Divide and Conquer

Strassen Matrix Multiplication

Divide and Conquer

- An important general technique for designing algorithms:
  - divide problem into subproblems
  - recursively solve subproblems
  - combine solutions to subproblems to get solution to original problem
- Use recurrences to analyze the running time of such algorithms

Additional D&C Algorithms

- binary search
  - divide sequence into two halves by comparing search key to midpoint
  - recursively search in one of the two halves
  - combine step is empty
- quicksort
  - divide sequence into two parts by comparing pivot to each key
  - recursively sort the two parts
  - combine step is empty

Additional D&C applications

- computational geometry
  - finding closest pair of points
  - finding convex hull
- mathematical calculations
  - converting binary to decimal
  - integer multiplication
  - matrix multiplication
  - matrix inversion
  - Fast Fourier Transform

Strassen’s Matrix Multiplication

Matrix Multiplication

- Consider two n by n matrices A and B
- Definition of AxB is n by n matrix C whose $(i,j)^{th}$ entry is computed like this:
  - consider row i of A and column j of B
  - multiply together the first entries of the row and column, the second entries, etc.
  - then add up all the products
- Number of scalar operations (multiplies and adds) in straightforward algorithm is **$O(n^3)$.**
- Can we do it faster?

Divide-and-Conquer

$$A \quad \times \quad B \quad = \quad C$$

$$\begin{bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{bmatrix} \times \begin{bmatrix} B_0 & B_1 \\ B_2 & B_3 \end{bmatrix} = \begin{bmatrix} \mathbf{A_0 \times B_0 + A_1 \times B_2} & \mathbf{A_0 \times B_1 + A_1 \times B_3} \\ \mathbf{A_2 \times B_0 + A_3 \times B_2} & \mathbf{A_2 \times B_1 + A_3 \times B_3} \end{bmatrix}$$

- Divide matrices A and B into four submatrices each
- We have 8 smaller matrix multiplications and 4 additions. Is it faster?

Divide-and-Conquer

Let us investigate this recursive version of the matrix multiplication.

Since we divide *A*, *B* and *C* into 4 submatrices each, we can compute the resulting matrix C by

- 8 matrix multiplications on the submatrices of *A* and *B*,
- plus $\Theta(n^2)$ scalar operations

Divide-and-Conquer

- Running time of recursive version of straightforward algorithm is

  $$T(n) = 8T(n/2) + \Theta(n^2)$$
  $$T(2) = \Theta(1)$$

  where $T(n)$ is running time on an n x n matrix

- Master theorem gives us:

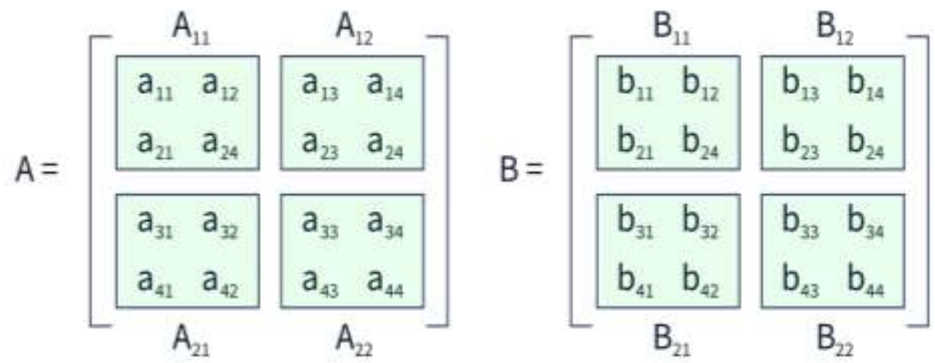
  $$T(n) = \Theta(n^3)$$

- Can we do fewer recursive calls (fewer multiplications of the $n/2$ x $n/2$ submatrices)?

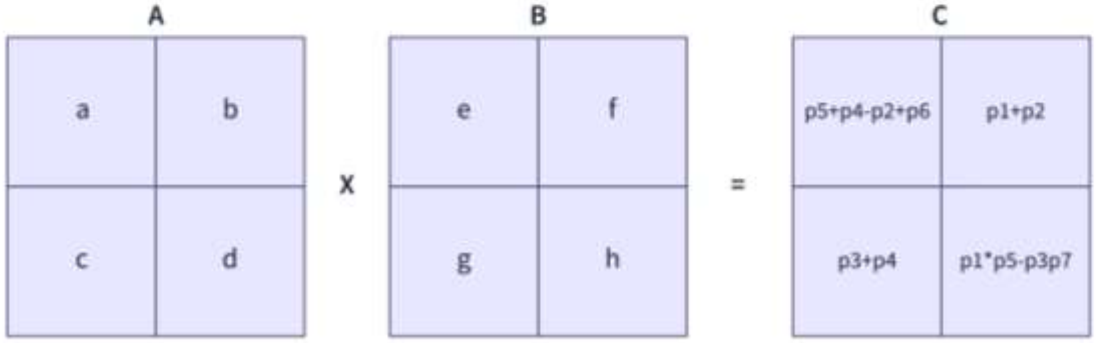
Algorithm:

Algorithm MatMul(A, B, n){
 if(n<=2) return
$C_{11} = a_{11}xb_{11} + a_{12}xb_{21}$
$C_{12} = a_{11}xb_{12} + a_{12}xb_{22}$
$C_{21} = a_{21}xb_{11} + a_{22}xb_{21}$
$C_{22} = a_{21}xb_{12} + a_{22}xb_{22}$
}

MatMul(A_{11}, B_{11}, n/2) + MatMul(A_{12}, B_{21}, n/2)
MatMul(A_{11}, B_{12}, n/2) + MatMul(A_{12}, B_{22}, n/2)
MatMul(A_{21}, B_{11}, n/2) + MatMul(A_{22}, B_{21}, n/2)
MatMul(A_{21}, B_{12}, n/2) + MatMul(A_{22}, B_{22}, n/2)



p1 = a(f-h)
p2 = (a+b)h
p3 = (c+d)e
p4 = d(g-e)
p5 = (a+d)(e+h)
p6 = (b-d) (g+h)
p7 = (a-c)(e+f)



Strassen's Matrix Multiplication

$$A \times B = C$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

Strassen's Matrix Multiplication

- Strassen found a way to get all the required information with only 7 matrix multiplications, instead of 8.

- Recurrence for new algorithm is

$$T(\mathrm{n}) = 7T(n/2) + \Theta(n^2)$$

Solving the Recurrence Relation

Applying the Master Theorem to

$T(n) = a\,T(n/b) + f(n)$

with a=7, b=2, and $f(n)=\Theta(n^2)$.

Since $f(n) = O(n^{\log_b(a)-\varepsilon}) = O(n^{\log_2(7)-\varepsilon})$,

case 1) applies and we get

$T(n)= \Theta(n^{\log_b(a)}) = \Theta(n^{\log_2(7)}) = O(n^{2.81})$.