

	<b>VICERRECTORADO DOCENTE</b>	<b>Código:</b> GUIA-PRL-001
	CONSEJO ACADÉMICO	<b>Aprobación:</b> 2016/04/06
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		<b>PRÁCTICA DE LABORATORIO</b>
<b>CARRERA:</b> Computación		<b>ASIGNATURA:</b> Computo Paralelo
<b>NRO. PRÁCTICA:</b>	3	<b>TÍTULO PRÁCTICA:</b> Desarrollo e implementación de aplicaciones de cómputo paralelo basado en paso de mensajes
<b>OBJETIVO ALCANZADO:</b>  <b>Desarrollo e implementación de aplicaciones de cómputo paralelo basado en paso de mensajes (MPI)</b>  <p>Con base al examen de interciclo se pide implementar la solución a dicho problema utilizando el protocolo de paso de mensajes (MPI), y las diferentes técnicas de comunicación (pointToPoint, broadcast, scatter, gather, y allToAll). Finalmente, en el informe de la práctica se debe indicar una gráfica en la que se pueda visualizar:</p> <ul style="list-style-type: none"> <li>• Comparación del tiempo de ejecución entre las diferentes técnicas de comunicación de MPI</li> <li>• Comparación del tiempo de ejecución entre secuencial, procesos (examen) y MPI (la mejor técnica en cuanto a tiempo de procesamiento)</li> <li>• Para la recolección de los tiempos de ejecución se pide ejecutar la multiplicación en matrices de: 64x64, 128x128, 256x256, 512x512, 1024x1024 y 2048x2048.</li> </ul>		

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

## ACTIVIDADES DESARROLLADAS

### 1. Point To Point

En la practica de punto a punto se utilizo 9 procesos, en donde el proceso que inicia (Rank=0) es el encargado de realizar las matrices y las divisiones de estas, para que se pueda enviar a los procesos que van a realizar la multiplicación de la matriz, los demás procesos (Rank 1-8) realizan la multiplicación de la matriz y proceden a enviar los datos nuevamente al proceso inicial (Rank==0) el cual es encargado de unir la matriz resultante y comparar los resultados con la función numpy

```
from mpi4py import MPI
import time
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.rank
# print("my rank is : ", rank)

def matrizrrr(data):
    filas_a = len(data[0])
    filas_b = len(data[1])
    columnas_a = len(data[0][0])
    columnas_b = len(data[1][0])
    size_ = (filas_a, columnas_a)
    calculo = np.zeros(dtype=float, shape=size_)
    for i in range(filas_a):
        for j in range(columnas_b):
            suma = 0
            for k in range(columnas_a):
                suma += data[0][i][k] * data[1][k][j]
            calculo[i][j] = suma
    return calculo

if rank == 0:
    size_ = (512, 512)
    matriz1 = np.random.randint(10, size=size_).astype("float") / 100
    matriz2 = np.random.randint(10, size=size_).astype("float") / 100
    producto_res = np.dot(matriz1, matriz2)
    filas = np.vsplit(matriz1, 8)
    columnas = matriz2
    inicio = time.time()
    for i in range(1, 9):
        fila = filas[i-1]
        destination_process = i
        comm.send([fila, columnas], dest=destination_process)

    data2 = list([0, 0, 0, 0, 0, 0, 0, 0])

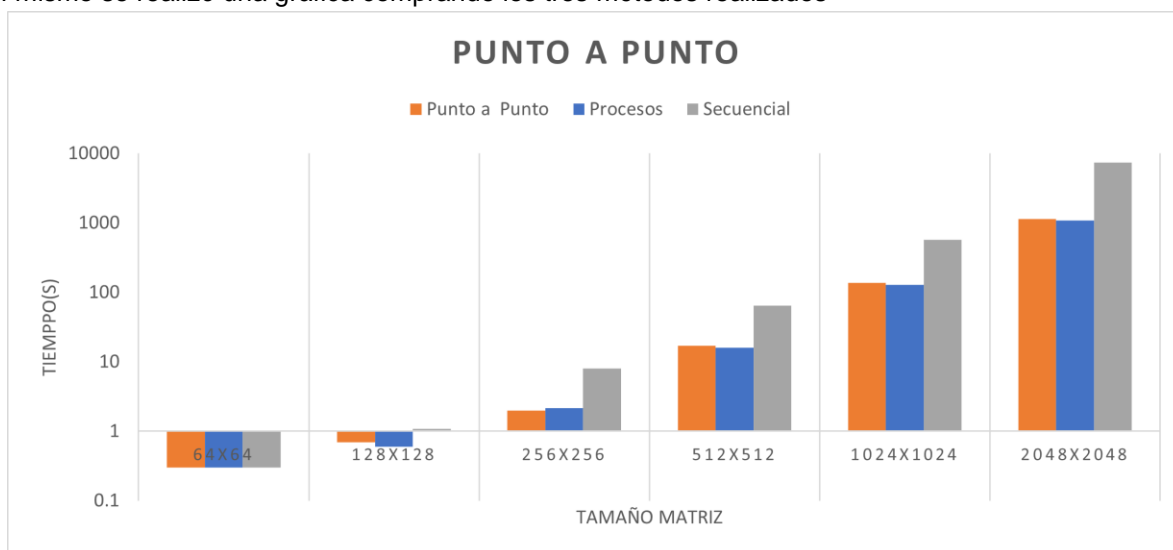
    for i in range(1, 9):
        data2[i-1] = comm.recv(source=i)
    ListaUnida = np.concatenate((data2[0], data2[1], data2[2], data2[3],
    ], data2[4], data2[5], data2[6], data2[7]), axis=0)
    fin = time.time()
    print('¿Los resultados son correctos?', np.allclose(ListaUnida,
    producto_res))
    print("
    El proceso de multiplicar las matrices paralelamente PuntoAPunto se e
    jecutó en
    %d segundos" % (fin - inicio))


    for kk in range(1, 9):
        if rank == kk:
            data = comm.recv(source=0)
            multiplicacion = matrizrrr(data)
            destination_process = 0
            comm.send(multiplicacion, dest=destination_process)
```

Además, se realizó la comparación de los tiempos de cada una de las matrices solicitadas, con los métodos Secuencial y Paralelo

Matrices	Punto a Punto	Procesos	Secuencial
64x64	0.3	0.3	0.3
128x128	0.7	0.6	1.08
256x256	2	2.14	8
512x512	17	16	64
1024x1024	138	128.4	573.6
2048x2048	1146	1086.6	7.416

Así mismo se realizó una gráfica comprando los tres métodos realizados



	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

## 1. Broadcast

En la práctica de **Broadcast** se utilizó 10 procesos, en donde el proceso que inicia (Rank=0) es el encargado de realizar las matrices, para que se pueda enviar a los procesos que van a realizar la multiplicación de la matriz, los demás procesos (Rank 1-8) realizan la multiplicación de la matriz y proceden a enviar los datos al proceso (Rank==9) el cual es encargado de unir la matriz resultante y comparar los resultados con la función numpy

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    size_ = (512, 512)
    matriz1 = np.random.randint(10, size=size_).astype("float") / 100
    matriz2 = np.random.randint(10, size=size_).astype("float") / 100
    producto_res = np.dot(matriz1, matriz2)
    variable_to_share = [matriz1, matriz2]

else:
    variable_to_share = 100

def calculoMatriz(pos, matriz):
    filas = np.vsplit(matriz[0], 8)
    columnas = matriz[1]
    fila = filas[pos-1]
    filas_a = len(fila)
    filas_b = len(columnas)
    columnas_a = len(fila[0])
    columnas_b = len(columnas[0])
    size_ = (filas_a, columnas_a)
    calculo = np.zeros(dtype=float, shape=size_)
    for i in range(filas_a):
        for j in range(columnas_b):
            suma = 0
            for k in range(columnas_a):
                suma += fila[i][k] * columnas[k][j]
            calculo[i][j] = suma
    return calculo

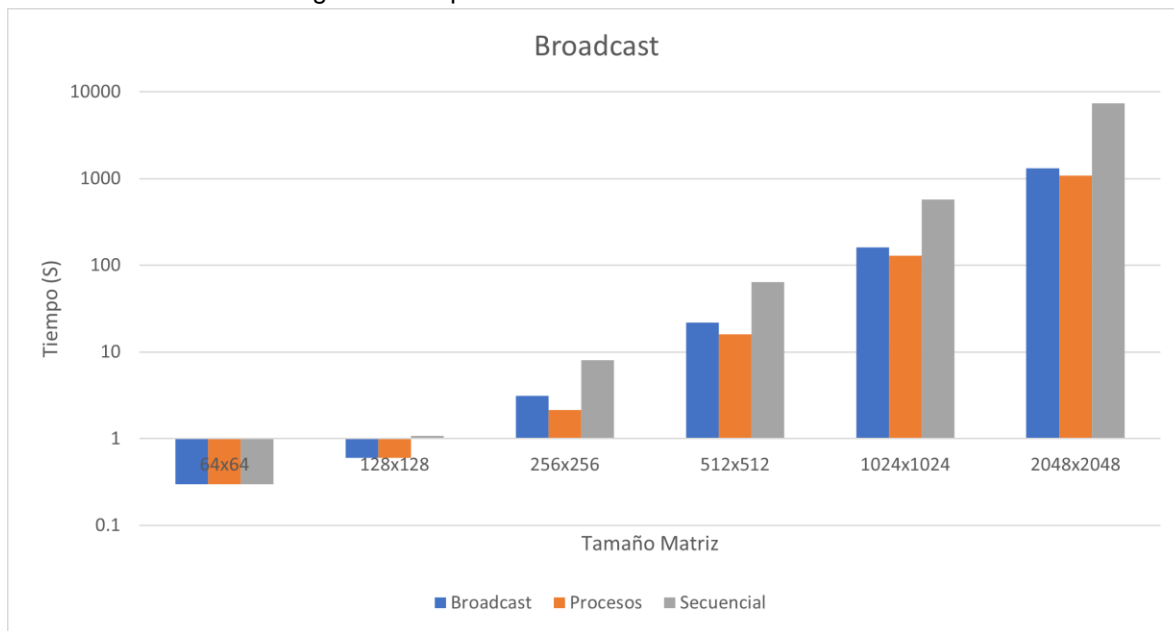
variable_to_share = comm.bcast(variable_to_share, root=0)
for kk in range(1, 9):
    if rank == kk:
        envia1 = calculoMatriz(rank, variable_to_share)
        comm.send(envia1, dest=9)

if rank == 9:
    producto_res = np.dot(variable_to_share[0], variable_to_share[1])
    data2 = list([0, 0, 0, 0, 0, 0, 0, 0, 0])
    for i in range(1, 9):
        data2[i-1] = comm.recv(source=i)
    ListaUnida = np.concatenate((data2[0], data2[1], data2[2], data2[3], data2[4], data2[5], data2[6], data2[7]), axis=0)
    print('¿Los resultados son correctos?', np.allclose(ListaUnida, producto_res))
```

Además, se realizó la comparación de los tiempos de cada una de las matrices solicitadas, con los métodos Secuencial y Paralelo

Matrices	Broadcast	Procesos	Secuencial
64x64	0.3	0.3	0.3
128x128	0.6	0.6	1.08
256x256	3.1	2.14	8
512x512	21.69	16	64
1024x1024	160.2	128.4	573.6
2048x2048	1321.2	1086.6	7,416

Así mismo se realizó una gráfica comprando los tres métodos realizados



**1. Scatter**

2. En la práctica de **Scatter** se utilizó 9 procesos, en donde el proceso que inicia (Rank = 0) es el encargado de realizar las matrices y las divisiones de estas, para que se pueda enviar a los procesos que van a realizar la multiplicación de la matriz, el método Scatter reutiliza el proceso inicial (Rank = 0) por lo cual también realiza la multiplicación de la matriz, la ventaja de este método es que se puede especificar qué datos le envió a cada proceso

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    # tm = int(input("Ingrese el tamaño de la matriz: "))
    size_ = (2048, 2048)
    matriz1 = np.random.randint(10, size=size_).astype("float") / 100
    matriz2 = np.random.randint(10, size=size_).astype("float") / 100
    producto_res = np.dot(matriz1, matriz2)
    filas = np.vsplit(matriz1, 8)
    columnas = matriz2
    array_to_share = [[filas[0], columnas], [filas[1], columnas], [filas[2], columnas], [filas[3], columnas], [filas[4], columnas], [filas[5], columnas], [filas[6], columnas], [filas[7], columnas], [producto_res]]
else:
    array_to_share = None

def calculoMatriz(matrizs):
    filas = matrizs[0]
    columnas = matrizs[1]
    filas_a = len(filas)
    filas_b = len(columnas)
    columnas_a = len(filas[0])
    columnas_b = len(columnas[0])
    size_ = (filas_a, columnas_a)
    calculo = np.zeros(dtype=float, shape=size_)
    for i in range(filas_a):
        for j in range(columnas_b):
            suma = 0
            for k in range(columnas_a):
                suma += filas[i][k] * columnas[k][j]
            calculo[i][j] = suma
    return calculo

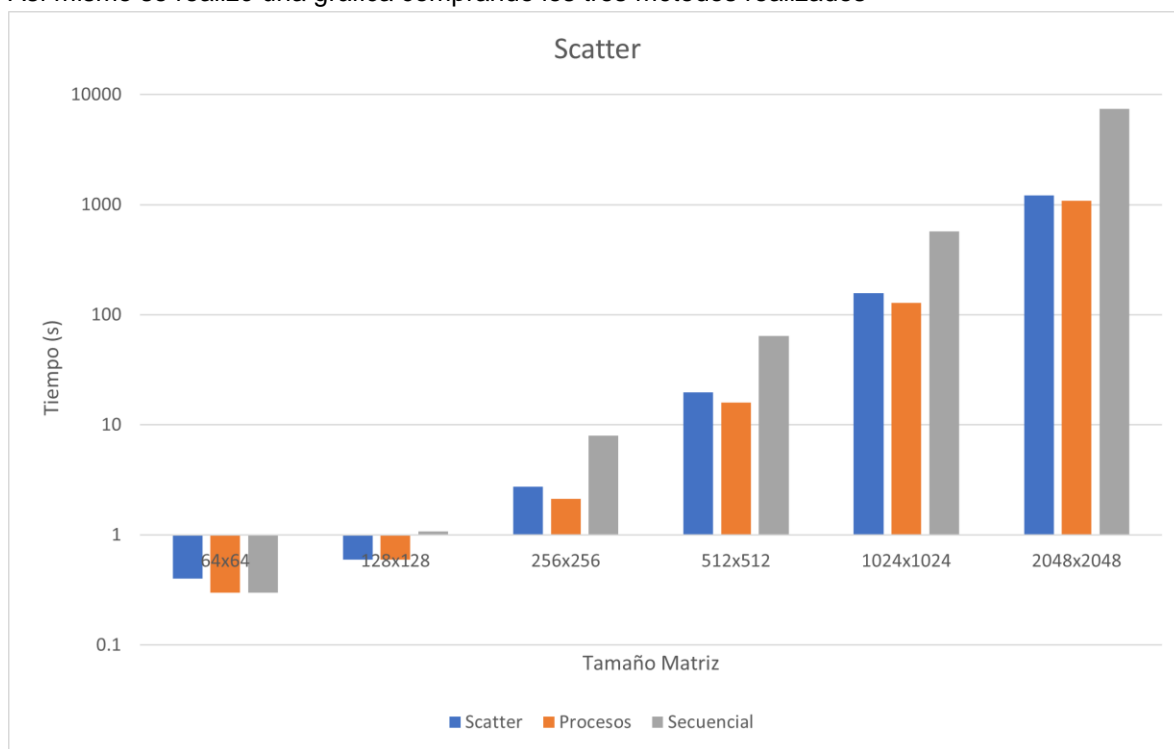
recvbuf = comm.scatter(array_to_share, root=0)
for kk in range(8):
    if rank == kk:
        envia1 = calculoMatriz(recvbuf)
        comm.send(envia1, dest=8)

if rank == 8:
    respuestaNumpy = recvbuf
    data2 = list([0, 0, 0, 0, 0, 0, 0, 0])
    for i in range(8):
        data2[i] = comm.recv(source=i)
    .istaUnida = np.concatenate((data2[0], data2[1], data2[2], data2[3], data2[4], data2[5], data2[6], data2[7]), axis=0)
    print('¿Los resultados son correctos?', np.allclose(ListaUnida, respuestaNumpy))
```

Se realizó la comparación de los tiempos de cada una de las matrices solicitadas, con los métodos Secuencial y Paralelo

Matrices	Scatter	Procesos	Secuencial
64x64	0.4	0.3	0.3
128x128	0.6	0.6	1.08
256x256	2.75	2.14	8
512x512	19.78	16	64
1024x1024	157.8	128.4	573.6
2048x2048	1210.2	1086.6	7,416

Así mismo se realizó una gráfica comprando los tres métodos realizados



## 1. Gather

En la práctica de **Gather** se utilizó 8 procesos, en donde el proceso que inicia (Rank=0) es el encargado de realizar las matrices y las divisiones de estas, a su vez también realiza el proceso de multiplicación de la matriz, los demás procesos (Rank 1-7) realizan la multiplicación de la matriz y proceden a enviar los datos al proceso (Rank==0) va a recibir los datos de todos los demás parámetros

```
from mpi4py import MPI
import time
import numpy as np

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

def calculoMatriz(pos,matriz):
    filas = np.vsplit(matriz[0], 8)
    columnas = matriz[1]
    fila = filas[pos]
    filas_a = len(fila)
    filas_b = len(columnas)
    columnas_a = len(fila[0])
    columnas_b = len(columnas[0])
    size_ = (filas_a, columnas_a)
    calculo = np.zeros(dtype=float, shape=size_)
    for i in range(filas_a):
        for j in range(columnas_b):
            suma = 0
            for k in range(columnas_a):
                suma += fila[i][k] * columnas[k][j]
            calculo[i][j] = suma
    return calculo


data = 0

if rank == 0:
    size_ = (64, 64)
    matriz1 = np.random.randint(10, size=size_).astype("float") / 100
    matriz2 = np.random.randint(10, size=size_).astype("float") / 100
    matrices = [matriz1,matriz2]
    data = calculoMatriz(rank,matrices)
    for kk in range(1, 8):
        comm.send(matrices, dest=kk)

for mt in range(1, 8):
    if rank == mt:
        datos = comm.recv(source=0)
        data = calculoMatriz(mt,datos)

data2 = comm.gather(data, root=0)
if rank == 0:
    producto_res = np.dot(matriz1, matriz2)
    paralela_res = np.reshape(data2, size_)
    print('¿Los resultados son correctos?', np.allclose(paralela_res,
    producto_res))
```

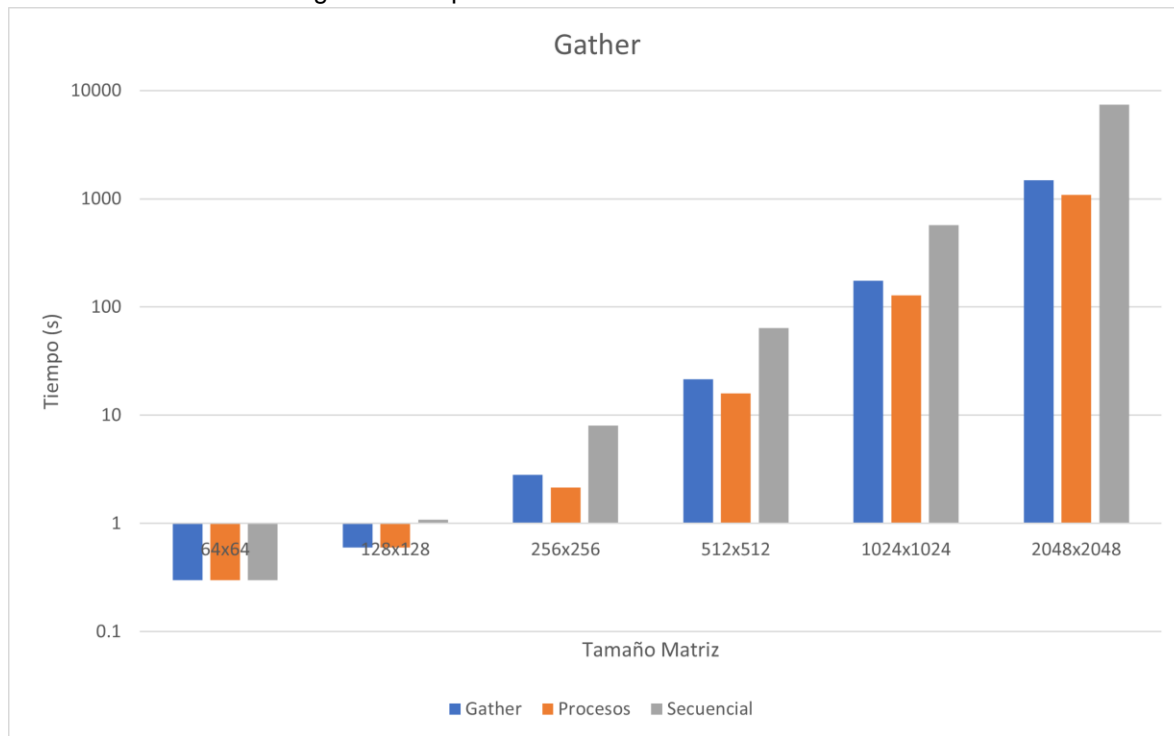


	<b>VICERRECTORADO DOCENTE</b>	<b>Código:</b> GUIA-PRL-001
	CONSEJO ACADÉMICO	<b>Aprobación:</b> 2016/04/06
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Se realizó la comparación de los demás métodos

Matrices	Gather	Procesos	Secuencial
64x64	0.3	0.3	0.3
128x128	0.6	0.6	1.08
256x256	2.81	2.14	8
512x512	21.59	16	64
1024x1024	174.6	128.4	573.6
2048x2048	1485	1086.6	7,416

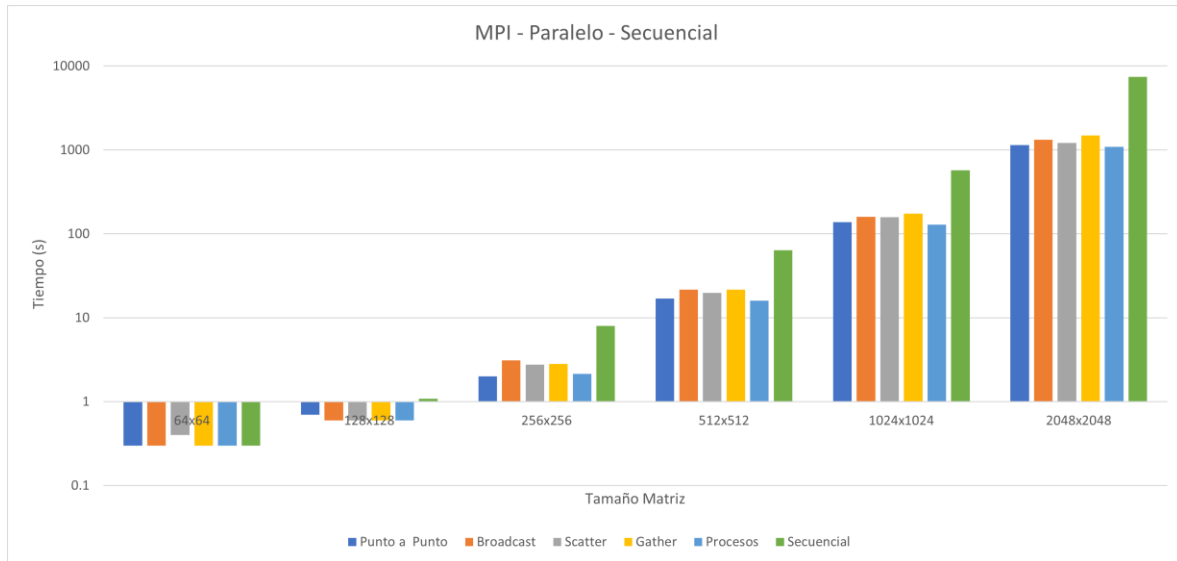
Así mismo se realizó una gráfica comprando los tres métodos realizados



# 1. Tabla con todos los métodos realizados

Matrices	Punto a Punto	Broadcast	Scatter	Gather	Procesos	Secuencial
64x64	0.3	0.3	0.4	0.3	0.3	0.3
128x128	0.7	0.6	0.6	0.6	0.6	1.08
256x256	2.00	3.1	2.75	2.81	2.14	8
512x512	17.00	21.69	19.78	21.59	16	64
1024x1024	138.00	160.2	157.8	174.6	128.4	573.6
2048x2048	1146.00	1321.2	1210.2	1485	1086.6	7,416

## 2. Grafica de los métodos realizados




### RESULTADO(S) OBTENIDO(S):

- Al realizar los diferentes métodos y obtener los resultados observamos los métodos MPI es muy buenos, pero el método de Punto a Punto es mucho más rápido, en cambio el secuencial, se demora demasiado.
- El mejor método que se pudo obtener es el de Procesos, teniendo los tiempos más bajos según las gráficas obtenidas
- Podemos ver que el mejor de los algoritmos en MPI para nuestro caso es de Punto a Punto ya que podemos ver que su tiempo de ejecución es menor, este teniendo en cuenta el número de elementos de la matriz, ya que mientras más grande sea la matriz se puede notar la diferencia de tiempo en comparación de la programación secuencias
- El proceso que es más demorado es el Gather en la parte de MPI, además, vemos que la diferencia de tiempos no es significativa con excepción del algoritmo secuencial

### CONCLUSIONES:

- El computo paralelo nos ayuda a realizar operaciones matemáticas más rápidamente, especialmente en operaciones donde se puede dividir el cálculo, sin embargo podemos ver que mientras más grande sean nuestros datos más se lograra ver una diferencia en el cálculo, es decir, un arreglo o matriz con pocas dimensiones dará como resultado un tiempo similar en computación paralela y secuencial, pero a medida que se agranda el número de datos se puede apreciar más la diferencia dando como resultado un tiempo menor de cálculo el computo paralelo
- Mediante el computo paralelo podemos ver el veneficio de usar mpic4py, es decir que dentro los posibles algoritmos esta, es uno de los las eficientes

	<b>VICERRECTORADO DOCENTE</b>	<b>Código:</b> GUIA-PRL-001
	CONSEJO ACADÉMICO	<b>Aprobación:</b> 2016/04/06
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

#### RECOMENDACIONES:

- Realizar la multiplicación de matrices con tiempo, puesto que, si se toma su tiempo y es muy demoroso, especialmente el secuencial que tardo alrededor de dos horas
- Se recomienda utilizar un solo sistema operativo, puesto que, por ovias razones, en el sistema operativo ciertos procesos varían en tiempo al momento de ejecutarse
- Se recomienda utilizar un solo sistema operativo, puesto que, por ovias razones, en el sistema operativo ciertos procesos varían en tiempo al momento de ejecutarse

**Nombre de estudiante:** Adrian Angamarca, Miguel Samaniego

**Firma de estudiante:** \_\_\_\_\_