Actividad 5: Ejercicios de MPI y Map-reduce.

Cómputo de Alto Rendimiento

**Luis Fernando Izquierdo Berdugo**

23 de marzo de 2025

Ejercicio 1

Código:

from mpi4py import MPI

import numpy as np

import time

def calcular\_pi(n):

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

# El proceso 0 pide el número de subdivisiones

if rank == 0:

n = int(input("Ingrese el número de subdivisiones: "))

else:

n = None

# Se transmite el valor de n a todos los procesos

n = comm.bcast(n, root=0)

# Medición del tiempo de inicio en paralelo

comm.Barrier()

start\_time = MPI.Wtime()

# Definir la suma de Riemann

h = 1.0 / n # Ancho de cada subintervalo

local\_sum = 0.0

for i in range(rank, n, size):

x = (i + 0.5) \* h

local\_sum += 4.0 / (1.0 + x \* x)

local\_sum \*= h

# Reducir los resultados parciales al proceso 0

pi\_total = comm.reduce(local\_sum, op=MPI.SUM, root=0)

# Medición del tiempo de finalización en paralelo

comm.Barrier()

end\_time = MPI.Wtime()

elapsed\_time = end\_time - start\_time

# Medición de tiempo en versión secuencial (solo en el proceso 0)

if rank == 0:

start\_seq = time.time()

pi\_seq = sum(4.0 / (1.0 + ((i + 0.5) \* h) \*\* 2) for i in range(n)) \* h

end\_seq = time.time()

elapsed\_seq = end\_seq - start\_seq

print(f"Aproximación de π con {n} subdivisiones: {pi\_total}")

print(f"Tiempo transcurrido (paralelo): {elapsed\_time:.6f} segundos")

print(f"Tiempo transcurrido (secuencial): {elapsed\_seq:.6f} segundos")

# Cálculo del Speed-up y eficiencia

speed\_up = elapsed\_seq / elapsed\_time

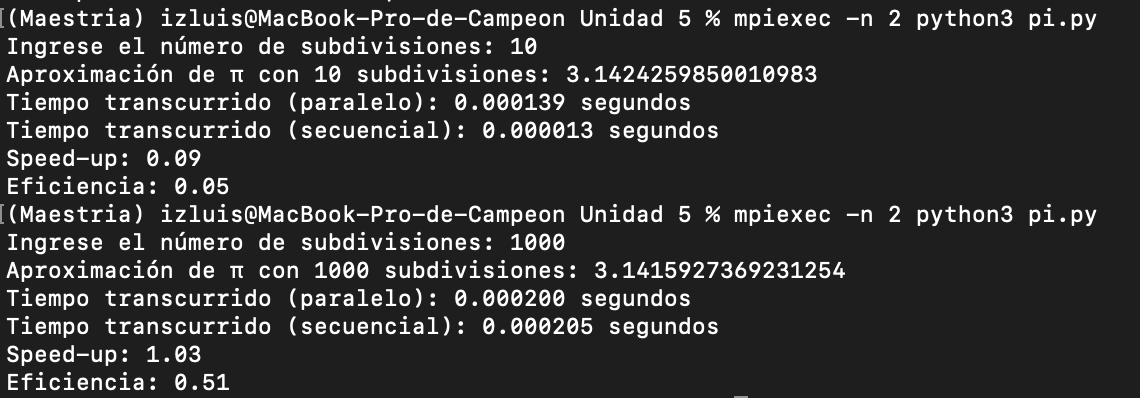
efficiency = speed\_up / size

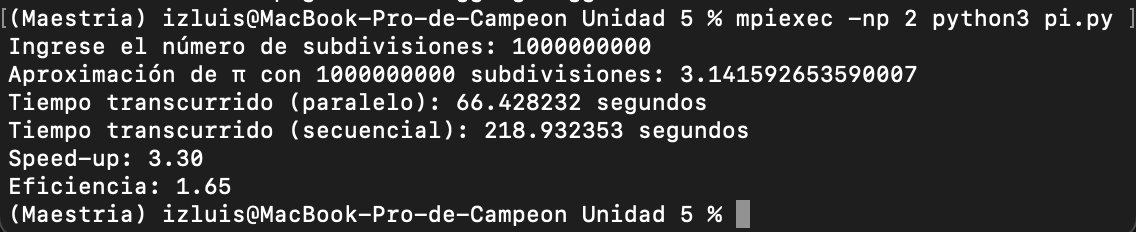
print(f"Speed-up: {speed\_up:.2f}")

print(f"Eficiencia: {efficiency:.2f}")

if \_\_name\_\_ == "\_\_main\_\_":

calcular\_pi(0)

Captura de pantalla de ejecución del código.



En las métricas se puede observar que en numeros pequeños de subdivisiones como 10 y 1000, la implementación secuencial es mejor en cuento a tiempo que el paralelo, sin embargo, al incrementar a 1,000,000,000 de subdivisiones se observa un tiempo mejor, lo cual incrementa el valor de speed-up y de eficiencia. De igual manera, se observa que el valor de pi se hace más exacto conforme a más subdivisiones se ejecuten.

Ejercicio 2

Código

from mpi4py import MPI

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

# Tiempo de inicio

start\_time = MPI.Wtime()

if rank == 0:

mensaje = "test paralelo" # Mensaje inicial

print(f"Soy el proceso {rank} y envío el mensaje {mensaje} al proceso {rank + 1}")

comm.send(mensaje, dest=rank + 1)

else:

mensaje = comm.recv(source=rank - 1)

print(f"Soy el proceso {rank} y he recibido {mensaje}")

if rank < size - 1:

print(f"Soy el proceso {rank} y envío el mensaje {mensaje} al proceso {rank + 1}")

comm.send(mensaje, dest=rank + 1)

# Tiempo final y cálculo de métricas

end\_time = MPI.Wtime()

elapsed\_time = end\_time - start\_time

total\_time = comm.reduce(elapsed\_time, op=MPI.MAX, root=0)

if rank == 0:

secuential\_time = total\_time \* size # Tiempo secuencial supuesto

print(f"Tiempo paralelo: {total\_time:.6f} segundos")

print(f"Tiempo secuencial supuesto: {secuential\_time:.6f} segundos")

speedup = secuential\_time / total\_time

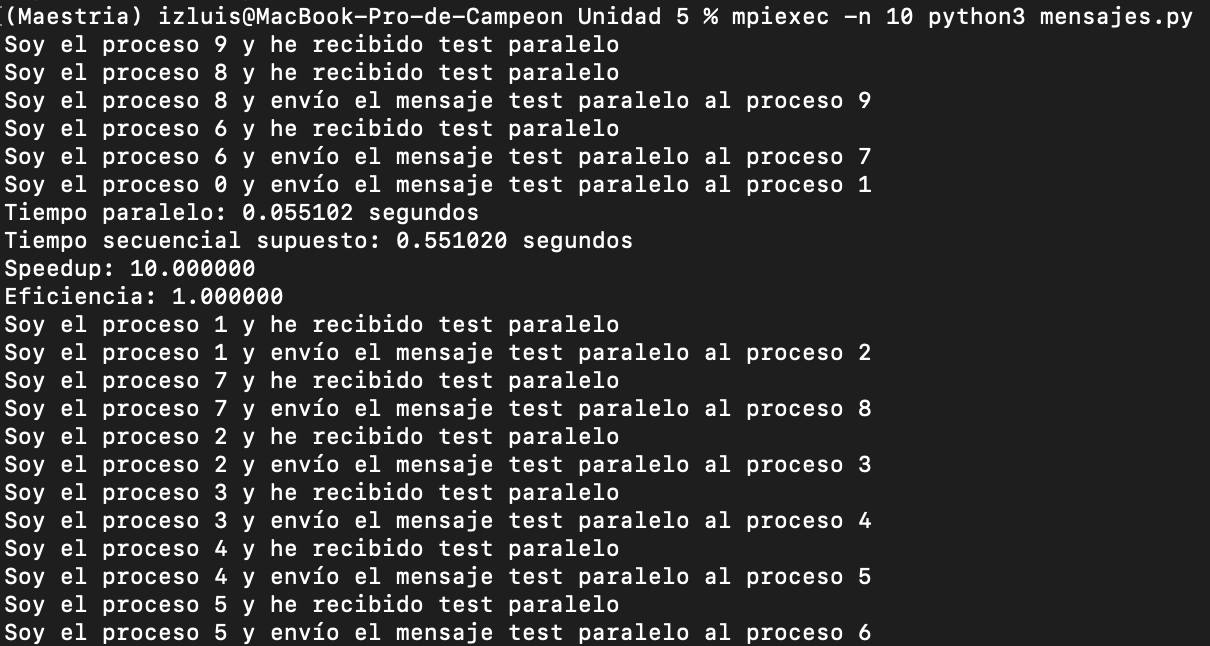
efficiency = speedup / size

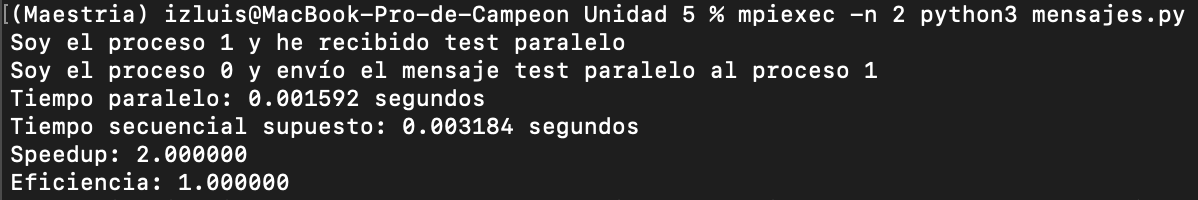
print(f"Speedup: {speedup:.6f}")

print(f"Eficiencia: {efficiency:.6f}")

MPI.Finalize()

Ejecución





Se estima el tiempo secuencial usando el tiempo total multiplicado por el número de procesos utilizados., siendo el speedup siempre el número de procesos y una eficiencia constante de 1. Algo que llamó mi atención en este ejercicio fue que cuando aumentaba el número de procesos a más que el número de núcleos de mi PC (2), el programa empezaba a imprimir los envíos y recibos de mensajes después de arrojarme el resultado.

Ejercicio 3

Código

from mpi4py import MPI

import numpy as np

import time

def producto\_escalar():

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

N = 10 # Tamaño del vector

if rank == 0:

A = np.random.rand(N)

B = np.random.rand(N)

else:

A = None

B = None

# Tamaño del fragmento que recibe cada proceso

local\_N = N // size

local\_A = np.zeros(local\_N)

local\_B = np.zeros(local\_N)

# Distribuir los vectores entre los procesos

comm.Scatter(A, local\_A, root=0)

comm.Scatter(B, local\_B, root=0)

# Medición del tiempo en paralelo

comm.Barrier()

start\_time = MPI.Wtime()

# Producto escalar parcial

local\_dot = np.dot(local\_A, local\_B)

# Reunir resultados en el proceso 0

total\_dot = comm.reduce(local\_dot, op=MPI.SUM, root=0)

comm.Barrier()

end\_time = MPI.Wtime()

elapsed\_time = end\_time - start\_time

if rank == 0:

# Versión secuencial

start\_seq = time.time()

dot\_seq = np.dot(A, B)

end\_seq = time.time()

elapsed\_seq = end\_seq - start\_seq

print(f"Producto escalar paralelo: {total\_dot}")

print(f"Producto escalar secuencial: {dot\_seq}")

print(f"Tiempo transcurrido (paralelo): {elapsed\_time:.6f} segundos")

print(f"Tiempo transcurrido (secuencial): {elapsed\_seq:.6f} segundos")

# Cálculo del Speed-up y eficiencia

speed\_up = elapsed\_seq / elapsed\_time

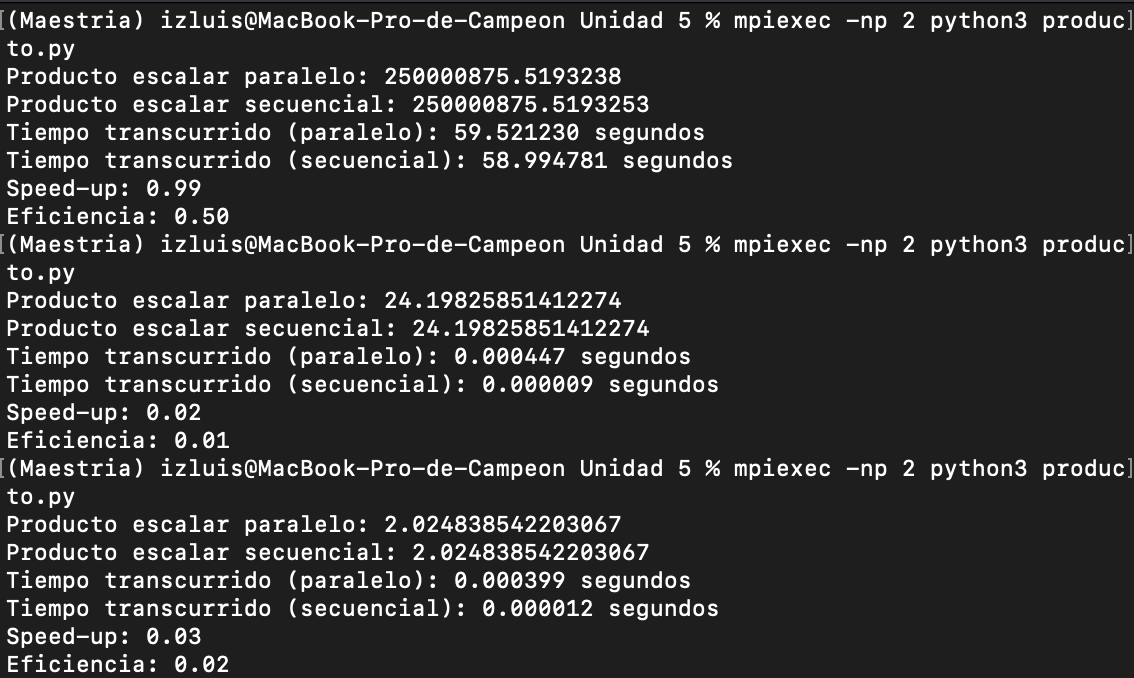
efficiency = speed\_up / size

print(f"Speed-up: {speed\_up:.2f}")

print(f"Eficiencia: {efficiency:.2f}")

if \_\_name\_\_ == "\_\_main\_\_":

producto\_escalar()



Se ejecutó el código con vectores tamaño 1,000,000,000, 100 y 10 donde se observó de nuevo que en tiempo de procesamiento no hay una gran diferencia incluso en vectores muy grandes, siendo el Speed-Up más grande de 0.99, eficiencia de 0.5 y diferencia de tiempo de poco menos de un segundo.

Ejercicio 4

Código

from mpi4py import MPI

import numpy as np

import time

def multiplicacion\_matriz\_vector():

comm = MPI.COMM\_WORLD

rank = comm.Get\_rank()

size = comm.Get\_size()

N = size # La matriz debe ser NxN, con N igual al número de procesos

if rank == 0:

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

x = np.random.rand(N)

else:

A = None

x = np.zeros(N)

# Cada proceso recibe una fila de A

local\_A = np.zeros(N)

comm.Scatter(A, local\_A, root=0)

# Se transmite el vector x a todos los procesos

comm.Bcast(x, root=0)

# Medición del tiempo en paralelo

comm.Barrier()

start\_time = MPI.Wtime()

# Cada proceso calcula su producto fila-vector

local\_result = np.dot(local\_A, x)

# Reunir resultados en el proceso 0

result = None

if rank == 0:

result = np.zeros(N)

comm.Gather(local\_result, result, root=0)

comm.Barrier()

end\_time = MPI.Wtime()

elapsed\_time = end\_time - start\_time

if rank == 0:

# Versión secuencial

start\_seq = time.time()

result\_seq = np.dot(A, x)

end\_seq = time.time()

elapsed\_seq = end\_seq - start\_seq

print(f"Prueba con un tamaño de {N}x{N}")

print(f"Resultado paralelo: {result}")

print(f"Resultado secuencial: {result\_seq}")

print(f"Tiempo transcurrido (paralelo): {elapsed\_time:.6f} segundos")

print(f"Tiempo transcurrido (secuencial): {elapsed\_seq:.6f} segundos")

# Cálculo del Speed-up y eficiencia

speed\_up = elapsed\_seq / elapsed\_time

efficiency = speed\_up / size

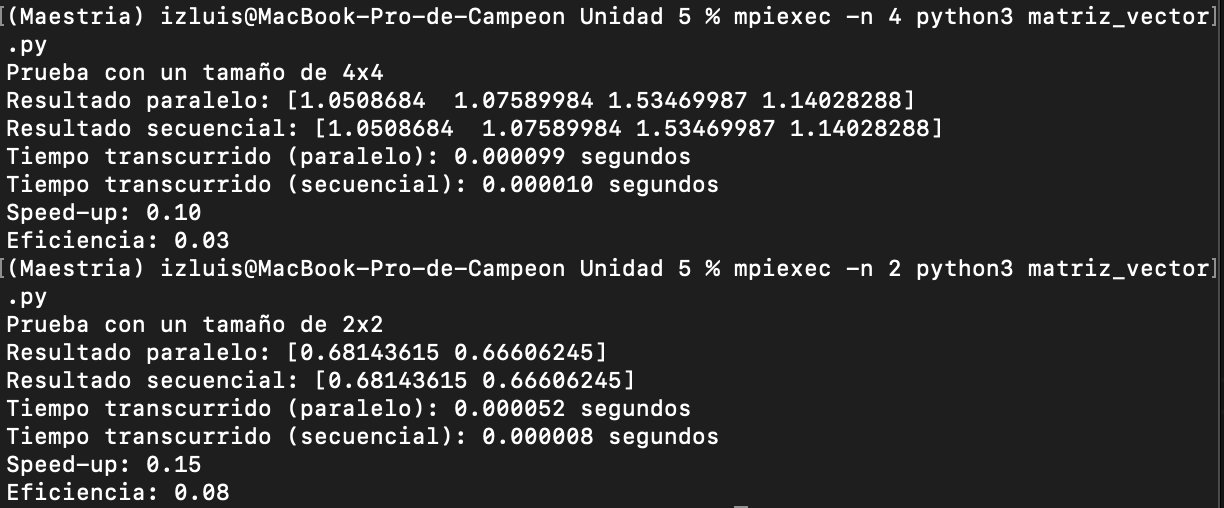
print(f"Speed-up: {speed\_up:.2f}")

print(f"Eficiencia: {efficiency:.2f}")

if \_\_name\_\_ == "\_\_main\_\_":

multiplicacion\_matriz\_vector()

Ejecución



Se hizo la prueba con vectores matrices de tamaño 4x4 y 2x2, en lo cual se observó que los procesos secuenciales son mejores que los paralelos, teniendo speed-up de tamaños 0.1 y 0.15, así como eficiencias de 0.03 y 0.08.

Ejercicio 5

Código

Map.py

import sys

import time

import re

def map\_words():

"""

Fase Map: Lee líneas de texto desde la entrada estándar (sys.stdin),

elimina caracteres especiales excepto ñ y letras con acentos,

convierte a minúsculas, divide cada línea en palabras y emite pares palabra\t1.

"""

# Inicia un temporizador para medir el tiempo de ejecución

start\_time = time.time()

# Lee líneas de la entrada estándar

for line in sys.stdin:

# Elimina caracteres especiales excepto letras con acentos y ñ

clean\_line = re.sub(r'[^a-zA-ZáéíóúÁÉÍÓÚñÑ0-9\s]', '', line).lower()

# Divide la línea en palabras

words = clean\_line.strip().split()

# Emite cada palabra con un conteo inicial de 1

for word in words:

print(f"{word}\t1")

# Calcula el tiempo total de ejecución

end\_time = time.time()

elapsed\_time = end\_time - start\_time

# Imprime el tiempo transcurrido en la salida de error estándar (sys.stderr)

print(f"Tiempo transcurrido (Map): {elapsed\_time:.6f} segundos", file=sys.stderr)

if \_\_name\_\_ == "\_\_main\_\_":

map\_words()

reduce.py

import sys

from collections import defaultdict

import time

def reduce\_words():

"""

Fase Reduce: Lee pares palabra\tconteo desde la entrada estándar (sys.stdin),

agrupa las palabras y suma sus conteos totales.

"""

# Inicia un temporizador para medir el tiempo de ejecución

start\_time = time.time()

# Diccionario para almacenar la cuenta de cada palabra

# defaultdict(int) inicializa automáticamente los valores en 0

word\_counts = defaultdict(int)

# Lee líneas de la entrada estándar

# Cada línea contiene una palabra y su cuenta separadas por un tabulador

for line in sys.stdin:

word, count = line.strip().split("\t") # Divide la línea en palabra y cuenta

word\_counts[word] += int(count) # Suma la cuenta al total de la palabra

# Itera sobre las palabras y sus cuentas, ordenadas alfabéticamente

for word, count in sorted(word\_counts.items()):

# Imprime cada palabra y su cuenta separadas por un tabulador

print(f"{word}\t{count}")

# Calcula el tiempo total de ejecución

end\_time = time.time()

elapsed\_time = end\_time - start\_time

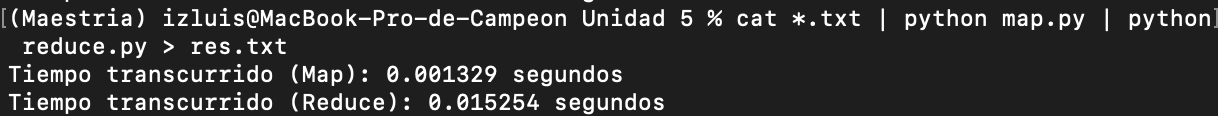
# Imprime el tiempo transcurrido en la salida de error estándar (sys.stderr)

print(f"Tiempo transcurrido (Reduce): {elapsed\_time:.6f} segundos", file=sys.stderr)

if \_\_name\_\_ == "\_\_main\_\_":

reduce\_words()

Ejecución



res.txt

0000 2

1 184

10 1

11 1

12 1

13 2

15 1

18 1

180 2

180c 2

2 44

20 2

200 2

25 2

250 2

26 1

3 21

350 3

380 2

4 12

40 1

5 4

6 4

7 5

70 2

8 3

9 4

90 2

a 14

aadir 2

acudi 1

acudió 1

advenimiento 2

agregar 3

ahora 3

aislamiento 2

al 4

alados 2

alc 1

alcé 1

alegra 1

alegría 9

all 1

allí 1

almendras 3

amable 2

amables 2

ambamos 1

amorosa 2

amábamos 2

arenosa 2

azcar 5

azotea 3

bajo 2

barnizar 3

besaban 2

bol 2

bondadoso 2

brazos 2

calle 2

caminbamos 1

camino 2

caminábamos 1

canciones 2

canela 3

cantbamos 1

canto 2

cantos 2

cantábamos 1

casa 3

cerezas 3

charola 2

colocar 2

como 4

compadecidas 2

con 13

contemplar 3

corazn 1

corazón 1

cortar 2

creci 1

creció 1

cuando 12

cubos 2

cuchicheaba 2

cuidados 2

cuidando 2

dar 2

das 1

de 19

desde 2

deshidratadas 2

dicen 2

diferentes 2

dulzura 2

durante 2

días 2

el 10

ella 2

elocuente 3

en 7

enferm 1

enfermó 1

enfriar 2

engalanadas 2

envidiara 2

era 6

eran 3

es 2

escuchan 2

escucharme 2

escucharnos 2

espolvoreada 2

espolvorear 3

estaba 2

estaban 3

estbamos 1

este 2

estábamos 1

extender 2

extraos 1

extraños 1

extremada 2

forma 2

fuera 2

fuerte 2

fuerza 2

g 11

gente 2

gozaba 2

gracias 2

grande 2

gritar 2

ha 2

haba 1

hablbamos 1

hablo 2

hablábamos 1

había 1

harina 4

harinas 2

hasta 3

hasto 1

hastío 1

hermosa 2

hermosura 2

hoja 2

hombre 2

hornear 5

hoy 2

huevo 3

huevos 3

impregnadas 2

instante 2

juntos 3

la 14

labios 3

las 4

le 3

leche 3

lengua 2

llena 2

lleno 2

los 5

luego 4

lunas 2

mantequilla 3

mar 2

maravillosas 2

masa 2

me 4

melodas 1

melodías 1

mesa 2

mezcla 2

mezclado 2

mi 27

mil 2

minutos 2

mira 2

miraba 2

mirad 2

mis 7

ml 2

mo 1

molida 3

ms 1

muerta 3

mundo 2

muri 1

murió 3

más 1

mío 2

naci 1

nacido 2

nació 4

nadie 4

ningn 1

ningún 1

no 4

noble 2

noches 2

nos 5

nuestras 3

nuestros 4

obtener 2

odos 1

oigo 2

ojos 2

orgulloso 2

orse 1

otoo 1

otoño 1

oídos 1

oírse 1

palabras 2

palideci 1

palideció 1

para 5

pasar 2

pero 5

pesadas 2

placentero 2

polvo 3

por 3

porque 4

proclam 1

proclamé 1

prodigu 1

prodigué 1

profundos 2

pues 5

qu 1

que 8

qued 1

quedé 1

quien 2

quiso 2

qué 1

raspa 2

re 1

recordar 2

recuerdo 3

recuerdos 2

reflexiones 2

rejilla 2

retirar 2

rodeaba 2

rodillo 2

ríe 1

sal 3

scones 2

se 2

seguidas 2

senta 1

sentbanse 1

sentábanse 1

sentía 1

ser 4

seres 2

siete 2

sin 2

slo 1

sobre 4

sol 2

soledad 2

solo 4

solos 2

sorpresa 2

su 3

sub 1

subí 1

suenan 2

sueos 1

sueños 2

suficiente 2

sus 2

susurra 2

sólo 4

tambin 1

también 3

ternura 2

todas 2

todo 2

todos 3

tostadas 2

trabajar 4

tringulos 2

triste 2

tristeza 16

un 5

una 6

unirse 2

vaciar 2

vecino 2

vecinos 4

venid 4

ventana 2

ver 2

viento 2

vigil 1

vigilé 1

visitarnos 2

viviente 2

vivientes 2

voces 2

vueltas 2

vuelve 2

y 41

ya 3

yace 2

yo 8

za 2

En este código se utilizó cat en vez de ls en el comando de ejecución, ya que, al ejecutar con ls, solamente se leía el nombre de los archivos y no el contenido de estos. De igual se observan tiempos muy pequeños al ejecutar el map y el reduce.

**Referencias**:

Arellano Vázquez, M. (s.f.). *Paradigam map-reduce*. Ciudad de México, México: INFOTEC.

Arellano Vázquez, M. (s.f.). *Lenguaje MPI.* Ciudad de México, México: INFOTEC.

Arellano Vázquez, M. (2025). *Manual de Funciones de MPI en C++ y Python 3.* Ciudad de México, México: INFOTEC.