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
(Curse: CE-4302 Computer Architecture II)

Documento de Diseño Proyecto #3
(Design Document Project #3)

Realizado por:
Made by:
José Daniel Badilla Umaña, 201271708
Jason Leitón Jiménez, 2014052303
Carlos Eduardo Peralta Coto, 201237200

Profesor:
(Professor)
Ing. Jeferson González Gómez, M.Sc

Fecha: Cartago, Noviembre 15, 2017
(Date: Cartago, November 15th, 2017)

Metodología de diseño de sistema

En la metodología de diseño para la implementación se utiliza el modelo de desarrollo en cascada, el cual permite manejar por etapas el proyecto. En este modelo se realiza un análisis de los requisitos, diseño del sistema, implementación, pruebas y mantenimiento. De esta forma se realiza la captura de requisitos del sistema con base en la especificación brindada para su diseño e implementación.

Durante la etapa de diseño del software se realiza la creación del clúster similar al clúster Beowulf, además se crea el diseño para el filtro de Sobel. Para luego, en la etapa de implementación tener una versión ejecutable del sistema, la cual, se le realizará pruebas y se dará mantenimiento en las siguientes etapas.

Análisis del problema

Como parte del crecimiento del paralelismo en los sistemas computacionales se han desarrollado grandes sistemas a escala, los cuales, poseen un nivel de computación alto debido a la unión de múltiples computadoras conectadas a través de una red local. El problema que existe en la actualidad, se ve en el crecimiento en la cantidad de datos que se necesitan procesar como en algunos casos de simulaciones, filtros de imágenes, entre otros. Estas ocupan de cálculos muy grandes y la problemática presente es que los tiempos de ejecución deben ser casi instantáneos.

Para la resolución de la problemática se pretende realizar un sistema distribuido, similar al clúster Beowulf para realizar un procesamiento basado en paso de mensajes, con el fin de realizar tareas específicas. Como parte del sistema distribuido se pretende que los tiempos de ejecución disminuyan, así obtener un mejor rendimiento en la ejecución de problemas.

Investigación respectiva

Cluster

Un cluster es un grupo de equipos independientes interconectados o unidos por una red de comunicación, que ejecutan una serie de aplicaciones de forma conjunta y aparecen ante clientes y aplicaciones como un solo sistema. Los clusters permiten aumentar la escalabilidad, disponibilidad y fiabilidad de múltiples niveles de red [1].

Cluster Beowulf

Este proyecto se basa en usar PVM y MPI, añadiendo algún programa más que se usan para monitorizar, realizar benchmarks y facilitar el manejo del cluster. Entre las posibilidades que integra este proyecto se encuentra la posibilidad de que algunos equipos

no necesiten discos duros, por eso se consideran que no son un cluster de estaciones de trabajo, sino que dicen que pueden introducir nodos heterogéneos. Esta posibilidad la da otro programa y Beowulf lo añade a su distribución. Beowulf puede verse como un empaquetado de PVM/MPI junto con más software[1].

Para realizar la comunicación de una manera más efectiva, se utilizan los siguientes pasos para evitar la introducción de la contraseña [2].

```
ssh-keygen -b 4096 -t rsa
ssh-copy-id root@ip
```

Con esta medida, ya es posible hacer la conexión con el ssh sin necesidad de ingresar la contraseña, y eso permite hacer el clúster beowulf.

Filtros

El realce de bordes transforma una imagen de manera que exhibe sólo el detalle de bordes o fronteras. Los bordes aparecen como las líneas de contorno de los objetos dentro de la imagen. Estos contornos pueden utilizarse en posteriores operaciones de análisis de imágenes para el reconocimiento de objetos o rasgos. Los realces de bordes son implementados a través de filtros espaciales. Los más utilizados son desplazamiento y sustracción, filtros Laplacianos para bordes, gradiente de Prewitt, Roberts, Sobel y Kirsch [3].

Propuestas de diseño

Filtro Sobel

El filtro de detección de bordes Sobel funciona haciendo un filtrado en el eje x, obteniendo el resultado F_x, haciendo un filtrado en el eje y y obteniendo el resultado F_y, y aplicando la siguiente ecuación:

$$S_{i,j} = \sqrt{F_x^2 + F_y^2} \quad (1)$$

Para obtener las imágenes F_x y F_y se utilizan los siguientes kernels [4]:

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Fig. 01 Kernels de Sobel para el filtrado vertical y horizontal

Para realizar de una manera más sencilla el procesamiento, a los kernels de la figura 01 se reflejan de manera vertical y horizontal, dando como resultado las siguientes máscaras:

1	2	1
0	0	0
-1	-2	-1

1	0	-1
2	0	-2
1	0	-1

Fig. 02 Kernels de Sobel para la convolución

Este filtro tiene la ventaja de que se puede paralelizar el procesamiento aplicando el filtro vertical de manera paralela al filtro horizontal.

Filtro Gaussiano

El filtro Gaussiano se utiliza para realizar un desenfoque que permite eliminar el ruido Gaussiano, mediante la convolución con una máscara que utiliza los valores de los píxeles vecinos para disminuir el efecto del ruido.

$^1/_{16}$	$^1/_8$	$^1/_{16}$
$^1/_8$	$^1/_4$	$^1/_8$
$^1/_{16}$	$^1/_8$	$^1/_{16}$

Fig. 03 Kernel de Gauss para la convolución

La máscara de la figura 03 se puede transformar a la siguiente máscara, en donde se realiza la multiplicación a cada por la fracción, pasando de 9 divisiones con 8 sumas a una división, 8 sumas y 5 multiplicaciones.

$^1/_{16}$ x

1	2	1
2	4	2
1	2	1

Fig. 04 Kernel de Gauss para la convolución

Filtro Prewitt

Este filtro es similar al filtro de Sobel, en donde se buscan resaltar los bordes. Las máscaras para la convolución en este caso son las siguientes [5]:

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Fig. 05 Kernels de Prewitt para el filtrado vertical y horizontal

Después de obtener las imágenes filtradas tanto de manera horizontal como de manera vertical, se aplica la misma ecuación (1) para obtener el valor final del pixel.

Comparación y evaluación de propuestas

Cantidad de operaciones

Para obtener el filtro que tenga la mayor cantidad de operaciones, se escriben las siguientes ecuaciones que implementarían cada uno de los filtros, para poder contar las operaciones:

- Filtro Sobel:

$$S(i,j) = \sqrt{(F_y)^2 + (F_x)^2}$$

$$F_x = P_{i-1,j-1} + 2 \times P_{i,j-1} + P_{i+1,j-1} - P_{i+1,j-1} - 2 \times P_{i+1,j} - P_{i+1,j+1}$$

$$F_y = P_{i-1,j-1} + 2 \times P_{i-1,j} + P_{i-1,j+1} - P_{i-1,j+1} - 2 \times P_{i,j+1} - P_{i+1,j+1}$$

- Filtro Gaussiano:

$$G(i,j) = \frac{1}{16} \times F_{i,j}$$

$$F_{i,j} = P_{i-1,j-1} + 2 \times P_{i-1,j} + P_{i-1,j+1} + 2 \times P_{i,j-1} + 4 \times P_{i,j} + 2 \times P_{i,j+1} + P_{i+1,j-1} + 2 \times P_{i+1,j} + P_{i+1,j+1}$$

- Filtro Prewitt:

$$S(i,j) = \sqrt{(F_y)^2 + (F_x)^2}$$

$$F_x = P_{i-1,j-1} + P_{i,j-1} + P_{i+1,j-1} - P_{i+1,j-1} - P_{i+1,j} - P_{i+1,j+1}$$

$$F_y = P_{i-1,j-1} + P_{i-1,j} + P_{i-1,j+1} - P_{i-1,j+1} - P_{i,j+1} - P_{i+1,j+1}$$

En resumen, se utilizan las siguientes operaciones:

- **Filtro Sobel**: Se utilizan 4 multiplicaciones, 6 sumas, y una raíz cuadrada.
- **Filtro Gaussiano**: Requiere 1 división, 8 sumas y 5 multiplicaciones.
- **Filtro Prewitt**: Necesita 2 multiplicaciones, 6 sumas y una raíz cuadrada.

Se puede observar que el filtro Sobel es el algoritmo que requiere una mayor cantidad de operaciones para su implementación, lo cual lo favorece para que sea el filtro a implementar.

Resultados de la imagen

Utilizando el programa de edición de imágenes GIMP, se pueden realizar los filtros a la imagen de Lena, mostrada en la figura 06.

Fig. 06 Imagen de Lena

(a) (b) (c)

Fig. 07 Imagen filtrada con GIMP

En la figura 07.a se muestra el resultado del filtro de detección de bordes Sobel, en la figura 07.b se muestra el resultado del filtro Prewitt y en la figura 07.c se muestra el resultado del desenfoque gaussiano con kernel de 3x3. Como se puede observar, el filtro de desenfoque Gaussiano no genera un cambio muy notable en la imagen, aunque sí disminuye el ruido. Los filtros de Sobel y Prewitt tienen un resultado mucho más notable, y por eso resultan una mejor opción para ser aplicados.

Selección de filtro

Debido a que se desea utilizar un filtro que requiera una gran cantidad de procesamiento, y que esté implementado de una manera correcta, fácil de verificar de manera visual, se decide utilizar el filtro de detección de bordes de Sobel.

Diseño de software

Diagramas UML

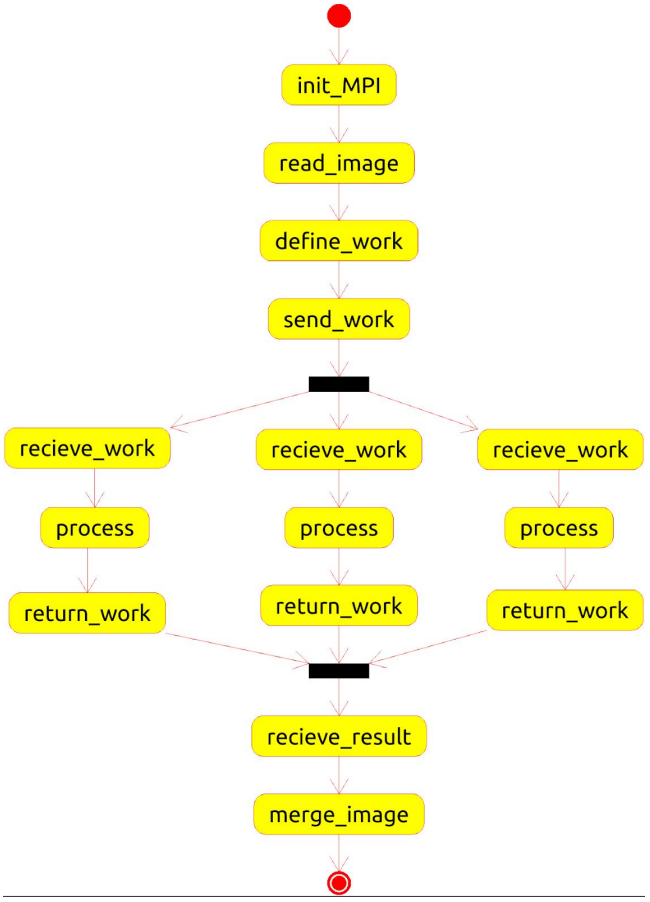


Fig. 08 Diagrama de actividades del programa

Se puede observar en la figura 08 el diagrama de actividades del programa, en donde se nota que existe un proceso que es el encargado de coordinar a los demás procesos, enviarles la información a procesar y recuperar la imagen luego. Este proceso es el maestro, que se encarga de la sincronización de los procesos.

Descripción de métodos

Para la implementación del filtro se realizó el código en C, utilizando MPI para la comunicación de los procesos entre los distintos procesos.

El código implementado funciona de la siguiente manera:

---

int main():

Declara las variables;
Inicializa MPI;
Calcula el tamaño del trabajo;
---------- Si es el proceso master ----------
Envía la imagen a los demás procesos;
Envía los valores que va a procesar cada uno;
---------- Si es otro proceso ----------
Recibe la imagen
Recibe los valores que va a procesar
---------- Todos ----------
Procesan su porción de imagen
---------- Si es distinto al master ----------
Envía los datos procesados al master
---------- Si es el master ----------
Recibe los datos procesados
Junta la imagen

Bibliotecas utilizadas

Libmath

Para el desarrollo del filtro se utiliza la biblioteca math de C, para importar los métodos que aplican tanto la raíz como la potencia al cuadrado.

Requisitos de software del sistema

- Implementación funcional de cluster Beowulf.
- Implementación de algoritmo seleccionado en nodo maestro.
- Implementación paralela del 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)

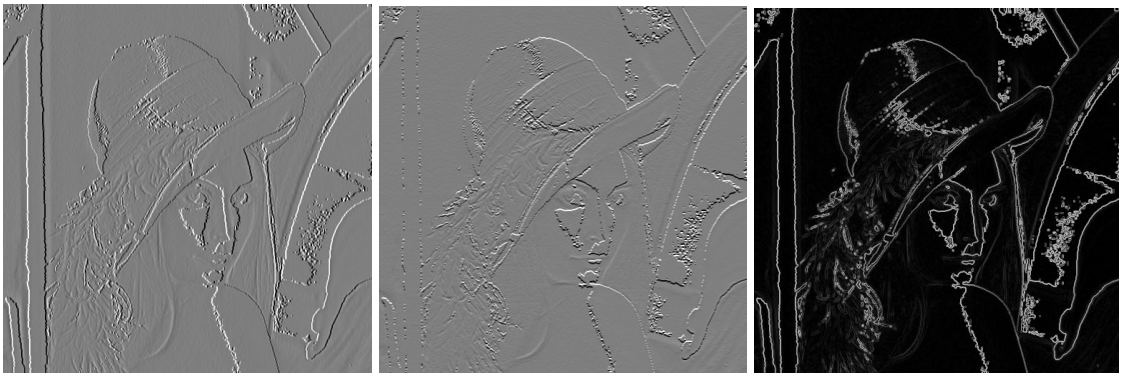
Cumplimiento de los requisitos de software

Requisito	**Descripción**	**Cumplimiento**
Implementación del Clúster	Creación del clúster con 4 nodos, siendo un nodo master y 3 slaves	Correcto
Implementación del algoritmo	Filtro de Sobel, para ambos ejes a escala de grises.	Correcto
Implementación paralela	Distribuir las tareas entre los nodos para realizar el filtro.	

Resultados obtenidos

Filtro

Para probar el filtro implementado se utilizó la figura 06, y se aplicó solamente el filtro en *x* (figura 09.a), luego solamente el filtro en *y* (figura 09.b), y por último el filtro en ambas direcciones (figura 09.c). Los resultados se pueden observar en la siguiente imagen:



(a) (b) (c)

Fig. 09 Resultados del filtro programado

Al realizar la ejecución del algoritmo de filtrado se determinó los siguiente resultados mostrados en la Tabla 1. Además, en la Figura 10 se demuestra los resultados con base en estos datos obtenidos.

Tiempos de ejecución

Núcleos	Nodos	Tiempo 1	Tiempo 2	Tiempo 3	Tiempo 4	Tiempo 5	Promedio Tiempo	Cantidad de Procesos
4	1	0,096613	0,096106	0,086550	0,105061	0,082690	0,093404	4
2	4	0,288897	0,299660	0,292081	0,284879	0,277868	0,288677	8
4	2	0,234883	0,228270	0,241864	0,228392	0,249918	0,236665	8
4	3	0,341563	0,336217	0,321561	0,370146	0,339014	0,341700	12
4	4	0,472564	0,515268	0,454254	0,440616	0,480080	0,472556	16

Tabla 1. Resultados obtenidos en la ejecución del filtro en el clúster.

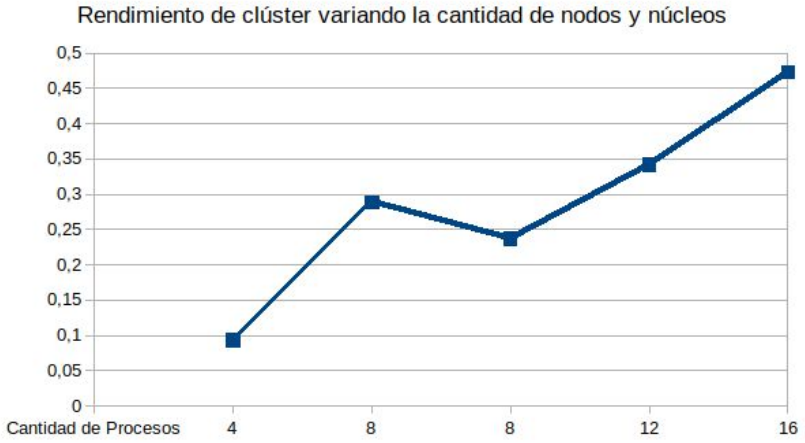


Fig 10. Rendimiento de la ejecución del filtro.

Uso de procesadores en un nodo

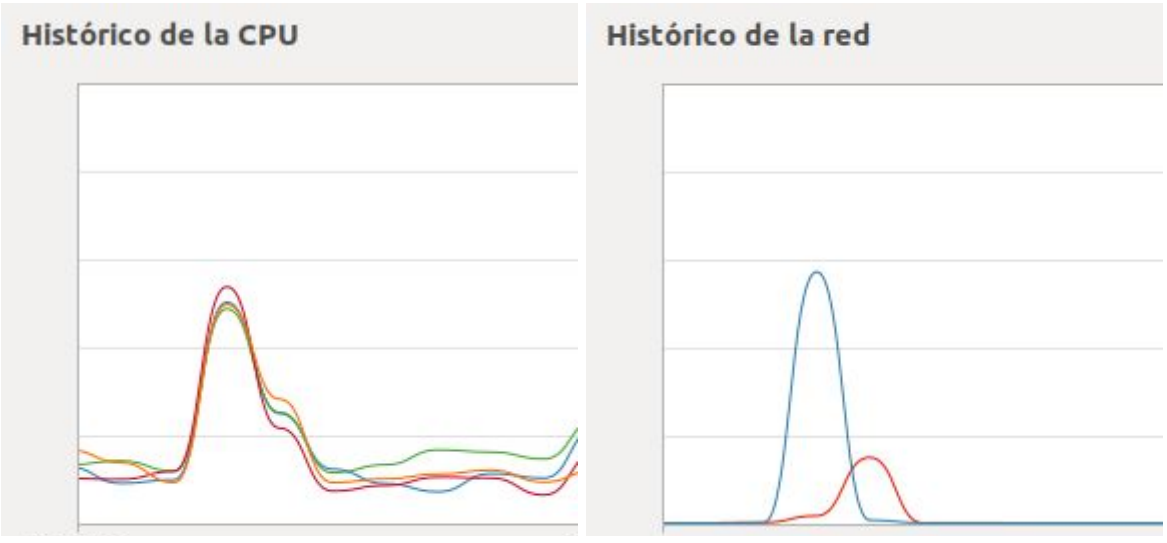


Fig 11. Capturas de la red y procesamiento en un instante de tiempo.

Análisis de resultados

Como se puede observar en la figura 09, el filtrado se realizó de un forma exitosa, realizando un resaltado de los bordes existentes y mostrándolo a una escala de blanco y negro.

Dados los resultados cuantitativos obtenidos anteriormente, se puede decir que se observa que con 4 procesos se logra un tiempo aproximado a 0,1 segundos en contraste con la utilización de 8 procesos, el cual, tiene un tiempo de ejecución de 0,3 aproximado.

De la misma manera, se observa que hay una diferencia poco significativa entre utilizar 8 procesos con 4 núcleos y dos nodos, en contraste con utilizar 2 núcleos y 4 nodos

Conclusiones

Con base en los resultados mostrados se puede concluir que el sistema implementado con un filtro Sobel posee un overhead de tiempo debido a la comunicación entre nodos, ya que se envía toda la imágen a través de la red.

Así mismo, se puede mencionar que aumentar el tamaño de los datos también aumentaría el overhead, dado que el procesamiento de la imágen con menos procesos brinda tiempos de ejecución menores. Solo en casos en que la imagen sea lo suficientemente grande es que se podría empezar a mostrar una mejora en el tiempo de procesamiento.

Referencias

[1] E. Correa y M. Morejón, “Estudio y diseño de cluster Beowulf bajo la plataforma Linux para la elaboración de un Webcluster”, Licenciatura, Universidad Técnica de Ambato, 2005.

[2] "Configura conexiones SSH sin password en solo 3 pasos | Desde Linux", *Desde Linux*, 2011. [Online]. Disponible en: https://blog.desdelinux.net/ssh-sin-password-solo-3-pasos/. [Accesado: 13- Nov- 2017].

[3] B. Aldalur y M. Santamaría. "Realce de imágenes: filtrado espacial." *Revista de teledetección* 17 (2002) pp. 31-42.

[4] O. R. Vincent y O. Folorunso. "A descriptive algorithm for sobel image edge detection." *Proceedings of Informing Science & IT Education Conference (InSITE)*. Vol. 40. 2009.

[5] "Filtros de realce", *Www6.uniovi.es*, 1995. [Online]. Disponible en: http://www6.uniovi.es/vision/intro/node43.html. [Accesado: 13- Nov- 2017].