

# Matrix Multiplication and Polynomial Operations

Advanced Algorithms (CSE 794)

Instructor: Tamal K. Dey

Scribe: Chaitanya Chitale \*

April 30, 2009

## 1 Introduction

Operations on matrices and polynomials are very important in scientific computing. Many algorithms working with these operations exist. Here, Strassen's algorithm for matrix multiplication is discussed. Coefficient and point-value representations of the polynomials and different operations on those representations are also discussed.

## 2 Matrix multiplication

We wish to find  $C = AB$ , where each of  $A, B$  and  $C$  are  $n \times n$  matrices. We assume that  $n$  is an exact power of 2.

### 2.1 Trivial algorithm

```
MATRIX-MULTIPLY( $A, B$ ) :  $C$ 
1.  $n \leftarrow \text{rows}[A]$ 
2. let  $C$  be an  $n \times n$  matrix
3. for  $i \leftarrow 1$  to  $n$ 
4.   do for  $j \leftarrow 1$  to  $n$ 
5.     do  $c_{ij} \leftarrow 0$ 
6.       for  $k \leftarrow 1$  to  $n$ 
7.         do  $c_{ij} \leftarrow c_{ij} + a_{ik} \times b_{kj}$ 
8. Return  $C$ 
```

The running time for above algorithm is  $\Theta(n^3)$ .

---

\*Department of Computer Science and Engineering, The Ohio State University, Columbus, Ohio, USA.

## 2.2 Strassen's algorithm

Can we improve on the above algorithm by using divide and conquer? We divide each of  $A, B$  and  $C$  into four  $n/2 \times n/2$  matrices. Thus,

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

and

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

Let  $C_{11} = A_{11}B_{11} + A_{12}B_{21}$ ,  $C_{12} = A_{11}B_{12} + A_{12}B_{22}$ ,  $C_{21} = A_{21}B_{11} + A_{22}B_{21}$ , and  $C_{22} = A_{21}B_{12} + A_{22}B_{22}$ . Each of the four terms of  $C$  requires two multiplications of  $n/2 \times n/2$  matrices and the addition of their  $n/2 \times n/2$  products. Thus the recurrence relation is:

$$T(n) = 8T(n/2) + \Theta(n^2).$$

The solution of the above recurrence relation is  $T(n) = \Theta(n^3)$  and the running time is same as that of the trivial algorithm. Strassen's algorithm requires only 7 multiplications of  $n/2 \times n/2$  matrices and  $\Theta(n^2)$  additions and subtractions of their  $n/2 \times n/2$  products. The recurrence relation is:

$$T(n) = 7T(n/2) + \Theta(n^2).$$

$$T(n) = O(n^{2.81}).$$

The idea is to form seven  $n/2 \times n/2$  matrices by 7 multiplications and additions and subtractions and then performing additions and subtractions on those 7 matrices to get the matrix  $C$ . The seven matrices are as follows:

$$P_1 = A_{11}(B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12})B_{22}$$

$$P_3 = (A_{21} + A_{22})B_{11}$$

$$P_4 = A_{22}(B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_6 = (A_{12} + A_{22})(B_{21} + B_{22})$$

$$P_7 = (A_{11} - A_{21})(B_{11} + B_{12}).$$

The individual  $n/2 \times n/2$  matrices of  $C$  can be calculated as:

$$C_{11} = P_5 + P_4 - P_2 + P_3$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7.$$

### 3 Polynomial operations

There are two types of polynomial representations. Coefficient representation and point-value representation.

#### 3.1 Coefficient representation

Any integer strictly greater than the degree of a polynomial is a degree-bound of that polynomial. A coefficient representation of a polynomial  $A(x) = \sum_{j=0}^{n-1} a_j x^j$  of degree-bound  $n$  is a vector of coefficients

$$a = (a_0, a_1, \dots, a_{n-1}).$$

#### Polynomial addition:

Let  $p(x)$  and  $q(x)$  be two polynomials of degree-bound  $n$  such that  $p(x) = \sum_{j=0}^{n-1} p_j x^j$  and  $q(x) = \sum_{n=0}^{n-1} q_j x^j$ . If

the degree-bound of the two polynomials is not same, then pad up zeros as the coefficients of the polynomial having lower degree-bound.

$$p(x) + q(x) = \sum_{i=0}^{n-1} (p_i + q_i) x^i.$$

This takes  $\Theta(n)$  time.

#### Polynomial multiplication:

Let  $p(x)$  and  $q(x)$  be two polynomials of degree-bounds  $n(p)$  and  $n(q)$  respectively such that

$$p(x) = \sum_{j=0}^{n(p)-1} p_j x^j \text{ and } q(x) = \sum_{n=0}^{n(q)-1} q_j x^j.$$

$$p(x) \times q(x) = \sum_{i=0}^{n(p)+n(q)-2} \left( \sum_{j=0}^i (p_j + q_{i-j}) \right) x^i.$$

The running time of the algorithm described below is  $\Theta(n^2)$ . We can improve the running time by applying divide and conquer strategy.

POLY-MULTIPLY( $p, q$ ) :  $r$

1. for  $i \leftarrow n(p)$  to  $n(p) + n(q) - 2$  do  $p_i = 0$
2. for  $i \leftarrow n(q)$  to  $n(p) + n(q) - 2$  do  $q_i = 0$
3. for  $i \leftarrow 0$  to  $n(p) + n(q) - 2$
4.     do  $r_i = 0$
5.         for  $j \leftarrow 0$  to  $i$
6.             do  $r_i = r_i + (p_j \times q_{i-j})$
7. Return  $r$

### Polynomial multiplication with divide and conquer:

The method is demonstrated on polynomials of degree-bound 2 and then it will be generalized. Let  $p(x) = p_0 + p_1x$  and  $q(x) = q_0 + q_1x$ . Number of multiplications required to compute  $p(x) \times q(x)$  using above algorithm is 4.

$$p(x) \times q(x) = (p_0 + p_1x) \times (q_0 + q_1x) = A + Bx + cX^2$$

$$A = p_0q_0, C = p_1q_1$$

$$B = p_0q_1 + p_1q_0 = (p_0 + p_1)(q_0 + q_1) - A - C.$$

Number of multiplications have been reduced to 3. This method can be generalized to polynomials of degree-bound  $n$  as follows. Let  $p(x)$  and  $q(x)$  be two polynomials of degree-bound  $n$  such that  $p(x) = \sum_{j=0}^{n-1} p_jx^j$  and  $q(x) = \sum_{j=0}^{n-1} q_jx^j$ . Assume that  $n$  is an exact power of 2.

Define:

$$a(x) = p_0 + p_1x + \dots + p_{n/2-1}x^{n/2-1}$$

$$b(x) = p_{n/2} + p_{n/2+1}x + \dots + p_{n-1}x^{n/2-1}$$

$$c(x) = q_0 + q_1x + \dots + q_{n/2-1}x^{n/2-1}$$

$$d(x) = q_{n/2} + q_{n/2+1}x + \dots + q_{n-1}x^{n/2-1}$$

Then:

$$p(x) = a(x) + x_{n/2}b(x)$$

$$q(x) = c(x) + x_{n/2}d(x)$$

$$r(x) = p(x)q(x) = A(x) + B(x)x^{n/2} + C(c)x^n$$

Substituting values of  $p(x)$  and  $q(x)$ ,

$$A(x) + B(x)x^{n/2} + C(x)x^n = (a(x) + x_{n/2}b(x))(c(x) + x_{n/2}d(x))$$

$$A(x) = a(x)c(x), C(x) = b(x)d(x)$$

$$B(x) = (a(x) + b(x))(c(x) + d(x)) - A(x) - c(x).$$

This method involves 3 multiplications of polynomials of degree bound  $n/2$  along with additions and subtractions. The recurrence relation is

$$T(n) = 3T(n/2) + \Theta(n)$$

$$T(n) = O(n^{1.58})$$

### Polynomial evaluation:

Let  $p(x)$  be a polynomial of degree-bound  $n$  such that  $p(x) = \sum_{j=0}^{n-1} p_j x^j$ . We want to evaluate value of  $p(x)$  for some value of  $x$ . If we evaluate the polynomial by computing each term separately, it will take  $\Theta(n^2)$  time. But using Horner's rule:

$$p(x_0) = p_0 + x_0(p_1 + x_0(a_2 + \dots + x_0(p_{n-2} + x_0(p_{n-1}))))$$

evaluation takes  $\Theta(n)$  time as in the algorithm described below.

POLY-EVALUATION( $p, x_0$ ) :  $y_0$

1.  $y_0 = 0$
2. for  $i \leftarrow n - 1$  down to 0
3.   do  $y_0 = y_0 x_0 + p_i$
4. Return  $y_0$

### 3.2 Point-value representation

A polynomial  $p$  with degree-bound  $n$  can be represented by  $n$  point-value pairs  $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$  where  $y_i = p(x_i)$  and  $x_i \neq x_j$  for  $i \neq j$ .

**Theorem 1.** For any set  $\{(x_i, y_i) \mid 0 \leq i \leq n - 1 \text{ and } x_i \neq x_j \text{ for } j \neq i\}$  there is an unique polynomial of degree  $n-1$  or less such that  $y_i = p(x_i)$ .

*Proof.* Suppose that such a polynomial  $p$  of degree-bound  $n$  exists and has coefficients of the form  $p_0, p_1, \dots, p_{n-1}$ . Then the following equation must hold and must have an unique solution,

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdot & \cdot & \cdot & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdot & \cdot & \cdot & x_1^{n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & x_{n-1} & x_{n-1}^2 & \cdot & \cdot & \cdot & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_{n-1} \end{pmatrix}$$

The matrix on the left is denoted as  $V(x_0, x_1, \dots, x_{n-1})$  and is called Vandermonde matrix. This matrix has determinant

$$\prod_{0 \leq j < k \leq n-1} (x_k - x_j).$$

This determinant is always not zero since  $x_k$  and  $x_j$  are distinct. Thus the Vandermonde matrix is invertible and the above equation can be rewritten as

$$\begin{pmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdot & \cdot & \cdot & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdot & \cdot & \cdot & x_1^{n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & x_{n-1} & x_{n-1}^2 & \cdot & \cdot & \cdot & x_{n-1}^{n-1} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_{n-1} \end{pmatrix}$$

Thus, there exists an unique set of coefficients for a given point-value representation.  $\square$

### Polynomial addition:

If we have point-value representation for for  $p$ ,

$$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

and for  $q$ ,

$$\{(x_0, y'_0), (x_1, y'_1), \dots, (x_{n-1}, y'_{n-1})\}$$

( $p$  and  $q$  are evaluated at same points) then point-value representation for  $r$ , where  $r(x) = p(x) + q(x)$ , is,

$$\{(x_0, y_0 + y'_0), (x_1, y_1 + y'_1), \dots, (x_{n-1}, y_{n-1} + y'_{n-1})\}$$

Thus, time required to add two polynomials in point-value form is  $\Theta(n)$

### Polynomial multiplication:

The point-value form is convinient for multiplying polynomials. If  $r(x) = p(x) \times q(x)$  then,  $r(x_k) = p(x_k)q(x_k)$ . If  $p$  and  $q$  have the degree-bound  $n$ , the degree-bound of  $r$  is  $2n - 1$ . So we need  $2n - 1$  points for a point-value representation of  $r$ . Thus, we need  $2n - 1$  point value pairs for  $p$  and  $q$ . Thus, time required to multiply two polynomials in point-value form is  $\Theta(n)$

### 3.3 Converting from coefficient representation into point-value representation

Converting from coefficient representation into point-value representation is straightforward. Just select  $n$  distinct points, where  $n$  is the degree-bound of the polynomial, and evaluate value of the polynomial at those points to get the point-value representation. Evaluation of polynomial at one point takes  $\Theta(n)$  time. Thus, evaluation of polynomial at  $n$  points takes  $\Theta(n^2)$  time.

### 3.4 Converting from point-value representation into coefficient representation

This is also called as interpolation. It can be done by solving the above equation involving the inverse of the Vandermonde matrix. The equation can be solved in  $O(n^3)$  time. A faster algorithm is based on Lagrange's formula:

$$p(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)}$$

Let,

$$a = \prod_j (x - x_j),$$

$a$  can be computed in  $\Theta(n^2)$  time. Then,

$$p(x) = \sum_{k=0}^{n-1} y_k \frac{a/(x - x_k)}{\prod_{j \neq k} (x_k - x_j)}.$$

Also,

$$\prod_{j \neq k} (x_k - x_j)$$

can be computed in  $\Theta(n)$  time for a given  $k$ .  $a/(x - x_k)$  can be computed in  $\Theta(n)$  time by using synthetic division. Thus, the time required to compute coefficients of  $p$  by using Lagrange's formula is  $\Theta(n^2)$ . If we choose  $n^{\text{th}}$  complex roots of unity for the evaluation and interpolation then the conversion takes  $\Theta(n \lg n)$  time. This is the basis of DFT and FFT.

## References

- [1] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. ISBN 0-262-53196-8.