Universidad del País Vasco
Euskal Herriko Unibertsitatea

Heurísticos de Búsqueda

Tema 1 - Operaciones vectoriales y complejidad en tiempo y memoria

Josu Ceberio e Inigo Lopez-Gazpio

07/10/2023

Índice

1. Enunciado

En el amplio campo de la programación, el cómo se abordan los problemas computacionales puede marcar la diferencia entre una solución eficiente y una que consume recursos desmedidamente. Uno de los enfoques más eficaces para la optimización es a través del uso de código vectorizado.

En comparación con el código no optimizado, el código vectorizado aprovecha las capacidades avanzadas de los modernos procesadores, permitiendo que se realicen múltiples operaciones simultáneamente en distintos datos. Por ejemplo, al trabajar con grandes matrices o arrays, en lugar de usar bucles que procesen cada elemento individualmente, el código vectorizado puede procesar varios elementos a la vez, reduciendo drásticamente el tiempo necesario para completar la operación.

Esta eficiencia no solo se refleja en una ejecución más rápida. En términos de complejidad, mientras que un código no optimizado podría tener una complejidad temporal creciente, el código vectorizado puede, en muchos casos, mantenerla constante o logarítmica. Además, al reducir la necesidad de operaciones repetitivas, se consume menos memoria, lo que es esencial cuando se trabaja con conjuntos de datos grandes o en sistemas con recursos limitados.

No obstante, saber si estamos realmente optimizando nuestro código requiere herramientas específicas. Aquí es donde entran las herramientas de profiling. Estas herramientas permiten a los desarrolladores analizar en detalle el comportamiento de sus programas, identificando las secciones de código que más tiempo consumen o que usan más memoria. Mediante el profiling, es posible aislar y mejorar esas áreas problemáticas, garantizando así que el código no solo funcione, sino que lo haga de la manera más eficiente posible.

El objetivo de este laboratorio no es más que el de ser consciente de que si no programamos de manera eficiente nuestros algoritmos de búsqueda van a ser muy ineficaces,

para ello puedes elegir el problema que más te guste, e implementarlo de distintas maneras para comparar sus tiempos de ejecución y uso de memoria. Como ejemplo, te propongo que elijas algunos de los siguientes problemas:

- Función fitness del cálculo de la desviación en los contenedores
- Multiplicación de matrices
- Quadratic Assignment Problem (QAP)
- Graph Partitioning Problem (GPP)

La implementación ineficiente será una implementación realizada mediante bucles mientras que en la implementación eficiente deberéis de utilizar la librería **numpy**.

2. Código auxiliar

Para medir los tiempos de ejecución de código python puedes utilizar la librería **time**:

```
time0 = time.time()
....
time1 = time.time()
print ("Tiempo ejecución: {}".format(time1 - time0))
```

También podrás realizar algunos cálculos más complejos (uso de memoria y llamadas al sistema) utilizando la librería de profiling. Puedes encontrar algunos ejemplos aquí y aquí.

Existe también un amplio abanico de herramientas de terminal para realizar observaciones de profiling, como: time, top, htop, etc.

Tienes disponible un jupyter-notebook auxiliar aquí. Fíjate sobre todo en la parte de experimentación y en el uso de la librería **pandas** para almacenar la información y realizar algún **plot básico**. Es importante que vayas trabajando estas librerías ya que van a ser parte de la evaluación.

3. Entrega

Entregar un documento pdf que contenga el pseudocódigo del problema elegido y una gráfica en el que se compare la complejidad en tiempo y memoria de los distintos algoritmos.

4. Rúbrica

Esta entrega no tiene una calificación directa asociada.