Memoria sobre la primera fase

Grupo A – PL1, Arquitectura de Computadores (21-22)

UO283319 – Juan Francisco Mier Montoto
UO270399 – Gaspar Campos-Ansó Fernández
UO284226 – Pablo del Dago Sordo
UO285381 – Jorge Loureiro Peña

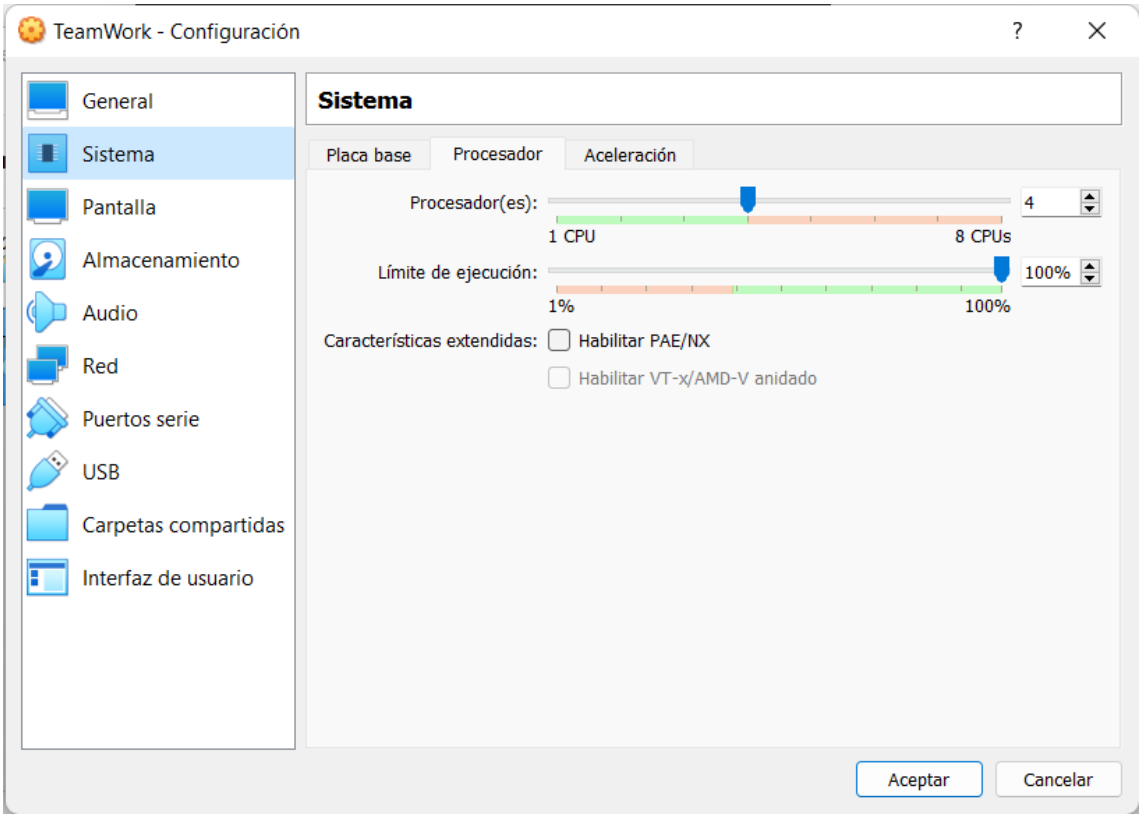
Índice

Repositorio y configuración inicial

Para el desarrollo de esta fase se han seguido todos los pasos incluidos en el enunciado del trabajo:

- Se ha creado un fork en BitBucket del repositorio original para almacenar y compartir el código. Dicho fork se puede encontrar aquí.
- Se han utilizado máquinas virtuales con Ubuntu con las especificaciones indicadas. Para obtener los tiempos se han utilizado 4 cores a 1.2GHz con 8GB de RAM. Las especificaciones más concretas se pueden encontrar en el archivo "cpuinfo" del repositorio.
- Dentro de la propia máquina virtual se ha actualizado Ubuntu a la versión 20.04, se han actualizado todos los paquetes a través de dpkg y se han utilizado algunas extensiones extra dentro de VSCode.

Como especificado, se ha utilizado la siguiente configuración de la máquina virtual:



Desarrollo del algoritmo

Explicación del código

La estructura del código viene proporcionada por el repositorio original, de modo que solo hay que hacer unas pequeñas modificaciones para adaptar el código al algoritmo en cuestión, en nuestro caso el nº12.

Como para nuestro algoritmo se requieren mezclar dos imágenes, hay que añadir al código un objeto que almacene la imagen en la que apoyarse. Dicha imagen, proporcionada por el paquete de imágenes de prueba, es “background_V.bmp”, por lo que por defecto el programa buscará esta imagen en el directorio raíz.

```
// Se inicializan los objetos principales y se abren las imágenes.
CImg<data_t> srcImage(SOURCE_IMG);
CImg<data_t> aidImage(HELP_IMG);
```

También se tienen que crear variables que almacenen los componentes RGB de la imagen auxiliar.

```
data_t *pRaid, *pGaid, *pBaid;
```

Con el objetivo de comprobar que el algoritmo en sí funciona, se ha utilizado la misma imagen que la proporcionada en el ejemplo del algoritmo, “bailarina.bmp”. Además, la imagen resultante de la mezcla se guarda en “result.bmp”.

```
const char* SOURCE_IMG      = "bailarina.bmp"; //
const char* HELP_IMG        = "background_V.bmp";
const char* DESTINATION_IMG = "result.bmp"; // nor
```

Antes de comenzar el algoritmo, se comprueban que ambas imágenes tengan el mismo tamaño, puesto que no tendría sentido sumar dos imágenes con diferentes dimensiones.

Se inicializan las componentes:

```
// Punteros a la imagen original
pRsrc = srcImage.data();          // componente roja
pGsrc = pRsrc + height * width; // componente verde
pBsrc = pGsrc + height * width; // componente azul

// Punteros a la imagen de apoyo
pRaid = aidImage.data();          // componente roja
pGaid = pRaid + height * width; // componente verde
pBaid = pGaid + height * width; // componente azul

// Punteros a la imagen resultante
pRdest = pDstImage;                // componente roja
pGdest = pRdest + height * width; // componente verde
pBdest = pGdest + height * width; // componente azul
```

se guarda el tiempo inicial;

```c
// Tiempo inicial
if (clock_gettime(CLOCK_REALTIME, &tStart)==-1) {
    printf("Error al obtener el tiempo inicial.");
    exit(1);
}
```

y se comienza con el tratamiento de imágenes. El algoritmo se repite 17 veces para que esté dentro del objetivo puesto en el enunciado de estar entre 5 y 10 segundos.

El algoritmo en cuestión se divide en tres fases:

1. Se aplica la fórmula y se guardan los valores obtenidos.
2. Se comprueban que los valores estén entre 0 y 255.
3. Se aplican los valores al píxel de la imagen resultante.

Traducido a código:

1.
```c
// Algoritmo real:      You, a day ago • añadidas repeticio
// Blend: blacken mode #12
red = 255 - ((256 * (255 - pRaid[i])) / (pRsrc[i] + 1));
green = 255 - ((256 * (255 - pGaid[i])) / (pGsrc[i] + 1));
blue = 255 - ((256 * (255 - pBaid[i])) / (pBsrc[i] + 1));
```

2.
```c
red = max(min(red, 255), 0);
green = max(min(green, 255), 0);
blue = max(min(blue, 255), 0);
```

3.
```c
pRdest[i] = red;
pGdest[i] = green;
pBdest[i] = blue;
```

Resultados de tiempos

Se registran los tiempos con la menor carga de CPU posible para obtener resultados reales:

T1	T2	T3	T4	T5
6,265039	6,600691	5,797651	6,169624	6,856766
T6	**T7**	**T8**	**T9**	**T10**
5,716565	6,087448	5,568398	6,283128	5,704669
Media	**Desviación T.**	**Interv. Conf-**	**Interv. Conf+**	
6,1049979	0,41716502	5,68683762	6,52315818	

Como esperado, los tiempos están entre cinco y diez segundos. Se utilizarán estos tiempos en la siguiente fase para comparar el rendimiento de los programas y obtener aceleraciones.

Reparto del trabajo

El trabajo se ha repartido de esta manera:

- Gaspar Campos-Ansó: control de tiempos. (30%)
- Juan Mier: memoria, cambios menores en el código. (30%)
- Pablo del Dago: algoritmo. (30%)
- Jorge Loureiro: control de calidad. (10%)