Análisis de rendimiento aplicando concurrencia en Python

Cómputo de Alto Rendimiento con Lenguajes de Alto Nivel
Miniproyecto 1

Miguel Ángel Veloz Lucas *[1]

[a] *Universidad Nacional Autónoma de México*
[b] *Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas*
[c] *Posgrado en Ciencia e Ingeniería de la Computación*

Resumen

La computación concurrente es una forma de computación en la cual varios cálculos se ejecutan simultáneamente en lugar de secuencialmente. Formalmente, la computación concurrente se define como una forma de computación donde las ejecuciones se producen en tiempos de ejecución solapados, esto es "concurrentemente", en vez de ejecutarse secuencialmente (uno completando su ejecución antes de que el siguiente empiece con la suya propia). El principal problema de la programación concurrente corresponde a no saber en que orden se ejecutan los programas. Se debe tener especial cuidado en que este orden no afecte el resultado de los programas. Distintos lenguajes de programación están diseñados para admitir la programación concurrente mediante el uso de subprocesos. Los objetos y recursos pueden ser accedidos por múltiples hilos; cada subproceso puede potencialmente acceder a cualquier objeto en el programa y el usuario debe garantizar que los accesos de lectura y escritura a los objetos estén correctamente sincronizados entre los subprocesos.

Palabras clave: Concurrecia. Threads. Speed-up. Python.

Abstract

Definición: Este proyecto consiste en aplicar técnicas de cómputo concurrente para medir el rendimiento de la multiplicación de matrices utilizando el lenguaje Python, también se comprobará la eficiencia de este lenguaje de programación sin la implementación de bibliotecas para realizar tareas de cómputo científico.

Método: La forma en como esta desarrollado el ejercicio consiste en: 1. Una introducción que resume a grandes rasgos el tema de concurrencia y el lenguaje Python como herramienta para el cómputo científico; 2. La implementación de un programa secuencial para probar y medir la multiplicación de matrices que varían de dimensiones de 100, 300 y 1000 elementos por fila; 3. Se refactoriza la implementación secuencial de la multiplicación y se genera un algoritmo concurrente utilizando Threads para realizar la multiplicación de matrices variando el número de hilos y los elementos de cada matriz, donde a cada hilo realizá el cálculo de un subconjunto del producto de la matriz; 4. Se Modifica la implementación del algoritmo concurrente para realizar mediciones del tiempo de ejecución y proceder a calcular el rendimiento para cada uno de los casos.

Resultados: Los resultados que se obtienen al refactorizar el algoritmo concurrente y al realizar las mediciones de tiempo de ejecución se muestran con gráficos tanto para el speed-up como para la eficiencia.

Conclusiones: Teniendo en cuenta la presencia del lenguaje Python en la implementación de proyectos de cómputo científico, sabemos que ofrece muchas herramientas que facilita la realización de tareas orientadas a esta área. Pero también sabemos que muchas de estas herramientas que ofrece el lenguaje están basadas en otros lenguajes de programación compilados que complementan las limitaciones de Python debido a su naturaleza de lenguaje interpretado. Si bien la refactorización de una tarea utilizando hilos (threads) beneficia el rendimiento y la ejecución de un programa, esta también depende del lenguaje de programación y su velocidad de respuesta, como en este caso que el lenguaje Python por si solo no da el rendimiento que se desearía en la ejecución de un problema aritmético relativamente sencillo como es la multiplicación de matrices.

Keywords: Concurrencia. Python. Rendimiento.

*Email: miguel.velozlucas@gmail.com

Textolivre
Linguagem e Tecnologia

Publicado el:
18 de marzo de 2022

Esta obra está bajo una licencia «CC BY 4.0».

1 Introducción

La computación concurrente es una forma de computación en la cual varios cálculos se ejecutan simultáneamente en lugar de secuencialmente. Formalmente, la computación concurrente se define como una forma de computación donde las ejecuciones se producen en tiempos de ejecución solapados, esto es "concurrentemente", en vez de ejecutarse secuencialmente (uno completando su ejecución antes de que el siguiente empiece con la suya propia). El principal problema de la programación concurrente corresponde a no saber en que orden se ejecutan los programas. Se debe tener especial cuidado en que este orden no afecte el resultado de los programas.

Distintos lenguajes de programación están diseñados para admitir la programación concurrente mediante el uso de subprocesos. Los objetos y recursos pueden ser accedidos por múltiples hilos; cada subproceso puede potencialmente acceder a cualquier objeto en el programa y el usuario debe garantizar que los accesos de lectura y escritura a los objetos estén correctamente sincronizados entre los subprocesos.

Si los diferentes procesos de un programa concurrente tienen acceso a variables globales o secciones de memoria comunes, la transferencia de datos a través de ella es una vía habitual de comunicación y sincronización entre ellos. Las primitivas para programación concurrente basada en memoria compartida resuelven los problemas de sincronización entre procesos y de exclusión mutua utilizando la semántica de acceso a memoria compartida.

El lenguaje de programación Python para ciencia de datos proporciona todas las herramientas necesarias para llevar a cabo este tipo de proceso de manera efectiva con bibliotecas dedicadas para cada caso, aunque cabe señalar que algunas bibliotecas son construidas en lenguajes como c o c++ como lo comentaremos más adelante en este artículo.

2 Implementación secuencial

Para comenzar implementamos un programa secuencial que calcule el producto de dos matrices (considerablemente grandes) iniciando con dimensiones de 100 x 100, 300 x 300 hasta 1000 x 1000. Esto para obtener una referencia del tiempo de ejecución al generar estos productos.

3 Refactorización del algoritmo

Con base en el algoritmo secuencial para obtener el producto de matrices, reimplementamos la idea para generar un programa concurrente que asigne a un hilo el calculo de un subconjunto del producto de la multiplicación de matrices.

Con la intención de agilizar la ejecución de la multiplicación de matrices se definieron funciones recursivas que realizan el cálculo de los subconjuntos asignados a cada hilo. Esto, sin embargo, no tuvo mucho impacto en la reducción de los tiempos de ejecución que se verá al obtener el cálculo de rendimiento de las ejecuciones.

4 Medición de rendimiento

Para realizar un análisis de rendimiento se utilizó la función timeit.timeit() de la biblioteca timeit, que aporta, entre otras características, el tiempo total en que se ejecuta una función repetidas n veces.

Para obtener las métricas de rendimiento iniciamos la ejecutando la multiplicación de matrices secuencial, variando la dimensión de las matrices y con la función timeit() damos la instrucción de que cada multiplicación se realice 5 veces para obtener un promedio de tiempo del tiempo de ejecución para cada producto.

Continuamos obteniendo los tiempos de ejecución pero ahora para el algoritmo concurrente. De manera análoga en como se obtuvieron los tiempos de la multiplicación secuencial, se varía las dimensiones de las matrices, pero ahora también variaremos el número de hilos en los que se repartirá el cálculo del producto de las matrices, y junto a esto también se da la instrucción de realizar cinco veces cada multiplicación para obtener un promedio de cada producto utilizando la función timeit().

5 Resultados

El resultado de los tiempos de ejecución se almacenaron en dos arreglos, el primero para los tiempos obtenidos de la multiplicación secuencial y el segundo para el algoritmo concurrente. De estos se cálculo el rendimiento aplicando las siguientes formulas.

$$S_p = \frac{t_s}{t_p} \qquad E_p = \frac{S_p}{p} = \frac{t_s}{p\,t_p}$$

(a) Speed-up (b) Eficiencia

Figura 1. Formulas para el cálculo de la aceleración y eficiencia de un programa que implementa concurrencia. Donde ts es el tiempo de ejecución secuencial, y tp es el tiempo de ejecución en paralelo con p procesadores o hilos

Con base en las formulas anteriores y utilizando los datos de tiempos de ejecución recopilados se obtuvieron los siguientes resultados mostrados en la Figura 2, donde las columnas indican el número de hilos en los que se reparte el cálculo del producto de matrices y las filas contienen el resultado al calcular speed-up y eficiencia.

Tabla de rendimiento

	1	2	4	8	16
100 - S	1.1414	1.1499	1.1147	1.0468	0.9023
100 - E	1.1414	0.5749	0.2787	0.1308	0.0564
300 - S	1.1765	1.2135	1.2020	1.1911	1.2095
300 - E	1.1765	0.6068	0.3005	0.1489	0.0756
1000 - S	1.2773	1.2619	1.2672	1.2983	1.2545
1000 - E	1.2773	0.6310	0.3168	0.1623	0.0784

Figura 2. Tabla de rendimiento variando el número de hilos de 1 a 16 y las dimensiones las matrices de 100 x 100, 300 x 300 y 1000 x 1000

Fuente: https://github.com/mvelozlucas/HPC-LA–2022-II

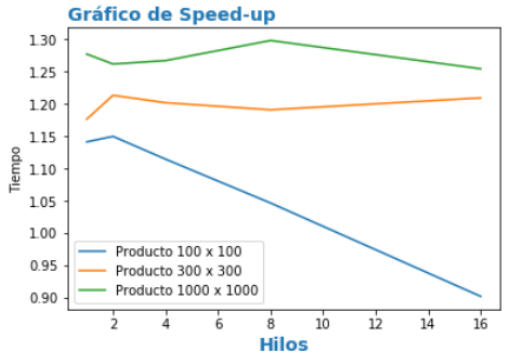

Figura 3. Gráfica de speed-up variando el número de hilos de 1 a 16 y las dimensiones las matrices de 100 x 100, 300 x 300 y 1000 x 1000

Fuente: https://github.com/mvelozlucas/HPC-LA–2022-II

Al revisar los gráficos de speed-up y eficiencia notamos que no hay mucha variación entre los tiempos de ejecución utilizando de 1 a 16 hilos, generando ejecuciones que llegan a demorarse prácticamente el mismo tiempo que una ejecución secuencial.

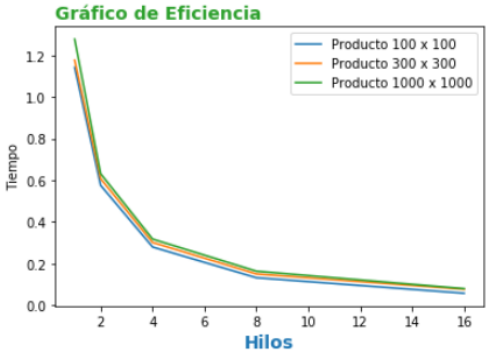

Figura 4. Gráfica de eficiencia variando el número de hilos de 1 a 16 y las dimensiones las matrices de 100 x 100, 300 x 300 y 1000 x 1000

Fuente: https://github.com/mvelozlucas/HPC-LA–2022-II

6 Concluiones

Teniendo en cuenta la presencia del lenguaje Python en la implementación de proyectos de cómputo científico, sabemos que ofrece muchas herramientas que facilita la realización de tareas orientadas a esta área. Pero tambien sabemos que muchas de estas herramientas que ofrece el lenguaje están basadas en otros lenguajes de programación compilados que complementan las limitaciones de Python debido a su naturaleza de lenguaje interpretado. Si bien la refactorización de una tarea utilizando hilos (threads) beneficia el rendimiento y la ejecución de un programa, esta también depende del lenguaje de programación y su velocidad de respuesta, como en este caso que el lenguaje Python por si solo no da el rendimiento que se desearía en la ejecución de un problema aritmético relativamente sencillo como es la multiplicación de matrices.

7 Referencias

STALLINGS W.,Sistemas Operativos, 2ed.PRENTICEHALL. Madrid, 1997.

SILBERSCHATZ A.,Fundamentos de sistemas operativos, 7ad. MacGrawHill, 2005.