CSE548 & AMS542: Analysis of Algorithms, Fall 2017 **Lecturer:** Rezaul A. Chowdhury
Topic: Matrix and polynomial multiplication **Date:** 11 September, 2017
Scribe: Shalini Bhaskara

5.1 Iterative Matrix Multiplication

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$$

$$\begin{bmatrix} Z_{11} & Z_{12} & \dots & Z_{1n} \\ Z_{21} & Z_{22} & \dots & Z_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{n1} & Z_{n2} & \dots & Z_{nn} \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1n} \\ X_{21} & X_{22} & \dots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \dots & X_{nn} \end{bmatrix} \times \begin{bmatrix} Y_{11} & Y_{12} & \dots & Y_{1n} \\ Y_{21} & Y_{22} & \dots & Y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{n1} & Y_{n2} & \dots & Y_{nn} \end{bmatrix}$$

If X, Y, Z are $n \times n$ matrices where n is a positive integer, *Iter-MM*(Z, X, Y) can be defined as:

Algorithm 1 *Iter-MM*(Z, X, Y)

```
1: for i ← 1 to n do
2:     for j ← 1 to n do
3:         Z[i][j] ← 0
4:         for k ← 1 to n do
5:             Z[i][j] ← Z[i][j] + X[i][k].Y[k][j]
```

Traditionally, this approach will have a complexity of $O(n^3)$ as there are three indented *for* loops. Also note from our earlier classes that this approach is likely to incur a lot of cache misses when we multiply each horizontal row of X with each vertical row of Y. As we go further, we will try to reduce the complexity of multiplying two large matrices by a small extent and work towards incurring fewer cache misses.

5.2 Recursive (Divide & Conquer) Matrix Multiplication

With this new algorithm we will try to use the concept of *Divide & Conquer* to see if we can reduce the complexity of the earlier approach and simultaneously decrease the number of cache misses, if possible. Here, we first divide the matrices Z, X, Y into $\frac{n}{2} \times \frac{n}{2}$ matrices each and figure out the effective way of solving the problem.

$$\begin{bmatrix} Z_{11} & \vdots & Z_{12} \\ \dots & \dots & \dots \\ Z_{21} & \vdots & Z_{22} \end{bmatrix} = \begin{bmatrix} X_{11} & \vdots & X_{12} \\ \dots & \dots & \dots \\ X_{21} & \vdots & X_{22} \end{bmatrix} \times \begin{bmatrix} Y_{11} & \vdots & Y_{12} \\ \dots & \dots & \dots \\ Y_{21} & \vdots & Y_{22} \end{bmatrix}$$

$$= \begin{bmatrix} X_{11}Y_{11} & + & X_{12}Y_{21} & \vdots & X_{11}Y_{12} & + & X_{12}Y_{22} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ X_{21}Y_{11} & + & X_{22}Y_{21} & \vdots & X_{21}Y_{12} & + & X_{22}Y_{22} \end{bmatrix}$$

Notice that we have divided the $n \times n$ matrices to 4 $\frac{n}{2} \times \frac{n}{2}$ sizes each. On resolving these matrices, we have to deal with 8 products of smaller size i.e $X_{11}Y_{11}, X_{12}Y_{21}, X_{11}Y_{12}, X_{12}Y_{22}, X_{21}Y_{11}, X_{22}Y_{21}, X_{21}Y_{12}, X_{22}Y_{22}$ and the summations of these terms as shown in the matrix above. If we keep reducing the size of these $\frac{n}{2} \times \frac{n}{2}$ matrices further down to a 1×1 matrix, we have to ultimately multiply just two numbers. Let us write down the algorithm for this.

Algorithm 2 *Rec-MM*(X, Y)

1: Let Z be a new $n \times n$ matrix
2: ***if*** n = 1 then
3: $Z \leftarrow$ X.Y
4: ***else***
5: $Z_{11} \leftarrow$ ***Rec-MM***$(X_{11}, Y_{11}) +$ ***Rec-MM***(X_{12}, Y_{21})
6: $Z_{12} \leftarrow$ ***Rec-MM***$(X_{11}, Y_{12}) +$ ***Rec-MM***(X_{12}, Y_{22})
7: $Z_{21} \leftarrow$ ***Rec-MM***$(X_{21}, Y_{11}) +$ ***Rec-MM***(X_{22}, Y_{21})
8: $Z_{22} \leftarrow$ ***Rec-MM***$(X_{21}, Y_{12}) +$ ***Rec-MM***(X_{22}, Y_{22})
9: ***return*** Z

$$\begin{aligned} T(n) &= \begin{cases} \Theta(1) & \text{if n=1.} \\ 8T\left(\frac{n}{2}\right) + \Theta(n^2) & \text{otherwise.} \end{cases} \\ &= \Theta(n^3) \end{aligned} \tag{5.2.1}$$

Calculating the complexity of such an approach, if we consider complexity of multiplying two $n \times n$ as T(n) then we get the corresponding complexity of $8T(\frac{n}{2})$ for the multiplication of 8 smaller $\frac{n}{2}$ matrices. As we know that the product of two $\frac{n}{2} \times \frac{n}{2}$ matrices will have $\frac{n^2}{4}$ entries, the summation of two such matrix products will result in $\Theta(n^2)$ complexity. Therefore, adding these two up, the total complexity still boils down to be $\Theta(n^3)$ so asymptotically we have not achieved anything. Also, we will later figure out that when this approach is used on asymptotic numbers it incurs fewer cache misses, improves reuse and will end up running 10X times faster than the previous traditional approach. The only overhead with this approach is that it is based on recursion and therefore the compiler would not know how to optimize it.

5.3 Strassens Algorithms for Matrix Multiplication

In 1968 Volker Strassen came up with a recursive MM algorithm that runs asymptotically faster than the classical $\Theta(n^3)$ algorithm. In each level of recursion the algorithm uses 7 recursive matrix multiplications

(instead to 8) and 18 matrix additions (instead of 4). Using Strassens algorithm on matrix multiplication, we get the following,

Additions:

We first divide the X and Y matrices into 4 smaller quadrants as we did earlier. Then separately add adjacent smaller quadrants and subtract the quadrants on the vertical axis in the following manner.

$$X_{r1} = X_{11} + X_{12} \qquad Y_{r1} = Y_{11} + Y_{12}$$
$$X_{r2} = X_{21} + X_{22} \qquad Y_{r2} = Y_{21} + Y_{22}$$
$$X_{c1} = X_{11} - X_{21} \qquad Y_{c1} = Y_{11} - Y_{21}$$
$$X_{c2} = X_{12} - X_{22} \qquad Y_{c2} = Y_{12} - Y_{22}$$
$$X_{d1} = X_{11} + X_{22} \qquad Y_{d1} = Y_{11} + Y_{22}$$

Products:

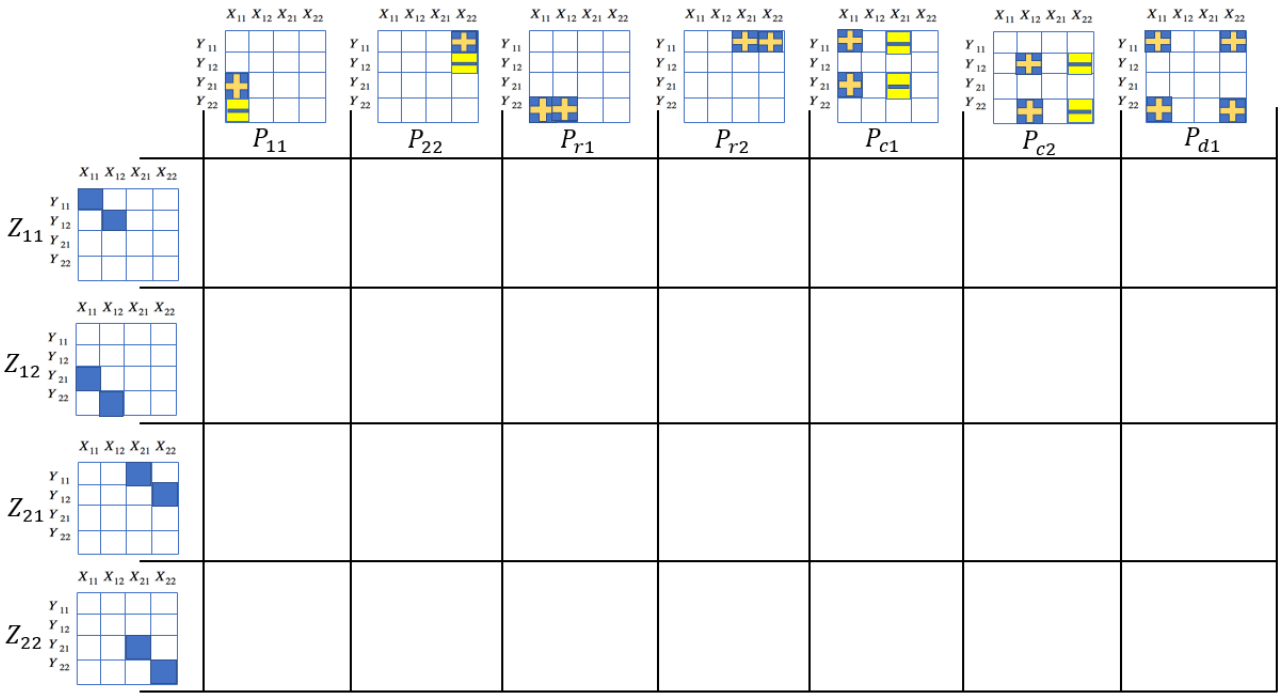
Using the above equations we deduce the following equations,

$$P_{r1} = (X_{11} + X_{12}) + Y_{22}$$
$$P_{r2} = (X_{21} + X_{22}) + Y_{11}$$
$$P_{c1} = (X_{11} - X_{21}) - (Y_{11} + Y_{12})$$
$$P_{c2} = (X_{12} - X_{22}) - (Y_{21} + Y_{22})$$
$$P_{d1} = (X_{11} + X_{22}) + (Y_{11} + Y_{22})$$

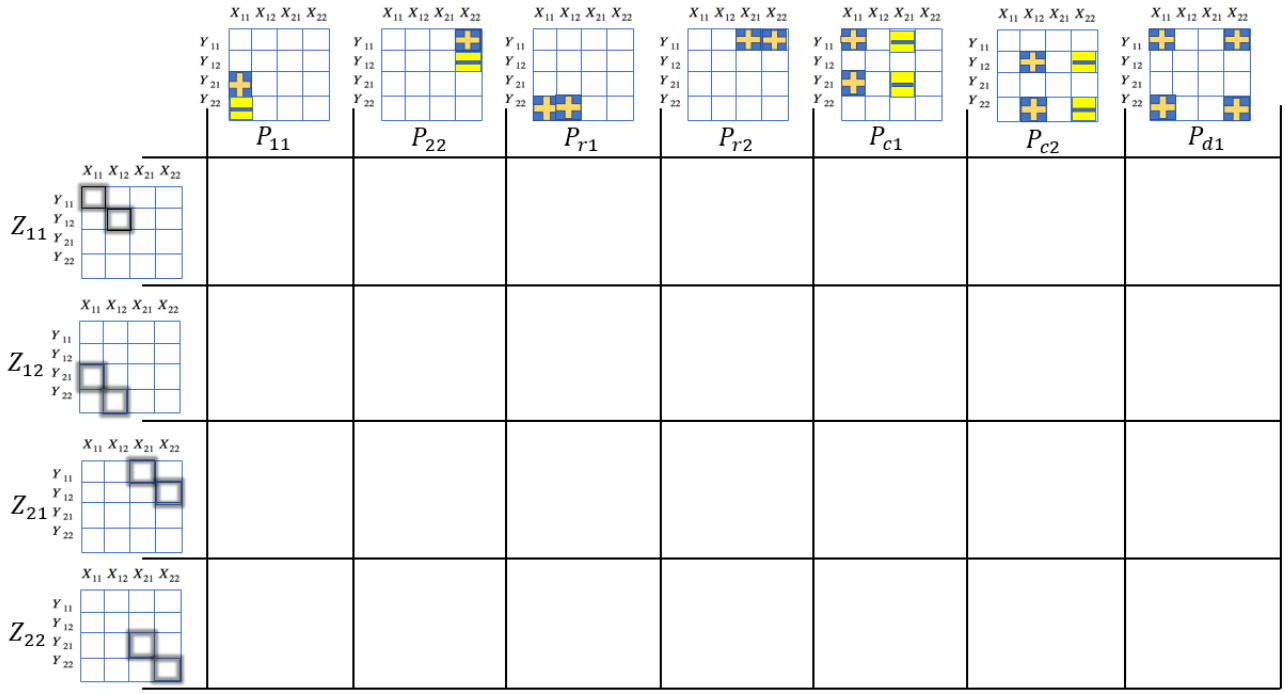
$$P_{11} = X_{11} + (Y_{12} - Y_{22})$$
$$P_{22} = X_{22} + (Y_{11} - Y_{21})$$

Well, how did we derive it? Oh, it's easy, just think real hard!

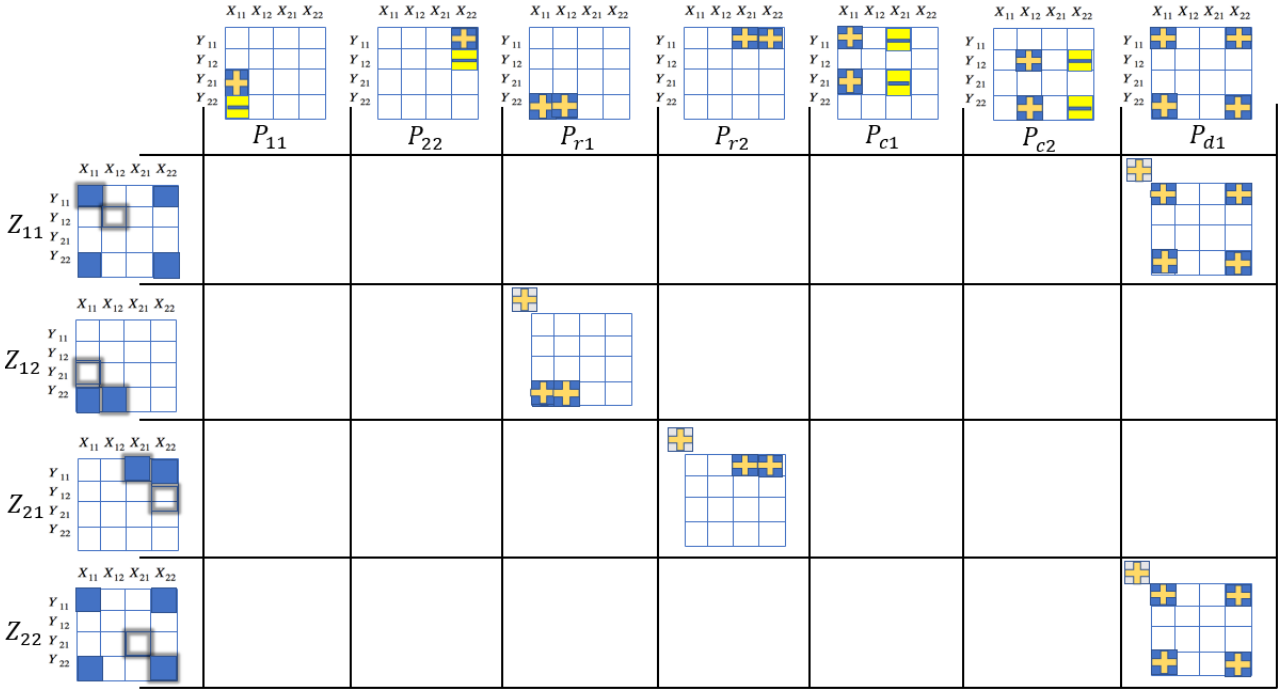
Didn't get it still? Alright, lets dive into it.



Here, we have taken a grid with resultant matrices $Z_{11}, Z_{12}, Z_{21}, Z_{22}$ on the y axis and the products we have derived before on the x axis. The aim is to somehow fill up the blocks in the resultant matrices with the '+' and '-' variants of the products on x-axis.

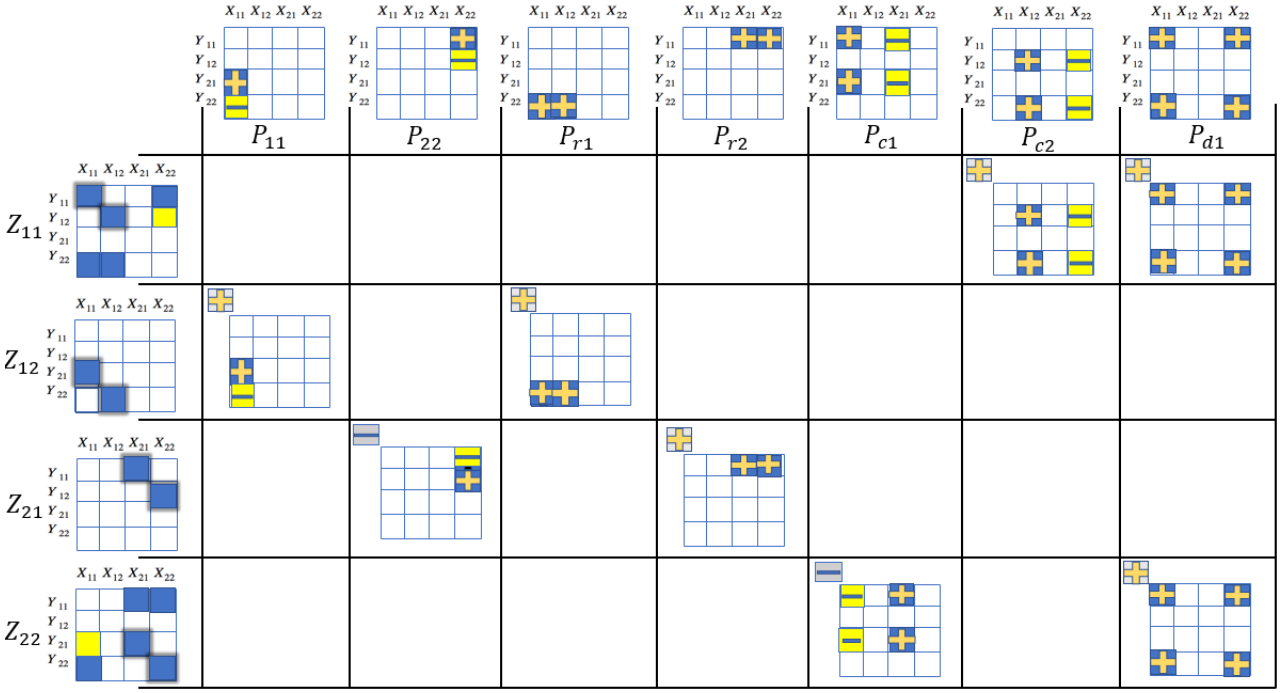


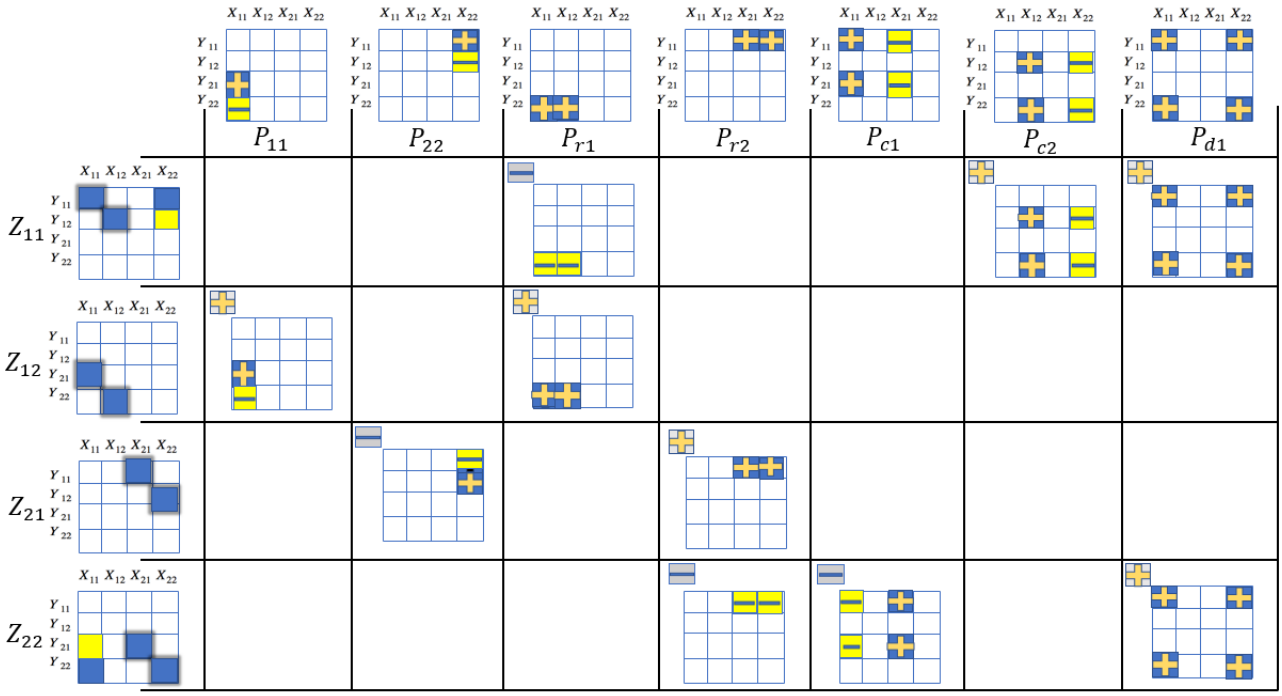
In the above image, the blocks of the resultant matrices to be filled have been highlighted.



Here we have added P_{d1} to calculate Z_{11}, P_{r1} to calculate Z_{12}, P_{r2} to calculate Z_{21} and P_{d1} to calculate

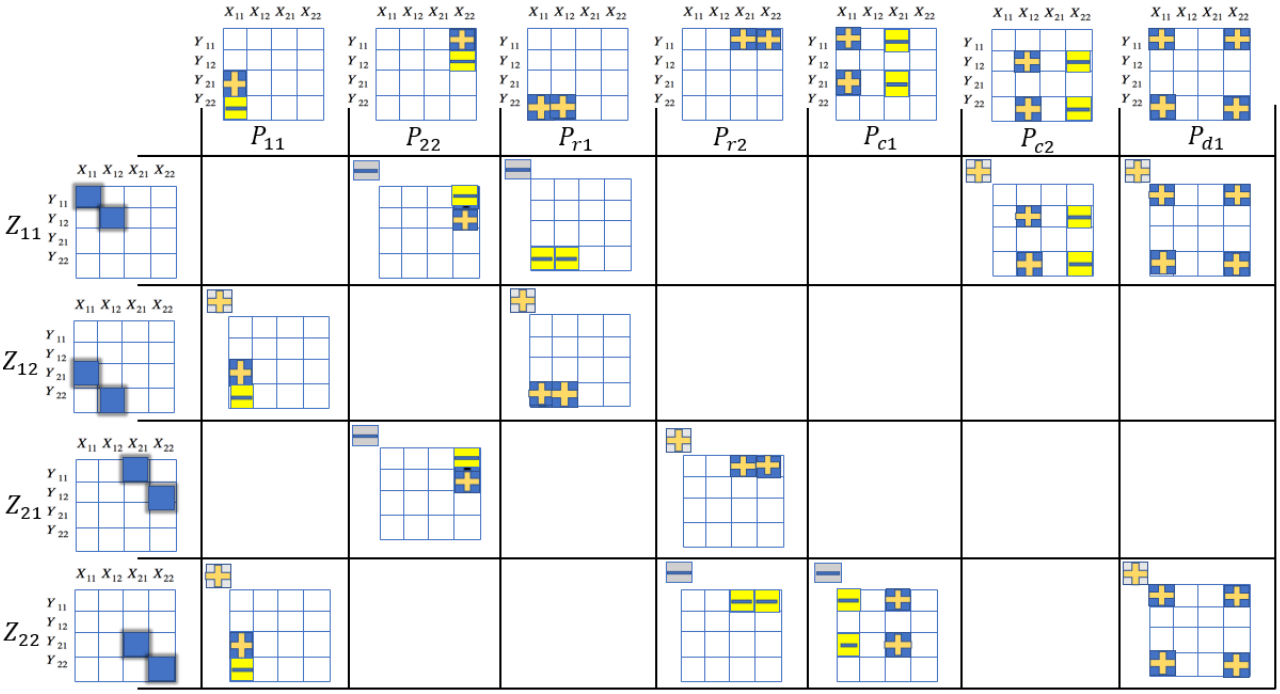
Z_{22}. The y-axis now portrays impressions of those columns added to it.





Here we have added P_{c2} to calculate Z_{11}, P_{11} to calculate Z_{12}, P_{22} to calculate Z_{21} and P_{c1} to calculate

Z_{22}, P_{r2} to calculate Z_{22} and P_{r1} to calculate Z_{11} in the above images.



We finally arrived at the desired results we aimed for. The permutations on the horizontal axes match those of the resultant matrices. This is how we came up with the formula for Strassen's algorithm for matrix multiplication on asymptotic bounds.

To calculate running time for this approach, we need to multiply 7 $\frac{n}{2} \times \frac{n}{2}$ matrices which account for $T\left(\frac{n}{2}\right)$ complexity and 18 summations which account for $\Theta(n^2)$ complexity like the previous approach.

$$T(n) = \begin{cases} \Theta(1) & \text{if n=1.} \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2) & \text{otherwise.} \end{cases} \tag{5.3.2}$$
$$= \Theta(n^{log_2 7}) = \Theta(n^{2.81})$$

This finally comes upto $\Theta(n^{2.81})$ which is slightly lesser than $\Theta(n^3)$ from our earlier approaches and also has the added advantages of fewer number of cache misses. Mission Accomplished!

Now lets go on to polynomial multiplication and understand the complexities we face there.

5.4 Polynomial Multiplication

Coefficient Representation of Polynomials

$$A(x) = \sum_{k=0}^{n-1} a_j x^j$$
$$= a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1}$$

A(x) is a polynomial of degree bound n represented as a vector $a = (a_0, a_1, ..., a_{n-1})$ of coefficients.
The degree of A(x) is k provided it is the largest integer such that a_k is nonzero. Clearly, $0 \leq k \leq n-1$.

Evaluating A(x) at a given point:
Takes $\Theta(n)$ time using Horner's rule:

$$A(x_0) = a_0 + a_1 x_0 + a_2 {x_0}^2 + ... + a_{n-1} {x_0}^{n-1}$$
$$= a_0 + x_0(a_1 + x_0(a_2 + ... + x_0(a_{n-2} + x_0(a_{n-1}))...))$$

For instance, if we take an example of the polynomial:

$$2x^3 - 6x^2 + 2x - 1$$

To evaluate the value of this polynomial at x=3 in $\Theta(n)$ time, we write it in the format specified by Horner's rule :

$$-1 + x(2 - x(6 - x(2)))$$
$$= -1 + x(2 - x(6 - 2 * 3))$$
$$= -1 + x(2)$$
$$= -1 + 3 * 2$$
$$= 5$$

5.5 Adding two polynomials

Adding two polynomials of degree bound n takes $\Theta(n)$ time.

$$C(x) = A(x) + B(x)$$

where $A(x) = \sum_{j=0}^{n-1} a_j x^j$ and $B(x) = \sum_{j=0}^{n-1} b_j x^j$.
Then $C(x) = \sum_{j=0}^{n-1} c_j x^j$, where $c_j = a_j + b_j$ for $0 \leq j \leq n-1$.

5.6 Multiplying two polynomials

The product of two polynomials of degree bound *n* is another polynomial of degree bound 2n - 1.

$$C(x) = A(x)B(x)$$

where $A(x) = \sum_{j=0}^{n-1} a_j x^j$ and $B(x) = \sum_{j=0}^{n-1} b_j x^j$.
Then $C(x) = \sum_{j=0}^{2n-2} c_j x^j$, where $c_j = \sum_{k=0}^{j} a_k b_{j-k}$ for $0 \leq j \leq 2n-2$.

The coefficient vector $c = (c_0, c_1, ..., c_{2n-2})$, denoted by $c = a \times b$ is also called the convolution of vectors $a = (a_0, a_1, ..., a_{n-1})$ and $b = (b_0, b_1, ..., b_{n-1})$.

To explain it further, let us take a small example:

$$(x^2 + 2x + 3) \times (1 + 2x + 3x^2)$$

Start from the constant terms in the equations :- $a_0 * b_0$ = 1*3 = 3
Next, proceed to the coefficients of x :- $a_0 b_1 x + a_1 b_0 x$ = 1*2x + 3*2x = 8x
Similarly for rest of the coefficients..
:- $a_0 b_2 x^2 + a_1 b_1 x^2 + a_2 b_0 x^2 = 1 * 1x^2 + 2 * 2x^2 + 3 * 3x^2 = 14x^2$
:- $a_1 b_2 x^3 + a_2 b_1 x^3 = 2 * 1x^3 + 3 * 2x^3 = 8x^3$
:- $a_2 b_2 x^4 = 3 * 1x^4 = 3x^4$

Therefore, the final result is:

$$3x^4 + 8x^3 + 14x^2 + 8x + 3$$

Clearly, this approach to multiplication takes $\Theta(n^2)$ time. We will discuss about an efficient approach to multiplication using Karatsuba's Algorithm in the next class!
Class ends for now!!

Thank you.