Escuela Politécnica Superior

Algoritmos paralelos para la eliminación de ruido en imágenes médicas

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
José Antonio Agulló García

Tutor/es:
Josep Arnal García

Universitat d'Alacant
Universidad de Alicante

Junio 2019

Algoritmos paralelos para la eliminación de ruido en imágenes médicas

Autor

José Antonio Agulló García

Tutor

Josep Arnal García

Departamento de Ciencia de la Computación e Inteligencia Artificial

GRADO EN INGENIERÍA INFORMÁTICA

Escuela Politécnica Superior

Alicante - Junio 2019

Resumen

En el preprocesado de imágenes digitales cobra gran importancia la tarea de filtrado y eliminación de ruido. Es necesario llevar a cabo este proceso si se quiere mejorar la calidad de imagen y obtener una representación más fiel de la realidad. En el ámbito de las imágenes médicas es de gran interés aplicar métodos de filtrado para facilitar el diagnóstico en imágenes tales como tomografías o radiografías en las que se necesita gran nitidez para fijarse en los pequeños detalles.

Este proyecto se centra en la implementación de un algoritmo paralelo capaz de eliminar ruido gaussiano en imágenes médicas. Las principales fuentes de ruido gaussiano en las imágenes digitales surgen durante la adquisición. El sensor tiene ruido inherente debido al nivel de iluminación y a su propia temperatura. Además, los circuitos electrónicos conectados al sensor inyectan su propia parte de ruido del circuito electrónico. Es un tipo de ruido estadístico con una función de densidad de probabilidad igual a la de la distribución gaussiana o normal; es aditivo e independiente en cada píxel.

En este trabajo se ha llevado a cabo la paralelización de un algoritmo basado en las medias no locales. Se ha elegito este método por su alta capacidad de eliminación de ruido. El principal objetivo de esta paralelización es mejorar el tiempo de preprocesado y poder realizar el filtrado en tiempo real en imágenes con grandes resoluciones, ya que el algoritmo secuencial lleva a cabo una gran cantidad de cálculos. El algoritmo paralelo se ha implementado en lenguaje C haciendo uso del estándar MPI (*Message Passing Interface*) para paralelizar en un entorno distribuido.

Abstract

In the pre-processing of digital images, the task of filtering and removing noise becomes very important. This process is necessary in order to improve image quality and obtain a more accurate representation of reality. In the field of medical images it is of great interest to apply filtering methods to facilitate the diagnosis in images such as tomography or radiographs in which great clarity is needed to look at the small details.

This project focuses on the implementation of a parallel algorithm capable of eliminating gaussian noise in medical images. The main sources of gaussian noise in digital images arise during acquisition. The sensor has inherent noise due to the level of illumination and its own temperature. In addition, the electronic circuits connected to the sensor inject their own portion of noise from the electronic circuit. It is a type of statistical noise with a probability density function equal to that of the gaussian or normal distribution; it is additive and independent at each pixel.

The algorithm has been parallelized in order to improve the preprocessing time and be able to filter in real time in images with high resolutions, since the sequential algorithm performs a large number of calculations. The parallel algorithm has been implemented in C language using the MPI standard (Message Passing Interface) to parallelize in a distributed environment.

Justificación y objetivos

Las imágenes generalmente se adquieren a través de métodos foto electrónicos o foto químicos. En este proceso de adquisición, los mismos sensores encargados de digitalizar dicha imagen, generalmente introducen ruido en ella debido a sus temperaturas. En definitiva, los sensores de imagen son dispositivos ruidosos en sí mismos. El ruido puede definirse como una variación aleatoria indeseada que degrada la imagen con su consiguiente pérdida de calidad.

En el campo de la medicina también es común este problema, lo que causa dificultad para interpretar pequeños detalles y características morfológicas de baja intensidad requeridas para un diagnóstico clínico certero. En concreto, en la tomografía computarizada (TC), durante el proceso de adquisición de imágenes se aumenta la cantidad de ruido presente en ellas si se reduce la radiación ionizada que puede ser maligna para los pacientes.

Es pues, de especial interés, procesar o filtrar la imagen obtenida por dichos sensores para eliminar el ruido presente y poder obtener una imagen con mayor calidad que represente la realidad de manera más fiel, facilitar así el diagnóstico y asegurar el mayor bienestar posible para el paciente.

Los métodos de filtrado suelen introducir otro problema y es la imposibilidad de realizar el tratamiento de imágenes en tiempo real cuando estas son de gran resolución. Normalmente son métodos que requieren grandes cantidades de cómputo, lo cual hace que el proceso sea lento. Por esta razón, se realizará una paralelización del proceso con el objetivo de aumentar el rendimiento y poder obtener la imagen filtrada en menor tiempo.

Agradecimientos

Numerosas son las personas a las que me gustaría agradecer su apoyo, no solo durante el desarrollo de este proyecto, sino durante todo mi paso por la universidad.

En primer lugar, agradecer a mi tutor y profesor Josep Arnal García. Su apoyo y atención a lo largo del desarrollo de este trabajo ha sido fundamental para llevarlo a cabo. No solo se ha dedicado a guiarme, sino también a ayudar en todo lo posible. Desde la explicación del algoritmo hasta la corrección de *bugs* indetectables por mi parte. Por esto y por los años en los que te he tenido de profesor, muchas gracias. Has sabido guiar y transmitir tus conocimientos de la mejor manera posible.

También agradecer a mis compañeros y amigos por esos momentos de desconexión, esos momentos de compartir malas rachas y salir juntos de ellas, esos momentos de ayuda mutua. Porque no todo es estudio, sino también diversión, gracias.

Por último, agradecer a mis padres y hermanos el apoyo recibido durante esta etapa de mi vida. Gracias por aguantar mis quejas, la pantalla de mi portátil encendida a horas intempestivas. Gracias por escucharme y entenderme.

Muchas gracias a todos.

José Antonio Agulló García

Alicante, Junio de 2019

Somos el tiempo que nos queda.

José Manuel Caballero Bonald

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

El término tomografía computarizada o TC, se refiere a un procedimiento computarizado de imágenes en el que se proyecta un haz de rayos X a un paciente desde diferentes ángulos al rededor del cuerpo, produciendo así señales que son procesadas por un computador para generar imágenes transversales del cuerpo. Estos cortes se llaman imágenes tomográficas y contienen información más detallada que los rayos X convencionales.

La tomografía computarizada por rayos X proporciona información muy valiosa para el diagnóstico y el tratamiento del paciente. De esta manera, constituye una de las principales herramientas para diagnosticar múltiples patologías, incluyendo el cáncer, derrames cerebrales y enfermedades cardiovasculares entre otras. Sin embargo, el uso de esta técnica conlleva una gran limitación. Existe un riesgo potencial por el uso de radiación ionizante, la cual podría inducir malignidad en pacientes que son expuestos a esta [1].

El riesgo proveniente del uso de radiación ionizante durante el proceso de adquisición de las imágenes es muy bajo para un individuo. En la mayoría de los casos está justificado su uso, pues un diagnóstico acertado y a tiempo es beneficioso para el paciente. Aun así, es de especial interés diminuir dichas dosis de radiación al adquirir las imágenes. Sobre todo en el caso de niños y jóvenes que podrían acumular una dosis significativa de radiación al exponerse múltiples veces al escaneo, lo cual podría desencadenar efectos secundarios peligrosos [2, 3].

No obstante, no es tan sencillo como reducir la cantidad de radiación al obtener las imágenes, pues en la TC existe una relación directa entre la calidad de la imagen y la dosis de radiación ionizante emitida. A mayor cantidad de radiación, mayor es la calidad de la imagen [4]. En definitiva, con dosis bajas, se aumenta la cantidad de ruido en las imágenes, el

cual puede ocultar detalles anatómicos y disminuir la detectabilidad de lesiones con bajo contraste. Además, a dosis muy bajas, pueden exacerbarse efectos indeseados en las imágenes, tales como artefactos. Esto se debe a que los sensores del tomógrafo reciben una cantidad menor de fotones, disminuyendo así la relación señal a ruido.

De este modo, para conservar un balance adecuado, el principio por el cual se rige la TC es el de usar las menores dosis de radiación que conserven razonablemente la calidad de la imagen adquirida. Por esta razón, un área actual de investigación consiste en buscar métodos que reduzcan esta dosis de radiación pero manteniendo una calidad de imagen propia de escaneos con mayores cantidades de radiación. Las estrategias empleadas generalmente utilizan mejoras en los sistemas de adquisición o en la reconstrucción de la imagen, así como en el procesado de las imágenes una vez reconstruidas.

Este proyecto se centra en una estrategia basada en la aplicación del filtro de medias no locales (*Non-Local Means*) [5] sobre las imágenes reconstruidas a partir de las proyecciones de los rayos X. El gran problema de este algoritmo es el gran coste computacional que supone al aplicarse en imágenes de altas resoluciones. Esto conduce a ser un algoritmo prohibitivo en caso de querer utilizarse para filtrar imágenes en tiempo real. Por este motivo, durante el desarrollo de este trabajo se diseña y se implementa el algoritmo de medias no locales de manera paralela para poder ejecutarse en varias unidades de cómputo simultáneamente y así poder obtener las imágenes filtradas en tiempo real.

1.1. Objetivos

Los principales objetivos del proyecto son, a partir de un algoritmo secuencial para el filtrado de ruido gaussiano, implementarlo en paralelo y posteriormente realizar un estudio de la ganancia de velocidad (*speedup*). Las tareas seguidas son las siguientes:

- Estudio del algoritmo e implementación secuencial.
- Diseño del algoritmo paralelo.
- Estudio del estándar MPI e implementación paralela.
- Experimentación y extracción de conclusiones.

1.2. Estructura del trabajo

- ***Capítulo 1: Introducción*** ⇒ Introducción del problema a resolver y los objetivos del proyecto. Así como la estructura del documento.

- ***Capítulo 2: Estado del arte*** ⇒ Base teórica de varios algoritmos existente para la eliminación de ruido en imágenes digitales. Primer contacto con el algoritmo que se implementa en el proyecto.

- ***Capítulo 3: Planificación*** ⇒ Planificación secuencial del flujo de trabajo realizado a lo largo del proyecto.

- ***Capítulo 4: Tecnologías*** ⇒ Tecnologías utilizadas a lo largo del desarrollo del proyecto, como pueden ser editor de código fuente, sistema de control de versiones, lenguaje de programación, etc.

- ***Capítulo 5: Algoritmo*** ⇒ Análisis y descripción detallada del algoritmo Non-Local Means. También se describe la implementación secuencial realizada y su respectiva paralelización.

- ***Capítulo 6: Experimentación*** ⇒ Descripción de la experimentación realizada durante el proyecto. Se detallan tanto los resultados obtenidos en cuanto a la calidad de filtrado del algoritmo, como los resultados obtenidos en cuanto a la ganancia del algoritmo paralelo.

- ***Capítulo 7: Conclusiones*** ⇒ Conclusiones que se extraen del desarrollo del proyecto y del resultado obtenido.

- ***Capítulo 8: Líneas futuras de trabajo*** ⇒ Se estudian posibles ampliaciones futuras del proyecto.

2 Estado del arte

Un área actual de investigación consiste en buscar métodos que reduzcan la dosis de radiación ionizante en el proceso de adquisición de imágenes por tomografía computacional pero manteniendo una calidad de imagen propia de adquisiciones con mayores cantidades de dicha radiación. Las métodos empleados para ello generalmente consisten en mejorar los sistemas de adquisición/reconstrucción de imágenes o el procesamiento de las imágenes reconstruidas.

Este proyecto se centra en el procesamiento de las imágenes con el fin de reducir el ruido presente en las mismas. Es por esto por lo que se van a describir diferentes algoritmos existentes para el filtrado de imágenes.

2.1. Filtro de la Mediana

El filtro de la mediana [6] es un filtro no lineal ampliamente utilizado por su efectividad a la hora de eliminar ruido preservando los bordes de la imagen. Es un método local en el cual el píxel a filtrar se procesa teniendo en cuenta los valores de sus píxeles vecinos. Los píxeles vecinos son aquellos que se encuentran dentro de la ventana de vecindad, que no es más que una ventana de radio definido centrada en el píxel a filtrar. Es un algoritmo sencillo, cuyo objetivo es reemplazar cada píxel de la imagen por la mediana de los píxeles que se encuentran dentro de su ventana de vecindad.

2.2. Filtro de la Media

El filtro de la media [7] es un filtro lineal empleado para suavizar imágenes reduciendo la variación de intensidad entre píxeles contiguos. Se utiliza para eliminar ruido en imágenes, aunque tiende a suavizar los bordes de las mismas, perdiendo así nitidez. Al igual que el método anterior, es un algoritmo local, por lo que también tiene en cuenta los píxeles vecinos al píxel que se quiere filtrar. La idea de este método de filtrado es simplemente reemplazar cada píxel de la imagen con el valor promedio de sus vecinos, incluyéndose a sí mismo.

2.3. Filtro Bilateral

El filtro bilateral [8,9] es un filtro adaptativo que suaviza imágenes mediante una combinación no lineal de valores de píxeles cercanos. Es un método no iterativo y local que permite conservar los bordes tras el filtrado. En este filtro, un píxel se procesa teniendo en cuenta criterios geométrico y fotométricos; es decir, entran en juego las distancias espaciales entre píxeles así como la diferencia entre intensidades.

De esta manera, dos píxeles contiguos tienden a promediarse si únicamente se utiliza un criterio de distancia espacial; sin embargo, en el filtro bilateral, también se tiene en cuenta su intensidad. Si las intensidades son similares, los píxeles tienden a igualarlas; pero, si son muy distintas, dicha diferencia se aplica al cómputo del algoritmo y ambos píxeles conservan sus intensidades originales. Esto ocurre en el caso de los bordes de las imágenes.

2.4. Filtro de Medias no Locales

Este trabajo se centra en el filtro de medias no locales [5], un algoritmo no local en el que el valor de los píxeles de la imagen de entrada se modifica mediante un promedio de aquellos píxeles que se consideran de mayor similitud, independientemente de su localización en la imagen. En el capítulo 5 se estudia en profundidad este método de filtrado.

3 Planificación

Se han definido diferentes etapas secuenciales para la realización correcta del proyecto. Entre ellas destacan etapas de formación, implementación, diseño y experimentación.

- **Investigación.** Como primera toma de contacto, se ha investigado a cerca de los problemas de la tomografía computarizada en cuanto a la radiación ionizada. De esta manera, se entiende y se define el problema a resolver, que es el aumento de ruido en la adquisición de imágenes médicas cuando se reduce la dosis de radiación.

- **Formación.** Con el fin de poder resolver el problema, se ha estudiado el algoritmo Non-Local Means para el filtrado de ruido en imágenes.

- **Implementación.** Una vez aprendido el algoritmo, se ha procedido a implementar el algoritmo en lenguaje C de manera secuencial.

- **Formación.** Antes de empezar con la paralelización del algoritmo ha sido necesario estudiar las principales funciones que otorga el estándar MPI, así como su integración en C. De esta manera se facilita el proceso de implementación.

- **Diseño.** La siguiente etapa corresponde al diseño del algoritmo paralelo. Es importante diseñar cómo se va a abordar la solución antes de empezar con la implementación. En esta etapa sobre todo se ha estudiado la manera de dividir el proceso entre las diferentes unidades de cómputo.

- **Implementación.** Una vez diseñado el algoritmo, se procede a adaptar el código secuencial para implementarlo en paralelo. En esta etapa también se sigue con la formación en el estándar MPI debido a los problemas que surgen durante el desarrollo.

- **Experimentación.** Ejecución de diferentes pruebas con el algoritmo secuencial y paralelo con su consiguiente extracción de datos: resolución de imagen de entrada, tiempo de ejecución, número de unidades de cómputo en las que se ejecuta el algoritmo paralelizado, etc.

- **Conclusiones.** Estudio de los resultados obtenidos en la fase de experimentación mediante comparativas de ambos algoritmos.

4 Tecnologías

Se han utilizado varias tecnologías a lo largo del desarrollo. Entre ellas destacan Git, Visual Studio Code, C, GCC y MPI. Cabe destacar que todo el desarrollo del proyecto se ha realizado bajo el Sistema Operativo Ubuntu Xenial 16.04, exceptuando la experimentación que se ha realizado en el supercomputador descrito en la sección 4.2.1.

4.1. Software

4.1.1. Sistema de control de versiones

Como sistema de control de versiones se ha utilizado de Git [10], diseñado por Linus Torvalds, inicialmente lanzado en 2005. Esta herramienta permite la gestión automática de los diversos cambios que se realizan sobre los elementos del proyecto, en este caso, sobre el código fuente. No es una tecnología completamente necesaria para el desarrollo del proyecto, pero facilita la tarea de guardar avances y deshacer cambios en el código que no producen el comportamiento deseado.

4.1.2. Editor de código fuente

Visual Studio Code, desarrollado por Miscrosoft y lanzado en 2015, se ha utilizado como editor del código fuente. Es un editor de texto altamente personalizable que incluye soporte para la depuración y es compatible con varios lenguajes de programación, entre ellos C, el lenguaje utilizado en este proyecto. Esta compatibilidad facilita el desarrollo, pues permite

al usuario completar código de manera inteligente, añadir fragmentos de código y refactorizarlo. Además, permite añadir plugins que resaltan la sintaxis del lenguaje y lo formatean automáticamente para una correcta visualización. También tiene integración con el sitema de control de versiones Git, mencionado anteriormente.

4.1.3. Lenguaje de programación

El lenguaje de programación utilizado durante el desarrollo del proyecto ha sido C [11], desarrollado por Dennis Ritchie y lanzado en 1972. Es un lenguaje de programación imperativo que dispone de estructuras típicas de lenguajes de alto nivel y, a su vez, permite un control a muy bajo nivel. Se ha utilizado por su eficiencia y por ser uno de los lenguajes compatibles con MPI.

4.1.4. MPI

El pilar del proyecto reside en el uso del estándar MPI [12], pues es la tecnología que permite la parelización de código. La Interfaz de Paso de Mensajes (MPI, *Message Passing Interface*) es un protocolo de comunicación entre computadoras. De esta manera, permite la comunicación entre nodos que ejecutan un programa en un sistema de memoria distribuida (sistema en el que cada una de las unidades de cómputo posee su espacio de memoria). Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que pueden ser utilizadas en programas escritos en los lenguajes de programación C, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que la utilizan son portables, dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida, y rápidos, pues cada implementación de la librería ha sido optimizada para el hardware en el cual se ejecuta.

4.1.5. Octave

GNU Octave [13] es un lenguaje de alto nivel desarrollado por el Proyecto GNU y lanzado en 1988 destinado principalmente a cálculos numéricos. Proporciona una interfaz de línea de comandos conveniente para resolver problemas lineales y no lineales numéricamente y para realizar otros experimentos numéricos. Tiene extensas herramientas para resolver problemas comunes de álgebra lineal numérica, encontrar las raíces de ecuaciones no lineales, integrar funciones ordinarias, manipular polinomios e integrar ecuaciones diferenciales y diferenciales-algebraicas ordinarias.

Se ha utilizado principalmente para el cálculo del PSNR (ver sección 6.1) y la adición de ruido gaussiano con diferentes desviaciones estándar a las imágenes de prueba.

4.1.6. Compiladores

GCC (*GNU Compiler Collection*) [14], es un conjunto de compiladores de software libre desarrollados por el Proyecto GNU y lanzados en 1987. Se ha utilizado en su versión 5.4.0 para compilar el programa secuencial escrito en C. El código paralelo se ha compilando haciendo uso de MPICC, que proporciona las opciones y las librerías necesarias para compilar y enlazar programas MPI escritos en C.

Para facilitar la compilación del proyecto se ha hecho uso de la herramienta GNU Make. Esta herramiento permite la gestión de dependencias, típicamente, existentes entre los archivos que componen el código fuente de un programa, para dirigir su compilación/recompilación automáticamente.

4.2. Hardware

4.2.1. Supercomputador

El clúster con el que se ha realizado la experimentación y el cálculo de tiempos, posee el sistema operativo CentOS Linux versión 5.6 para la arquitectura x86 de 64bit [15]. Su composición es de veintiséis nodos de cálculo HP Proliant SL 390 G7. La composición de cada uno de estos nodos es

- 2 x CPU Intel Xeon X5660
  - 6 cores de 2,80 Ghz, Max Turbo 3,2 Ghz
  - 12 MB cache Level 2
- Memoria RAM 48 GB DDR3-1333
- Disco duro SATA de 500 GB

lo cual hace un total de $26 \times 2 \times 6$ unidades de cómputo. Cabe destacar que la comunicación entre estos nodos se realiza a través de una red Gigabit Ethernet y una red de baja latencia basada en Infiniband QDR.

5 Algoritmo

En este capítulo se describirá el algoritmo Non-Local Means para la correción de ruido en imágenes digitales con su implementación en secuencial y, posteriormente, cómo se ha diseñado su implementación en paralelo.

5.1. Non-Local Means

El filtro de medias no locales, *Non-Local Means* [5], es un algoritmo empleado en el procesamiento de imágenes digitales para la corrección de ruido. Como su nombre indica, es un algoritmo no local. Esto significa que, a diferencia de otros filtros locales de corrección de ruido, el valor de los píxeles de la imagen de entrada se modifica mediante un promedio de aquellos píxeles que se consideran de mayor similitud, independientemente de su localización en la imagen. En definitiva, este algoritmo se puede aplicar recorriendo todos los píxeles de la imagen para cada uno de los píxeles a filtrar. Esta característica otorga al algoritmo una mayor calidad de filtrado, pues se produce una menor pérdida de detalles en la imagen. La descripción formal del algoritmo es la siguiente.

Dada una imagen discreta ruidosa $v = \{v(i)|i \in I\}$, el valor estimado en la imagen filtrada $NL[v](i)$, para un píxel i, viene dado por una media ponderada de todos los píxeles en la imagen

$$NL[v](i) = \sum_{j \in I} w(i,j)v(j) \tag{5.1}$$

donde el conjunto de pesos $\{w(i,j)\}_j$ depende de la similitud entre los píxeles i y j y satisface las condiciones $0 \leq w(i,j) \leq 1$ y $\sum_j w(i,j)v(j) = 1$.

Esta similitud depende de la similitud del vector de intensidad del nivel de gris $v(N_i)$ y $v(N_j)$, donde N_k es el vecindario cuadrado de tamaño fijo centrado en el píxel k. Esta similitud se mide como una función decreciente de la distancia euclídea ponderada, $\| v(N_i) - v(N_j) \|^2_{2,a}$, donde $a > 0$ es la desviación estándar del kernel gaussiano. Aquellos píxeles con un vencindario similar en nivel de grises a $v(N_i)$ tienen un mayor peso en el promedio. Dichos pesos están definidos como

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{\|v(N_i)-v(N_j)\|^2_{2,a}}{h^2}} \tag{5.2}$$

donde $Z(i)$ es la constante de normalización

$$Z(i) = \sum_j e^{-\frac{\|v(N_i)-v(N_j)\|^2_{2,a}}{h^2}} \tag{5.3}$$

y el parámetro h actúa como grado de filtrado y controla la caída de la función exponencial y, por lo tanto, la caída de los pesos en función de las distancias euclídeas.

5.1.1. Adaptación del algoritmo

Para la implementación del algoritmo descrito, se ha añadido una ventana de búsqueda. Esto implica que, para cada píxel de la imagen, no se recorren todos los píxeles de la imagen, sino que se recorren los píxeles de dicha ventana. De esta manera, la ecuación 5.1 vendría dada por

$$NL[v](i) = \sum_{j \in R} w(i,j) v(j) \tag{5.4}$$

donde R es la ventana de búsqueda definida con i como píxel central.

5.2. Notas previas

Es importante explicar algunas de las estructuras de datos empleadas para el desarrollo del algoritmo. Las más importantes corresponden a dos *struct* definidos para encapsular la información relativa a la imagen de entrada y a las ventanas de similitud del algoritmo.

En primer lugar, cabe destacar que el código funciona con imágenes en formato PGM (*Portable Gray Map*) versión P5, también conocida como versión raw [16]. Es una versión del formato binaria. También existe una versión no binaria, la versión P2; también conocida como versión plana del formato. Una imagen PGM representa una imagen gráfica en escala de grises. Este formato contiene lo siguientes datos, separados por espacios en blanco o tabuladores:

- Dos caracteres que identifican la versión del fichero. En el caso descrito, los caracteres «P5».
- La anchura (x) y altura (y) de la imagen formateadas como caracteres ASCII. Sus valores se escriben en decimal.
- El máximo valor de gris, también escrito en ASCII decimal. Debe ser un valor inferior a 65536 y mayor que 0.
- Una conjunto de x filas ordenadas de arriba hacia abajo. Cada una de las filas consta de y valores de gris ordenados de izquierda a derecha. Cada valor gris es un número comprendido entre 0 (negro) y el máximo valor de gris (blanco). Cada valor de gris corresponde a un píxel y se representa en binario puro por uno o dos bytes. Si el máximo valor de gris es inferior a 256, es de un byte. En caso contrario, es de dos bytes; siendo el primero de ellos el más significativo.

De esta manera, se ha definido una estructura que contiene toda la información relativa a la imagen de entrada que recibe el programa: número de filas, número de columnas, máximo valor de gris y la matriz correspondiente a cada uno de los píxeles de la imagen. Esta información se extrae con el correspondiente método encargado de leer el fichero de entrada. También se ha definido un método encargado de escribir en un fichero esta estructura.

Con respecto a la matriz de la imagen, es necesario remarcar que se define de manera contigua en memoria fila a fila. Se ha definido un método encargado de reservar el espacio en memoria de esta manera específica, porque si bien C almacena cada fila de manera contigua en memoria, no lo hace entre las diferentes filas de una matriz. Esta elección facilita posteriormente el envío de mensajes utilizando MPI.

La segunda estructura de datos que se ha definido contiene la información relativa a una ventana: matriz sobre la que trabaja, con sus respectivos límites (fila inicial, fila final, columna inicial y columna final); así como la anchura y la altura de la ventana. Así pues, según la definición del tipo de datos, se accede al píxel $i \times j$ de la ventana mediante la operación $i \times j = matriz[fila\_inicial + i][columna\_inicial + j]$, donde $0 < i < altura$ y $0 < j < anchura$.

5.3. Implementación secuencial

Con todo lo descrito en las secciones anteriores, el algoritmo finalmente implementado es el siguiente. Para todos los píxeles p de la imagen, tendríamos que el píxel filtrado es

$$p' = \frac{1}{\sum_{q \in B(p,r)} w(p,q)} \sum_{q \in B(q,r)} q \cdot w(p,q) \tag{5.5}$$

donde $B(i,x)$ es una ventana de vecindad centrada en el píxel i y de radio x, lo cual supone un tamaño de $(2x+1) \times (2x+1)$. De esta manera, $B(p,r)$ y $B(q,r)$ hacen referencia a la ventana de búsqueda mencionada en la sección 5.1.1 con un radio r fijado por parámetro en el código del programa.

El peso o similitud entre píxeles se computa de la siguiente manera

$$w(p,q) = e^{-\frac{max(d^2 - 2\sigma^2, 0,0)}{h^2}} \tag{5.6}$$

$$w(p,p) = max\{w(p,q),\ q \in B(p,r)\} \tag{5.7}$$

donde σ es la desviación estándar del ruido, h es el parámetro de filtrado fijado en función del valor de σ y d es la distancia euclídea ponderada. Esta distancia entre dos ventanas de

similitud de radio fijado f, $B(p,f)$ y $B(q,f)$, se calcula según la siguiente expresión

$$d^2 = \frac{1}{(2f+1)^2} \sum_{k \in K} (B(p,f)[k] - B(q,f)[k])^2 \tag{5.8}$$

donde K es el conjunto de índices de una ventana de radio f y $B(p,f)[k]$ hace referencia al píxel k en la ventana $B(p,f)$. De la misma manera se obtiene el píxel k en la segunda ventana de similitud.

Para implementar la ecuación 5.5, se va acumulando la suma de los pesos que se van calculando (denominador de la expresión) y también se acumulan las multiplicaciones del píxel q con su respectivo peso (numerador de la expresión). La expresión 5.7 se traduce por el máximo $w(p,q)$ para todo q perteneciente a la ventana de búsqueda centrada en p. Esto se ha implementado almacenando el mayor peso computado hasta el momento en cada iteración del bucle principal.

Como se puede observar, en la expresión 5.6 se hace uso de la distancia euclídea descrita en 5.8. Esta distancia se ha calculado mediante la implementación del *struct* mencionado anteriormente y que representa las ventanas de similitud de cada píxel. De esta manera, se ha implementado un método que recibe dos ventanas de similitud y computa la diferencia entre ellas.

Es importante remarcar que la imagen original de entrada se acolcha f píxeles por todos los lados de la misma. Esto se ha diseñado así para evitar problemas de rango a la hora de recorrer píxeles de la imagen o de las ventanas definidas. En el caso de las ventanas de radio r se hacen comprobaciones antes de recorrerlas para no salirse de los rangos.

5.4. Implementación paralela

En un algoritmo paralelo se tienen diferentes unidades de cómputo o procesos, que llevan a cabo tareas de manera simultánea. Para la explicación de esta implementación, se define K como el número total de unidades de cómputo disponibles. De todos estos procesos, uno de ellos es el proceso *root* o padre, que se diferencia de los demás en que este se encarga de la lectura de la imagen de entrada y la escritura de la imagen filtrada.

El esquema básico de paralelización diseñado para la implementación del algoritmo consiste en las siguientes partes:

- Lectura y acolchado de la imagen de entrada por el proceso padre.
- División de los datos de entrada entre las diferentes unidades de cómputo.
- Cómputo paralelo del algoritmo. En definitiva, aplicación del algoritmo en los diferentes procesos.
- Unión de los resultados de todos los procesos al proceso padre para la escritura de la imagen filtrada.

Así pues, en primer lugar, el proceso padre lee la imagen de entrada y la acolcha mientras el resto de procesos espera en lo que se conoce como una barrera. Con respecto al acolchado, cabe destacar que no se acolcha con f píxeles como en el algoritmo secuencial, sino que se acolcha con $r+f$ píxeles a todos los lados de la imagen; siendo f el radio de la ventana de similitud y r el radio de la ventana de búsqueda.

El principal problema a resolver en la paralelización del algoritmo reside en dividir la tarea entre las diferentes unidades de cómputo disponibles. Esta división se lleva a cabo mediante la descomposición de la imagen de entrada I en K partes, siendo K el número de unidades de cómputo disponibles. Así, se tiene que cada unidad de cómputo k, trabaja con I_k como imagen de entrada. Con el objetivo de implementar un algoritmo eficiente, se realiza una división de la imagen por filas debido a la manera en que C almacena en memoria los elementos de un array, como se menciona en la sección 5.2.

Siendo N el número total de filas de la imagen, la cantidad de filas n que le corresponde a cada proceso k, se calcula de manera trivial

$$n_k = \begin{cases} \frac{N}{K} + N \text{ mód } K & \text{si } k = 0 \\ \\ \frac{N}{K} & \text{si } k \neq 0 \end{cases} \tag{5.9}$$

donde n_0 es el proceso padre. Con esto, el proceso padre envía n_k filas de la imagen a cada uno de los procesos de manera contigua. A partir de ahora, se entiende como n_k a las filas concretas que recibe el proceso k y no al número de filas en sí.

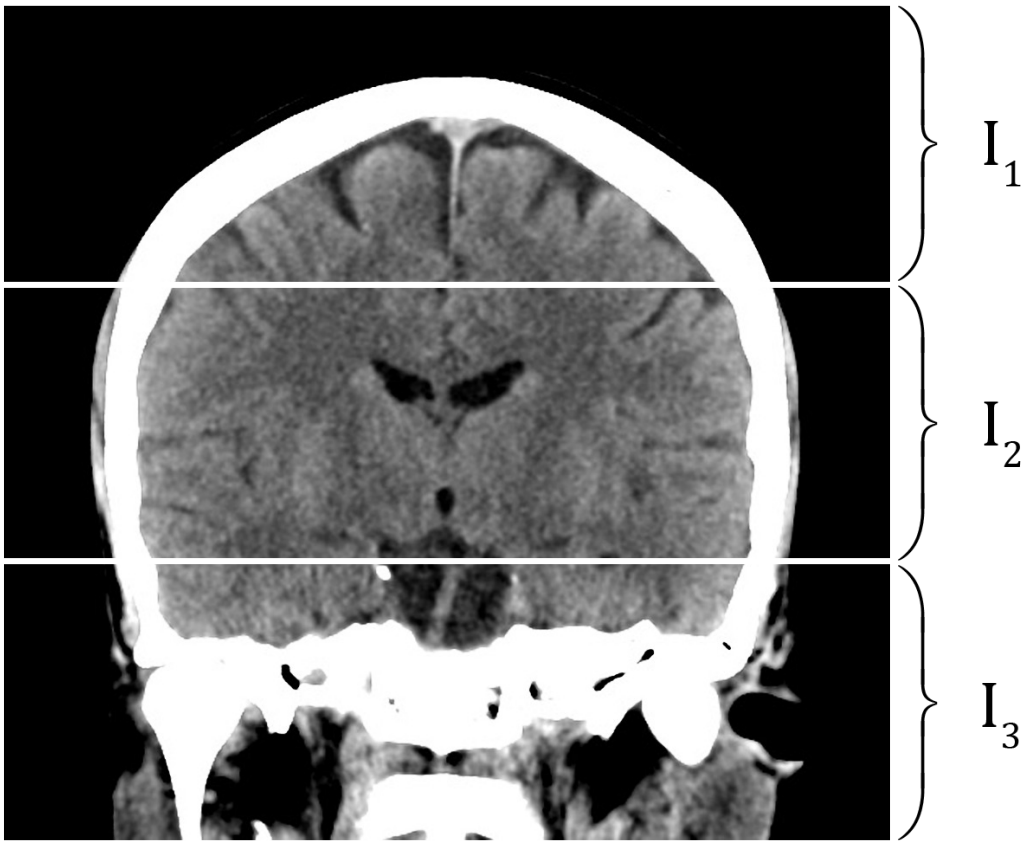


Figura 5.1: División de la imagen en tres unidades de cómputo (elaboración propia)

Es más problemática la división de la imagen acolchada, pues no basta con calcular el número de filas que le corresponde a cada proceso. En esta división se tiene en cuenta que cada k necesita que su porción de imagen de entrada esté acolchada $r+f$ píxeles. Es por esto por lo que la división de la imagen acolchada ya no se realiza de manera contigua, sino que se hace mediante solapamiento. Esto quiere decir, por ejemplo, que el proceso $k=2$, en su imagen acolchada, posee $r+f$ filas pertenecientes a n_1 y n_3; suponiendo que $K>2$.

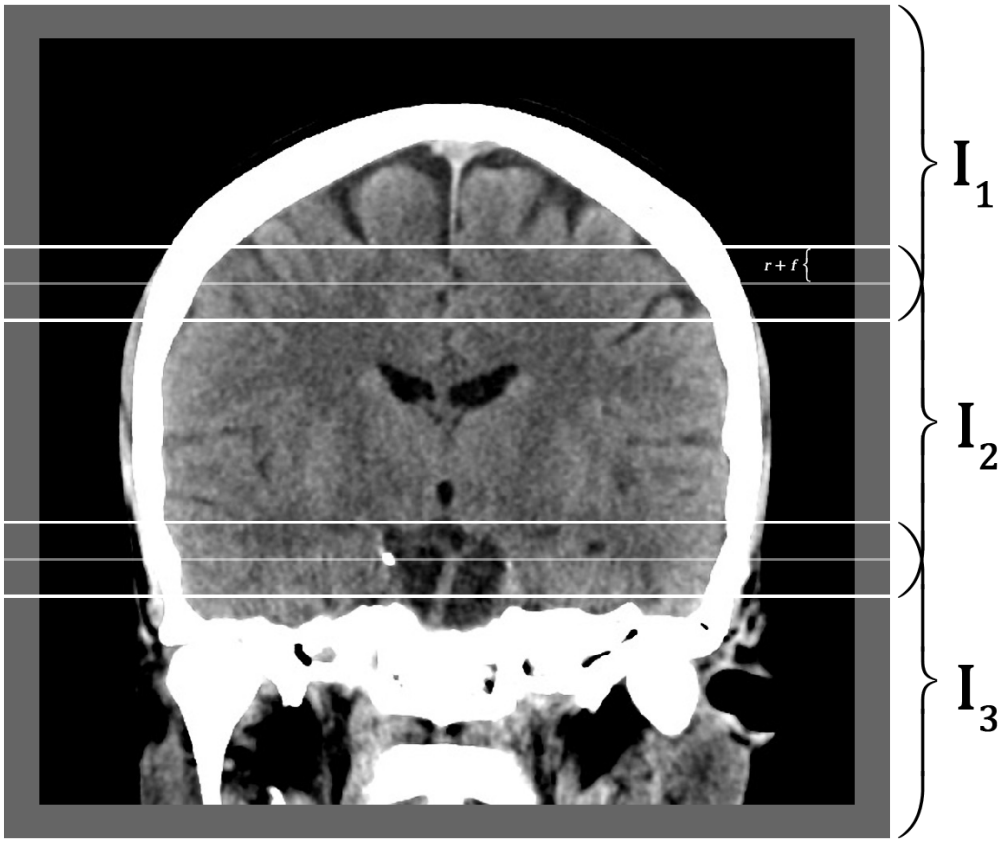


Figura 5.2: División de la imagen acolchada $r+f$ píxeles (elaboración propia)

Con su porción de imagen original e imagen acolchada, cada uno de los procesos puede aplicar el algoritmo. Gracias al acolchado de $r+f$ y la división solapada de la imagen, se consigue un resultado equivalente al algoritmo en secuencial. Una vez todos los procesos obtienen su parte de la imagen filtrada, la envían al proceso padre, que se encarga de unirlas y escribirlas en el fichero de salida.

6 Experimentación

En este capítulo se realiza una descripción de la experimentación realizada durante el proyecto. Es importante destacar que dichas pruebas se han ejecutado bajo el clúster descrito anteriormente en la sección 4.2.

Por un lado, se comentan los resultados obtenidos desde el punto de vista de la calidad de filtrado del algoritmo descrito. De esta manera, se muestran diferentes imágenes médicas con ruido y su posterior imagen filtrada, así como los valores del PSNR (*Peak Signal-to-Noise Ratio*) [17].

Por otro lado, se realiza un análisis de rendimiento del algoritmo paralelo con respecto al secuencial. Para ello se miden los tiempos de ejecución del programa en secuencial y paralelo con la misma imagen de entrada y los mismos parámetros, con el fin de calcular la ganancia o *speedup*.

6.1. Resultados

En primer lugar, es necesario remarcar que existen diferentes métodos para estimar la calidad de una imagen procesada con respecto a la imagen original. Uno de ellos es el PSNR, que representa la relación entre la potencia máxima posible de una señal y la potencia de ruido corruptor que afecta a la fidelidad de su representación. En el caso que se estudia, la señal corresponde con la imagen original y el ruido con la imagen con ruido gaussiano introducido tras su filtrado.

De esta manera, el PSNR se define fácilmente mediante el error cuadrático medio, conocido por sus siglas en inglés MSE (*Mean Squared Error*). El MSE se puede definir de la siguiente manera, dada una imagen I monocroma sin ruido de dimensiones $m \times n$ y la misma imagen con ruido filtrada K

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I(i,j) - K(i,j))^2 \tag{6.1}$$

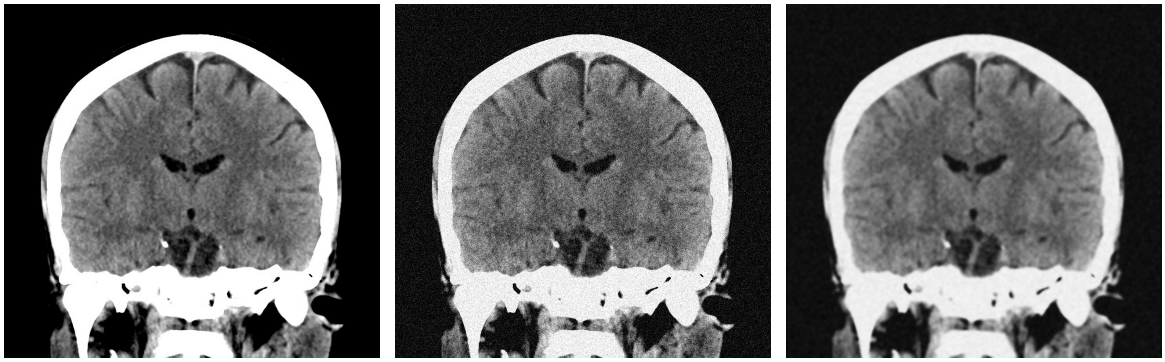
El PSNR, en decibelios, se define como

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \tag{6.2}$$

donde MAX_I es el máximo valor posible de un píxel en la imagen I. Cabe destacar que, a mayor valor de PSNR, mayor es la calidad de la imagen filtrada.

En segundo lugar, procede señalar que los resultados que se muestran a continuación se han obtenido con una ventana de búsqueda r de dimensiones 11×11 (radio 5), una ventana de similitud f de dimensiones 5×5 (radio 2) y con el parámetro de filtrado fijado a $h = 5 \cdot \sigma$, donde σ es la desviación estándar del ruido añadido a la imagen original. Estos valores de los parámetros se han elegido teniendo en cuenta los resultados empíricos que demuestran su buen comportamiento [5, 18].

Se ha escogido un conjunto de imágenes médicas a las cuales se les ha añadido ruido gaussiano con diferentes desviaciones: 10, 20, 30 y 40. Una vez obtenidas todas las imágenes, se ha calculado el PSNR entre la imagen original y la imagen ruidosa. A continuación, tras aplicar el algoritmo de filtrado, se ha calculado el PSNR entre la imagen original y la imagen filtrada. De esta manera se puede realizar una comparación entre ambos PSNR y determinar si el algoritmo aumenta la calidad de la imagen. Con el fin de que sea más ilustrativo el resultado visual, únicamente se muestran aquellos ejemplos con $\sigma = 40$.

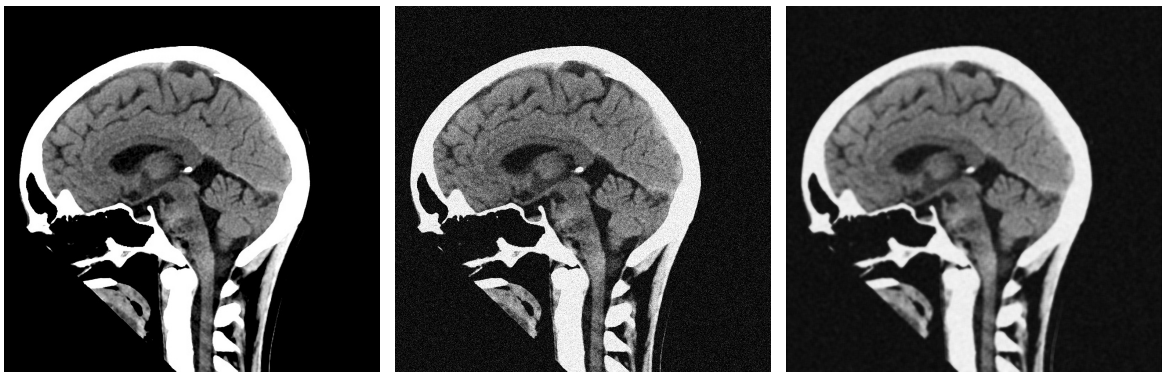


(a) Imagen original, 1024 x 964 (b) Imagen con ruido gaussiano añadido de desviación 40 (c) Imagen filtrada

Figura 6.1: Imagen axial (cortesía del Prof. Frank Gaillard, radiopaedia.org, rID: 37008)

(a) Imagen original, 1024 x 1024 (b) Imagen con ruido gaussiano añadido de desviación 40 (c) Imagen filtrada

Figura 6.2: Imagen coronal (cortesía del Prof. Frank Gaillard, radiopaedia.org, rID: 37008)



(a) Imagen original, 1024 x 997 (b) Imagen con ruido gaussiano añadido de desviación 40 (c) Imagen filtrada

Figura 6.3: Imagen sagital (cortesía del Prof. Frank Gaillard, radiopaedia.org, rID: 37008)

A continuación, se muestran los valores de PSNR entre las imágenes de ejemplo. Esto es, el cálculo de PSNR para todas las combinaciones entre imagen original e imagen ruidosa; así como todas las combinaciones entre imagen original e imagen filtrada. Teniendo en cuenta lo mencionado anteriormente, existen imágenes ruidosas de diferentes desviaciones. También se muestra la diferencia de PSNR con el fin de remarcar el aumento del mismo tras filtrar la imagen ruidosa.

σ	Imagen original y ruidosa	Imagen original y filtrada	Diferencia
10	29.57	32.95	3.38
20	23.55	28.67	5.12
30	20.07	25.91	5.84
40	17.64	24.01	6.38

Tabla 6.1: Valores de PSNR entre las diferentes imágenes axiales (elaboración propia)

σ	Imagen original y ruidosa	Imagen original y filtrada	Diferencia
10	29.97	32.89	2.90
20	24.00	28.10	4.12
30	20.48	25.01	4.53
40	18.03	23.00	4.98

Tabla 6.2: Valores de PSNR entre las diferentes imágenes coronales (elaboración propia)

σ	Imagen original y ruidosa	Imagen original y filtrada	Diferencia
10	29.98	32.36	2.38
20	23.99	27.75	3.75
30	20.52	24.71	4.19
40	18.10	22.73	4.63

Tabla 6.3: Valores de PSNR entre las diferentes imágenes sagitales (elaboración propia)

Como se puede observar, el PSNR entre imágenes originales y ruidosas disminuye cuanto mayor es la desviación estándar del ruido como cabe esperar. Tras el filtrado de las imágenes, se mantiene esta decadencia del ratio; sin embargo, este es mayor entre las imágenes filtradas y la original, lo cual indica que el algoritmo de filtrado implementado funciona correctamente. Además, se puede observar en los datos extraídos que, cuanto mayor es la desviación del ruido añadido a la imagen original, mayor es el aumento del PSNR tras filtrar la imagen. Esto también es un buen indicativo de que el algoritmo funciona correctamente, puesto que en estos casos se realiza una mayor reconstrucción de la imagen en comparación a imágenes débilmente contaminadas con ruido.

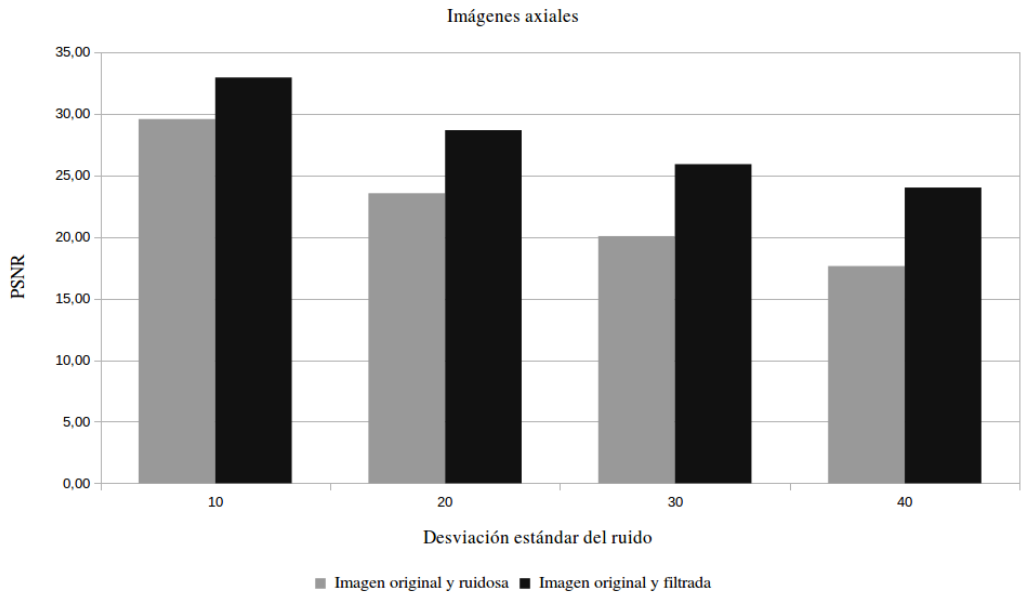


Figura 6.4: Resultados obtenidos para las imágenes axiales (elaboración propia)

Con el fin de dotar los resultados obtenidos de mayor visualidad, se muestra una gráfica de barras de los resultados extraídos para las imágenes axiales. Por último, destacar que los resultados han sido similares para todas las imágenes testadas.

6.2. Rendimiento

Con el fin de obtener una cuantificación del rendimiento del algoritmo paralelo desarrollado, se ha calculado la ganancia o el *speedup* del algoritmo paralelo ejecutado en diferentes unidades de cómputo con respecto al algoritmo secuencial. La fórmula utilizada es la siguiente

$$S = \frac{T_{secuencial}}{T_{paralelo}} \tag{6.3}$$

donde $T_{secuencial}$ es el tiempo de ejecución del algoritmo secuencial y $T_{paralelo}$ es el tiempo de ejecución del algoritmo paralelo. Procede destacar que el tiempo de ejecución T es el tiempo total de ejecución de todo el proceso; lo cual conlleva la lectura de la imagen, el acolchado, el cómputo de la imagen resultante y su posterior escritura.

Cabe destacar que los resultados que se muestran a continuación se han obtenido con una ventana de búsqueda r de dimensiones 21×21 (radio 10) y una ventana de similitud f de

dimensiones 5×5 (radio 2). La imagen de entrada tiene unas dimensiones de 660×1024 píxeles.

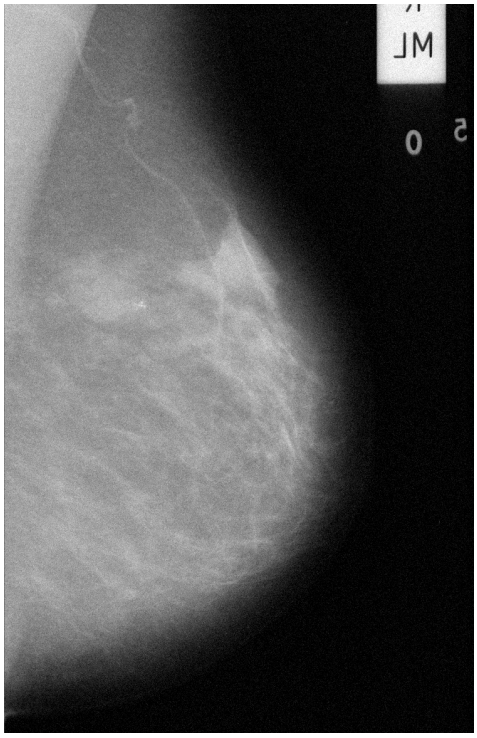


Figura 6.5: Mamografía utilizada para el estudio de la ganancia del algoritmo paralelo [19]

Se han definidos varios casos de estudio para el cálculo de la ganancia. En primer lugar, hay que tener en cuenta que, como se ha mencionado en la sección 4.2, el clúster bajo el que se han realizado las pruebas posee un total de veintiséis nodos con doce cores cada uno. De esta manera, los casos de estudios consisten en ejecutar el código en un único nodo y variando el número de cores utilizados hasta utilizar los doce. Además, se estudian otros casos variando el número de nodos empleando los doce nodos en cada uno de ellos. Esto es, en definitiva, realizar pruebas $\forall K \in \{1,2,3,4,5,6,7,8,9,10,11,12,24,36,48,60,72,84\}$, donde K es el número de unidades de cómputo.

Primero se estudian los casos de aquellas ejecuciones en un mismo nodo. A continuación, se muestran los resultados obtenidos para dichos casos de estudios, que son el tiempo de ejecución junto con la ganancia con respecto al algoritmo secuencial. Se entiende que la ejecución secuencial corresponde a aquella ejecutada en un único core o unidad de cómputo.

Número de cores	Tiempo de ejecución (s)	Ganancia
1	147.026	1
2	72.309	2.033
3	50.169	2.931
4	39.359	3.736
5	31.506	4.667
6	26.753	5.496
7	22.960	6.404
8	20.722	7.095
9	18.059	8.141
10	17.453	8.424
11	15.433	9.527
12	14.346	10.249

Tabla 6.4: Tiempo de ejecución y ganancia en los cores de un nodo (elaboración propia)

Como se puede observar en los resultados que se muestran en la tabla, se tiene que el algoritmo secuencial filtra la imagen en 147.026 segundos. Esto es equivalente a 2 minutos y 27 segundos, lo cual supone un tiempo prohibitivo para aplicar el algoritmo en tiempo real. Si se le añade la necesidad de filtrar varias imágenes, con mayores resoluciones y con ventanas de búsqueda más grandes, los tiempos aumentarían de manera muy considerable. Convirtiendo al algoritmo en una opción no viable.

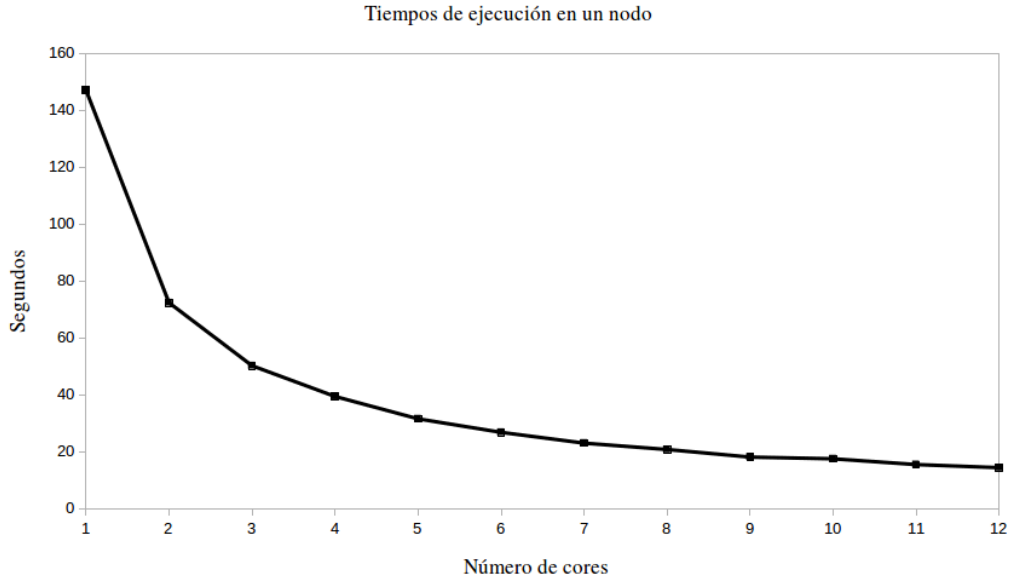


Figura 6.6: Gráfica de los tiempos de ejecución en un nodo (elaboración propia)

En la gráfica anterior se ve con claridad cómo disminuye el tiempo de ejecución a medida que aumentan las unidades de cómputo, como cabría esperar. Debido a que el algoritmo paralelo divide el dominio del problema en subdominios independientes a priori de igual tamaño, teóricamente se puede obtener una paralelización perfecta. Por esto, a continuación se muestra una gráfica comparativa entre el tiempo de ejecución obtenido y el tiempo de ejecución ideal suponiendo una paralelización perfecta, donde

$$T_{ideal} = \frac{T_{secuencial}}{K} \tag{6.4}$$

y K es el número de cores.

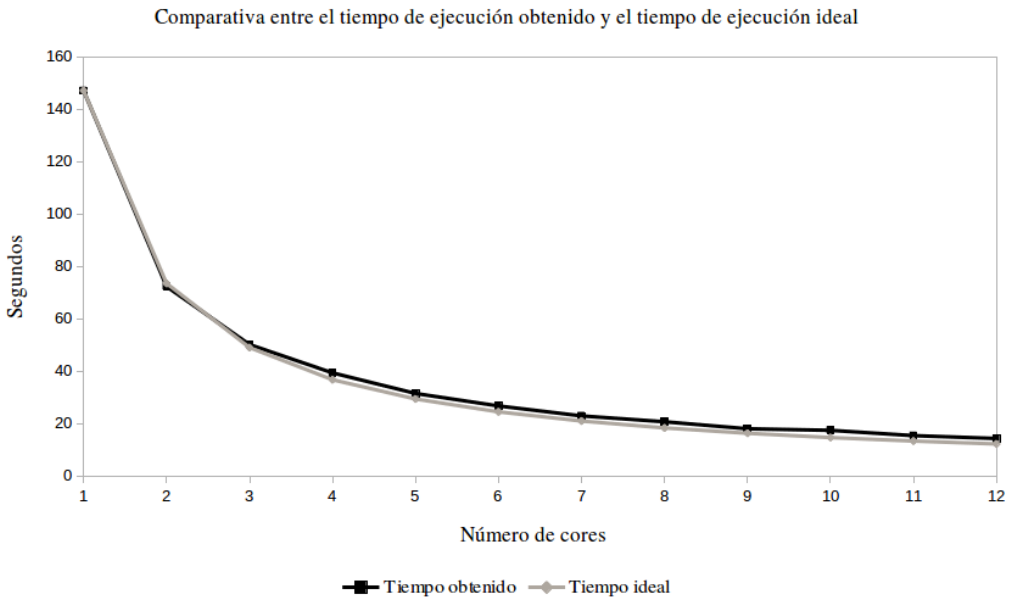


Figura 6.7: Gráfica de la comparativa entre el tiempo obtenido y el ideal (elaboración propia)

Se observa que el tiempo obtenido se ajusta muy bien al tiempo ideal. Lo cual indica que se ha diseñado el algoritmo correctamente. El pequeño aumento de tiempo es inherente a este tipo de paralelización, pues requiere de una comunicación mediante el paso de mensaje entre las diferentes unidades de cómputo.

El siguiente conjunto de casos de estudio precisamente ilustra bien esa pérdida de velocidad en tiempos de ejecución debida al aumento de los cores en los que se ejecuta el algoritmo. Se prueba el algoritmo paralelo con varios nodos utilizando todos sus cores, de manera que se realizan grandes cantidades de pasos de mensajes.

Número de cores	Tiempo de ejecución (s)	Ganancia
1	147.026	1
24	9.237	15.917
36	7.669	19.171
48	6.103	24.091
60	3.931	37.402
72	5.320	27.636
84	6.367	15.696

Tabla 6.5: Tiempo de ejecución y ganancia en los cores de varios nodos (elaboración propia)

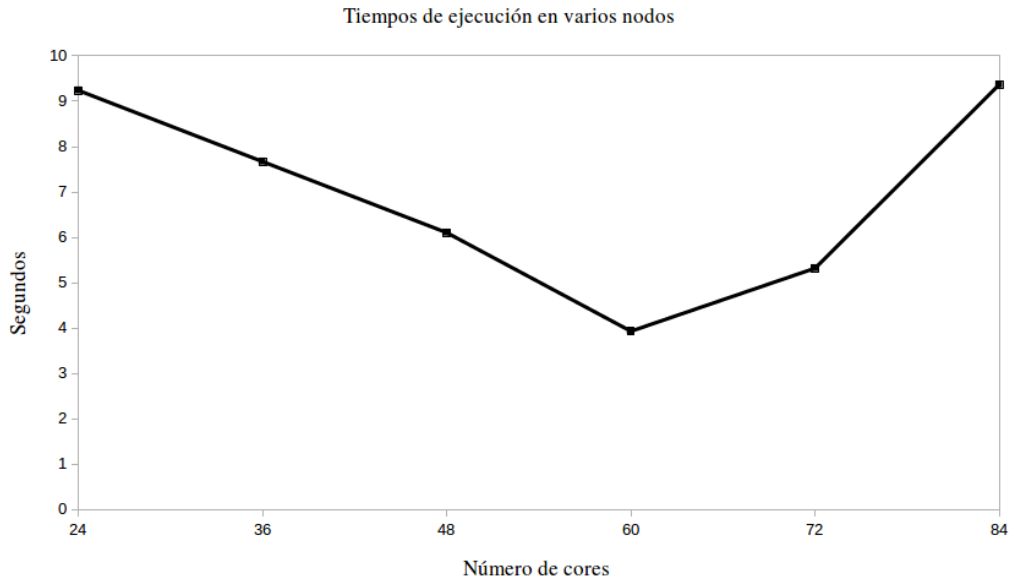


Figura 6.8: Gráfica de los tiempos de ejecución en varios nodos (elaboración propia)

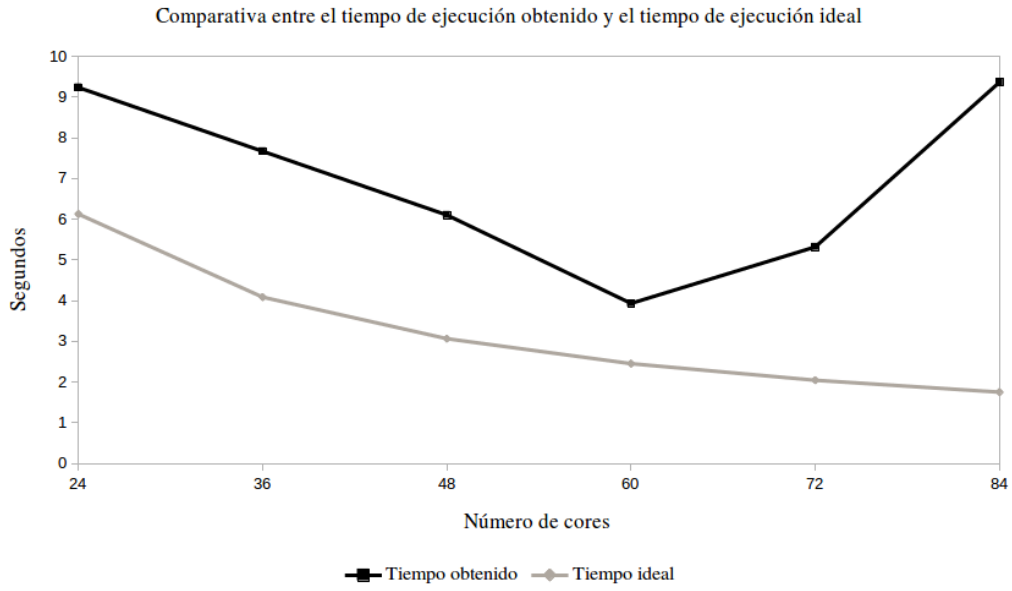


Figura 6.9: Tiempo obtenido y tiempo ideal en varios nodos (elaboración propia)

En las gráficas anteriores se muestra claramente cómo el tiempo de ejecución obtenido y el ideal difieren más que en la gráfica 6.7 debido al aumento considerable de las unidades de cómputo. Es más, se produce un punto de inflexión entre 60 y 72 nodos en el que deja de ser óptimo paralelizar más la ejecución del algoritmo. El volumen de datos no es lo suficientemente grande como para que la pérdida por comunicaciones rentabilice la paralelización.

A continuación se representan los datos obtenidos en cuanto a la ganancia.

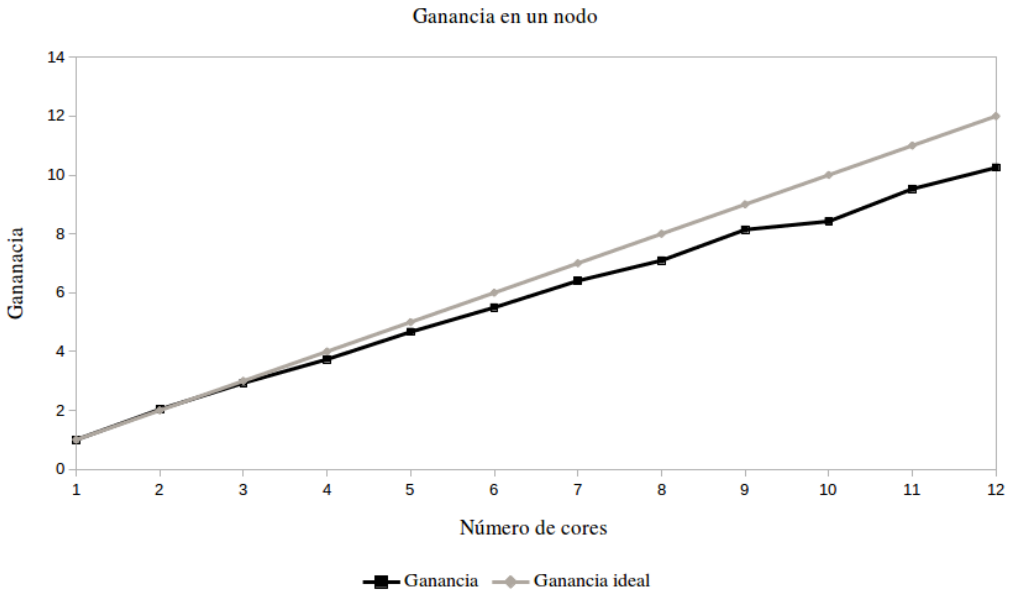


Figura 6.10: Ganancia obtenida y ganancia ideal en un nodo (elaboración propia)

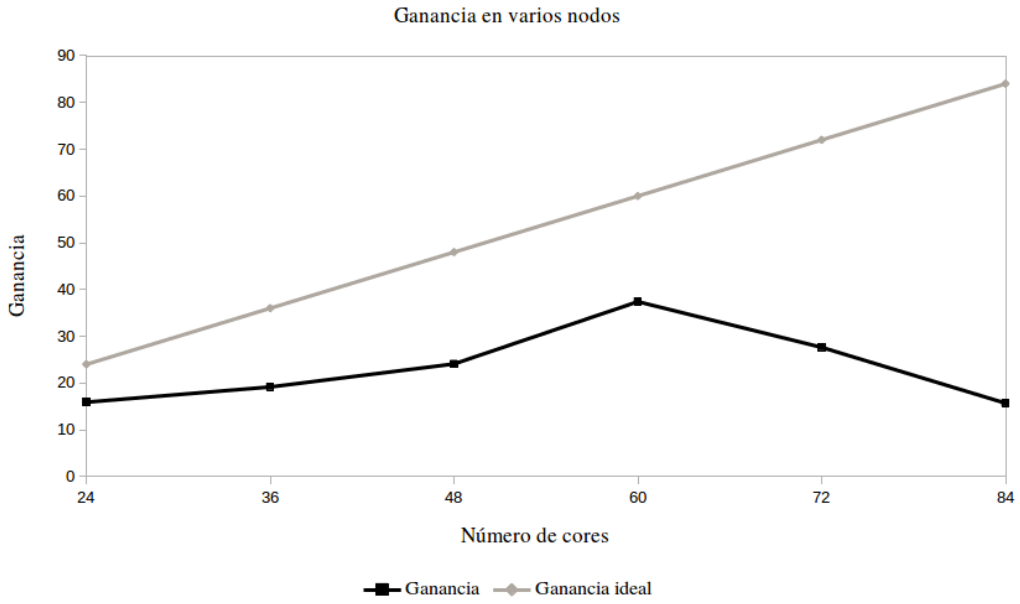


Figura 6.11: Ganancia obtenida y ganancia ideal en varios nodos (elaboración propia)

Hay que tener en cuenta que la ganancia ideal según una paralelización perfecta sería equivalente al número de cores en los que se ejecuta el algoritmo, puesto que nos indica cuántas veces es más rápida la ejecución paralela con respecto a la ejecución secuencial. De estas gráficas se pueden extraer las mismas conclusiones que de las anteriores, simplemente enfoca los resultados desde otro punto de vista.

En un primer momento, con un único nodo, la ganancia obtenida se ajusta muy bien a la ideal, a pesar de que se van distanciando con el aumento de cores. Se puede observar como, en los datos obtenidos, el algoritmo paralelo llega a ser 37.402 veces más rápido que el secuencial con 60 cores; sin embargo, con 72 el algoritmo solo es 27.636 veces más rápido y con 84 únicamente 15.696 veces más rápido, debido a lo mencionado anteriormente en el análisis de los tiempos de ejecución.

7 Conclusiones

En este trabajo se ha propuesto un algoritmo paralelo basado en el algoritmo de medias no locales capaz de filtrar el ruido en imágenes médicas. De esta manera, se le permite al usuario final eliminar el ruido de imágenes con grandes resoluciones en el menor tiempo posible, pudiéndose aplicar el algoritmo en tiempo real; lo cual es imposible en la versión secuencial del mismo. Los experimentos realizados a lo largo del desarrollo ponen de manifiesto la utilidad del algoritmo.

En primer lugar, se ha comprobado que, como se espera de un algoritmo de filtrado, el PSNR entre la imagen original y la imagen filtrada aumenta con respecto al PSNR entre la imagen original y la imagen ruidosa. Este aumento es considerable, lo cual es indicativo del aumento de la calidad de la imagen.

En segundo lugar, con respecto a los tiempos de ejecución, se muestra que efectivamente el algoritmo es más rápido conforme más unidades de cómputo se utilicen. Se realiza una paralelización prácticamente perfecta del problema. Únicamente se producen pérdidas debido al paso de mensajes entre nodos, los cuales son inherentes al estándar MPI empleado para paralelizar el problema.

Estas características hacen del algoritmo una herramienta que podría utilizarse de manera generalizada en hospitales y centros médicos para obtener imágenes TC de calidad reduciendo las dosis de radiación ionizante que se ha demostrado que puede causar maliginidad en los pacientes que son expuestos a esta.

8 Líneas futuras de trabajo

Se han estudiado varias posibles ampliaciones futuras durante el desarrollo del proyecto. Entre ellas, destacan aquellas orientadas a la mejora de la calidad de la imagen filtrada y aquellas orientadas a aumentar la ganancia del algoritmo.

Es de especial interés en el ámbito de la medicina mejorar la calidad de la imagen resultante. Por esto, dado que el algoritmo empleado para el filtrado de ruido tiende a suavizar toda la imagen, una posible ampliación consistiría en modificar el algoritmo para conseguir una mejor conservación de los bordes. En primer lugar se aplicaría un algoritmo capaz de detectar los bordes de la imagen y en aquellos píxeles que se sitúen en ellos aplicar un método de filtrado diferente.

Como se ha comentado anteriormente, es muy útil disminuir el tiempo de ejecución del algoritmo para facilitar la obtención de imágenes filtradas en el menor tiempo posible. Una ampliación para mejorar este aspecto consistiría en aplicar paralelización a nivel de hilos mediante OpenMP [20]. Es una interfaz de programación de aplicaciones para la programación multiproceso de memoria compartida que puede integrarse en programas escritos en C, como es el caso. Se basa en el modelo *fork-join*, donde una tarea muy pesada se divide en un número de hilos (*fork*) con menor peso, para luego unir sus resultados en uno único (*join*). De esta manera, se aplicaría paralelización mediante MPI para dividir el problema en diferentes unidades de cómputo junto con OpenMP para dividir el problema de cada unidad de cómputo en sus diferentes hilos.

También existe la posiblidad de paralelizar a nivel de hilos utilizando CUDA [21]. Es una plataforma de cálculo paralelo y un modelo de programación desarrollado por NVIDIA para la computación general en unidades de procesamiento gráfico (GPU). Con CUDA, los

desarrolladores pueden acelerar considerablemente las aplicaciones de cálculo aprovechando la potencia de las GPU. En las aplicaciones aceleradas por la GPU, la parte secuencial de la carga de trabajo se ejecuta en la CPU, que está optimizada para un rendimiento con un único subproceso, mientras que la parte de la aplicación que requiere un uso intensivo de la computación se ejecuta en miles de núcleos de GPU en paralelo.

Otras líneas de investigación futuras podrían estar centradas en la flexibilidad de la aplicación. Permitiendo así filtrar imágenes TC en 3D en las cuales el tiempo de cómputo es mucho más elevado. También existe la posibilidad de filtrar imágenes en RGB o en diferentes formatos, no únicamente PGM. Así como diseñar una interfaz amigable para el usuario y facilitar la ejecución en paralelo del programa.

Referencias

[1] David J Brenner and Eric J Hall. Computed tomography—an increasing source of radiation exposure. *New England Journal of Medicine*, 357(22):2277–2284, 2007.

[2] Andrew J Einstein, Milena J Henzlova, and Sanjay Rajagopalan. Estimating risk of cancer associated with radiation exposure from 64-slice computed tomography coronary angiography. *Jama*, 298(3):317–323, 2007.

[3] David J Brenner. Estimating cancer risks from pediatric CT: going from the qualitative to the quantitative. *Pediatric radiology*, 32(4):228–231, 2002.

[4] Juan Carlos Ramírez Giraldo, Carolina Arboleda Clavijo, and Cynthia H McCollough. Tomografía computarizada por rayos x: fundamentos y actualidad. *Revista Ingeniería Biomédica*, 2(4):54–66, 2008.

[5] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 60–65. IEEE, 2005.

[6] Abdessamad Ben Hamza, Pedro L Luque-Escamilla, José Martínez-Aroza, and Ramón Román-Roldán. Removing noise and preserving details with relaxed median filters. *Journal of Mathematical Imaging and Vision*, 11(2):161–177, 1999.

[7] Mukesh C Motwani, Mukesh C Gadiya, Rakhi C Motwani, and Frederick C Harris. Survey of image denoising techniques. In *Proceedings of GSPX*, pages 27–30, 2004.

[8] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In

Iccv, volume 98, page 2, 1998.

[9] Juan Carlos Ramírez Giraldo, Joel J Fletcher, and Cynthia H McCollough. Reducción del ruido en imágenes de tomografía computarizada usando un filtro bilateral anisotrópico. *Revista Ingeniería Biomédica*, 4(7):55–62, 2010.

[10] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. "O'Reilly Media, Inc.", 2012.

[11] Dennis M Ritchie, Brian W Kernighan, and Michael E Lesk. *The C programming language*. Bell Laboratories, 1975.

[12] William D Gropp, William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[13] John W Eaton, David Bateman, and Soren Hauberg. *GNU Octave manual*. Network Theory Ltd. Bristol, UK, 2002.

[14] Richard M Stallman et al. Using the GNU Compiler Collection. *Free Software Foundation*, 4(02), 2003.

[15] Instituto Universitario de Investigación Informática de la Universidad de Alicante. Cluster de Computación - IUII. `https://web.ua.es/es/cluster-iuii/cluster-de-computacion-iuii.html`, 2019.

[16] Jef Poskanzer. PGM format specification. `http://netpbm.sourceforge.net/doc/pgm.html`, 2016.

[17] Alain Hore and Djemel Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369. IEEE, 2010.

[18] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[19] J. Suclking. The mammographic image analysis society digital mammogram database.

Digital Mammo, pages 375–386, 1994.

[20] Rohit Chandra, Leo Dagum, David Kohr, Ramesh Menon, Dror Maydan, and Jeff McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[21] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley Professional, 2010.

[22] Marc Snir, William Gropp, Steve Otto, Steven Huss-Lederman, Jack Dongarra, and David Walker. *MPI–the Complete Reference: The MPI core*, volume 1. MIT press, 1998.

[23] Michael A King, Paul W Doherty, Ronald B Schwinger, and Bill C Penney. A Wiener filter for nuclear medicine images. *Medical physics*, 10(6):876–880, 1983.