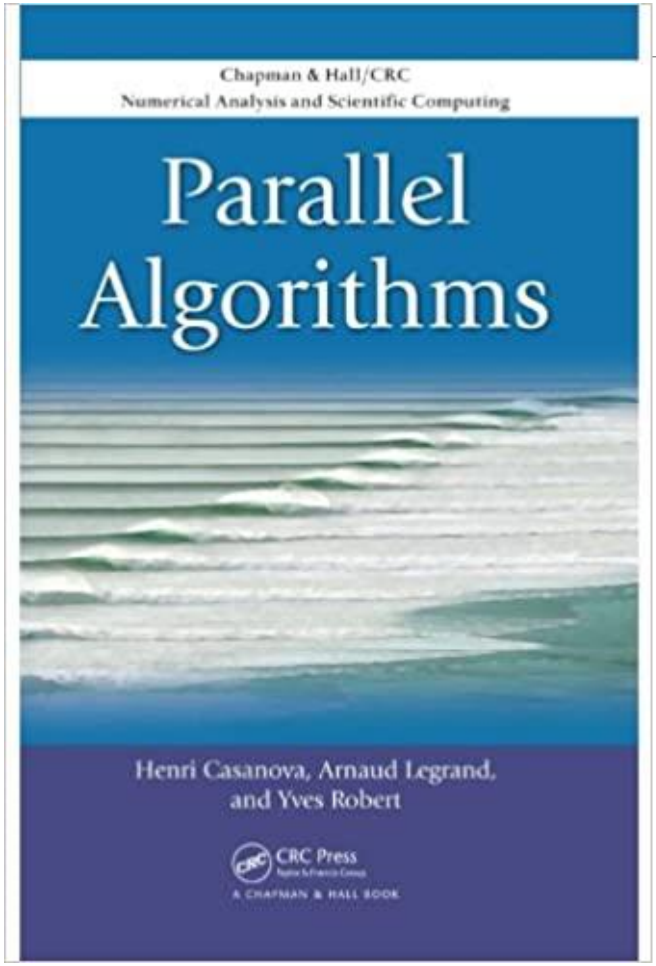
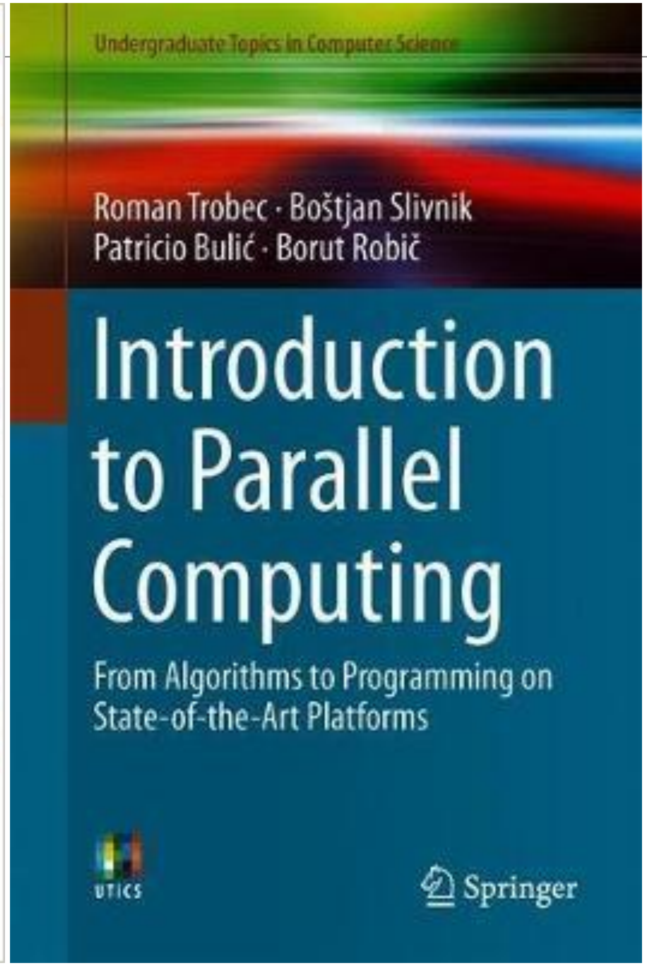
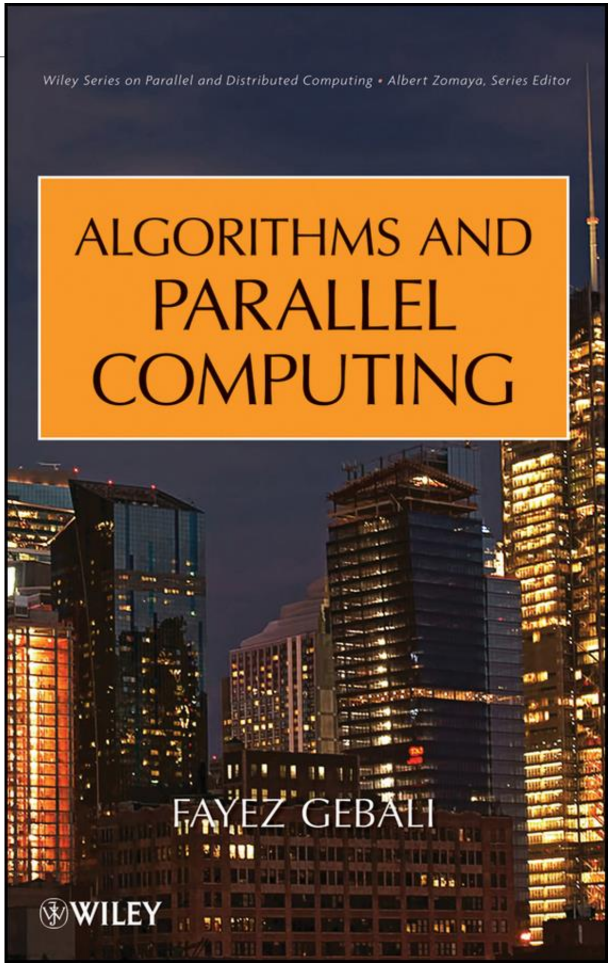
Parallel Programming

Lec 7

Books

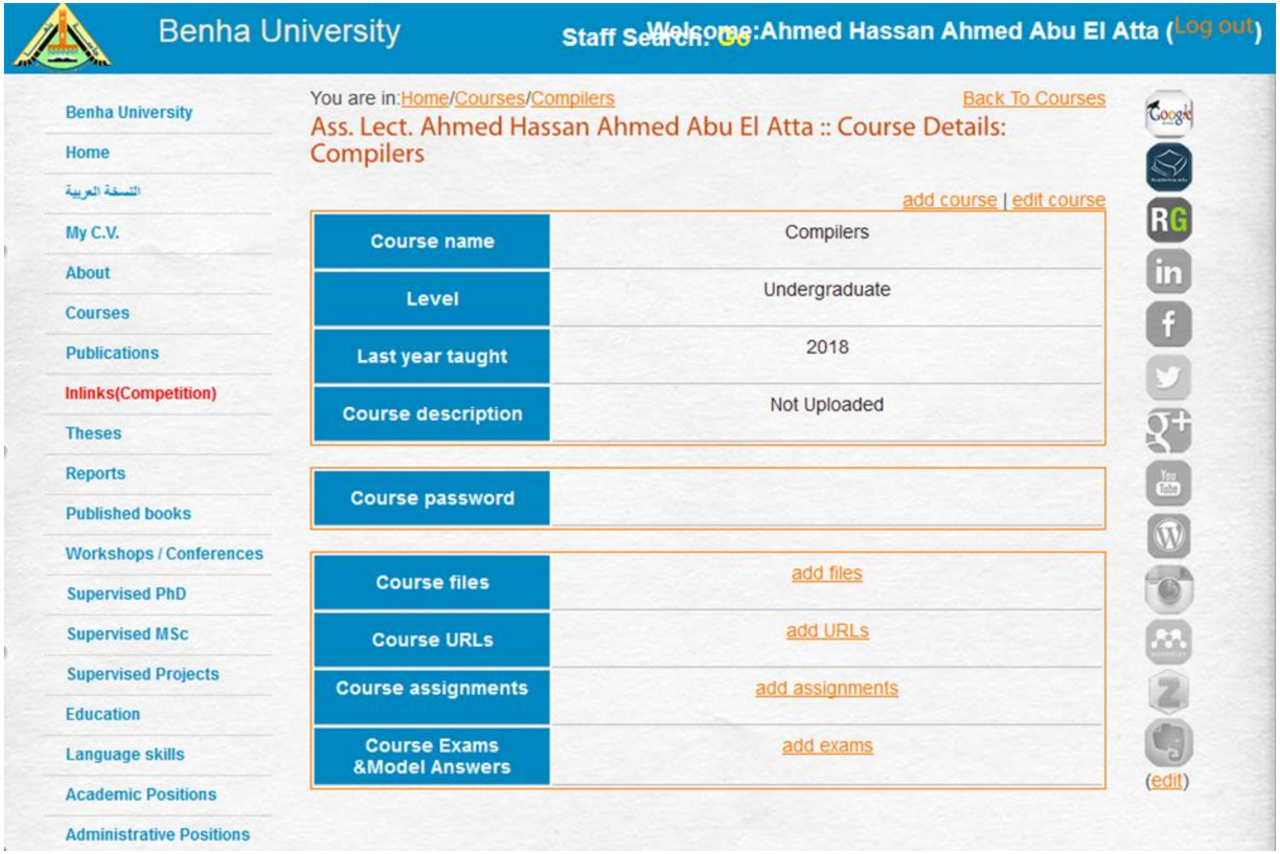

Chapman & Hall/CRC
Numerical Analysis and Scientific Computing
Parallel Algorithms
Henri Casanova, Arnaud Legrand, and Yves Robert
CRC Press
A CHAPMAN & HALL BOOK


Undergraduate Topics in Computer Science
Roman Trobec · Boštjan Slivnik
Patricio Bulić · Borut Robič
Introduction to Parallel Computing
From Algorithms to Programming on State-of-the-Art Platforms
UTICS
Springer


Wiley Series on Parallel and Distributed Computing • Albert Zomaya, Series Editor
ALGORITHMS AND PARALLEL COMPUTING
FAYEZ GEBALI
WILEY

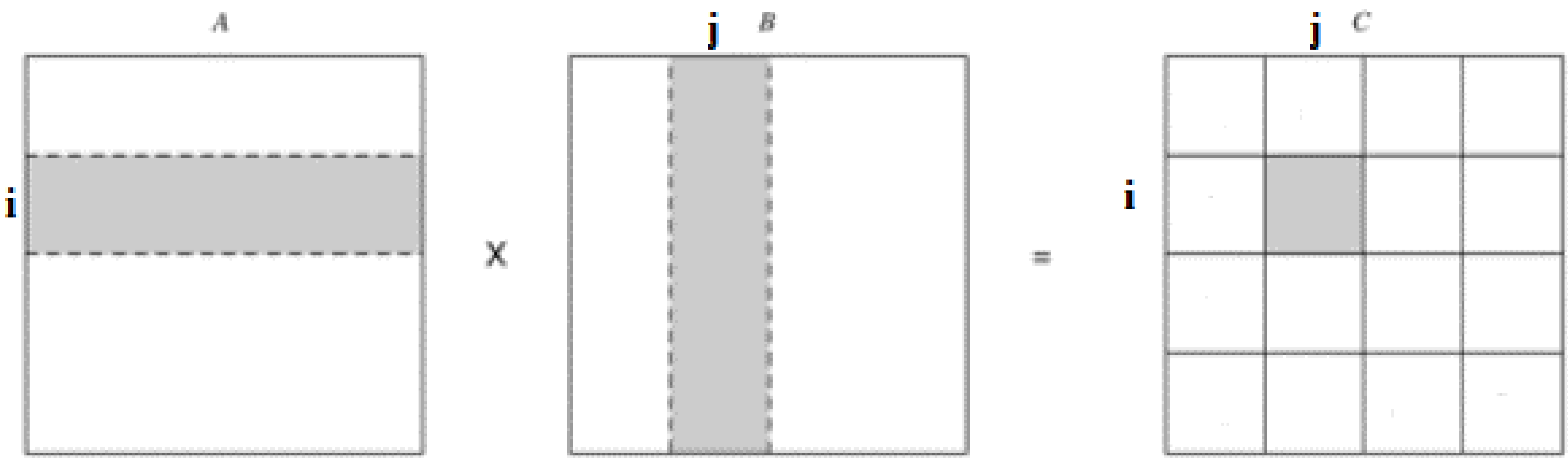
PowerPoint

http://www.bu.edu.eg/staff/ahmedaboalatah14-courses/14779



Matrix-Matrix Multiplication

Problem Statement



Problem Statement

The result of multiplying the matrix A of order m × r by matrix B of order r × n, is the matrix C of order m × n

C = A × B, is such that each of its elements is denoted ij with 0 ≤ i < m and 0 ≤ j < n, and is calculated follows

$$c_{ij} = \sum_{k=0}^{r-1} a_{ik} \times b_{kj}$$

Multiplying a square matrix by a square matrix (**Sequential algorithm**)

Input: Matrix A[n][n]

Matrix B[n][n]

Output: Matrix C[n][n]

```
for ( i = 0; i < n; i++ )
    for ( j = 0; j < n; j++ )
        C[i][j] = 0;
        for(k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Multiplying a square matrix by a square matrix (Sequential algorithm)

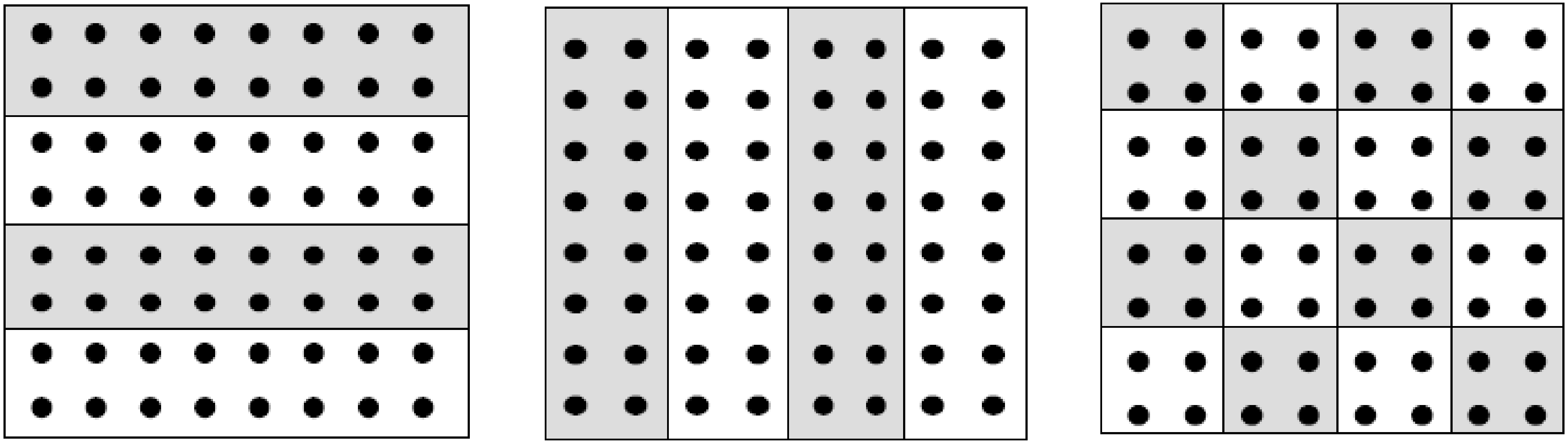
The number of operation required to multiply A x B is:

$$n \times n \times n$$

$T_s(n) = O(n^3)$

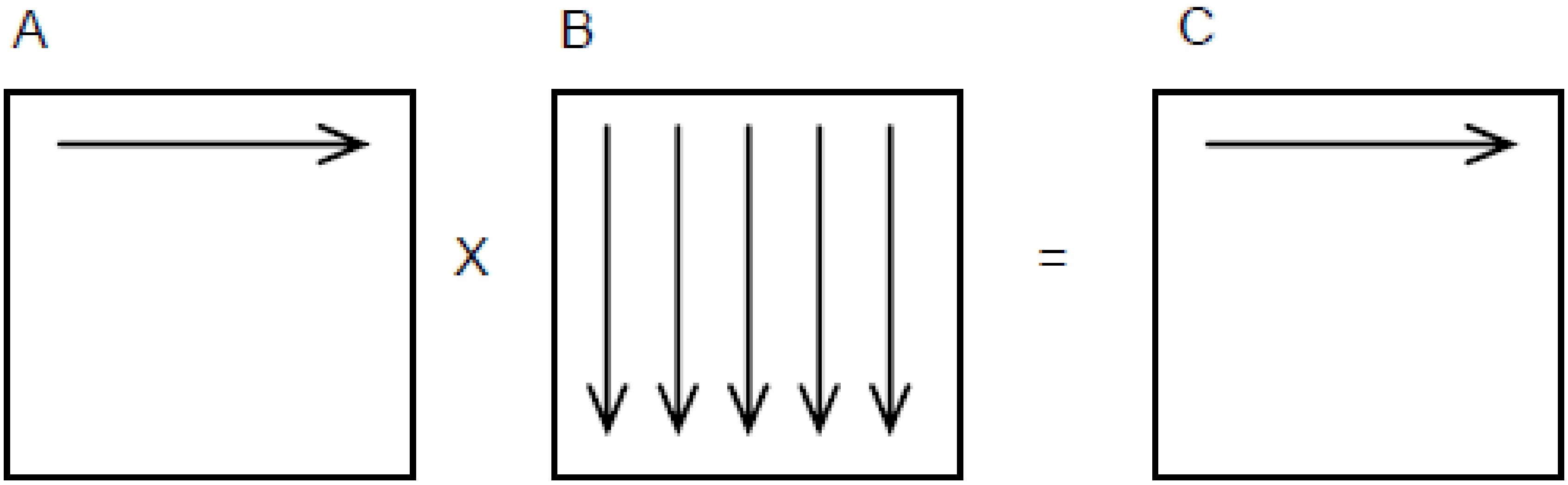
Parallel Methods for Matrix-Matrix Multiplication

Data Distribution



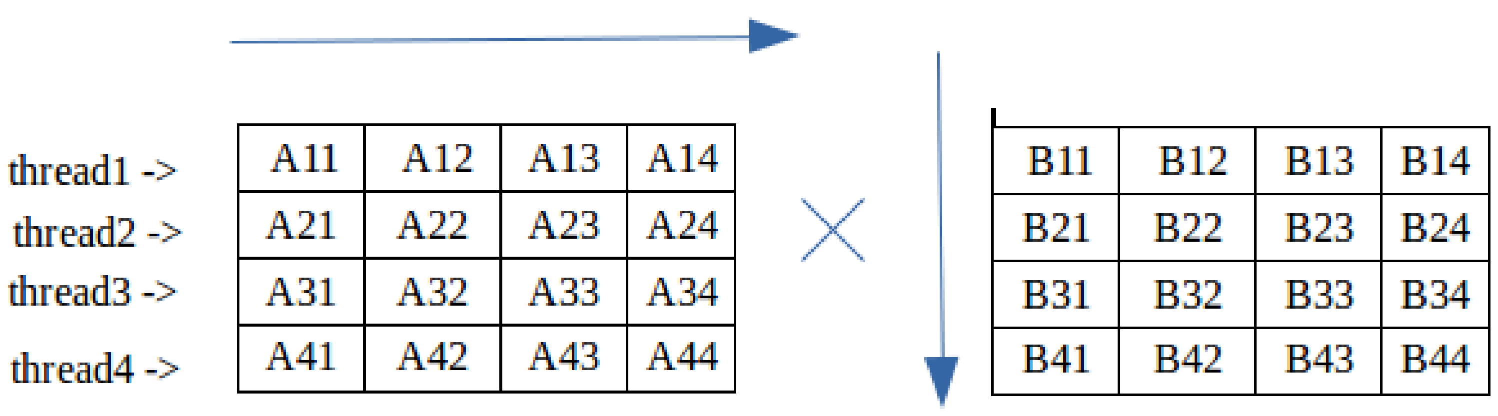
Matrix-Matrix Multiplication in Case of 1-Dim

Matrix-Matrix Multiplication in 1-Dim



Matrix-Matrix Multiplication in 1-Dim

The $C_{n \times n}$ matrix is partitioned among n processors, with each processor computes row of the matrix.



Matrix-Matrix Multiplication in 1-Dim Parallel Algorithm

Input:	Matrix A[n][n]
	Matrix B[n][n]
Output:	Matrix C[n][n]

```
for ( i = 0; i < n; i++ ) do in parallel
    for ( j = 0; j < n; j++ )
        C[i][j] = 0;
        for(k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Matrix-Matrix Multiplication in 1-Dim Parallel Algorithm

$T_p(n) = O(n^2)$

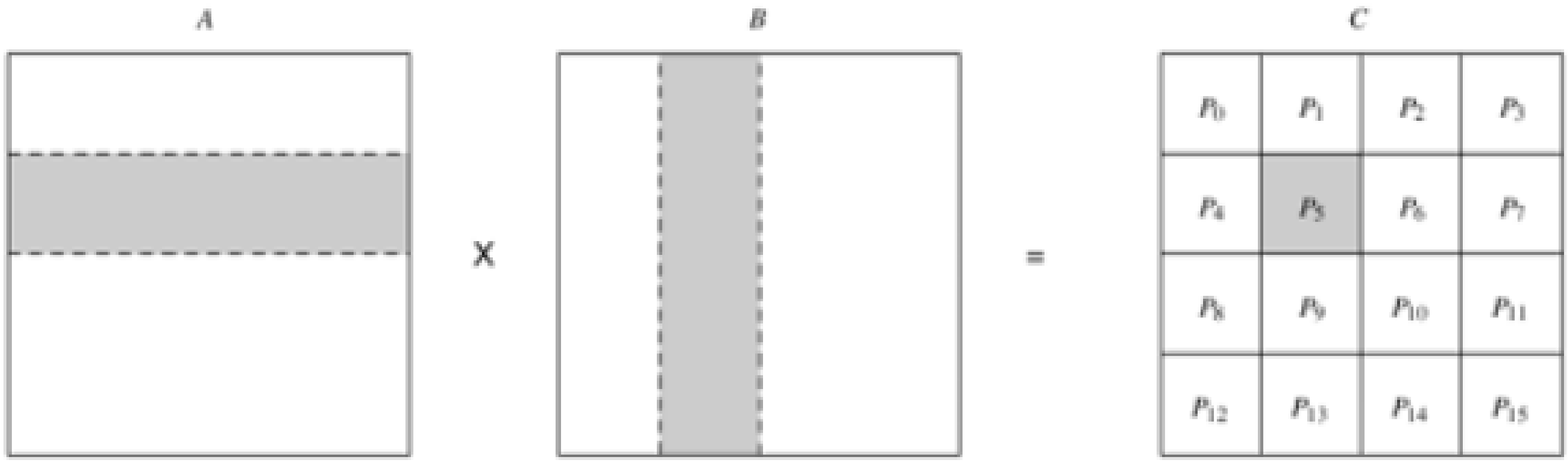
$S_p(n) = \frac{n^3}{n^2} = n;\ S_p(n) = O(n)$

$C_p(n) = O(n^3)$

$E_p(n) = \frac{n^3}{n * n^2} = 1$

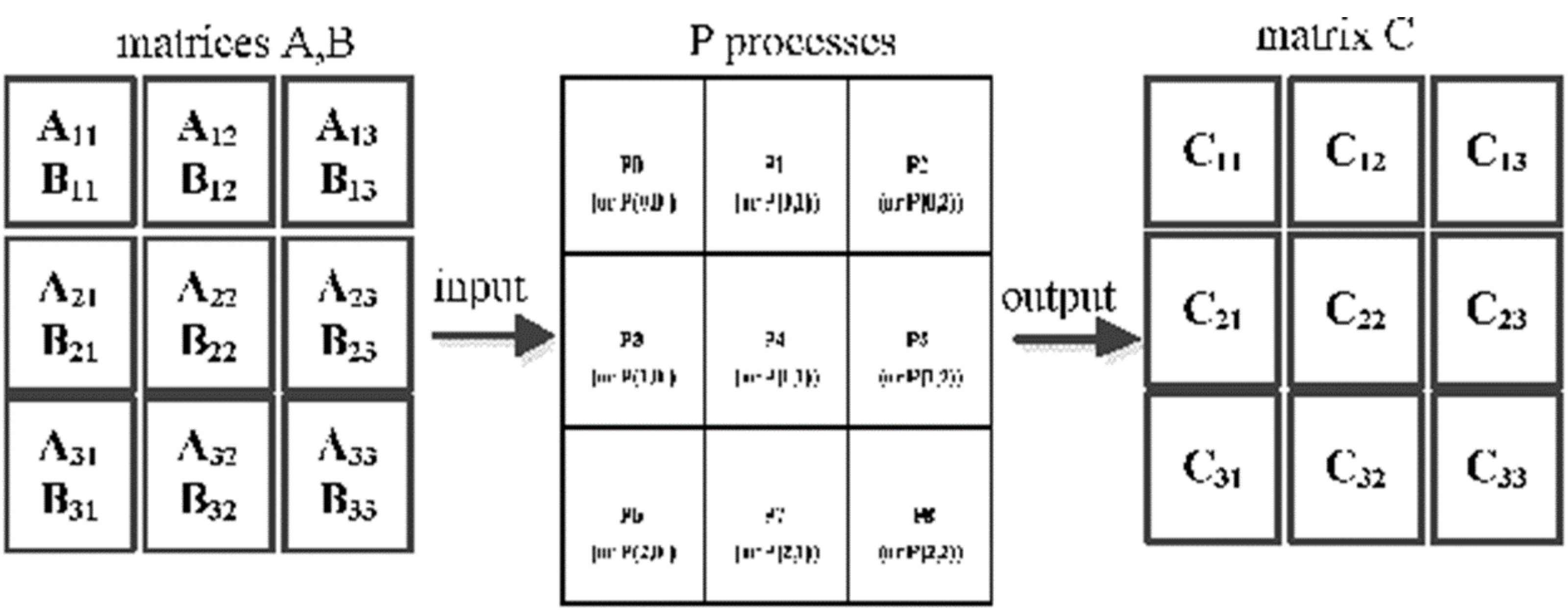
Matrix-Matrix Multiplication in Case of 2-Dim

Matrix-Matrix Multiplication in 2-Dim



Matrix-Matrix Multiplication in 2-Dim

The $C_{n \times n}$ matrix is partitioned among n^2 processors, with each processor computes one element of the matrix.



Matrix-Matrix Multiplication in 2-Dim Parallel Algorithm

Input:	Matrix A[n][n]
	Matrix B[n][n]
Output:	Matrix C[n][n]

```
for ( i = 0; i < n; i++ ) do in parallel
    for ( j = 0; j < n; j++ ) do in parallel
            C[i][j] = 0;
            for(k = 0; k < n; k++)
                        C[i][j] += A[i][k] * B[k][j];
```

Matrix-Matrix Multiplication in 2-Dim Parallel Algorithm

$T_p(n) = O(n)$

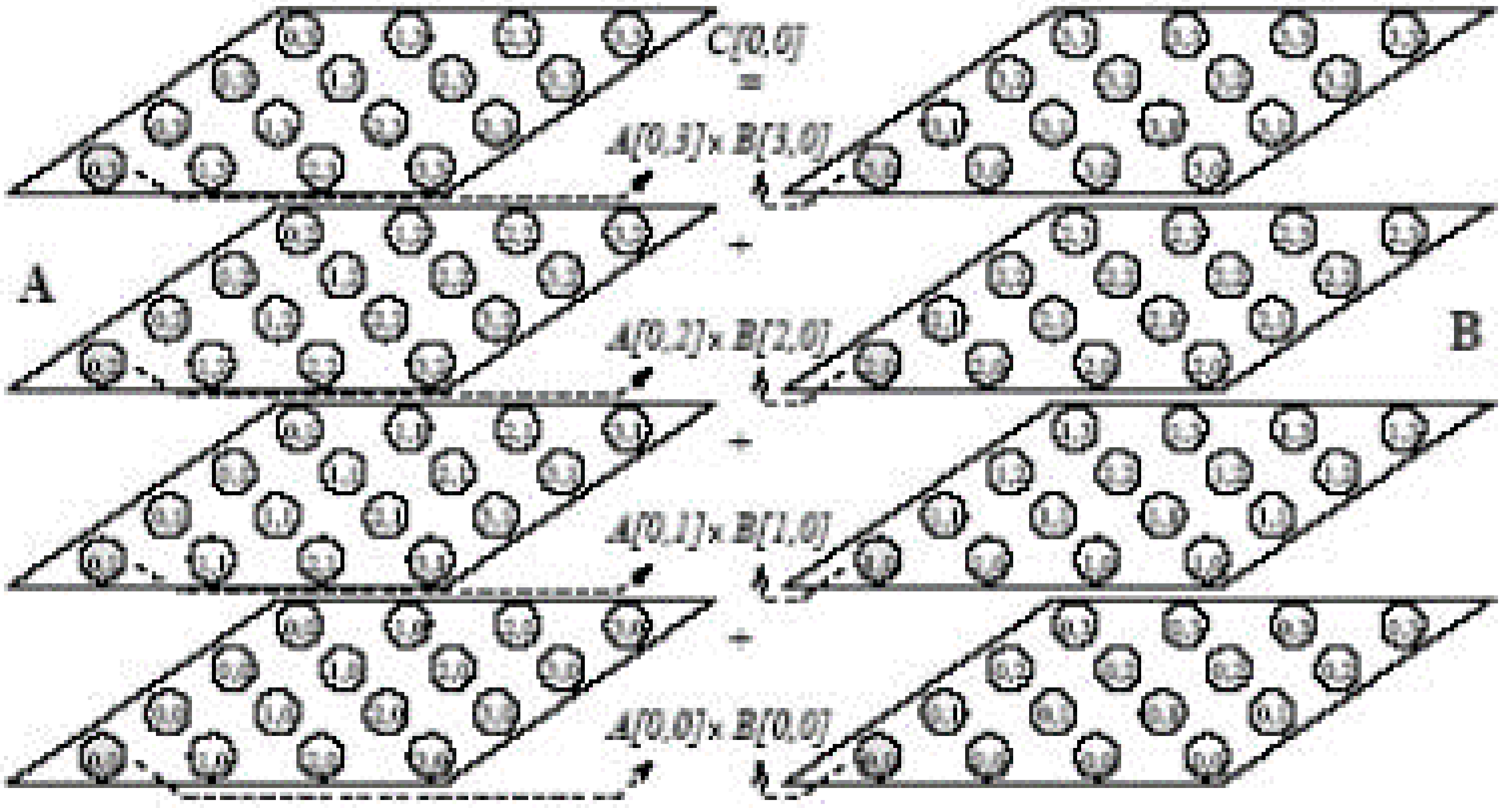
$S_p(n) = \frac{n^3}{n} = n^2;\ S_p(n) = O(n^2)$

$C_p(n) = O(n^3)$

$E_p(n) = \frac{n^3}{n^2 * n} = 1$

Matrix-Matrix Multiplication in Case of 3-Dim

Matrix-Matrix Multiplication: DNS Algorithm



Matrix-Matrix Multiplication: DNS Algorithm

Using fewer than n^3 processors.

Each processor computes a single add-multiply.

This is followed by an accumulation along the C dimension.

Since each add-multiply takes constant time and accumulation and broadcast takes log *n* time, the total runtime is log *n*.

Matrix-Matrix Multiplication: DNS Algorithm

$T_p(n) = O(\log n)$

$S_p(n) = \frac{n^3}{\log n}$

$C_p(n) = O(n^3 \ * \log n)$

$E_p(n) = \frac{n^3}{n^3 * \log n} = {}^1/_{\log n}$