miércoles, 30 de octubre de 2024

18:15

TRABAJO ARQUITECTURA DE COMPUTADORES

Índice

[1. Introducción 3](#_Toc183801920)

[2. El algoritmo de procesamiento de imágenes 4](#_Toc183801921)

[2.1. Detalle del algoritmo. 4](#_Toc183801922)

[2.2. Capturas de imágenes 6](#_Toc183801923)

[2.2.1. Imágenes originales y capturas tras aplicar filtros 6](#_Toc183801924)

[2.3. Comparaciones utilizando DiffImages 8](#_Toc183801925)

[2.4. Medidas de rendimiento. 9](#_Toc183801926)

[2.4.1. Tiempos de respuesta y desviación típica 9](#_Toc183801927)

[2.4.2. Cálculo de aceleración 9](#_Toc183801928)

[3. Análisis de resultados 10](#_Toc183801929)

[4. Reparto del trabajo del grupo 10](#_Toc183801930)

[5. Conclusión 10](#_Toc183801931)

# 1. Introducción

En este trabajo trataremos las mejoras de rendimiento que pueden lograrse al implementar algoritmos para el procesamiento de imágenes digitales utilizando técnicas avanzadas de programación, como la paralelización mediante hilos múltiples y el uso de instrucciones SIMD. Estas técnicas permiten aprovechar al máximo las capacidades de hardware, mejorando el tiempo de ejecución y aumentando la eficiencia de las operaciones.

El trabajo se va a dividir en tres fases, comenzando con la implementación de un programa monohilo básico que actuará como referencia. A partir de esta base, se desarrollan versiones optimizadas que emplean extensiones SIMD y multihilos para procesar imágenes de manera más eficiente. Para garantizar la precisión de los resultados y comprobar si nuestro algoritmo funciona correctamente en todas las versiones, comparamos las imágenes generadas por las distintas versiones utilizando la herramienta **diffImages**, que evalúa diferencias visuales de las imágenes generadas.

El desarrollo de la versión SIMD implica el análisis del soporte de extensiones multimedia en el procesador y la utilización de funciones intrínsecas para realizar operaciones vectoriales. Por su parte, la versión multihilo emplea la concurrencia para dividir el procesamiento entre varios hilos, ajustando el número de estos al grado de paralelismo soportado por el sistema.

Además de implementar las versiones mejoradas, este trabajo evalúa el rendimiento de cada implementación midiendo tiempos de ejecución, desviación estándar y aceleración lograda respecto a la versión básica. Este análisis proporciona una visión de como estas nuevas versiones mejoradas actúan sobre la versión del programa monohilo en el procesamiento de imágenes y nos da a entender la importancia de las técnicas de programación paralela.

# 2. El algoritmo de procesamiento de imágenes

## 2.1. Detalle del algoritmo.

El algoritmo implementado realiza un **filtro de imágenes** basado en una **mezcla de componentes de color** (rojo, verde y azul) de dos imágenes, denominadas aquí como X e Y, para generar una imagen resultante. El proceso utiliza una fórmula matemática específica para calcular cada componente de color (R, G, B) de los píxeles resultantes.

*Siendo I, la imagen destino, X la imagen origen, e Y la segunda imagen con la que se aplicará el filtro.*

Este algoritmo se conoce como blend: whiten mode, debido a que mezcla dos imágenes y aumenta su luminosidad.

* Monohilo

Para el programa monohilo, el algoritmo simplemente recorre píxel por píxel y aplica la fórmula matemática antes expuesta.

* Multihilo

Para el programa multihilo, utilizaremos el mismo algoritmo que en el programa monohilo con la diferencia de como creamos las imágenes.

El programa aplica un filtro a la imagen de entrada usando una imagen de gradiente para crear una imagen de salida procesada. Utiliza múltiples hilos (especificados como NUM\_THREADS) para dividir el procesamiento de la imagen en partes más pequeñas y realizarlo en paralelo, así reducir el tiempo de procesamiento.

|  |
| --- |
| Captura de pantalla de computadora  Descripción generada automáticamente  Esta imagen muestra la configuración de la máquina virtual, como se ve, está utilizando 12 CPUs. |

* **SIMD** (**S**ingle **I**nstruction **M**ultiple **D**ata)

Para empezar a desarrollar el programa que emplea SIMD o instrucciones intrínsecas, lo primero que hicimos fue identificar las instrucciones SIMD soportadas por el ordenador, esta información se encuentra en el fichero proc/cpuinfo.

|  |
| --- |
| Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico  Descripción generada automáticamente |
| Captura del fichero cpuinfo, podemos observar en el apartado flags que esta cpu soporta SIMD de la familia AVX2, pero no soporta AVX512. |

Entonces utilizamos instrucciones SIMD de AVX Family. Lo primero que haremos será definir ITEMS\_PER\_PACKET, que nos indica cuantos datos, podría guardar un \_m256, como los datos de los pixeles son float ocupan 32 bits, y 256/32=8. Por lo tanto, cada paquete podrá manejar 8 datos. A continuación, reservamos memoria para la imagen de destino, con la instrucción \_mm\_malloc(), que al final del programa será liberada con la orden \_mm\_free(). Para el filtro, hemos creado una función llamada filter\_simd() que calcula el número de paquetes que tiene la imagen dividiendo el número de pixeles entre el número de ítems por paquete. Estos paquetes serán procesados con órdenes SIMD, con la misma fórmula que para el programa monohilo; al final se almacenan los paquetes con la orden \_mm256\_storeu\_ps, que almacena los paquetes (de floats (\_ps en instrucción)) en una memoria no alineada, que nos soluciona problemas que podrían darse si la memoria no estuviera alineada (como en el caso que el número de píxeles sea impar).  
El resto de los píxeles serán procesados de manera secuencial, con la misma fórmula que empleamos para el programa monohilo.

Partimos de dos imágenes, que son las mostradas a la izquierda de la flecha, y mediante el algoritmo que se encuentra encima de ellas, que será programado, generaremos la imagen resultante, a la derecha de la flecha.

## 2.2. Capturas de imágenes

### 2.2.1. Imágenes originales y capturas tras aplicar filtros

**-**Imágenes originales:

|  |  |
| --- | --- |
| Imagen que contiene exterior, edificio, camino, calle  Descripción generada automáticamente | Patrón de fondo  Descripción generada automáticamente |

-Resultados:

|  |  |
| --- | --- |
| Programa secuencial | Programa multihilo |
| Una estatua de una persona  Descripción generada automáticamente con confianza media |  |
| Programa SIMD | A primera vista parecen idénticas estas imágenes pero lo comprobaremos utilizando el programa diffImages. |
|  |

## 2.3. Comparaciones utilizando DiffImages

1.Comparación SIMD vs secuencial





Como podemos apreciar, no hay ningún tipo de diferencia entre la imagen creada por el programa multihilo y la imagen creada por el programa secuencial.

2.3.1. Comparación multihilo vs secuencial

Imagen que contiene foto, calle, frente, fuego

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Como podemos apreciar, no hay ningún tipo de diferencia entre la imagen creada por el programa SIMD y la imagen creada por el programa secuencial.

# 2.4. Medidas de rendimiento.

## 2.4.1. Tiempos de respuesta y desviación típica

Los tiempos medidos en los diferentes programas han sido:



## 2.4.2. Cálculo de aceleración

El programa multihilo tiene una aceleración de 4,131 respecto al programa monohilo, está bastante alejado de la aceleración teórica de 12 al emplear 12 CPU. El programa SIMD posee una aceleración de 2,365 respecto al programa monohilo, y por lo tanto resulta más lento que el programa multihilo según los tiempos medidos.

# 3. Análisis de resultados

Como podemos observar en el programa monohilo los tiempos son relativamente superiores respecto a los medidos en los programas multihilo y SIMD.

En el programa multihilo, tras ajustar el número de procesadores y número de hilos adecuadamente, se produce una mejora considerable respecto al programa monohilo.

Por otro lado, el programa SIMD también es considerablemente más rápido que el programa monohilo ya que el uso de funciones intrínsecas, que permiten la ejecución de operaciones específicas de manera óptima, entre otras implementaciones, hacen que se produzca una mejora.

# 4. Reparto del trabajo del grupo

# 5. Conclusión

En este trabajo se ha explorado y analizado el impacto de las técnicas de programación utilizando paralelismo y optimización mediante SIMD en el procesamiento de imágenes. A partir de un programa monohilo básico, se han implementado versiones optimizadas que aprovechan mejor los recursos de hardware disponibles.

Los resultados demuestran que tanto el uso de múltiples hilos como las instrucciones SIMD contribuyen significativamente a reducir los tiempos de ejecución del procesamiento de imágenes. El programa multihilo mostró una mejora notable gracias a la distribución del trabajo entre 12 núcleos.

Por otra parte, el programa SIMD redujo también considerablemente los tiempos de ejecuciones, lo que evidencia la eficiencia de esta técnica.

El análisis de las imágenes resultantes y las comparaciones realizadas con herramientas como diffImages confirmaron la precisión de los algoritmos en sus distintas implementaciones, asegurando la consistencia en los resultados obtenidos.

En conclusión, este proyecto subraya la importancia de las técnicas de programación paralela y optimización avanzada en la mejora del rendimiento. Estas prácticas no solo reducen los tiempos de ejecución, sino que también aprovechan al máximo las capacidades del hardware moderno, destacando su relevancia en aplicaciones prácticas como el procesamiento de imágenes.