Instituto Tecnológico de Costa Rica
Engineering School
Computer Engineering
CE 4301 — Arquitectura de Computadores I
I Semester 2020

Project

Diseño e Implementación de una aplicación para Nitidez en Imágenes

María Gabriela Ávila Quesada
Group 01
Professor: Luis Chavarría Zamora

30 de mayo de 2020

Índice

1. Listado de requerimientos del sistema

Para implementación de esta aplicación se requiriere una computadora con al menos las características:

- Procesador Dual core, con 4 Gb de ram, y 2 Gb libres de espacio.
- Tener instalado al menos el sistema operativo de Windows 7.
- Tener instalado Python, las bibliotecas de Python cv2, pyplot, tkinter.
- Tener el emulador de mips Mars4_5.
- Tener instalado java.

En el mercado existen diferentes aplicaciones en alto nivel y el bajo nivel que procesan o aplican filtros a la imagenes, como por ejemplo, la aplicación que realizó Giménez-Palomare [1] junto con un grupo de trabajo donde se aplica diferentes kernel a las imágenes, en este caso, ellos lo desarrollaron en Matlab.

2. Elaboración de opciones de solución al problema

1. **Primera propuesta**

   En la primera propuesta la parte de visualización, donde se selecciona la imagen y donde se visualiza después de hacer el procesamiento de la misma se realiza en python, siendo este el lenguaje de alto nivel.

   La parte de ensamblador sería en x86, para ambas propuesta el diagrama de la imagen 3 sería el mismo.

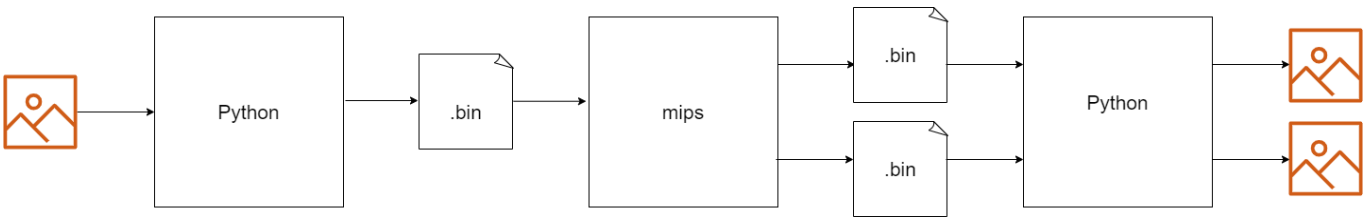

Figura 1: Diagrama de bloques propuesta 1.

   El flujo que tendría el diagrama de bloques de la imagen 1 es: por medio de un lenguaje de alto nivel, en este caso Python se selecciona la imagen que se desea procesar, en esta parte de la implementación la imagen se convierte en un archivo binario.

   En x86 se haría una manipulación del archivo binario generado anteriormente, por medio del diagrama de flujo de la imagen 3 se genera dos binario, los cuales, corresponderían al resultado de la convolución entre el kernel de sharpened y oversharpened, cada kernel guardaría el resultado su propio archivo.

Por ultimo, en Python se haría la conversión de los archivos binarios de la parte de ensamblador a las imagen obtenidas, además, se visualizan en pantalla.

Para ejecutar los código se tiene un makefile .bat.

2. **Segunda propuesta**

En esta propuesta el lenguajes de alto nivel sería Python y el lenguaje de bajo nivel sería mips y se ejecutaría en el simulador desarrollado por Missouri State University Mars.

Partiendo del diagrama de bloques de la imagen 2, se tiene que por medio de Python se selecciona la imagen que se desea procesar, como Mars tiene la limitación que solo puede abrir archivos que pesen menos de 4 MB, entonces en Python se tiene que dividir la imagen en 6 archivos, en mips se procesa la imagen con esos se archivos, se genera los binarios correspondiente a los kernel y se visualiza por medio de Python.

Para ejecutar los código se tiene un makefile .bat.

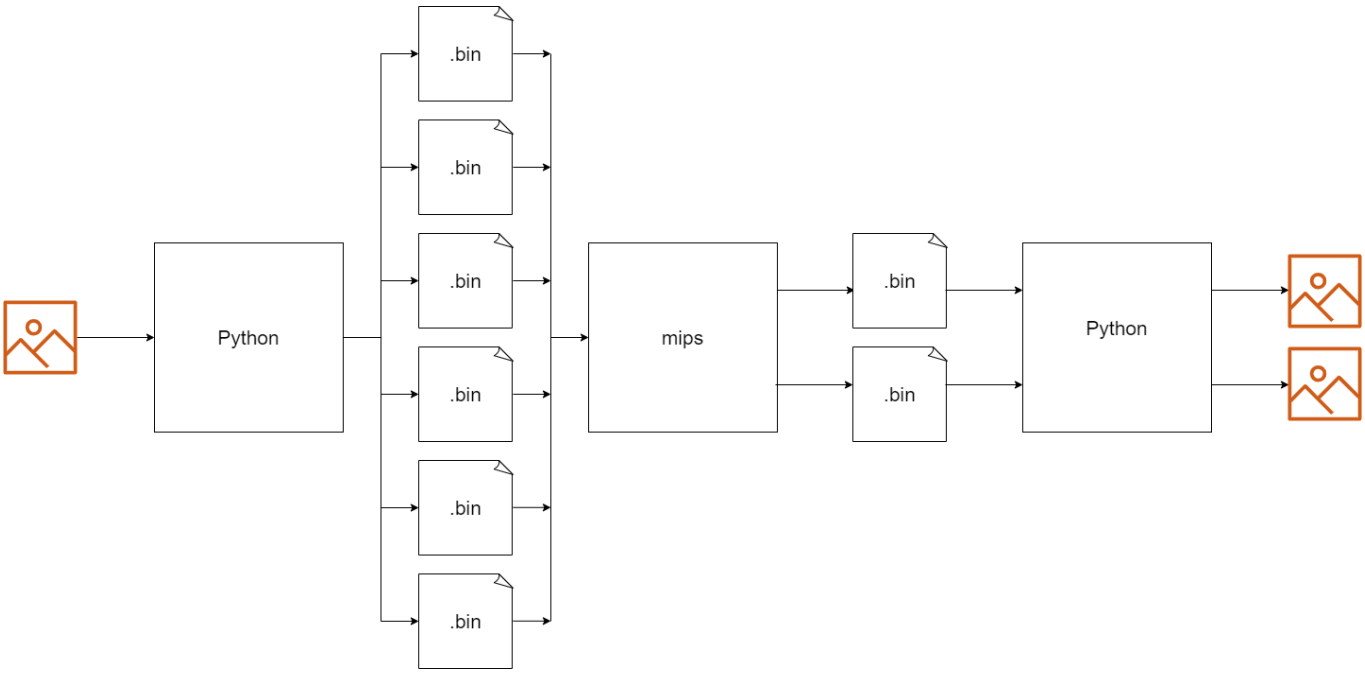

Figura 2: Diagrama de bloques propuesta 2.

3. Comparación de opciones de solución

Como en ambas propuestas el flujo de la imagen 3 es el mismo, se hará una comparación con respecto a la características de cada uno.

x86 es un ISA con arquitectura CISC por lo tanto tiene pocos registros, es más amigable con el programador, normalmente los simuladores que corren x86 no suelen presentar problemas con la

memoria, por lo que se puede manipular archivos de gran tamaño.

Por otro lado, mips es una arquitectura RISC, y las principales características que son menos instrucciones por lo que el peso recae en el programador, tiene muchos registros.

Por otro lado, el simulador escogido para mips tiene la limitante que no puede abrir archivos mas grandes de 4 MB, por lo que es una consideración a la hora de la implementación.

En ambas ISA se encuentra suficiente información en internet y en libros, ya que son arquitecturas muy conocidos por desarrolladores de bajo nivel, además, ambas se pueden programar en sistemas operativos de Windows y distribuciones de Linux.

Otra comparación entre ambos ISA, como se mencionó anteriormente, el tamaño que puede tener un archivo para ser abierto, aunque esto recae en el simulador que se desea desarrollar, pero en el caso de mips se tiene que hacer una manipulación un poco mas exhaustiva de archivos, además, el lenguaje de alto nivel que pasa la imagen a binario tiene que tener cierta lógica para guardar en los archivos, aunque esto no es un impedimento la idea principal era que los programas de alto nivel tuvieran poca de inteligencia, es decir, que solo se pasar de imagen a archivo binario y de archivo binario a imagen.

Para desarrollar en mips el simulador solo ocupa tener instalado java y descargar el simulador Mars, ya que mars fue desarrollado en ese entorno, pero para desarrollar en x86 se tiene que preparar el entorno ya sea usando masm en windows o nasm para distribuciones de linux.

4. Selección de la propuesta final

Para esta sección se va a explicar el diagrama de flujo de la imagen 3 que ha sido mencionado varias veces, este diagrama sería para la parte de ensamblador.

Primero se tiene que ingresar en ancho y el alto de la imagen, se inicializan los contadores de ancho y altura en 1, se inicia el contador del archivo en 0, se inicia la bandera en 1, se obtiene el alto de la imagen, se separa 8 espacios del heap, se abre el primer archivo, se abre los archivos donde se va a guardar los resultados de la convolución y se abre el archivo de lectura.

Se verifica si el contador del alto de la imagen es igual es alto + 1 si es así se termina el programa, si no, se evalúa el contador del alto con 1 y con el alto de la imagen.

Para los casos anteriores se compara si el contador del ancho es 1 en caso de ser positivo se suma 1 al contador del ancho y 8 al contador del archivo, en casi que el contado del ancho no es igual al ancho entonces también se le suma un 1 al contador del ancho y 8 al contador del archivo.

En caso que el contador del ancho sea igual al ancho se suma un 1 al contador de altura, el contador del ancho se ponen en 1 y se suma 8 al contador del archivo.

Cuando se incrementaron los contadores, se obtiene la submatriz, cuando la bandera es 1 se realiza la convolución sharperned se guarda en el archivo sharperned, cuando la bandera es 2 se hace

la convolución oversharperned y se guarda en el archivo correspondiente.

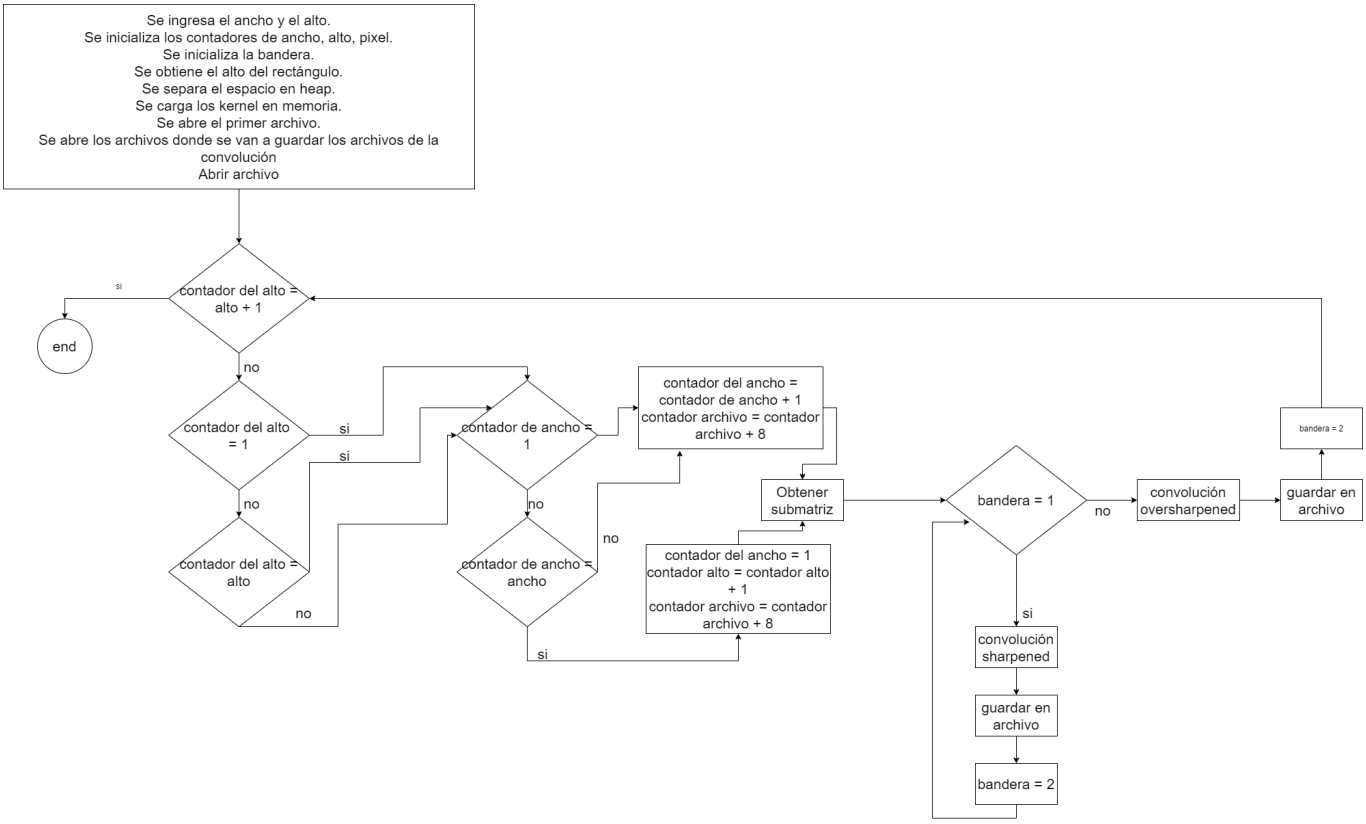

Figura 3: Diagrama de flujo del proyecto.

El diagrama explicado anteriormente no afecta a la propuesta se seccionada, ya que esta pensada en como recorrer la imagen y no en en ISA seleccionado.

Por lo tanto, se escogió la segunda propuesta basándose en que mips solo se necesita tener instalado java y descargar el emulador, por ende, se ocupa menos tiempo para preparar el entorno de trabajo.

Se trato de evitar preparar un entorno de trabajo, ya que para desarrollar la implantación se cuanta con 4G de RAM, se quiere evitar que el entorno pueda llegar a ocupar mas de eso, por lo tanto se prefirió escoger mips.

Por otro lado, se tiene experiencia previa programando en mips, por lo que la curva de aprendizaje era menor en comparación con x86.

5. Referencias

Referencias

[1] Giménez-Palomares, F., Monsoriu, J., Alemany-Martínez, E. (2016). Aplicación de la convolución de matrices al filtrado de imágenes. Riunet.upv.es. Retrieved 28 May 2020, from

https://riunet.upv.es/bitstream/handle/10251/69639/4524-15767-1-PB.pdf?sequence=1.