# A New Parallel Algorithm For Fast Generation of Discrete Chebyshev Polynomials

BELGACEM BEN YOUSSEF
School of Interactive Arts and Technology
Simon Fraser University
2400 Central City, 10153 K. G. Highway
Surrey, British Columbia, V3T 2W1
CANADA

## Abstract

We present a new parallel algorithm for the fast generation of discrete Chebyshev polynomials. By fast we mean that the time complexity of the obtained parallel algorithm is of order $O(\log n)$, where $n$ is the degree of the $(n+1)^{st}$ polynomial to be generated, and the number of available processors is assumed to be equal to a polynomial order of the input size. The parallel algorithm makes extensive use of parallel algorithms for the well-known *prefix* computation problem. The parallel generation of orthogonal polynomials has numerous applications in approximation theory, interpolation, and numerical analysis.

*Key-Words:* - Parallel algorithms, parallel prefix computation, matrix multiplication, discrete Chebyshev polynomials.

## 1 Introduction

### 1.1 Definition and Properties

The Chebyshev polynomials were discovered more than a century ago by the Russian mathematician *Pafnouty Lvovitch Chebyshev*. Denoted by $T_i(x)$ for $i \geq 0$, they satisfy the orthogonality relationship given by,

$$\int_{-1}^{1} T_i(x) T_j(x) w(x)\, dx = 0 \,,$$

for $i \neq j$, and

$$\int_{-1}^{1} T_i^2(x) w(x)\, dx = \left\langle \begin{array}{lll} \frac{\pi}{2}\,, & \text{for} & i \neq 0\,, \\ \pi\,, & \text{for} & i = 0\,, \end{array} \right.$$

while assuming a weighting function $w(x) = (1-x^2)^{-\frac{1}{2}}$ [4]. These polynomials can be obtained by using the following three-term recursion relation

$$T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x)\,, \tag{1}$$

where $i \geq 1$, $T_0(x) = 1$, and $T_1(x) = x$. Below, we list the first seven Chebyshev polynomials

$$\begin{array}{l} T_0(x) = 1\,, \\ T_1(x) = x\,, \\ T_2(x) = -1 + 2x^2\,, \\ T_3(x) = -3x + 4x^3\,, \\ T_4(x) = 1 - 8x^2 + 8x^4\,, \\ T_5(x) = 5x - 20x^3 + 16x^5\,, \\ T_6(x) = -1 + 18x^2 - 48x^4 + 32x^6\,. \end{array} \tag{2}$$

We also observe the following:

- Since $T_i(-x) = (-1)^i\, T_i(x)$, this makes $T_i(x)$ an even function of $x$ when $i$ is even, and an odd function of $x$ when $i$ is odd.
- The nonzero coefficients of $T_i(x)$ are integers that alternate in sign whereas the leading coefficient, that is the coefficient of $x^i$ in $T_i(x)$, is always a positive integer.
- The leading coefficient of $T_i(x)$ is equal to $2^{i-1}$ for $i > 0$ and 1 otherwise.
- Any $T_i(x)$, $i \geq 0$, is determined by $\lfloor \frac{i}{2} + 1 \rfloor$ nonzero coefficients.

Thus, the Chebyshev polynomials are *simple* in the sense that any $T_i(x)$ is completely specified by a finite number of coefficcicents. It is this finiteness that makes these polynomials particularly suitable as approximations to more complicated functions [15].

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 4 & 0 & 0 & 0 \\ 1 & 0 & -8 & 0 & 8 & 0 & 0 \\ 0 & 5 & 0 & -20 & 0 & 16 & 0 \\ -1 & 0 & 18 & 0 & -48 & 0 & 32 \end{bmatrix}$$

Figure 1: The matrix $S$ when $n = 6$.

Let $S = [S_{i,k}]$ be an $(n+1) \times (n+1)$ matrix in which the $(i+1)^{st}$ row consists of the coefficients of $T_i(x)$, i.e.,

$$T_i(x) = \sum_{k=0}^{i} S_{i,k} x^k ,$$

and where $S_{i,k}$ represents the $(k+1)^{st}$ coefficient of $T_i(x)$. We note that $S$ is a lower triangular matrix whose entries satisfy a doubly-indexed recursion relation given by

$$S_{i+1,k} = 2S_{i,k-1} - S_{i-1,k} , \tag{3}$$

for $0 \le k < i \le n-1$ and assuming that $S_{i,-1} = S_{-1,k} = 0$ for $0 \le i,k \le n$. From the previous discussion, the diagonal entries of matrix $S$ — which correspond to the leading coefficients of the Chebyshev polynomials — are then yielded as follows

$$S_{i,i} = \begin{cases} 2^{i-1}, & \text{if} \quad i > 0 , \\ 1, & \text{if} \quad i = 0. \end{cases} \tag{4}$$

Furthermore, starting with a nonzero diagonal, $S$ is found to depict alternating zero and nonzero subdiagonals . This is due to the odd-even property of these polynomials mentioned earlier. Figure 1 displays the matrix $S$ for $n = 6$.

### 1.2 Parallel Prefix Computation

The prefix computation problem is to compute all $n$ initial products $q_1 \circ q_2 \circ \cdots \circ q_i$, for $1 \le i \le n$ of a set $Q$ of $n$ elements $\{q_1, q_2, \ldots, q_n\}$, where $\circ$ is an associative binary operation on the set $Q$. The prefix computation is a powerful and common operation that is used in many algorithms. It occurs in the evaluation and generation of polynomials, general Horner expressions, and general arithmetic formulae. In addition, its utilization can be seen in the solution of linear recurrences, carry look-ahead circuits, ranking and packing problems, and scheduling problems. In addition, it is extensively used in the design of parallel algorithms, partly because it is useful in reallocating processors to subproblems [6, 7, 8]. A *parallel prefix algorithm* represents the result of the parallelization of the prefix computation problem. We present without proof the following lemma [12].

**Lemma 1** *The $n$ input parallel prefix computation can be performed in $\lceil \log n \rceil$ time with $n$ processors*[1].

Below, we present a detailed description of the parallel prefix algorithm and summarize the required computations.

```
PROCEDURE Parallel.Prefix($n, q$)
    FOR $j := 0$ TO $\lceil \log n \rceil - 1$ DO
        FORALL $i \in \{2^j + 1, \ldots, n\}$ DO IN PARALLEL
            new$q[i] := q[i] \circ q[i - 2^j]$
        END FORALL
        FORALL $i \in \{2^j + 1, \ldots, n\}$ DO IN PARALLEL
            $q[i] :=$ new$q[i]$
        END FORALL
    END FOR
END PROCEDURE $\{q[i]$ contains $q_1 \circ q_2 \circ \cdots \circ q_i\}$.
```

We observe that the outer *FOR-DO* loop is executed $\lceil \log n \rceil$ times whereas the two inner *FORALL-DO-IN-PARALLEL* loops are each executed once every pass. We assume that identical arithmetic processors, each of which can perform any one of the binary operations of $+$, $-$, $\times$, or $\div$ in unit time, are available and may be reused. Moreover, a floating-point arithmetic operation is assumed to take a single step while all issues related to memory access and programming models are ignored.

## 2 Parallel Generation

Given a particular value for $n$, our objective is to generate the set of all Chebyshev polynomials $T_0(x), T_1(x), \ldots, T_n(x)$. Since $T_0(x)$, and $T_1(x)$ are known in advance, our task is then reduced to the calculation of the $n-1$ Chebyshev polynomials: $T_2(x), T_3(x), \ldots, T_n(x)$.

### 2.1 Derivation of the Parallel Algorithm

Using the fact that $S$ is a lower triangular matrix in conjunction with Equation (3) and Equation (4) yields the following

$$S_{i,k} \quad = \quad 0 , \quad \text{for} \quad 0 \le i < k \le n .$$

The zero subdiagonals are then given by

$$S_{i,i-1} = S_{i,i-3} = \cdots = S_{i,i-\delta} = 0 ,$$

[1]Throughout this paper $\log n = \log_2 n$, unless stated otherwise.

for $1 \leq i \leq n$ and $\delta = 2\alpha + 1$, where $0 \leq \alpha \leq \lfloor \frac{n}{2} \rfloor$ if $n$ is odd; and $0 \leq \alpha \leq \lfloor \frac{n}{2} - 1 \rfloor$ if $n$ is even. On the other hand, the entries in the first nonzero subdiagonal conform to

$$S_{i,i-2} = 2S_{i-1,i-3} - S_{i-2,i-2} \; , \tag{5}$$

for $2 \leq i \leq n$. We also note that the entries in the $k^{th}$ nonzero subdiagonal are equal to $S_{i,i-2k}$, where $2k \leq i \leq n$.

Let $X^0$ be a column vector of length $n+1$ containig all the diagonal entries. It follows that the $(i+1)^{st}$ entry of $X^0$ is equal to

$$X_i^0 = S_{i,i} \; , \tag{6}$$

where $i$ varies from 0 to $n$. Next, we define $X^1$ to be a column vector that contains the $n-1$ entries of the first nonzero subdiagonal in matrix $S$. Thus, the first two entries in this subdiagonal are now given by

$$\begin{cases} X_{i-2}^1 = S_{i,i-2} \; , \\ X_{i-3}^1 = S_{i-1,i-3} \; , \end{cases}$$

where $2 \leq i \leq n$. Consequently, by substituting the above equalities into Equation (5) and shifting up the indices by two, a new equivalent equation can be written as follows

$$X_i^1 = 2X_{i-1}^1 - X_i^0 \; , \tag{7}$$

for $0 \leq i \leq n-2$. We observe that Equation (7) is similar to a *first-order linear recurrence* relation. Similarly, the second nonzero subdiagonal of $S$ has entries that satisfy the following formula

$$S_{i,i-4} = 2S_{i-1,i-5} - S_{i-2,i-4} \; , \tag{8}$$

for $4 \leq i \leq n$. The last term in the above equation is equal to $X_{i-4}^1$, which represents the $(i-3)^{rd}$ entry in the first nonzero subdiagonal. We then define $X^2$ as a column vector that contains the $n-3$ entries of the second nonzero subdiagonal and allow for the first two terms in Equation (8) to be redefined as follows

$$\begin{cases} X_{i-4}^2 = S_{i,i-4} \; , \\ X_{i-5}^2 = S_{i-1,i-5} \; . \end{cases}$$

As a result, the following relation is obtained after appropriately shifting up the indices

$$X_i^2 = 2X_{i-1}^2 - X_i^1 \; , \tag{9}$$

for $0 \leq i \leq n-4$. Since the term $X_i^1$ is given by Equation (7), it is substituted into the above equation to yield

$$\begin{aligned} X_i^2 &= 2X_{i-1}^2 - 2X_{i-1}^1 + X_i^0 \\ &= -2X_{i-1}^1 + 2X_{i-1}^2 + X_i^0 \; . \end{aligned} \tag{10}$$

It can be shown that the coefficient matrix $S$ has exactly $d = \lfloor \frac{n}{2} \rfloor$ nonzero subdiagonals. Thus, we need to generalize the aforementioned results. In order to do this, we allow the column vector $X^i$ to denote the vector containing the entries of the $i^{th}$ nonzero subdiagonal, whereas $X_j^i$ corresponds to the $(j+1)^{st}$ element in that particular subdiagonal. It follows that the remaining nonzero subdiagonals of matrix $S$ have their entries satisfying the following equations

$$\begin{aligned} X_i^3 &= 2X_{i-1}^3 - X_i^2 \\ &= 2X_{i-1}^3 - 2X_{i-1}^2 + 2X_{i-1}^1 - X_i^0 \\ &= 2X_{i-1}^1 - 2X_{i-1}^2 + 2X_{i-1}^3 - X_i^0 \; , \end{aligned}$$

$$\begin{aligned} X_i^4 &= 2X_{i-1}^4 - X_i^3 \\ &= 2X_{i-1}^4 - 2X_{i-1}^3 + 2X_{i-1}^2 - 2X_{i-1}^1 + X_i^0 \\ &= -2X_{i-1}^1 + 2X_{i-1}^2 - 2X_{i-1}^3 + 2X_{i-1}^4 + X_i^0 \; , \\ &\vdots \end{aligned}$$

and

$$\begin{aligned} X_i^d &= 2X_{i-1}^d - X_i^{d-1} \\ &= 2X_{i-1}^d - 2X_{i-1}^{d-1} + 2X_{i-1}^{d-2} - \cdots + 2X_{i-1}^1 - X_i^0 \\ &= 2X_{i-1}^d - 2X_{i-1}^{d-1} + \cdots + 2X_{i-1}^2 - 2X_{i-1}^1 + X_i^0 \; . \end{aligned} \tag{11}$$

We note that the algebraic signs of the coefficients of the terms on the right hand side depend solely on the value of $d$. That is, if $d$ is even, then the coefficient of $X_{i-1}^1$ is negative; otherwise it is positive. By incorporating the $\pm$ and the $\mp$ symbols, Equation (11) can be written as

$$\begin{aligned} X_i^d &= 2X_{i-1}^d - 2X_{i-1}^{d-1} + 2X_{i-1}^{d-2} - \cdots \pm 2X_{i-1}^1 \mp X_i^0 \\ &= \pm 2X_{i-1}^1 \mp 2X_{i-1}^2 \pm 2X_{i-1}^3 \mp \cdots + 2X_{i-1}^d \mp X_i^0 \; . \end{aligned}$$

The main point, here, is to notice that the coefficients of the terms on the right hand side of the equation of any $X_i^j$ do alternate in sign, while the first $d$ coefficients of $X_{i-1}^j$ in the same equation are always equal to 2 in the absolute-value sense. These vector recursion relations allow us to calculate all the $X^i$'s for $1 \leq i \leq d$ provided that the vector containing all the diagonal entries, $X^0$, is available. The length of all column vectors $X^0, X^1, \ldots, X^d$ is assumed to be equal to $n+1$.

The above recurrence formulae can be conveniently put in a matrix form as

$$\begin{bmatrix} X_i^1 \\ X_i^2 \\ X_i^3 \\ X_i^4 \\ X_i^5 \\ \vdots \\ X_i^d \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & -X_i^0 \\ -2 & 2 & 0 & \cdots & 0 & X_i^0 \\ 2 & -2 & 2 & \cdots & 0 & -X_i^0 \\ -2 & 2 & -2 & \cdots & 0 & X_i^0 \\ 2 & -2 & 2 & \cdots & 0 & -X_i^0 \\ \vdots & \vdots & \ddots & \cdots & \vdots & \vdots \\ \pm 2 & \mp 2 & \mp 2 & \cdots & 2 & \mp X_i^0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{i-1}^1 \\ X_{i-1}^2 \\ X_{i-1}^3 \\ X_{i-1}^4 \\ X_{i-1}^5 \\ \vdots \\ X_{i-1}^d \\ 1 \end{bmatrix} .$$

We observe that the entry $X_i^0$ of the above square matrix is preceded by a minus sign whenever $d$ is odd, and vice versa. We let $Y_i$ denote the column vector on the left hand side of the previous equation. This is a $(d+1) \times 1$ vector containing as its first $d$ elements the $i^{th}$ entry of each nonzero subdiagonal. Its $(d+1)^{st}$ entry is equal to 1. It follows that the other $(d+1) \times 1$ vector appearing on the right hand side of the same equation is equal to $Y_{i-1}$. Furthermore, we allow $M_i$ to be the square matrix of rank $d+1$ shown in the above equation. This matrix can be easily obtained, for the entries in the last column are known in advance, and are given by Equation (6). A new and more concise equation is defined as follows

$$Y_i = M_i Y_{i-1} . \qquad (12)$$

Because it is known that, for a given value of $n$, the number of entries in the diagonal of matrix $S$ is equal to $n+1$, and since a zero subdiagonal — of $n$ zeros — precedes the first nonzero subdiagonal (as one proceeds downward from the diagonal), then the first nonzero subdiagonal has exactly $n-1$ entries. Since no other nonzero subdiagonal contains more elements, we conclude that the index $i$ in Equation (12) varies from 0 to $n-2$ to yield the needed entries of the $d$ nonzero subdiagonals. Moreover, we assume that $S_{i,-1} = S_{-1,k} = 0$ for $0 \leq i,k \leq n$. This implies that

$$Y_{-1} = \begin{bmatrix} X_{-1}^1 \\ X_{-1}^2 \\ X_{-1}^3 \\ X_{-1}^4 \\ X_{-1}^5 \\ \vdots \\ X_{-1}^d \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} .$$

Hence, $Y_{-1}$ can be readily obtained at no computational cost. By repeatedly applying Equation (12) for $0 \leq i \leq n-2$, we obtain the following results

$$\begin{aligned} Y_0 &= M_0 Y_{-1} , \\ Y_1 &= M_1 M_0 Y_{-1} , \\ Y_2 &= M_2 M_1 M_0 Y_{-1} , \\ &\vdots \\ Y_{n-3} &= M_{n-3} M_{n-4} \ldots M_2 M_1 M_0 Y_{-1} , \end{aligned}$$

and

$$Y_{n-2} = M_{n-2} M_{n-3} \ldots M_2 M_1 M_0 Y_{-1} . \qquad (13)$$

Therefore, in addition to $Y_{-1}$, $n-1$ square matrices — $M_0$, $M_1$, ... , $M_{n-3}$, $M_{n-2}$ — are needed as inputs for the computation of the entries in the nonzero subdiagonals of $S$. Let $C_i$ be a $(d+1) \times (d+1)$ matrix such that

$$C_i = M_i M_{i-1} \ldots M_2 M_1 M_0 ,$$

for $0 \leq i \leq n-2$. Consequently, $Y_i = C_i Y_{-1}$ where $i$ varies from 0 to $n-2$. The computation of $C_0$, $C_1$, ... , $C_{n-3}$, and $C_{n-2}$ essentially calculates the $n-1$ prefix products of the set of $(d+1) \times (d+1)$ matrices $\{M_0, M_1, M_2, \ldots, M_{n-3}, M_{n-2}\}$. Moreover, we observe that since $Y_{-1}$ is equal to a column vector of $d$ zeros, and a last entry of 1, the final multiplication of $Y_{-1}$ with $C_i$ to calculate $Y_i$ for $0 \leq i \leq n-2$ is not needed. This is because the vectors $Y_0$, $Y_1$, ... , and $Y_{n-2}$ can be determined by just taking the last column in $C_0$, $C_1$, ... , and $C_{n-2}$, respectively.

## 2.2 Construction of the Matrix $S$

As mentioned earlier, $Y_i$ yields the $(i+1)^{st}$ entries in each of the possible nonzero subdiagonals. Due to the fact that the number of subdiagonal entries linearly decreases from right to left (i.e., going from column $n$ to column 0 and starting from the diagonal), not all of the entries in the $Y_i$'s are going to be used to generate the Chebyshev polynomials. Therefore, once $Y_0, Y_1, \ldots,$ and $Y_{n-2}$ are computed, we are left with the task of determining the matrix $S$ in order to obtain the coefficients of all the Chebyshev polynomials. The following is an algorithm for constructing $S$.

**The Construct.$S$ Algorithm**

**Input.** $n$, $d = \lfloor \frac{n}{2} \rfloor$, and $Y_i$ for $0 \leq i \leq n-2$.

**Output.** The $(n+1) \times (n+1)$ coefficient matrix $S$.

**Step 1.** Set $S_{i,k} = 0$, for $0 \leq i < k \leq n$.

**Step 2.** Set $S_{i,i} = X_i^0$, for $0 \le i \le n$.

**Step 3.** Use the following procedure to set the entries in the zero subdiagonals equal to 0.

```
FOR $i = 1$ TO $n$ STEP 2 DO
    IF $i \le n$ THEN DO
    BEGIN
        FOR $j = 0$ TO $n - i$ DO
            $S_{i,j} = 0$
            $i = i + 1$
        END FOR
    END {of if-then}
END FOR
```

The following two steps allow us to obtain the nonzero entries of matrix $S$ from the calculated vectors $Y_0, Y_1, \dots, Y_{n-2}$.

**Step 4.** If $n$ is even, then:

**4.1** The first $d$ elements of $Y_0$ are the first entries of the $d$ nonzero subdiagonals.

- The following code gives the procedure to extract the first $d$ entries of $Y_0$.

```
FOR $j = 1$ TO $d$ DO
    $S_{2j,0} = Y_0(j)$
END FOR
```

**4.2** Extract the first $d-1$ entries of both $Y_1$, and $Y_2$, where $Y_1$ ($Y_2$) has the second (third) entries of the remaining $d-1$ nonzero subdiagonals.

- Extraction procedure from $Y_1$ :

```
FOR $j = 1$ TO $d - 1$ DO
    $S_{2j+1,1} = Y_1(j)$
END FOR
```

- Extraction procedure from $Y_2$ :

```
FOR $j = 1$ TO $d - 1$ DO
    $S_{2j+2,2} = Y_2(j)$
END FOR
```

**4.3** Repeat 4.2 by replacing $Y_1$ and $Y_2$ with $Y_3$ and $Y_4$, and $d-1$ with $d-2$, etc ... until $Y_{n-2}$ is reached (the entries in $S$ are represented by $S_{2j+3,3}$ and $S_{2j+4,4}$, respectively).

**Step 5.** If $n$ is odd, then:

**5.1** The first $d$ entries $Y_0$ ($Y_1$) are the first (second) entries of the $d$ nonzero subdiagonals.

$$S = \begin{bmatrix} X_0^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & X_1^0 & 0 & 0 & 0 & 0 & 0 \\ Y_0(1) & 0 & X_2^0 & 0 & 0 & 0 & 0 \\ 0 & Y_1(1) & 0 & X_3^0 & 0 & 0 & 0 \\ Y_0(2) & 0 & Y_2(1) & 0 & X_4^0 & 0 & 0 \\ 0 & Y_1(2) & 0 & Y_3(1) & 0 & X_5^0 & 0 \\ Y_0(3) & 0 & Y_2(2) & 0 & Y_4(1) & 0 & X_6^0 \end{bmatrix}$$

Figure 2: Illustration of the algorithm Construct.$S$ when $S$ is a $7 \times 7$ matrix ($n = 6$).

- This code constitutes the procedure needed to extract the first $d$ entries of $Y_0$.

```
FOR $j = 1$ TO $d$ DO
    $S_{2j,0} = Y_0(j)$
END FOR
```

- This is the procedure needed to extract the first $d$ entries of $Y_1$.

```
FOR $j = 1$ TO $d$ DO
    $S_{2j+1,1} = Y_1(j)$
END FOR
```

**5.2** Use only the first $d-1$ entries of both $Y_2$, and $Y_3$, where $Y_2$ ($Y_3$) has the third (fourth) entries of the remaining $d-1$ nonzero subdiagonals.

- Extraction procedure from $Y_2$ :

```
FOR $j = 1$ TO $d - 1$ DO
    $S_{2j+2,2} = Y_2(j)$
END FOR
```

- Extraction procedure from $Y_3$ :

```
FOR $j = 1$ TO $d - 1$ DO
    $S_{2j+3,3} = Y_3(j)$
END FOR
```

**5.3** Repeat 5.2 by replacing $Y_2$ and $Y_3$ with $Y_4$ and $Y_5$ and $d-1$ with $d-2$, etc ... until $Y_{n-2}$ is processed (the entries in $S$ are represented by $S_{2j+4,4}$ and $S_{2j+5,5}$, respectively).

At the end, the matrix $S$ is obtained.

**Example 1** *Figure 2 illustrates the use of the Construct.S algorithm in the case of* $n = 6$, $d = 3$, *and having* $Y_0, Y_1, \dots, Y_4$ *available.*

## 2.3 The Parallel Algorithm and its Complexity

We call our new parallel algorithm the *PGCP Algorithm*, where PGCP stands for Parallel Generation of Chebyshev Polynomials. It is described as follows:

**The PGCP Algorithm**

**Input.** $n$, $d = \lfloor \frac{n}{2} \rfloor$, $T_0(x)$, $X_0^0$, $T_1(x)$, and $X_1^0$.

**Output.** $T_2(x), T_3(x), \ldots, T_n(x)$.

**Step 1.** Compute in parallel the diagonal entries $X_2^0, \ldots, X_n^0$.

**Step 2.** Form the $n-1$ $(d+1) \times (d+1)$ matrices $M_0, M_1, \ldots, M_{n-2}$.

**Step 3.** Compute in parallel the $n-1$ $(d+1) \times (d+1)$ matrices $C_0, C_1, \ldots, C_{n-2}$.

**Step 4.** Form the $n-1$ $(d+1) \times 1$ column vectors $Y_0, Y_1, \ldots, Y_{n-2}$ by taking the last column of $C_0, C_1, \ldots, C_{n-2}$, respectively.

**Step 5.** Use the Construct.$S$ algorithm to form the matrix $S$.

**Step 6.** Get $T_2(x), T_3(x), \ldots, T_n(x)$ by applying the following equation: $T_i(x) = \sum_{k=0}^{i} S_{i,k} x^k$.

At the end of Step 6, all the Chebyshev polynomials are generated.

**Theorem 1** *The new PGCP algorithm requires $O(\log n)$ parallel arithmetic operations to compute the $n+1$ Chebyshev polynomials $T_0(x)$, $T_1(x)$, ..., $T_n(x)$ with $O(n^4)$ processors working in parallel.*

**Proof**. An inspection of the PGCP algorithm reveals that only two steps involve computations, namely, Step 1, and Step 3. In Step 1, $n-1$ concurrent processors can compute the last $n-1$ diagonal entries of matrix $S$ in $\lceil \log(n-1) \rceil$ parallel arithmetic steps. Since, the set $\{2, 2^2, \ldots, 2^{n-1}\}$ is to be calculated here, a parallel prefix algorithm can be used where all the inputs to the algrithm are equal to 2, and multiplication is employed as the associative binary operation. Likewise, Step 3 basically calculates in parallel the $n-1$ prefix matrix products of the $n-1$ $(d+1) \times (d+1)$ matrices of the set $\{M_0, M_1, \ldots M_{n-2}\}$. The output is equal to the set $\{C_0, C_1, \ldots, C_{n-2}\}$. It is known that the product of two $n \times n$ matrices can be achieved in $O(\log n)$ time while having $O(n^3)$ processors workng in parallel [2]. Furthermore, it is well known that the $n$ input parallel prefix computation is performed in $\lceil \log n \rceil$ time with $n$ processors. Hence, by combining these two well known results, we deduce that the parallel prefix algorithm can be utilized to compute $C_0$, $C_1$, ..., $C_{n-2}$ in $O(\log n)$ parallel arithmetic steps while using $O(n^4)$ processors in parallel, where matrix multiplication is the associative binary operation employed in the algorithm. As result, the PGCP algorithm computes the Chebyshev polynomials in $O(\log n)$ parallel time with $O(n^4)$ parallel processors.

## 3 Related Work

Eğecioğlu and Koç presented in [5] a sequential algorithm and its parallel version to construct an orthogonal family of polynomials $p_i(x)$ for $0 \le i \le n$ with respect to a given discrete inner product given by

$$\langle u(x), v(x) \rangle = \sum_{j=0}^{n} w_j u(x_j) v(x_j) \ ,$$

and a three-term recursion formula

$$p_{i+1}(x) = (x - \alpha_i) p_i(x) - \beta_i p_{i-1}(x) \ ,$$

for $0 \le i \le n-1$ while having $p_{-1}(x) = 0$ and $p_0(x) = 1$. The constants $\alpha_i$ and $\beta_i$ are determined as follows

$$\alpha_i = \frac{\langle x p_i(x), p_i(x) \rangle}{\langle p_i(x), p_i(x) \rangle} \ , \quad \beta_i = \frac{\langle p_i(x), p_i(x) \rangle}{\langle p_{i-1}(x), p_{i-1}(x) \rangle} \ .$$

The generated polynomials are monic polynomials. This algorithm generates both the coefficients and values, that is $p_i(x_j)$ for $0 \le i, j \le n$, of the orthogonal polynomials. However, the input to the parallel algorithm is dependent on the use of two data sets of $n+1$ distinct values representing the node points $x_j$ and positive weights $w_j$ for $0 \le j \le n$, respectively. Furthermore, this parallel algorithm does not employ the prefix computation in its methodology.

Other related work involved polynomial interpolation. Suppose we are given $n+1$ pairs of numbers $(x_i, y_i)$, $i = 0, 1, \ldots, n$ and an arbitrary real-valued function $f$, such that

1. $f$ is defined on the $x_i$ over an interval $[a, b]$,
2. $y_i = f(x_i)$, and
3. $a \le x_0 < x_1 < x_2 < \ldots < x_n \le b$.

The problem of *polynomial interpolation* is that of creating a polynomial $p_n(x)$ of degree $n$ using the pairs $(x_i, y_i)$ such that $p_n(x_i) = y_i$ for $i = 0, 1, \ldots, n$. In the *Newton's interpolation method,* such polynomial is obtained by using the *divided differences* of $f$. The other type of interpolation scheme is called the *Hermite* interpolation, wherein the aim is to construct the $n^{th}$ degree polynomial $p_n(x)$ that interpolates $f(x)$ from the give values of

$$f(x_i), f^{(1)}(x_i), f^{(2)}(x_i), \ldots, f^{(n_i-1)}(x_i) \; ,$$

for $i = 0, 1, 2, \ldots, n$, where $f^{(j)}(x_i)$ denotes the $j^{th}$ derivative of $f(x)$ at $x = x_i$. This method uses the *generalized divided differences* of $f$ to obtain the approximating polynomial $p_n(x)$. The reader is referred to [6], [7], [9], and [13] for more information on both of these interpolation methods and their associated parallel algorithms. In comparing our parallel algorithm to other parallel algorithms based on these two methods, we note the following

- Our parallel algorithm does not require $i + 1$ distinct data values for each Chebyshev polynomial $T_i(x)$ as inputs. It only needs $T_0(x)$ and $T_1(x)$, which are readily available.

- Our parallel algorithm is not an approximation to the Chebyshev polynomials as interpolation is. It is an exact construction of them.

- While all three parallel algorithms depend heavily on parallel prefix computations, they differ on the type of associative biary operation used. Our parallel algorithm uses matrix multiplication whereas the other two make extensive use of floating-point number addition or multiplication [7].

## 4 Conclusions

In this paper, we introduced a new parallel algorithm for the fast generation of discrete Chebyshev polynomials. We also discussed its time and space complexity. The algorithm was designed to take advantage of the parallel prefix computation method. Future work will include the design of parallel algorithms for the evaluation of these types of orthogonal polynomials first at a single data point and then at multiple data points and their implementation on a parallel machine with a limited number of processors.

## Acknowledgment

This work is partially supported by a start-up research grant from Simon Fraser University.

## References

[1] A. Abd Elsamea, H. Eldeeb, and S. Nassar, PC Cluster as a Platform for Parallel Applications, *Fourth WSEAS International Conference on Information Science, Communications and Applications,* Miami, FL, USA, April 21-23, 2004.

[2] S. G. Akl, *Parallel Computation: Models and Methods,* Prentice Hall, Inc., 1997.

[3] G. Blelloch, Scans as Primitive Parallel Operations, *Proceedings of the IEEE International Conference on Parallel Processing,* 1987, pp. 355–362.

[4] T. S. Chihara, *An Introduction to Orthogonal Polynomials,* Gordon and Breach Science Publishers, Inc., 1978.

[5] Ö. Eğecioğlu and Ç. K. Koç, A Parallel Algorithm for Generating Discrete Orthogonal Polynomials, *Parallel Computing,* Vol. 18, No. 6, 1992, pp. 649–659.

[6] Ö. Eğecioğlu, E. Gallopoulos, and Ç. K. Koç, A Parallel Method for Fast and Practical High-Order Newton Interpolation, *BIT,* Vol.30, No.2, 1990, pp. 268–288.

[7] Ö. Eğecioğlu, E. Gallopoulos, and Ç. K. Koç, Fast Computation of Divided Differences and Parallel Hermite Interpolation, *Journal of Complexity,* Vol.5, No.4, December 1989, pp. 417–437.

[8] Ö. Eğecioğlu, Ç. K. Koç, and A. J. Laub, A Recursive Doubling Algorithm for Solution of Triadiagonal Systems on Hypercube Multiprocessors, *Journal of Computational and Applied Mathematics,* Vol.27, No.1+2, 1989, pp. 95–108.

[9] F. B. Hildebrand, *Introduction to Numerical Analysis,* second edition, McGraw-Hill Publishing Company, 1974.

[10] C. P. Kruskal, L. Rudolph, and M. Snir, The Power of Parallel Prefix, *IEEE Transactions on Computers,* Vol. 34, No.10, October 1985, pp. 965–968.

[11] R. E. Ladner and M. J. Fisher, Parallel Prefix Computation, *Journal of the ACM,* Vol.27, No.4, 1980, pp. 831–838.

[12] S. Lakshmivarahan and S. K. Dhall, *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems,* McGraw-Hill Publishing Company, 1990.

[13] S. Lakshmivarahan and S. K. Dhall, *Parallel Computing Using the Prefix Problem,* Oxford University Press, 1994.

[14] E. L. Leiss, *Parallel and Vector Computing: A Practical Introduction,* McGraw-Hill, Inc., 1995.

[15] T. J. Rivlin, *Chebyshev Polynomials: from Approximation Theory to Algebra and Number Theory,* John Wiley & Sons, Inc., second edition, 1990.