**Programming Assignment 1: Python Performance with and without NumPy**

1. Introducción

Actualmente existen numerosos lenguajes de programación que llevan décadas en funcionamiento. En la programación contemporánea destacan varios lenguajes pero unos más que otros, y en este caso se hablará de Python. Python es uno de los lenguajes más utilizados por su gran versatilidad. En esa versatilidad se encuentra lo que es su sintaxis intuitiva, la abundancia de bibliotecas y sus habilidades para utilizarse en la automatización, análisis de datos e inteligencia artificial. No obstante, si se entra en el ámbito de operaciones numéricas y la ciencia de la computación junto a los números, se tiene que hablar del protagonismo de NumPy ó Numerical Python dentro del lenguaje Python. NumPy es una biblioteca que maneja de forma eficiente y ventajosa la modificación o manipulación de procedimiento matemático complejo y arreglos de diferentes dimensiones. ¿Será que NumPy supera y potencia a Python en su forma estándar? [1]

En esta tarea se experimentará y explorará las ventajas, diferencias y limitaciones entre NumPy y Python en su forma estándar. Se comparará como ambas herramientas se complementan para desarrollarse en diferentes áreas. En este caso, se evaluará la eficiencia en la gestión y manejo de operaciones matemáticas que implican arreglos multidimensionales. Para comparar el desempeño de Python puro, frente a la librería NumPy, se utilizó diferentes dimensiones para los vectores que van desde 10 hasta 500 elementos y para las matrices van desde 10x10 hasta 500x500. ¿Habrá alguna diferencia la gestión eficiente de memoria y la optimización del tiempo de ejecución?

Se formula que NumPy es significativamente más eficiente a Python estándar cuando se habla de operaciones de matemática compleja y a la gestión de alto volumen de datos. Esto se debe, a sus distintas optimizaciones en C/Fortran y su manejo efectivo de memoria contigua, evitando así sobrecargas en las estructuras dinámicas de Python. Python solo será útil en operaciones de menor gestión de datos y cálculos iterativos; en casos de manejo de alto volúmenes de datos, será menos eficiente por su mala gestión de recursos.

1. Organización estructural
   1. Enfoques de implementación

Para realizar el experimento se establecieron dos enfoques de implementación. Cuando se habla de Python estándar se tiene la utilización de bucles anidados para realizar la multiplicación de matrices y vectores. Esta tarea de índole sencillo hará que la complejidad temporal sea elevada. Por otro lado, cuando se tiene en cuenta a NumPy, se utilizó operaciones vectorizadas de su misma biblioteca, donde se aprovecha en su totalidad la contigüidad en memoria para que su procesamiento sea más rápido y su tiempo de ejecución sea menor. Esto ocurre por la procedencia de sus rutinas, C/Fortran. [3]

* 1. Estructura del código

El benchmarking se implementó con el módulo time para la medición de tiempos de ejecución en cada iteración. La lógica de la implementación se basa en lo siguiente:

* Dimensiones de Arreglos:
  + Vectores (1D) tamaños 10, 50, 100, 200, 500.
  + Matrices (2D) tamaños 10x10, 100x100, 200x200, 500x500.
* Tiempos:
  + Se utilizó time.time() antes de la operación de multiplicación y después.
  + Se computo el promedio de varias ejecuciones para mejor precisión.
* Gráficos:
  + Se generaron gráficos con Matplotlib para el análisis de los tiempos de ejecución.
  1. Código implementado:

<file:///Users/shadiellopez/Downloads/Assignment1.ipynb>

1. Resultados

Luego de realizar e implementar el código se obtuvieron unos resultados claves y claros. En la tabla 1, adjuntada en el documento, se tabulan los tiempos promedios de ejecución para cada prueba y diferentes tamaños de los arreglos. Al realizar varias compilaciones, estos fueron los valores en segundos:

A table with numbers and numbers

AI-generated content may be incorrect.

Tabla 1. Tiempo promedio de ejecución para distintas dimensiones

A graph with a line and a line

AI-generated content may be incorrect.

Grafica 1. Rendimiento Multiplicación 1D

* 1. Análisis de resultados en 1D

En los resultados de la tabla 1 y la gráfica 1 se puede notar que para tamaños menores, tales como 10 y 50 elementos, Python estándar no se aleja de forma abrupta de los tiempos de ejecución comparados con NumPy. Es decir, NumPy mantiene una pequeña ventaja por la sencillez de las operaciones realizadas. No obstante, mientras el tamaño del vector sigue aumentando (100, 200 y 500 elementos), el tiempo de ejecución de Python estándar aumenta de manera abrupta. Por otro lado, los tiempos de ejecución de NumPy se mantienen oscilando en el mismo valor. Por ende, se puede ver la eficacia de las optimizaciones internas y el uso eficiente de contigüidad de memoria.

A graph with a line and a line

AI-generated content may be incorrect.

Grafica 2. Rendimiento Multiplicación 2D

* 1. Análisis de resultados en 2D

En este caso se discutirán los resultados de la multiplicación 2D de la tabla 1 y la gráfica 2, donde se puede notar que Python estándar necesita la implementación de tres bucles anidados. Por ende, su tiempo de ejecución aumentará de manera rápida. Sin embargo, NumPy utiliza las funciones vectorizadas que están optimizad as (np.dot), haciendo que su tiempo de ejecución sea mucho menor. Esto se puede ver en los resultados en matrices 500x500, NumPy cuenta con un tiempo de ejecución de 0.00054 s, mientras que Python estándar apenas alcanza 0.030059 s. Se puede observar que existe una diferencia de tiempo de ejecución muy notable. En ambos casos, resultados en 1D y 2D, el tiempo de ejecución de Python estándar aumenta de manera casi exponencial con el tamaño de la matriz, mientras que el tiempo de ejecución de NumPy permanece casi constante.

1. Discusión
   1. Razones por las cuales NumPy es más eficiente y rápido

La razón principal por la cual NumPy es más rápido es por la implementación de las librerías en C y Fortran, las cuales hacen que las operaciones matemáticas se ejecuten de manera rápida y eficiente, en comparación a Python estándar. Por otro lado, NumPy cuenta con el almacenamiento contiguo de datos junto con la vectorización. Esto permite que procese bloques abundantes de información en un solo llamado a funciones optimizadas, evitando así la sobrecarga de los bucles.

* 1. Limitaciones

NumPy es superior en la gestión de alto volumen de datos, pero aun así Python estándar cuenta con varias ventajas en algunos contextos:

* Cuando se trata de datos pequeños, Python estándar es más directo e intuitivo por sí solo. No necesita de librerías externas.
* Costo computacional: Utilizar NumPy crea una sobrecarga por su implementación y configuración en su biblioteca. Aunque no sea visible en aplicaciones de alto volumen, en scripts para fines didácticos puede llegar a ser muy notable.

Resumiendo los datos discutidos, se debe tomar en consideración la necesidad de la optimización en el procesamiento y el tamaño de los datos que se fueran a utilizar.

1. Conclusiones y Aprendizajes

En conclusión, los resultados obtenidos reafirman y ratifican la hipótesis propuesta. NumPy confirma un rendimiento y optimización significativamente superior en las operaciones complejas numéricas en alto volumen comparado con Python estándar. La ejecución de operaciones vectorizadas, su manejo eficiente de memoria contigua junto al paralelismo, permiten que NumPy sea una herramienta necesaria para la manipulación de alta demanda de datos, como en las aplicaciones de inteligencia artificial y aprendizaje automático. Cabe mencionar que su gran agilidad, eficiencia y capacidad hizo que su tiempo de ejecución en las pruebas realizadas fueran casi lineales. [2]

La discusión de los datos informa como Python estándar es útil para aplicaciones simples o manejo de datos regulado . No obstante, por su naturaleza interpretada junto a la falta de las optimizaciones internas hacen que sea bastante ineficiente cuando se trata de grandes cantidades de datos. Esto hizo que su tiempo de ejecución en las pruebas realizadas aumentara de manera casi exponencial. Por ende, NumPy ofrece una eficiencia y escalabilidad significativa aunque requiera de un costo inicial en configuración.

Este experimento ha marcado la necesidad e importancia de elegir con sabiduría la herramienta adecuada dependiendo los requerimientos y contexto. También, resaltó la relevancia y necesidad de las optimizaciones de bajo nivel en el desarrollo de aplicaciones de ingeniería. Por otro lado, se pudo comprender la asociación entre la sobrecarga y los bucles de Python con o sin vectorización. Por último, la necesidad o importancia de la visualización de los resultados junto el trabajar en equipo para un análisis completo.

1. Referencias:

[1] “Python (programming language),” Wikipedia, https://en.wikipedia.org/wiki/Python\_(programming\_language) (accessed Feb. 19, 2025).

[2] Veyak, “Numpy vs traditional python lists: A performance showdown,” Medium, https://medium.com/@vakgul/numpy-vs-traditional-python-lists-a-performance-showdown-1e8bebc55933 (accessed Feb. 19, 2025).

[3] “W3schools.com,” Introduction to NumPy, https://www.w3schools.com/python/numpy/numpy\_intro.asp (accessed Feb. 19, 2025).