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

		PRÁCTICA DE LABORATORIO	
CARRERA: INGENIERIA DE SISTEMAS/COMPUTACION		ASIGNATURA: Computo Paralelo	
NRO. PRÁCTICA:	0	TÍTULO PRÁCTICA: Examen Interciclo	
OBJETIVO Implementación de MPI			
INSTRUCCIONES		<ol style="list-style-type: none"> 1. Crear un script en Python utilizando spyder o jupyter e identificarlo con sus nombres completos, en el siguiente formato: <i>Apellidos_Nombre_Examen</i>. Por ejemplo: <i>LeonParedes_Gabriel_Examen.py</i>. 2. Desarrollar el examen. 3. Subir al AVAC: <ol style="list-style-type: none"> a. Informe detallado de la resolución del examen en formato .pdf b. Script de python en formato (.py o .ipynb) 	

DESARROLLO
<p>Enunciado</p> <p>El siguiente código secuencial implementa el producto de una matriz B de dimensión $N \times N$ por un vector c de dimensión N.</p> <pre>void prodmv(double a[N], double c[N], double B[N][N]) { int i, j; double sum; for (i=0; i<N; i++) { sum = 0; for (j=0; j<N; j++) sum += B[i][j] * c[j]; a[i] = sum; } }</pre> <p>Implementación del algoritmo secuencial:</p> <pre>import numpy as np import time from random import randint</pre>

```
#import sys
N = 1000
A = np.empty(N)
B = np.empty((N,N))
C = np.empty(N)
start = time.time()
for i in range(N):
    for j in range(N):
        B[i][j]= randint(1, 50)
        C[j]= randint(1, 50)
    for i in range(N):
        sum = 0
        for j in range(N):
            sum += B[i][j] * C[j]
        A[i] = sum

print("Resultado: A", A)
print("Tiempo total: ", (time.time() - start))
```

Implementación del algoritmo con MPI:

Este algoritmo cuenta de 5 funciones las cuales realizan diferentes funciones y está basado en el algoritmo de MPI de Scatter para poder dividir la matriz de NxN en un vector con N arreglos y poder mandar una a una cada fila a los diferentes procesos:

En la primera función se procede a generar datos aleatorios en base a una entrada de usuario y se van llenando la matriz B y los vectores C y A, de igual manera se especifica que los valores a generar serán en el rango de 1-50.

```
def inicializarMatrices():
    global A
    global B
    global C
    for i in range(N):
        for j in range(N):
            B[i][j]= randint(1, 50)
            C[j]= randint(1, 50)
```

Para realizar el producto de las matrices se hace uso de la lógica del algoritmo secuencial presentado por el docente, esta función retorna:

```
def productoMatrices(A,c,b):
    for i in range(len(b)):
        sum = 0.0
        for j in range(len(c)):
```

```

        sum+=b[i][j] * c[j]
    A[i]= sum
    return A

```

Para la división de la matriz es importante que N sea un numero divisible para la cantidad de procesos determinados al momento de ejecutar mpiexec, la función dividir toma la matriz $B = [N][N]$ y guarda cada fila de la matriz B en un vector de filas el cual es retornado por esta función:

```
def dividir(mat, t):
```

```

    filas = []
    n = len(mat) / t
    r = len(mat) % t
    B, e = 0, n + min(1, r)
    for i in range(t):
        filas.append(mat[(int)(B):(int)(e)])
        r= max(0, r -1)
        B,e=e, e+n+min(1, r)

    return filas

```

Dentro de la función distribuirMatriz() se procede a realizar un scatter en el cual se envia cada arreglo del vector de filas a los procesos, y al hacer esto se incrementa el id del proceso, de igual manera se realiza el proceso de scatter para el vecto $C=[N]$:

```

#id proceso, aqui se van generando nuevos procesos dependiendo de la
# cantidad de filas
pid = 1
#Gather
for f in filas:
    #aqui se envia cada fila por separado a los procesos
    comm.send(f, dest=pid, tag=1)
    comm.send(C, dest=pid, tag=2)
    pid = pid+1

```

Se define una función que sería la función Amo, procesoPrincipal(), la cual se encarga de realizar la distribución de la matriz:

```
def procesoPrincipal():
    distribuirMatriz()
```

En cuanto a la función secundaria o esclava, en esta definimos los procesos y los datos que reciben para realizar el procesamiento, y también un vector el cual almacenara los resultados:

```
def procesosSecundarios():

    b = comm.recv(source = 0, tag = 1)
    c = comm.recv(source = 0, tag = 2)

```

```
a = productoMatrices(A, c, b)
```

```
comm.send(a, dest = 0, tag = rank )
```

Resultados obtenidos:

Proceso realizado con 1000 datos:

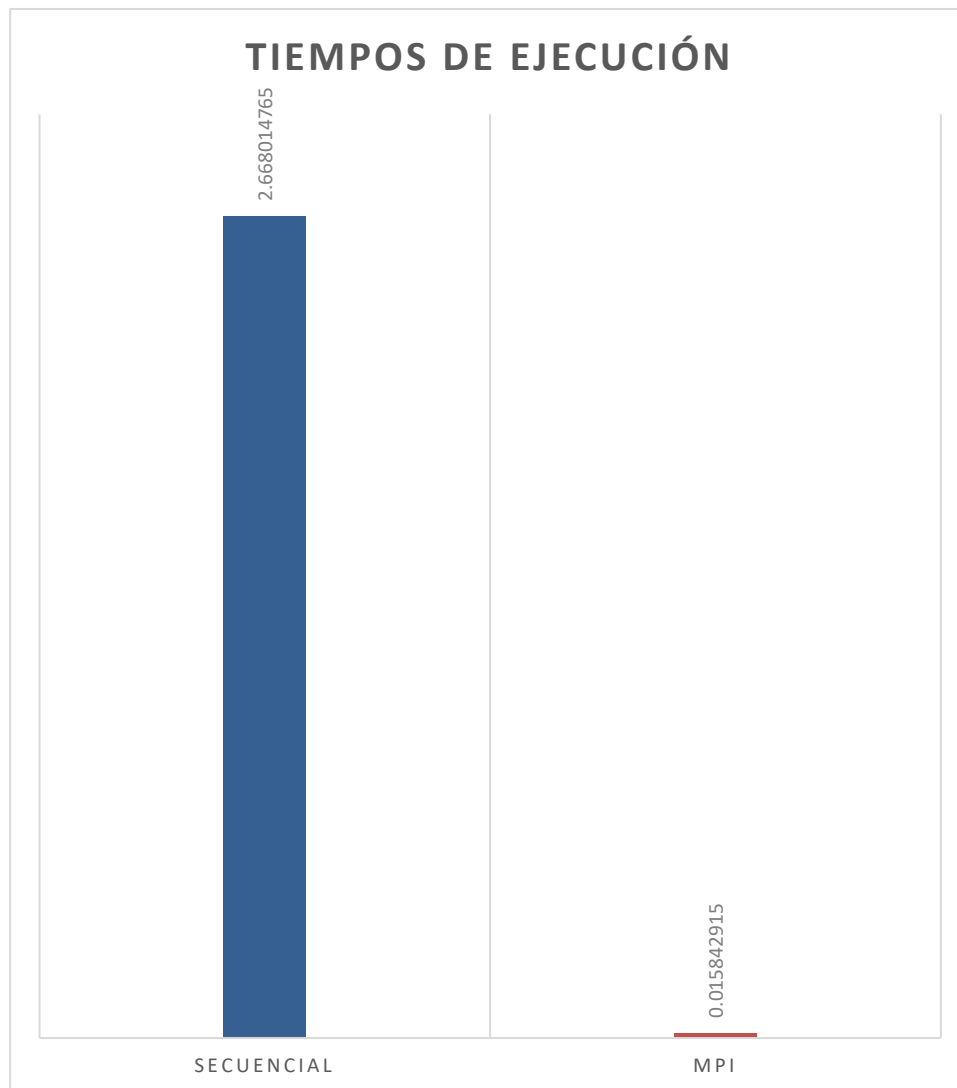


Ilustración 1 Tiempos de ejecución

Cálculos:

Tiempo ejecución algoritmo secuencial = 2.66801476478576

Tiempo ejecución algoritmo MPI = 0.0158429145812988

Eficiencia = (Tiempo ejecución algoritmo secuencial) / (Tiempo ejecución algoritmo MPI)

$$E = \frac{2.66801476478576}{0.0158429145812988}$$
$$E = 168.404289$$


Acelereación = (Tiempo ejecución algoritmo secuencial) / (núcleos o procesos) * (tiempo ejecución algoritmo MPI)

$$S = \frac{(2.66801476478576)}{(4) * (0.0158429145812988)}$$
$$S = 0.01056728$$

Conclusiones:

En este caso podemos notar que se ha obtenido una Eficiencia $E < 1$ por lo que se puede clasificar a este problema como un caso real mas no un linear, esto se puede deber a la implementación al generar los numeros aleatorios de las matrices y vectores.

MPI resulta ser un gran aliado al momento de procesar grandes volúmenes de datos, esto al permitir descomponer los problemas en porciones menores para ser asignados a diferentes procesos distribuidos.

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

RESULTADO(S) OBTENIDO(S):

- Implementar algoritmos de computación distribuida.

CONCLUSIONES:

- MPI es mucho mas eficiente al momento de procesar grandes niveles de información.

RECOMENDACIONES:

- Tomar los tiempos de ejecución.

Docente: Ing. Gabriel Leon, PhD.

Estudiante: Alex Reinoso

Firma: _____

Firma:  _____