Instituto Tecnológico de Costa Rica

Área de Ingeniería en Computadores

Arquitectura de Computadores II

Proyecto II
Metodología de Diseño de Sistema

Estudiantes:
Luis Alonso Barboza Artavia
Malcolm Davis Steele
Sebastián Víquez Rojas

Profesor:
Ing. Jeferson González Gómez

18 de mayo de 2018

Índice

Índice de figuras

Metodología de Diseño

Para el presente proyecto, se utilizó una metodología de *Desarrollo en Cascada*, el cual es un enfoque metodológico que ordena las etapas en el desarrollo de un sistema, de tal forma que el inicio de una etapa posterior debe esperar a que se finalice la etapa anterior.[1]

En las siguientes secciones se detallan cada una de estas fases, comenzando con el *Análisis del Problema*, donde se extraen los requerimientos del sistema a diseñar, continuando con la parte de la *Investigación* donde se tiene una idea más clara acerca de lo que se requiere implementar. Posteriormente, la parte de *Comparación y Evaluación de Propuestas de Diseño*, donde se hace un análisis de lo extraído en la parte de Investigación, con argumentos como ventajas, desventajas y consideraciones adicionales para tomar una decisión acerca de lo que se implementará posteriormente en el desarrollo del proyecto.

1. Análisis del problema

Con todos los avances tecnológicos en la industria de la computación e ha llegado a la creación de *supercomputadoras*. Estos son ordenadores con capacidades superiores de procesamiento que son usadas para fines específicos, en los que se trabaja con una gran cantidad de datos y se requieren altos volúmenes de potencia en el procesamiento de dichos datos.

Claro está que concentrar en un único supercomputador que contenga todos los recursos necesarios para el procesamiento requerido no es una buena opción, sobre todo en términos de consumo de potencia. Entonces, hace unas cuantas décadas surgió la idea de integrar varias computadoras en una misma red, de manera que puedan funcionar todos como un sistema computacional que responda a los requerimientos de procesamiento y almacenamiento.

Es así como actualmente es conocida la computación de alto rendimiento (*HPC*), en donde dichas redes se conocen como *clusters*, siendo estos base de aplicaciones de procesamiento en el campo científico, procesamiento de imágenes y vídeo, simulaciones y demás campos en los que se requiere trabajar con gran cantidad de datos.

Es así como para el presente proyecto, se requiere diseñar e implementar un cluster tipo *Beowulf* con 4 nodos, como el que se muestra en la fig.1. Se necesita la implementación de dicho clúster para el desarrollo de un sistema encargado del procesamiento de imágenes mediante un algoritmo a seleccionar, y distribuir la carga de trabajo entre el nodo maestro y los esclavos.

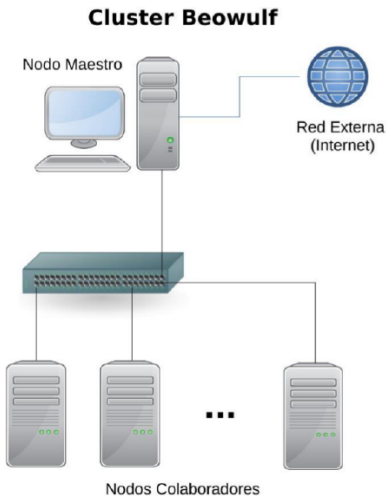

Figura 1: Cluster Beowulf requerido para el sistema.[2]

2. Investigación

2.1. Cluster

Los clusters son conjuntos de computadoras de escritorio o servidores conectados por redes de área local que permiten que funcione dicho grupo de dispositivos interconectados como si fuera una única computadora. La comunicación de los nodos se da mediante un protocolo definido.[3]

Son muy utilizados en aplicaciones de procesamiento de datos como imágenes, vídeo y simulaciones, en donde se requiere mucho procesamiento paralelo. Además, los clusters también han encontrado un campo esencial para el crecimiento del *Software como Servicio* (*SaaS* por sus siglas en inglés) en aplicaciones como búsquedas, redes sociales, compras por internet, entre otras, en donde se requiere de procesamiento de grandes volúmenes de datos.

2.2. Cluster Beowulf

Un cluster Beowulf es una arquitectura de múltiples computadoras conectadas entre sí por una red que puede ser usada para computación paralela. Usualmente consiste en un nodo maestro que actúa como servidor y otros nodos esclavos interconectados a este por medio de alguna conexión de red. El servidor es el encargado de controlar el cluster, así como de abastecer archivos y datos a los demás nodos clientes para el procesamiento y cálculo de la información. Además, por lo general el cluster se encuentra conectado al mundo exterior por un único nodo, el maestro que actúa como consola y puerta de enlace (*gateway*).[4]

2.2.1. Historia

Fue en el año 1994 cuando Thomas Sterling y Don Becker construyeron un cluster de 16 procesadores Intel DX4, de la generación 80486 y que estaban conectados por medio de un canal Ethernet a 10 Mbps. En ese tiempo, la velocidad de procesamiento era muy superior a la velocidad de comunicación, por lo que fue necesario que Becker modificarla los controladores de las tarjetas de red para el sistema operativo Linux, de código abierto. La idea era crear uno que permitiera distribuir el tráfico de red en dos o más tarjetas, para que así se pudiera balancear el sistema.[5]

2.2.2. Ventajas y desventajas

La gran ventaja de un cluster Beowulf es su rentabilidad, ya que únicamente requiere computadores de escritorio. En materia de hardware estas son computadoras relativamente simples, equipadas con procesadores comerciales y recursos que se consiguen fácilmente como memorias RAM y discos duros grandes para el almacenamiento. En cuanto al software, normalmente trabajan con software libre y de código abierto como sistema operativo, por lo que esto abarata mucho los costos.

Mientras, un punto en contra sería una mayor latencia y un menor ancho de banda de las comunicaciones entre los nodos, por lo que no es la mejor opción para cuando se requiere más comunicación que procesamiento de datos en sí, por lo que son ampliamente utilizados en sistemas donde se necesita distribución de trabajo computacional, mas no así en sistemas donde se necesite comunicación constante entre nodos.[6]

2.3. Algoritmos de procesamiento de imágenes

2.3.1. Rayleigh

El Rayleigh es una clase de ruido para imágenes que aparece típicamente en imágenes de velocidad y con alcance radial. Es derivado del ruido uniforme y está basado en la Distribución Rayleigh que está definida para variables aleatorias positivas. En la ecuación de la fig.2 el valor de $\boldsymbol{x}$ es el nivel del valor de grises de la imagen en el pixel actual.

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)}, \quad x \geq 0$$

Figura 2: Función de densidad Rayleigh

En la fig.3 se tiene el uso del filtro Rayleigh, a la izquierda la imagen original, mientras que a la derecha se tiene la imagen procesada con el ruido Rayleigh con un valor de variación de 600.

Figura 3: Ejemplo de uso del filtro Rayleigh en una imagen [7]

2.3.2. Gaussiano

Los filtros Gaussianos son máscaras formadas por una distribución Gaussiana en una matriz de dos dimensiones. Estas máscaras hacen que se elimine el ruido generado por la alta frecuencia pero hacen que la imagen se difumine. Entre más grande sea la máscara, mayor será el desenfoque.[7].

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Figura 4: Función de densidad Gaussiano

En la fórmula de función de densidad del histograma del Gaussiano la z representa el valor de gris, μ como la media de la distribución y σ como la desviación estándar o varianza.

En la fig.5 se muestra la imagen original a la izquierda sin ruido, mientras que a la derecha se tiene la imagen degradada con el filtro Gaussiano con una media de 0 y una variación de 800.

Figura 5: Ejemplo de uso del ruido Gaussiano en una imagen.[7]

2.3.3. Laplaciano

El Laplaciano es un filtro espacial de una generalización de la segunda derivada tomada en dos dimensiones. Se encarga en áreas de cambio rápido únicamente, por ejemplo en una imagen cuando

hay un cambio notorio en la escala de grises de un pixel a otro en un borde.

Como las imágenes son representadas por un conjunto discreto de pixeles, se necesita encontrar una matriz de convolución llamada *kernel*. Esta matriz de dimensiones usualmente mucho menores a la de la imagen original se utiliza para encontrar una aproximación a la segunda derivada de la definición Laplaciana de una imagen.[8]

0	1	0
1	-4	1
0	1	0

Figura 6: Ejemplo de kernel utilizado en el filtro Laplaciano.

Con métodos de convolución es posible obtener una tercera matriz, que es la de la imagen resultante, operando la matriz de valores en escala de grises perteneciente a la imagen original con el kernel seleccionado.

Figura 7: Ejemplo de imagen con filtro Laplaciano.[7]

3. Propuestas, comparación y decisiones de diseño

3.1. Algoritmos de procesamiento de imágenes

Opciones	Consideraciones	Conclusiones
Rayleigh	• Es un tipo de ruido en imágenes. Aparece comúnmente en imágenes tomadas con velocidad. • Una aplicación de la fórmula de la Distribución de Rayleigh es utilizada en el área de imágenes de resonancia magnética para obtener información sobre la estructura y composición del cuerpo.	Se escogió el **Rayleigh** debido a que su fórmula hace que sea paralelizable el cálculo. Además, la importancia y aplicación que tiene en el campo de las imágenes médicas, en las de resonancia magnética. Para los filtros Gaussiano y Laplaciano es necesario el cálculo de matrices auxiliares para el procesamiento final de la imagen, por lo que no son tan efectivos. El filtro Laplaciano es usado en imágenes donde se tengan muchos cambios abruptos de color, o muchos bordes, por lo que su aplicación se reduce a ese tipo de imágenes.
Gaussiano	• Tipo de filtro espacial. • Máscaras formadas en una matriz de dos dimensiones. • Entre más grande sea la máscara, mayor es el desenfoque. • Hace que se elimine el ruido generado en imágenes por alta frecuencia. • La imagen tiende a difuminarse una vez aplicado el filtro.	
Laplaciano	• Es un tipo de filtro espacial. • Se encarga en áreas de cambio rápido, como bordes de imágenes cuando hay un cambio notorio en los colores de la imagen. • Es necesario encontrar una matriz de convolución *kernel* para procesar la imagen y aplicarle el filtro.	

3.2. Interfaz de paso de mensajes (MPI)

Opciones	Consideraciones	Conclusiones
MPICH	• Es un software libre y de código abierto. • Disponible para distribuciones basadas en UNIX como Linux y Mac OS X. • Es una de las implementaciones más populares de MPI. • La utilizan compañías como Intel, IBM Cray y Microsoft en sus propias implementaciones de interfaz de paso de mensajes, derivadas de MPICH. • En el ránking de supercomputadoras del año 2016 se usó en 9 de los 10 primeros lugares, incluyendo la Taihu Light (número 1).	Se escogió **MPICH** como interfaz de paso de mensajes por su gran disponibilidad para versiones de SO basados en Linux, su gran portabilidad, alto rendimiento y su fácil implementación en el código del proyecto.
OpenMPI	• Biblioteca que combina tecnologías y recursos de otros proyectos antecesores en la interfaz de paso de mensajes para computación paralela. • Es usada por muchas de las supercomputadoras presentes en el TOP500, ranking de las computadoras con mayor rendimiento del mundo. • Busca crear una fuente de código abierto que sea la combinación de las mejores ideas y tecnologías en el paso de mensajes. • Busca proveer un ancho de banda grande y una latencia baja en el rendimiento de los programas creados con esta interfaz.	

4. Diseño del sistema

4.1. Descripción del sistema

El sistema desarrollado está compuesto por un cluster Beowulf con 4 nodos, un nodo maestro y 3 esclavos en los que se comunican entre sí por medio de una interfaz de paso de mensajes (MPI) basada en MPICH. Dicha comunicación se da para distribuir el trabajo de cálculos mediante el paralelismo en la aplicación del filtro Rayleigh a una imagen con la ayuda de OpenCV.

El MPICH sería para la comunicación y distribución de trabajo entre los nodos, que se comunican mediante el protocolo SSH. Además, los datos entre los nodos se comparten mediante el protocolo NFS (Network File System), el cual es un sistema de archivos de red que permite a los nodos esclavos interactuar con el directorio creado por el nodo maestro.

La aplicación es una aplicación simple compuesta de un método para filtrado y las colectivas necesarias para distribuir el trabajo en el main.

4.1.1. Métodos

- ***main***: En el main, se aprovecha el hecho de que el mismo código se ejecuta en los procesos para distribuir el procesamiento del filtro de la imagen. Lo primero que se hace es inicializar la biblioteca de MPICH para crear y sincronizar los procesos(lineas 44-47). Luego, el main calcula según la cantidad de procesos y el tamaño de la imagen, el tamaño de la carga que le toca a cada proceso. Seguidamente, se hace un scatter(se divide con una colectiva de comunicación) de los datos binarios de la imagen para aplicar el filtro. Se sincronizan los procesos para asegurar que no empiecen a procesar sin tener los datos de la imagen. Se procesan con el método de filtrado y por último se hace un gather(se junta la imagen con una colectiva de comunicación) y se muestra dicha imagen.
- ***rayleighNoise***: Este método es el encargado de filtrar los datos de la imagen, se utiliza la fórmula del ruido de Rayleigh para generar valores aleatorios con una cierta distribución y se le agregan esos valores a los píxeles de la imagen.

4.1.2. Bibliotecas

- ***OpenCV***: Se utiliza para simplificar el manejo de las imágenes.
- ***MPICH***: Se utiliza para poder distribuir la carga a los diferentes nodos del cluster.

5. Requisitos de software del sistema

- **RSS-1:** Implementación funcional del cluster Beowulf.
- **RSS-2:** Implementación del algoritmo de procesamiento de imágenes (Rayleigh) en nodo maestro.
- Implementación paralela de algoritmo Rayleigh mediante interfaz de paso de mensajes con:
  - **RSS-3:** 1 nodo y N núcleos por nodo.
  - **RSS-4:** 2 nodos y N núcleos por nodo.
  - **RSS-5:** 3 nodos y N núcleos por nodo.
  - **RSS-6:** 4 nodos y N núcleos por nodo.

5.1. Lista de chequeo de cumplimiento de requisitos de software del sistema

En el cuadro 1 se observan los requerimientos de software del sistema y su estado actual en el mismo.

Cuadro 1: Matriz de requerimientos

Requerimiento	Completo
RSS-1	SÍ
RSS-2	SÍ
RSS-3	SÍ
RSS-4	SÍ
RSS-5	SÍ
RSS-6	SÍ

Referencias

[1] Pressman, R. (1996) *Ingeniería del Software*, Madrid: McGraw-Hill, pp.26-30.

[2] González, J. (2018) *Especificación Proyecto II*, Cartago, p.3.

[3] Hennessy, J. y Patterson, D. (1989) *Computer Architecture: A Quantitative Approach*, Elsevier, p.8.

[4] "Creación de un Beowulf", *Universitat Politécnica de Catalunya*, 2006. [Online]. Available: http://personals.ac.upc.edu/enric/PFC/Beowulf/beowulf.html. [Accessed: 06-May-2018].

[5] "Clusters Beowulf", *Revista Universidad Autónoma de México*, 2009. [Online]. Available: http://www.revista.unam.mx/vol.4/num2/art3/clustb.htm [Accessed: 06-May-2018].

[6] "Beowulf Cluster Computing", *Michigan Tech*, 2010. [Online]. Available: http://www.cs.mtu.edu/beowulf/ [Accessed: 07-May-2018].

[7] Myler, H. y Weeks, A. (1993) *The Pocket Handbook of Image Processing Algorithms in C*, New Jersey: Prentice Hall.

[8] "Laplace of Gaussian Filter", *Marquette University*, 2001. [Online]. Available: http://academic.mu.edu/phys/matthysd/web226/Lab02.htm [Accessed: 11-May-2018].