

Práctica 1 - Computación Paralela

Juan Sebastián Pachón Carvajal

Departamento de Sistemas e Industrial

Universidad Nacional de Colombia

Bogotá D.C., Colombia

jupachonc@unal.edu.co

Abstract— This paper describes the development of the first practice of the subject of parallel and distributed computing, which consists of making an algorithm capable of processing video to identify faces and blur them, in the first version the program is executed sequentially using a single processor (thread) for execution. Video processing is a very demanding task for a computer, so the use of parallel and distributed computing will be of great help to improve the speed and efficiency of execution, this is a first version of the parallelized algorithm using OpenMP.

Keywords— Video, Filtro, Distorsión, Convolución, Paralelización, Rostros, OpenMP.

I. INTRODUCCIÓN

La computación paralela es una técnica que consiste en la realización de varias tareas en paralelo, utilizando varios recursos de procesamiento, para aumentar la velocidad de ejecución. La técnica se basa en el principio según el cual, algunas tareas se pueden dividir en partes más pequeñas que pueden ser resueltas simultáneamente. Por ello es interesante emplear la computación paralela en diversos ámbitos que requieren una gran cantidad de cómputo y que resultan altamente demandante para su ejecución secuencial, como por ejemplo, el procesamiento de video, dado que se puede entender como el procesamiento de imágenes en masa que ya de por sí es una tarea demandante; paralelizar este tipo de algoritmos nos lleva a poseer la gran ventaja de reducir el tiempo de procesamiento de los videos al realizar varias tareas a la vez.

En el presente informe se explora el procesamiento de videos a través de la detección de rostros en cada frame del video para posteriormente ser distorsionados, esta tarea se divide en dos grandes procesos a llevar a cabo, en primer lugar se necesita identificar los rostros en el video para esta tarea se utilizará el método de detección de objetos mediante de clasificadores de cascada basado en HAAR, propuesto por Paul Viola y Michael Jones, se trata de un enfoque basado en el aprendizaje automático en el que se entrena una función en cascada a partir de un montón de imágenes positivas y negativas. A continuación, se utiliza para detectar objetos en otras imágenes.

Una vez identificados los rostros, la segunda tarea es la distorsión de los rostros, para lo cual se realiza una adaptación de la técnica de procesado de imágenes basado en convolución, para la aplicación de filtros a imágenes, tarea que se realizará frame por frame. Los filtros se utilizan para mejorar la calidad de la imagen de ráster al eliminar datos falsos o mejorar las identidades de los datos. Estos filtros de convolución se aplican a un kernel móvil o superpuesto (ventana o vecindad), como 3 x 3. Los filtros de convolución actúan calculando el valor de píxel en función de la ponderación de sus vecinos.

El objetivo de esta práctica es utilizar el algoritmo descrito durante la primera práctica para definir una versión paralelizable y finalmente realizar su implementación usando la librería para paralelización *OpenMP*.

II. DESARROLLO

El programa se realiza en C++, haciendo uso principalmente de la librería de procesamiento de imágenes *OpenCV* y los elementos estándar del lenguaje. El funcionamiento se basa en dos parámetros el primero el video a procesar y el segundo el nombre del archivo donde se guardará el video procesado.

A. Lectura del video

La primera tarea a realizar es la lectura del video de entrada para poder ser tratado dentro del programa, esto se logra de manera sencilla empleando la clase *VideoCapture* de *OpenCV* la cuál permite leer de manera secuencial el video cuadro a cuadro, guardando la información de cada uno en los contenedores de imágenes *Mat* que facilitaran su procesamiento con las demás herramientas de la librería.

B. Detección de rostros

Para la detección de rostros se hace uso de la librería *OpenCV* usando los modelos de detección de objetos HAAR para detectar rostros humanos en los videos, para realizar esta tarea se hace un análisis de cada frame, para lo cual se convierte cada imagen a escala de grises y se ecualizan los histogramas para finalmente utilizar el clasificador de cascada, definir las coincidencias y se almacenan los rostros encontrados en un vector de *Rect* donde se almacenan las

coordenadas dentro del frame donde se encuentran los rostros para su posterior tratamiento en el proceso de distorsión, para lo cual se hará un llamado a la función por cada elemento identificado en el vector faces.

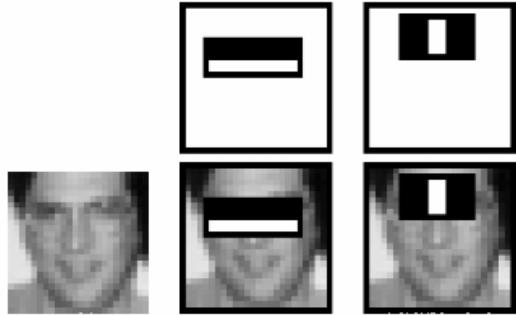


Figura 1. Funcionamiento de los modelos de cascada

C. Distorsión de rostros

En el proceso de distorsión de rostro se reciben en la función dos elementos el frame principal al cuál se le realizará la transformación y el rectángulo que define la posición de la cara dentro del mismo, a partir de esto se empezará a iterar tomando matrices de píxeles a las cuales se les hará una transformación de tomandolas según el criterio dado. El proceso a aplicar guarda sus similitudes con la convolución, sin embargo, dado que no se busca aplicar un filtro de alta resolución a la imagen este no sigue las mismas reglas.

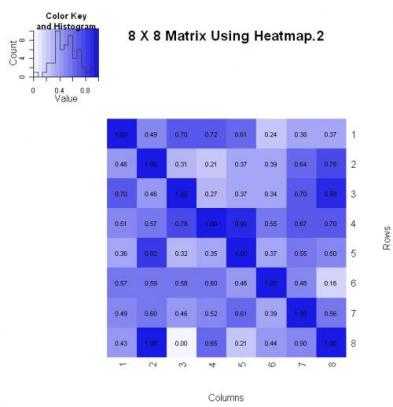


Figura 2. Matriz de píxeles

Para cada matriz de píxeles dada se guardarán sus posiciones dentro del frame en un arreglo y posterior a esto se realizará una separación de estos por cada canal de color bajo el esquema BGR usado en OpenCV.

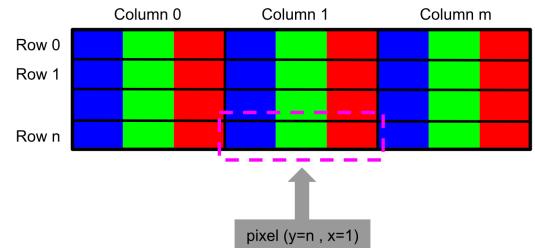


Figura 3. Pixel de la imagen

Con el fin de llevar a cabo la separación por canales se recorre el arreglo de posiciones recuperando pixel a pixel sus valores en cada canal, lo cual es fundamental para la transformación a realizar que se basa en encontrar el promedio de los valores de color por cada canal y asignarlo a cada uno de los elementos que conforman la matriz de píxeles, logrando el proceso de distorsión

D. Paralelización

El primer paso para la paralelización del algoritmo es identificar qué partes son las de mayor coste computacional, de esta manera se identificó que la detección de rostros y la posterior distorsión son las que más coste tienen. Luego se analizaron estas dos secciones, para la primera, la detección de rostros, dado que se utiliza métodos internos de OpenCV, implementar una paralelización con OpenMP en el código fuente no es alternativa, por lo que se utilizó la función propia de OpenCV para aplicar el número de hilos con los cuales se realizarán las tareas `setNumThreads(numThreads)` con lo cual se asegura que se entrega una distribución de trabajo uniforme a los hilos en toda la ejecución del programa.

Para la segunda sección, el algoritmo de distorsión si se puede implementar una paralelización directa usando OpenMP, por lo que se analiza cuál es la mejor división para esto, en primera instancia se pensó dividir esta tarea por fragmentos de videos de tal forma que cada hilo se encargara de ciertos frames, sin embargo, esto no resulta de buena manera dado que no en todos los frames se van a encontrar rostros y conlleva a que no se haga una buena repartición de la carga en los hilos. Por lo que se llegó a la idea que la mejor forma de repartir la carga entre núcleos es garantizando que todos realicen la misma tarea, por ello, la paralelización se lleva a cabo en la parte del algoritmo de distorsión que recorre el rostro ya identificado y aplica el filtro, haciendo que todos los hilos trabajen para optimizar el tiempo en el que se aplica el filtro a la imagen, y esta tarea se repite con todas las incidencias de rostros; Para lograr esto se dividió el ciclo que recorre las columnas de la imagen de tal forma que se asigne un rango de estas a cada hilo para que posteriormente las recorra el ciclo interno de las filas aplicando el filtro a cada matriz definida, con esto al final de recorrer todas las columnas tenemos el filtro aplicado en el sector

correspondiente donde se ubica un rostro, haciendo uso eficiente de los hilos.

III. PRUEBAS

Para las pruebas se utilizaron cuatro videos de distinta duración donde se pueden observar rostros de personas bajo diversas características físicas.

En el primer video se encuentra un hombre mirando directamente a la cámara mientras esta se mueve lentamente. Cuenta con una duración de 9 segundos y una resolución de 720p.



Figura 4. Video 1

En el segundo video tenemos dos personas con movimientos más notorios, cambiando sus expresiones faciales y con luz directa del sol. Cuenta con una duración de 6 segundos y una resolución de 720p.

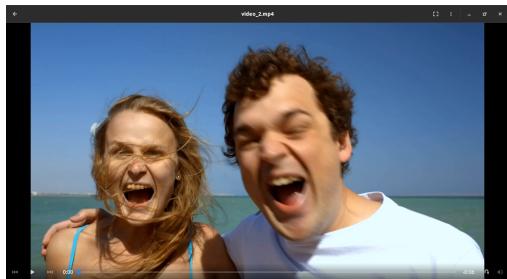


Figura 5. Video 2

En el tercer video se encuentra una escena mucho más complicada, dado que se encuentran con presencia de varias personas bajo condiciones de luz poco favorable y un ruido alto en la imagen. Cuenta con una duración de 8 segundos y una resolución de 540p.

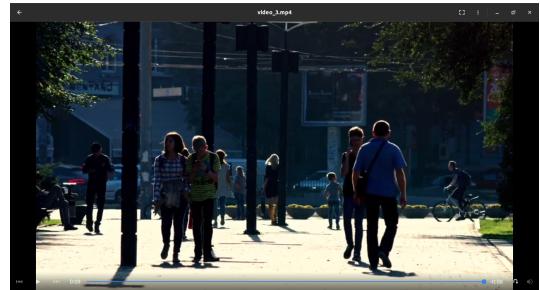


Figura 6. Video 3

Para el cuarto video se observan dos personas con iluminación artificial y con una cámara que juega con el enfoque. Cuenta con una duración de tres segundos y una resolución de 1080p.

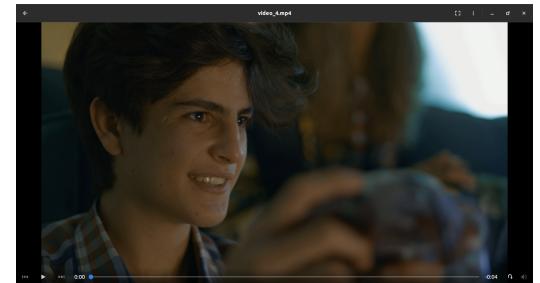


Figura 7. Video 4

IV. RESULTADOS

Se obtuvieron los siguientes resultados al procesar los videos de prueba.

A. Video 1

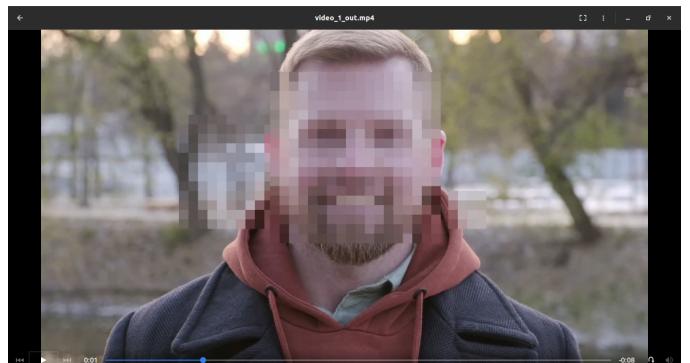


Figura 8. Resultado video 1

Teniendo un resultado de la misma manera que los obtenidos en la práctica uno, presentamos los siguientes datos de la ejecución de forma paralela.

- Resolución. 720p
- Duración. 9 segundos
- Rostros. 1 persona

Número de Hilos	Tiempo
1	242,1
2	172,24
4	133,1
8	120,91
16	103,25

En los tiempos se evidencia como la paralelización reduce los tiempos de ejecución del algoritmo, se observan reducciones considerables desde la paralelización a dos hilos y finalmente con 16 hilos se observa que el tiempo es menos de la mitad del secuencial.

Número de Hilos	SpeedUp
1	1,00
2	1,41
4	1,82
8	2,00
16	2,34

En esta ejecución podemos ver fácilmente los resultados de la paralelización, donde desde el primer momento al usar dos hilos se ve un aumento de 41% en la velocidad de procesamiento.

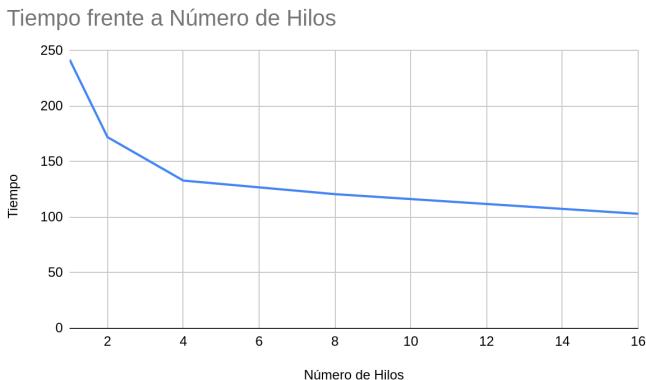


Figura 9. Gráfica ejecución vídeo 1

Al analizar la gráfica se puede observar un comportamiento bastante interesante y es que hay una gran caída en el tiempo de ejecución hasta los 4 hilos que corresponde al número de núcleos que posee la máquina

donde se ejecutó, luego de esto el tiempo sigue bajando de forma considerable pero de manera más lineal.

SpeedUp frente a Número de Hilos

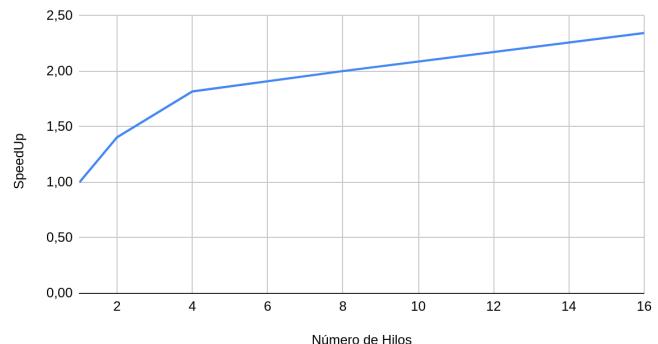


Figura 10. Gráfica speedUp vídeo 1

El comportamiento del speedUp es interesante a nivel de valores, aunque no se encuentra una razón de incremento constante se puede ver que se sitúa finalmente cerca a 2,5

B. Vídeo 2

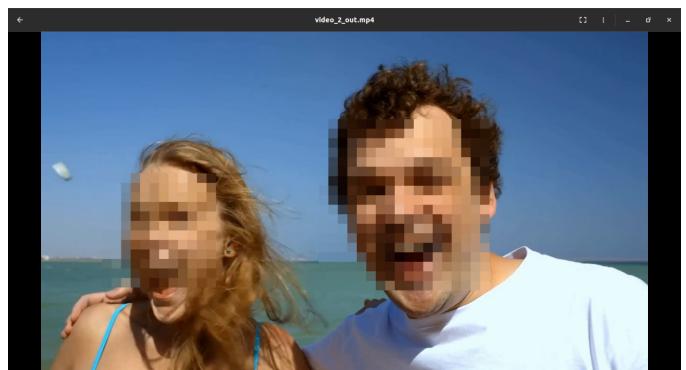


Figura 11. Resultado video 2

Teniendo un resultado de la misma manera que los obtenidos en la práctica uno, presentamos los siguientes datos de la ejecución de forma paralela.

- Resolución. 720p
- Duración. 6 segundos
- Rostros. 2 personas

Número de Hilos	Tiempo
1	122,24
2	94,16
4	77,97

8	74,11
16	68,07

En esta ejecución se puede notar que las caídas en el tiempo de procesamiento son ligeramente menores que los obtenidos en el primer vídeo, se puede ver fácilmente que no se logró correr a menos de la mitad del tiempo de lo secuencial como si se logró en el primer vídeo.

Número de Hilos	SpeedUp
1	1,00
2	1,30
4	1,57
8	1,65
16	1,80

Tal como se corresponde con los tiempos, en el speedUp no logramos un 2,0 sin embargo la subida de velocidad alcanza a un 80% que es una cifra bastante considerable.

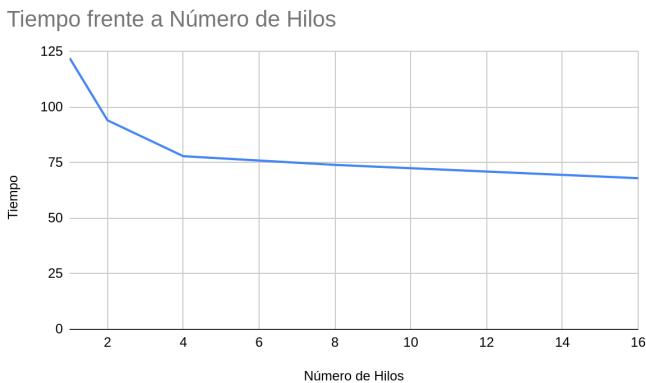


Figura 12. Gráfica ejecución vídeo 2

Como reflejo de los datos vistos en las tablas en la gráfica se describe una reducción a menor escala, sin embargo, se mantiene la tendencia a la baja al aumentar el número de hilos.

SpeedUp frente a Número de Hilos

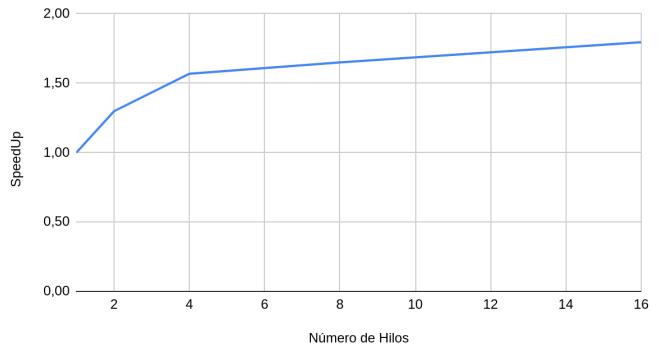


Figura 13. Gráfica speedUp vídeo 2

El speedUp tiene una subida similar sobre los 4 hilos como se observó en el primer vídeo, aunque a menor escala este se queda con una cota superior cercana a 2, lo que representa una reducción de 0,5 frente al obtenido en el primer vídeo.

C. Video 3



Figura 14. Resultado video 3

Teniendo un resultado de la misma manera que los obtenidos en la práctica uno, presentamos los siguientes datos de la ejecución de forma paralela.

- Resolución. 540p
- Duración. 8 segundos
- Rostros. 6 personas

Número de Hilos	Tiempo
1	269,26
2	179,6
4	130,73
8	107,7
16	96,52

Los tiempos de ejecución en este vídeo son bastante interesantes dado los resultados obtenidos en la primera práctica, donde a pesar de ser el que estaba en menor resolución y por lo cual era menos información a procesar duraba un tiempo prolongado de procesamiento debido a la cantidad de caras presentes, es el que mayor reducción de tiempo ha presentado dado que la ejecución en 16 hilos es casi $\frac{1}{3}$ de la secuencial.

Número de Hilos	SpeedUp
1	1,00
2	1,50
4	2,06
8	2,50
16	2,79

Como era de esperarse dados los resultados de tiempo es donde presentamos el speedUp más alto, siendo casi de 3 en la ejecución a 16 hilos.

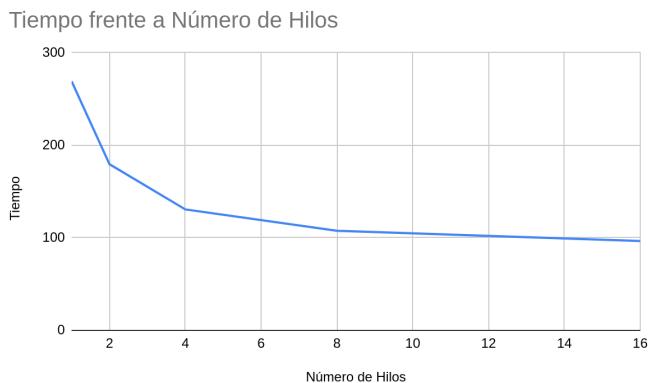


Figura 15. Gráfica ejecución vídeo 3

En esta gráfica se puede ver que guarda similitudes con las ejecuciones anteriores, pero al llegar al número de hilos del procesador (8) el tiempo de ejecución tiende a estabilizarse y no a tener una reducción lineal como se presentó en los vídeos anteriores.

SpeedUp frente a Número de Hilos

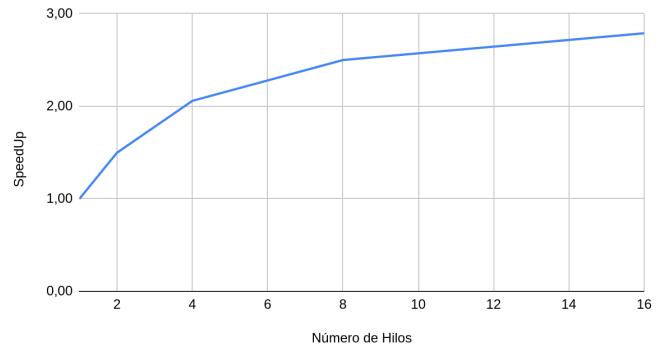


Figura 16. Gráfica speedUp vídeo 3

El speedUp corresponde con los analizado anteriormente y aunque es mayor que el alcanzado en cualquiera de las otras ejecuciones, vemos como tiende a estabilizarse cerca al 3.

D. Video 4

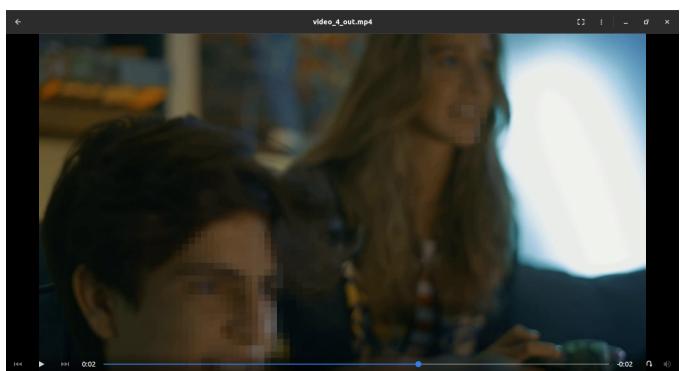


Figura 17. Resultado video 4

Teniendo un resultado de la misma manera que los obtenidos en la práctica uno, presentamos los siguientes datos de la ejecución de forma paralela.

- Resolución. 1080p
- Duración. 3 segundos
- Rostros. 2 personas

Número de Hilos	Tiempo
1	128,5
2	90,66
4	71,99
8	62,97
16	58,71

En esta ejecución, la de mayor resolución, se observa un resultado similar al de la primera, dado que a los 8 hilos ya se logró ejecutar el algoritmo en la mitad del tiempo de lo que toma secuencial.

Número de Hilos	SpeedUp
1	1,00
2	1,42
4	1,78
8	2,04
16	2,19

El speedUp se comporta inicialmente con un crecimiento mayor que el del primer vídeo, aunque siguiendo un patrón de crecimiento similar, ya finalmente vemos que en el speedUp final a 16 hilos este es inferior, por lo que podemos ver que se empieza a acercar a la cota de aumento de velocidad.

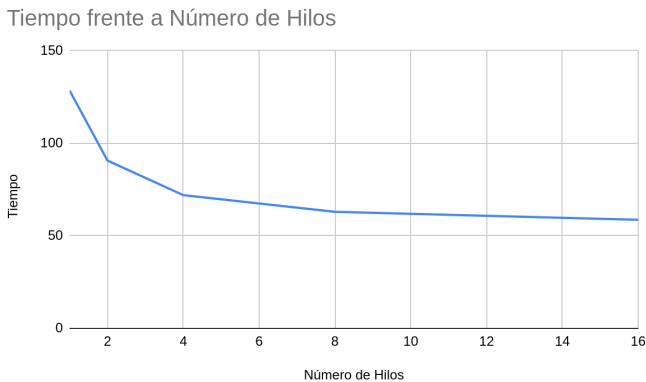


Figura 18. Gráfica ejecución vídeo 4

Al observar la gráfica se evidencia que hay una caída importante en la ejecución hasta los 4 hilos, luego un poco más reducido a los 8 donde ya empieza a estabilizarse cerca de los 50 segundos.

SpeedUp frente a Número de Hilos

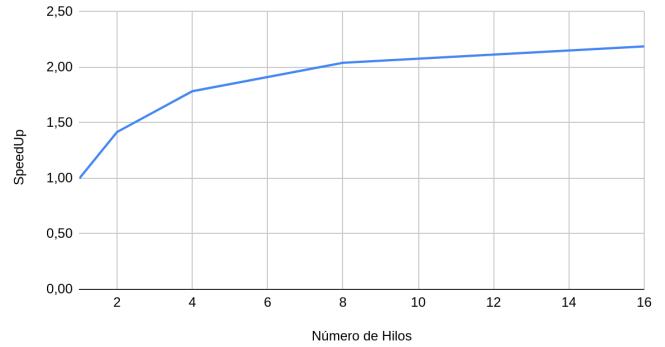


Figura 19. Gráfica speedUp vídeo 4

En el speedUp se puede ver la subida de forma rápida hasta 2,0 y luego se empieza a acotar cerca de 2,20 aunque con una pendiente considerable aún.

V. CONCLUSIONES

1. El algoritmo de detección de rostros funciona bien con condiciones de buena iluminación y bajo ruido, pudiendo identificar varios rostros a la vez en una imagen dada.
2. El algoritmo de distorsión funciona de manera óptima sobre las imágenes, sin embargo, es necesario ajustar el tamaño de la matriz de distorsión dada la resolución del vídeo para dar resultados acordes.
3. El factor más determinante a la hora del procesamiento de la imagen es el número de rostros que aparecen en la misma, dado que ha mayor número se eleva el tiempo de ejecución.
4. La resolución del vídeo afecta positivamente el tiempo de ejecución, ya que se hace el reconocimiento de rostros más rápido y por lo tanto compensa la mayor cantidad de información a procesar.
5. Como se suponía la duración de los vídeos repercute directamente el tiempo de procesamiento, aunque no es tan determinante en este.
6. El procesamiento de los vídeos se ve afectado por varios factores dada la naturaleza propia de la imagen como la cantidad de rostros y la nitidez de la imagen, y de la cantidad de datos a procesar como la resolución y la duración del vídeo.
7. El comportamiento de la reducción de tiempo de procesamiento de los vídeos no se puede tratar de forma lineal, dado que hay tareas que no se pueden parallelizar, además que hay ciertos escenarios de los datos de entrada que condicionan la velocidad de

- procesamiento más allá de la capacidad de cómputo de los mismos.
- 8. El speedUp es una medida interesante de la paralelización del algoritmo, cuantifica el nivel de utilidad de la paralelización.
 - 9. El speedUp varía de forma considerable en los vídeos procesados, por lo que se puede concluir que el crecimiento de este indicador depende más de la complejidad de los datos de entrada que de el número de hilos bajo los cuales se ejecute un algoritmo.
 - 10. La eficiencia de la paralelización se empieza a ver reducida cuando se alcanzan los límites físicos de las máquinas donde se corren los programas, por lo que conocer la información del número de hilos de cada máquina es primordial para determinar el número de hilos sobre los cuales se obtienen los mejores resultados en equilibrio.

REFERENCES

- [1] «La computación paralela: alta capacidad de procesamiento», Teldat Blog - Connectando el mundo, 14 de julio de 2020. <https://www.teldat.com/blog/es/computacion-paralela-capacidad-de-procesamiento/> (accedido 29 de septiembre de 2022).
- [2] «OpenCV: Cascade Classifier». https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (accedido 29 de septiembre de 2022).
- [3] «Función de convolución—ArcMap | Documentación». <https://desktop.arcgis.com/es/arcmap/latest/manage-data/raster-and-images/convolution-function.htm#:~:text=La%20funci%C3%B3n%20de%20convoluci%C3%B3n%20realiza,realces%20basados%20en%20el%20kernel.> (accedido 29 de septiembre de 2022).
- [4] «OpenCV C++ Program for Face Detection», GeeksforGeeks, 17 de junio de 2017. <https://www.geeksforgeeks.org/opencv-c-program-face-detection/> (accedido 29 de septiembre de 2022).