Tercer Parcial: Comparativa Python y Cython en la Ecuación del Calor

Luis Alejandro Vejarano Gutierrez
Escuela de Ciencias Exactas e Ingeniería
Universidad Sergio Arboleda -Bogotá, Colombia
luis.vejarano01@correo.usa.edu.co

Jessica Valentina Parrado Alfonso
Escuela de Ciencias Exactas e Ingeniería
Universidad Sergio Arboleda -Bogotá, Colombia
jessica.parrado01@correo.usa.edu.co

Lady Geraldine Salazar Bayona
Escuela de Ciencias Exactas e Ingeniería
Universidad Sergio Arboleda -Bogotá, Colombia
lady.salazar01@correo.usa.edu.co

Michael Steven Pinilla Mateus
Escuela de Ciencias Exactas e Ingeniería
Universidad Sergio Arboleda -Bogotá, Colombia
michael.pinilla01@correo.usa.edu.co

Computación Paralela
Mayo 2021

Resumen.

Se busca comparar tiempos de ejecución de dos lenguajes de programación, Cython y Python, de un mismo programa (ecuación de calor bidimensional) y así analizar su rendimiento. Para esto, se propone disminuir la sobrecarga de interpretación de python a través de la "conversión" a Cython; se plantea solucionarlo con la comparación de procesos y el tiempo de ejecución de cada uno de ellos para llegar a la mejor opción usando perl, makefile y el archivo autogenerable html. Por lo que se espera obtener mejores resultados de tiempo de ejecución en el programa basado en lenguaje Cython, ya que este es un lenguaje compilado apoya considerablemente a mejorar el rendimiento. Este proceso nos llevó a concluir que el programa realizado en Cython es el más rápido y óptimo comparado con Python, demostrando una mejora en el rendimiento.

Palabras Claves: Cython, Python, Makefile, Comparativa y Ecuación de calor.

I. Introducción.

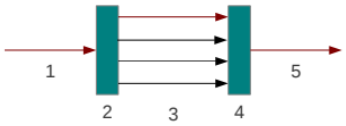
En el desarrollo de este curso estamos profundizando en cómo optimizar procesos y memoria de los equipos, utilizando al máximo todos los recursos, para aumentar el rendimiento de diferentes programas, por eso es fundamental saber cómo funcionan y desarrollan los módulos y dónde se deben manejar ciertos lenguajes de programación para sacar el mejor provecho.

El disminuir 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.

II. Fundamentos

Computación paralela

La computación paralela es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente. El uso de varios procesadores permite dividir los problemas en varios subprocesos y así resolverlos más rápido. En algunos casos, la computación paralela también da acceso a más memoria, lo que da la posibilidad de resolver problemas mayores. En la figura 1 se muestra una ilustración de un programa paralelo, con 4 procesadores (Ortiz, 2020).



Un programa paralelo puede trabajar de la siguiente manera:

1. Parte secuencial.
2. Divida el trabajo y cree nuevos hilos.
3. Pieza paralela (con cuatro hilos).
4. Recopile resultados y finalice los hilos adicionales.
5. Parte secuencial.

Figura 1. Ilustración de un programa paralelo

Sin embargo, la computación paralela también presenta 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 los hace difíciles de depurar (Ortiz, 2020).

En la computación paralela la memoria compartida (SMPP) es visible para todos los procesadores, donde la comunicación entre los subprocesos se realiza a través de esta memoria. Sin embargo, si los datos se comparten de forma no intencionada, pueden producirse cuellos de botella. Un ejemplo de esto son los conflictos de escritura, donde los subprocesos escriben en la misma ranura de memoria al mismo tiempo, posiblemente sobre escribiéndose entre sí (Ortiz, 2020).

La programación en paralelo también se puede realizar con memoria distribuida (DMPP). En este caso, cada hilo tiene su propio almacenamiento privado. Por lo tanto, cada hilo puede funcionar sin tener en cuenta los otros hilos y tratar cuestiones como los conflictos de escritura y los de intercambio. Sin embargo, para comunicarse, los subprocesos deben enviar y recibir fragmentos de datos unos con otros, lo que supone un coste adicional (Ortiz, 2020).

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.

En conclusión, Python es uno de los mejores lenguajes que se pueda utilizar al momento de programar y con el auge que tiene hasta el

momento, es una gran alternativa para crear grandes proyectos. (Robledano, 2020) (Pardo, 2003)

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. Por ejemplo declarar una variable como indica la figura 2.

```
1.  i = 1
```

A la hora de usar Cython se tiene que escribir

```
1.  cpdef int i = 1
```

Fig 2. Ejemplo traducción de Python a 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_cython.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 figura 3 usando la serie de Fibonacci

```
from distutils.core import setup
from Cython.Build import cythonize

setup(ext_modules = cythonize('fibonacci_cython.pyx'))
```

Fig 3. 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. (Johns, 2016) (Correoso, 2015) (Rodríguez, 2019)

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 lo muestra la figura 4:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u$$

Fig 4 . Ecuación de Calor

donde u(x, y, t) es el campo de temperatura que varía en el espacio y el tiempo, y α es la constante de difusividad térmica. La ecuación se puede resolver numéricamente en dos dimensiones discretizando primero el operador laplace con diferencias finitas como lo demuestra la figura 5:

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

Fig 5. Ecuación con dos dimensiones

Dada una condición inicial como lo enseña la figura 6:

$$(u(t=0) = u_0)$$

Fig 6. Condición Inicial

Entonces se puede seguir la dependencia del tiempo del campo de temperatura con el explícito método de evolución en el tiempo como lo muestra la figura 7:

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

Fig 7. Nueva ecuación con dependencia del tiempo

Además, el algoritmo es estable en un caso y es el de la figura 8:

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

Fig 8. Caso de estabilidad (Widder, 1976)

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. (Montes, 2014)

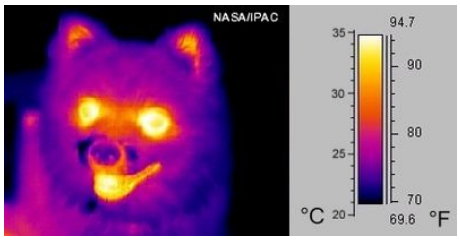


Ilustración 1. Radiación térmica de un perro

En la ilustración 1 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. (Cengel, 2004)

Ilustración 2. Dibujo ilustrativo de la transferencia de calor

III. Metodología del experimento

La intención de esta práctica es poder analizar cómo el uso de Cython llega a mejorar el tiempo de ejecución en un programa hecho principalmente en Python.

Para esto se implementa la ecuación de calor bidimensional con 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 x 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°C hacía una botella que se encuentra a una temperatura B°C siendo menor a la del ambiente, A > B.

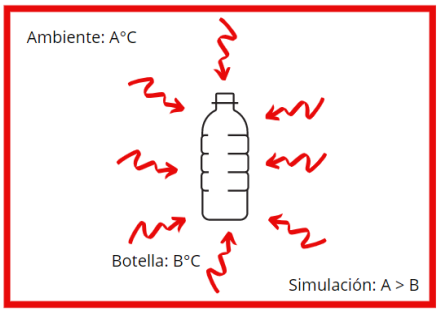


Ilustración 3. Ilustración de la simulación realizada en los programas.

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.

Las especificaciones de cada una de las máquinas utilizadas en la práctica son las siguientes:

	Máquina A	**Máquina B**	**Máquina C**	**Máquina D**
Número de núcleos	8	8	4	4
RAM (GB)	16	4	8	4
Sistema Operativo	Windows 10 de 64 bits	Windows 10 de 64 bits	Windows 10 de 64 bits	Windows 10 de 64 bits
Procesador	Intel(R) Core(TM) i5-8300H	Intel(R) Core(TM) i5-8250U	Intel(R) Core(TM) i5-6200U	AMD A9-9425
Velocidad del procesador	2.30 GHz	1.60 GHz	2.30 GHz	2.00 GHz

Tabla 1. Configuración de las máquinas

El link del repositorio para poder ver los programas es el siguiente: https://github.com/JessicaParrado/Parcial3HeatEquation

IV. Resultados

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:

Imágenes de la radiación térmica

Todos los cuerpos, en este caso botellas, emiten cierta cantidad de radiación térmica en función de su temperatura. Además, generalmente los objetos con mayor temperatura emiten más radiación que los que poseen menor temperatura y esto se evidencia al ejecutar el programa.

En las figuras 9, 10, 11, 12, 13 y 14 muestran las áreas más calientes de un cuerpo en blanco y las menos en café, y con matices grises los grados de temperatura intermedios entre los límites térmicos, todos los cuerpos observándose en un fondo verde.

- **Imágenes “bottle.dat”**

Fig 9. Temperatura en el estado inicial

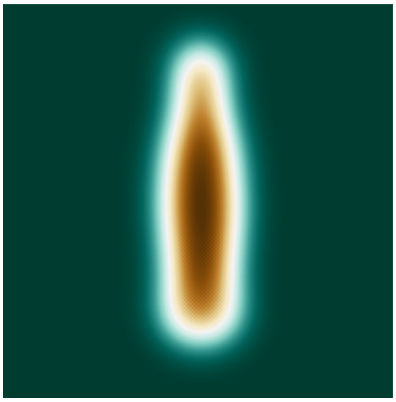


Fig 10. Evolución temperatura cuando pasa el tiempo

- **Imágenes “bottle_medium.dat”**

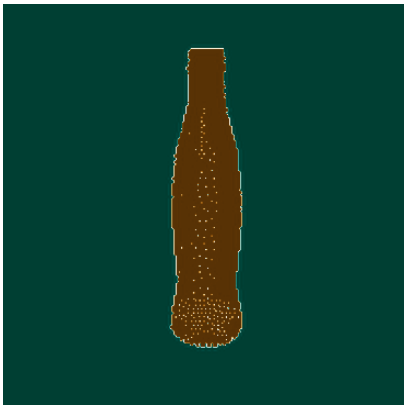


Fig 11. Temperatura en el estado inicial

Fig 12 Evolución temperatura cuando pasa el tiempo

- **Imágenes “bottle_large.dat”**

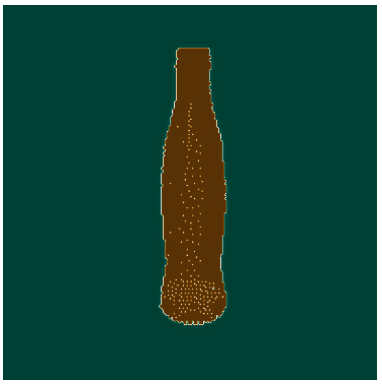


Fig 13. Temperatura en el estado inicial

Fig 14. Evolución temperatura cuando pasa el tiempo

Además de las imágenes, la ejecución también entrega datos en la terminal, como el título de la práctica, 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 lo muestra la figura 15:

```
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
```

Fig 15.Terminal ejecución bottle.dat

Luego de comprender que datos e imágenes nos entrega el programa, se procede a la toma de los tiempos promedios (el programa se ejecutó 30 veces para luego obtener el promedio de esas ejecuciones) (el tiempo está en segundos)(Recuerde que los programas utilizados están en el link del repositorio que se entregó en la sección de “Metodología del experimento”):

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

Tabla 2. Tiempos promedios en Python

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) y ordenar los tiempos, nos entrega las siguientes imágenes por cada tamaño de botella:

CProfile "bottle.dat"

```
jessica@DESKTOP-DQE9AS4:~/Parcial3$ python3 -m pstats perfil01.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)
74733/73703    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    0.002 npyio.py:1074(<listcomp>)
    42464    0.079    0.000    0.079    0.000 {method 'lower' of 'str' objects}
```

Fig 16. Estadísticas del programa en bottle.dat

CProfile "bottle_medium.dat"

```
jessica@DESKTOP-DQE9AS4:~/Parcial3$ python3 -m pstats perfil02.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    0.106    0.000    0.115    0.000 {built-in method builtins.isinstance}
```

Fig 17. 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:997(<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}
```

Fig 18. Estadísticas del programa en bottle_large.dat

Cada uno de los cprofile nos entrega que el módulo heat.py en la función de "evolve" es donde el programa dura más. Al analizar detenidamente esa parte del código, se puede determinar que el causante de la demora es la iteración entre dos "for", los cuales son esenciales para hacer la evolución de la temperatura de la botella, mientra va pasando el tiempo y así generar el resultado final con el cambio. Sin embargo, el tratar de cambiar esa parte de código por otra para que el rendimiento sea mejor, usando otros condicionales y otras variables, no funciona, realmente lo empeora, por lo que decidimos mejor no cambiar el código para que quede lo más optimizado posible.

Teniendo en cuenta que el código no se cambió, se hará uso de los resultados de la tabla 2 para el código en Python. Además, para poder comprender de mejor manera los tiempos obtenidos, se presentan las figuras 19, 20, 21 y 22.

Comparación Tiempos "bottle.dat" por cada máquina

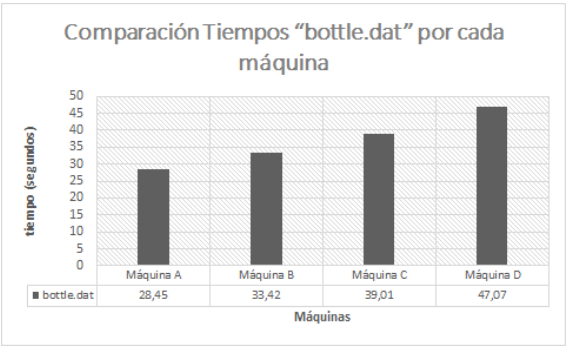


Fig 19. Gráfica de los tiempos de bottle.dat

La figura 19 muestra tiempos entre los 28 y 48 segundos, siendo la máquina A la más rápida de todas y la máquina D la más deficiente. Sus tiempos tan pequeños (no sobrepasan el minuto) se debe al tamaño de 200x200 que la botella tiene, por lo que los cálculos a realizar para ver la evolución de la temperatura en este pequeño objeto, no son tantas comparado con otros tamaños.

Comparación Tiempos "bottle_medium.dat" por cada máquina

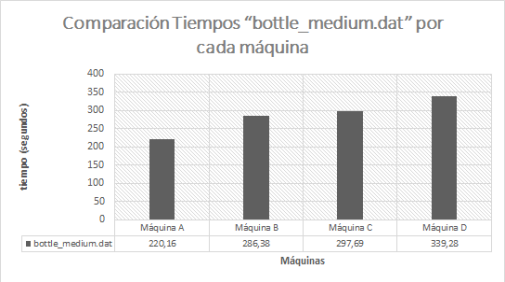


Fig 20. Gráfica de los tiempos de bottle_medium.dat

La figura 20 muestra tiempos entre los 220 y 340 segundos en promedio, siendo la máquina A la más eficiente y la máquina D la más deficiente. Al ser un tamaño de 528x528 es un poco más del doble que bottle.dat por lo que el incremento de sus tiempos era algo de esperarse, aunque no con tiempos tan grandes, esto debe ser por las especificaciones de cada máquina y los procesos en segundo plano que llegan a manchar los datos de la práctica.

Comparación Tiempos "bottle_large.dat" por cada máquina

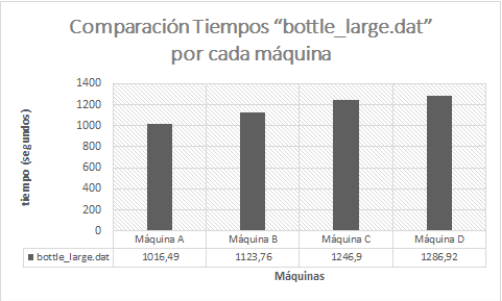


Fig 21. Gráfica de los tiempos de bottle_large.dat

La figura 21 muestra tiempos entre los 1016 y 1287 segundos en promedio, siendo la máquina A la más rápida y la máquina D la más deficiente. Para esta botella se tiene un tamaño de 1056x1056, lo cual es aproximadamente 5 veces más grande que bottle.dat y entrega tiempos muy altos debido a los cálculos internos del programa.

Comparación Tiempos de todas las botellas por cada máquina

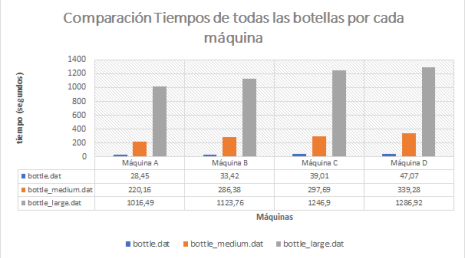


Fig 22. Gráfica de los tiempos de todas las botellas

Por último se encuentra la figura 22 el cual es una comparación con todos los tamaños de las botellas en todas las máquinas. Aquí se evidencia la eficiencia que tuvo la máquina A en todas la ejecuciones y los grandes cambios de tiempos que hay por cada una de las botellas, entregando tiempos muy variados.

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 (el programa se ejecutó 30 veces para luego obtener el promedio de esas ejecuciones) (el tiempo está en segundos) (Recuerde que los programas utilizados están en el link del repositorio que se entregó en la sección de "Metodología del experimento"):

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

Tabla 3. Tiempos promedios en Cython

Teniendo en cuenta los resultados obtenidos en la tabla 3 se van a presentar las figuras 23, 24, 25 y 26

Comparación Tiempos "bottle.dat" por cada máquina

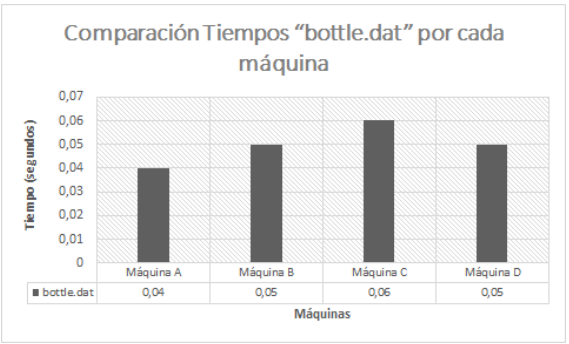


Fig 23. Gráfica de los tiempos de bottle.dat

En la figura 23 se observa el rendimiento de la simulación usando la botella más pequeña, bottle.dat, y mostrando que la máquina con mejor rendimiento es la A, y que la máquina B y la máquina D sus tiempos promedios de ejecución son

muy cercanos (aclarar que se desprecian los demás dígitos decimales).

Comparación Tiempos "bottle_medium.dat" por cada máquina

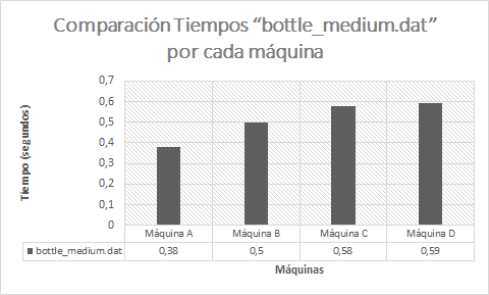


Fig 24. Gráfica de los tiempos de bottle_medium.dat

Ahora, dentro de la figura 24 se comparan los tiempos de simulación con el uso de la botella mediana, bottle_medium.dat. Nuevamente, la máquina A tiene el mejor rendimiento, pero a comparación con la figura 23 se evidencia que a la máquina D le toma más tiempo para procesar este tamaño de botella haciendo que quede de último lugar.

Comparación Tiempos "bottle_large.dat" por cada máquina

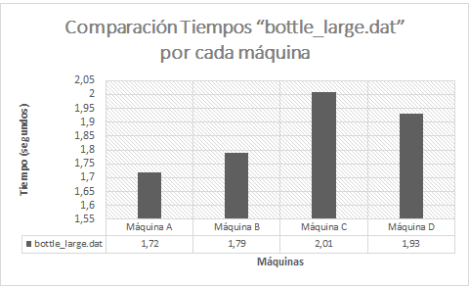


Fig 25. Gráfica de los tiempos de bottle_large.dat

Es turno de realizar la simulación con la botella más grande, bottle_large.dat, donde sus tiempos de ejecución se encuentran en la figura 25 y se muestra que la máquina C y la máquina D son las más ineficientes con un tardío aproximado del 11,6% a comparación de la más rápida (máquina A).

Comparación Tiempos de todas las botellas por cada máquina

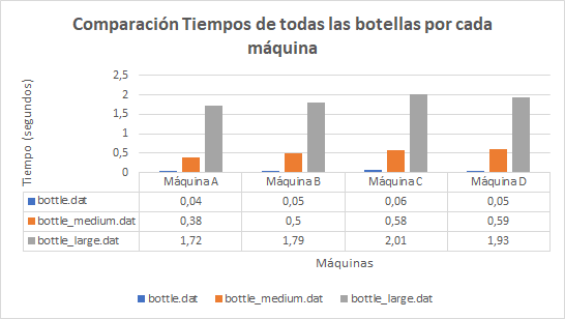


Fig 26. Gráfica de los tiempos de todas las botellas

Ya teniendo los resultados con Python y Cython, se realiza ahora la comparación de los tiempos finales para cada tamaño de botella y el speedup que tuvieron por medio de gráficas.

Comparación Cython vs Python

En las figuras 27, 28, 29 y 30 se observa los tiempos de ejecución de python, cython y el speedup además de la tabla con los datos de cada una, debido a que el speedup es tan grande, es decir que la reducción de tiempo de ejecución que se obtiene con cython es considerable, la barra representativa de cython no es visible en la gráfica, ya que en promedio cython representa la reducción de un 99% del tiempo de ejecución.

- **Comparación Tiempos "Máquina A" en todas las botellas**

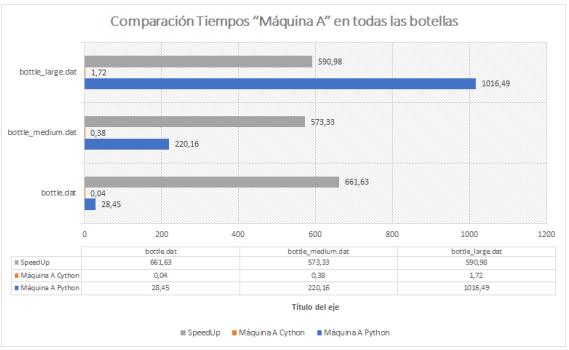


Fig 27. Gráfica de los tiempos de todas las botella en la máquina A

En la figura 27, cython para bottle.dat reduce el tiempo de ejecución un 99,85%, para bottle_medium.dat un 99,82% y para bottle_large.dat una reducción del 99,83%.

- **Comparación Tiempos "Máquina B" en todas las botellas**

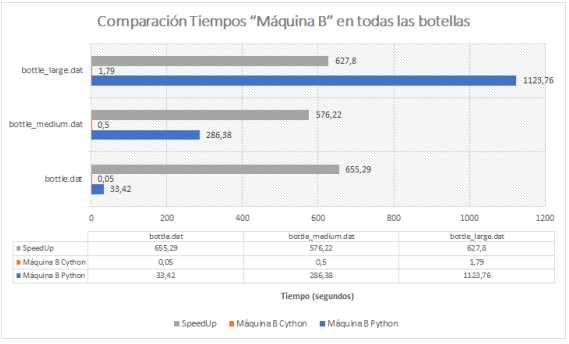


Fig 28. Gráfica de los tiempos de todas las botella en la máquina B

En la figura 28, cython aumenta el rendimiento para bottle.dat un 99,85%, para bottle_medium.dat un 99,82% y para bottle_large.dat aumenta el rendimiento un 99,84%.

- **Comparación Tiempos "Máquina C" en todas las botellas**

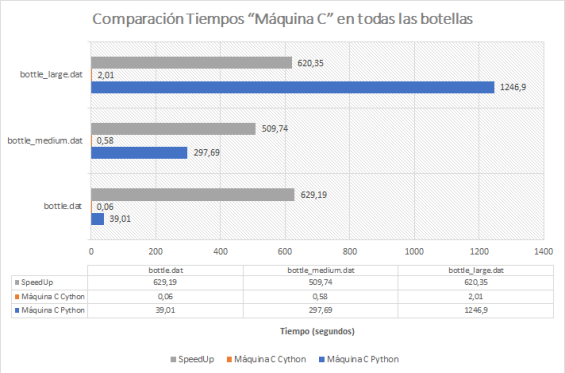


Fig 29. Gráfica de los tiempos de todas las botella en la máquina C

En la figura 29, se observa que python es más lento a comparación de cython para bottle.dat un 99,84%, para bottle_medium.dat un 99,8% y para bottle_large.dat un total de 99,83%.

- **Comparación Tiempos "Máquina D" en todas las botellas**

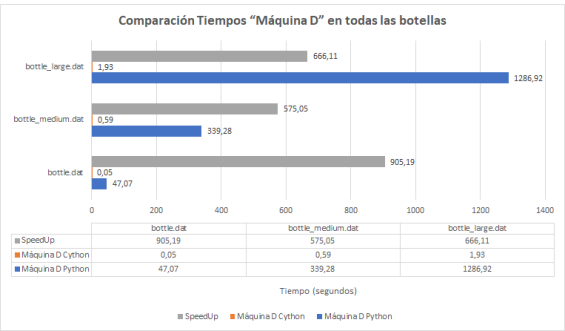


Fig 30. Gráfica de los tiempos de todas las botella en la máquina D

Ahora, en la figura 30 ocurre una reducción en el tiempo de ejecución muy similar a las figuras 27, 28 y 29, ya que cython reduce para bottle.dat un 99,82% el tiempo de ejecución de python, para bottle_medium.dat un 99,84% y para bottle_large.dat un 99,85%.

- **Comparación Tiempos en todas las botellas en todas las máquinas**

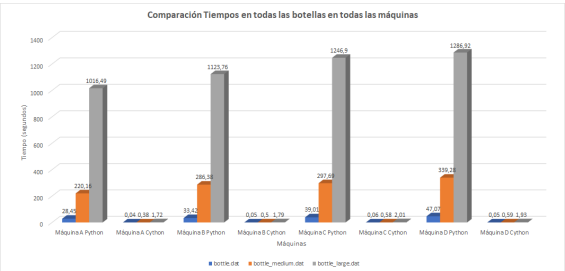


Fig 31. Gráfica de los tiempos de todas las botella en todas las máquinas

La condensación de todos los tiempos de ejecución se encuentran en la figura 31, en la que con certeza muestra el rendimiento por máquina siendo la que tiene mejor rendimiento la máquina A, luego la B, por tercera la máquina C y de últimas la máquina D. Además, de mostrar la reducción drástica de tiempo de ejecución que se logra al disminuir la sobrecarga de interpretación con cython.

V. Conclusiones

Durante el proceso se practica acerca del ajuste de un programa hecho principalmente en Python, para poder pasarlo a Cython y obtener una mejora de rendimiento y tiempos de ejecución. Asimismo, comprender implícitamente la importancia y la función de la ecuación del calor y como afecta superficies que vayan cambiando de temperatura.

Además, al analizar y comparar las gráficas de los tiempos de ejecución, se puede determinar que la máquina A en los tiempos de Python y Cython tuvo el mejor rendimiento, ya que fue el más rápido. Sin embargo, las máquinas C y D fueron las más deficientes, teniendo tiempos muy altos, en Python la más deficiente fue la máquina D y para Cython la más deficiente fue la máquina C. Por otro lado, la máquina B se mantuvo equilibrada, teniendo tiempos medios con respecto a las otras máquinas.

También hay que tener en cuenta que existe la probabilidad de que se presentarán programas en segundo plano que se hayan ejecutado o problemas de hardware que hayan aumentado el tiempo de ejecución "manchando" algunos datos y aumentaran el tiempo de respuesta, y esta puede ser una de las razones para que las máquinas C y D variaran sus tiempos entre Python y Cython.

Además se pudo observar en la gráficas la 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% de los resultados de la tabla 2, confirmando la gran optimización que Cython le realizó al código.

Para finalizar, se tuvo que ser muy minuciosos en la ejecución, teniendo en cuenta los requisitos pedidos, para que así la toma de

rendimientos cumpliera el objetivo satisfactoriamente.

VI. Referencias

- Ortiz, J. (2020, 14 julio). La computación paralela: alta capacidad de procesamiento. Teldat Blog - Conectando el mundo. https://www.teldat.com/blog/es/computacion-paralela-capacidad-procesamiento/
- Widder, D. V. (1976). The heat equation (Vol. 67). Academic Press
- Bluman, G. W., & Cole, J. D. (1969). The general similarity solution of the heat equation. Journal of Mathematics and Mechanics, 18(11), 1025-1042.
- Robledano, A. (2020, 3 julio). Qué es Python: Características, evolución y futuro. OpenWebinars.net. https://openwebinars.net/blog/que-es-python/
- Pardo. (2003, 19 noviembre). Qué es Python. Desarrollo Web. https://desarrolloweb.com/articulos/1325.php
- Johns. (2016). ¿Qué es Cython? Python a la velocidad de C. Small business tracker. https://spa.small-business-tracker.com/what-is-cython-python-speed-c-397206
- Correoso, K. (2015, 9 marzo). C elemental, querido Cython – Pybonacci. Pybonacci. https://pybonacci.org/2015/03/09/c-elemental-querido-cython/
- Rodríguez, D. (2019, 6 septiembre). Aumentar el rendimiento de Python con Cython. Analytics Lane. https://www.analyticslane.com/2019/09/30/aumentar-el-rendimiento-de-python-con-cython/
- Montes Pita, M., Muñoz Domínguz, M., Rovira de Antonio, A. J. (2014). Ingeniería térmica.
- Cengel, Y. A., & Pérez, H. (2004). Heat transfer: a practical approach. transferencia de calor.

VII. BIOGRAFÍAS

Jessica Valentina Parrado Alfonso nació el 31 de Enero del 2001 en Bogotá, Colombia. Actualmente, es estudiante de Ingeniería de Sistemas y Telecomunicaciones en la Universidad Sergio Arboleda. Durante este trabajo realizó el 25% de su totalidad.

Michael Steven Pinilla Mateus nació el 9 de Octubre de 2000 en Bogotá, Colombia. En el presente estudia la carrera profesional de Ingeniería de Sistemas y Telecomunicaciones en la Universidad Sergio Arboleda. Durante este trabajo realizó el 25% de su totalidad.

Lady Geraldine Salazar Bayona nació el 6 de Agosto de 1998 en Bogotá, Colombia. En los últimos años ha estado estudiando Ingeniería de Sistemas y Telecomunicaciones en la Universidad Sergio Arboleda. Durante este trabajo realizó el 25% de su totalidad.

Luis Alejandro Vejarano Gutierrez nació el 22 de Mayo del 2000 en Bogotá, Colombia.. Se desempeña como estudiante de Ingeniería de Sistemas en la Universidad Sergio Arboleda. Durante este trabajo realizó el 25% de su totalidad.