uc3m | Universidad Carlos III de Madrid

Proyecto de programación orientada al rendimiento

Grupo reducido
85
Equipo asignado
08
Nombre y NIA de todos los autores:
Inés Fuai Guillén Peña | 100495752
Claudia Fernández Menéndez | 100495919
Nicola Stefania Cindea | 100495713

uc3m | Universidad Carlos III de Madrid

DISEÑO

Índice de sección:

1. progargs

La funcionalidad de *progargs* es la de comprobar la correcta inserción de los parámetros requeridos por nuestra aplicación para funcionar.
Lo primero que examina es que incluya dos parámetros (*que corresponderían a input.ppm y output.ppm*), revisando con *argc*. Una vez verificado, incluimos una lista con las posibles operaciones a realizar, que corresponderían al tercer parámetro.
Una vez revisados estos parámetros "comunes" y que la operación a realizar está implementada, revisamos los parámetros para cada función, utilizando una estructura con el estilo:

```
Unset
if (operacion == "op_válida") {
      if (argc != n){ // n = nº parámetros para esa operación
            ...
            std::cerr << "Salida estándar de error\n"
            return -1;
      }
}
```

Donde para cada función debemos verificar:

info	- *nº argumentos: 3*
maxlevel	- *nº argumentos: 4* - *valor 4º argumento ∈ [0, 65535]*
resize	- *nº argumentos: 5* - *valor 4º argumento ≥ 0 (∈ Z) → ancho* - *valor 5º argumento ≥ 0 (∈ Z) → alto*
cutfreq	- *nº argumentos: 4* - *valor 4º argumento ≥ 0 (∈ Z)*
compress	- *nº argumentos: 3*

Funcionalidades a destacar:

- Transformación de argumento a *string.* De esta manera, podremos devolver el valor tal y como fue escrito por el usuario al devolver el error.

```C/C++
                        std::string arg = argv[4];
```

- Transformación de string a número. Debido a la restricción sobre el uso de *atoi/stoi,* implementación de condicional sencillo para transformar una cadena de caracteres en un número, en caso de que lo sea:

```C/C++
int maxlevel_valor = 0;
for (char c: arg) {
      maxlevel_valor = maxlevel_valor * 10 + (c - '0');
}
```

  Para cada elemento de la cadena, multiplica su valor por $10^{posición}$ Necesitamos restar '0' pues c da lugar a un número en código ASCII, donde '0' = 48, '1' = 49…Dependiendo del valor de c, debemos restar el '0' para obtener el verdadero valor decimal.

2. Diferenciación *AOS/SOA*

Hemos trabajado tratando las imágenes como objetos. La clase *ImageSOA/AOS* implementa los parámetros que tiene nuestra imagen:

- *input_file*
- *output_file*
- *formato:* P6, pues trabajamos con archivos .ppm
- *valor_max_color:* Valor de la intensidad con mayor valor de entre los píxeles
- *width:* ancho de la imagen
- *height :* alto de la imagen

Nuestra implementación en *AOS (Array of Structures)* consiste en una estructura de píxeles. En concreto, una estructura con tres valores: *red, green, blue*. Hemos implementado dos estructuras base: una con píxeles de 1 byte (8 bits) y otra con 2 bytes (16 bits). Esta última será necesaria en funciones como *maxlevel,* pero generalmente se ve de la siguiente manera:

```C/C++
 struct Pixel {
      uint8_t r;  // rojo
      uint8_t g;  // verde
      uint8_t b;  // azul
};
```

La clase *ImageAOS* utiliza un vector de objetos *Pixel* que agrupa estos tres valores en un solo bloque para cada píxel. No tiene una estructura explícita para mapear píxeles a índices a través de un *unordered_map*, pero se proporciona una estructura *PixelHash* para permitir el uso de *Pixel* como claves en un *unordered_map* mediante la combinación de los valores de las componentes r, g y b.

Por otra parte, la implementación con *SOA (Structure of Arrays)* implementa tres vectores: *rojo, verde y azul*. En este caso, el vector *rojo* tendría en su primera posición el valor rojo del primer píxel, en la segunda el del segundo píxel… Esto da lugar a *tres vectores con el mismo tamaño,* donde cada posición *n* corresponde a los tres colores para el píxel *n+1 (la posición 0 corresponde al 1er pixel.)*

```C/C++
std::vector<uint8_t> r_8;
std::vector<uint8_t> g_8;
std::vector<uint8_t> b_8;
```

Estas estructuras se implementarían en la parte *privada* de la clase. En la parte pública, simplemente declaramos el objeto y las funciones disponibles.

3. Función *info*

La función *info* no cambia dependiendo de AOS/SAO. Función sencilla que se encarga de abrir un archivo .ppm en modo binario para leer la cabecera y extraer información.
Tras abrir el archivo, simplemente lee los datos de la imagen (*recordar que tratamos las imágenes como objetos, por lo que info no necesita que se le proporcione ningún parámetro externo*) y devolverlos al usuario usando un '*std::cout'.*

4. Función *maxlevel*

ver pt.9 de la sección para funcionamiento de 'read' y 'write'

- Implementación en AOS. Recibir input, output y valor numérico. Leer imagen correspondiente al input. Operación *reescalado*, que consiste en una división del valor numérico dado entre el mayor valor de intensidad actual (*C++ trunca el resultado de esta operación de forma automática*). Para todos los píxeles de la imagen, realizar un reescalado de *red, green* y *blue.* Si ambos valores son menores de 255, se guardan en la estructura de 8 bits (*uint8_t*). Si no, guardar en una estructura de píxeles de 16bits *(uint16_t).* Escribir los nuevos datos en el archivo dado como *output.*
- Implementación en SOA. Recibir input, output y valor numérico. Leer imagen correspondiente al input. Operación *reescalado*, que consiste en una división del valor numérico dado entre el mayor valor de intensidad actual (*C++ trunca el resultado de esta operación de forma automática*). En el rango del tamaño de uno de los vectores (*ya hemos establecido que todos serán del mismo tamaño*), para cada elemento en esa posición, multiplicar por el resultado de *reescalado*. Si el valor dado es autor que 255, debemos asignar a los vectores *uint16_t* su valor equivalente en 8bits y multiplicar por el reescalado, haciendo *write* de los de 16bits (si no, el programa devuelve un .ppm completamente negra)

5. Función *resize*

- Implementación en AOS: Recibe como parámetros el input(ruta de la imagen), output(ruta para guardar la nueva imagen), ancho y alto al que queremos convertir la imagen. La función lee el input con readPPM_maxlevel y se calcula el factor de escala en los ejes x e y. Para redimensionar la imagen se utiliza la interpolación bilineal(eliminar píxeles en función de los valores de los píxeles vecinos). Después se hace una validación para comprobar que los índices están dentro del rango se aplica la interpolación bilineal para calcular los nuevos colores realizando la interpretación de red, green y blue.. Finalmente se asigna el pixel calculado a su posición correspondiente y la función writePPM_maxlevel realiza la escritura del output.
- Implementación en SOA: Leemos la imagen igual que en AO, y creamos un objeto imagen. Leemos la imagen original, y redimensionar los vectores *r_8, g_8 y b_8* al tamaño de la imagen original. Calculamos las proporciones de redimensionamiento (*div_x, div_y*), que son el ancho y alto originales divididos entre el ancho y alto recibido. Si visualizamos la imagen como una matriz de píxeles, la matriz la recorreremos de izquierda a derecha y de arriba a abajo (*todos los píxeles de la primera fila, todos los de la segunda…*). Después, para cada píxel, calculamos los 4 píxeles más próximos al actual. Hacemos el interpolado y lo aplicamos para cada píxel (rgb). Finalmente, escribimos el nuevo valor de cada píxel y llamamos a *write* para escribir la imagen redimensionada en el output.

6. Función cutfreq

- Implementacion en AOS. El código elimina los *n* colores menos frecuentes de una imagen y los reemplaza por colores más cercanos en términos de distancia euclidiana. Primero, cuenta la frecuencia de cada color, identifica los menos comunes, y busca para cada uno el color más cercano que no esté en la lista de eliminación. Luego reemplaza esos colores en los píxeles de la imagen y guarda el resultado en un archivo de salida.
- Implementación en SOA. Primero, cuenta la frecuencia de cada color en la imagen y luego ordena estos colores por su frecuencia en orden ascendente. Después, selecciona los *n* colores menos frecuentes, los cuales serán eliminados, y busca el color más cercano a cada uno, basado en la distancia euclidiana. Finalmente, reemplaza los colores menos frecuentes en la imagen y guarda el resultado en un archivo en formato PPM.

7. Función *compress*

ver pt.9 de la sección para funcionamiento de 'read' y 'write'

- Implementación en AOS. Primero, crea una tabla de colores única y un mapa de índices, iterando sobre todos los píxeles de la imagen y asignando un índice único a cada color. Luego, calcula el tamaño adecuado para los índices (1, 2 o 4 bytes) según la cantidad de colores en la tabla. A continuación, escribe un encabezado binario con información sobre la imagen, como dimensiones y número de colores, seguido de la tabla de colores en el archivo de salida. Finalmente, la función escribe los índices de los píxeles de la imagen, utilizando el tamaño de índice previamente calculado, y cierra el archivo. Esto reduce el tamaño de la imagen al almacenar los colores repetidos solo una vez en la tabla, representándolos luego por su índice.

- Implementación en SOA. La función toma una imagen, crea una tabla de colores única y la almacena en un formato binario comprimido. Primero, recorre todos los píxeles de la imagen, generando una tabla con los colores únicos y asignando un índice a cada uno. Luego, determina el tamaño adecuado para los índices según la cantidad de colores únicos, almacenándolos en 1, 2 o 4 bytes. A continuación, escribe un encabezado con información de la imagen, como dimensiones y número de colores, seguido de la tabla de colores en el archivo de salida. Finalmente, se almacenan los índices de los colores para cada píxel de la imagen y se cierra el archivo.

8. main

La estructura de los *main* es prácticamente idéntica para AOS y SOA. Primero incluimos las bibliotecas necesarias: *common, image(aos/soa).* Tomamos como inputs *argc* y *argv,* y llamamos a *progargs* para comprobar que los parámetros fueron introducidos correctamente. A continuación (*progargs no ha devuelto errores*), de argv sacamos input (argv[3]), output (argv[3]) y operación a realizar (argv[3]). Generamos un objeto ImageAOS/SOA con ese input para trabajar con él, y generamos distintos *if* y *else-if* para llamar a las distintas funciones dependiendo del valor de la operación (*argv[3]*).

9. Funciones externas

AOS

- *readPPM:* Función que abre un archivo en binario y extrae sus datos de cabecera (*número mágico, ancho alto y valor máxima intensidad*) y actualiza el tamaño de los píxeles. Este último será *ancho·alto·3,* pues los datos se expresan como una cadena *RGBRGBRGBRGB…*, donde cada 'RGB' corresponde a un píxel.
- *writePPM:* Crea o abre un directorio dado, escribe datos valorando el tamaño de la intensidad (para ver si necesita usar vectores de 1 o 2 bytes).
- *readPPM_maxlevel:* Función *read* optimizada para maxlevel. Funcionamiento similar al normal, enfocado principalmente en asegurar que el formato del número mágico sea *P6* y leyendo/reescalando los vectores para los píxeles, basándose en que el número de intensidad máxima sea mayor o menor que 255, para guardarlos en una estructura de 1 o 2 bytes.
- *writePPM_maxlevel:* Función *write* optimizada para maxlevel. De nuevo, especial énfasis en el formato y cambia la escritura dependiendo de si el valor nuevo para maxlevel es mayor o menor que 255.
- *euclidean_distance*: Calcula la distancia euclidiana entre dos colores representados por los píxeles p1 y p2 en el espacio RGB. Para hacerlo, toma las diferencias entre las componentes r, g y b de los dos píxeles, eleva cada diferencia al cuadrado, suma estos valores y devuelve la raíz cuadrada del resultado.

Nota: Tratamos de encontrar una implementación en la que no fuera necesario tener dos funciones read y dos write, sin embargo el entorno de programación nos daba problemas y los distintos mínimos cambios que alguien introducía hacía que la función de otra persona dejara de funcionar. Es por esto que nos vimos obligadas a realizar estas implementaciones. Para la función resize utilizamos readPPM_maxlevel y writePPM_maxlevel.

SOA:

- *readPPM:* Función que abre un archivo en binario y extrae sus datos de cabecera (*número mágico, ancho alto y valor máxima intensidad*) y actualiza el tamaño de los vectores *r, g, b*. El tamaño de cada vector será el resultado de hacer *ancho·alto.*
- *writePPM:* Crea o abre un directorio dado, escribe datos valorando el tamaño de la intensidad (para ver si necesita usar vectores de 1 o 2 bytes). Debe escribir tres veces, una por cada vector. Se utilizan bucles *for* basados en el tamaño del vector *r (como los tres vectores corresponden a la misma imagen, tienen el mismo tamaño, pues el conjunto cada posición de cada vector son los colores de un píxel).*
- *readPPM_maxlevel:* Función *read* optimizada para maxlevel. Funcionamiento similar al normal, enfocado principalmente en asegurar que el formato del número mágico sea *P6* y leyendo/reescalando los vectores para los píxeles, basándose en que el número de intensidad máxima sea mayor o menor que 255, para guardarlos en una estructura de 1 o 2 bytes. Tanto el resize como las lecturas deben hacerse en los tres vectores correspondientes a *r, g, b..*
- *writePPM_maxlevel:* Función *write* optimizada para maxlevel. De nuevo, especial énfasis en el formato y cambia la escritura dependiendo de si el valor nuevo para maxlevel es mayor o menor que 255. De nuevo, debemos hacer un write para cada vector.
- *euclidean_distance*: Calcula la distancia euclidiana entre dos colores representados en el espacio RGB, donde cada color está compuesto por tres componentes: rojo (r), verde (g) y azul (b). Primero, convierte los valores de los colores de tipo *uint8_t* (que pueden estar en el rango de 0 a 255) a *int* para evitar problemas de desbordamiento al realizar la resta. Luego, calcula las diferencias entre los componentes de los colores (*dr*, *dg* y *db*) y utiliza la fórmula de la distancia euclidiana para obtener la distancia entre los dos colores. El resultado es un valor de tipo *double* que representa cuán similares o diferentes son los dos colores en el espacio RGB.

OPTIMIZACIONES

maxlevel(AOS):

- Utilización de la variable *reescalado* para no tener que calcular *nuevo_valor_max/valor_max_color* cada vez que necesitemos utilizarlo.
- Uso de tipos diferenciados, *unsigned char* y *uint16_t* dependiendo del rango de valores para cada situación, asegurando compatibilidad.
- Bucle *for* compacto y dependiente de un rango (*cuántos píxeles hay en una imagen*)
- Uso directo de la estructura *Pixel*
- Uso de condicionales de tipo *if* para cambiar valores solo en casos estrictamente necesarios

maxlevel(SOA):

- Variable *reescalado* para no tener que calcular *nuevo_valor_max/valor_max_color* cada vez que necesitemos utilizarlo.
- Uso de tipos diferenciados, *uint8_t* y *uint16_t* dependiendo del rango de valores para cada situación, asegurando compatibilidad.
- Bucle *for* compacto y dependiente de un rango (*tamaño de un vector*)
- Uso de condicionales de tipo *if* para cambiar valores solo en casos estrictamente necesarios

Ambas funciones hacen uso de las funciones externas readPPM_maxlevel y writePPM_maxlevel, reduciendo el tamaño de la función y permitiendo centrarnos únicamente en el reescalado.

resize(AOS):

- Utilización de variables *div_x* y *div_y,* para no tener que realizar la operación cada vez que necesitemos saber su valor.
- Interpolación bilineal: La interpolación en sí misma es un método enfocado en la optimización.

resize(SOA):

- Utilización de variables *div_x* y *div_y,* para no tener que realizar la operación cada vez que necesitemos saber su valor.
- Interpolación bilineal: La interpolación en sí misma es un método enfocado en la optimización. Se realiza de forma directa sobre cada canal individual.

Ambas funciones hacen uso de las funciones externas readPPM_maxlevel y writePPM_maxlevel, pues por la naturaleza de la función estas se ajustan convenientemente a nuestros requisitos, reduciendo el tamaño de la función y permitiendo centrarnos únicamente en el reescalado.

compress (AOS):

- Uso de *unordered_map* para búsqueda rápida con un hash personalizado para almacenar el índice de cada color (*Pixel*) único.
- Optimización en la creación de la tabla de colores, se construye de manera eficiente al iterar solo una vez sobre los píxeles y almacenar cada color único junto con su índice
- Escritura eficiente en formato binario. En lugar de escribir los datos como texto, se

utilizan operaciones de escritura binaria (*file.write*), lo que mejora considerablemente la velocidad y la eficiencia en términos de espacio en disco.
- Gestión eficiente de la memoria: El uso de una estructura de datos como *std::vector<Pixel>* para almacenar los colores únicos y su índice, junto con el *unordered_map* para la indexación rápida, asegura que el proceso de compresión sea eficiente tanto en términos de tiempo como de espacio.

compress (SOA):
- Uso de *unordered_map* para la tabla de colores para asociar cada color único con un índice.
- Optimización en la creación de la tabla de colores. Al iterar sobre los píxeles, se verifica si un color ya está en la tabla mediante *color_to_index.find(color)*. Si no está, se agrega a la tabla y se asigna un índice, lo que evita duplicados y asegura que cada color solo se almacene una vez.
- Escritura binaria optimizada: Se usa *file.write* en lugar de escribir en formato texto, lo que es más rápido y eficiente para la compresión de datos. Además, al escribir los colores, se diferencia entre escribirlos como 1, 2 o 3 bytes, dependiendo del valor máximo del color (*valor_max_color*)

Ambas funciones hacen uso de las funciones externas readPPM y writePPM, pues por la naturaleza de la función estas se ajustan convenientemente a nuestros requisitos, reduciendo el tamaño de la función y permitiendo centrarnos únicamente en el reescalado

PRUEBAS REALIZADAS

progargs: Revisamos la entrada de argumentos, centrándonos en forzar errores (menos de tres argumentos, operaciones no válidas…). También probamos un par de entradas correcta, para verificar que nos devuelve ‘0’.

info: Buscamos poder leer correctamente un archivo, forzar error si introducimos un archivo no existente, forzar error por formato incorrecto o forzar error por cabecera incompleta.

maxlevel: Input de pixeles “a mano” para comprobar que los cálculos se hacen de forma correcta, forzar errores en los archivos de entrada o salida, probar con un nuevo valor máximo mayor que 255

compress: Verificar si la compresión funciona correctamente cuando la imagen tiene una pequeña tabla de colores (solo 3 colores únicos en este caso), que maneje correctamente cuando se proporciona una ruta de archivo inválida, que falle si hay más colores únicos de los que el formato soporta, verificar que la compresión funcione correctamente con colores de 16 bits (en lugar de 8 bits)

Las salidas de imagen han sido comprobadas mediante comprobación entre nuestros outputs y los outputs de prueba facilitados. Lo ideal habría sido implementar una función inversa para comprobar exactamente que los valores introducidos son como los proporcionados, pero por desgracia no hemos contado con el tiempo suficiente para desarrollarlo, y hemos decidido priorizar otros aspectos del proyecto con mayor peso en la evaluación general.

EVALUACIÓN RENDIMIENTO Y ENERGÍA

Evaluación maxlevel

maxlevel 65535 para lake-large.ppm

AOS:

```
 Performance counter stats for './build/imtool-aos/imtool-aos /home/alumnos/a0495919/arquis/arquitect3/in/lake-large.ppm /home/alumnos/a0495919/arqu
s/out.cppm maxlevel 65535':

          284,09 msec task-clock                #    0,015 CPUs utilized
           2.020      context-switches          #    7,111 K/sec
              12      cpu-migrations            #   42,241 /sec
           1.563      page-faults               #    5,502 K/sec
     909.032.787      cycles                    #    3,200 GHz
   1.165.200.004      instructions              #    1,28  insn per cycle
     149.210.904      branches                  #  525,232 M/sec
         712.510      branch-misses             #    0,48% of all branches

    18,770065014 seconds time elapsed

     0,115609000 seconds user
     0,181671000 seconds sys
```

```
18,770065014 seconds time elapsed

 0,115609000 seconds user
 0,181671000 seconds sys
```

SOA:

```
Performance counter stats for './build/imtool-soa/imtool-soa /home/alumnos/a0495919/arquis/arquitect/in/lake-large.ppm /home/alumnos/a0495919/arquis/out.ppm maxlevel 65535':

       1.150,70 msec task-clock                #    0,155 CPUs utilized
             680      context-switches          #  590,943 /sec
               0      cpu-migrations            #    0,000 /sec
           3.508      page-faults               #    3,049 K/sec
   3.863.367.379      cycles                    #    3,357 GHz
  12.304.876.430      instructions              #    3,19  insn per cycle
   2.545.266.693      branches                  #    2,212 G/sec
         570.465      branch-misses             #    0,02% of all branches

    7,419776171 seconds time elapsed

    1,080529000 seconds user
    0,076604000 seconds sys
```

```
7,419776171 seconds time elapsed

1,080529000 seconds user
0,076604000 seconds sys
```

Podemos observar que SOA es más rápido. En SOA los datos están almacenados de forma alineada y contigua (rrrr…rrgggg……ggbbbb…bb). Permite aplicar operaciones sin necesidad de reordenar. En AOS, los datos se ven como *rgbrgbrgbrgb…..,* que dificulta el acceso a cambios en colores en específico.

Evaluación anterior:

El código en AOS presentaba un error que en su momento no detectamos, pues al optimizar perdíamos el output buscado. En concreto ,la línea que causa el delay es:

```
if (valor_max_color != nuevo_valor_max) {
    valor_max_color = static_cast<uint16_t>(nuevo_valor_max);
}
```

Quitar esta línea supone que el tiempo de compilación se reduzca a unos 3 segundos, pero el output que devuelve es incorrecto. Tras un arduo análisis, no sabemos cómo optimizar el asignamiento de este valor sin sacrificar tiempo de compilación. Desgraciadamente, nuestro maxlevel con AOS no llega al mínimo de tiempo permitido. Usando la implementación errónea:

```
Performance counter stats for './build/imtool-aos/imtool-aos /home/alumnos/a0495919/arquis/arquitect/in/lake-large.ppm /home/alumnos/a0495919/arquis/out.ppm maxlevel 65535':

          206,33 msec task-clock                #    0,062 CPUs utilized
               330      context-switches          #    1,599 K/sec
                 1      cpu-migrations            #    4,847 /sec
             1.558      page-faults               #    7,551 K/sec
       663.288.852      cycles                    #    3,215 GHz
     1.049.682.149      instructions              #    1,58  insn per cycle
        95.273.880      branches                  #  461,765 M/sec
           152.003      branch-misses             #    0,16% of all branches

      3,343221240 seconds time elapsed

      0,123783000 seconds user
      0,085101000 seconds sys
```

```
3,343221240 seconds time elapsed

0,123783000 seconds user
0,085101000 seconds sys
```

Evaluación *resize:*

AOS

```
Performance counter stats for './build/imtool-aos/imtool-aos /home/alumnos/a0495919/arquis/arquitect3/in/lake-small.ppm /home/alumnos/a0495919/arquis/out.ppm resize 8000 8000':

        1.721,32 msec task-clock                #    0,094 CPUs utilized
            1.709      context-switches          #  992,843 /sec
                1      cpu-migrations            #    0,581 /sec
           77.424      page-faults               #   44,979 K/sec
    5.617.333.128      cycles                    #    3,263 GHz
   11.338.561.329      instructions              #    2,02  insn per cycle
      498.802.270      branches                  #  289,779 M/sec
          791.409      branch-misses             #    0,16% of all branches

     18,257971368 seconds time elapsed

      1,459150000 seconds user
      0,275839000 seconds sys
```

```
18,257971368 seconds time elapsed

 1,459150000 seconds user
 0,275839000 seconds sys
```

SOA

```
Performance counter stats for './build/imtool-soa/imtool-soa /home/alumnos/a0495919/arquis/arquitect3/in/lake-small.ppm /home/alumnos/a0495919/arquis/out.ppm resize 8000 8000':

        4.532,41 msec task-clock                #    0,215 CPUs utilized
            1.711      context-switches          #  377,504 /sec
                7      cpu-migrations            #    1,544 /sec
           15.853      page-faults               #    3,498 K/sec
   15.290.992.953      cycles                    #    3,374 GHz
   43.225.220.734      instructions              #    2,83  insn per cycle
    7.632.399.825      branches                  #    1,684 G/sec
        1.258.664      branch-misses             #    0,02% of all branches

     21,034255667 seconds time elapsed

      4,379646000 seconds user
      0,163986000 seconds sys
```

```
21,034255667 seconds time elapsed

 4,379646000 seconds user
 0,163986000 seconds sys
```

En este caso, la implementación mediante SOA es ligeramente más rápida que la de AOS. La principal diferencia es que AOS maneja un solo vector que almacena estructuras, mientras que SOA trabaja con tres vectores. Esto hace que AOS pueda realizar la interpolación directamente sobre los valores RGB de cada píxel, mientras que SOA requiere más pasos para cada color y le hace perder algo de eficiencia en ese aspecto. Esto provoca esa pequeña diferencia de tiempo entre uno y otro.

Igualmente, estamos considerablemente por debajo del umbral permitido (33 segundos), lo que nos lleva a pensar que nuestra función está bien optimizada.

compress
AOS

```
  3.632.136      instructions                     #    0,99  insn per cycle
    725.042      branches                         #  440,797 M/sec
     22.748      branch-misses                    #    3,14% of all branches

0,107171044 seconds time elapsed

0,002308000 seconds user
0,000000000 seconds sys
```

SOA

```
     26.437.266      instructions                     #    1,06  insn per cycle
      5.788.086      branches                         #  702,497 M/sec
         71.885      branch-misses                    #    1,24% of all branches

    0,024982299 seconds time elapsed

    0,008615000 seconds user
    0,000000000 seconds sys
```

ORGANIZACIÓN DEL TRABAJO

Claudia Fernández - 100495919	Estructuración del proyecto (3 horas) Desarrollo de *progargs.cpp* (2 horas) Desarrollo de *info* (1,5 horas) Desarrollo de *maxlevel AOS*(7 horas) Desarrollo de *maxlevel SOA*(2 horas) Unitest *info, progargs* (2 horas) Unittest *maxlevel_AOS* (1 hora) Optimización maxlevel AOS (2 horas) Debugging *resize AOS* (2 horas) Debugging *resize SOA* (3 horas) Optimización *maxlevel* (2 horas) - sin éxito Configuraciones CMakes/clang-tidy (3 horas)
Inés Guillén - 100495752	Desarrollo de *binario.cpp* (2 horas) Desarrollo de *compress AOS* (6 horas) Desarrollo de *compress SOA* (3 horas) Unitest *compress* (1 hora) Desarrollo de *resize SOA* (7 horas) Desarrollo de *resize AOS* (4 horas) Desarrollo de *cutfreq SOA* (3 horas) - sin éxito Desarrollo de *cutfreq AOS* (3 horas) - sin éxito
Nicola Cindea - 100495713	Planteamiento de SOA (2 horas) Planteamiento de AOS (3 horas)

uc3m | Universidad Carlos III de Madrid

DECLARACIÓN USO DE IA

Los miembros del equipo 8 del grupo reducido 85 de la asignatura Arquitectura de Computadores declaramos el uso de Inteligencia Artificial como herramienta y apoyo a la programación. Destacar su uso para la creación e implementación de pruebas unitarias, así como apoyo en el proceso de debugging y asesoramiento en el desarrollo de las funciones.
Todas las decisiones finales sobre la implementación, resolución de problemas, optimización y generalmente diseño han sido tomadas por miembros del grupo. Podemos afirmar que comprendemos el código presentado y que estas herramientas han sido utilizadas exclusivamente como apoyo.

CONCLUSIONES Y PROBLEMAS ENCONTRADOS

El desarrollo de este proyecto nos ha permitido poner en práctica conceptos aprendidos en clase y en los laboratorios. Principalmente, familiarizarnos con el trabajo en un clúster, en el que podemos movernos únicamente mediante comandos por terminal, nos ha permitido adquirir más conocimiento del desarrollo en este tipo de entornos. Hemos podido trabajar con objetos (*OOP*) aplicando distintos tipos y tamaños de operadores, observar la potencia y energía consumida y evaluar el rendimiento de nuestros programas. Hemos podido aplicar las distintas técnicas de optimización de compilador para nuestros propios programas.

Nos gustaría mencionar problemas con los que nos hemos topado durante el desarrollo. Principalmente mencionar la compilación en Avignon, en la que clang-tidy nos daba un error interno al compilar. Este error aparecía, aparentemente, de forma arbitraria. Por recomendación del coordinador de la asignatura, el CMakeLists de nuestro proyecto se entrega con la parte de clang-tidy comentada. Somos conscientes de que necesitamos clang-tidy-19 para compilar en avignon. El hecho de que no aparezca el -19 es debido a que en local, por estructuración de nuestro proyecto, debíamos quitar ese especificador de versión para que nos funcionara (de nuevo, esta solución fue propuesta por uno de los profesores de la asignatura para trabajar de forma local). Hemos tratado de resolver todos los errores de clang-tidy posibles, pero no podemos confirmar que se hayan añadido todos.

Por último, destacar la ausencia de la función *cutfreq*. La función ha sido planteada y comentada en el código para evitar problemas. Los problemas relacionados con miembros del equipo, como una dispensa temprana o poca base sobre la que trabajar, nos han llevado a enfrentarnos a una carga de trabajo desigual, lo que ha hecho que este código esté desarrollado en su gran mayoría tan solo por dos personas, lo que ha supuesto un gran esfuerzo para llegar a los plazos establecidos (contando con una ampliación de tiempo de 3 días) y garantizar la cobertura de todos los aspectos propuestos en la práctica.