

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería en Computadores
(Computer Engineering Academic Area)

Programa de Licenciatura en Ingeniería en Computadores
(Licentiate Degree Program in Computer Engineering)

Curso: CE-4302 Arquitectura de Computadores II
(Course: CE-4302 Computer Architecture II)



Clúster Beowulf para Algoritmo de Procesamiento de Imágenes
(Second Project)

Realizado por:

Made by:

Fabián Astorga Cerdas 2014040808

Javier Sancho Marín 2014159997

Óscar Ulate Alpízar 201229559

Profesor:

(Professor)

Jefferson González Gómez

Fecha: Cartago, 18 mayo, 2018

(Date: Cartago, May 18, 2018)

Diseño de Software

Esta sección detalla el software utilizado en el sistema del cluster Beowulf. Esto incluye la configuración del cluster, interfaz de paso de mensajes de MPICH y el algoritmo del filtro de Gauss utilizado para probar el funcionamiento del cluster.

Aplicación

La aplicación realizada para este proyecto es la de un filtro de difuminación de Gauss. El mismo consiste en la creación de una máscara, llamada kernel, que al convolucionar con cada píxel de la imagen, genera el efecto de difuminado deseado. El kernel, también conocido como matriz de convolución, es calculado por medio de (1).

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

donde x y y hacen referencia a la posición del píxel en la imagen y la letra griega *sigma* (σ) es la desviación estándar de la distribución de Gauss.

Una vez calculado el kernel, se convoluciona el mismo con cada píxel de la imagen. Mientras más grande sea la matriz de convolución, más preciso es el filtro, pero requiere un mayor procesamiento para ser completado.



Figura 1. Ejemplo de filtro de difuminación de Gauss con un kernel de tamaño 15, un sigma de 10 y con efecto “reflect”.

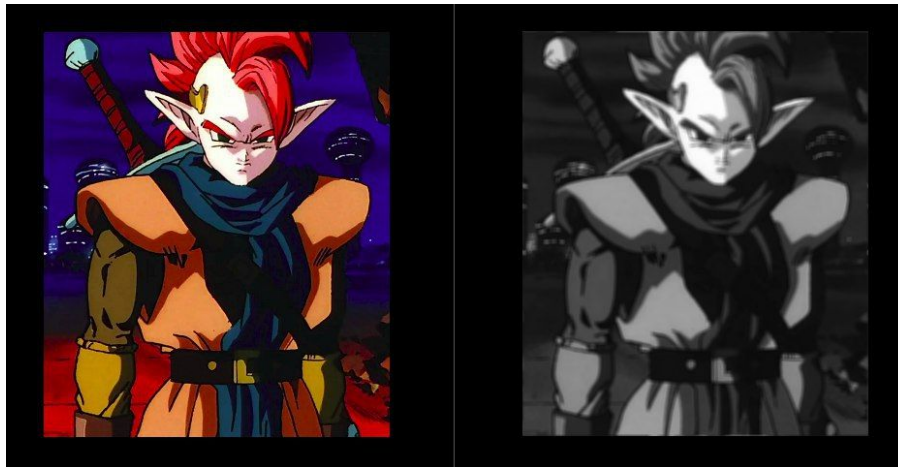


Figura 2. Ejemplo de filtro de difuminación de Gauss con un kernel de tamaño 5, un sigma de 8 y con efecto “circular”.

El efecto “*reflect*” hace referencia a que los bordes de la imagen, utilizan su pixel opuesto (espejo) para realizar el algoritmo. El efecto “*circular*” utilizado en el segundo ejemplo es otra manera de utilizar los píxeles para manejar los bordes.

Diagrama UML

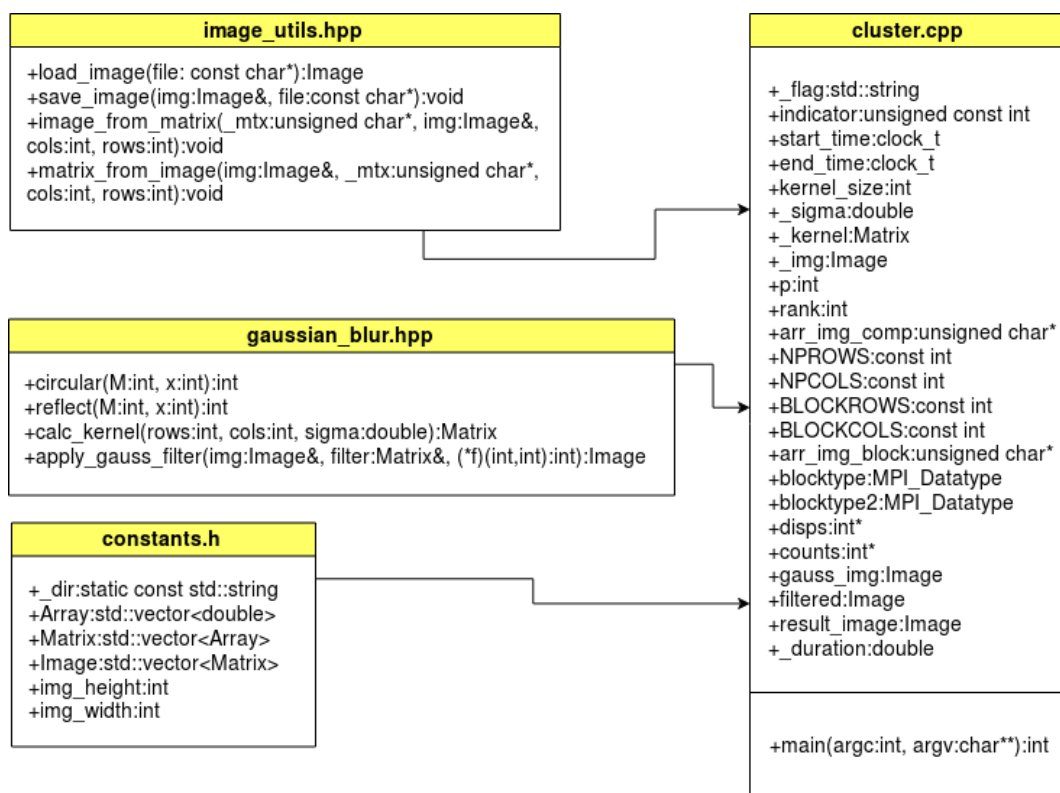


Figura 3. Diagrama UML del sistema implementado

Descripción de funciones

El proyecto se conforma de 4 archivos con funcionalidades específicas:

constants.h: contiene los datos que se utilizan en el resto del proyecto, definiciones de datos constantes y estructuras de datos, como lo es la matrix para manejar las imágenes.

- Estructura de datos Image: por medio de la biblioteca vector se genera un vector de profundidad 3, donde la profundidad 1 representa los 3 canales de la imagen (constante para cualquier imagen), la profundidad 2 representa la cantidad de filas de la imagen, y la profundidad 3 representa las columnas.
- Estructura de datos matriz: se utiliza para la manipulación de los datos con la biblioteca mpich, es un arreglo de datos unsigned char de “2 dimensiones”, que se accede por medio de la manipulación de un índice de la forma: $j+cols*i$, donde i y j son variables para recorrer la matriz y $cols$ la cantidad de columnas de la imagen.

image_utils.hpp: posee todas las funcionalidades correspondientes al manejo de imágenes.

- **load_image**: carga la imagen del directorio `\img` dentro de la estructura de documentos del proyecto, el nombre de la imagen es introducida como parámetro por medio de la línea de comandos.
- **save_image**: guarda la imagen procesada en el directorio `\img` dentro de la estructura de documentos del proyecto, el nombre de la imagen de salida es introducida por el usuario como parámetro por medio de la línea de comandos.
- **image_from_matrix**: recibe como parámetro una estructura de datos matriz y una estructura de datos imagen y copia los datos de la matriz a la imagen. Ambos parámetros ingresan por referencia para no realizar un retorno en el método.
- **matrix_from_image**: recibe como parámetro una estructura de datos matriz y una estructura de datos imagen y copia los datos de la imagen a la matriz. Ambos parámetros ingresan por referencia para no realizar un retorno en el método.

gaussian_blur.hpp: implementación de los métodos que hacen posible el cálculo del filtro gaussian blur.

- **circular:** función auxiliar para el cálculo de la convolución de los bordes de la imagen con el kernel. Es un método de indexación circular, cuando llega al borde de una imagen y requiere de un dato aledaño obtiene el índice circular correspondiente para finalizar la operación sobre el píxel específico.
- **reflect:** función auxiliar para el cálculo de la convolución de los bordes de la imagen con el kernel. Es un método de indexación de espejo, cuando llega al borde de una imagen y requiere de un dato aledaño obtiene el índice espejo correspondiente para finalizar la operación sobre el píxel específico.
- **calc_kernel:** Obtiene los valores respectivos del kernel gaussiano con respecto a los valores de sigma y tamaño del kernel ingresados por el usuario, se utiliza (1) para el cálculo del respectivo kernel.
- **apply_gaussian_filter:** Retorna el resultado de la convolución de la imagen original con el kernel gaussiano obtenido anteriormente.

cluster.cpp: Este fichero contiene la función principal de la aplicación donde realiza la instanciación de la imagen y la llamada de los respectivos métodos para la funcionalidad del sistema.

- **main:** Procesa adecuadamente los parámetros ingresados por el usuario para la configuración, división y sincronización del trabajo del cluster Beowulf.

Descripción de bibliotecas

Las bibliotecas utilizadas para este proyecto están escritas en el lenguaje de programación C y son las siguientes:

- **stdlib.h:** Esta es la biblioteca estándar de C y contiene prototipos de funciones de C para gestión de memoria dinámica y control de procesos, entre otros.
- **stdio.h:** Esta biblioteca funciona para proveer soporte de entrada y salida estándar a la aplicación.
- **string.h:** Esta biblioteca contiene soporte para operaciones con strings en C.
- **png.h:** Esta biblioteca funciona para la manipulación de imágenes en C, las funcionalidades utilizadas de esta biblioteca fueron las de cargar y guardar una imagen.
- **cmath.h:** declara un conjunto de funciones para procesar operaciones matemáticas comunes.
- **ctime.h:** Contiene definiciones de funciones para obtener y manipular información sobre fechas y tiempo.

- **Mpich:** Mpich es una implementación portable y de alto rendimiento de la interfaz MPI (Message Passing Interface) estándar. Dicha implementación es utilizada en nueve de las diez supercomputadoras con mejor desempeño a nivel mundial. Según sus desarrolladores, los objetivos de MPICH son: proporcionar una implementación de MPI que soporte eficientemente diferentes plataformas de computación y comunicación incluyendo clusters (sistemas de escritorio, sistemas de memoria compartida, arquitecturas multinúcleo), redes de alta velocidad (10 Gigabit Ethernet, InfiniBand, Myrinet , Quadrics) y sistemas de computación de gama alta patentados (como Blue Gene o Cray). Así como permitir la investigación de vanguardia en MPI a través de un marco modular fácil de extender para otras implementaciones derivadas. En la siguiente figura se puede observar la arquitectura a grandes rasgos de mpich: [1]

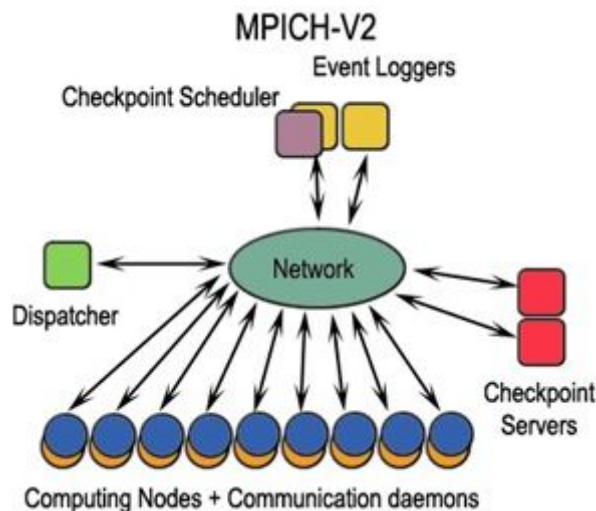


Figura 4. Arquitectura general de MPICH.

Diseño del sistema

En esta sección se detalla la solución del proyecto del punto de vista de arquitectura del sistema. Esto incluye la configuración del cluster Beowulf en las conexiones de red internas y de red y la distribución de carga computacional entre los nodos del cluster. En resumen cluster es una colección de computadores de escritorio o servidores conectados por una red de área local para ser utilizados como un solo computador. En el presente proyecto se utilizó una configuración de cuatro nodos, en la cual uno actúa como el nodo maestro, es decir, dicho nodo realiza la distribución de la carga hacia los demás nodos. La configuración del cluster en cuestión se observa en la siguiente figura:

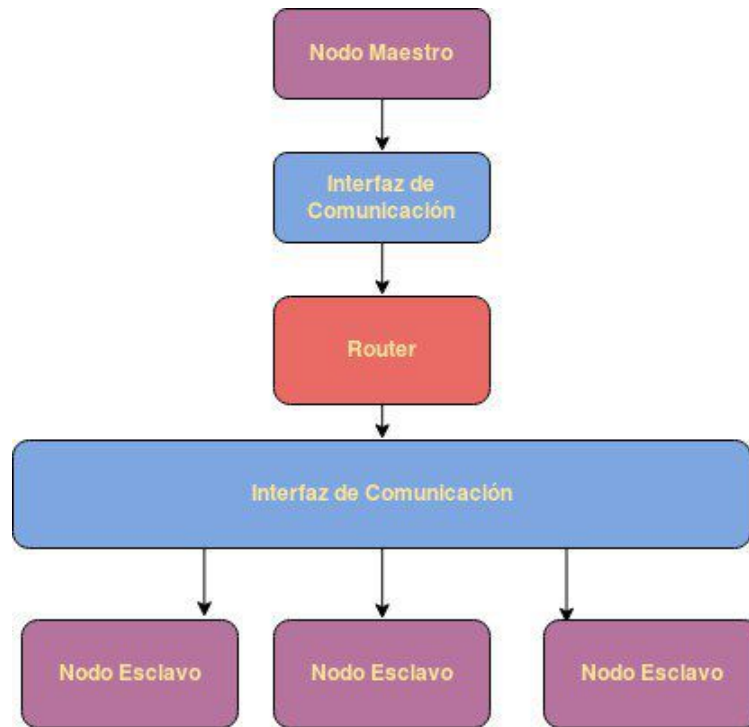


Figura 5. Diagrama de flujo de datos del sistema desarrollado.

Captura de requerimientos

En esta sección se enlistan los requerimientos para el proyecto de Arquitectura de Computadores II. Cada requerimiento incluye un identificador único.

Requerimiento R1: Implementación funcional de cluster Beowulf. Producto esperado: Sistema con funcionalidad demostrable.

Requerimiento R2: Implementación de algoritmo seleccionado en nodo maestro (corrida simple). Producto esperado: Resultados de tiempos de ejecución, sección de resultados.

Requerimiento R3: Implementación paralela de algoritmo seleccionado, por medio de una interfaz de paso de mensajes, entre los 4 nodos del clúster, con al menos 5 configuraciones distintas (1,2,3 y 4 nodos, con N cantidad de núcleos por nodo). Producto esperado: Resultados de tiempo de ejecución, ancho de banda y cualquier otra variable que sea relevante sección de resultados.

R1: Completado.

R2: Completado.

R3: Completado.

Metodología de diseño del Sistema

Análisis del problema

Las tecnologías de la computación avanzan a un ritmo bastante rápido. No es de extrañarse que las computadoras queden obsoletas en un tiempo relativamente corto. Por esta razón, se plantea la tarea de realizar clusters. Estos permiten crear un sistema heterogéneo de computadoras que pueden procesar de manera paralela algoritmos, que por sí solos, tomarían mucho tiempo.

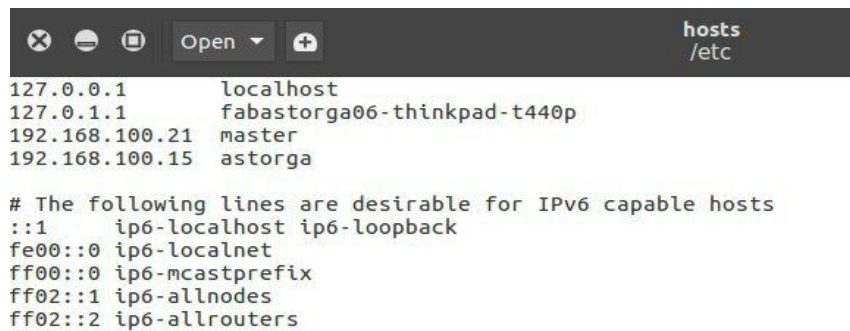
Un cluster Beowulf permite tomar entonces, componentes con prestaciones modestas para crear un supercomputador con la capacidad de lograr altos niveles de procesamiento. Este acercamiento permite no solo tener más núcleos de procesamiento, sino que potencialmente puede aumentar la memoria del sistema, espacio en disco y ancho de banda. [2]

Cluster Beowulf

Un cluster Beowulf es una clase de computador masivamente paralelo de altas prestaciones principalmente construido a base de un cluster de componentes hardware estándar. Un Beowulf ejecuta un sistema operativo de libre distribución como Linux o FreeBSD, y se interconecta mediante una red privada de gran velocidad. Generalmente se compone de un grupo de PCs o estaciones de trabajo dedicados a ejecutar tareas que precisan una alta capacidad de cálculo. Los nodos en el cluster de computadoras no se hayan en los puestos de trabajo de los usuarios, sino que están totalmente dedicados a las tareas asignadas al cluster. Generalmente, el cluster se haya conectado al mundo exterior por un solo nodo. [3]

Para la implementación del cluster Beowulf nos basamos en la guía realizada por Serrano Pereira [4]. El primer paso que se menciona es configurar cada uno de los nodos. Para esto se debe primero editar el archivo de anfitriones (*host file*) para que cada nodo recuerde al master y viceversa.

```
~$ sudo vim /etc/hosts
```

```
127.0.0.1      localhost
127.0.1.1      fabastorga06-thinkpad-t440p
192.168.100.21 master
192.168.100.15 astorga

# The following lines are desirable for IPv6 capable hosts
::1          ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
```

Figura 6. Archivo de hosts configurado para pruebas del proyecto desde uno de los nodos.

Como segundo paso, se crea el mismo usuario en todas las máquinas, no es estrictamente necesario que el usuario sea el mismo para todos los nodos, pero facilita el trabajo que todos tengan el mismo usuario, con el mismo nombre e identificador. El mismo se crea de manera simple con el siguiente comando:

```
~$ sudo adduser mpiuser --uid 998
```

El *id* del usuario puede ser cualquier número, mientras que sea el mismo en cada nodo. En el proyecto se escoge el 998 de manera arbitraria.

Una vez configurados todos los nodos, se debe configurar el *Filesystem*. Esto es necesario para poder compartir carpetas y archivos a través de la red local. El sistema de archivos NFS (*Network File Systems*) fue el escogido porque permite montar parte de un sistema de archivos remoto para que otros nodos puedan acceder a los documentos como si fueran un directorio local. Para instalarlo se utiliza el siguiente comando (solo en el nodo master):

```
~$ sudo apt-get install nfs-kernel-server
```

Para cada uno de los nodos se debe instalar el paquete *nfs-common* para que sea posible montar el sistema de archivos:

```
~$ sudo apt-get install nfs-common
```

Ahora en el archivo `/etc/exports` se debe agregar la siguiente línea para exportar el sistema de archivos a los demás nodos (solo en el master).

```
/home/mpiuser *(rw, sync, no_subtree_check)
```

Posteriormente, el nodo *master* debe permitir el acceso de los nodos clientes por medio de la red porque el firewall por defecto bloquea todos los accesos. Para esto el *master* debe abrir el firewall para estos accesos de un subnet específico con el siguiente comando:

```
~$ sudo ufw allow from q92.168.100.0/24
```

Esa IP que aparece en el comando, es la utilizada para todas las pruebas realizadas para el sistema. Una vez realizada la configuración, cada nodo puede entonces montar el sistema de archivos mencionado. Esto se debe realizar en cada nodo con el siguiente formato:

```
~$ sudo mount master:/home/mpiuser /home/mpiuser
```

Ya con el sistemas de archivos funcionando, se debe configurar el Secure Shell (SSH) para la comunicación entre los nodos. Cuando esto se configura el nodo maestro ya puede comunicarse con los demás nodos. Esto es necesario, según indica la guía, para que el *daemon* de MPI puedan correr en los nodos.

Primero se debe instalar SSH:

```
~$ sudo apt-get install ssh
```

Ahora se debe iniciar sesión con el usuario **mpiuser** creado anteriormente para crear la clave de acceso. Esto se debe realizar en todos los nodos. Se recomienda configurar el keygen sin contraseña para que sea sencillo acceder:

```
~$ su mpiuser
```

```
~$ ssh-keygen
```

Para que todos los nodos tengan la clave copiada en la carpeta compartida, el usuario **mpiuser** puede copiar la contraseña en la carpeta compartida:

```
~$ ssh-copy-id localhost
```

El último paso de configuración del cluster es el de configurar el *controlador de procesos*. Para esto se instala MPICH en todos los nodos:

```
~$ sudo apt-get install mpich
```

El controlador de procesos utilizados es HYDRA. Para configurar este, simplemente es necesario crear un archivo en la carpeta compartida del usuario **mpiuser** que contenga el nombre de cada uno de los nodos que vayan a ser parte del cluster.

Una vez concluidos los pasos anteriores, ya el cluster está completamente configurado. Para ejecutar el programa solo se debe ahora ejecutar con el siguiente comando:

```
Cluster_Beowulf$ mpiexec -n 4 -f hosts ./cluster {kernel} {sigma} ejemplo.png  
salida.png
```

Donde:

1. -n: Bandera que indica la cantidad de procesos en la que se debe dividir el procesamiento.
2. -f: Archivo del cuál se leen los nombres de los nodos que van a participar en el cluster.
3. ./cluster: Aplicación que va a ser ejecutada.
4. {kernel}: Valor entero impar que indica el tamaño del filtro que se le va a aplicar a la imagen.
5. {sigma}: Valor entero que es parámetro de la función de creación del filtro kernel.
6. ejemplo.png: Nombre de la imagen a la que se le desea aplicar el filtro.
7. salida.png: Nombre de la imagen de salida con el filtro ya aplicado.

Algoritmos de procesamiento de imágenes

El procesamiento o tratamiento digital de imágenes consiste en procesos algorítmicos que transforman una imagen en otra en donde se resalta cierta información de interés, y/o se atenúa o elimina información irrelevante para la aplicación. Así, las tareas del procesamiento de imágenes comprenden la supresión de ruido, mejoramientos de contraste, eliminación de efectos no deseados en

la captura como difuminaciones o distorsiones por efectos ópticos o de movimiento, mapeos geométricos, transformaciones de color, etc.

1. Vecindades

Las distancias entre las posiciones de un pixel a otros pixeles se puede observar como la vecindad de un pixel, la distancia no es más que un operador entre los componentes espaciales de la matrix que constituye la imagen. Una de las métricas en las que se basa dicho operador son las métricas de Minkowsky [5].

$$L_p(\bar{x}, \bar{y}) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (2)$$

En la fórmula anterior se define L2 como la distancia euclidiana y L1 la distancia de cuerdas de ciudad.

Las transformaciones de distancias utilizadas en imágenes se basan en esta operación de vecindades. La transformación de distancia es un operador normalmente solo aplicado a imágenes binarias. El resultado de la transformación es una imagen con niveles de grises similar a la imagen de entrada, excepto que la intensidad de puntos de escala de grises dentro de las regiones de primer plano están cambiadas para mostrar la distancia al límite más cercano de cada punto [6].

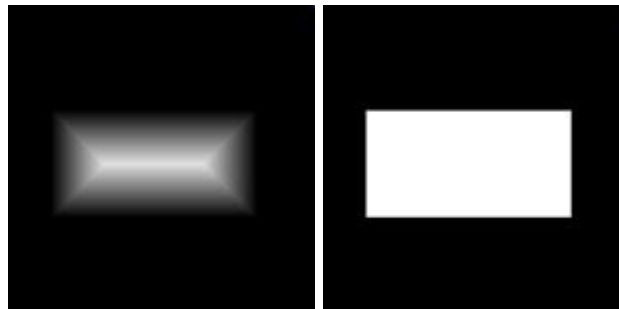


Figura 7. Ejemplo de aplicación de una transformación de distancia [7]

2. Técnicas en el dominio del espacio

Estas técnicas tratan directamente con los pixeles de la imagen. Los valores son manipulados de tal forma que se alcance una mejora deseada. Las técnicas de dominio en el espacio como las transformaciones logarítmicas, histogramas de ecualización, están basados en la manipulación directa de los píxeles de la imagen. Estas técnicas son particularmente útiles para alterar los valores de los píxeles individuales a un nivel de grises y por lo tanto el contraste general de toda la imagen. Sin

embargo, usualmente mejoran la imagen entera en una manera uniforme lo cual en algunos casos produce un resultado indeseable. No es posible de manera selectiva mejorar los bordes u otra información eficientemente con estas técnicas.[8]

Una de las técnicas utilizadas son las convoluciones de imágenes. Los filtros de convolución son usados para modificar la frecuencia de las características espaciales de una imagen.

La convolución un filtro de efectos de propósito general para imágenes. Se constituye de una matriz aplicada a una imagen para operaciones matemáticas, compuestas por enteros y números de punto flotante. Funcional al determinar el valor central de un píxel al combinar y realizar alguna operación sobre los píxeles vecinos. El resultado de esta modificación de píxeles es la imagen filtrada. [7]

La convolución es realidad al multiplicar los colores de un píxel y sus vecinos por una matriz. Un Kernel es usualmente una matriz de números que se utiliza en convolución de imágenes. Tamaños diferentes de kernels, conteniendo diferentes patrones de números, producen resultados diferentes bajo la convolución. [7]

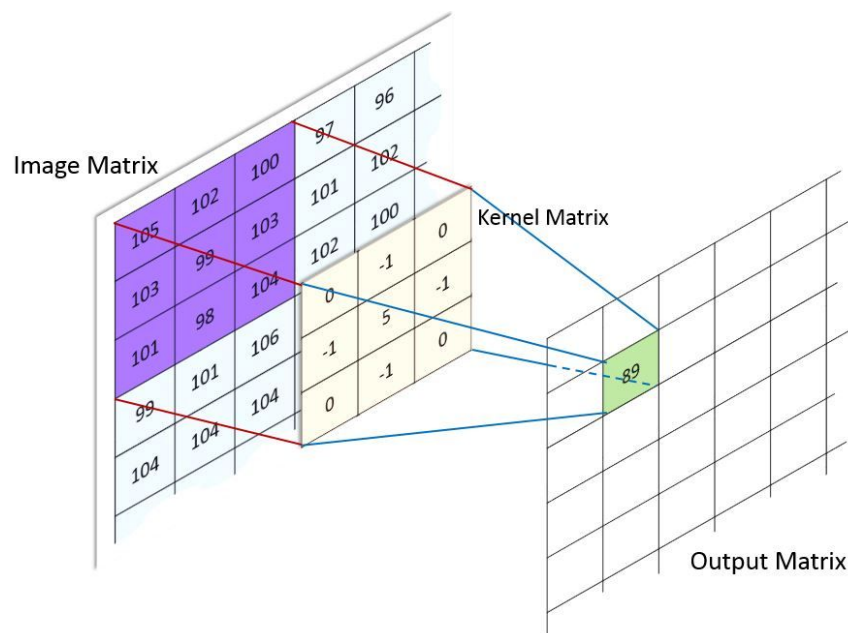


Figura 8. Ejemplo de una convolución de una imagen con su respectivo kernel.

3. Transformaciones de dominio

Las transformaciones de dominio están basadas en la manipulación de la transformada ortogonal de la imagen más no de la imagen como tal. Las transformaciones de dominio están creadas a la medida

para procesar la imagen de acuerdo a su contenido en la frecuencia. El principio detrás de los métodos de mejora de imagen en el dominio de la frecuencia consiste en la computación de una transformada discreta unitaria de 2D de la imagen, por ejemplo la transformada discreta de Fourier de 2D (DFT), manipula los coeficientes de la transformada por medio de un operador y luego realiza la transformada inversa. La transformada ortogonal de la imagen posee dos componentes, magnitud y fase. La magnitud consiste en el contenido en la frecuencia de la imagen y la fase es utilizada para restaurar la imagen de vuelta al dominio en el espacio. Las transformadas ortogonales comunes son transformada discreta cosenoidal, la transformada discreta de Fourier, la transformada de Hartley. La transformada en el dominio permite operar el contenido de la imagen en la frecuencia, con lo cual se puede mejorar fácilmente información de la imagen como los bordes o contenido de altas frecuencias [9].

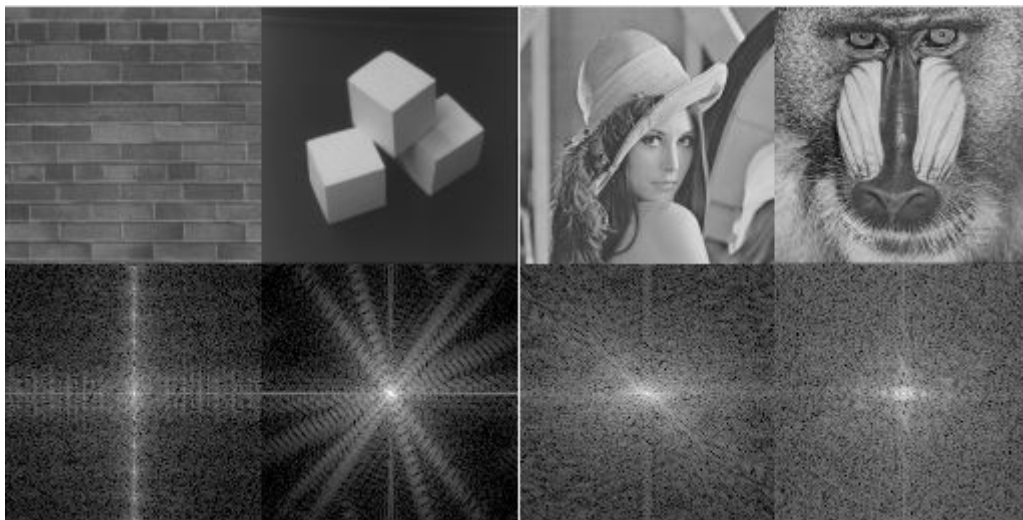


Figura 9. Resultado de la transformada discreta de Fourier sobre algunas imágenes.

Comparación y evaluación de propuestas

Las vecindades en comparación con los otros métodos de procesamiento de imágenes, no es tan eficiente a la hora de paralelizar ya que al dividir la imagen en varias porciones más pequeñas se pierde información. Esto sucede porque los píxeles de los nuevos bordes de las sub imágenes generadas aplican la operación con la información existente, es decir, al existir nuevos bordes la información de la imagen original no está dentro de cada sub imagen para realizar el cálculo adecuado y obtener un resultado coherente y preciso.

Las transformaciones de dominio a pesar de que permiten la detección de información en la imagen que no se puede observar en el dominio de espacio, requiere de operaciones matemáticas complejas para pasar de un dominio a otro y realizar su inversa también, y la imagen original se puede reconvertir únicamente si la transformada es reversible. La interpretación de la información en otros dominios puede ser malinterpretarse si no se posee suficiente conocimiento sobre el dominio al que se desea pasar la imagen.

Las técnicas de manipulación de la información en el dominio del espacio no requieren una conversión de dominio, lo que disminuye en cierta parte la pérdida de información de la imagen original, al manipular directamente la imagen se reduce este riesgo. La mejora deseada es considerablemente uniforme para toda la imagen, lo cual es deseado.

En el proyecto se utilizó una técnica de procesamiento en el espacio, más específicamente convolución de imágenes con el filtro de difuminación gaussiana, la principal razón se debe a que no se debe realizar ninguna transformación a la información de la imagen, si se desea una mejora deseada es más manejable obtenerla ya que se manipula directamente los píxeles de la imagen original. Otra razón es por la manera en que se puede paralelizar el procesamiento de la imagen, al subdividir la imagen en porciones más pequeñas se le puede asignar un proceso a cada subimagen y luego realizar una sincronización de los datos de manera uniforme para obtener la imagen filtrada.

Referencias bibliográficas

- [1] Mpich.org. (2017). MPICH | High-Performance Portable MPI . [online] Available at: <https://www.mpich.org/> [Accessed 17 May. 2018].
- [2] Gropp, W., Lusk, E. and Sterling, T. (2003). Beowulf cluster computing with Linux. 2nd ed. Cambridge, Mass.: MIT.
- [3] Personals.ac.upc.edu. (n.d.). *Creación de un Beowulf*. [online] Disponible en: <http://personals.ac.upc.edu/enric/PFC/Beowulf/beowulf.html> [Accesado el 12 de mayo, 2018].
- [4] Pereira, S. (2013). Building a simple Beowulf cluster with Ubuntu. [online] [Www-users.cs.york.ac.uk](http://www-users.cs.york.ac.uk). Disponible en: https://www-users.cs.york.ac.uk/~mjf/pi_cluster/src/Building_a_simple_Beowulf_cluster.html [Accesado el 12 de mayo, 2018].
- [5] Alvarado P. Notas de clase: Procesamiento y Análisis de Imágenes Digitales. Escuela de Electrónica. Tecnológico de Costa Rica. 2005-2012.
- [6] **Jain** *Fundamentals of Digital Image Processing*, Prentice-Hall, 1989, Chap. 9.

- [7] Henry Kwong. (2018). Distance Transforms. May 16, 2018, de Wolfram Demonstrations Project & Contributors Website: <http://demonstrations.wolfram.com/DistanceTransforms/>
- [8] International Journal of Engineering Science and Innovative Technology (IJESIT) Volumen 1, Issue 2, Noviembre 2012.
- [9] Ludwing, J. (n.d.). Image Convolution . 1st ed. [ebook] Portland State University.