TAREA 1. ANÁLISIS DE RENDIMENTO Y PARALELIZACIÓN DE UNA APLICACIÓN

El objetivo de esta tarea consiste en aplicar las técnicas de profiling y paralelización vistas en clase sobre un código de vuestro interés desarrollado durante alguna de las asignaturas que habéis cursado en el grado. Este código no debe ser demasiado complejo y debe estar escrito en Python sin llamadas a librerías externas.

Si no dispones de un código concreto que te pueda servir, puedes utilizar un código de respaldo que os proporcionamos (`backup_code_sobel_edge_detection_clean`). Este código localiza los bordes de una imagen mediante un operador Sobel a partir de convoluciones (https://es.wikipedia.org/wiki/Operador_Sobel).

1. Descripción

Tomando como punto de partida un código Python, esta tarea consiste en realizar los siguientes pasos:

Paso 1. Realizar un profiling del código con el objetivo de localizar las funciones que más tiempo de ejecución consumen. Para ello se deberán emplear la/s herramienta/s vistas en las sesiones prácticas de la asignatura.

Paso 2. Paralelizar dichas funciones usando la librería `multiprocessing` tal y como se ha visto en clase de prácticas. Puedes aplicar distintas estrategias para dividir el trabajo entre diferentes procesos. Por ejemplo, puedes aplicarlas dividiendo la imagen verticalmente entre los distintos procesos (por ejemplo, si la imagen tiene 400 filas y tenemos 4 procesos, asignando 100 a cada proceso).

Paso 3. Optimizar el programa inicial (secuencial) por medio de librerías especializadas (p.ej. OpenCV) para implementar de manera más optimizada la funcionalidad que se necesita (p.ej. una convolución o una llamada directa a Sobel) en sustitución de las funciones que más tiempo consumen detectadas en el paso 1.

Estas librerías ejecutan código nativo pre-compilado (binarios) que es más eficiente en ILP que el código interpretado de Python, y además suelen están paralelizadas y vectorizadas. Por ejemplo, para la librería OpenCV podemos obtener la información de compilación:

```
# pip install opencv-python

  import cv2
  build_info = cv2.getBuildInformation()
  print(build_info)
```

En la salida nos dirá las extensiones SIMD soportadas y el framework de paralelización empleado. Diferentes librerías tendrán diferentes llamadas para obtener la información de compilación (lamentablemente no existe un estándar).

Paso 4: De manera opcional, el alumno podrá desarrollar alguna/s actividad/es adicional/es relacionadas con el análisis y optimización del código empleado por el alumno para desarrollar este trabajo. Se proponen las siguientes:

[OP1] Hacer profiling a más bajo nivel analizando contadores de programa usando `pylikwid`. Esta actividad opcional requiere de ejecución en hardware real y permisos de superusuario.

- En la dirección https://www.ditec.um.es/~jcebrian/ppd/Instrucciones.txt tenéis las instrucciones para crear un USB de arranque basado en Ubuntu 22.04 y poder ejecutar vuestras aplicaciones en un entorno Ubuntu sin instalación (arrancando desde el USB).
- Las instrucciones describen la instalación de `likwid` y `pylikwid` en el entorno Ubuntu 22.04 mencionado en el apartado anterior.

[OP2] Analizar la escalabilidad de código según el número de procesos/núcleos empleados. Al igual que la anterior, esta actividad requiere de ejecución local del notebook para acceder a los núcleos físicos de tu máquina o remota en un servicio con varios núcleos de ejecución físicos. Para ello puedes:

- Ejecutar el código en manera local empleando una instalación VSCode en tu ordenador. Se recomienda emplear WSL, aunque también es posible realizar pruebas sobre Windows.
- Alternativamente, también puedes emplear el USB de arranque Ubuntu 22.04 necesario para la actividad opcional OP1.

[OP3] Uso de librerías estándar (p.ej. `numba`) que combinadas con `multiprocessing` permita reducir el tiempo de ejecución del código.

[OP4] Evaluar el uso de otras librerías alternativas a `multiprocessing` que permitan la ejecución paralela en Python (p.ej. joblib, PyOMP, etc.) de aquellas funciones más pesadas.

[OP5] Otra actividad propuesta por el alumno.

2. Criterios de evaluación

Se entregará un documento con portada e índice donde se describa el proceso completo de la realización de la tarea, acompañado de capturas de pantalla de los pasos más importantes. Así mismo, se debe entregar el código fuente original empleado para desarrollar la tarea, junto con las diferentes versiones optimizadas que ha desarrollado para cumplir con los distintos pasos descritos anteriormente .

- Corrección de la tarea: 5 puntos
- Cumplir plazo de entrega: 1 punto
- Calidad del documento: 2 puntos
- Realización de actividad/es adicional/es: 2 puntos

3. Normas de entrega

La fecha de entrega de esta tarea queda fijada para el **2 de Marzo**, hasta las **23:55 horas**.

Se admitirán entregas después de esta fecha, aunque tendrán una penalización de 1 punto por incumplir el plazo de entrega y no se recibirá la valoración por parte del profesorado durante el desarrollo del cuatrimestre. Se establece como fecha máxima para la entrega de esta tarea el día de realización del examen final de la asignatura de la convocatoria ordinaria de Junio (**18 de Junio de 2025**).

La entrega de la documentación se realizará a través del **Aul@ Virtual**, a través de una tarea que se encuentra activa para tal efecto en el curso de la asignatura.