

# Comparativa de rendimiento entre los lenguajes de programación Python y Cython haciendo uso de nodos de cómputo por medio de aplicaciones de juguete (Benchmarks)

Oscar Julián Reyes Torres  
Universidad Sergio Arboleda.  
Bogotá, Colombia  
oscar.reyes0101@correo.usa.edu.co

**Resumen**—In this report we want to compare the performance of two programming languages that are Python and Cython by means of experimental batteries that simulate the two-dimensional heat equations in different physical bodies, in this case bottles of different sizes, buying the execution time and through the Cython programming language optimize this process by making use of the most available resources on each compute node.

Later, the decrease in the compilation load will be verified, given the use of languages such as Cython, which at the machine level is much more efficient than Python, but the learning curve is a little slower compared to the interpreted language such as Python.

**Index Terms**—Cython, Python, Radiación, Térmico, Calor y Comparativa.

## I. OBJETIVO

Comparar de forma exhaustiva el rendimiento de diferentes nodos de cómputo aprovechando el rendimiento que ofrece el lenguaje de programación Cython frente a Python en cada arquitectura.

## II. INTRODUCCIÓN

En este informe se desea comparar el rendimiento de dos lenguajes de programación que son Python y Cython por medio de baterías experimentales que simulan las ecuaciones de calor bidimensional en diferentes cuerpos físicos, en este caso botellas de diferentes tamaños comprando el tiempo de ejecución y a través del lenguaje de programación cython optimizar este proceso haciendo uso de la mayoría de recursos disponibles en cada nodo de cómputo.

Posteriormente se verificará la disminución en la carga de compilación, dado el uso de lenguajes como cython el cual a nivel de maquina es mucho más eficiente que Python, pero la curva de aprendizaje es un poco más lenta en comparación al lenguaje interpretado como lo es Python.

## III. FUNDAMENTOS

### Python

Python es un lenguaje de scripting (son lenguajes que se usan para satisfacer rápidamente las exigencias más comunes de los usuarios) y orientado a objetos. Además, está preparado para poder crear cualquier tipo de programa para Windows, servidores de red, páginas web, entre otros.

Una de las grandes ventajas es que es un lenguaje interpretado, lo que significa que no hay necesidad de compilar el código para poder ejecutarlo, y esto trae consigo rapidez en el desarrollo de los programas.

Actualmente, es uno de los lenguajes más populares debido a razones como:

- La gran variedad de bibliotecas que tiene datos y funciones propias del lenguaje, lo que ayuda al programador a realizar tareas habituales sin la necesidad de tener que programarlas él mismo.
- Un programa realizado en python puede tener de 3 a 5 líneas de código menos que lenguajes como Java o C. Se puede desarrollar en plataformas como Unix, Windows, OS/2, Mac, entre otros.
- Es totalmente gratuito, ya sea para entornos personales o empresariales.
- Dispone de un intérprete por línea de comandos, en donde se pueden introducir sentencias y cada una de estas sentencias se ejecutan y producen un resultado visible para el programador, lo que ayuda a entender mejor el lenguaje y probar partes del código antes de ejecutarlo completamente.
- Permite utilizar programas de orientados a objetos con componentes reutilizables.
- Es un lenguaje sencillo, legible, simplificado, rápido, ordenado, portable, flexible y elegante que está bajo un conjunto de reglas no tan estrictas, que hacen muy corta su curva de aprendizaje.

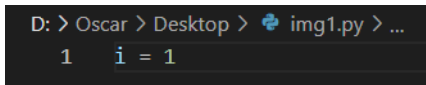
### Cython

Cython es un lenguaje de programación (conocido como un

superconjunto de Python) que une Python con C y C++, es decir, Cython es un compilador que traduce código fuente escrito en Python en eficiente código C o C++, con esto se busca aprovechar las fortalezas de Python y C y así combinar una sintaxis con el poder y la velocidad, lo que produce aumentos de rendimiento que pueden ir desde un pequeño porcentaje hasta varios órdenes de magnitud, dependiendo de la tarea en cuestión. Si se trata de programas que están limitados por los tipos de objetos nativos de Python, el aumento de rendimiento no será de gran escala. Sin embargo, si hay operaciones numéricas, o programas que no tengan objetos nativos de Python, el rendimiento puede ser bastante alto.

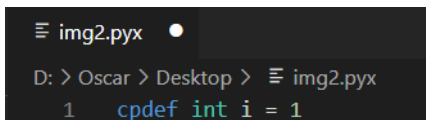
Utilizando Cython puede evitar muchas de las limitaciones que tiene Python sin tener que renunciar a la simplicidad y comodidad de Python.

Para traducir nuestro código de Python a C, primero se debe hacer la instalación de Cython en la máquina, para esto se debe poner el comando: *pip install cython*, si es el caso de usar pip3 entonces usar: *pip3 install cython*. Luego, es necesario hacer el ajuste en el programa, que es indicar el tipo de dato de todas las variables y funciones. Para hacerlo se utiliza la instrucción *def* (se usa desde Python), *cdef* (se usa desde Python y Cython) o *cpdef* (se usa desde Python y Cython) seguida del tipo de dato como se evidencia a continuación:



```
D: > Oscar > Desktop > img1.py > ...
1   i = 1
```

Figura 1: Declaración de una variable en Python

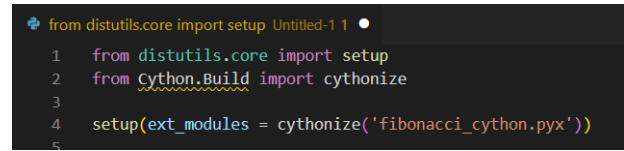


```
img2.pyx
D: > Oscar > Desktop > img2.pyx
1   cpdef int i = 1
```

Figura 2: Declaración de una variable en Cython

Hay que tener en cuenta los tipos de datos soportados por Cython, los estándar de C/C++: *int*, *char*, *float*, *double*, *list*, *dict* y *object*.

Ese nuevo código con el ajuste hecho se debe guardar en un archivo con extensión pyx como por ejemplo fibonacci.pyx. Después de guardar el archivo pyx, se debe hacer un script en Python para compilar el código, poniendo el nombre del archivo. Por ejemplo, en la siguiente imagen usando la serie de Fibonacci.



```
from distutils.core import setup
1  from distutils.core import setup
2  from Cython.Build import cythonize
3
4  setup(ext_modules = cythonize('fibonacci_cython.pyx'))
5
```

Figura 3: Declaración del Setup para la serie de Fibonacci

Y se guarda el archivo como setup.py

Por último, hay que ejecutarlo usando el comando: *python compile.py build-ext -inplace* o si lo desea puede usar un *Mikefile* para mayor comodidad. Si todo va bien se generará un archivo con extensión c con el código traducido a C y otro archivo con el compilado y solo quedará ejecutarlo para así comparar los tiempos de ejecución que dura en Python y Cython y poder ver en tiempo real la diferencia en el rendimiento de cada uno. [1].

### Ecuación de calor bidimensional

La ecuación de calor o difusión es una ecuación diferencial parcial que describe cómo la temperatura varía en el espacio con el tiempo. La solución numérica de la ecuación de calor contiene aspectos de desempeño que están presente también en muchos otros problemas.

La ecuación de calor se puede escribir como:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \quad (1)$$

donde  $\mathbf{u}(\mathbf{x}, \mathbf{y}, \mathbf{t})$  es el campo de temperatura que varía en el espacio y el tiempo, y  $\alpha$  es la constante de difusividad térmica. La ecuación se puede resolver numéricamente en dos dimensiones discretizando primero el operador laplace con diferencias finitas como:

$$\begin{aligned} \nabla^2 u = & \frac{u(i-1, j) - 2u(i, j) + u(i+1, j)}{(\Delta x)^2} \\ & + \frac{u(i, j-1) - 2u(i, j) + u(i, j+1)}{(\Delta y)^2} \end{aligned} \quad (2)$$

Dada una condición inicial como lo enseña la ecuación 2:

$$(u(t=0) = u_0) \quad (3)$$

Entonces se puede seguir la dependencia del tiempo del campo de temperatura con el explícito método de evolución en el tiempo:

$$u_{m+1}(i, j) = u_m(i, j) + \Delta t \alpha \nabla^2 u_m(i, j) \quad (4)$$

Además, el algoritmo es estable en un caso:

$$\Delta t < \frac{1}{2\alpha} \frac{(\Delta x \Delta y)^2}{(\Delta x)^2 + (\Delta y)^2} \quad (5)$$

### Radiación térmica

También llamada radiación calorífica, es la radiación emitida por un cuerpo debido a su temperatura. Esta radiación es radiación electromagnética que se genera por el movimiento térmico de las partículas cargadas que hay en la materia. Todos los cuerpos (salvo uno cuya temperatura fuera cero absoluto) emiten debido a este efecto radiación electromagnética, siendo su intensidad dependiente de la temperatura y de la longitud de onda considerada. La radiación térmica es uno de los mecanismos fundamentales de la transferencia térmica. [2]



Figura 4: Radiación térmica de un animal

En la figura 4 se observa a la derecha la escala de temperaturas y a la izquierda la asignación arbitraria de colores a las temperaturas en el cuerpo.

### Transferencia de calor

La transferencia de calor es el proceso físico de propagación del calor en distintos medios. La transferencia de calor se produce siempre que existe un gradiente térmico en un sistema o cuando dos sistemas con diferentes temperaturas se ponen en contacto.

El proceso persiste hasta alcanzar el equilibrio térmico, es decir, hasta que se igualan las temperaturas. Cuando existe una diferencia de temperatura entre dos objetos cercanos o regiones lo suficientemente próximas se transfiere calor más rápido. [3]



Figura 5: Ilustración de la transferencia de calor

### Computación Paralela

La computación paralela es una forma de cómputo en la que se hace uso de 2 o más procesadores para resolver una tarea. La técnica se basa en el principio según el cual, algunas tareas se pueden dividir en partes más pequeñas que pueden ser resueltas simultáneamente como en la siguiente figura:

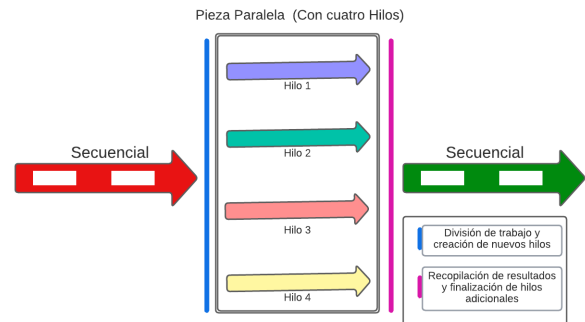


Figura 6: Ilustración de un programa Paralelo

Como se puede observar en la Figura 6 se divide la tarea, en este caso un proceso en varios subprocesos para de esta forma resolverlo de forma más rápida. En algunos casos la computación paralela también da acceso a más memoria, lo que da la posibilidad de resolver problemas mayores.

Pero esto puede presentar algunos desafíos. Además de resolver el problema en cuestión, se agregan las tareas adicionales de dividir y recolectar el trabajo. Es más, puede haber un costo adicional en la creación de nuevos hilos. Los diferentes procesadores también necesitan comunicarse, para que estén sincronizados y los problemas se resuelvan correctamente. Además de esto, la ejecución de un programa paralelo puede ser no determinista, por lo que hace difíciles de depurar [4]

En la computación paralela existe una memoria compartida (SMPP) la cual es visible para todos los procesadores y es la encargada de la comunicación entre los subprocesos, sin embargo si los datos se comparten de forma no intencionada pueden producirse cuellos de botella, por ejemplo, dos o más subprocesos escriben en la misma ranura de memoria simultáneamente, acarreado en sobre escritura entre sí.

Pero la programación paralela también se puede ejecutar sobre una memoria distribuida (DMPP), donde cada hilo se le asigna su propio almacenamiento privado, lo que significa que cada hilo trabaja sin tener en cuenta otros hilos y de esta forma se corrigen cuestiones como los conflictos de escritura y los de intercambio. Sin embargo, al momento de comunicarse entre subprocesos se debe enviar y recibir fragmentos de datos entre todos, suponiendo un coste adicional.

#### IV. METODOLOGÍA

La elaboracion de este informe tiene como fin el poder observar y determinart como el uso de Cython mejora el tiempo de ejecucion de un benchmark inicialmente programado en lenguaje Python.

Lo anterior, haciendo uso de la ecuacion de calor y la libreria NumPy utilizando el campo de temperatura inicial teniendo en cuenta el archivo .dat (el archivo consta de un encabezado y una matriz de datos de  $N \times N$ , siendo  $N$  el tamaño de la botella, el cual se establece dentro del archivo.dat). Como condiciones de contorno, se utilizan valores fijos como se indica en los parámetros de la función main(). El programa principal que se proporciona es en heat-main.py, y se implementa la funcionalidad requerida en el módulo heat.py. Es decir, que los programas junto a sus parámetros asignados simulan la transferencia de calor de un ambiente con temperatura  $A^{\circ}\text{C}$  hacia una botella que se encuentra a una temperatura  $B^{\circ}\text{C}$  siendo menor a la del ambiente,  $A \geq B$ . Para realizar la comparativa del rendimiento de los nodos de cómputo, se desarrollaron tres benchmarks que realizan la multiplicación matricial, haciendo uso de los Cores disponibles en cada nodo de cómputo, para así tener un mejor rendimiento y optimización del cálculo matemático.

Generando dos imágenes como resultado en el que se observa la radiación térmica al empezar y al terminar la simulación, estas imágenes las puede observar en la sección Imágenes de la radiación térmica. Teniendo ya el programa en python se harán las comparaciones en cada una de las máquinas para determinar el tiempo que dura entre ellas (por cada una de las botellas), después se establecerá el cprofile con las estadísticas, y posteriormente se hará el paso a Cython para hacer la comparación final con Python con cada una de las botellas, en cada una de las máquinas usadas.

A continuación se presenta la configuración de cada nodo de cómputo utilizado:

##### IV-A. Configuraciones de los Nodos

Tabla I: Configuraciones de nodos de computo

Nodo	Nucleos	Ram	Procesador	Velocidad
A	8	4 GB	Intel(R) Core(TM) i5-8250	1.6 GHz
B	8	8 GB	AMD Ryzen 7 3700U	2.3 GHz
C	8	16 GB	Intel(R) Core(TM) i7-1185G7	3 GHz
D	8	8 GB	Intel(R) Core(TM) i7-8550U	1.8 GHz

Vale la pena mencionar, que, para la parte experimental, los nodos presentados en la tabla I trabajan con el sistema operativo Linux en algunas de sus diferentes distribuciones.

Los códigos empleados en esta comparativa, están disponibles en el siguiente repositorio: [click aqui](#)

#### V. PROCEDIMIENTO

La primera parte de la práctica, es utilizar el código de Python entregado por el docente y hacer la toma de los tiempos correspondientes a cada una de las botellas por cada una de las máquinas.

Al momento de ejecutar el programa en Python, el código entrega 2 imágenes por cada botella, en dos momentos diferentes del experimento, la primera imagen es la botella en el estado inicial, donde la botella está fría en un entorno caliente y la segunda imagen, es la evolución de la temperatura de la botella durante unos cientos de pasos de tiempo. Las imágenes que entrega por cada botella son las siguientes:



(a) Temperatura en el estado inicial (b) Evolución temperatura cuando pasa el tiempo

Figura 7: Imágenes “bottle.dat”



(a) Temperatura en el estado inicial (b) Evolución temperatura cuando pasa el tiempo

Figura 8: Imágenes “bottle\_medium.dat”



(a) Temperatura en el estado inicial (b) Evolución temperatura cuando pasa el tiempo

Figura 9: Imágenes “bottle\_large.dat”

Adicionalmente, en la ejecución podemos observar desde la terminal el titulo de la practica, la constante de difusión, el nombre del archivo que se está ejecutando, los parámetros que entrega el archivo .dat y el tiempo de ejecución como se observa a continuación:

```
Heat equation solver
Diffusion constant: 0.5
Input file: bottle.dat
Parameters
-----
nx=200 ny=200 dx=0.01 dy=0.01
time steps=200 image interval=4000
Simulation finished in 70.96398568153381 s
```

Figura 10: Terminal ejecución bottle.dat

Luego de observar los datos e imágenes que nos entrega el programa, se procede a la toma de los tiempos promedios, vale la pena mencionar que el software se ejecutó 31 veces, lo anterior con el fin de promediar los tiempos de ejecución, estos datos vienen en unidades de segundos.

Tabla II: Rendimiento de los nodos

Nodo	bottle.dat	bottle_medium.dat	bottle_large.dat
A	47,07s	339,28s	1286,92s
B	33,42s	286,38s	1123,76s
C	28,45s	220,16s	1086,49s
D	39,01s	297,69s	1246,9s

Observando los resultados obtenidos se pasa a realizar el cprofile para obtener las estadísticas del programa y saber en qué partes del código la ejecución se demora más. Al momento de crear los CProfile (perfil01 para bottle.dat, perfil02 para bottle\_medium.dat y perfil03 para bottle\_large.dat) ordenar los tiempos, nos entrega las siguientes imágenes por cada tamaño de botella:

#### CProfile "bottle.dat"

```
Welcome to the profile statistics browser.
perfil01.dat% strip
perfil01.dat% sort time
perfil01.dat% stats 10
Tue May 25 12:07:17 2021 perfil01.dat

771196 function calls (760020 primitive calls) in 75.285 seconds

Ordered by: internal time
List reduced from 3885 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
200    70.450    0.352    70.450    0.352 heat.py:9(evolve)
2742    0.264    0.000    0.511    0.000 inspect.py:625(cleandoc)
1075    0.166    0.000    0.283    0.000 inspect.py:2889(_bind)
40000   0.163    0.000    0.237    0.000 npyio.py:777(floatconv)
74733/73783 0.142    0.000    0.146    0.000 {built-in method builtins.len}
1      0.124    0.124    0.124    0.124 {built-in method posixsubprocess.fork_exec}
321    0.105    0.000    0.105    0.000 {built-in method marshal.loads}
45112   0.101    0.000    0.111    0.000 {built-in method builtins.isinstance}
200    0.084    0.000    0.321    0.000 npyio.py:1074(<listcomp>)
42464   0.079    0.000    0.079    0.000 {method 'lower' of 'str' objects}
```

Figura 11: Estadísticas del programa en bottle.dat

#### CProfile "bottle\_medium.dat"

```
Welcome to the profile statistics browser.
perfil02.dat% strip
perfil02.dat% sort time
perfil02.dat% stats 10
Tue May 25 12:18:50 2021 perfil02.dat

1254178 function calls (1242346 primitive calls) in 444.236 seconds

Ordered by: internal time
List reduced from 3885 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
200    436.558    2.183    436.558    2.183 heat.py:9(evolve)
278784  1.538    0.000    2.183    0.000 npyio.py:777(floatconv)
528    0.757    0.001    2.941    0.006 npyio.py:1074(<listcomp>)
281248  0.651    0.000    0.651    0.000 {method 'lower' of 'str' objects}
2742    0.346    0.000    0.662    0.000 inspect.py:625(cleandoc)
75717/74687 0.171    0.000    0.175    0.000 {built-in method builtins.len}
1075    0.130    0.000    0.241    0.000 inspect.py:2889(_bind)
1      0.115    0.115    0.115    0.115 {built-in method posixsubprocess.fork_exec}
321    0.113    0.000    0.113    0.000 {built-in method marshal.loads}
45112   0.100    0.000    0.115    0.000 {built-in method builtins.isinstance}
```

Figura 12: Estadísticas del programa en bottle\_medium.dat

#### CProfile "bottle\_large.dat"

```
Tue May 25 12:34:33 2021 perfil03.dat

2932286 function calls (2919769 primitive calls) in 1332.158 seconds

Ordered by: internal time
List reduced from 3613 to 10 due to restriction <10>

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
200    1325.636    6.628    1325.636    6.628 heat.py:9(evolve)
1115136  1.117    0.000    1.332    0.000 npyio.py:730(floatconv)
1056    0.653    0.001    1.985    0.002 npyio.py:907(<listcomp>)
1343    0.542    0.000    0.542    0.000 {built-in method nt.stat}
8      0.533    0.067    0.535    0.067 {built-in method matplotlib.image.resample}
273    0.318    0.001    0.318    0.001 {built-in method io.open_code}
3133    0.237    0.000    0.237    0.000 {built-in method numpy.array}
1118611  0.216    0.000    0.216    0.000 {method 'lower' of 'str' objects}
2      0.206    0.103    2.443    1.222 npyio.py:970(read_data)
273    0.135    0.000    0.135    0.000 {built-in method marshal.loads}
```

Figura 13: Estadísticas del programa en bottle\_large.dat

## VI. RESULTADOS

A continuación se presentan los resultados obtenidos. Primero se comparala el tiempo promedio que le tomo a cada nodo de cómputo ejecutar cada benchmark en lenguaje Python, los resultados se muestran a continuación:

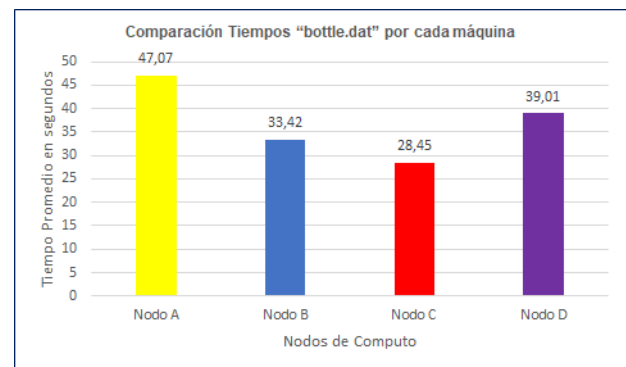


Figura 14: Gráfica de los tiempos promedios de bottle.dat con un tamaño de 200 x 200

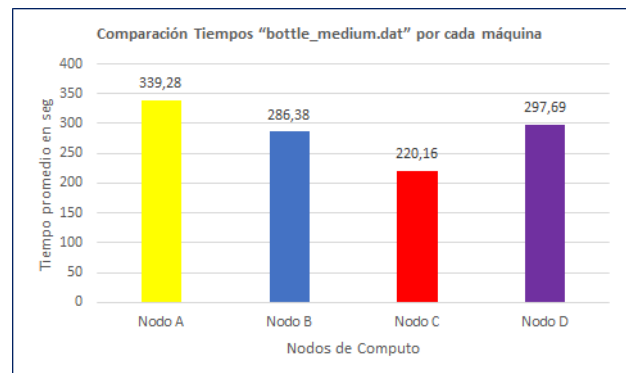


Figura 15: Gráfica de los tiempos promedios de bottle.dat con un tamaño de 528 x 528

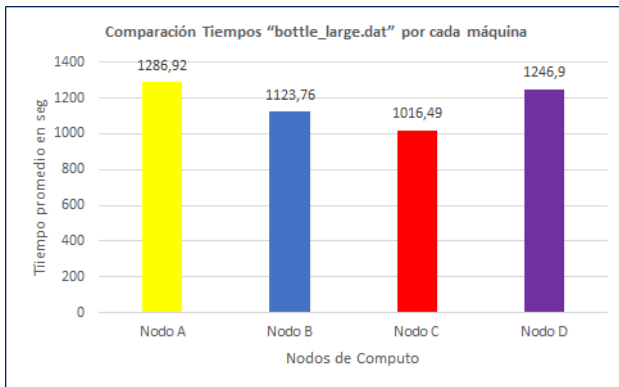


Figura 16: Gráfica de los tiempos promedios de bottle.dat con un tamaño de 1056 x 1056

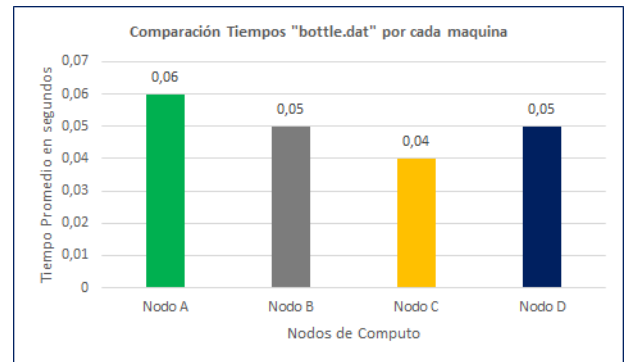


Figura 18: Gráfica de los tiempos promedios de bottle.dat con un tamaño de 200 x 200 en Cython

#### Comparación Tiempos de todos los nodos de computo

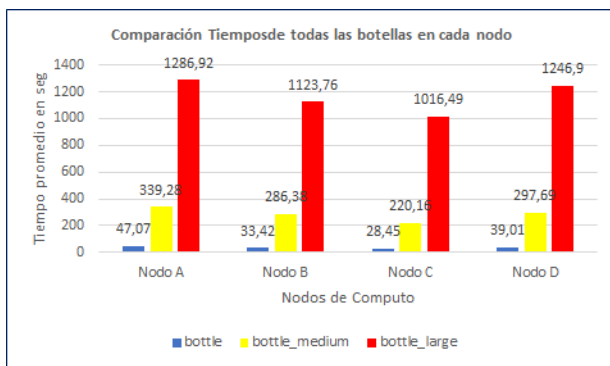


Figura 17: Estadísticas de los tiempos de todas las botellas

Ahora, luego de haber analizado, ejecutado y comparado el programa en el lenguaje de Python, se hará el cambio a Cython utilizando un archivo.pyx y el setup.py para su ejecución, el cual entrega estos nuevos tiempos, vale la pena mencionar que el software se ejecutó 31 veces, lo anterior con el fin de establecer el promedio esas ejecuciones en segundos:

Tabla III: Rendimiento de los nodos

Nodo	bottle.dat	bottle_medium. dat	bottle_large.dat
A	0,05s	0,59s	1,93s
B	0,05s	0,50s	1,79s
C	0,04s	0,38s	1,72s
D	0,06s	0,58s	2,01s

Dada la anterior información se procede a graficar los resultados obtenidos evidenciando lo siguiente:

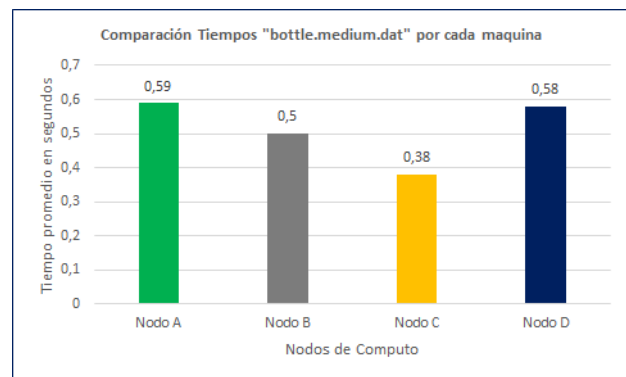


Figura 19: Gráfica de los tiempos promedios de bottle\_medium.dat con un tamaño de 580 x 580 en Cython

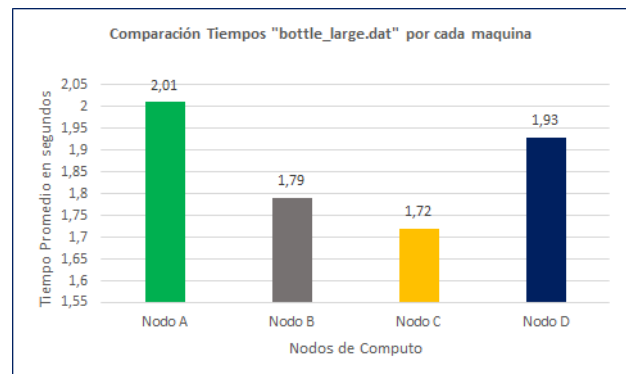


Figura 20: Gráfica de los tiempos promedios de bottle\_large.dat con un tamaño de 1056 x 1056 en Cython

#### Comparación Tiempos de todos los nodos de computo



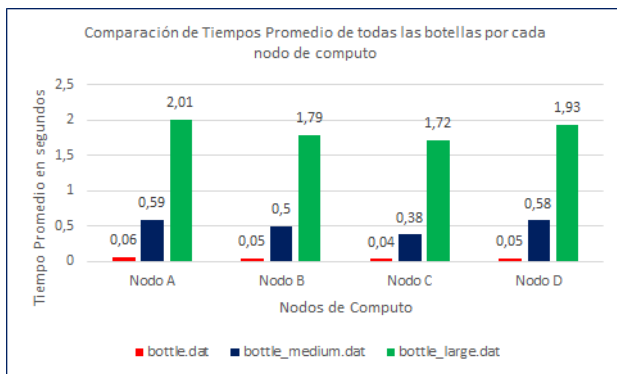


Figura 21: Gráfica de los tiempos promedios para todas las botellas

Una vez obtenidos los resultados con Python y Cython, se procede a comparar de los tiempos finales para cada tamaño de botella y el speedup que tuvieron:

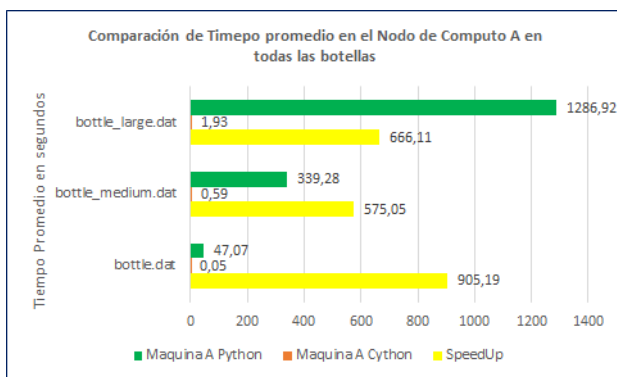


Figura 22: Gráfica de los tiempos promedios de todas las botellas en el Nodo A

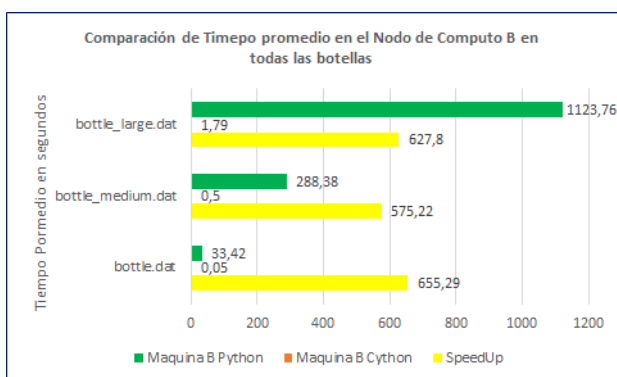


Figura 23: Gráfica de los tiempos promedios de todas las botellas en el Nodo B

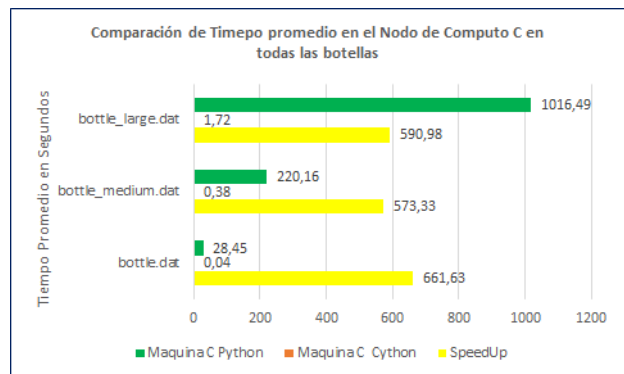


Figura 24: Gráfica de los tiempos promedios de todas las botellas en el Nodo C

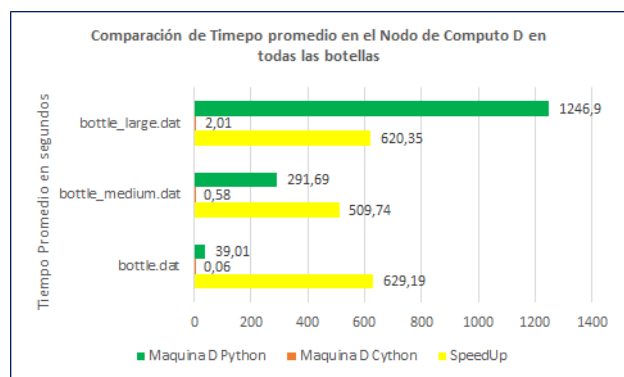


Figura 25: Gráfica de los tiempos promedios de todas las botellas en el Nodo D

Para la segunda parte de este informe se va a evaluar la teoria de ondas largas o mendel:

A conitnuacion se evidencia la los tiempos promedios en la ejecucion de un bencharmk programado en lenguaje de Python y posteriormente en Cython utilizando un archivo.pyx y el setup.py para su ejecución, el cual entrega estos nuevos tiempos. vale la pena mencionar que el software se ejecutó 31 veces, lo anterior con el fin de establecer el promedio esas ejecuciones en segundos:

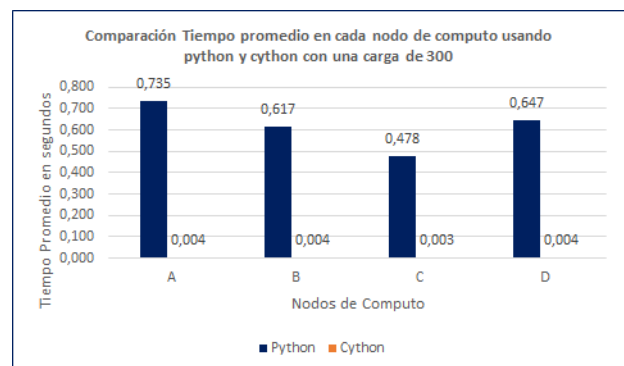


Figura 26: Gráfica del tiempo promedio para una carga de 300 en Python y Cython en los nodos de computo

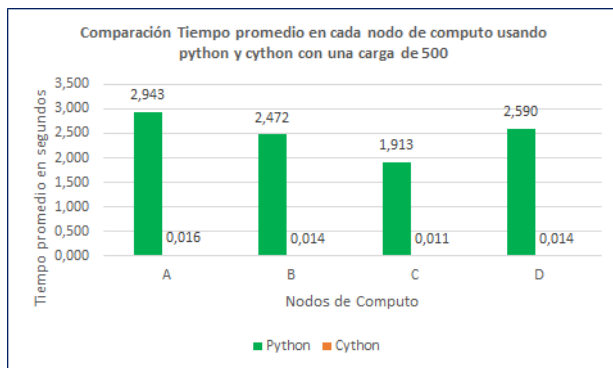


Figura 27: Gráfica del tiempo promedio para una carga de 500 en Python y Cython en los nodos de computo

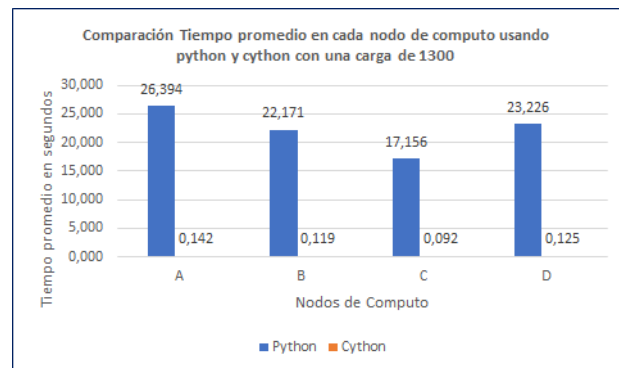


Figura 30: Gráfica del tiempo promedio para una carga de 1300 en Python y Cython en los nodos de computo

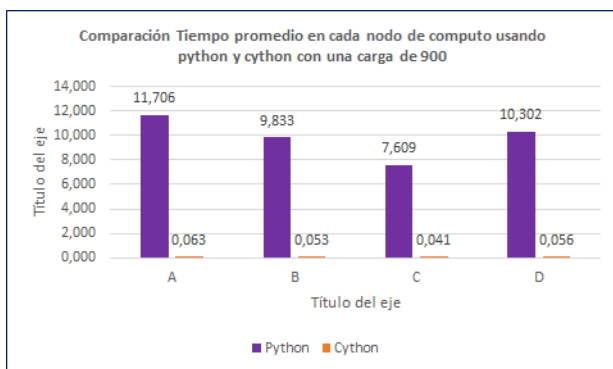


Figura 28: Gráfica del tiempo promedio para una carga de 900 en Python y Cython en los nodos de computo

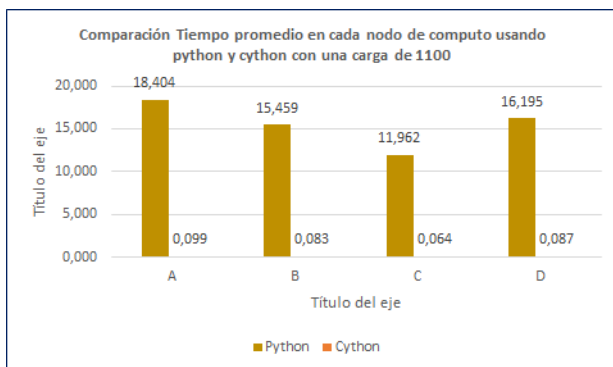


Figura 29: Gráfica del tiempo promedio para una carga de 1100 en Python y Cython en los nodos de computo

Como se puede evidenciar, existe un gran margen de diferencia en el rendimiento dada la compilación y ejecución de los benchmark programados en Python y en Cython, por lo anterior, se procede a mostrar dos graficas con los tiempos promedios de ejecución para una carga de 1300 en Python y en Cython, dado que esta carga es la más grande y nos permite evidenciar el nodo de cómputo con mejor rendimiento para ambos lenguajes de programación:

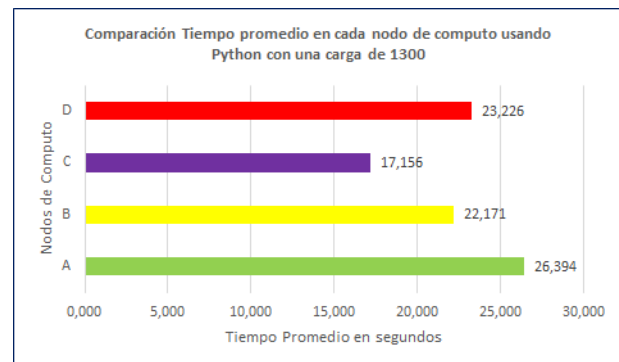


Figura 31: Gráfica del tiempo promedio para una carga de 1300 en Python en los nodos de computo

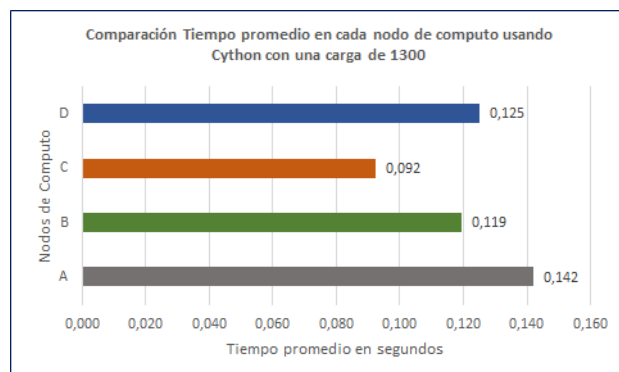


Figura 32: Gráfica del tiempo promedio para una carga de 1300 en Cython en los nodos de computo



Dado lo anterior y según especificaciones de la tabla II el nodo de cómputo C presenta el mejor rendimiento, lo anterior, dada su arquitectura de cómputo que presenta una superioridad ajustada tomando como referencia los demás nodos de cómputo utilizados.

## VII. DISCUSIÓN

En los resultados de la primera comparativa del rendimiento de los nodos, *el nodo de cómputo C* obtuvo el mejor resultado, ya que sus tiempos promedio son menores en todas las cargas y benchmarks en comparación de los otros tres. A modo de ranking y observando los tiempos de desempeño, los nodos quedan organizados de la siguiente forma:

- Nodo C.
- Nodo B.
- Nodo D.
- Nodo A.

La condensación de todos los tiempos de ejecución para el lenguaje Python se encuentran en la figura 31 y para Cython en la figura 32 en la que con certeza podemos evidenciar la eficiencia y mejora de rendimiento al usar Cython, dado que sin importar la arquitectura de cualquier nodo de cómputo, este lenguaje puede garantizar hasta un 46 % de eficiencia en la ejecución de cualquier benchmark que inicialmente se haya programado en Python y que mantenga la misma lógica y estructura.

## VIII. CONCLUSIONES

- Como se pudo evidenciar, existe una gran diferencia entre los tiempos de los dos lenguajes, en los cuales se puede evidenciar en la tabla 3 una reducción de casi el 99 por ciento de los resultados de la tabla 2, confirmando la gran optimización que Cython le realizó al código..
- Los lenguajes de programación utilizados, juegan un papel fundamental a la hora de poner a prueba el rendimiento de un nodo de cómputo específico, pero, evidentemente también comprender implícitamente la importancia y la función de la ecuación del calor y como afecta superficies que vayan cambiando de temperatura
- Para finalizar, la disminución en la sobrecarga de interpretación, aunque es un tema simple, es esencial en la creación de programas y aplicaciones los cuales pueden ayudar en la resolución de los problemas del mundo real.

## REFERENCIAS

- [1] Walt, K. Jarrod Millman 2019 "Cython Tutorial"Departamento de Matematicas - Universidad de Washington. [PDF](#)
- [2] Frank P. Incropera 2020 "Fundamentos de transferencia de calor"PEARSON Prentice Hall. [PDF](#)

- [3] Yunus A. Cengel 2021 "Transferencia de calor y masa". Universidad Autonoma de Tamaulipas. [PDF](#)
- [4] Hoeger H. 2020 "Introducción a la Computación Paralela". Centro Nacional de Cálculo Científico Universidad de Los Andes [PDF](#)

## IX. BIOGRAFÍA



Oscar Julian Reyes Torres nació el 29 de junio de 1997 en Duitama, Boyacá, Colombia. Es estudiante de noveno semestre de Ingeniería de Sistemas y Telecomunicaciones en la universidad Sergio Arboleda. Ha trabajado en la Rama Judicial brindando soporte y capacitación en los nuevos softwares de reparto en la Oficina Judicial de Tunja. Porcentaje de Contribución:100 %