ENES Juriquilla

Licenciatura en Tecnología

Proyecto 1er parcial: Open CV

Cómputo Paralelo

ALUMNOS:

Fabián Poisot

Mario Yahir García Hernández

Victor Hugo Mejía Trejo

PROFESOR:

Dr. Ulises Olivares Pinto

Introducción.

En la actualidad, el procesamiento de imágenes y video en tiempo real es un área de gran interés en diversas aplicaciones, desde la visión por computadora hasta la inteligencia artificial y la robótica. Con el crecimiento del hardware multinúcleo y las arquitecturas de procesamiento paralelo, se ha vuelto esencial aprovechar los recursos computacionales de manera eficiente para mejorar el rendimiento de los sistemas que requieren un alto procesamiento de datos.

Una de las herramientas utilizadas en este proyecto es OpenCV (Open Source Computer Vision Library), una biblioteca de código abierto que facilita el uso de cámaras y el procesamiento de imágenes. Aunque su uso en este caso fue principalmente para la captura de video, OpenCV cuenta con una amplia gama de algoritmos optimizados para tareas como detección de objetos, análisis de movimiento y reconstrucción 3D.

Para mejorar la eficiencia del procesamiento de imágenes en tiempo real, se recurrió a la computación paralela, una técnica que permite dividir las tareas en subprocesos que se ejecutan simultáneamente en distintos núcleos del procesador. Para ello, se utilizó OpenMP (Open Multi-Processing), una API que permite paralelizar código de manera sencilla en arquitecturas de memoria compartida. Su principal ventaja es que facilita la paralelización sin modificar drásticamente la estructura del programa original.

El presente proyecto se enfocó en la implementación de un algoritmo de inversión de colores en tiempo real, comparando su ejecución en modo secuencial y paralelo con el fin de medir el impacto del procesamiento paralelo en el rendimiento del sistema. Se realizaron pruebas de ejecución y se calculó el Speed-Up, una métrica utilizada para evaluar la mejora en el tiempo de procesamiento obtenida mediante paralelización.

A lo largo de este documento, se presentan los detalles de la implementación, los resultados obtenidos y las posibles mejoras que podrían optimizar el rendimiento del algoritmo.

Descripción del Proyecto

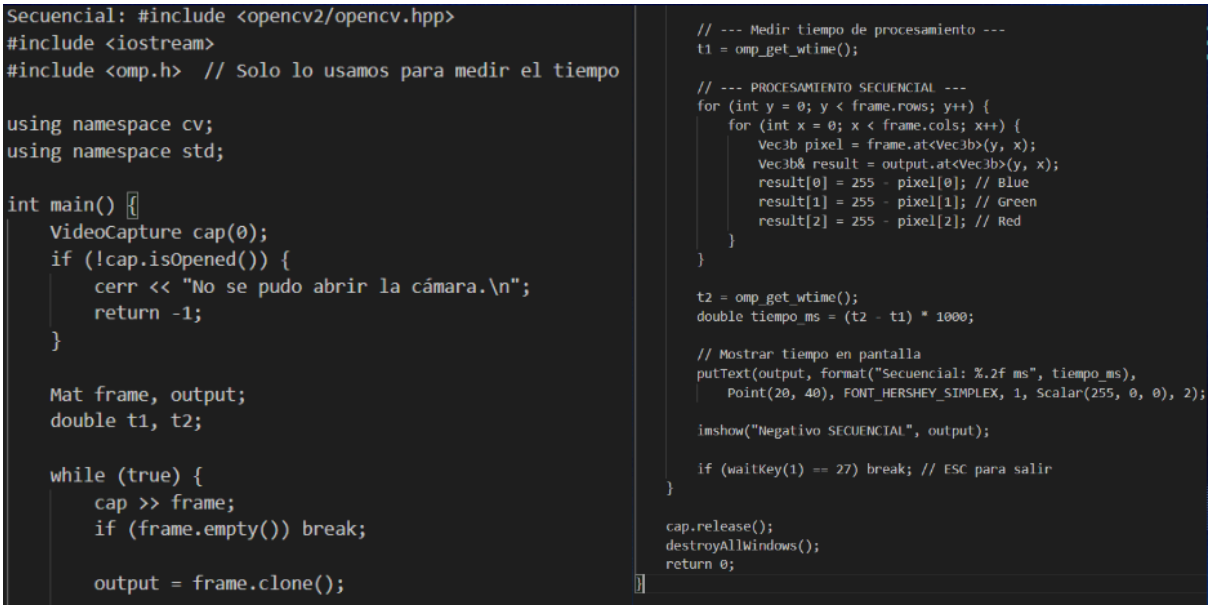
El objetivo del proyecto fue aplicar procesamiento paralelo para mejorar el tiempo de ejecución de un algoritmo de inversión de colores en tiempo real, utilizando la librería OpenCV para la captura de video y OpenMP para la paralelización del procesamiento.

Para ello, se implementaron dos versiones del programa:

1. *Versión Secuencial*: Procesa la imagen de forma tradicional, recorriendo cada píxel y aplicando la inversión de colores uno por uno.
2. *Versión Paralela*: Utiliza OpenMP para distribuir el procesamiento de los píxeles en múltiples hilos, permitiendo la ejecución simultánea en distintos núcleos del procesador.

Ambas implementaciones fueron evaluadas y se compararon los tiempos de ejecución para determinar la eficiencia del procesamiento paralelo.

Código Secuencial



```
Secuencial: #include <opencv2/opencv.hpp>
#include <iostream>
#include <omp.h>  // Solo lo usamos para medir el tiempo

using namespace cv;
using namespace std;

int main() {
    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "No se pudo abrir la cámara.\n";
        return -1;
    }

    Mat frame, output;
    double t1, t2;

    while (true) {
        cap >> frame;
        if (frame.empty()) break;

        output = frame.clone();

        // --- Medir tiempo de procesamiento ---
        t1 = omp_get_wtime();

        // --- PROCESAMIENTO SECUENCIAL ---
        for (int y = 0; y < frame.rows; y++) {
            for (int x = 0; x < frame.cols; x++) {
                Vec3b pixel = frame.at<Vec3b>(y, x);
                Vec3b& result = output.at<Vec3b>(y, x);
                result[0] = 255 - pixel[0]; // Blue
                result[1] = 255 - pixel[1]; // Green
                result[2] = 255 - pixel[2]; // Red
            }
        }

        t2 = omp_get_wtime();
        double tiempo_ms = (t2 - t1) * 1000;

        // Mostrar tiempo en pantalla
        putText(output, format("Secuencial: %.2f ms", tiempo_ms),
            Point(20, 40), FONT_HERSHEY_SIMPLEX, 1, Scalar(255, 0, 0), 2);

        imshow("Negativo SECUENCIAL", output);

        if (waitKey(1) == 27) break; // ESC para salir
    }

    cap.release();
    destroyAllWindows();
    return 0;
}
```

Código Paralelo

Paralelo:

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include <omp.h>

using namespace cv;
using namespace std;

int main() {
    VideoCapture cap(0);
    if (!cap.isOpened()) {
        cerr << "No se pudo abrir la cámara.\n";
        return -1;
    }

    Mat frame, output;
    double t1, t2;

    while (true) {
        cap >> frame;
        if (frame.empty()) break;

        output = frame.clone();

        // --- MEDIR TIEMPO ---
        t1 = omp_get_wtime();

        // --- PROCESAMIENTO PARALELO ---
#pragma omp parallel for collapse(2)
        for (int y = 0; y < frame.rows; y++) {
            for (int x = 0; x < frame.cols; x++) {
                Vec3b pixel = frame.at<Vec3b>(y, x);
                Vec3b& result = output.at<Vec3b>(y, x);
                result[0] = 255 - pixel[0]; // Blue
                result[1] = 255 - pixel[1]; // Green
                result[2] = 255 - pixel[2]; // Red
            }
        }

        t2 = omp_get_wtime();
        double tiempo_ms = (t2 - t1) * 1000;

        putText(output, format("OpenMP: %.2f ms", tiempo_ms),
            Point(20, 40), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 255, 0), 2);

        imshow("Negativo con OpenMP", output);
        if (waitKey(1) == 27) break; // ESC para salir
    }

    cap.release();
    destroyAllWindows();
    return 0;
}
```

Cálculo del Speed-Up

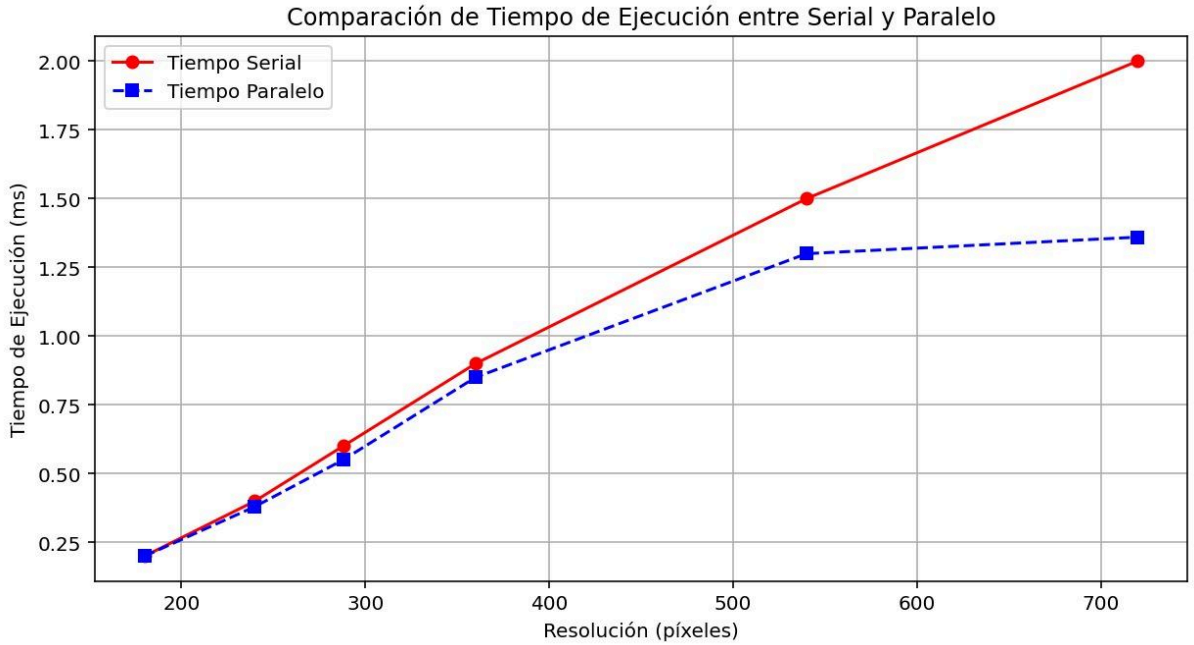
Para evaluar la mejora en el rendimiento del código paralelo en comparación con el secuencial, se calculó el Speed-Up utilizando la siguiente fórmula:

$$S = \frac{Tsecuencial}{Tparalelo}$$

Donde:

- $Tseuencial$ es el tiempo de ejecución del algoritmo secuencial.
- $Tparalelo$ es el tiempo de ejecución del algoritmo paralelo.

Se obtuvo un Speed-Up = 1.47 con 4 hilos y, en casos de movimientos rápidos, este valor llegó hasta 2.0, indicando una mejora significativa en la eficiencia del procesamiento.



Conclusiones

- La paralelización del algoritmo mejoró el rendimiento, aunque la ganancia fue limitada por el tamaño reducido del problema.
- En escenarios con mayor carga computacional, el Speed-Up podría mejorar.
- Se podría probar con imágenes de mayor resolución o algoritmos más complejos para evaluar mejor el impacto de OpenMP en la aceleración del código.