Práctica 3
Paralelismo a nivel de hilos: Paralelización mediante OpenMP y programación asíncrona del análisis forense de manipulación de imágenes digitales

Objetivos:

- Aprender a paralelizar una aplicación en una máquina paralela de memoria centralizada mediante hilos (*threads*) usando el estilo de *variables compartidas*.
- Estudiar la API de OpenMP y aplicar distintas estrategias de paralelismo en su aplicación.
- Estudiar la API de C++ de programación asíncrona para explotar el paralelismo funcional.
- Aplicar métodos y técnicas propios de esta asignatura para estimar las ganancias máximas y la eficiencia del proceso de paralelización.
- Aplicar todo lo anterior a un problema de complejidad y envergadura suficiente.

Desarrollo:
En esta práctica, tendréis que paralelizar, usando OpenMP y programación asíncrona, la solución a un problema dado para aprovechar los distintos núcleos de los que dispone cada ordenador de prácticas. Se paralelizará por tanto para un sistema multiprocesador (máquina paralela de memoria centralizada), en el que todos los núcleos de un mismo encapsulado ven la misma memoria, es decir, un puntero en un núcleo es el mismo puntero para el resto de los núcleos del microprocesador.

Tarea 0.1 Entrenamiento previo OpenMP:

Observa el siguiente programa en C donde se suman dos vectores de *floats* empleando OpenMP para paralelizar el cálculo.

```
#include <omp.h>
#define N 1000
#define CHUNKSIZE 100

main(int argc, char *argv[]) {

    int i, chunk;
    float a[N], b[N], c[N];

    /* Inicializamos los vectores */
    for (i=0; i < N; i++)
         a[i] = b[i] = i * 1.0;
    chunk = CHUNKSIZE;

    #pragma omp parallel shared(a,b,c,chunk) private(i)
    {
         #pragma omp for schedule(dynamic,chunk) nowait
         for (i=0; i < N; i++)
                c[i] = a[i] + b[i];
    }   /* end of parallel region */

}
```

Revisa la documentación de OpenMP (API, tutoriales, este enlace: https://computing.llnl.gov/tutorials/openMP/, etc...) y responda:

0.1.1 ¿Para qué sirve la variable **`chunk`**?

0.1.2 Explica **completamente** el *pragma* :

```
#pragma omp parallel shared(a,b,c,chunk) private(i)
```

- ¿Por qué y para qué se usa **`shared(a,b,c,chunk)`** en este programa?
- ¿Por qué la variable `i` está etiquetada como **`private`** en el pragma?

0.1.3 ¿Para qué sirve **`schedule`**? ¿Qué otras posibilidades hay?

0.1.4 ¿Qué tiempos y otras medidas de rendimiento podemos medir en secciones de código paralelizadas con OpenMP?

Tarea 0.2: Entrenamiento previo std::async

Observa el siguiente programa en c++ donde se llama a dos funciones con std::async:

```
#include <iostream>
#include <future>
#include <chrono>

int task(int id, int millis) {
    std::this_thread::sleep_for(std::chrono::milliseconds(millis));
    std::cout<<"Task "<<id<<" completed"<<std::endl;
    return id;
}

int main() {
    auto start = std::chrono::high_resolution_clock::now();
    std::future<int> task1 = std::async(std::launch::async, task, 1, 2000);
    std::future<int> task2 = std::async(std::launch::async, task, 2, 3000);

    task1.wait();
    int taskId = task2.get();

    auto end = std::chrono::high_resolution_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(end-start);

    std::cout<<"Completed in: "<<elapsed.count()<<"ms"<<std::endl;
}
```

Para compilarlo usa g++ y enlaza con la librería pthreads con -lpthread.
Revisa la documentación de std::async y contesta a las siguientes preguntas:

0.2.1 ¿Para qué sirve el parámetro `std::launch::async`?

0.2.2 Calcula el tiempo que tarda el programa con `std::launch::async` y `std::launch::deferred`. ¿A qué se debe la diferencia de tiempos?

0.2.3 ¿Qué diferencia hay entre los métodos `wait` y `get` de `std::future`?

0.2.4 ¿Qué ventajas ofrece `std::async` frente a `std::thread`?

Tarea 0.3: Entrenamiento previo std::vector

Observa el siguiente programa en c++ donde se inicializa un vector stl y se rellena con valores:

```
#include <vector>
#include <iostream>
```

```
#include <chrono>

int main() {
        // default initialization and add elements with push_back
        auto start = std::chrono::high_resolution_clock::now();

        std::vector<float> v1;
        for (int i = 0; i < 10000; i++)
                v1.push_back(i);

        auto end = std::chrono::high_resolution_clock::now();
        auto elapsed =
        std::chrono::duration_cast<std::chrono::milliseconds>(end-start);
        std::cout<<"Default initialization: "<<elapsed.count()<<"ms"<<std::endl;

        // initialized with required size and add elements with direct access
        start = std::chrono::high_resolution_clock::now();

        std::vector<float> v2(10000);
        for (int i = 0; i < 10000; i++)
                v1[i] = i;
        end = std::chrono::high_resolution_clock::now();

        elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(end-start);
        std::cout<<"Initialization with size: "<<elapsed.count()<<"ms"<<std::endl;
}
```

Responde a las siguientes preguntas:

0.3.1: ¿Cuál de las dos formas de inicializar el vector y rellenarlo es más eficiente? ¿Por qué?

0.3.2: ¿Podría ocurrir algún problema al paralelizar los dos bucles for? ¿Por qué?

Tarea 1: Paralelización del análisis forense de manipulación de imágenes digitales

Todos los grupos de cada turno de prácticas deben acometer la paralelización del problema planteado. Para ello se analizará la solución secuencial facilitada siguiendo las indicaciones del enunciado y los profesores. Posteriormente, se transformará ésta, utilizando OpenMP y programación asíncrona, para que incorpore paralelismo a nivel de hilos.

Problema:

El problema planteado es el análisis forense de manipulación de imágenes digitales.

En la actualidad, las manipulaciones digitales tanto de imágenes como videos son cada vez más frecuentes. Los nuevos algoritmos generadores de deepfake o face swap son capaces de generar imágenes manipuladas prácticamente indistinguibles de una imagen original. Por tanto, es importante saber identificar estas imágenes manipuladas.
La forma clásica de abordar la detección de manipulaciones digitales se basa en el análisis del patrón de ruido. El patrón de ruido que genera cada cámara (o incluso la misma cámara con diferente configuración) hace que cuando se realiza una manipulación cortando y pegando de una imagen a otra haya una discontinuidad en el patrón de ruido que hace visible la manipulación. Una forma sencilla de obtener este patrón de ruido es con una convolución con el kernel Steganalysis Rich Model (SRM):

Fig 1. Al hacer zoom se puede apreciar ligeramente que el patrón de ruido de la región de la cara es diferente al cuerpo.

Sin embargo, la posterior aplicación de algoritmos de compresión con pérdida hace que parte de ese patrón de ruido se vea alterado y puede afectar a este tipo de análisis. Otra forma de detectar las manipulaciones es justamente utilizando el algoritmo de compresión JPEG para detectar discrepancias en zonas de la imagen que hayan sido comprimidas con distinto nivel de compresión. La forma de llevar a cabo este análisis es sencilla, se recomprime la imagen con diferentes niveles de compresión y se restan a la imagen original para ver diferencias. De esta forma se resaltan las zonas de diferente compresión. Este algoritmo se conoce como Error Level Analysis (ELA) y suele evidenciar zonas que provengan de diferentes imágenes comprimidas.

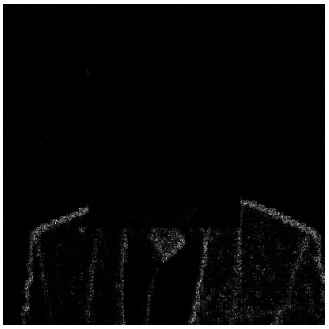

Fig 2. Se observa que en la región de la cara hay un cuadrado negro que representa una zona con la misma compresión mientras que en el cuerpo se aprecian zonas con diferente compresión.

En la actualidad, las manipulaciones digitales ya no se hacen con un simple copiar y pegar si no que se utilizan sofisticados algoritmos de IA generativa capaces de generar imágenes desde cero. Alguno de estos algoritmos es DeepLive Cam que permite realizar face swap en tiempo real a partir de una única foto de una persona. Sin embargo, a pesar de la fidelidad con la que consigue realizar el face swap, para optimizar la ejecución, sólo aplica el cambio a una región alrededor de la cara. Por tanto, podemos analizar la imagen para detectar la zona en la que se ha realizado la manipulación. Una característica que permite identificar a estos tipos de algoritmos es su dificultad para generar patrones de altas frecuencias como es el propio ruido de la imagen. Es por ello que un análisis clásico como los mencionados anteriormente puede revelar las zonas manipuladas de la imagen.

Otra forma de analizar las frecuencias presentes en una imagen es la Transformada Discreta del Coseno (DCT) que descompone la imagen en distintos coeficientes de la

función coseno. Esta transformada se usa en el algoritmo de compresión JPEG ya que tiende a concentrar la mayor parte de la información en un subconjunto de coeficientes reducido. por lo que se pueden eliminar algunos y mantener gran parte de la información. Las bajas frecuencias son las de mayor energía y están localizadas en la esquina superior izquierda mientras que las altas frecuencias tienen menor energía y están localizadas en la esquina inferior derecha. Los algoritmos como DeepLive Cam suelen presentar una menor energía en esta esquina inferior derecha debido a su dificultad para generar altas frecuencias por lo que es una buena forma de determinar la manipulación en la imagen.

Fig 3. Se puede observar ligeramente un cuadrado más oscuro en la región de la cara.

Tarea 1.1: Analiza el código e identifica los distintos procesos

En esta tarea deberéis analizar el código utilizando las estrategias vistas en la práctica anterior para identificar los distintos procesos que se aplican sobre la imagen manipulada y elaborar un diagrama de dependencias.
Identifica si se puede ejecutar algunos procesos en paralelo y las dependencias entre ellos.

Tarea 1.2: Analiza el código de cada proceso y el tiempo

En esta tarea deberéis analizar el código correspondiente a cada proceso identificado en la tarea 3.1. Realiza un diagrama de flujo de cada uno e identificar posibles puntos paralelizables. Ejecuta el código secuencial y calcula el tiempo que tarda cada proceso.

Tarea 1.3 Paralelización

Identificadas las partes paralelizables del código y su coste temporal, ahora toca aplicar el paralelismo con OpenMP y programación asíncrona. Haced pruebas paralelizando diferentes partes y usando diferentes estrategias para observar la ganancia de rendimiento.
Responded a las siguientes preguntas:

- ¿Se obtiene un mejor rendimiento paralelizando todos las partes posibles?
- ¿Se degrada el rendimiento al paralelizar ciertas partes?
- Si es así, ¿a qué creéis que se debe esa degradación?
- ¿Cúal es la mejor estrategia de paralelización?

Objetivos:

- Obtener una versión paralela con OpenMP y std::async de la solución secuencial proporcionada
- Analizar y explotar en todos los casos los diferentes tipos posibles de paralelismo que se perciban en la solución secuencial dada.

- Realizar pruebas suficientes que permitan estimar valores de rendimiento de la versión paralela frente a la secuencial.

Pasos:

a. Analizad la solución secuencial facilitada para asegurar la comprensión del problema. Prestar atención a detalles vistos en prácticas anteriores que tengan importancia para diseñar la solución paralela: variables, talla del problema, etc.
b. Haced una representación gráfica en forma de diagrama de dependencias del funcionamiento de la solución secuencial dada.
c. Utilizando OpenMP y std::async y a la vista de los detalles anteriores, diseña una solución paralela a nivel de hilo a partir de la versión secuencial
d. Haced pruebas de la versión paralela para diferentes casos y configuraciones tomando medidas de tiempos y ganancias.
e. Haced una representación gráfica en forma de grafo de control de flujo del funcionamiento de la solución paralela basada en OpenMP y std::async.
f. Comente los siguientes aspectos tanto de la solución secuencial como de la paralela según se indique:

Sobre el código implementado

- Comentar las porciones de código de más interés tanto de la solución secuencial como de la paralela implementadas:
  - Aspectos que definan la talla del problema.
  - Estructuras de control del código de especial interés en la solución al problema.
  - Es importante que se justifique **lo más detalladamente** posible los cambios que se hayan realizado con respecto a la versión secuencial para paralelizar la solución.
  - Instrucciones y bloques de OpenMP o std::async utilizados para la paralelización del código.

Sobre el paralelismo explotado

- **Revisando la documentación de la "*Unidad 3. Computación paralela*"**. Indique lo siguiente con respecto a su práctica:
  - Tipos de paralelismo usado.
  - Modo de programación paralela.
  - Qué alternativas de comunicación (explícitas o implícitas emplea su programa).
  - Estilo de la programación paralela empleado.
  - Tipo de estructura paralela del programa

Sobre los resultados de las pruebas realizadas y su contexto

- Caracterización de la máquina paralela en la que se ejecuta el programa. (p.ej. Número de nodos de cómputo, sistema de caché, tipo de memoria, etc.).
  En Linux use la orden: `cat /proc/cpuinfo` para acceder a esta información o `lscpu`.

- ¿Qué significa la palabra `ht` en la salida de la invocación de la orden anterior? ¿Aparece en su ordenador de prácticas?
- Calcular lo siguiente (con gráficas asociadas y su explicación breve):
  - Ganancia en velocidad (*speed-up*) en función del número de unidades de cómputo (*threads* en este caso) y en función de los parámetros que modifiquen el **tamaño del problema** (p.ej. dimensiones de una matriz, número de iteraciones considerado, etc.…).
    - Comente los resultados de forma razonada. ¿Cuál es la ganancia en velocidad máxima teórica?
      Nota: Puede entregar gráficas en 2D o 3D según resulte más ilustrativo.
  - **Responda de forma justificada:** ¿Cuál es la implementación más eficiente de las 2?

Tarea 2: Redacción de una memoria en la que se analice el diseño utilizando OpenMP y std::async y los resultados obtenidos.
Se redactará una memoria en la que se comenten y expliquen diferentes aspectos tanto de la versión secuencial como de la implementación paralela propuesta. También se analizarán los resultados obtenidos probándolas. La memoria deberá dar clara respuesta a las cuestiones planteadas en cada tarea.

Notas generales a la práctica:

- Las respuestas y la documentación generadas de las tareas se integrarán en la memoria conjunta y estructurada. Para el desarrollo y la memoria se utilizará la plataforma de desarrollo GitHub. Esta permitirá una gestión de un **repositorio común de trabajo** que deberá crear cada grupo incluyendo a sus miembros y al profesor. Este dará a conocer su usuario en dicha plataforma para ser añadido.
- La implementación realizada tendrá que poder ejecutarse bajo el sistema operativo Linux del laboratorio de prácticas.
- El código adjuntado utiliza la librería libjpeg y libpng para ejecutar el procesamiento de la imagen. En los PCs de los laboratorios ya viene instalada esta librería, en cambio, para hacerlo funcionar en vuestros PCs, es necesario compilar e instalarlas usando el comando `sudo apt install libpng-dev libjpeg-dev`.
- Para compilar, el código adjuntado incluye un archivo CMakeLists.txt que contiene la información de dependencias de librerías externas y cómo construir los fuentes. Este archivo sirve para construir un Makefile utilizando CMake. Para ello, ejecuta el comando `cmake -S . -B ./build`. Una vez ejecutado el comando, dentro de la carpeta build habrá un fichero Makefile con el que poder ejecutar el comando make para construir el ejecutable. Dentro de la carpeta build, ejecuta el comando make para compilar el ejecutable.
- Para ejecutar el programa hay que usar el comando `./detect <imagen>`, por ejemplo, `./dct ../imagen.png`. El código proporcionado sólo funciona con imágenes cuadradas PNG o JPG.

- La información referente a OpenMP se encuentra en www.openmp.org. Para compilar use el make generado por cmake que ya incluye el flag -fopenmp.
- La práctica se deberá entregar mediante el método que escoja su profesor de prácticas tras 4 sesiones.

*Nota:

Los trabajos teóricos/prácticos realizados han de ser originales. La detección de copia o plagio supondrá la calificación de “0” en la prueba correspondiente. Se informará la dirección de Departamento y de la EPS sobre esta incidencia. La reiteración en la conducta en esta u otra asignatura conllevará la notificación al vicerrectorado correspondiente de las faltas cometidas para que estudien el caso y sancionen según la legislación.