(u)$

2. Compute the FFT of $v=(t_0,t_{-1},\cdots,t_{-n+1},0,t_{n-1},\cdots,t_1);X_k=FFT(v)$

3. for $k=0$ to $2n-1$

do $Z_k=X_kY_k$

end for

4. $z=IFFT(Z)$

**Definition.2.10.** A vandermonde matrix of order is defined to be [ ]

$$V=V(z_0,z_1,\cdots,z_{n-1})=\begin{pmatrix}1 & 1 & \cdots & 1\\ z_0 & z_1 & \cdots & z_{n-1}\\ \cdots & \cdots & \cdots & \cdots\\ z_0^{n-1} & z_1^{n-1} & \cdots & z_{n-1}^{n-1}\end{pmatrix}$$

, where $z_0,z_1,\cdots,z_{n-1}$ are complex numbers.

3. **Multipoint Polynomial Evaluation**: Let $f(x)=f_0+f_1x+f_2x^2+\ldots+f_{n-1}x^{n-1}$ be a generic univariate polynomial of degree $n-1$ defined in [x]. Let $\mathbb{Z}=\{z_0,z_1,\cdots,z_{n-1}\}$ be set of $n$ complex numbers. Here we restrict our task to compute $f$ at all the arguments in $\mathbb{Z}$. That is, we wish to compute the collection:

$$\left\{\sum_{i=0}^{n-1} f_i x_k^{\,i}\right\}_{k=0,\ldots n-1}.$$

**Theorem 3.1.** The task of evaluating the polynomial $f$ at all the arguments in $\mathbf{Z}$ can be performed in $O(n\log^2 n)$ arithmetic operations.

**Proof:** We follow the proof in [4].

Now we see that:

$$\begin{aligned}
f(z_0) &= f_0 + f_1 z_0 + \cdots + f_{n-1} z_0^{\,n-1} \\
f(z_1) &= f_0 + f_1 z_1 + \cdots + f_{n-1} z_1^{\,n-1} \\
&\cdots \qquad\qquad \cdots \qquad\qquad \cdots \\
f(z_{n-1}) &= f_0 + f_1 z_{n-1} + \cdots + f_{n-1} z_{n-1}^{\,n-1}
\end{aligned}$$

This further can be written in the following matrix equation.

$$\begin{pmatrix} f(x_0) \\ f(x_1) \\ \cdots \\ f(x_{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & z_0 & \cdots & z_0^{\,n-1} \\ 1 & z_1 & \cdots & z_1^{\,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & z_{n-1} & \cdots & z_{n-1}^{\,n-1} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \cdots \\ f_{n-1} \end{pmatrix},$$

$$\text{Let} \quad V = \begin{pmatrix} 1 & z_0 & \cdots & z_0^{\,n-1} \\ 1 & z_1 & \cdots & z_1^{\,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & z_{n-1} & \cdots & z_{n-1}^{\,n-1} \end{pmatrix}^T = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ z_0 & z_1 & \cdots & z_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ z_0^{\,n-1} & z_1^{\,n-1} & \cdots & z_{n-1}^{\,(n-1)(n-1)} \end{pmatrix}.$$

Let $f = (f_0, f_1, \cdots, f_{n-1})^T$.

Now we see that the matrix $V$ is the so-defined Vandermonde matrix and that the task of solving the MPE problem is equivalent to the task of computing the matrix-vector product $V^T f$. Let $M(z) = \sum_{i=0}^{n-1} f_i z^i$. We shall calculate $M(z)$ at each $z_i$ for $i = 0,1,\cdots,n-1$.

Define: $m_{(i,j)}(z)=\left(z-z_{i(n/2^j)}\right)\cdots\left(z-z_{(i+1)(n/2^j)-1}\right)$ for $i=0,1,\cdots,\log n.$ and for $j=1,2,\cdots\log n$, for example, if $j=1,i=0,1$ , then $m(0,1)(z)=(z-z_0)\cdots(z-z_{n/2-1})$ and $m_{(1,1)}(z)=(z-z_{n/2})\cdots(z-z_{n-1})$.

Divide $M(z)$ by $m_{(0,1)}(z)$ and by $m_{(1,1)}(z)$ seperately. Then, by the division algorithm, we get:

$M(z)=q_{(0,1)}(z)m_{(0,1)}(z)+r_{(0,1)}(z),\ M(z)=q_{(1,1)}(z)m_{(1,1)}+r_{(1,1)}(z).$

where $r_{(0,1)}(z), q_{(0,1)}(z)$ are the remainder and the quotient when is divided by ,respectively and so are $r_{(1,1)}(z), q_{(0,1)}(z)$ when $M(z)$ is divided by $m_{(0,1)}(z)$.

Next divide $r_{(0,1)}(z)$ by $m_{(0,2)}(z)$ and by $m_{(1,2)}(z)$, separately. Then we have:

$$r_{(0,1)}(z)=q_{(0,2)}m_{(0,2)}(z)+r_{(0,2)}(z) \text{ and } r_{(0,1)}(z)=q(1,2)(z)m_{(1,2)}(z)+r_{(1,2)}(z).$$

Also, divide $r_{(1,1)}(z)$ by $m_{(2,2)}(z)$ and by $m_{(3,2)}(z)$ ,separately. Then, we have:

$$r_{(1,1)}(z)=q_{(2,2)}m_{(2,2)}(z)+r_{(1,2)}(z) \text{ and } r_{(1,1)}(z)=q_{(3,2)}m_{(3,2)}(z)+r_{(3,2)}(z)$$

We continue this procedure until we get $r_{(0,k)}(z), r_{(1,k)}(z),\cdots, r_{(n-1,k)}(z)$. Also, it can be seen that:

$$M(z)=q_{(l,1)}m_{(l,1)}(z)+q_{(l,2)}(z)m_{(l,2)}(z)+\cdots+q_{(l,k-1)}(z)m_{(l,k-1)}(z)+q_{((l,k))}(z)m_{(l,k)}(z)+r_{(l,k)}(z).$$

It should be noted that $r_{(l,k)}(z)=r_{(l,k)}$ is constant (real or complex) and that:

$$\begin{pmatrix} M(z_0) \\ M(z_1) \\ \vdots \\ M(z_{n-1}) \end{pmatrix}=\begin{pmatrix} r_{(0,k)} \\ r_{(1,k)} \\ \vdots \\ r_{(n-1,k)} \end{pmatrix}.$$

This mechanism can be interpreted in the following diagram :

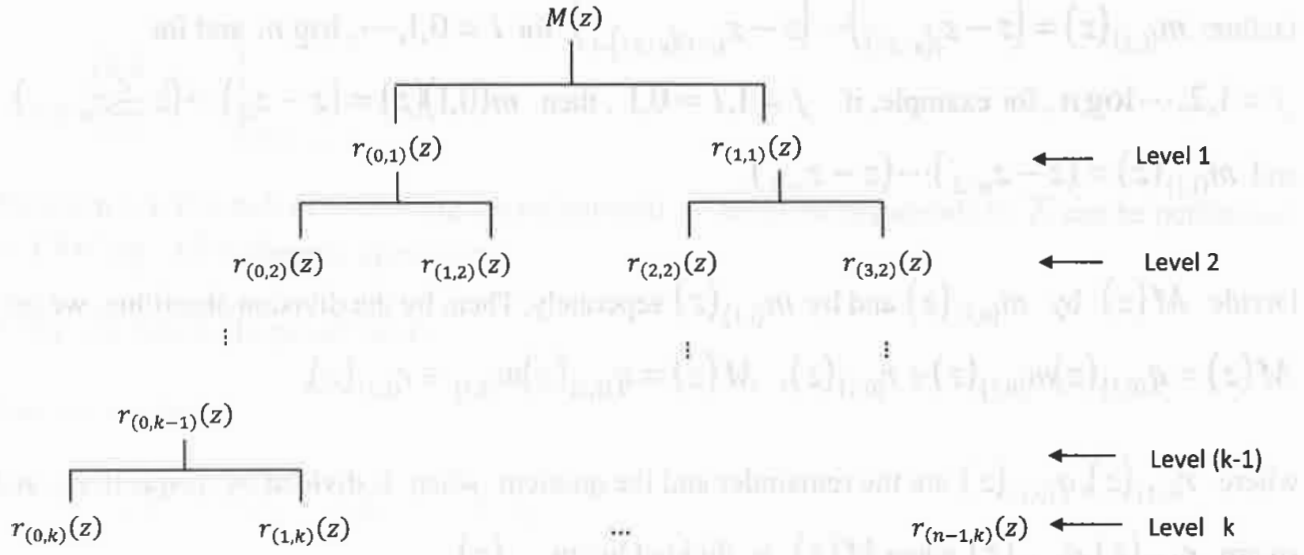


**Figure 1**.Diagram for evaluating a polynomial $M(z)$ at $n = 2^k$ points.

In this method, we find each remainder at all the levels instead of calculating $M(z)$ at each $z_0, z_1, \cdots, z_{n-1}$ and the remainders at the bottom most level in the diagram (Level k) give the corresponding values of $M(z)$. Therefore, it is worthwhile to have an efficient method to calculate the remainder $r(z)$ and the quotient $q(z)$ in the division algorithm

$$p(z) = q(z)m(z) + r(z),$$

where takes a form of: $m_{(i,j)}(z) = \left(z - z_{i(n/2^j)}\right) \cdots \left(z - z_{(i+1)(n/2^j)-1}\right)$.

Suppose that the degree of $p(z)$ is $u-1$, where $u$ is a power of 2.

Then, the degree of is and the degree of is.

Now, we have

$$r(z) = p(z) - q(z)m(z) \qquad (*)$$

$$\text{Let: } \begin{cases} p(z) = p_0 + p_1 z + p_2 z^2 + \cdots + p_{u-1} z^{u-1} \\ q(z) = q_0 + q_1 z + q_2 z^2 + \cdots + q_{\frac{u}{2}-1} z^{u/2-1} \\ r(z) = r + r_1 z + r_2 z^2 + \cdots + r_{\frac{u}{2}-1} z^{u/2-1} \\ m(z) = m_0 + m_1 z + m_2 z^2 + \cdots + m_{\frac{u}{2}} z^{u/2} \end{cases}$$

Then we get from (*) the following:

$$\begin{pmatrix} r_0 \\ \vdots \\ r_{\frac{u}{2}-2} \\ r_{\frac{u}{2}-1} \end{pmatrix} = \begin{pmatrix} p_{\frac{u}{2}-1} - m_0 q_{\frac{u}{2}-1} - m_1 q_{\frac{u}{2}-2} - \cdots - m_{\frac{u}{2}-2} q_1 - m_{\frac{u}{2}-2} q_0 p_0 - m_0 q_0 \\ \vdots \\ p_1 - m_0 q_1 - m_1 q_0 \\ q_{0_0} p_0 - m_0 q_0 \end{pmatrix}$$

$$\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{\frac{u}{2}-1} \end{pmatrix} - \begin{pmatrix} m_0 & 0 & \cdots & 0 \\ m_1 & & & \\ \vdots & & \ddots & \vdots \\ m_{\frac{u}{2}-1} & \cdots & m_1 & m_0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ \vdots \\ q_{\frac{u}{2}-1} \end{pmatrix} \qquad (1)$$

Also, we get from (*)

$$\begin{pmatrix} p_{\frac{u}{2}} - m_1 q_{\frac{u}{2}-1} - \cdots - m_{\frac{u}{2}} q_0 \\ \vdots \\ p_{u-1} - m_{\frac{u}{2}} q_{\frac{u}{2}-1} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \qquad (2)$$

And from this we have:

$$\begin{pmatrix} p_{\frac{u}{2}} \\ \vdots \\ p_{u-1} \end{pmatrix} = \begin{pmatrix} m_{\frac{u}{2}} & \cdots & m_1 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & m_{\frac{u}{2}} \end{pmatrix} \begin{pmatrix} q_0 \\ \vdots \\ q_{\frac{u}{2}-1} \end{pmatrix} \qquad (3)$$

Let $A = \begin{pmatrix} m_0 & 0 & \cdots & 0 \\ m_1 & & & \\ \vdots & \ddots & \ddots & \vdots \\ m_{\frac{u}{2}-1} & \cdots & m_1 & m_0 \end{pmatrix}$ and $B = \begin{pmatrix} m_{\frac{u}{2}} & m_{\frac{u}{2}-1} & \cdots & m_1 \\ \vdots & \ddots & & \vdots \\ & & & \\ 0 & & \cdots & m_{\frac{u}{2}} \end{pmatrix}^{-1}$

Here $A$ is an upper triangular Toeplitz matrix and $B$ is a non-singular lower triangular Toeplitz matrix (since $m_{m/2} = 1$).By combining and , we get,

$$\begin{pmatrix} r_0 \\ \vdots \\ r_{\frac{u}{2}-2} \\ r_{\frac{u}{2}-1} \end{pmatrix} = \begin{pmatrix} p_0 \\ \vdots \\ p_{\frac{u}{2}-2} \\ p_{\frac{u}{2}-1} \end{pmatrix} - \boldsymbol{B} \begin{pmatrix} p_{\frac{u}{2}} \\ \vdots \\ p_{u-2} \\ p_{u-1} \end{pmatrix} = \left( \begin{array}{c|c} I_{\frac{u}{2}} & -\boldsymbol{B} \end{array} \right) \begin{pmatrix} p_0 \\ \vdots \\ p_{\frac{u}{2}-1} \\ - - - \\ p_{\frac{u}{2}} \\ \vdots \\ p_{u-1} \end{pmatrix} \quad (5)$$

, where $I_{u/2} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$ is the identity matrix of order $u/2$.

**Computational cost:** It should be noticed that using Toeplitz matrix vector multiplication, the product $\boldsymbol{B} \begin{pmatrix} p_{u/2} \\ \vdots \\ p_{u-2} \\ p_{u-1} \end{pmatrix}$ can be computed in $2(9u/2)\log u + 2(u/2) = 9u\log u + u$.

Also the product $I_{u/2} \begin{pmatrix} p_{u/2-1} \\ \vdots \\ p_0 \end{pmatrix}$ needs $u/2$ multiplications.

Now the grand total number of arithmetic operations needed to obtain the column vector

$\begin{pmatrix} r_{u/2-1} \\ r_{u/2-2} \\ \vdots \\ r_0 \end{pmatrix}$ is $9u\log u + u + u/2 = 9u\log u + 3u/2$ .

Now let be the total number of operations required to carry out the entire problem of order $n$. At level 1 of Figure 1 , the order problem is split into two sub- problem of order $n/2$ each and hence the number of arithmetic operations required to compute $r_{(0,1)}(z)$ and $r_{(1,1)}(z)$ is equal to

$2(9n\log+3n/2)=8\ n\log n+3n$ .Now we have the following recurrence:

$$T(n)=2T(n/2)+2(9n\log n+3n/2)=2T(n/2)+8\ n\log n+3n \qquad (6)$$

By iterating (6) ,we can show that

$$T(n)\le 8\ n\log^2 n+3n\log n\text{, whenever } n>1.$$

That is, the matrix vector product $V^T f$ can be done in $O(n\log^2 n)$ arithmetic operations.

To put it another way, the polynomial $M(z)=f_0+f_1z+\cdots+f_{n-1}z^{n-1}$ can be computed at all the points $z_0,z_1,\cdots,z_{n-1}$ in $O(n\log^2 n)$ arithmetic operations .

---

**Algorithm 3.2.** Fast multipoint evaluation of univariate polynomials

---

**Input:** $n=2^k, Z=\{z_0,\cdots,z_{n-1}\}, f(z)=f_0+f_1z+\cdots f_{n-1}z^{n-1}$ ,where $z_i,f_i\in\mathbb{C}\ \ (i=0,\cdots,n-1)$

**Output:** $f(z_0)f(z_1),\cdots,f(z_{n-1})$.

**Stages: r**

0 If $n=1$, then return $y=V^T.f$

1 $K\leftarrow\log_2 n$

for $k=K$ to 1 do

for $j=0$ to $2^k-1$ do

$r\leftarrow j(n/2^k)$

$s\leftarrow(j+1)(n/2^k)-1$

$u\leftarrow n/2^k$

$m_{(i,j)}(z)\leftarrow(z-z_r)\ldots(z-z_s)\ \leftarrow m_0+m_1z+\cdots m_{u/2-1}z^{u/2-1}+m_{u/2}z^{u/2}$ ←Using FPM

2 for $k=1:K$

for j=1: $2^k$

$A(j.k)\leftarrow$ Toeplitz matrix generated by $\{0,0,\cdots,m_0,m_0,m_1,\cdots,m_{u/2-1}\}$ the elemen $0,0,\cdots,m_0$ in the first row and the element $m_0,m_1,\cdots,m_{u/2-1}$ are in the first column of the matrix.

$B(j,k)\leftarrow$ Toeplitz matrix generated by $\{m_{u/2},m_{u/2-1},\cdots,0,0,\cdots,0\}$ ; the elements $m_{u/2},m_{u/2-1},\cdots,m_1$ are in the first row and the elements $m_{u/2},0,\cdots,0$ are in the first column of the matrix.

$C(j,k) \leftarrow$ Inverse of $B(j,k)$

endfor(loopj)

end for (loop k)

3

$r_{(j,k)}(z) \leftarrow r_0(j,k) + r_1(j,k)z + \cdots r_{u-1}(j,k)z^{u-1}$

If k=1, then

for j=1:2

$$\begin{pmatrix} r_0(j,k) \\ r_1(j,k) \\ \vdots \\ r_{n/2-1}(j,k) \end{pmatrix} = \left(\begin{array}{c|c} I_{d/2} & A(j,k)C(j,k) \end{array}\right) \begin{pmatrix} f_0 \\ \vdots \\ f_{n/2-1} \\ \hline f_{n/2} \\ \vdots \\ f_{n-1} \end{pmatrix} \leftarrow \text{Using TMVM}$$

endif

for k=1:K-1

for j=0:$2^k$-1

for m=2j:2j+1

$$\begin{pmatrix} r_0(m,k+1) \\ r_1(m,k+1) \\ \vdots \\ r_{d/2-1}(m,k+1) \end{pmatrix} = \left(\begin{array}{c|c} I_{d/2} & -A(m,k+1)C(m,k+1) \end{array}\right) \begin{pmatrix} r_0(j,k) \\ \vdots \\ r_{d/2-1}(j,k) \\ \hline r_{d/2}(j,k) \\ \vdots \\ r_{d-1}(j,k) \end{pmatrix} \leftarrow$$

Using TMVM

endfor (loop m)

endfor (loop j)

endfor( loop k)

**Note that fast polynomials multiplication (FPM):** The multiplication of two polynomials of degree $n$ each can be performed efficiently in $O(n\log n)$ arithmetic operations using the FFT (see [2] , p.86).Using this we can find all the coefficients $m_i$ of all the polynomials in the form of $m_{(i,j)}(z)$ in Algorithm 3.

**4. Numerical Experiment**: Here both the naïve algorithm and the optimized algorithm ( ) were implemented in MATLAB soft ware in order to compute the CPU time taken by each algorithm to evaluate Chebyshev polynomials of first kind at the Chebyshe nodes. We calculated the times taken for the Chebyshev polynomials of degrees of 8, 16, 32, 64, 128, 256, 512.

**Example 4.1:** Let $x \in [-1,1]$.The Chebyshev polynomials of the first kind are the sequence of polynomials defined by:

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad T_0(x), \quad T_1(x) = x .$$

These polynomials also satisfy $T_k(\cos\theta) = \cos(\cos k\theta)$ for all real $\theta$. Chebyshev nodes are given by: $x_j^N = \cos\left(\frac{j\pi}{n}\right), j = 0,\cdots,n$ .

In the following table we summarize the times taken for each algorithm. The column labeled "$n$" lists the length (=the number of coefficients) of a polynomial. The column tabled "t (Direct)" lists the times taken to compute the polynomials using the direct method. The column labeled "t(Fast)" lists the time taken to compute the polynomials using the fast algorithm.

**Table 1**

Times taken by the direct and fast algorithms for the Chebyshev polynomials.

| N | t(Direct | t(Fast) |
|---|---|---|
| 8 | 0 | 0 |
| 16 | 0.0156 | 0 |
| 32 | 0.4368 | 0.0780 |
| 64 | 2.059 | 0.0468 |
| 128 | 20.81 | 0.3744 |
| 256 | 156.4 | 2.778 |
| 512 | 1743 | 37.53 |

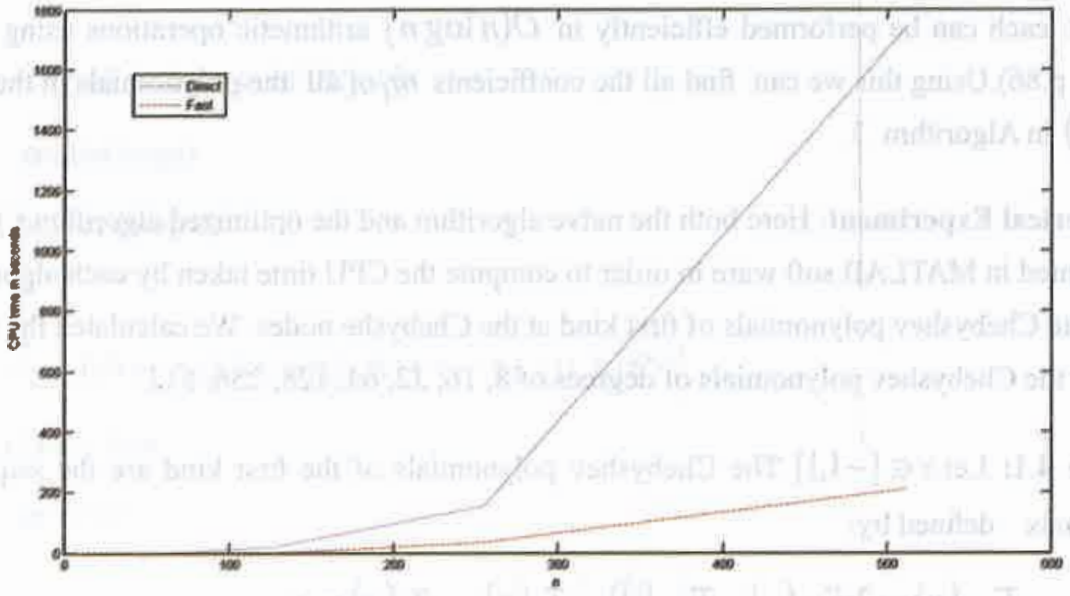


**Figure 2.**Comparition of the fast algorithm with the direct method for evaluating the Chebyshev polynomials.

**5 Conclusion:** We have presented an $O(n\log^2 n)$ algorithm to evaluate simultaneously a univariate polynomial of degree $n-1$ at $n$ arguments, where $n$ is a power of 2 We implemented this algorithm in MATLAB software to evaluate the Chebyshev polynomial of the first kind of various degrees at the Chebyshev nodes to examine its efficiency. These experimental results have demonstrated that the fast algorithm presented here is much faster than the corresponding naïve algorithm when $n \geq 128$. The basic idea of this fast algorithm is to factor the transpose of the Vandermond matrix into sparse matrices each including products of triangular Toeplitz sub blocks along its diagonal, so that the hybrid technique of FFT and the circulant matrix vector product can be applied to reduce the number of operations. Hence, similar strtructured factorizations for the Vandermonde matrix and the inverse of the Vandermonde matrix can be obtained so that we may solve problems efficiently like interpolation problems and systems of linear equations with Vandermonde matrices in $O(n\log^2 n)$ arithmetic operations.

Despite particular cases like FFT, the fastest algorithms to compute the polynomials, reported in the literature to date, require $O(n\log^2 n)$ arithmetic operations (in particular $8\ n\log^2 n + 3n\log n$ at most in the present paper). Then the obvious question is whether the approach described in the present paper can improve this bound for some or all types of univariate polynomial evaluation. Commenting on this question and numerical stability of the presented algorithm will be the subject of a forthcoming paper.

## REFERENCES

[1] Alin Bostan and Eric Schost, *On the complexities of multipoint evaluation and Interpolation.* Laboratoire STIX, Ecole polytechnique, 91128 Palaiseau, France.

[2] A.Borodin and I.Munro, *The Computational Complexity of Algebraic and Numeric Problems*. Elsevier, New York 1975.

[3] W.Cooly and John W . Tukey, An *Algorithm for the Machine Calculation of Complex Fourier Series*, Math.Comput. 19-297-302, 1965.

[4] J.R. Driscoll, D.M. .Healy. JR, and D.Rockmore, *Fast Discrete Polynomial Transforms with Applications to Data Analysis for Distance Transitive Graphs*.

[5] I.Gohberg and V.Olshevsky, *Fast algorithms with preprocessing for matrix – vector multiplication problems.* School of Mathematical Sciences ,Raymond and Beverly Sackler Faculty of Exact Sciences. Journal of Complexity (1994) .

[6] Jams S. Walker, *Fast Fourier Transforms*, second edition,(studies in Advanced Mathematics), 0-8493-7163-5, 1996.

[7] Kenneth H.Rosen, *Discrete Mathematics and its Applications,* fifth edition, ISBN 0-07-119881-4.

[8] Michael Nusken and Martin Ziegler, Fast Multipoint Evaluation of Bivariate Polynomials. University of Paderborn, 33095 Paderborn, GERMANY.

[9] Robert M. Gray, Department of Electrical Engineering, Stanford University, USA, Toeplitz and Circulant Matrices: A review.

[10] Zhhui Tang, Ramani Duraiswami, and Nail Gumerov, *Fast Algorithms to Compute Matrix-Vector Products for Pascal Matrices*