



Computación Paralela

Alumna:

Esther Milagros Bautista Peralata

Semestre: *VIII - semestre*

Docente:	<i>Fred Torres Cruz</i>
Facultad:	Ingeniería Estadística e Informática
Año académico:	October 17, 2022- II

Contents

1	Trabajo Práctico - N° 001	1
2	Código en Python	1
3	Capturas de los resultados	2
4	Interpretación de resultados	5
5	Evidencia del trabajo en LATEX	5

1 Trabajo Práctico - N° 001

Implementar según el ejemplo de clase una función `numPar` que reciba un parámetro `n` y calcule el números pares entre 1 y `n`, así mismo realizar el cálculo de tiempo para la evaluación de la implementación, para hacer el llamado a la función se deberá hacer a través de uno o más threads.

Las pruebas unitarias deberán ser con los siguientes parámetros a la función `numPar`:

- * `n = 10`
- * `n = 30`
- * `n = 200`
- * `n = 5030`

2 Códido en Python

El código se ejecutó en `GOOGLE COLAB` utilizando un thread para imprimir números pares para `n` datos.

```
from threading import Thread
import time

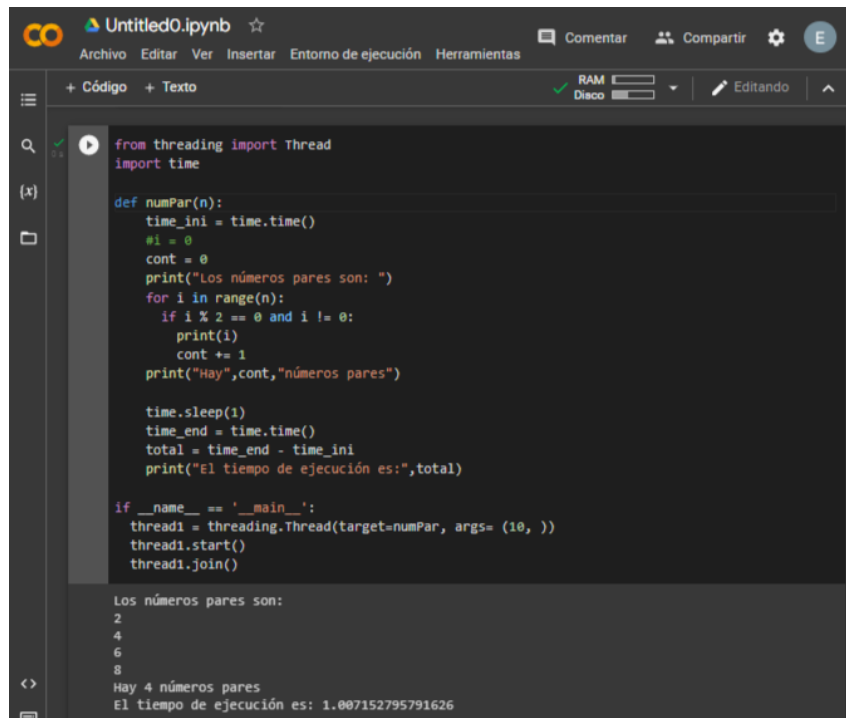
def numPar(n):
    time_ini = time.time()
    cont = 0
    print("Los números pares son: ")
    for i in range(n):
        if i % 2 == 0 and i != 0:
            print(i)
            cont += 1
    print("Hay",cont,"números pares")

    time.sleep(1)
    time_end = time.time()
    total = time_end - time_ini
    print("El tiempo de ejecución es:",total)

if __name__ == '__main__':
    thread1 = threading.Thread(target=numPar, args= (10, ))
    thread1.start()
    thread1.join()
```

3 Capturas de los resultados

Se muestra capturas de pantallas de los resultados del código para $n = 10$, $n = 30$, $n = 200$ y $n = 5030$.



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
from threading import Thread
import time

def numPar(n):
    time_ini = time.time()
    #i = 0
    cont = 0
    print("Los números pares son: ")
    for i in range(n):
        if i % 2 == 0 and i != 0:
            print(i)
            cont += 1
    print("Hay",cont,"números pares")

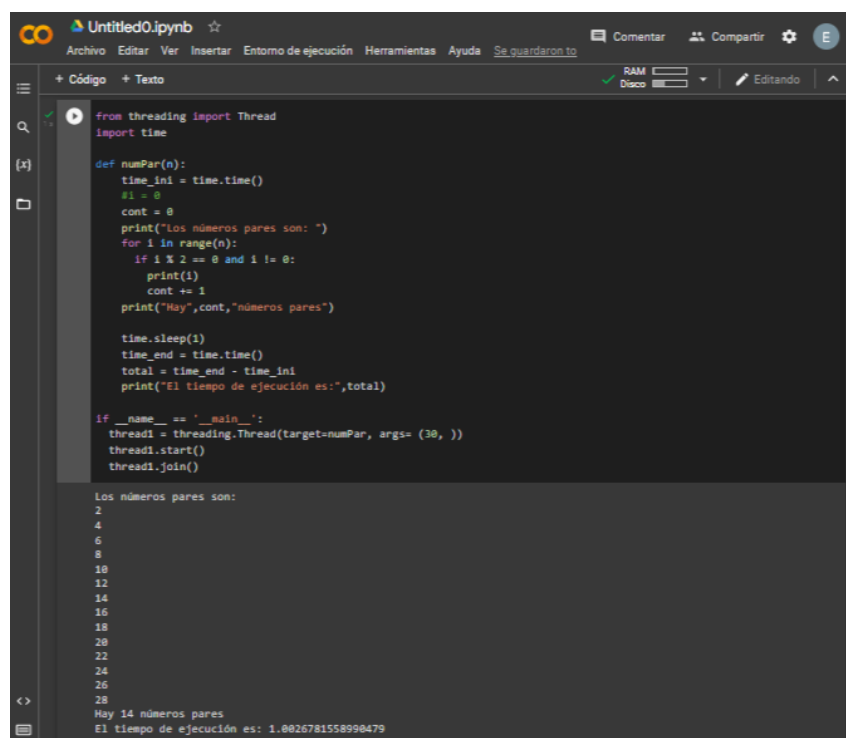
    time.sleep(1)
    time_end = time.time()
    total = time_end - time_ini
    print("El tiempo de ejecución es:",total)

if __name__ == '__main__':
    thread1 = threading.Thread(target=numPar, args= (10, ))
    thread1.start()
    thread1.join()
```

Output:

```
Los números pares son:
2
4
6
8
Hay 4 números pares
El tiempo de ejecución es: 1.007152795791626
```

Figure 1: Ejecutando el código con 10 datos



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
from threading import Thread
import time

def numPar(n):
    time_ini = time.time()
    #i = 0
    cont = 0
    print("Los números pares son: ")
    for i in range(n):
        if i % 2 == 0 and i != 0:
            print(i)
            cont += 1
    print("Hay",cont,"números pares")

    time.sleep(1)
    time_end = time.time()
    total = time_end - time_ini
    print("El tiempo de ejecución es:",total)

if __name__ == '__main__':
    thread1 = threading.Thread(target=numPar, args= (30, ))
    thread1.start()
    thread1.join()
```

Output:

```
Los números pares son:
2
4
6
8
10
12
14
16
18
20
22
24
26
28
Hay 14 números pares
El tiempo de ejecución es: 1.0026781558998479
```

Figure 2: Ejecutando el código con 20 datos

The figure consists of two screenshots of a Jupyter Notebook interface, showing the execution of a Python program that counts even numbers in parallel using threads.

Top Screenshot: The code defines a function `numPar(n)` that counts even numbers from 0 to `n-1` using a loop and a thread. The main function calls `numPar(200)` and prints the total execution time. The output shows the first 10 even numbers: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34.

```
from threading import Thread
import time

def numPar(n):
    time_ini = time.time()
    #i = 0
    cont = 0
    print("Los números pares son: ")
    for i in range(n):
        if i % 2 == 0 and i != 0:
            print(i)
            cont += 1
    print("Hay", cont, "números pares")

    time.sleep(1)
    time_end = time.time()
    total = time_end - time_ini
    print("El tiempo de ejecución es:", total)

if __name__ == '__main__':
    thread1 = threading.Thread(target=numPar, args= (200, ))
    thread1.start()
    thread1.join()
```

Los números pares son:
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34

Bottom Screenshot: The output continues with the next set of even numbers: 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198. The final output shows "Hay 99 números pares" and "El tiempo de ejecución es: 1.0114798545837482".

```
120  
122  
124  
126  
128  
130  
132  
134  
136  
138  
140  
142  
144  
146  
148  
150  
152  
154  
156  
158  
160  
162  
164  
166  
168  
170  
172  
174  
176  
178  
180  
182  
184  
186  
188  
190  
192  
194  
196  
198  
Hay 99 números pares  
El tiempo de ejecución es: 1.0114798545837482
```

Figure 3: Ejecutando el código con 200 datos

The figure consists of two screenshots of a Jupyter Notebook interface, showing the execution of a Python script. The notebook is titled "Untitled0.ipynb".

Top Screenshot: The code cell contains the following Python code:

```
from threading import Thread
import time

def numPar(n):
    time_ini = time.time()
    #i = 0
    cont = 0
    print("Los números pares son: ")
    for i in range(n):
        if i % 2 == 0 and i != 0:
            print(i)
            cont += 1
    print("Hay", cont, "números pares")

    time.sleep(1)
    time_end = time.time()
    total = time_end - time_ini
    print("El tiempo de ejecución es:", total)

if __name__ == '__main__':
    thread1 = threading.Thread(target=numPar, args= (5030, ))
    thread1.start()
    thread1.join()
```

The output cell shows the start of the execution, with line numbers 4918 through 4950 visible.

Bottom Screenshot: The code cell is the same as in the top screenshot. The output cell shows the completion of the execution, with line numbers 4948 through 5028 visible. The final output is:

```
Hay 2514 números pares
El tiempo de ejecución es: 1.7799322605133057
```

Figure 4: Ejecutando el código con 5030 datos

4 Interpretación de resultados

Cuando ejecutamos el código en Google Colab con 10 datos podemos ver en la (Figura 1) que este demora 1.0071527... segundos, cuando ejecutamos el mismo código esta vez con 30 datos vemos en la (Figura 2) que tarda en ejecutarse 1.0026781... segundos, volvemos a ejecutar el mismo código esta vez con 200 datos y vemos en la (Figura 3) el resultado de 1.0114798... segundos y por último ejecutamos el código con 5030 datos y podemos ver demora 1.7799322... segundos en la (Figura 4).

Por otro lado, un thread es la unidad más pequeña a la cual un procesador puede asignar tiempo. Los threads poseerán la secuencia más pequeña de instrucciones a ejecutar. Los threads se crean, ejecutan y mueren dentro de los procesos, siendo capaces de compartir información entre ellos.

Con los threads y los procesos seremos capaces de implementar la programación concurrente, y, dependiendo de la cantidad de procesadores la programación en paralelo.

5 Evidencia del trabajo en LATEX

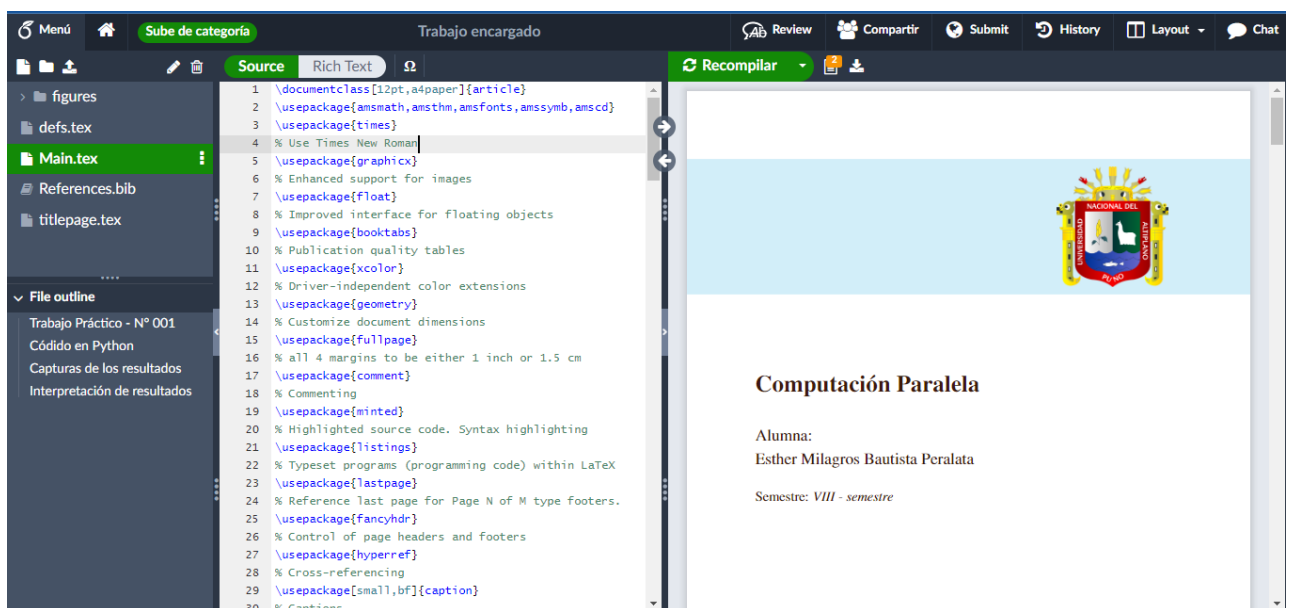


Figure 5: <https://es.overleaf.com/read/rnmospfhrdgc>