Bogotá DC, 08 de noviembre del 2022
Universidad Sergio Arboleda
Ciencias de la Computación e inteligencia Artificial
Computación Paralela y Distribuida

Comparativa de rendimiento entre Python y optimización en Cython

Jesús Chaves Acero

Resumen:

Se realiza un experimento donde se realiza la comparativa del rendimiento en tiempos de ejecución entre un programa desarrollado en Python y este mismo siendo optimizado en Cython, básicamente, lo que se busca es cuantificar de manera aproximada la mejora en tiempos de ejecución de acuerdo a la optimización que se logra Cython sin ahondar en profundidad en dicho lenguaje. Para esta experimentación se realiza un programa en Python el cual calcula la órbita de un planeta.

Introducción:

Cython es un lenguaje de programación para simplificar la escritura de módulos de extensión para Python en C y C++. Siendo estrictos, la sintaxis de Cython es la misma de Python pero con algunos agregados. Se pueden llamar funciones en C, o funciones/métodos de C++, directamente desde el código en Cython. Es posible usar tipos estáticos en las variables (enteros, flotantes, o cualquier tipo de dato). Cython compila a código en C o C++ desde Python, y el resultado puede ser usado desde Python como un "Módulo de extensión", o como una aplicación embebida en el intérprete CPython.

Metodología:

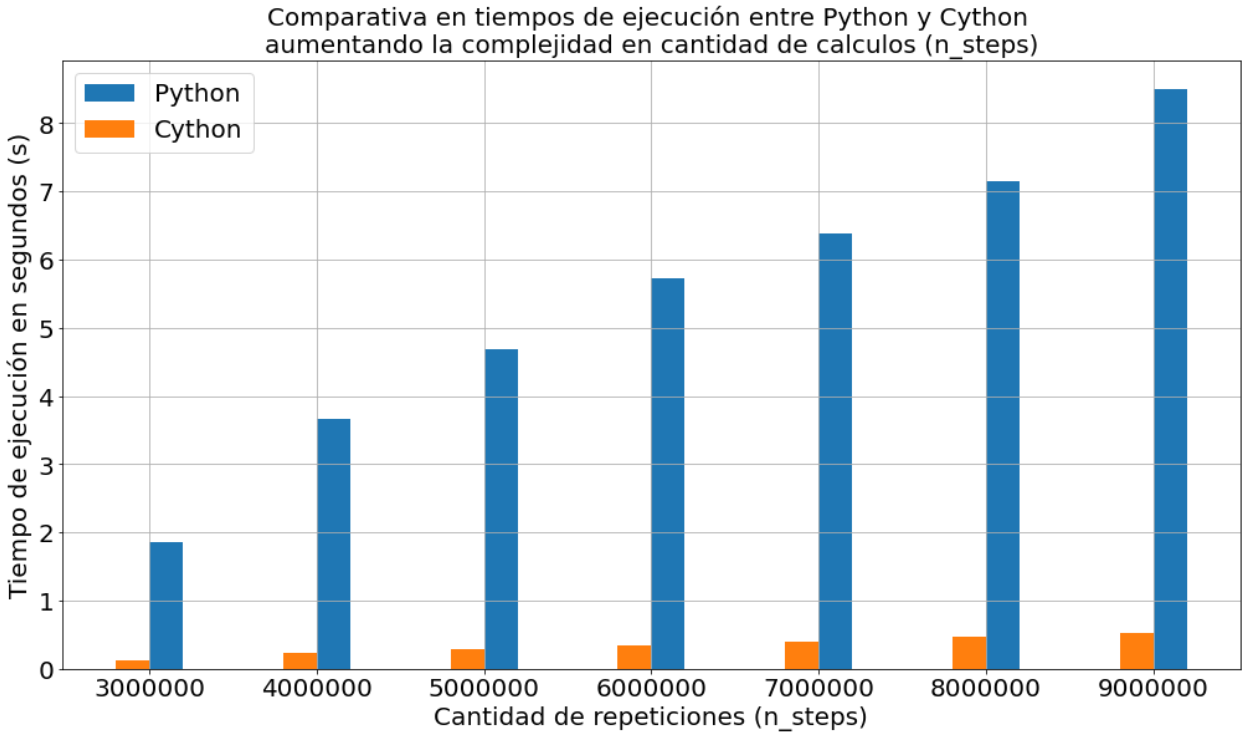
Se realiza un programa inicialmente en Python para el cálculo de la órbita de un planeta dada una posición inicial y masa del planeta, de acuerdo a la posición inicial realiza el cálculo de la distancia y la derivada de la posición de acuerdo a la velocidad. Como variables de entrada recibe los datos del planeta, el espacio de tiempo y el número de pasos o etapas calcular. Al tener el programa en Python se procede a optimizar el código realizado en Cython, para esta experimentación se hará una optimización muy superficial únicamente declarando los tipos de variables a manejar en C, así mismo como las funciones que retornan algún tipo de variable. Para los cálculos de los tiempos de ejecución se realiza un fichero en Python el cual funciona como lanzador del programa para así poder realizar varios experimentos consecutivos con el fin de sacar un tiempo promedio y así reducir el ruido Gaussiano en la experimentación en una secuencia de 30 repeticiones para por último obtener un valor representativo en el tiempo de

ejecución de cada experimento. Se realizan 7 experimentos los cuales partirán desde 3’000.000 de números de pasos los cuales aumentaron en 1’000.000 para cada experimento finalizando en 7’000.000 de pasos para el experimento final. Luego de la experimentación se procede a realizar el análisis entre los tiempos de ejecución respecto a la complejidad de cálculos a realizar representada en el aumento de números de pasos en los 7 experimentos.

Resultados:

Se realiza una gráfica de barras la cual muestra el valor representativo de cada experimento, además, se realiza una tabla con el cálculo de la desviación estándar y promedio de cada experimento así como los valores máximos y mínimos:

		Unidad de tiempo en segundos (s)			
Steps (Millones)	Iteraciones	Promedio	Desviación (std)	Mínimo	Máximo
3M steps Python	30	1.857050	0.032167	1.816285	1.941108
3M steps Cython	30	0.116722	0.004467	0.114726	0.132708
4M steps Python	30	3.665618	0.049180	3.635203	3.900866
4M steps Cython	30	0.231645	0.004423	0.229403	0.249458
5M steps Python	30	4.692408	0.080056	4.632622	4.880662
5M steps Cython	30	0.291328	0.013369	0.286745	0.348008
6M steps Python	30	5.718335	0.074404	5.617453	5.923275
6M steps Cython	30	0.350681	0.012630	0.344085	0.406561
7M steps Python	30	6.388758	0.020146	6.366404	6.478752
7M steps Cython	30	0.404273	0.004520	0.401445	0.420333
8M steps Python	30	7.147220	0.031584	7.081652	7.275547
8M steps Cython	30	0.464987	0.017755	0.458790	0.551632
9M steps Python	30	8.504226	0.022183	8.457839	8.584707
9M steps Cython	30	0.521150	0.005105	0.516207	0.543020



Gráfica con la reducción del ruido Gaussiano
(valores representativos de cada muestra experimental).

Se realiza una tabla con el cálculo de mejoría en los tiempos de ejecución de Cython respecto a Python con los promedios de los tiempos de acuerdo a cada experimentación:

Steps (Millones)	Mejoria a partir de optimización en Cython
3M steps	Cython es aproximadamente 15.910026 veces mas rapido que Python
4M steps	Cython es aproximadamente 15.824291 veces mas rapido que Python
5M steps	Cython es aproximadamente 16.106958 veces mas rapido que Python
6M steps	Cython es aproximadamente 16.306372 veces mas rapido que Python
7M steps	Cython es aproximadamente 15.803079 veces mas rapido que Python
8M steps	Cython es aproximadamente 15.370795 veces mas rapido que Python
9M steps	Cython es aproximadamente 16.318192 veces mas rapido que Python

Conclusiones:

A partir de la experimentación se puede concluir que:

1. El desempeño en tiempos de ejecución es sustancialmente amplio analizando la optimización realizada en Cython.
2. Si bien la mejora es bastante considerable respecto a los tiempos de Python, estos podrían mejorar si se profundiza un poco más en la optimización del código, cabe recordar que dicha optimización para este experimento fue bastante superficial.
3. Es recomendable hacer un análisis respecto al tiempo invertido en la optimización del código respecto al rendimiento conseguido, es decir, encontrar un punto de equilibrio dado que a mayor profundización en la optimización, mayor será el tiempo invertido lo cual a partir de un punto llega a ser ineficiente dado que el aumento en el rendimiento (tiempo de ejecución) será cada vez menor.
4. Si se requiere un mayor rendimiento en los tiempos ejecución es recomendable hacer uso de librerías propias de C, tales como MPI u OpenMP dado que tenemos acceso a ellas si estamos trabajando con Cython, las cuales sirven para aprovechar el uso de los cores a disponer del procesador según la arquitectura del procesador en el cual se está ejecutando la aplicación e incluso interconectar nodos adicionales.
5. De momento el comportamiento de mejoría es lineal, lo cual podría variar teniendo en cuenta el punto anterior, es decir, si usaramos computación en paralelo lo cual sería de gran utilidad al correr aplicaciones con mayor exigencia computacional.
6. Para la experimentación es recomendable aislar lo más que se pueda el computador el cual está corriendo la aplicación, esto con el fin de minimizar errores y la varianza entre resultados causados por tareas secundarias que puedan influir o alterar los tiempos de ejecución.

Referencias bibliográficas:

- https://cython.org/