**Lab 1: Distributed Data Analytics**

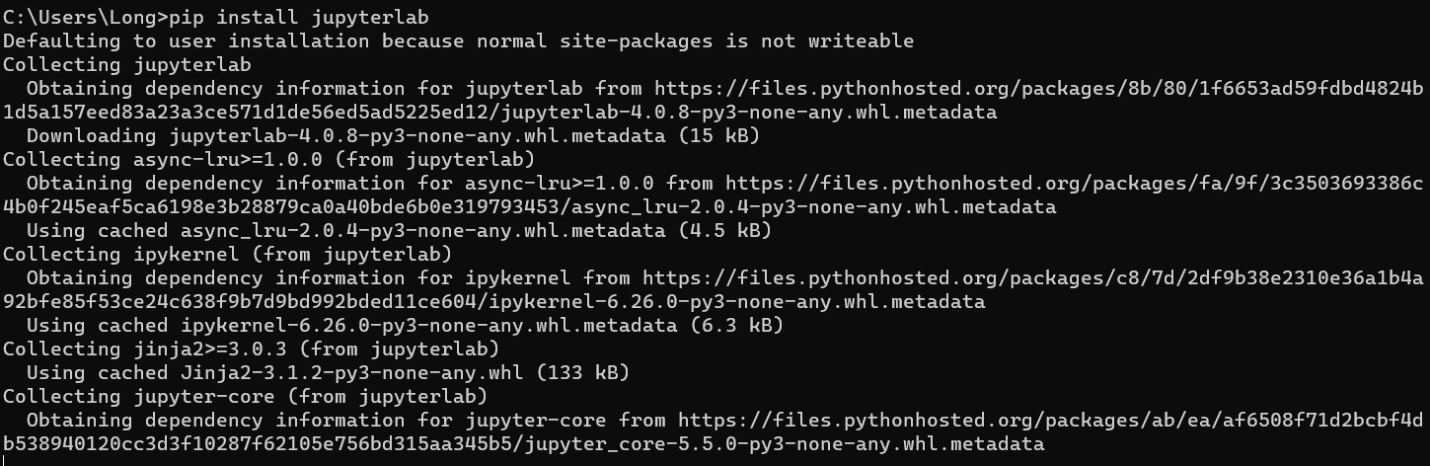
**Họ và Tên**: Trần Nguyễn Gia Long

**MSSV**: 1050080059

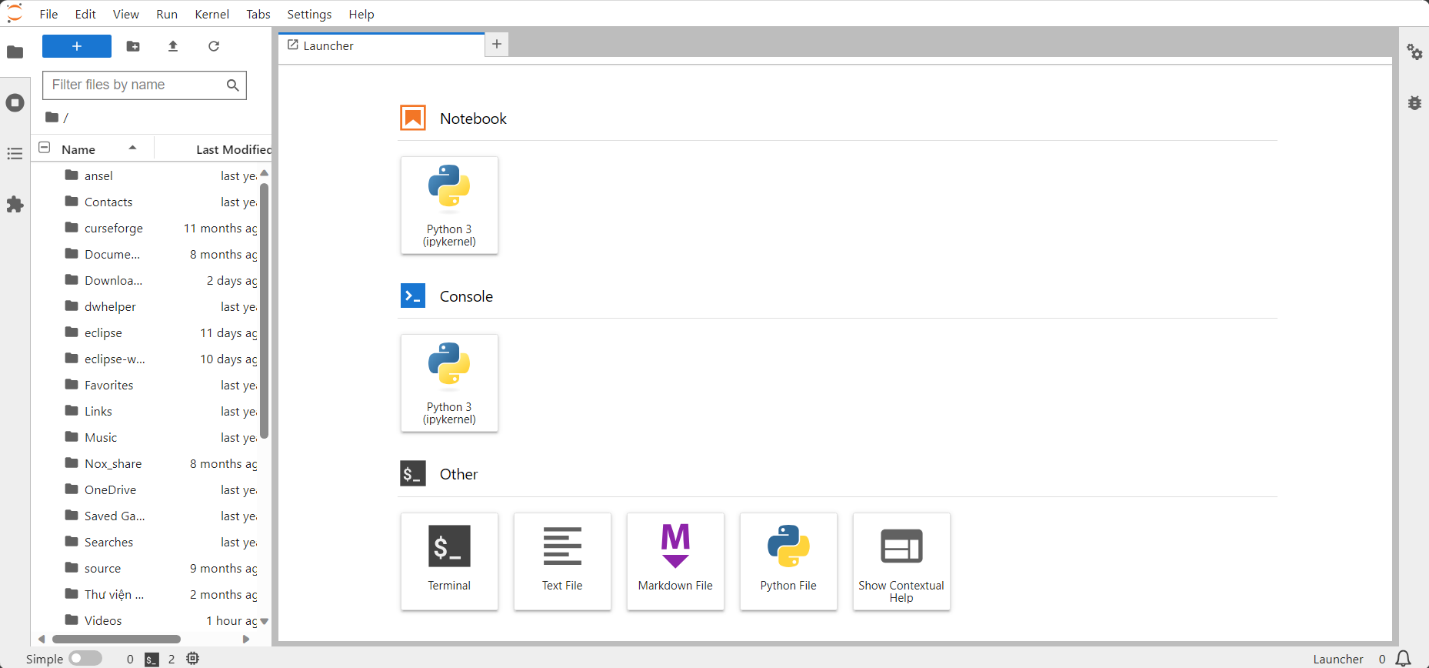
# Exercise 1: Basic Parallel Vectors Operations with MPI:

## 1. Cài đặt Jupyter Lab

- Dùng câu lệnh *pip install jupyterlab*

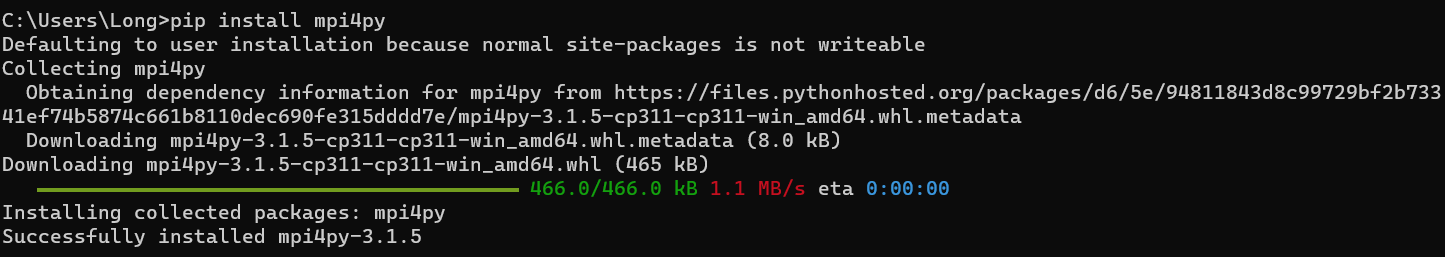


- Mở Jupyter Lab bằng câu lệnh *jupyter lab*

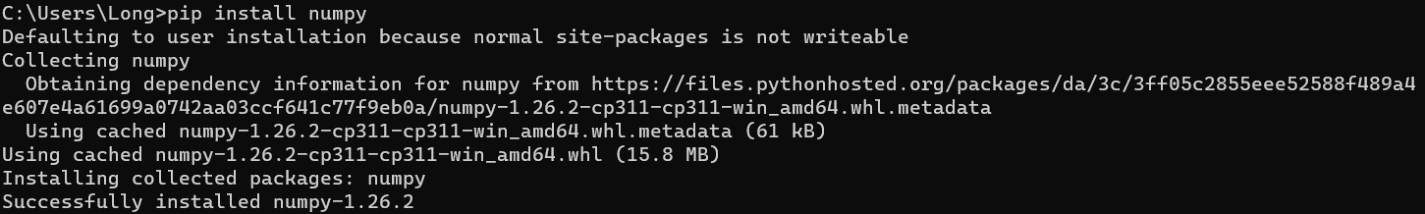


## 2. Cài đặt các thư viện cần thiết

- Cài đặt mpi4py bằng câu lệnh *pip install mpi4py*



- Cài đặt numpy bằng *pip install numpy*



## 3. Cài đặt MPI

- Thêm code vào Jupyter Lab.

from mpi4py import MPI

import numpy as np

#Initialize MPI

comm = MPI.COMM\_WORLD

#Get rank of the communicator

rank = comm.Get\_rank()

#Size of the process

size = comm.Get\_size()

#Variable Initialization

N = int(16)

sum\_xy = []

executiontime = 0

#Master Process

if rank != 0:

start = MPI.Wtime()

a = comm.recv(source = 0)

b = comm.recv(source = 0)

comm.send(a+b, dest = 0, tag = 1)

end = MPI.Wtime()

comm.send(end-start,dest=0, tag = 2)

print("Time taken by worker ",rank,"is: ", end - start)

#Worker Process

else:

start = MPI.Wtime()

x = np.random.randint(100,size = N)

y = np.random.randint(100, size = N)

#Splitting the dataset depending on the size of the workers

split\_x = np.array\_split(x,size)

split\_y = np.array\_split(y,size)

if rank == 0:

#Master's work

sum\_xy.extend(list(split\_x[rank]+split\_y[rank]))

for worker in range(1,size):

#Pointopoint communication to worker

comm.send(split\_x[worker], dest= worker)

comm.send(split\_y[worker], dest = worker)

sum\_xy.extend (list(comm.recv(source = worker,tag = 1)))

executiontime+=comm.recv(source = worker, tag = 2)

end = MPI.Wtime()

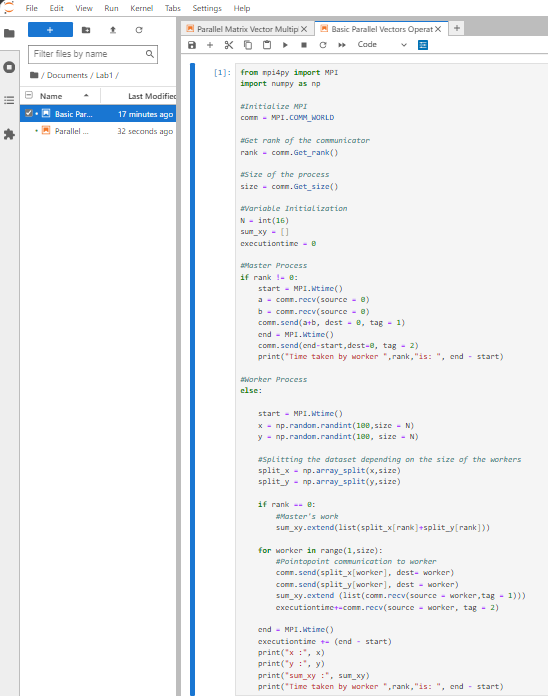
executiontime += (end - start)

print("x :", x)

print("y :", y)

print("sum\_xy :", sum\_xy)

print("Time taken by worker ",rank,"is: ", end - start)

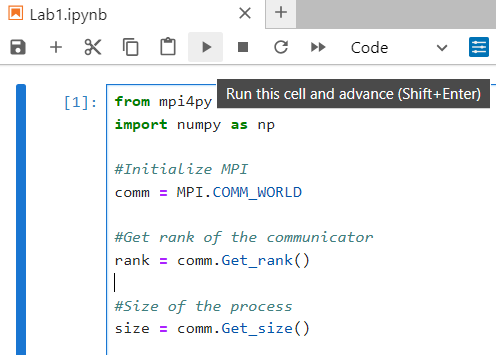


- Cách hoạt động:

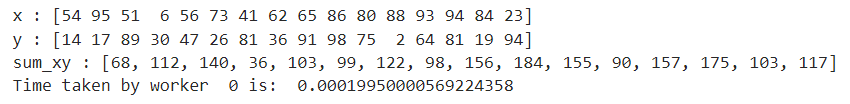
* Khởi tạo MPI: Đầu tiên tạo mã khởi tạo MPI và lấy hạng (rank) của trình điều khiển và kích thước của quá trình.
* Quy trình Master: Nếu hạng không phải là 0, đó là một quy trình worker. Quy trình này nhận hai vector từ quy trình master (source = 0), thực hiện phép cộng và sau đó gửi kết quả trở lại cho quy trình master. Thời gian thực thi cũng được tính và gửi trở lại cho quy trình master.
* Quy trình Worker: Nếu hạng là 0, đó là quy trình master. Quy trình này tạo hai vector ngẫu nhiên và chia chúng thành các phần nhỏ dựa trên kích thước của quá trình. Quy trình master sau đó thực hiện phép cộng trên phần của nó và gửi các phần còn lại đến các quy trình worker. Khi nhận được kết quả từ các quy trình worker, nó thêm chúng vào vector kết quả cuối cùng. Thời gian thực thi cũng được tính và in ra.

## 4. Chạy chương trình.

- Bấm Run để chạy



## 5. Kết quả



# Excercise 2: Parallel Matrix Vector Multiplication using MPI

## 1. Cài đặt MPI

#Import Library

from mpi4py import MPI

import numpy as np

#Initialize MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

N = int(1e4)

VecMult = []

executiontime = 0

if rank != 0:

start = MPI.Wtime()

A = comm.recv(source = 0)

b = comm.recv(source = 0)

comm.send(np.matmul (A,b), dest= 0, tag = 1)

end = MPI.Wtime()

comm.send(end-start, dest=0,tag=2)

print("Time taken by worker ", rank,"is: ", end - start)

else:

start = MPI.Wtime()

A = np.random.randint(100,size=(N,N))

b = np.random.randint(100,size=(N,1))

split\_A= np. array\_split(A, size)

if rank == 0:

out = np.matmul(split\_A[rank], b)

VecMult.extend (out.flatten().tolist())

for worker in range(1, size):

comm.send(split\_A[worker], dest = worker)

comm.send(b,dest-worker)

out = comm.recv(source=worker, tag=1)

VecMult.extend (out.flatten().tolist())

executiontime+=comm.recv(source=worker, tag=2)

end = MPI.Wtime()

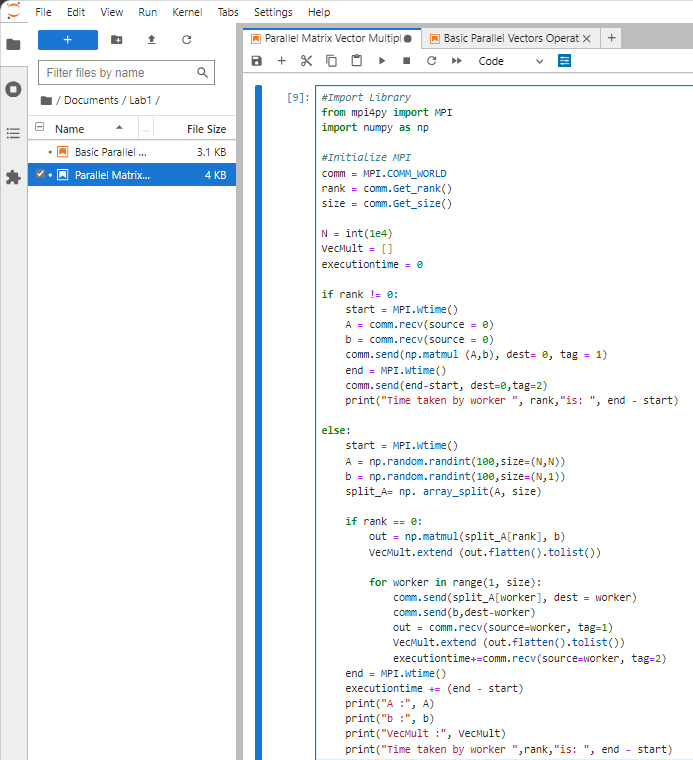
executiontime += (end - start)

print("A :", A)

print("b :", b)

print("VecMult :", VecMult)

print("Time taken by worker ",rank,"is: ", end - start)

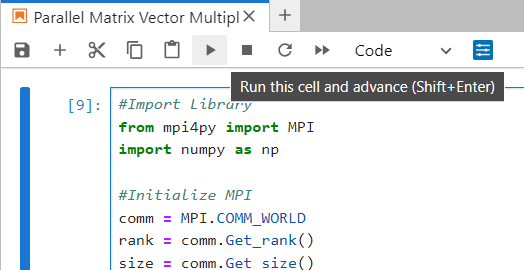


- Cách hoạt động:

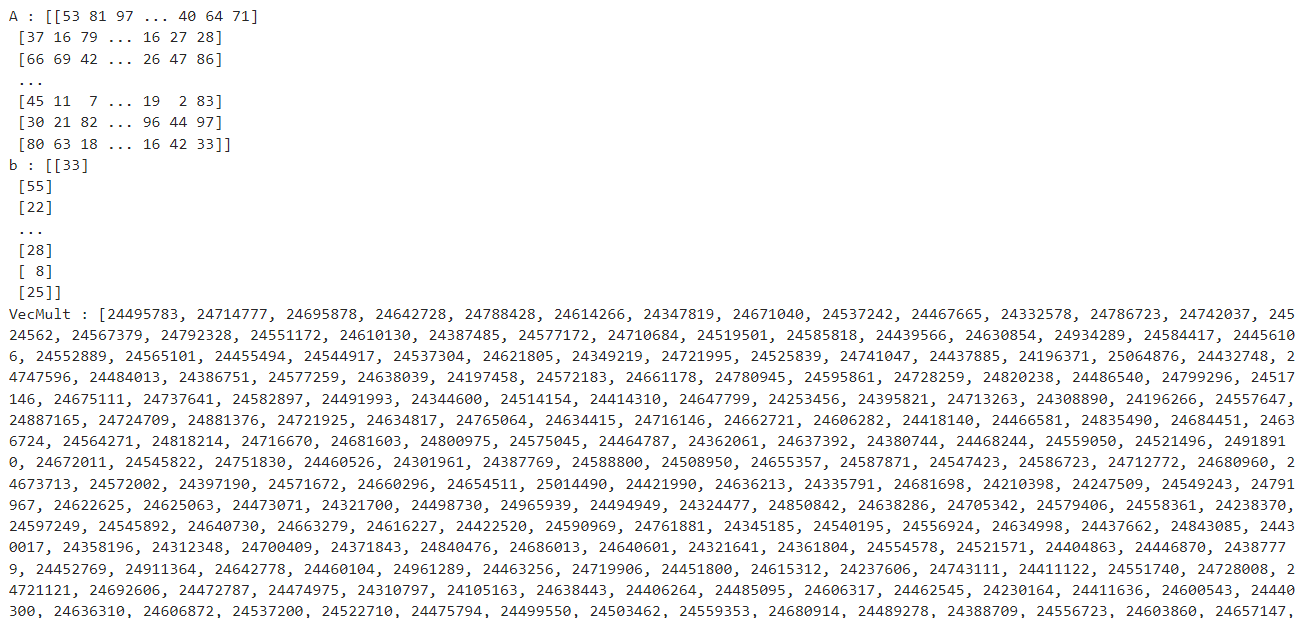
* Khởi tạo MPI: Đầu tiên, chương trình khởi tạo MPI và lấy rank (ID) và size (số lượng tiến trình) của tiến trình hiện tại.
* Tạo dữ liệu: Nếu tiến trình hiện tại là master (rank 0), nó sẽ tạo ra ma trận A và vector b với các giá trị ngẫu nhiên.
* Chia dữ liệu: Ma trận A sau đó được chia thành các phần nhỏ hơn, mỗi phần sẽ được gửi đến một tiến trình (bao gồm cả master). Mỗi tiến trình sẽ nhận một phần của ma trận A và toàn bộ vector b.
* Tính toán song song: Mỗi tiến trình (bao gồm cả master) sau đó thực hiện phép nhân ma trận-vector trên phần của ma trận mà nó nhận được. Kết quả sau đó được gửi trở lại cho master.
* Thu thập kết quả: Master nhận kết quả từ tất cả các tiến trình và kết hợp chúng lại để tạo ra vector kết quả cuối cùng.
* Tính toán thời gian: Thời gian thực hiện được tính toán và in ra màn hình. Thời gian này bao gồm thời gian tính toán của cả master và các tiến trình khác.

## 2. Chạy chương trình

- Sử dụng tổ hợp Shift + Enter hoặc bấm nút để chạy code.



## 3. Kết quả



# Exercise 3: Parallel Matrix Operation using MPI

## 1. Cài đặt MPI

#Import Library

from mpi4py import MPI

import numpy as np

#Initialize MPI

comm = MPI.COMM\_WORLD

size = comm.Get\_size()

rank = comm.Get\_rank()

#Initialize Matrices

A = None

B = None

N = int(1e4)

C = np.zeros((N,N))

if rank != 0:

B = np.empty((N,N))

else:

A = np.random.rand (N,N)

B = np.random.rand (N,N)

#Receiving variable

Arecv = np.empty((int(N/size),N))

start = MPI.Wtime()

#Separate data to all workers

comm.Scatter (A, Arecv, root=0)

#Send a copy of the vector to all workers

comm.Bcast (B,root=0)

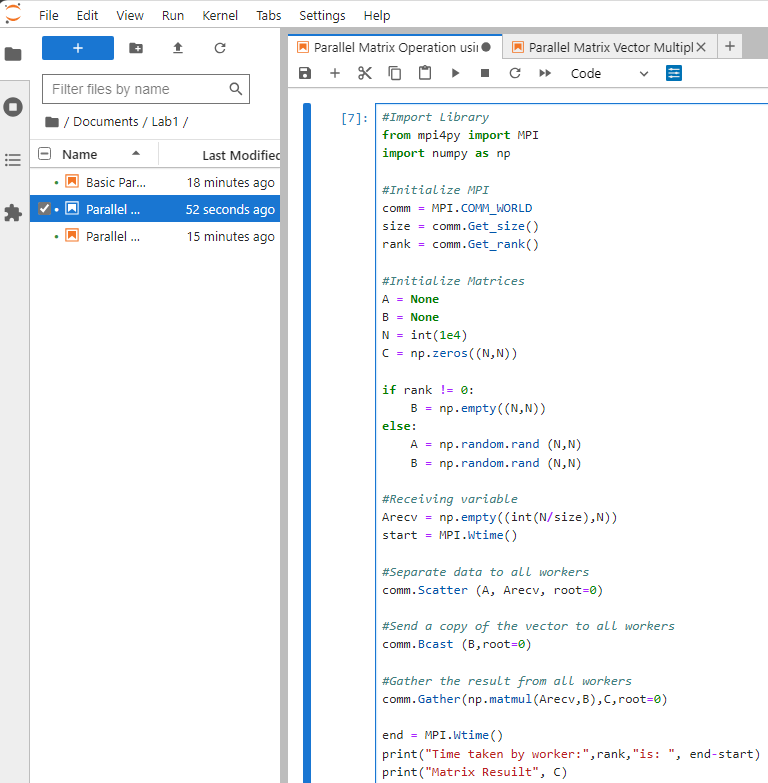
#Gather the result from all workers

comm.Gather(np.matmul(Arecv,B),C,root=0)

end = MPI.Wtime()

print("Time taken by worker:",rank,"is: ", end-start)

print("Matrix Resuilt", C)

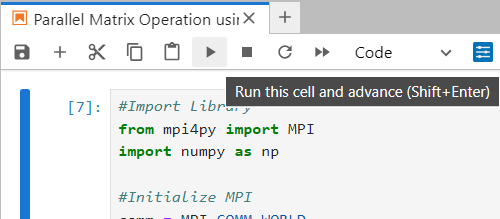


- Đoạn code thực hiện phép nhân ma trận song song bằng cách sử dụng MPI (Message Passing Interface). Cách hoạt động:

* Khởi tạo MPI: MPI được khởi tạo và số lượng tiến trình (size) và xếp hạng (rank) của tiến trình hiện tại được lấy ra. Xếp hạng 0 thường được gán cho tiến trình chính (master).
* Khởi tạo ma trận: Ma trận A và B được khởi tạo. Nếu tiến trình là master (rank 0), nó sẽ tạo ra hai ma trận ngẫu nhiên A và B. Các tiến trình khác (workers) sẽ tạo ra ma trận B rỗng.
* Phân chia dữ liệu: Master sẽ chia ma trận A thành các phần nhỏ hơn và gửi chúng đến các workers. Điều này được thực hiện bằng cách sử dụng hàm comm.Scatter.
* Gửi bản sao của ma trận B: Master sau đó gửi một bản sao của ma trận B đến tất cả các workers. Điều này được thực hiện bằng cách sử dụng hàm comm.Bcast.
* Thực hiện phép nhân ma trận: Các workers sau đó thực hiện phép nhân ma trận giữa phần của ma trận A mà chúng nhận được và ma trận B.
* Thu thập kết quả: Kết quả sau đó được thu thập lại bởi master và lưu trữ trong ma trận kết quả C. Điều này được thực hiện bằng cách sử dụng hàm comm.Gather.
* In thời gian và kết quả: Cuối cùng, thời gian mà mỗi worker mất để thực hiện công việc của mình và ma trận kết quả C được in ra. Thời gian được tính bằng cách lấy thời gian kết thúc trừ đi thời gian bắt đầu.

## 2. Chạy chương trình

- Sử dụng tổ hợp Shift + Enter hoặc bấm nút để chạy code.



## 3. Kết quả.

