

# Cómputo de Alto Rendimiento

## Actividad 5: Ejercicios de MPI y Map-reduce

**Nombre:** David Aaron Ramirez Olmeda

**Programa:** Maestría en Ciencia de Datos e Información



### Introducción:

En este conjunto de actividades, trabajamos en la implementación de códigos paralelos utilizando el paradigma Map-Reduce en un entorno de programación paralela con MPI (Message Passing Interface). Estos códigos se desarrollaron para realizar tareas específicas, como calcular el valor de  $\pi$ , realizar multiplicación de matrices y vectores, y contar palabras en archivos de texto. Utilizamos un enfoque paralelo para dividir el trabajo entre múltiples procesos y luego reunir los resultados. Este enfoque tiene como objetivo mejorar el rendimiento y la eficiencia al aprovechar la capacidad de procesamiento distribuido en sistemas paralelos.

### 1. Cálculo Paralelo de $\pi$ Utilizando MPI

```

In [ ]: from mpi4py import MPI
import time

start_time_parallel = time.time()
# Inicializa MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Define el número de divisiones para la aproximación
precision_factor = 1000000

# Calcula el rango de trabajo para cada proceso
chunk_size = precision_factor // size
start = rank * chunk_size
end = (rank + 1) * chunk_size

# Calcula la suma de Riemann en el rango asignado
partial_sum = 0.0
for i in range(start, end):
    x = (i + 0.5) / precision_factor
    partial_sum += 4.0 / (1.0 + x**2)

# Realiza la reducción para obtener el resultado global
total_sum = comm.reduce(partial_sum, op=MPI.SUM, root=0)

# En el proceso 0, muestra el resultado
if rank == 0:
    pi_parallel = total_sum / precision_factor
    end_time_parallel = time.time()
    print(f"Valor aproximado de  $\pi$  (paralelo): {pi_parallel}")
    print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_t

start_time_serial = time.time()

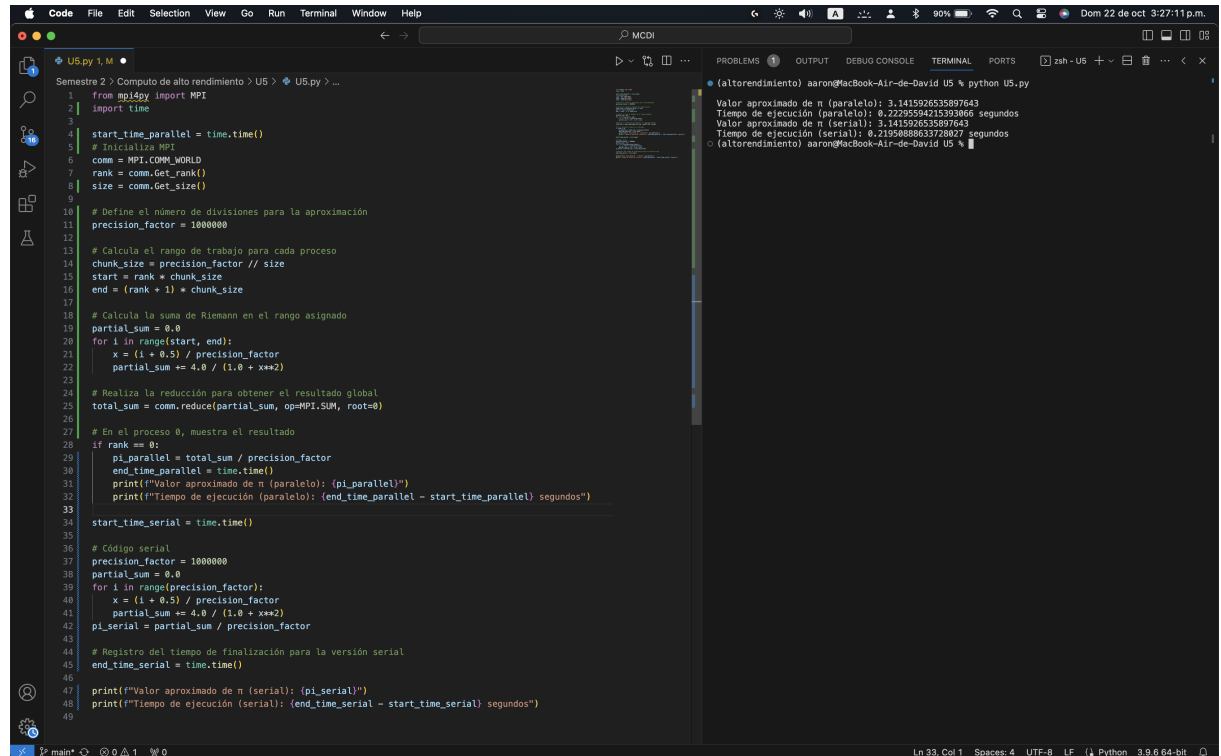
# Código serial
precision_factor = 1000000
partial_sum = 0.0
for i in range(precision_factor):
    x = (i + 0.5) / precision_factor
    partial_sum += 4.0 / (1.0 + x**2)
pi_serial = partial_sum / precision_factor

# Registro del tiempo de finalización para la versión serial
end_time_serial = time.time()

print(f"Valor aproximado de  $\pi$  (serial): {pi_serial}")
print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_serial}")

```

```
In [12]: from PIL import Image
import matplotlib.pyplot as plt
img = Image.open('/Users/aaron/Desktop/1.png')
display(img)
```



```
US.py 1, M
Semestre 2 > Computo de alto rendimiento > U5 > US.py > ...
1 from mpi4py import MPI
2 import time
3
4 start_time_parallel = time.time()
5 # Inicializa MPI
6 comm = MPI.COMM_WORLD
7 rank = comm.Get_rank()
8 size = comm.Get_size()
9
10 # Define el número de divisiones para la aproximación
11 precision_factor = 1000000
12
13 # Calcula el rango de trabajo para cada proceso
14 chunk_size = precision_factor // size
15 start = rank * chunk_size
16 end = (rank + 1) * chunk_size
17
18 # Calcula la suma de Riemann en el rango asignado
19 partial_sum = 0.0
20 for i in range(start, end):
21     x = (i + 0.5) / precision_factor
22     partial_sum += 4.0 / (1.0 + x*x)
23
24 # Realiza la reducción para obtener el resultado global
25 total_sum = comm.reduce(partial_sum, op=MPI.SUM, root=0)
26
27 # En el proceso 0, muestra el resultado
28 if rank == 0:
29     pi_parallel = total_sum / precision_factor
30     end_time_parallel = time.time()
31     print(f"Valor aproximado de pi (paralelo): {pi_parallel}")
32     print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_time_parallel} segundos")
33
34 start_time_serial = time.time()
35
36 # Código serial
37 precision_factor = 1000000
38 partial_sum = 0.0
39 for i in range(precision_factor):
40     x = (i + 0.5) / precision_factor
41     partial_sum += 4.0 / (1.0 + x*x)
42 pi_serial = partial_sum / precision_factor
43
44 # Registro del tiempo de finalización para la versión serial
45 end_time_serial = time.time()
46
47 print(f"Valor aproximado de pi (serial): {pi_serial}")
48 print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_serial} segundos")
49
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

(altorendimiento) aaron@MacBook-Air-de-David US % python US.py

Valor aproximado de pi (paralelo): 3.1415926535897643  
Tiempo de ejecución (paralelo): 0.2229594215293866 segundos  
Valor aproximado de pi (serial): 3.1415926535897643  
Tiempo de ejecución (serial): 0.219508863728027 segundos  
(altorendimiento) aaron@MacBook-Air-de-David US %

Ln 33, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 64-bit

## 2. Programación Paralela de Envío y Recepción de Mensajes con MPI

```

In [ ]: from mpi4py import MPI
import time

# Registro del tiempo de inicio para la versión paralela
start_time_parallel = time.time()

# Inicializa MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Definir el mensaje inicial
message = f"Mensaje de proceso {rank}"

# Enviar y recibir mensajes encadenados
for dest in range(1, size):
    if rank == dest:
        received_message = comm.recv(source=rank - 1)
        print(f"Soy el proceso {rank} y he recibido: {received_message}")
    elif rank == dest - 1:
        comm.send(message, dest=dest)

# Registro del tiempo de finalización para la versión paralela
end_time_parallel = time.time()

# Comparación de métricas de desempeño
if rank == 0:
    print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_t

# Registro del tiempo de inicio para la versión serial
start_time_serial = time.time()

# Código serial
for i in range(size - 1):
    message = f"Mensaje de proceso {i}"
    print(f"Soy el proceso {i} y he recibido: {message}")

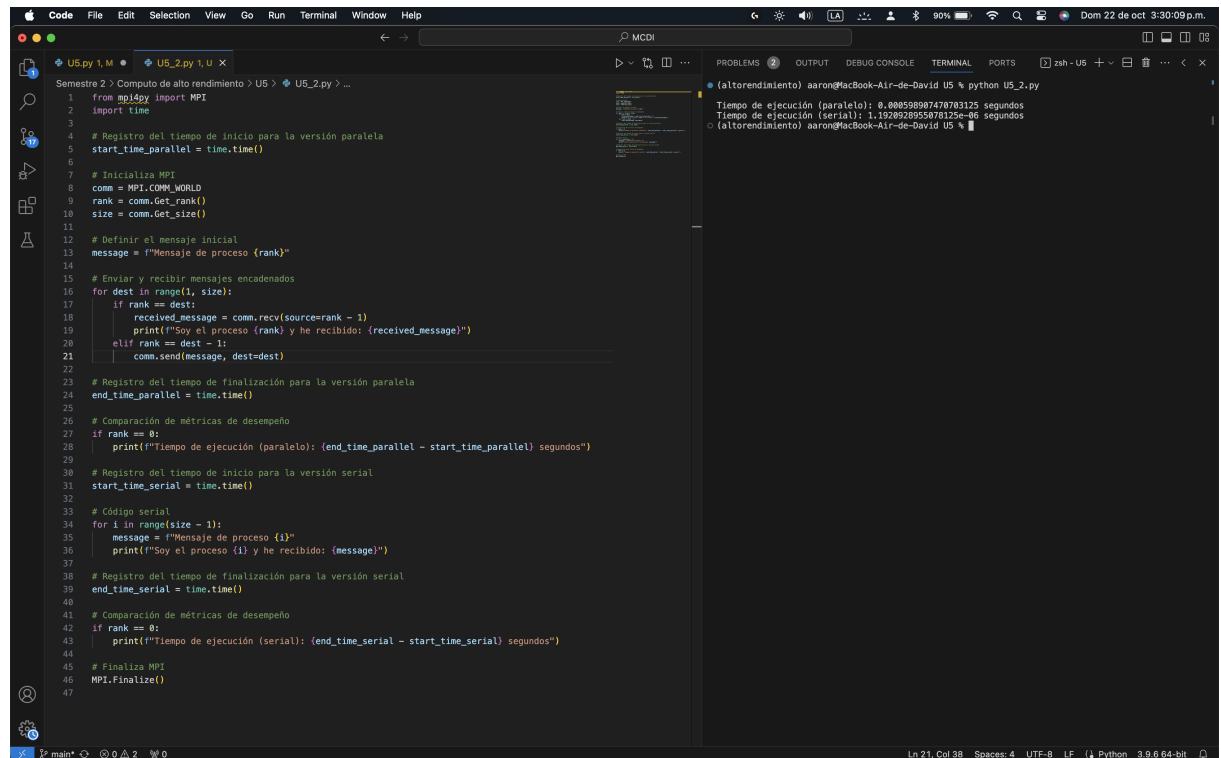
# Registro del tiempo de finalización para la versión serial
end_time_serial = time.time()

# Comparación de métricas de desempeño
if rank == 0:
    print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_

# Finaliza MPI
MPI.Finalize()

```

```
In [13]: from PIL import Image
import matplotlib.pyplot as plt
img = Image.open('/Users/aaron/Desktop/2.png')
display(img)
```



The screenshot shows a VS Code editor with a Python script named `US_2.py` and its execution output in the terminal. The script implements a parallel matrix multiplication using MPI. It compares the execution time of a parallel version (using MPI) with a serial version. The parallel version uses a loop to send and receive messages between processes. The serial version uses a single loop to calculate the product. The output in the terminal shows the execution times for both versions.

```
Semestre 2 > Computo de alto rendimiento > U5 > US_2.py > ...
1 from mpi4py import MPI
2 import time
3
4 # Registro del tiempo de inicio para la versión paralela
5 start_time_parallel = time.time()
6
7 # Inicializa MPI
8 comm = MPI.COMM_WORLD
9 rank = comm.Get_rank()
10 size = comm.Get_size()
11
12 # Definir el mensaje inicial
13 message = f"Mensaje de proceso {rank}"
14
15 # Enviar y recibir mensajes encadenados
16 for dest in range(1, size):
17     if rank == dest:
18         received_message = comm.recv(source=rank - 1)
19         print(f"Soy el proceso {rank} y he recibido: {received_message}")
20     elif rank == dest - 1:
21         comm.send(message, dest=dest)
22
23 # Registro del tiempo de finalización para la versión paralela
24 end_time_parallel = time.time()
25
26 # Comparación de métricas de desempeño
27 if rank == 0:
28     print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_time_parallel} segundos")
29
30 # Registro del tiempo de inicio para la versión serial
31 start_time_serial = time.time()
32
33 # Código serial
34 for i in range(size - 1):
35     message = f"Mensaje de proceso {i}"
36     print(f"Soy el proceso {i} y he recibido: {message}")
37
38 # Registro del tiempo de finalización para la versión serial
39 end_time_serial = time.time()
40
41 # Comparación de métricas de desempeño
42 if rank == 0:
43     print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_serial} segundos")
44
45 # Finaliza MPI
46 MPI.Finalize()
47
```

Terminal Output:

```
(altorendimiento) aaron@MacBook-Air-de-David U5 % python US_2.py
Tiempo de ejecución (paralelo): 0.000598907470703125 segundos
Tiempo de ejecución (serial): 1.1200938055070325e-05 segundos
(altorendimiento) aaron@MacBook-Air-de-David U5 %
```

### 3. Multiplicación Paralela de Matrices y Vectores con MPI

```

In [ ]: from mpi4py import MPI
import time

# Registro del tiempo de inicio para la versión paralela
start_time_parallel = time.time()

# Inicializa MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Define los vectores de prueba (completos)
vector_a = [1, 2, 3, 4]
vector_b = [5, 6, 7, 8]

# Realiza el producto escalar local
local_result = sum(a * b for a, b in zip(vector_a, vector_b))

# Reduce los resultados locales para obtener el resultado global
global_result = comm.reduce(local_result, op=MPI.SUM, root=0)

# Registro del tiempo de finalización para la versión paralela
end_time_parallel = time.time()

# Comparación de métricas de desempeño
if rank == 0:
    print(f"Resultado del producto escalar (paralelo): {global_result}")
    print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_time_parallel}")

# Registro del tiempo de inicio para la versión serial
start_time_serial = time.time()

# Código serial
serial_result = sum(a * b for a, b in zip(vector_a, vector_b))

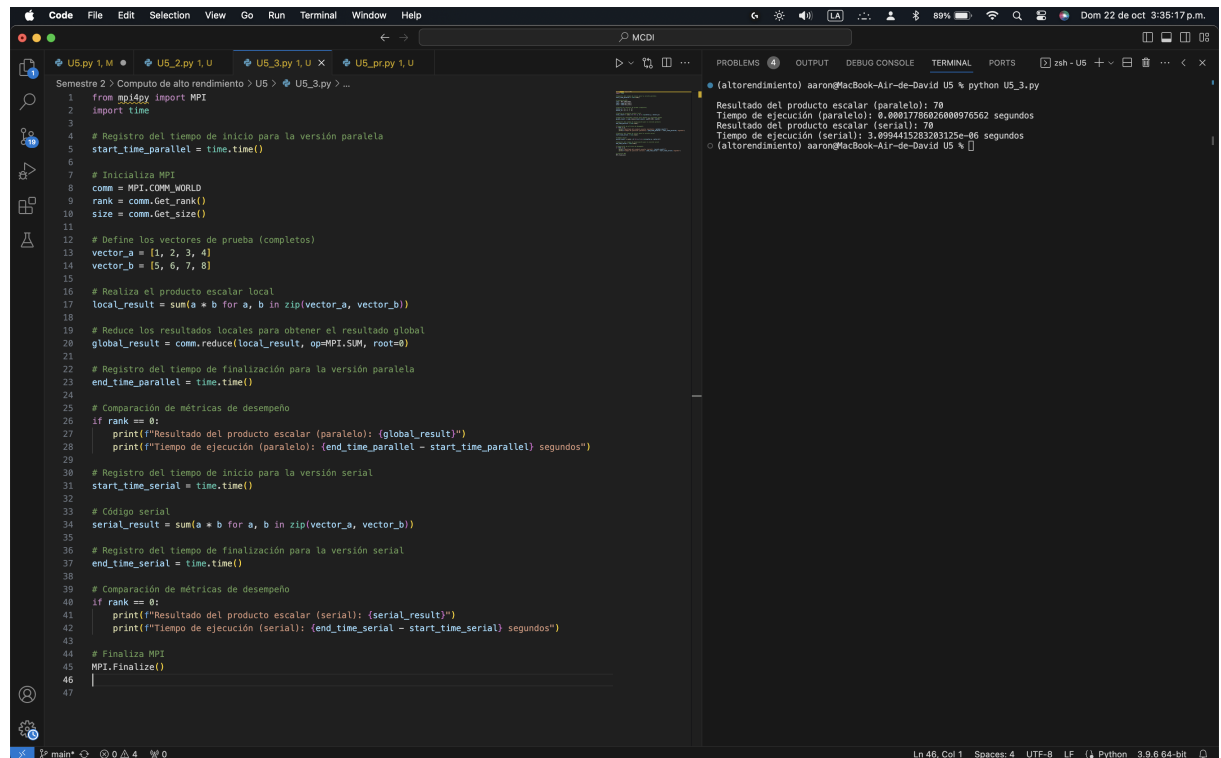
# Registro del tiempo de finalización para la versión serial
end_time_serial = time.time()

# Comparación de métricas de desempeño
if rank == 0:
    print(f"Resultado del producto escalar (serial): {serial_result}")
    print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_serial}")

# Finaliza MPI
MPI.Finalize()

```

```
In [14]: from PIL import Image
import matplotlib.pyplot as plt
img = Image.open('/Users/aaron/Desktop/3.png')
display(img)
```



```
US.py 1.M • US_2.py 1.U US_3.py 1.U US_prpy 1.U
Semestre 2 > Computo de alto rendimiento > U5 > US_3.py > ...
1 from mpi4py import MPI
2 import time
3
4 # Registro del tiempo de inicio para la versión paralela
5 start_time_parallel = time.time()
6
7 # Inicializa MPI
8 comm = MPI.COMM_WORLD
9 rank = comm.Get_rank()
10 size = comm.Get_size()
11
12 # Define los vectores de prueba (completos)
13 vector_a = [1, 2, 3, 4]
14 vector_b = [5, 6, 7, 8]
15
16 # Realiza el producto escalar local
17 local_result = sum(a * b for a, b in zip(vector_a, vector_b))
18
19 # Reduce los resultados locales para obtener el resultado global
20 global_result = comm.reduce(local_result, op=MPI.SUM, root=0)
21
22 # Registro del tiempo de finalización para la versión paralela
23 end_time_parallel = time.time()
24
25 # Comparación de métricas de desempeño
26 if rank == 0:
27     print(f"Resultado del producto escalar (paralelo): {global_result}")
28     print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_time_parallel} segundos")
29
30 # Registro del tiempo de inicio para la versión serial
31 start_time_serial = time.time()
32
33 # Código serial
34 serial_result = sum(a * b for a, b in zip(vector_a, vector_b))
35
36 # Registro del tiempo de finalización para la versión serial
37 end_time_serial = time.time()
38
39 # Comparación de métricas de desempeño
40 if rank == 0:
41     print(f"Resultado del producto escalar (serial): {serial_result}")
42     print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_serial} segundos")
43
44 # Finaliza MPI
45 MPI.Finalize()
46
47
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

• (altorendimiento) aaron@MacBook-Air-de-David US % python US\_3.py

Resultado del producto escalar (paralelo): 70  
Tiempo de ejecución (paralelo): 0.00037786826000976562 segundos  
Resultado del producto escalar (serial): 70  
Tiempo de ejecución (serial): 3.0994415283203125e-06 segundos  
• (altorendimiento) aaron@MacBook-Air-de-David US %

Ln 46, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 64-bit

#### 4. Multiplicación de una matriz en paralelo

```

In [ ]: from mpi4py import MPI
import time
import numpy as np

# Inicializa MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Definir el tamaño de la matriz
N = size # Asumiendo que el número de filas de la matriz es igual al nú

# Registro del tiempo de inicio para la versión paralela
start_time_parallel = time.time()

# Generar matriz A y vector x en el proceso 0
if rank == 0:
    matrix_A = np.random.rand(N, N)
    vector_x = np.random.rand(N, 1)
else:
    matrix_A = None
    vector_x = None

# Distribuir la matriz A y difundir el vector x
matrix_A = comm.scatter(matrix_A, root=0)
vector_x = comm.bcast(vector_x, root=0)

# Realizar la multiplicación de matriz-vector local
local_result = np.dot(matrix_A, vector_x)

# Recopilar los resultados locales en el proceso 0
global_result = comm.gather(local_result, root=0)

# Registro del tiempo de finalización para la versión paralela
end_time_parallel = time.time()

# Comparación de métricas de desempeño
if rank == 0:
    result_vector = np.sum(global_result, axis=0)
    print(f"Resultado de la multiplicación (paralelo):")
    print(result_vector)
    print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_t

# Registro del tiempo de inicio para la versión serial
start_time_serial = time.time()

# Código serial
if rank == 0:
    serial_result = np.dot(matrix_A, vector_x)
else:
    serial_result = None

# Recopilar los resultados locales en el proceso 0 (serial)
serial_result = comm.gather(serial_result, root=0)

# Registro del tiempo de finalización para la versión serial
end_time_serial = time.time()

```



```
# Comparación de métricas de desempeño
if rank == 0:
    result_vector = np.sum(serial_result, axis=0)
    print(f"Resultado de la multiplicación (serial):")
    print(result_vector)
    print(f"Tiempo de ejecución (serial): {end_time_serial - start_time_")

# Finaliza MPI
MPI.Finalize()
```

In [15]:

```
from PIL import Image
import matplotlib.pyplot as plt
img = Image.open('/Users/aaron/Desktop/4.png')
display(img)
```

The screenshot shows a code editor with a Python script for matrix multiplication performance comparison. The script is divided into two main sections: a parallel version using MPI and a serial version. The parallel version uses MPI to distribute the matrix and vector across multiple processes, while the serial version performs the calculation on a single process. The script compares the execution times of both versions and prints the results. The output shows that the parallel version is significantly faster than the serial version.

```
U5_4.py 1, U X
Semestre 2 > Computo de alto rendimiento > U5 > U5_4.py ...
5 # Inicializa MPI
6 comm = MPI.COMM_WORLD
7 rank = comm.Get_rank()
8 size = comm.Get_size()
9
10 # Definir el tamaño de la matriz
11 N = size # Asumiendo que el número de filas de la matriz es igual al número de procesos
12
13 # Registro del tiempo de inicio para la versión paralela
14 start_time_parallel = time.time()
15
16 # Generar matriz A y vector x en el proceso 0
17 if rank == 0:
18     matrix_A = np.random.rand(N, N)
19     vector_x = np.random.rand(N, 1)
20 else:
21     matrix_A = None
22     vector_x = None
23
24 # Distribuir la matriz A y difundir el vector x
25 matrix_A = comm.scatter(matrix_A, root=0)
26 vector_x = comm.bcast(vector_x, root=0)
27
28 # Realizar la multiplicación de matriz-vector local
29 local_result = np.dot(matrix_A, vector_x)
30
31 # Recopilar los resultados locales en el proceso 0
32 global_result = comm.gather(local_result, root=0)
33
34 # Registro del tiempo de finalización para la versión paralela
35 end_time_parallel = time.time()
36
37 # Comparación de métricas de desempeño
38 if rank == 0:
39     result_vector = np.sum(global_result, axis=0)
40     print(f"Resultado de la multiplicación (paralelo):")
41     print(result_vector)
42     print(f"Tiempo de ejecución (paralelo): {end_time_parallel - start_time_parallel} segundos")
43
44 # Registro del tiempo de inicio para la versión serial
45 start_time_serial = time.time()
46
47 # Código serial
48 if rank == 0:
49     serial_result = np.dot(matrix_A, vector_x)
50 else:
51     serial_result = None
52
53 # Recopilar los resultados locales en el proceso 0 (serial)
54 serial_result = comm.gather(serial_result, root=0)
55
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• (altorendimiento) aaron@MacBook-Air-de-David U5 % python U5_4.py
Resultado de la multiplicación (paralelo):
[0.13229888]
Tiempo de ejecución (paralelo): 0.0002810955047607422 segundos
Resultado de la multiplicación (serial):
[0.13229888]
Tiempo de ejecución (serial): 2.5987625122070312e-05 segundos
• (altorendimiento) aaron@MacBook-Air-de-David U5 %
```

## 5. Conteo de Palabras en Archivos de Texto con Map-Reduce

```
In [ ]: ### MAP ###

import sys
import re

# Función para dividir una línea en palabras y emitir pares (palabra, 1)
def map_function(line):
    words = re.findall(r'\w+', line) # Encuentra palabras usando expres.
    for word in words:
        print(f"{word.lower()}\t1") # Emitir palabra y 1

# Procesar líneas de entrada
for line in sys.stdin:
    map_function(line)

### REDUCE ###

import sys

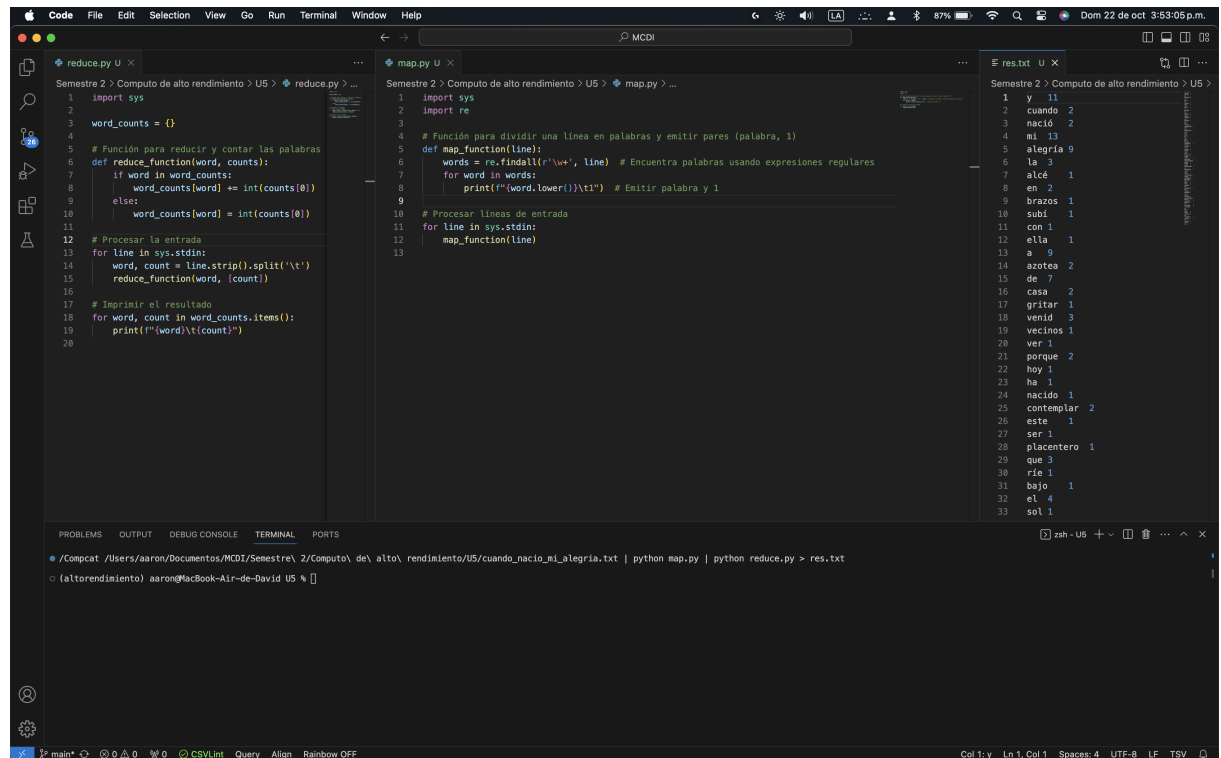
word_counts = {}

# Función para reducir y contar las palabras
def reduce_function(word, counts):
    if word in word_counts:
        word_counts[word] += int(counts[0])
    else:
        word_counts[word] = int(counts[0])

# Procesar la entrada
for line in sys.stdin:
    word, count = line.strip().split('\t')
    reduce_function(word, [count])

# Imprimir el resultado
for word, count in word_counts.items():
    print(f"{word}\t{count}")
```

```
In [16]: from PIL import Image
import matplotlib.pyplot as plt
img = Image.open('/Users/aaron/Desktop/5.png')
display(img)
```



```
Semestre 2 > Computo de alto rendimiento > U5 > reduce.py > ...
1 import sys
2
3 word_counts = {}
4
5 # Función para reducir y contar las palabras
6 def reduce_function(word, counts):
7     if word in word_counts:
8         word_counts[word] += int(counts[0])
9     else:
10        word_counts[word] = int(counts[0])
11
12 # Procesar la entrada
13 for line in sys.stdin:
14     word, count = line.strip().split('\t')
15     reduce_function(word, count)
16
17 # Imprimir el resultado
18 for word, count in word_counts.items():
19     print(f"{word}\t{count}")
20
```

```
Semestre 2 > Computo de alto rendimiento > U5 > map.py > ...
1 import sys
2 import re
3
4 # Función para dividir una línea en palabras y emitir pares (palabra, 1)
5 def map_function(line):
6     words = re.findall(r'\w+', line) # Encuentra palabras usando expresiones regulares
7     for word in words:
8         print(f"{word.lower()}\t1") # Emitir palabra y 1
9
10 # Procesar líneas de entrada
11 for line in sys.stdin:
12     map_function(line)
13
```

```
Semestre 2 > Computo de alto rendimiento > U5 > res.txt
1 y 11
2 cuando 2
3 nació 2
4 mi 13
5 alegría 9
6 la 3
7 alcé 1
8 en 2
9 brazos 1
10 subí 1
11 con 1
12 ella 1
13 a 9
14 azotea 2
15 de 7
16 casa 2
17 gritar 1
18 venid 3
19 vecinos 1
20 ver 1
21 porque 2
22 hoy 1
23 ha 1
24 nacido 1
25 contemplar 2
26 este 1
27 ser 1
28 placentero 1
29 que 3
30 río 1
31 bajo 1
32 el 4
33 sol 1
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
- /Compacat /Users/aaron/Documents/MCDI/Semestre 2/Computo de alto rendimiento/U5/cuando_nacio_el_alegria.txt | python map.py | python reduce.py > res.txt
- (altorendimiento) aaron@MacBook-Air-de-David U5 %
```

## Conclusión:

A lo largo de estas actividades, hemos explorado y aplicado conceptos fundamentales de programación paralela con MPI. Hemos aprendido cómo dividir y distribuir tareas entre procesos, realizar cálculos en paralelo y recolectar resultados. Además, hemos aplicado estos conceptos a problemas específicos, como el cálculo de  $\pi$ , la multiplicación de matrices y vectores, y el conteo de palabras en archivos de texto. Este enfoque paralelo permite mejorar el rendimiento y la escalabilidad de las aplicaciones, lo que es esencial en situaciones donde se requiere un alto poder de procesamiento. En resumen, estas actividades nos han proporcionado una introducción práctica a la programación paralela y al paradigma Map-Reduce, y nos han permitido comprender su utilidad en el procesamiento de datos y cálculos intensivos.