**Design Analysis and Algorithm**

**Mini Project**

**Code:**

"""

Problem Statement :- Write a program to implement matrix multiplication.

Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell.

Analyze and compare their performance.

"""

import threading import numpy as np import time Matrix\_A = [] Matrix\_B = [] Matrix\_C = []

def Input\_for\_matrix\_dimensions(): """

Function to take the dimensional Input from the User. """

global A\_row, A\_col, B\_row, B\_col global num\_of\_threads

global V

A\_row = int(input("Enter the number of rows in first matrix: ")) A\_col = int(input("Enter the number of cloumns in first matrix: ")) B\_row = int(input("Enter the number of rows in second matrix: "))

B\_col = int(input("Enter the number of columns in second matrix: "))

V = int(input("Do you want to see the verbose? (1 for yes, 0 for no): "))

# number of threads = number of rows in solution matrix C = number of rows in matrix A as dim(C) = (A\_row, B\_col)

num\_of\_threads = A\_row def Initialize\_Matrix():

"""

Funtion to initialize the matrices by populating it with random numbers from 1 to 10. """

global Matrix\_A global Matrix\_B global Matrix\_C

if A\_col == B\_row:

Matrix\_A = np.random.randint(10, size=(A\_row, A\_col)) Matrix\_B = np.random.randint(10, size=(B\_row, B\_col)) Matrix\_C = np.zeros((A\_row, B\_col)).astype(int)

else:

exit()

def Matrix\_multiply\_parallel(row, z): """

Performs multiplication with respect to a row thread. """

global Matrix\_A global Matrix\_B global Matrix\_C

for i in range(B\_col): for j in range(A\_col):

Matrix\_C[z][i] += int(row[j] \* Matrix\_B[j][i]) def Thread\_function():

"""

Function for creation and execution of the row threads. """

global num\_of\_threads threads = []

# Creation and execution of threads for j in range(0,num\_of\_threads):

t = threading.Thread(target = Matrix\_multiply\_parallel, args=(Matrix\_A[j], j)) threads.append(t)

t.start()

# Completion of the threads for t in threads:

t.join()

if name ==" main ":

"""

Driver Code """

# The multithreaded approach that provides the complexity of O(n^2.8074) Input\_for\_matrix\_dimensions()

Initialize\_Matrix() start\_time = time.time() Thread\_function() end\_time = time.time()

print("Time taken to multiply two matrices in parallel comes out to be :",round(end\_time - start\_time, 5),"seconds\n")

# The non thread approach that provides the time complexity of O(n^3). result = np.zeros((A\_row, B\_col)).astype(int)

stat\_time = time.time() # iterating by row of A for i in range(A\_row):

# iterating by column by B for j in range(B\_col):

# iterating by rows of B for k in range(B\_row):

result[i][j] += Matrix\_A[i][k] \* Matrix\_B[k][j]

end\_time = time.time()

print("Time taken to multiply two matrices comes out to be :",round(end\_time - start\_time, 5),"seconds\n")

if V and result.all()==Matrix\_C.all(): print(Matrix\_C,"\n") print(Matrix\_A,"\n") print(Matrix\_B,"\n")

# Output: