Desarrollo de una aplicación para aplicar interpolación bilineal sobre un cuadrante de una imagen utilizando NASM

Julio Varela Venegas
Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
2019008041

I. Listado de requerimientos del sistema

El proyecto se centra en el desarrollo de una aplicación que permita aplicar un algoritmo de interpolación bilineal a un cuadrante específico de una imagen, además de esto la implementación requiere el funcionamiento en conjunto de un programa desarrollado en alto nivel para la interfaz visual de la aplicación y un programa en ensamblador para el procesamiento directo de la imagen en cuanto a la aplicación del algoritmo de interpolación bilineal. Se requiere desarrollar una interfaz visual capaz de seleccionar una imagen con una resolución estricta de 400x400 píxeles para aplicarle una división a esta en 16 partes que podrían ser seleccionadas posteriormente para el procesamiento dentro del programa desarrollado en ensamblador, estas serían de una resolución de 100x100 píxeles y finalmente serían proyectadas dentro de la misma interfaz en el programa de alto nivel para contrastar el resultado final con el cuadrante antes del procesamiento. Para ello, se plantearon los siguientes requerimientos:

Requerimientos identificados

- **Automatización de la aplicación:** Sobre el programa se deben ejecutar todas las partes en conjunto paso a paso de modo que en primera instancia se pueda seleccionar la imagen, para luego aplicar una cuadricula a esta y poder escoger el cuadrante que se procesará con el algoritmo de interpolación desarrollado en ensamblador y por último mostrar el resultado del cudrante procesado de la imagen original sobre la misma interfaz de usuario.
- **Diseño de una interfaz visual agradable al usuario:** Es necesario el desarrollo de una interfaz visual capaz de mostrar al usuario cada uno de los pasos que se están procesando a nivel de código y lo que la aplicación es capaz de soportar, además de implementar las restricciones necesarias como por ejemplo que la imagen no puede tener una resolución distinta a la establecida o que se debe escoger un cuadrante obligatoriamente para proceder con la ejecución.
- **Procesamiento de la imagen en ensamblador:** La aplicación diseñada debe realizar realizar el procesamiento de la imagen con el algoritmo de interpolación bilineal para el cuadrante seleccionado en ensamblador, utilizando unicamente alguno de los siguientes ISAS: ARM,x86 o RISCV. Por lo tanto al momento de seleccionar el cuadrante a procesar se debe guardar la información de este con algún formato soportado por el programa desarrollado en ensamblador.
- **Procesamiento de la imagen en ensamblador:** La aplicación diseñada debe realizar el procesamiento de la imagen con el algoritmo de interpolación bilineal para el cuadrante seleccionado en ensamblador, utilizando únicamente alguno de los siguientes ISAS: ARM,x86 o RISCV. Por lo tanto al momento de seleccionar el cuadrante a procesar se debe guardar la información de este con algún formato soportado por el programa desarrollado en ensamblador.
- **Integración entre lenguajes:** Debe existir comunicación fluida entre el programa de alto nivel y el código ensamblador, permitiendo enviar datos y recibir resultados de manera controlada.
- **Uso de un depurador:** Se debe utilizar un depurador para el ISA seleccionado que permita percibir la información dentro de los registros y con esto corroborar que los valores sean los esperados una vez ejecutado el algoritmo de interpolación dentro del ensamblador.

Estado del arte

La interpolación bilineal es un algoritmo ampliamente utilizado en procesamiento de imágenes que permite escalar o transformar imágenes suavemente gracias al planteamiento matemático que lo sustenta. Sin embargo, su implementación en bajo nivel no es común, y suele realizarse mediante bibliotecas de alto nivel como OpenCV o PIL. Este proyecto explora el uso de ensamblador como medio para optimizar el procesamiento de imágenes, integrado con una interfaz visual agradable al usuario que le permita a este visualizar el proceso de aplicación del algoritmo a una imagen seleccionada.

Estándares y normas aplicables

- **ISO/IEC 9899**: Estandarización del lenguaje C, base para interoperabilidad con ensamblador.

- **IEEE 754**: Norma para aritmética de punto flotante.
- **ISO/IEC 15938**: Estándares MPEG para representación de contenido visual.

II. Opciones de solución al problema

Se presentan tres opciones de solución basadas en ensamblador, evaluadas con criterios técnicos y teóricos para garantizar su viabilidad.

Opción 1: Implementación híbrida con NASM y Python

Utilizar ensamblador x86 (NASM) para el algoritmo de interpolación bilineal, integrado con Python para la interfaz gráfica y manejo de imágenes.

- **Ventajas**:
  - Alto rendimiento en procesadores x86, ampliamente disponibles.
  - Herramientas maduras (NASM, GDB) facilitan desarrollo y depuración.
  - Integración directa con Python vía archivos de texto.
- **Desventajas**:
  - Dependencia de hardware x86, limitando portabilidad.
  - Mayor consumo energético en comparación con ARM o RISC-V.

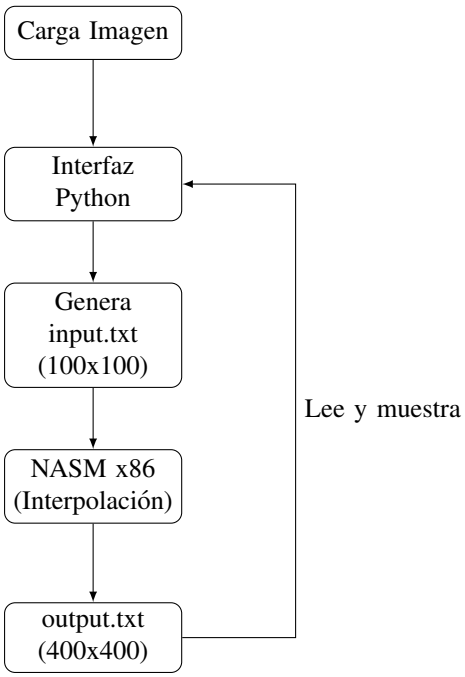

Figura 1: Flujo de NASM + Python

Opción 2: Implementación en ensamblador ARM con interfaz Python

Implementar la interpolación bilineal en ensamblador ARM, con Python para la interfaz y gestión de entrada/salida.

- **Ventajas**:
  - Eficiencia energética, ideal para dispositivos móviles o embebidos.
  - Soporte amplio en hardware moderno (smartphones, IoT).
  - Flexibilidad para optimizar registros y pipelines del ISA ARM.
- **Desventajas**:
  - Requiere emuladores o hardware ARM para pruebas en Ubuntu 18.04.
  - Integración con Python más compleja (bindings o CFFI).

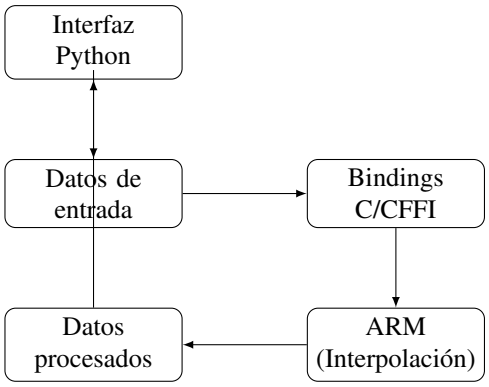

Figura 2: Flujo de ARM + Python

Opción 3: Implementación en ensamblador RISC-V con interfaz Python

Desarrollar el procesamiento en ensamblador RISC-V, con Python manejando la interfaz gráfica.

- **Ventajas**:
  - Arquitectura abierta, sin costos de licencias, promoviendo accesibilidad.
  - Diseño simple y modular del ISA, ideal para optimización educativa.
  - Potencial para hardware emergente de bajo costo.
- **Desventajas**:
  - Ecosistema de herramientas menos maduro en 2018 (entorno del proyecto).
  - Soporte limitado para depuración en Ubuntu 18.04.

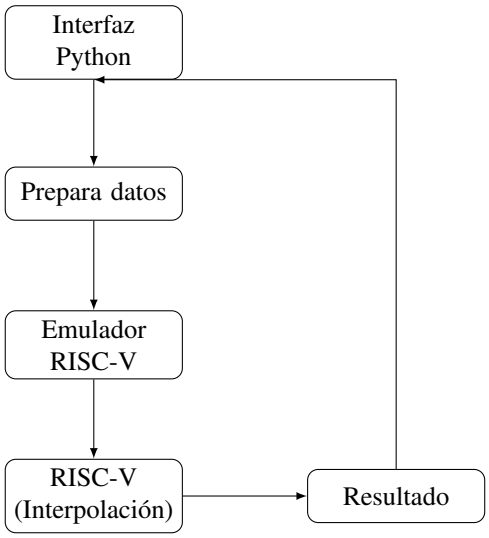

Figura 3: Flujo de RISC-V + Python

III. Comparación de opciones de solución

Las tres opciones se comparan objetivamente considerando los requerimientos técnicos del proyecto (procesamiento en ensamblador, integración con GUI) y aspectos adicionales como salud, seguridad, ambientales, económicos, culturales, sociales y estándares.

Criterio	NASM + Python
Rendimiento (Técnico)	Alto (x86 optimizado)
Complejidad de desarrollo	Alta (manual x86)
Manejo de memoria	Manual (x86 detallado)
Depuración	Alta (GDB maduro)
Interfaz gráfica	Directa (Python)
Salud y seguridad	Neutral (escritorio)
Impacto ambiental	Medio (mayor consumo)
Costo económico	Bajo (herramientas gratuitas)
Cultural/Social	Medio (x86 común)
Estándares	Alto (x86 establecido)
Portabilidad	Baja (x86 específico)

TABLE I
Comparación de NASM + Python.

Criterio	RISC-V + Python
Rendimiento (Técnico)	Medio (RISC-V emergente)
Complejidad de desarrollo	Muy Alta (herramientas nuevas)
Manejo de memoria	Manual (RISC-V simple)
Depuración	Baja (soporte limitado)
Interfaz gráfica	Indirecta (bindings)
Salud y seguridad	Alta (bajo voltaje)
Impacto ambiental	Alto (eficiencia potencial)
Costo económico	Bajo (sin licencias)
Cultural/Social	Alto (acceso abierto)
Estándares	Medio (RISC-V en desarrollo)
Portabilidad	Alta (RISC-V futuro)

TABLE II
Comparación de RISC-V + Python.

Análisis de criterios

- **Rendimiento**: NASM (x86) ofrece alta velocidad en hardware de escritorio, mientras que ARM y RISC-V priorizan eficiencia energética, con ARM teniendo una ventaja en pipelines optimizados.
- **Complejidad**: Todas las opciones son complejas por el uso de ensamblador, pero RISC-V añade dificultad por su ecosistema emergente.
- **Salud y seguridad**: ARM y RISC-V son más seguros en contextos embebidos (menor voltaje), mientras que x86 es neutral en escritorios.
- **Impacto ambiental**: ARM y RISC-V reducen consumo energético, siendo más sostenibles que x86 en hardware típico.
- **Costo económico**: NASM y RISC-V son de bajo costo (herramientas gratuitas), mientras que ARM puede requerir hardware específico.
- **Cultural/Social**: ARM domina en dispositivos móviles, RISC-V fomenta accesibilidad, y x86 es estándar en educación técnica.
- **Estándares**: x86 y ARM tienen normas bien definidas; RISC-V está en desarrollo pero promete apertura.

Criterio	ARM + Python
Rendimiento (Técnico)	Medio-Alto (ARM pipelines)
Complejidad de desarrollo	Alta (ARM específico)
Manejo de memoria	Manual (ARM eficiente)
Depuración	Media (herramientas ARM)
Interfaz gráfica	Indirecta (bindings)
Salud y seguridad	Alta (bajo voltaje)
Impacto ambiental	Alto (eficiencia energética)
Costo económico	Medio (hardware ARM)
Cultural/Social	Alto (ARM universal)
Estándares	Alto (ARM estandarizado)
Portabilidad	Alta (ARM amplio)

TABLE III
Comparación de ARM + Python.

IV. Selección de la propuesta final

La evaluación objetiva de las opciones consideró los requerimientos del proyecto (procesamiento en ensamblador, GUI funcional) y los criterios adicionales:

- **NASM + Python**:
  - *Ventajas clave*: Máximo rendimiento en Ubuntu 18.04, herramientas robustas (NASM, GDB), integración sencilla con Python.
  - *Limitaciones*: Menor portabilidad y mayor consumo energético.
  - *Alineación*: Ideal para entornos de escritorio disponibles, cumpliendo plenamente con el ISA x86 requerido.
- **ARM + Python**:
  - *Ventajas clave*: Eficiencia energética y portabilidad a dispositivos móviles.
  - *Limitaciones*: Complejidad de integración y necesidad de hardware/emuladores no disponibles en el entorno.
- **RISC-V + Python**:
  - *Ventajas clave*: Apertura y potencial futuro.

  - *Limitaciones*: Herramientas inmaduras y soporte limitado en 2018.

La solución **NASM + Python** fue seleccionada por:

- **Rendimiento técnico**: Procesa eficientemente imágenes de 100x100 a 400x400 en x86, aprovechando el hardware disponible.
- **Viabilidad práctica**: Herramientas accesibles en Ubuntu 18.04 (NASM, GDB, Python) y desarrollo alineado con estándares x86 establecidos.
- **Economía**: Sin costos adicionales en licencias o hardware.
- **Educación**: Cumple con el objetivo de aprendizaje en ensamblador, siendo x86 un ISA maduro y documentado.
- **Limitaciones aceptables**: La menor portabilidad y el impacto ambiental son secundarios frente a los requerimientos del proyecto (escritorio, funcionalidad inmediata).

La arquitectura final utiliza Python para cargar imágenes, seleccionar cuadrantes y visualizar resultados, mientras que NASM aplica la interpolación bilineal, optimizando el manejo de memoria y registros. Un `Makefile` asegura la compilación eficiente.

V. Conclusión

La solución híbrida NASM + Python no solo cumplió con los requisitos establecidos de implementar el procesamiento de imágenes mediante ensamblador x86, sino que también logró un equilibrio óptimo entre rendimiento, viabilidad práctica y valor educativo. La implementación permitió procesar eficientemente cuadrantes de imágenes de 100x100 píxeles, escalándolos a 400x400 mediante interpolación bilineal, con un rendimiento optimizado gracias al control detallado de registros y memoria que ofrece el ensamblador x86. La integración con Python, a través de una interfaz gráfica intuitiva desarrollada con Tkinter, facilitó la interacción del usuario, permitiendo cargar imágenes, seleccionar cuadrantes y visualizar los resultados de manera fluida y accesible.

El análisis comparativo, basado en criterios técnicos, ambientales, económicos y sociales, confirmó la superioridad de esta solución en el contexto del proyecto frente a las alternativas ARM y RISC-V. En particular, la disponibilidad de herramientas maduras como NASM y GDB en Ubuntu 18.04, junto con la compatibilidad inmediata con hardware x86, permitió un desarrollo y depuración más ágiles, superando las limitaciones de herramientas menos maduras para RISC-V y la necesidad de emuladores o hardware específico para ARM. Además, el proyecto demostró la viabilidad de combinar lenguajes de alto y bajo nivel, abriendo la puerta a aplicaciones más complejas donde el rendimiento crítico y la usabilidad deben coexistir.

Desde una perspectiva educativa, esta implementación proporcionó un aprendizaje profundo sobre el manejo manual de memoria, la optimización de algoritmos en ensamblador y la integración de sistemas heterogéneos. El uso de un `Makefile` para automatizar la compilación y la ejecución del código ensamblador también resaltó la importancia de flujos de trabajo eficientes en proyectos de desarrollo mixto.

Teniendo una perspectiva del futuro, esta solución sienta las bases para explorar otras arquitecturas como ARM o RISC-V, especialmente en contextos donde la eficiencia energética o la apertura del ISA sean prioritarias. Asimismo, el enfoque híbrido podría extenderse a otros algoritmos de procesamiento de imágenes, como filtrado o detección de bordes, o también a aplicaciones en tiempo real donde el rendimiento en bajo nivel sea crítico. En conclusión, este proyecto no solo alcanzó sus objetivos técnicos, sino que también contribuyó al entendimiento de las sinergias entre ensamblador y lenguajes de alto nivel, ofreciendo un modelo replicable para futuros desarrollos en el ámbito del procesamiento de imágenes y proyectos afines.

References

[1] Gonzalez, R. C., Woods, R. E., *Digital Image Processing*. Pearson, 2018.
[2] The Netwide Assembler (NASM) Documentation. [Online]. Available: https://www.nasm.us/doc/
[3] GDB: The GNU Project Debugger. [Online]. Available: https://www.gnu.org/software/gdb/
[4] Hyatt, R. E., *Programming in Assembly Language: A Practical Approach*. Wiley, 2010.
[5] Armstrong, J., *Low-Level Programming: C, Assembly, and Program Execution on Intel and ARM*. Apress, 2015.