|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Carlos Jesús Cortés Juárez | 01-julio-2024 | |  |  | |
| Extraordinario Computación Paralela  Materia: Computación Paralela  Código: 19110044  Carrera: Tecnologías de Software |

# Extraordinario



Proyecto “Análisis de imágenes médicas.”

**Problema:** Imaginemos que estamos trabajando en un hospital donde se generan y analizan imágenes médicas de alta resolución, como resonancias magnéticas (MRI) o tomografías computarizadas (CT). El proceso de análisis de estas imágenes puede ser intensivo en términos computacionales y puede beneficiarse significativamente de la computación paralela para mejorar la eficiencia y el rendimiento del diagnóstico médico.

La computación paralela puede acelerar operaciones como la segmentación de órganos, detección de anomalías o reconstrucción 3D a partir de imágenes médicas mediante la distribución de la carga de trabajo entre múltiples núcleos de CPU o incluso el uso de GPUs. Esto no solo mejora los tiempos de respuesta para los médicos, sino que también permite procesar mayores volúmenes de datos en menos tiempo.

**Configuración del entorno:**

* **CPU:** Se utilizará un sistema con al menos un procesador multi-core (como Intel Core i7 o superior).
* **GPU:** Opcionalmente, se aprovechará una GPU NVIDIA compatible con CUDA para aceleración de tareas específicas.
* **Sistema operativo:** se utilizará UNBUNTU LINUX como sistema operativo principal.

**Sus bibliotecas:**

* **Python 3.8+** como lenguaje de programación.
* **NumPy** y **SciPy** para operaciones numéricas y procesamiento de imágenes.
* **OpenCV** para manipulación y procesamiento de imágenes médicas.
* **CUDA Toolkit** y **CuPy** para la programación en GPU (opcional).

**Archivos requeriments.txt**



**Código de phyton en multiprocesing:**

import multiprocessing

import numpy as np

def process\_image\_chunk(image\_chunk):

# Procesamiento intensivo de un trozo de imagen

result\_chunk = image\_chunk \* 2 # Ejemplo de operación intensiva

return result\_chunk

def parallel\_process\_image(image\_data, num\_processes):

pool = multiprocessing.Pool(num\_processes)

results = pool.map(process\_image\_chunk, image\_data)

pool.close()

pool.join()

return results

# Ejemplo de uso

if \_\_name\_\_ == '\_\_main\_\_':

image\_data = np.random.rand(10, 10) # Datos de imagen ficticios

num\_processes = 4

processed\_results = parallel\_process\_image(image\_data, num\_processes)

print(processed\_results)

**Código en phyton para Multihilo:**

import threading

import numpy as np

def process\_image\_chunk(image\_chunk):

# Procesamiento intensivo de un trozo de imagen

result\_chunk = image\_chunk \* 2 # Ejemplo de operación intensiva

return result\_chunk

def parallel\_process\_image(image\_data, num\_threads):

results = []

threads = []

def worker(image\_chunk):

result = process\_image\_chunk(image\_chunk)

results.append(result)

for i in range(num\_threads):

thread = threading.Thread(target=worker, args=(image\_data[i],))

threads.append(thread)

thread.start()

for thread in threads:

thread.join()

return results

# Ejemplo de uso

if \_\_name\_\_ == '\_\_main\_\_':

image\_data = np.random.rand(10, 10) # Datos de imagen ficticios

num\_threads = 4

processed\_results = parallel\_process\_image(image\_data, num\_threads)

print(processed\_results)

**Código de phyton para GPU**

import cupy as cp

# Función de ejemplo utilizando CuPy para aceleración en GPU

def gpu\_process\_image(image\_data):

image\_data\_gpu = cp.asarray(image\_data)

result\_gpu = image\_data\_gpu \* 2 # Ejemplo de operación intensiva en GPU

return cp.asnumpy(result\_gpu)

# Ejemplo de uso

if \_\_name\_\_ == '\_\_main\_\_':

image\_data = np.random.rand(10, 10) # Datos de imagen ficticios

processed\_results\_gpu = gpu\_process\_image(image\_data)

print(processed\_results\_gpu)

### **Sincronización y control de concurrencia:**

Para evitar condiciones de carrera y otros problemas de concurrencia en Python, se pueden usar bloqueos (Lock) o semáforos (Semaphore) según sea necesario. Es crucial manejar adecuadamente el acceso a recursos compartidos entre hilos o procesos para asegurar la integridad de los datos.

**Análisis de Rendimiento:**

Se realizarán pruebas de rendimiento utilizando herramientas como timeit en Python para medir los tiempos de ejecución de las implementaciones paralelas en comparación con versiones secuenciales de los algoritmos. Además, se analizará el uso de recursos del sistema (CPU y GPU) para evaluar la eficiencia de cada enfoque.

**Documentación:**

Todo el código estará debidamente documentado utilizando comentarios claros y descriptivos. Se proporcionará un archivo README.md que describa el proyecto, cómo configurarlo, ejecutarlo y cómo interpretar los resultados obtenidos de las pruebas de rendimiento.

Este proyecto no solo mostrará cómo implementar computación paralela en Python para el procesamiento de imágenes médicas, sino que también destacará las diferencias clave entre el uso de multihilo y multiprocesamiento, así como la aceleración potencial utilizando GPUs con CUDA.