
	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:		TÍTULO PRÁCTICA: Examen de recuperación practico	
OBJETIVO Implementación de multiplicación de matriz por vector utilizando el módulo multiprocessing.			
INSTRUCCIONES		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: a) Informe detallado de la resolución del examen en formato .pdf b) Script de python en formato (.py o .ipynb)	

DESARROLLO	
<p>Enunciado. 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 align="center"><i>Ilustración 1 Enunciado del algoritmo</i></p> <p>Implementación del algoritmo secuencial:</p>	

```
import numpy as np
import time
from random import randint
from numpy import savetxt

N = 160
A = np.random.randint(60, size=N)
B = np.random.randint(60, size=(N,N))
C = np.random.randint(60, size=N)

start = time.time()
for i in range(N):
    sum = 0
    for j in range(N):
        sum += B[i][j] * C[j]
    A[i] = sum

print("Tiempo Secuencial: ", (time.time() - start))
savetxt('resultadoSecuencial.csv', A, delimiter=' ')
```

implementación algoritmo con procesos:

Para poder realizar procesos con multiprocesos es indispensable importar el módulo `multiprocessing` y de este también `Process` para poder ejecutar la multiplicación usando diferentes procesos.

Primeramente, importamos el módulo `numpy` para poder generar números aleatorios y llenar la matriz y los vectores.

- N: almacena la dimensión de la matriz y los vectores
- A: vector en donde se guardará el resultado de la multiplicación de la matriz B por el vector C
- B: matriz de NxN.
- C: Vector de tamaño N.

```
N = int(16000)
A = np.random.randint(60, size=N)
B = np.random.randint(60, size=(N,N))
C = np.random.randint(60, size=N)
```

El algoritmo cuenta con 2 funciones, la primera función recorre la matriz hasta la mitad y realiza el proceso de multiplicar la mitad de la matriz B por la mitad del vector C:

```
def multiplicar(B, C, N1, N2):
    for i in range(int(N1), int(N2)):
        sum2 = 0
        for j in range(int(N1), int(N2)):
            sum2 += B[i][j] * C[j]
```

```
A[i] = sum2
```

A esta función le pasamos 4 parámetros:

1. Matriz B
2. Matriz C
3. N1
4. N2

Por último, generamos 5 procesos por separado a los cuales les asignamos la función `multiplicar()` para multiplicar la matriz B por el vector C, y le definimos un punto de inicio y un punto final.

Para esto definimos un proceso `p1` hasta `p5` de tipo `Process`, y le asignamos como `target` a la función `multiplicar` y como `args` le pasamos la matriz B, vector C y el valor de N1 y N2 que serán el inicio y el final de los bucles, como en este caso queremos dividir la matriz en 5 partes iguales, entonces al primer proceso se le asigna $N1 = 0$ y $N2 = N * (\frac{1}{5})$, al segundo proceso $N1 = N * (\frac{1}{5})$ y $N2 = N * (\frac{2}{5})$ y así consecutivamente hasta llegar a $N2 = N$.

Luego mandamos a ejecutar los procesos con los métodos `start()` y `join()` para terminar los procesos una vez han realizado las operaciones de multiplicar la matriz B por el vector C.

```
if __name__ == '__main__':
    tInit = time.time()

    p1 = Process(target=multiplicar, args=(B, C, int(0),
int(N*(1/5),) ))
    p2 = Process(target=multiplicar, args=(B, C, int(N*(1/5)),
int(N*(2/5),) ))
    p3 = Process(target=multiplicar, args=(B, C, int(N*(2/5)),
int(N*(3/5),) ))
    p4 = Process(target=multiplicar, args=(B, C, int(N*(3/5)),
int(N*(4/5),) ))
    p5 = Process(target=multiplicar, args=(B, C, int(N*(4/5)),
int(N), ))

    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()

    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

    print("Tiempo procesos: ", (time.time()-tInit))
```

```
print("N: ", N)
print('processor count: ', multiprocessing.cpu_count())
savetxt('resultadoOperaciones.csv', A, delimiter='[]')
```

Comparativas de tiempo de ejecución:

Las comparativas de tiempo se las hizo en milisegundos haciendo uso del módulo time.

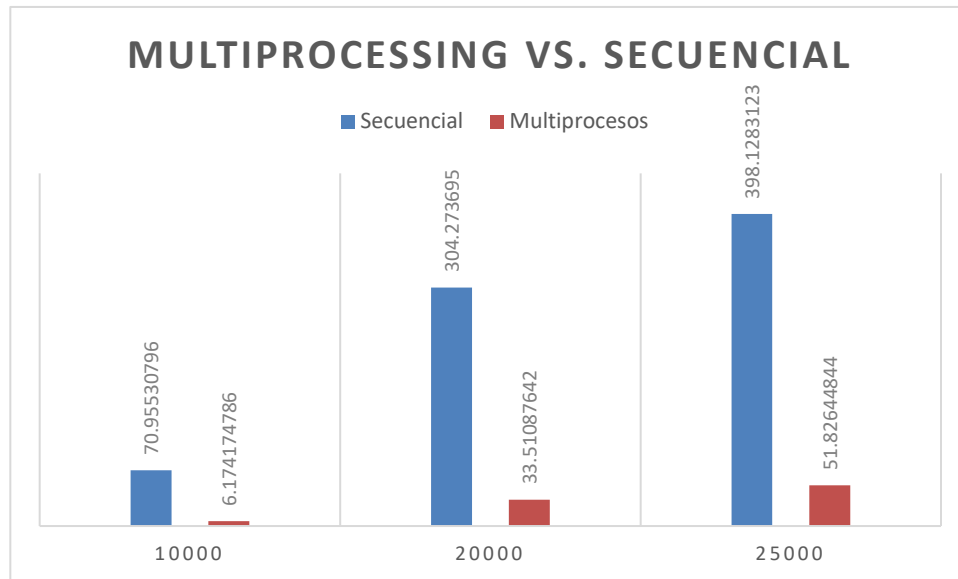


Ilustración 2 Comparativas tiempos de ejecución

Cálculos

Aceleración: $S = \frac{T_s}{T_p}$

Con N = 10,000:

$$T_s = 70.95531$$

$$T_p = 6.17417478561401$$

$$S = \frac{70.95531}{6.17417478561401} = 3.9339231689272567$$

Con N = 20,000:

$$T_s = 304.273694992065$$

$$T_p = 33.51087641716$$

$$S = \frac{304.273694992065}{33.51087641716} = 9.079848918432189$$

Con N = 25,000:

$$Ts = 398.128312349319$$

$$Tp = 51.8264484405517$$

$$S = \frac{398.128312349319}{51.8264484405517} = 7.681952445689927$$

Eficiencia: $E = \frac{Ts}{pTp}$

Con N = 10,000:

$$P = 12$$

$$Ts = 70.95531$$

$$Tp = 6.17417478561401$$

$$S = \frac{70.95531}{(12 * 6.17417478561401)} = 0.9576895221329516$$

Con N = 20,000:

$$P = 12$$

$$Ts = 304.273694992065$$

$$Tp = 33.51087641716$$

$$S = \frac{304.273694992065}{(12 * 33.51087641716)} = 0.7566540765360158$$

Con N = 25,000:

$$P = 2$$

$$Ts = 398.128312349319$$

$$Tp = 51.8264484405517$$

$$S = \frac{398.128312349319}{(12 * 51.8264484405517)} = 0.6401627038074938$$

Conclusiones:

En este caso se ha obtenido una eficiencia $E < 1$ por lo que se puede clasificar a este problema como un caso real mas no lineal, el caso que más se acercó a 1 es cuando se probó con un $N = 10,000$.

RESULTADO(S) OBTENIDO(S):


- Implementar un algoritmo por procesos y comparar sus ventajas sobre algoritmos secuenciales.

CONCLUSIONES:

- El algoritmo por procesos es significativamente más eficiente y veloz al momento de procesar grandes volúmenes de información al compararlo con un algoritmo secuencial

RECOMENDACIONES:


- Revisar el contenido teórico de la materia.
- Medir tiempos de ejecución.

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

Docente: Ing. Gabriel León, PhD.

Estudiante: Alex Reinoso

Firma: _____

Firma:  _____