
BITS Pilani
Pilani Campus

innovate achieve lead

CS F364 Design & Analysis of Algorithms

# DIVIDE-AND-CONQUER

## Example: Matrix Multiplication

### Square Division / Blocking : Time Complexity

### Strassen's algorithm

# Matrix Multiplication: Square Division

- Divide-and-conquer: Use a different division
  - Assume that A and B are square matrices each of size N*N
  - Divide A and B into four square sub-matrices (respectively), each of size N/2 * N/2
    - Then C = A*B can be computed as

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

    - where
      - $C_{11} = A_{11}B_{11} + A_{12}B_{21}$
      - $C_{12} = A_{11}B_{12} + A_{12}B_{22}$
      - $C_{21} = A_{21}B_{11} + A_{22}B_{21}$
      - $C_{22} = A_{21}B_{12} + A_{22}B_{22}$



# Matrix Multiplication: Square Division

- Exercise:
  - Write a recursive procedure for matrix multiplication based on the idea described in the previous slide.
    - Parameters of the recursive calls would notionally be sub-matrices
    - To reduce procedure call overhead i.e. to avoid carrying matrices around pass boundary indices around.
- Assume $N=2^k$ for some $k>0$, then the time complexity T of this procedure can be given as
  - $T(N) = b$ $N<=2$
  - $T(N) = 8*T(N/2) + c*N^2$ $N>2$
    - where b and c are constants.

Number of recursive calls

- Solving this relation gives you:
  - $T(N) = O(N^3)$
    - i.e. no better than linear division.

# Strassen's Matrix Multiplication

- Use the square division (of matrices A and B into sub-matrices of size N/2 * N/2) i.e.

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

but compute

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})\, B_{11}$$
$$R = A_{11}\,(B_{12} - B_{22})$$
$$S = A_{22}\,(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})\, B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

and then compute

$$C_{11} = P+S-T+V$$
$$C_{12} = R+T$$
$$C_{21} = Q+S$$
$$C_{11} = P+R-Q+U$$

This is referred to as the Strassen's algorithm for matrix multiplication.

# Strassen's Matrix Multiplication

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

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

$Q = (A_{21} + A_{22})\, B_{11}$

$R = A_{11}\, (B_{12} - B_{22})$

$S = A_{22}\, (B_{21} - B_{11})$

$T = (A_{11} + A_{12})\, B_{22}$

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

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

$C_{11} = P+S-T+V$

$C_{12} = R+T$

$C_{21} = Q+S$

$C_{11} = P+R-Q+U$

Strassen's algorithm requires:

- 7 multiplications (instead of 8) of matrices and
- several more additions

in each recursive step as compared to the classic algorithm.

# Strassen's Matrix Multiplication

- Exercise:
  - Prove that Strassen's method is correct.
- Exercise:
  - Write a recursive procedure based on Strassens' method.
- The time complexity of Strassen's method is given by:
  - $T(N) = b$ if $N<=2$
  - $T(N) = 7*T(N/2) + c*N^2$ otherwise
    - Solution (by master method): $O(n^{lg7})$
- Exercise:
  - Modify Strassen's method to handle matrices of size n x n where n is not a power of 2.

# Strassen's Matrix Multiplication

- Performance of Strassen's method:
  - Locality of reference:
    - Strassen's method accesses n/2 rows and n/2 columns of each of the two matrices at a time
    - The linear method accesses one row at a time of the first matrix (and the whole of the second matrix)
      - The latter usually provides faster access:
        - matrices are stored in row major (or column major) order
        - accessing elements in a sequence works better in a memory hierarchy.
- Exercise:
  - Implement the classic algorithm and Strassen's algorithm.
  - Evaluate their performance for different sizes of matrices and compare.