Arquitectura de Computadores

Proyecto

Grupo Reducido: 80, 81
Grupo de Trabajo: 14

Alberto Penas Díaz
100471939

Héctor Álvarez Marcos
100495794

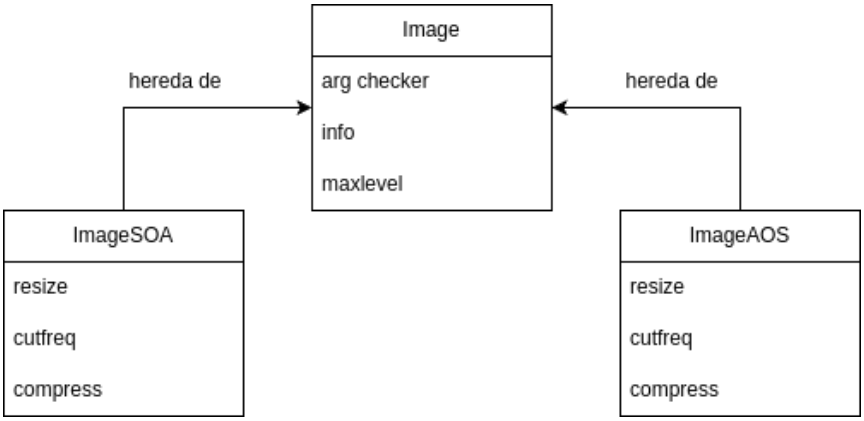
Jorge Agramunt Muro
100495764

Pablo Navarro Hurtado
100495879

Índice

Diseño

Para comenzar el desarrollo de nuestro proyecto, hemos analizado qué funciones son comunes a ambas estrategias (AOS y SOA) para **encapsular** las mismas en clases que acojan conceptualmente a las imágenes. De modo general, el diseño de nuestro programa puede ser resumido en el siguiente diagrama de clases:



Siguiendo el modelo que se nos ofrecía en el enunciado, todas las funcionalidades que comparten ambas estrategias están agrupadas en la librería "*common*". En esta librería, además de la superclase "*Image*", están presentes los siguientes ficheros:

- **binario**: Agrupación de todas las funciones de entrada y salida binaria, así como funciones específicas de manejo de bits.
- **struct-rgb**: Definición de las estructuras específicas usadas para almacenar píxeles en memoria, teniendo en cuenta las estrategias SOA y AOS

De forma general, hemos diseñado todas y cada una de las funciones para que acojan todos los posibles casos de uso posibles en la aplicación. De esta manera, nuestro diseño de cada función separa, en un primer condicional, el formato de entrada y salida de las imágenes para poder dirigirnos a la función específica que gestione la funcionalidad pedida. Esta metodología genera más líneas de código pero organiza el mismo en bloques muy identificativos, lo que ha sido de gran ayuda en el desarrollo del proyecto para navegar en el mismo.

Optimización

Complejidad final **teórica** obtenida:

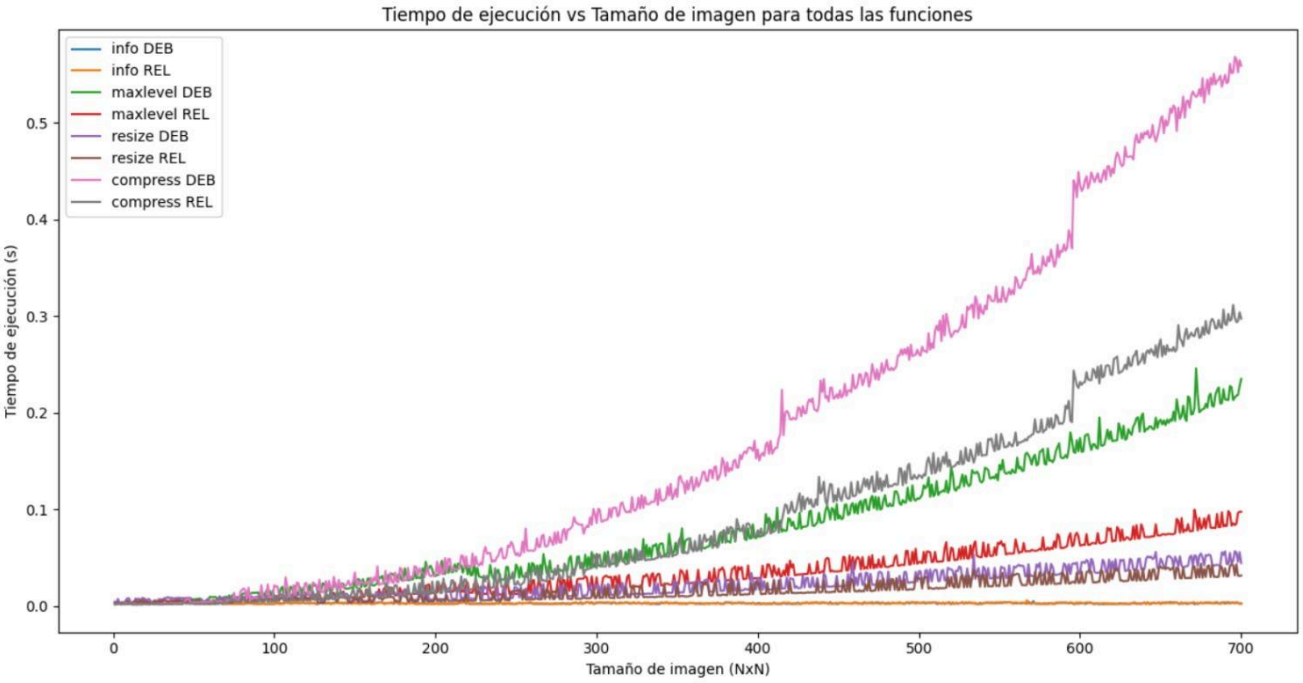
Complejidad	Info	Maxlevel	Resize	Cutfreq	Compress
Image AOS	$O(1)$	$O(n)$	$O(2n)$	-	$O(2n)$

Image SOA	$O(1)$	$O(n)$	$O(2n)$	-	$O(2n)$

No hemos puesto complejidad a la función Cutfreq debido al gran número de parámetros del que depende. Como optimizaciones comunes a todas las funciones, hemos optado por agregar unos flags de compilación específicos para aumentar al máximo el rendimiento de nuestro programa:

- **`-march=native`**: Esta opción le indica al compilador que optimice el código para la arquitectura específica de la CPU en la que se está compilando el código.
- **`-mtune=native`**: Este flag ajusta el rendimiento del código generado para la CPU específica en la que se está compilando. Mientras que `-march` define las instrucciones disponibles, `-mtune` ajusta los detalles de rendimiento (como la programación de instrucciones) para esa CPU en particular, tratando de optimizar su velocidad.
- **`-O3`**: Esta opción activa el nivel de optimización más alto. `-O3` habilita todas las optimizaciones de `-O2`, más algunas adicionales como la desenrollación de bucles y el uso de SIMD (instrucciones de múltiples datos) en bucles.
- **`-funroll-loops`**: Desenrolla bucles, es decir, transforma los bucles en una secuencia de instrucciones repetidas en lugar de mantener la estructura del bucle original.
- **`-ffast-math`**: Permite ciertas optimizaciones matemáticas que no garantizan un cumplimiento estricto de las reglas IEEE o ISO C en las operaciones de punto flotante. Por ejemplo, puede permitir el reordenamiento de operaciones, suposiciones de asociatividad y otras optimizaciones que pueden mejorar la velocidad de las operaciones matemáticas.
- **`-fstrict-aliasing`**: Indica al compilador que puede asumir que los punteros de distintos tipos no apuntan a la misma dirección en memoria. Esta suposición permite optimizaciones en el acceso a memoria.

Para verificar la utilidad de estos flags de compilación, hemos medido el tiempo de ejecución con respecto a la versión sin flags (debug) en nuestro equipo local:



A continuación, se especifican detalladamente las optimizaciones por las que se ha optado en esta función.

Maxlevel

Esta función (junto con "*info*", pero la omitimos debido a su trivialidad) es la única que NO hace uso de AOS o SOA, ya que no existe ninguna necesidad de almacenar la imagen en memoria. En esta función, nos recorremos la imagen píxel a píxel, operamos sobre cada píxel recogido en cada iteración y finalmente lo escribimos en la imagen de salida.

Resize

En esta función, la imagen de entrada es almacenada en memoria siguiendo las estrategias SOA o AOS. Dependiendo del formato[1] de la imagen de entrada, se tiene en cuenta el tipo de dato óptimo para almacenar cada canal de color, usando *unsigned char* o *unsigned short* según sea necesario. Esta distinción la hacemos para ahorrar memoria en caso de que la imagen sea con formato de 3 Bytes por píxel, ya que sobraría espacio utilizaramos únicamente un tipo de dato capaz de almacenar 16 bits para representar canales de color que únicamente necesitan 8 bits.

[1] El formato de la imagen puede ser de 3 Bytes por píxel o 6 Bytes por píxel.

Una vez se ha recorrido la imagen de entrada, nos recorremos la imagen de salida según el nuevo tamaño especificado en los argumentos de entrada al programa. Para cada píxel, hacemos las operaciones convenientes y lo escribimos en la imagen de salida.

Cutfreq

Nuestra implementación más compleja es de la función Cutfreq, pues hemos seguido un desarrollo enrevesado pero óptimo. Como más adelante se podrá observar en el diagrama de flujo, esta función se puede dividir en dos partes importantes:

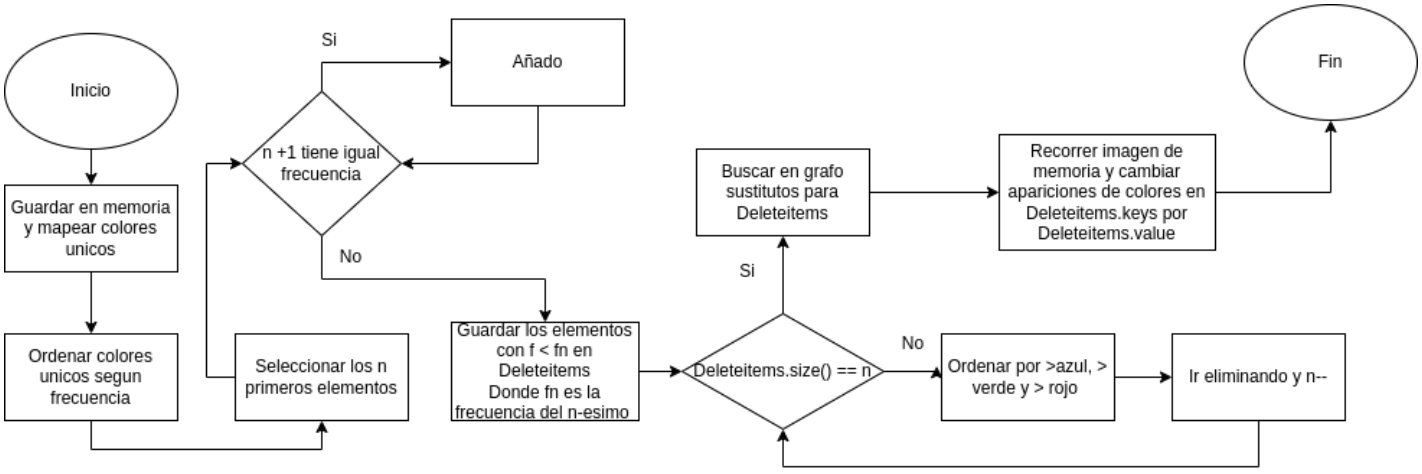
1. Obtención de colores a eliminar (selección de píxeles únicos y análisis de frecuencia).
2. Sustitución de colores a eliminar.

Para la obtención de colores a eliminar, por ejemplo si queremos eliminar los 3 colores menos frecuentes y tenemos 2 colores con frecuencia 1, son enviados directamente a eliminación, ya que no hace falta comprobar nada más, si tenemos dos colores con frecuencia 2, no sabemos cuál vamos a eliminar, ya que solo nos falta añadir un color más a eliminación. Por ello, a igualdad de frecuencia en la imagen, para elegir el color que vamos a eliminar de los dos….

Una vez hemos obtenido los colores finales a eliminar, falta ver por qué colores van a ser sustituidos. Para ello, a nivel conceptual, nuestra implementación representa los colores que se **no se van a eliminar** en un espacio tridimensional cuyos ejes son los canales de color r, g, b. Para buscar el sustituto de cada color que se va a eliminar, bastaría con recorrerse cada punto de este espacio hasta encontrar el punto con menor distancia euclídea. Sin embargo, esto puede ser MUY ineficiente, ya que hace falta recorrerse toda la nube de puntos para encontrar este sustituto. Por esto, nuestra primera idea consistió en implementar el algoritmo de aprendizaje automático **KNN** (siendo en nuestro caso k=1), que, dado un punto en una nube de puntos, obtiene el primer punto que entre dentro de una "esfera tridimensional" cuyo punto céntrico es el color a eliminar (representado en la imagen de la derecha en 2 dimensiones).

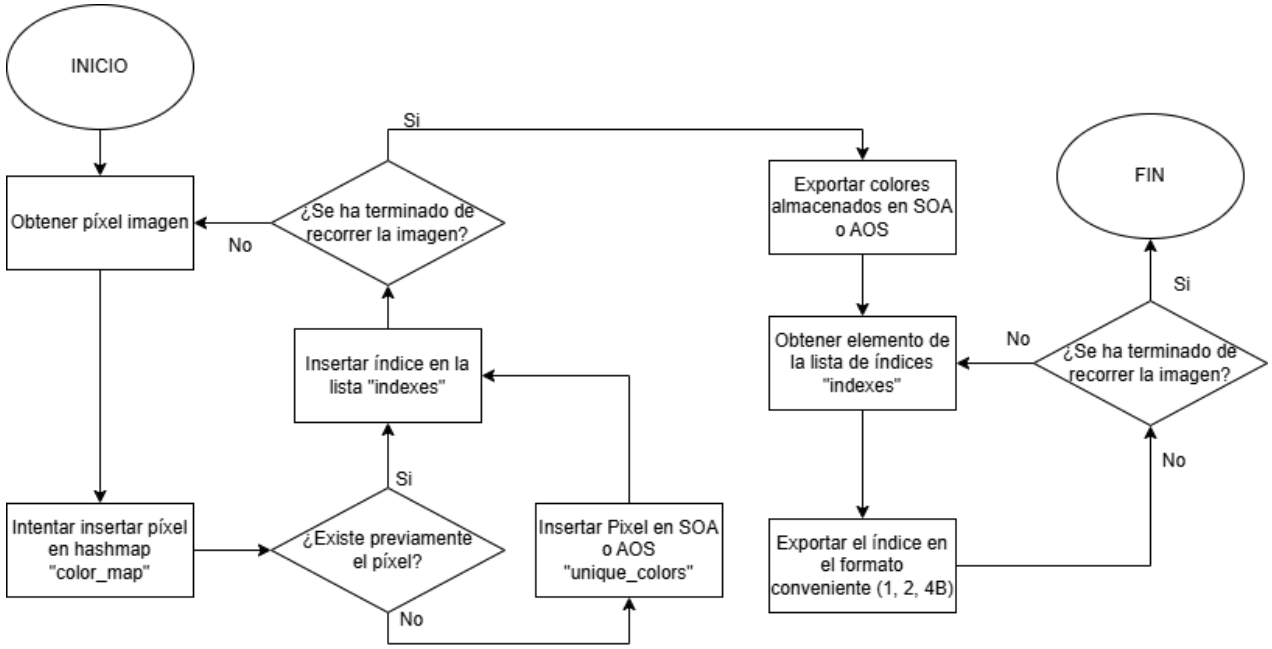
Sin embargo, esto también es muy ineficiente cuando quedan muy pocos colores en la imagen, ya que hay que recorrerse prácticamente todo el espacio. Por ello, la implementación definitiva fue dividir el espacio tridimensional en sub-espacios, representados en nuestro código en forma de grafo. De esta manera, por cada píxel que se pretende eliminar, buscamos ÚNICAMENTE en el subespacio que le corresponde, así como en los inmediatamente adyacentes. De esta forma, nos ahorramos comprobar la distancia euclídea de todos aquellos subespacios que estén alejados del punto que queremos eliminar.

De modo más detallado, adjuntamos el diagrama de flujo correspondiente a la función:



Compress

Para optimizar esta función, hemos tratado de tener mucho cuidado con respecto a las estructuras de datos que empleamos para buscar los píxeles. El diagrama de flujo de esta función es el siguiente:



Tal y como se puede observar, nos recorremos la imagen original 2 veces y en cada iteración del recorrido insertamos o buscamos el píxel almacenado en el hashmap[2]. Es importante mencionar que esta estructura únicamente se usa a **modo de catálogo**, siendo la estructura principal de almacenamiento el SOA o el AOS.

[2] Se ha hecho uso de la estructura nativa de c + + "unordered_map", que posee las propiedades de un hashmap, esto es, una estructura de datos cuyas operaciones de inserción, búsqueda y eliminación tienen complejidad O(1).

Somos plenamente conscientes de que, usando dos estructuras distintas para guardar la imagen y los índices al mismo tiempo, aumentamos significativamente el uso de memoria y por consiguiente, la tasa de accesos a la caché. Sin embargo, haciendo uso de ésta estrategia, disminuye drásticamente el tiempo de ejecución por lo que hemos optado por ella, ya que consideramos que la ganancia en tiempo de ejecución es proporcionalmente mejor (en este caso) al "excesivo" uso de memoria.

Pruebas Realizadas

Hemos creado tres diferentes directorios para la ejecución de los unitests: utest-common, utest-imgsoa y utest-imgaos. En cada uno de ellos hemos creado un fichero distinto para la realización de los distintos unitests dentro de los ficheros de comonn, imgsoa e imgaos respectivamente.

En binario_test creamos la clase BinaryTest para probar las distintas funciones dentro de binario.cpp, read_binary_8, write_binary_8, etc. En cada uno de los tests creamos un fichero temporal en el que escribimos y/o leemos valores en binario dependiendo de la función sobre la que se esté realizando el unitest.

Para cada una de las distintas funciones hemos hecho al menos un caso en el que la función funciona correctamente, un caso en el no funciona y casos extra dependiendo de si salta o no un error.

Con respecto a las pruebas funcionales, hemos usado el lenguaje de programación python para su realización, debido a la facilidad de uso y la versatilidad que ofrece, **hemos unificado** tanto las pruebas funcionales del AOS como del SOA en un único fichero que prueba directamente comandos de terminal que ejecuta nuestro programa y compara la salida de nuestras fotografías directamente con las salidas esperadas proporcionadas en Aula Global, es por ello (por la necesidad de comparar imágenes) *que el directorio "ftest" tiene a su vez dos subdirectorios; "input", donde se encuentran las imágenes de entrada y "expected" donde se encuentran las imágenes esperadas.* ***PARA LA FUNCIONALIDAD DE LOS FTESTS ES IMPORTANTE QUE SE CARGUEN EN LOS DIRECTORIOS LAS SIGUIENTES IMAGENES.*** Para mayor facilidad, hemos subido un archivo a drive con las imágenes en sus localizaciones correctas: https://drive.google.com/file/d/1hZVw0Jp252VRlmlQJfe9BQ8lBcUxt4Ij/view?usp=sharing

Estructura de directorios:

input: *deer-small.pm deer-large.ppm lake-small.ppm lake-large.ppm sabatini.ppm*

expected:

Directorio "**cutfreq**": Revision 2 de salidas para cutfreq(solo los archivos .ppm)

Directorio "**resize**": Revisión 2 de salidas para resize(solo los archivos .ppm)

Directorio "**compress**": Salidas de referencia(solo los archivos .cppm)

Directorio "**maxlevel**": Salidas de referencia + salidas adicionales(solo los archivos .ppm)

EL INDICAR LA ESTRUCTURA DE DIRECTORIOS PARA LOS FTESTS SE HA REALIZADO DEBIDO AL LÍMITE DE 500KB DEL ENTREGADOR.

Para el "*ftest*" del Cutfreq hemos cambiado la interpretación que se ha dado de permisividad de cambios porque consideramos que para esta función el enfoque otorgado como esperado no es el adecuado. Se nos permite una variación de +-5 en cada componente R,G y/o B. Nosotros para esta función hemos considerado 2 tipos de aciertos de colores "Distintos". El primero: Si la distancia euclídea entre el pixel antiguo y el cambiado por la salida esperada es igual a la distancia euclídea entre el pixel antiguo y nuestro cambio no se considera fallo. Ejemplo: Pixel original (10,10,10) Pixel esperado (5,5,5) Pixel nuestro (15,15,15). En este caso la diferencia es mayor de 5 entre el esperado y el nuestro, sin embargo la distancia euclídea es igual.

El segundo caso es, para tener en cuenta esos +-5 en diferencia de color, hemos tenido en cuenta el peor caso. Si la diferencia entre el pixel esperado y el nuestro es de 5 en cada componente, la distancia euclídea entre cambios será de 8.66 == sqrt($5^2 + 5^2 + 5^2$). Por lo que consideramos también acierto si la diferencia entre la distancia euclídea del pixel esperado y de la nuestra es < 8.66. Basamos la superación de la prueba si se consigue un porcentaje de acierto > 99.99 teniendo en cuenta que el % de acierto se mide **no sobre el total de píxeles**, si no sobre el % de píxeles modificados, es decir, los que han sido afectados por el cutfreq.

Evaluación de Rendimiento y Energía[3]

Las conclusiones e interpretaciones de los siguiente apartados están explicados en la sección Conclusiones.

Evaluación en Avignon

Análisis de rendimiento:

maxlevel lake-large.ppm 65535	SOA	AOS
Tiempo de ejecución (segundos)	9,724	8,678
Instrucciones ejecutadas (miles de millones)	20,53	20,53
Ciclos de reloj (miles de millones)	8,65	8,94
Tasa de instrucciones por ciclo (IPC)	**2,37**	**2,29**
Branches (millones)	4.582,6	4.666,4
Branches-misses (millones)	6	5,6
Porcentaje de fallo en predicción de branch	**0,13%**	**0,12%**

resize lake-small.ppm 8000 8000	SOA	AOS
Tiempo de ejecución (segundos)	22,917	22,741

[3] En toda la evaluación se ha obviado la función info, que tiene un carácter constante y no aporta nada la evaluación de rendimiento y energía.

Instrucciones ejecutadas (miles de millones)	48,223	49,81
Ciclos de reloj (miles de millones)	22,398	22,333
Tasa de instrucciones por ciclo (IPC)	**2,15**	**2,23**
Branches (millones)	8.211	8.199,5
Branches-misses (millones)	13	12
Porcentaje de fallo en predicción de branch	**0,16%**	**0,15%**

cutfreq lake-large.ppm 100000	SOA	AOS
Tiempo de ejecución (segundos)	16,286	16,243
Instrucciones ejecutadas (miles de millones)	94,258	94,224
Ciclos de reloj (miles de millones)	45,036	44,959
Tasa de instrucciones por ciclo (IPC)	**2,09**	**2,1**
Branches (millones)	11.650,5	11.638,32
Branches-misses (millones)	17	17
Porcentaje de fallo en predicción de branch	**0,15%**	**0,15%**

compress lake-small.ppm	SOA	AOS
Tiempo de ejecución (segundos)	0,022	0,023
Instrucciones ejecutadas (millones)	32,06	31,98
Ciclos de reloj (millones)	22,37	22,791
Tasa de instrucciones por ciclo (IPC)	**1,43**	**1,4**
Branches (millones)	6,905	6,887
Branches-misses (miles)	71,73	73,75
Porcentaje de fallo en predicción de branch	**1,04%**	**1,07%**

Como se puede observar en el análisis de rendimiento de las funciones, con una tasa de instrucciones media de **2,01 para SOA y 2,005 para AOS** y un porcentaje de fallo en predicción de branch de media **0,37% para SOA y 0,37% para AOS,** ambas configuraciones muestran un rendimiento similar,

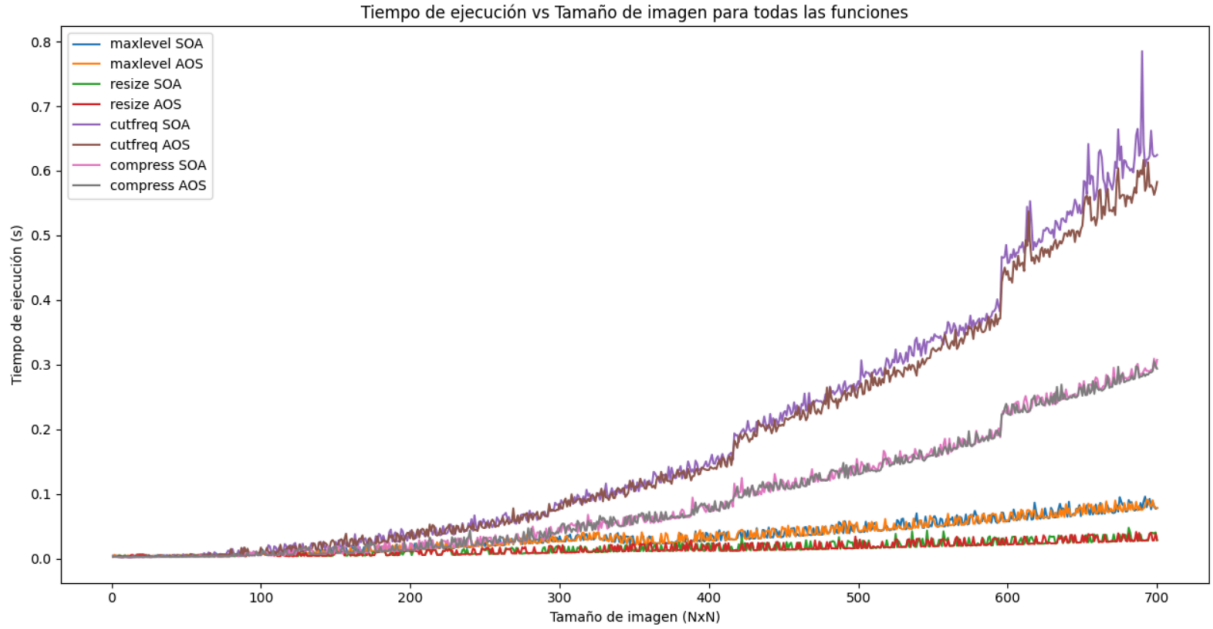
indicando que aprovechan el hardware de manera eficiente. La ligera ventaja de SOA podría deberse a un acceso a datos más optimizado para el tipo de operaciones en el programa.

Análisis de energía:

		SOA (Julios)	AOS (Julios)
Maxlevel	Cores	13,03	13,94
	GPU	0	0
	PKG	48,78	54,61
	RAM	20,51	20,34
Resize	Cores	33,92	33,21
	GPU	0	0,02
	PKG	130,4	127,77
	RAM	53,69	53,24
Cutfreq	Cores	64,04	63,21
	GPU	0,03	0,00
	PKG	146,04	144,05
	RAM	38,99	39,28
Compress	Cores	0,04	0,02
	GPU	0	0
	PKG	0,11	0,1
	RAM	0,04	0,04

Evaluación en máquina local

A continuación, se disponen las gráficas del resultado del análisis de rendimiento en una máquina local, con más recursos que el nodo de Avignon en el que se ha hecho el análisis anterior. Se dispone de un AMD Ryzen 7 4700U. Todas las gráficas fueron ejecutadas en la versión “release”.



Como se puede observar en la gráfica, nuestra función resize es la más costosa temporalmente y en donde más se nota la diferencia entre las versiones SOA y AOS, siendo esta última de una mejoría notable. Cabe destacar, además, que se distingue de forma general la linealidad de las funciones, siendo la mejor maxlevel[4], después resize, compress y por último cutfreq.

Organización del Trabajo

Para gestionar y organizar el desarrollo del proyecto, hemos usado el control de versiones de GitHub. A continuación se detallan las Tareas y Sub tareas en las que se ha dividido el proyecto.

Categoría	Tarea	Sub tarea	Persona asignada	Tiempo dedicado
Análisis Argumentos	Verificación parámetros de entrada	-	Jorge	3,5h
Funciones principales	desarrollo función info	-	Jorge	2h
	desarrollo función maxlevel	maxlevel SOA F3 a F3	Alberto	4h
		maxlevel SOA F3 a F6	Alberto	2h
		maxlevel SOA F6 a F3	Héctor	1h

[4] Aunque aparentemente resize funcione mejor, esta función depende del tamaño de imagen de salida, que en el caso de la gráfica fue especificado a 100x100; un valor relativamente bajo. Si hubiera sido mayor, se hubiera visto cómo maxlevel tiene un tiempo de ejecución menor.

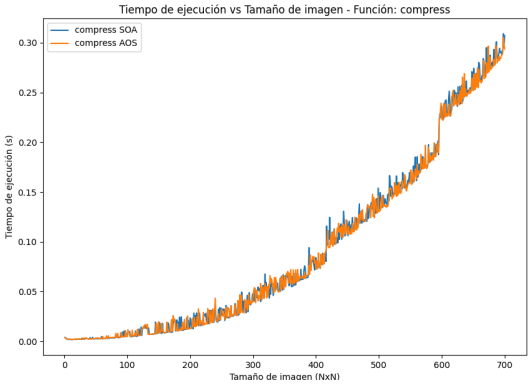
		maxlevel SOA F6 a F6	Héctor	2h
		maxlevel AOS F3 a F3	Alberto	4h
		maxlevel AOS F3 a F6	Alberto	2h
		maxlevel AOS F6 a F3	Héctor	1h
		maxlevel AOS F6 a F6	Héctor	2h
	desarrollo función resize	resize SOA F3 a F3	Alberto	15h
		resize SOA F6 a F6	Alberto	1h
		resize AOS F3 a F3	Alberto	<30m
		resize AOS F6 a F6	Alberto	<30m
	desarrollo función cutfreq	cutfreq SOA F3 a F3	Hector	50h
		cutfreq SOA F6 a F6	Hector	2h
		cutfreq AOS F3 a F3	Hector	2h
		cutfreq AOS F6 a F6	Hector	2h
	desarrollo función compress	compress SOA F3 a F3	Alberto	35h
		compress SOA F6 a F6	Alberto	1h
		compress AOS F3 a F3	Alberto	<30m
		compress AOS F6 a F6	Alberto	<30m
Pruebas	Pruebas funcionales	P.F. parámetros de entrada	Pablo	2h
		P.F info	Pablo	30m
		P.F maxlevel	Alberto	1h
		P.F resize	Alberto	1h
		P.F cutfreq	Hector	2h
		P.F compress	Alberto	4h
	Pruebas unitarias	P.U binario.cpp	Pablo	2h
		P.U progargs.cpp	Jorge	4h
		P.U imageaos.cpp	Pablo	10h

		P.U imagesoa.cpp	Pablo	10h
Evaluación	Evaluación temporal en máquina local	-	Alberto	5h
	Evaluación temporal en Avignon	-	Hector	5h
	Evaluación espacial en máquina local	-	Alberto	2h
	Evaluación energética en Avignon	-	Alberto	3h

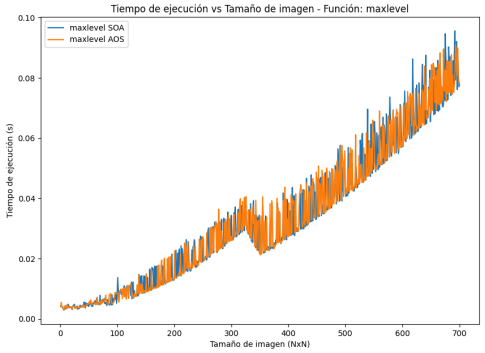
Conclusiones

Como conclusiones finales, en relación a los resultados obtenidos en la evaluación del rendimiento y energía, podemos afirmar la solidez y eficiencia de nuestra implementación, donde ya desde un principio, fue enfocada a la optimización del rendimiento y eficacia del código. Habiendo analizado los resultados obtenidos, consideramos que la estrategia AOS ofrece una ligerísima ventaja con respecto a SOA, pues si bien ambas obtienen Tasa de Instrucciones por Ciclo muy parecidas y un porcentaje de predicción de branches idénticas, la diferencia temporal hecha en nuestra máquina local **sí es notable** cuando estas estrategias se analizan por separado.

Es interesante mencionar, además, que **existe un gran incremento** en el tiempo de ejecución con respecto a las funciones compress y cutfreq (imagen de la derecha), correspondientes a imágenes superiores a 600x600, lo que es explicable, ya que esta cota es donde se empieza a hallar la mayor probabilidad de que el número de colores distintos sea superior a 65535, lo que hace que se tenga que cambiar el protocolo de escritura de los índices, en el caso del compress, y el análisis de los colores únicos, en el caso del cutfreq.



Por otro lado, queremos mencionar los resultados proporcionados por la función maxlevel (imagen de la derecha) y su inesperado decremento temporal correspondiente al valor aproximado de imágenes 310x310. Nuestra hipótesis es que esto se debe a alguna optimización interna del procesador o a la abrupta eficiencia de algún flag de compilación usado, que trata de optimizar las iteraciones de bucles. Es importante mencionar que el comando que se usó para sacar la gráfica mencionada fue maxlevel 200, con imágenes con maxval de 255 cuyos píxeles fueron generados aleatoriamente. Este



comportamiento fue replicado numerosas veces y no hace distinción entre SOA y AOS.

Por último, con respecto al consumo energético de las funciones, en general, SOA parece ser más eficiente en la operación 'Maxlevel' y tiene un consumo de energía algo mayor en la operación 'Resize'. Para 'Cutfreq' y 'Compress', la diferencia es mínima. En ninguno de los casos se utiliza la GPU y en la mayoría de los casos, ambos métodos tienen un consumo de RAM similar, aunque SOA tiende a consumir ligeramente más en 'Resize' y 'Maxlevel', lo que podría indicar que su estructura de datos ocupa más espacio de memoria en ciertas operaciones.

Declaración responsable

Declaramos que, durante el desarrollo de la práctica, se han utilizado herramientas de Inteligencia Artificial Generativa exclusivamente para la creación de fragmentos de código repetitivos o de carácter técnico que no implican procesos de razonamiento conceptual ni creativo. Es importante señalar que dichas herramientas no han sido empleadas para la generación de ideas conceptuales, ni para la planificación, evaluación o análisis de los resultados de la práctica. Todo el trabajo relacionado con el diseño, interpretación y análisis de datos, así como la resolución de problemas y toma de decisiones, ha sido realizado de manera íntegra y autónoma por los integrantes del equipo.

Conclusiones Personales

Tras haber finalizado el desarrollo del proyecto, podemos concluir que ha sido una de las más costosas con respecto a su duración y su dificultad en relación a las demás prácticas **realizadas en la carrera**. Si bien la misma nos ha resultado profundamente educativa, interesante y satisfactoria, nos fascina y desorienta lo poco relacionada que consideramos que está con los contenidos teóricos y prácticos de la asignatura (excepto en el término de evaluación y análisis del rendimiento; aspecto que es porcentualmente bajísimo con respecto a la totalidad de la práctica).

Aún habiendo comenzado la práctica unas horas después de su publicación en Aula Global, nos hemos visto apurados de tiempo, considerando que hemos tenido que cambiar nuestra implementación numerosas veces debido a la carencia de ejemplos significativos y casos de uso que se nos han proporcionado para verificar la solidez y correctitud de nuestra implementación. Si bien es cierto que, posteriormente se publicaron salidas esperadas para nuestras fotografías, las mismas no coincidían con nuestras salidas, lo que supuesto un quebradero de cabeza para, más adelante, ser informados de que:

1. En las salidas proporcionadas de maxlevel había un bit corrupto en la primera posición (después de la cabecera) de la fotografía deer-small-1000.ppm. Esto fue consultado con el coordinador de la asignatura y nos dijo que nuestra implementación era válida.

2. En las salidas proporcionadas de resize, se observaban enormes discrepancias visuales entre lo que parecía “lógico” y lo que nos fue proporcionado. Contactamos con el coordinador de la asignatura y puso a disposición más salidas consideradas como “correctas” para el resize.

3. En las salidas proporcionadas para cutfreq, había grandes discrepancias con respecto a la elección de colores a eliminar con la misma frecuencia (donde hay que verificar entonces la frecuencia de los canales de color por separado). Esta discrepancia también fue comunicada al coordinador y tras ella, puso a disposición las salidas corregidas del cutfreq.

4. En las salidas proporcionadas del compress, la tabla de colores (y por consiguiente, los índices) sigue una secuencia indescifrable. Es decir, tal y como dice el enunciado sobre la tabla de colores del formato ficticio .cppm: "*Una secuencia de valores RGB.*" como no se especifica la secuencia, esta puede ser **cualquiera**, por lo que evidentemente es imposible comparar la validez de nuestros resultados.

El factor común a todos estos inconvenientes es que hemos tenido que ser nosotros los que hemos tenido que avanzar y comunicar para verificar que lo que estábamos haciendo iba por buen camino, **LO QUE HA SUPUESTO PÉRDIDAS DE SEMANAS** para el desarrollo de la práctica, aunque como ya hemos comentado, nos consideramos satisfechos con el resultado de la misma. Para verificar la solidez y validez de nuestras funciones, se proporcionará un anexo a la práctica con todos los script usados para verificar las salidas, explicados brevemente en el apartado de esta memoria: "Anexo".

Aún con todo, agradecemos profundamente la resolución constante de dudas por el foro como a nivel personal, tanto del coordinador de la asignatura como de nuestro profesor de prácticas.

Anexo

Scripts usados para el desarrollo del proyecto:

1. **Recover**: script para descomprimir imágenes desde el formato .cppm de nuevo al formato .ppm. (Funcional para cualquier tipo de secuencia de valores RGB de la tabla de colores).

2. **Compare CPPM**: script para transformar dos imágenes a .cppm y comprobar colores únicos distintos (en caso de haberlos).

3. **checker_cutfreq**: script para verificar, dada una imagen original y dos imágenes procesadas por cutfreq, los píxeles originales, los píxeles cambiados y la distancia euclídea de una imagen y otra al píxel original.