# Parallel Programming

Lec 7

## Books

Chapman & Hall/CRC Numerical Analysis and Scientific Computing

#### Parallel Algorithms

Henri Casanova, Arnaud Legrand, and Yves Robert



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



Deringer

#### Wiley Series on Parallel and Distributed Computing • Albert Zomaya, Series Editor

#### ALGORITHMS AND PARALLEL COMPUTING



# PowerPoint

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

Benha Ui	niversity	Staff Settlefione:Ahmed Hassan Ahmed Abu El Atta (	Log out
Benha University Home	You are in: <u>Home/Courses/C</u> Ass. Lect. Ahmed Ha Compilers	ssan Ahmed Abu El Atta :: Course Details:	84
التسغة العربية		add course   edit course	
My C.V.	Course name	Compilers	6
About	Level	Undergraduate	
Publications	Last year taught	2018 f	
Inlinks(Competition)	Course description	Not Uploaded	C
Reporte			
Published books	Course password		
Workshops / Conferences			y .
Supervised PhD	Course files	add files	
Supervised MSc	Course URLs	add URLs	à
Supervised Projects	Course assignments	add assignments	
Education			
Language skills	Course Exams &Model Answers	add exams	
Academic Positions		ed (ed	<u>()</u>
Administrative Positions			

# Matrix-Matrix Multiplication

### Problem Statement



## Problem Statement

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

 $C = A \times B$ , is such that each of its elements is denoted ij with  $0 \le i < m$  and  $0 \le j < n$ , 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];</pre>
```

# 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.

					-				
thread1 ->	A11	A12	A13	A14		B11	B12	B13	B14
thread2 ->	A21	A22	A23	A24	$\times$	B21	B22	B23	B24
thread3 ->	A31	A32	A33	A34		B31	B32	B33	B34
thread4 ->	A41	A42	A43	A44		B41	B42	B43	B44

### 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];</pre>
```

### Matrix-Matrix Multiplication in 1-Dim Parallel Algorithm

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

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

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

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

# 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];</pre>
```

### 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<sup>3</sup> 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

 $T_p(n) = O(\log n)$   $S_p(n) = \frac{n^3}{\log n}$   $C_p(n) = O(n^3 * \log n)$  $E_p(n) = \frac{n^3}{n^3 * \log n} = \frac{1}{\log n}$ 

