PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO E IMPLEMETACIÓN DEL FILTRO MEDIANO DE DOS DIMENSIONES PARA ARQUITECTURAS SIMD

Tesis pata optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Ricardo Miguel Sánchez Loayza

ASESOR: Paul Antonio Rodríguez Valderrama

Lima, marzo del 2011

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título	:	Diseño e Implementación del Filtro Mediano de Dos Dimensiones para Arquitecturas SIMD
Área	:	Procesamiento Digital de Imágenes
Asesor	:	Paul Antonio Rodríguez Valderrama
Alumno	:	Ricardo Miguel Sánchez Loayza
Código	:	20030447.N
Fecha	:	29 de Mayo del 2009

712

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Facultad de Ciencias e Ingeniería
02 JUN. 2009
DANIEL TORREALVA DAVILA
DECANO.

Descripción y Objetivos

El filtro mediano es un filtro no lineal basado en la mediana estadística y fue propuesto por primera vez en 1974 para ser utilizado en el área de procesamiento digital de imágenes. Su característica principal es la eliminación de ruido impulsivo sin alterar la información de la imagen (bordes, etc.) de manera significativa. Debido a esta propiedad, es utilizado como parte del preprocesamiento de imágenes para análisis de mayor complejidad, como pueden ser la identificación de objetos, reconocimiento de caracteres, etc. Su principal desventaja es su alto costo computacional y las optimizaciones propuestas para diminuir este costo se basan, en su mayoría, en los paradigmas tradicionales de programación.

Los microprocesadores modernos, tales como los Intel Core, AMD Athlon, PowerPC G5, el Procesador Cell, cuentan con la unidad SIMD (Single Instruction, Multiple Data - Instrucción Única, Datos Múltiples), la cual se diferencia de las unidades ALU (Unidad Lógico Aritmética) y FPU (Unidad de Coma Flotante) en que sus operandos son vectores (de N escalares). El desempeño computacional de implementaciones que utilizan la unidad SIMD son, de modo teórico, N veces superior a aquellas que sólo utilizan la unidad FPU (ó ALU/FPU).

El presente trabajo tiene como objetivo el diseño y desarrollo de una implementación computacionalmente eficiente del filtro mediano para la arquitectura SIMD. Se comparará la eficacia computacional del presente desarrollo con implementaciones eficientes conocidas.

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
SECCION ELECTRICIDAD Y ELECTRONICA
Ing. ANDRES FLORES ESPINOZA
Coordinador de la Especialidad de Ingeniería Electrónica

MÁXIMO 100 PÁGINAS

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título : Diseño e Implementación del Filtro Mediano de Dos Dimensiones para Arquitecturas SIMD

Índice

PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
SECCION ELECTRICIDAD Y ELECTRONICA

Ing. ANDRES FLORES ESPINOZA
Coordinador de la Especialidad de Ingeniería Electrónica

MÁXIMO 100 PÁGINAS

Resumen

El filtro mediano es una de las operaciones básicas en el procesamiento de imágenes digitales, su función es la de eliminar el ruido impulsivo sin alterar la información de la imágen. A pesar de estas características, su uso se ve restringido debido al alto costo computacional del filtro. Las propuestas tradicionales de solución, consisten en disminuir la complejidad del algoritmo del filtro mediano, y en vectorizar los algoritmos existentes. Esta vectorización se realiza al utilizar las unidades SIMD (*Single Instruction Multiple Data* - Instrucción Única Múltiples Datos) de los procesadores modernos. Ésta les permite realizar una misma operación a un conjunto, o vector, de datos de manera simultánea, con lo que se obtiene un mejor desempeño computacional.

En el presente trabajo se implementa el filtro mediano con el algoritmo vectorial propuesto por Kolte [1], el cual aprovecha las ventajas de las unidades SIMD. La eficiencia computacional de la implementación realizada se compara con el algoritmo Filtro Mediano en Tiempo Constante, propuesto recientemente por Perreault [2], el cual presenta una complejidad de $O(1)$. La implementación realizada es 75 y 18.5 veces mas rápida que la implementación de referencia, para áreas de análisis de 3×3 y 5×5 respectivamente. Se concluye además que la vectorización de un algoritmo no necesariamente obtiene los mismos resultados que un algoritmo diseñado específicamente para ser implementado en unidades vectoriales [3].

Índice general

Índice de figuras

Introducción

El presente trabajo tiene como objetivo el implementar de una manera eficiente el filtro mediano, con el uso de nuevos paradigmas de programación, y aplicarlo al procesamiento digital de imágenes. Se utilizará la arquitectura SIMD (*Single Instruction Multiple Data - Instrucción Única Datos Múltiples*) para mejorar el rendimiento computacional del algoritmo del mencionado filtro.

El filtro mediano es un filtro no lineal propuesto por primera vez en 1974 [5], y su característica principal es la de disminuir el ruido del tipo impulsivo (comúnmente llamado *ruido sal y pimienta*) [6] sin alterar la información de la imagen. Debido a esta propiedad se utiliza como parte del preprosesamiento de imágenes para análisis de mayor complejidad, como son el reconocimiento óptico de caracteres, la identificación de objetos y rostros, etc. Sin embargo por ser un filtro no lineal la implementación de éste tiene una gran complejidad, lo cual lo convierte en costoso desde el punto de vista computacional. Se han propuesto diferentes ideas para mejorar su desempeño y asi beneficiarse con sus ventajas, sin embargo la mayoría de éstas se basan en el paradigma tradicional de programación que tiene como base los procesadores convencionales. Los procesadores modernos cuentan con una unidad SIMD, adicional a las unidades que normalmente se incluyen (ALU: *Arithmetic Logic Unit – Unidad Lógico Aritmética*, FPU: *Float-Point Unit – Unidad de Punto Flotante*), que permiten la ejecución de una misma instrucción a varios datos [7][8] con lo que se crea un nuevo paradigma de programación.

La unidad SIMD posee un grupo de registros en los que los datos se almacenan como arreglos de valores y las operaciones que la unidad realiza se hacen entre estos arreglos o vectores. Es así como se pueden procesar una mayor cantidad de datos en menor tiempo y los algoritmos que realizan una misma secuencia de operaciones a gran cantidad de información se ven muy beneficiados [9]. El filtro mediano se puede implementar de manera que aproveche al máximo la capacidad de operar múltiples datos a la vez en las unidades SIMD.

Capítulo 1

El Filtro Mediano de Dos Dimensiones: Teoría y Algoritmos

El filtro mediano es importante [6] ya que disminuye el ruido impulsivo de la imagen sin deformar de manera significativa la data que contiene la imagen, por esta característica su uso esta muy extendido y en él se basan sistemas de reconocimiento de caracteres, segmentación de imágenes, análisis morfológico, entre otros. Su funcionamiento consiste en obtener el valor mediano en un área de análisis y esta operación se repite por cada píxel de la imagen [5]. El algoritmo clásico [10] requiere hallar el histograma de $L \times L$ datos (área de análisis). Esta operación se repite $N \times M$ veces (tamaño de la imagen de $N \times M$ pixeles). El tiempo de ejecución se incrementa de manera no lineal al incremento del área de análisis. Es por ello que la optimización de este algortimo, con el uso del procesamiento en paralelo de datos que ofrecen las unidades SIMD, es de gran importancia.

1.1. Definición

El filtro mediano es una técnica del procesamiento digital de señales propuesta por Tukey [5], y se basa en utilizar la mediana estadística para eliminar el ruido impulsivo de una imagen, sin alterar de manera significativa la información que ésta contiene. En su forma unidimensional, el filtro mediano consiste en el desplazamiento de una ventana de análisis que agrupa un número impar de datos. El dato central de la ventana es reemplazado por el valor de la mediana de los datos que se agrupan en dicha ventana.

La mediana de una secuencia discreta de de datos $a_1, a_2, \ldots, a_N$, para N impar, es el miembro de la secuencia que tiene ${}^{(N-1)}/_2$ elementos menores o iguales a él en valor, y ${}^{(N-1)}/_2$ elementos mayores o iguales a él en valor. Por ejemplo, si los valores de los datos dentro de una ventana de cinco elementos son $0{,}1, 0{,}2, 0{,}9, 0{,}4, 0{,}5$, el dato central sería reemplazado por el valor $0{,}4$, que es el valor central de la secuencia ordenada $0{,}1, 0{,}2, 0{,}4, 0{,}5, 0{,}9$ [11].

El concepto del filtro mediano puede ser fácilmente extendido a dos dimensiones al utilizar una ventana bidimensional con alguna forma deseada como pueden ser un rectángulo o una aproximación discreta a un círculo. A esta región de análisis se le denomina *kernel* y a cada elemento del arreglo bidimensional, dado que éste representa una imagen digitalizada,

se le denomina *pixel*. Para cada pixel de la imagen, se toma una región centrada en dicho pixel, se obtiene la mediana de estos elementos y el resultado se almacena en una nueva imagen (Figura 1.1).

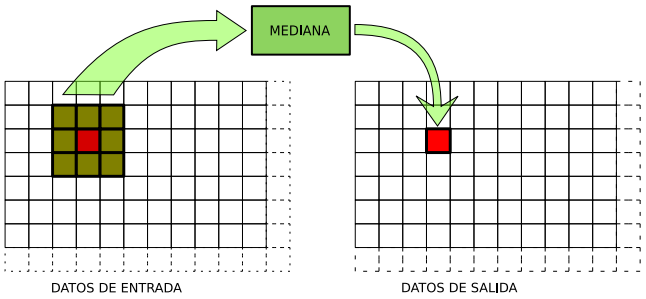


Figura 1.1: Aplicación del filtro mediano bidimensional.

1.2. Características

Una de las características principales del filtro mediano es su capacidad de eliminar el ruido impulsivo, que consiste en valores dentro de un conjunto de datos que difieren de gran manera del resto de los elementos. A éste ruido se le denomina también como ruido *sal y pimienta*, debido a que se presenta como puntos negros o blancos en la imagen. Otra propiedad de este filtro es que no altera señales del tipo escalón, las cuales se pueden considerar como los bordes de los objetos dentro de una imagen. La Figura 1.2 muestra una comparación entre la aplicación de un filtro de suavizado (pasabajos, promedio o media) y del filtro mediano, para algunas funciones discretas. La ventana que se utilizó agrupa cinco muestras. Se puede notar como el filtro mediano elimina las señales del tipo impulsivo y no modifica de manera significativa el resto de señales.

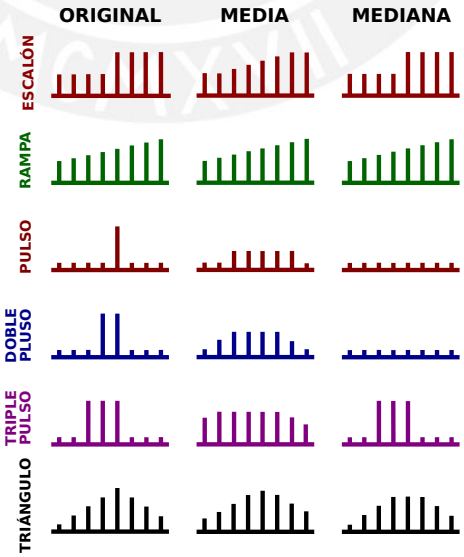
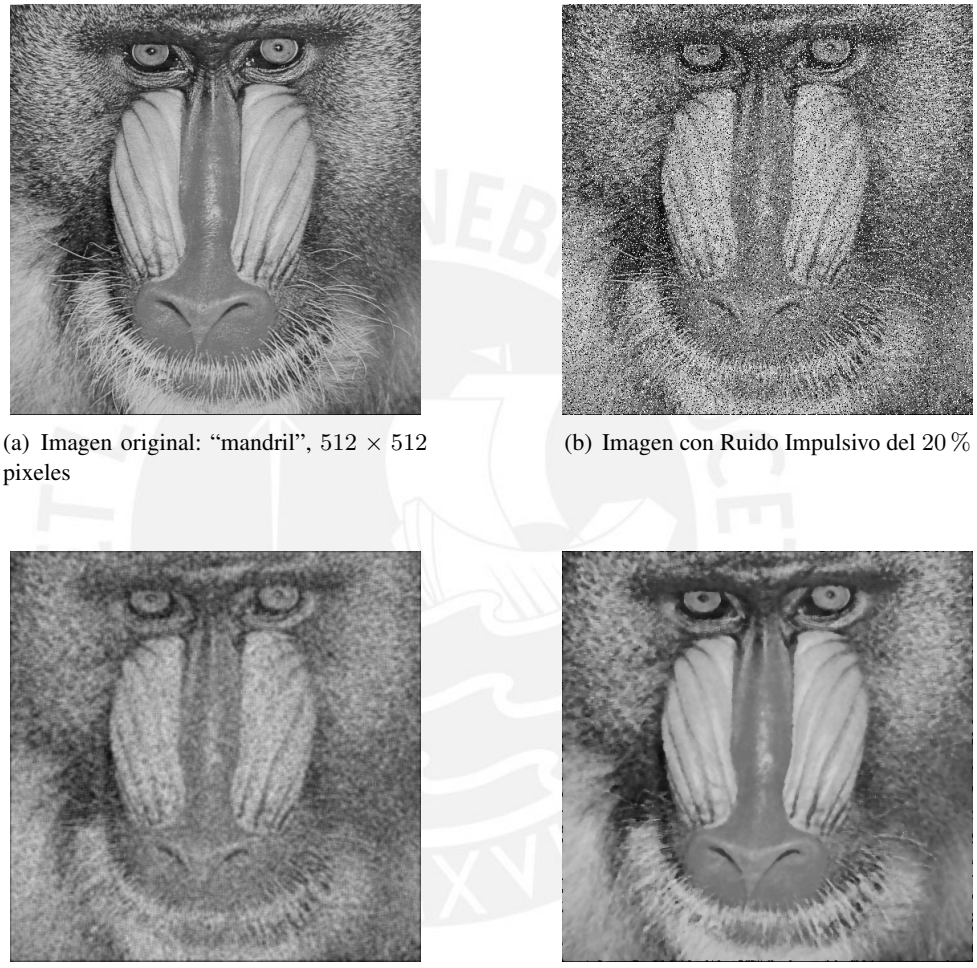


Figura 1.2: Comparación de los filtros promedio y mediana de cinco elementos aplicados a funciones discretas unidimensionales.

Otro ejemplo de estas características se puede observar en la figura 1.3. La imagen original (Figura 1.3(a)) es corrompida con ruido impulsivo, donde el $1{,}6\,\%$ de los pixeles son transformados (Figura 1.3(b)). La imagen mas el ruido es procesada mediante el filtro mediano (Figura 1.3(d)) y un filtro pasabajos (Figura 1.3(c)). Se puede notar que el filtro pasabajos no elimina el ruido impulsivo, además, que suaviza los bordes de las figuras incluidas en la imagen. En cambio, filtro mediano mantiene los bordes de la imagen y elimina el ruido impulsivo, de esta manera se obtiene una imagen muy similar a la imagen original.



(a) Imagen original: "mandril", 512×512 pixeles

(b) Imagen con Ruido Impulsivo del $20\,\%$

(c) Filtro Pasabajos de 5×5

(d) Filtro mediano de 5×5

Figura 1.3: Aplicación de los filtros pasabajos y mediano a una imagen con ruido impulsivo.

1.3. Algoritmos e Implementaciones

La proposición original del filtro mediano, realizada en 1974 [5], indica que se debe obtener la mediana del conjunto de datos que se encuentren dentro del área de análisis. Entonces, el aplicar el filtro mediano esta ligado a obtener el valor mediano de un conjunto de datos y éste se obtiene al ordenar los elementos del conjunto, ya sea de manera creciente o decreciente, y el valor de la mediana será el elemento central del arreglo. De esta manera,

el algoritmo para obtener el filtro mediano depende de un algoritmo de ordenamiento que se repite por cada elemento en la imagen (Figura 1.4).

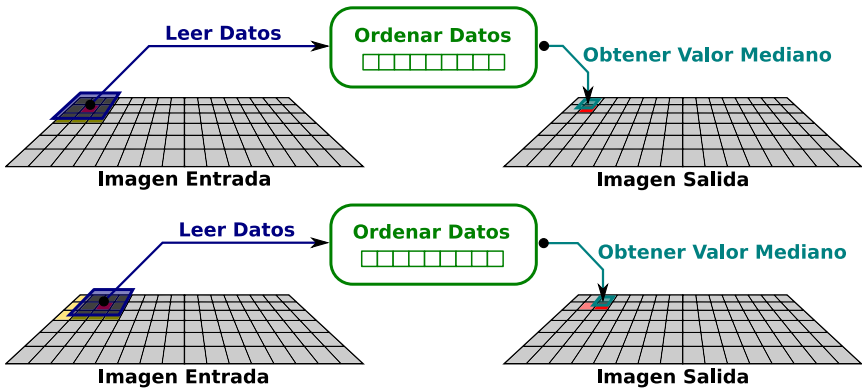


Figura 1.4: Esquema de la implementación clásica del filtro mediano. Se muestran dos iteraciones de pixeles contiguos

El algoritmo original presenta una desventaja que convierte al filtro mediano poco práctico para aplicaciones en imágenes grandes o en tiempo real: tiene un alto costo computacional, lo que conlleva a un tiempo grande de ejecución. Uno de los factores que influyen en este alto costo es el ordenamiento de datos, proceso que se realiza con el algoritmo llamado *buble sort*[12], el cual contiene dos bucles en su implementación. El tiempo de ejecución de este algoritmo se incrementa de manera cuadrática cada vez que el área de análisis crece. Por ejemplo, supongamos que para un kernel de radio r el tiempo de ejecución del ordenamiento es de 3 unidades de tiempo, para un kernel de radio $r+1$ el tiempo será de 9 unidades de tiempo. Esto define que la complejidad del algoritmo de *buble sort* sea de $O(r^2)$ (esta en función del cuadrado del radio del kernel).

El otro factor que incrementa el tiempo de ejecución del filtro mediano es la manera en que se accesa a los datos. Se puede observar que se leen r^2 pixeles de memoria para un kernel cuadrado por cada pixel de la imagen. Se deduce entonces que cada pixel de la imagen es leído r^2 veces. Si se tiene en cuenta el algoritmo de ordenamiento y el de acceso a la información, se obtiene que la complejidad de este algoritmo es de $O(r^2 \log r)$.

Es por esta razón que el uso del filtro mediano se ve restringido a imágenes pequeñas y se evita su uso en aplicaciones en tiempo real. El diseño de un algoritmo eficiente para su cálculo es el objetivo de muchas investigaciones.

1.3.1. El Filtro Mediano de Huang

Huang *et al.*[10] propone en 1979 un método para disminuir el tiempo que toma la aplicación del filtro mediano a una imagen. En su propuesta disminuye la cantidad de veces que cada pixel es leído de la memoria y reduce el tiempo que se requiere para calcular la mediana de un conjunto de datos. Ésto lo consigue con el uso de un histograma y memoria en la que éste se almacena.

El histograma es una herramienta estadística en la que se registra el número de ocurrencias de cada valor en un conjunto de datos en un arreglo o vector. Para imágenes, el tamaño

de arreglo que contiene el histograma es el del número de colores, y generalmente es de 256 elementos, pues las imágenes son de 8-bits. En un histograma los elementos estan ordenados, asi que el cálculo de la mediana es inmediato, solo se necesita sumar las ocurrencias hasta que éste valor sea mayor o igual a $(n-1)/2$, donde n es el número de elementos del área de análisis. El algoritmo del cálculo de la mediana a través del histograma se muestra a continuación:

Algoritmo 1: Cálculo de la mediana por medio del Histograma

Entrada: X es una Región de una imagen de tamaño $m \times n$ y con d colores.
Salida : $mediana$ es la mediana de la región.
Inicializar el arreglo H, de d elementos, con 0;
Para $i = 1$ *hasta* m **hacer**
    Para $j = 1$ *to* n **hacer**
        sea $indice = X_{i,j}$;
        sea $H_{indice} = H_{indice} + 1$;
sea $mediana = 0$;
Mientras $acumulador \leq \frac{(m \times n)-1}{2}$ **hacer**
    sea $acumulador = acumulador + H_{mediana}$;
    sea $mediana = mediana + 1$;

Para reducir las lecturas que se deben realizar a cada píxel se almacena el histograma obtenido de la primera región de análisis en un sector de la memoria y para el siguiente región solo retira del histograma almacenado los pixeles que pertenecieron al área anterior y no lo hacen al actual, y se agregan los píxeles que pertenecen al área actual y no lo hacían en el anterior. Es así que para el cálculo de la mediana de una serie de pixeles adyacentes las veces un píxel se lee desde la memoria se reduce de $(2r+1)^2$ a $2r+1$, donde r es el radio de un kernel cuadrado (para un kernel de $3 \times 3, r = 1$) (Figura 1.5).

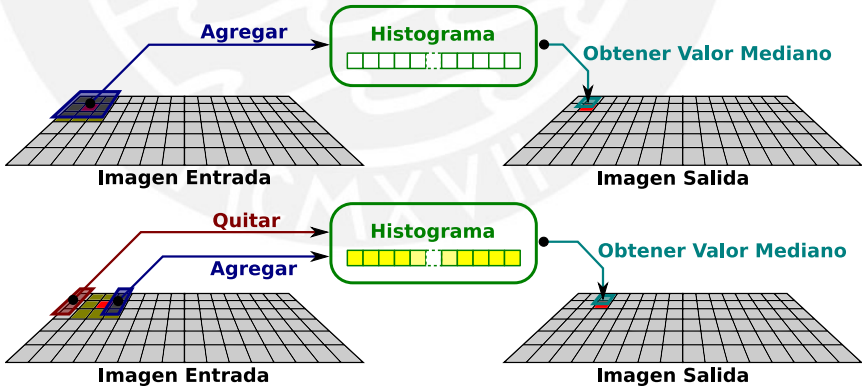


Figura 1.5: Desplazamiento de la región de análisis. Se observan los pixeles que se deben retirar y agregar del histograma.

El algoritmo final de Huang (Algoritmo 2) registra una complejidad de $O(r)$. Esto significa que el tiempo de ejecución se incrementa de manera lineal con los incrementos de r. Esta es la implementación mas utilizada, con ligeras modificaciones [13]. Se observa que para imágenes con mayor cantidad de colores se requiere más memoria para almacenar el histograma, y para kernels pequeños no es eficiente separar toda esa memoria.

Algoritmo 2: Filtro Mediano de Huang [10]

Entrada: X es una Imagen de tamaño $m \times n$, r es el radio del área de análisis
Salida : Y es una Imagen del mismo tamaño que X
Inicializar el histograma H del área de análisis
Para $i = 1$ *hasta* m **hacer**
  Para $j = 1$ *to* n **hacer**
    Para $k = -r$ *hasta* r **hacer**
      Quitar $X_{i+k,j-r-1}$ de H
      Agregar $X_{i+k,j+r}$ a H
    sea $Y_{i,j} = mediana(H)$

1.3.2. El Filtro Mediano de Chaudhuri

Chaudhuri[14] propone, en 1990, una mejora al agoritmo de Huang. Su modificación consiste en reemplazar el uso del histograma con una *lista enlazada*, con lo que disminuye el uso de la memoria.

Una lista enlazada es un conjunto de estructuras en las que por cada dato en la lista se incluye un puntero al elemento anterior y al elemento siguiente (Figura 1.6). Estas listas se caracterizan en que tienen un tiempo constante al agregar o retirar elementos en cualquier posición de la lista. Para agregar un elemento se toma el puntero a la estructura anterior y la siguiente y se almacenan en la estructura nueva. Luego se reemplaza, en la estructura anterior, el puntero al elemento siguiente con la dirección de la nueva estructura. Se realiza lo mismo con el puntero al elemento anterior de la estructura siguiente. Se procede de manera análoga para retirar un elemento de la lista.

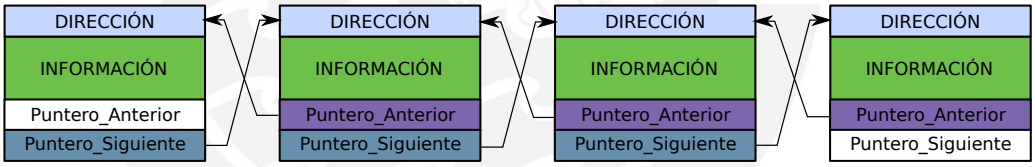


Figura 1.6: Diagrama de una lista enlazada.

No existe el direccionamiento indexado en estas listas, por lo que para accesar a un elemento en específico, se debe recorrer toda la lista hasta encontrarlo. Una vez ubicado el elemento, es sencillo agregar un elemento antes o después de éste, o retirarlo de la lista. Estas son las principales diferencias entre una lista enlazada y un arreglo de datos, ya que si bien en éstos últimos se puede utilizar índices para accesar a la información, agregar un dato en medio del arreglo es muy complicado, pues se debe asignar memoria y mover toda la información para habilitar el espacio que utilizará el nuevo dato.

De esta manera Chaudhuri plantea utilizar una lista enlazada ordenada para reemplazar el histograma en el algoritmo de Huang. Finalmente, esta implementación utiliza de manera más eficiente pero aumenta su complejidad a $O(r^2)$, y una consecuencia de esto es que su ejecución es más lenta que la expuesta anteriormente.

Capítulo 2

Arquitectura SIMD (Instrucción Única, Datos Múltiples ó Single Instruction Multiple Data)

El procesamiento de datos en paralelo es una forma de mejorar el desempeño los algoritmos y así disminuir el tiempo de ejecución de los mismos. De acuerdo a la clasificación propuesta por Flynn[15] las diversas arquitecturas de computadores se pueden agrupar en:

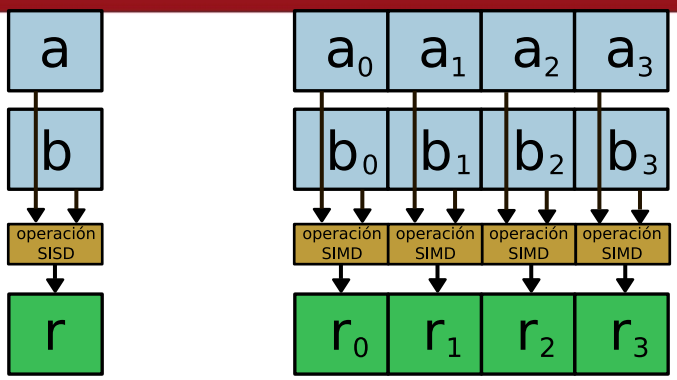
- SISD *(Single Instruction Single Data – Una Instrucción, Un Dato)*.
- MISD *(Multiple Instruction Single Data – Múltiples Instrucciones, Un Dato)*.
- SIMD *(Single Instruction Multiple Data – Una Instrucción, Múltiples Datos)*.
- MIMD *(Multiple Instrucion Multiple Data – Múltiples Intrucciones, Múltiples Datos)*.

La arquitectura mas utilizada es la SISD y se puede encontrar, por ejemplo, en los procesadores de los computadores personales, mientras las SIMD y MIMD se usan para supercomputadores, computadores de alto desempeño o en procesos en general que requieran gran capacidad de procesamiento.

2.1. Descripción

Instrucción Única, Datos Múltiples *(Single Instruction Multiple Data)*. Es una arquitectura de procesadores en la que se obtiene paralelismo a nivel de datos (Figura 2.1). Como su acrónimo indica, una sola instrucción del procesador afecta a varios registros del mismo [15].

Las unidades SIMD aprovechan una propiedad de los flujos de datos llamada **paralelismo de datos**. Se puede obtener paralelismo de datos cuando se tiene una cantidad de datos uniformes a los que se le realiza la misma operación. Un ejemplo clásico es la de invertir los colores de una imagen RGB. Se debe repetir la misma operación a cada uno de los elementos en un gran conjunto de datos [9].



(a) SISD (b) SIMD

Figura 2.1: Comparación entre las operaciones tradicionales SISD con las operaciones SIMD

Actualmente, gracias al avance tecnológico, los procesadores poseen unidades que les permiten realizar operaciones SIMD, esto quiere decir que permiten el procesar cierta de cantidad de datos con una sola instrucción. Las Tecnologías MMX, SSE de Intel [7], 3DNow! de AMD[8], AltiVec de Motorola/Freescale [16] son ejemplos de SIMD en procesadores comunmente utilizados en computadores personales de escritorio o de uso masivo. En los dispositivos como las cámaras digitales se pueden encontrar procesadores embebidos con capacidad SIMD: BlackFin [17], SHARC [18] de Analog Devices y todos los derivados de los procesadores ARM, a partir de su versión 6 [19], son ejemplos de procesadores que poseen unidades SIMD con optimizaciones para aplicaciones de audio y video en dispositivos de mano.

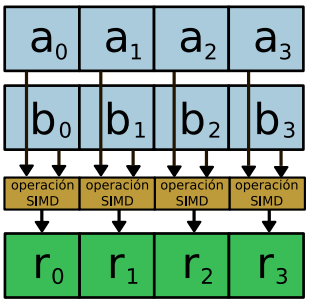
Debido a la disponibilidad de la unidad SIMD los algoritmos tradicionales que puedan beneficiarse de ésta deben ser reformulados para alcanzar mayor eficiencia. Los algoritmos que se utilizan en el procesamiento digital de imágenes son los candidatos inmediatos para la optimización que ofrece esta tecnología, pues estos procesos consisten en la iteración de un mismo conjunto de instrucciones en gran cantidad de data. Dentro de estos algoritmos el filtro Mediano resalta por su importancia en el preprocesamiento de imágenes.

2.2. Operaciones en la Unidad SIMD

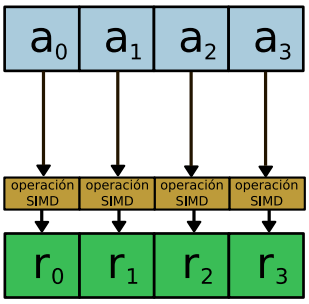
Para realizar operaciones en la Unidad SIMD es necesario cargar los datos a operar a los registros propios de la Unidad. Existen ciertas consideraciones a tomar en cuenta cuando se ingresa información a estos registros, pues los elementos del arreglo deben ser de la misma longitud. Adicionalmente es necesario que los datos estén alineados, lo que quiere decir que si el número de elementos a ingresar a los registros es menor a la longitud del registro, se deben completar los elementos faltantes con algún valor generalmente 0. Los tipos de datos que se pueden ingresar varían a la implementación de cada Unidad SIMD en el procesador y dependen de la longitud de los registros de la unidad.

Un tipo de operaciones que se pueden realizar en la Unidad SIMD son los que se realizan entre dos registros o vectores de la unidad y el resultado es almacenado en otro registro (Figura 2.2(a)). Otro grupo son las operaciones que se realizan entre elementos de un vector

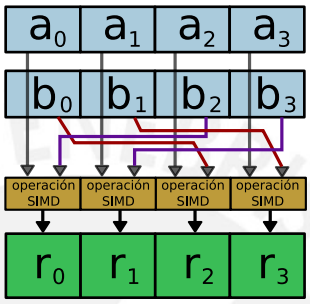
con otro elemento del mismo vector o de otro (Figuras 2.2(b) y 2.2(c)).



(a) Operaciones entre registros



(b) Operaciones entre elementos de un mismo registro



(c) Operaciones entre elementos de diferentes registros

Figura 2.2: Operaciones que se pueden realizar en las unidades SIMD.

2.3. Unidad SIMD de Intel: MMX/SSE/SSE2/SSE3/SSSE3/SSE4

A partir de las familias de procesadores Pentium II y los procesadores Pentium con tecnología MMX, se incorporan a los procesadores, de arquitectura Intel IA-32 y EM64T, seis extensiones que realizan operaciones SIMD. Estas incluyen la tecnología MMX, las extensiones SSE, SSE2, SSE3, SSSE3 y SSE4. SSE significa *Streaming SIMD Extensions* y SSSE, *Suplemental Streaming SIMD Extensions*. Para almacenar datos enteros se cuenta con los registros MMX de 64-bits de largo y los registros XMM de 128-bits de largo. Para datos de coma flotante se utiliza los registros XMM. Inicialmente, la unidad SIMD (MMX) utilizaba los mismos registros que la unidad de coma flotante (FPU *Floating Point Unit*), por lo que sólo se podía utilizar una de éstas unidades a la vez. Actualmente la unidad SIMD utiliza registros diferentes a los de la FPU [4].

La implementación de los procesadores IA-32 cuenta con ocho registros XMM y los tipo de datos que se pueden ingresar, a partir de las extensiones SSE2, son: Byte (8-bits), Word (16-bits), DoubleWord (32-bits), QuadWord(64-bits), Double QuadWord (128-bits) y Punto Flotante de Precisión Doble (64-bits) (Figura 2.3). La implementación de los procesadores EM64T poseen 8 registros adicionales.

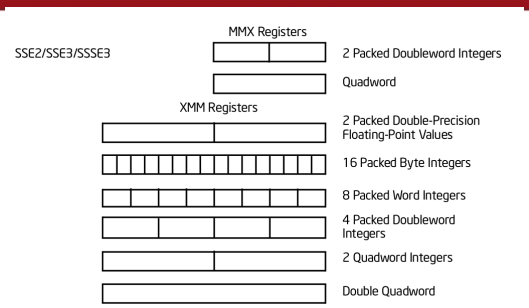


Figura 2.3: Distribución de los Registros y Tipo de dato en la unidad SIMD de Intel [4].

2.4. Aplicaciones

2.4.1. Acceso a la Unidad SIMD

En los procesadores Intel hay tres formas de utilizar la unidad SIMD [20]. Éstas se diferencias por el desempeño que se puede obtener, facilidad de programación y portabilidad del código:

Optimizaciones del compilador En este método el compilador se encarga de utilizar la unidad SIMD cuando éste encuentre alguna operación que se pueda optimizar. Se habilita al indicarle al compilador cuales son las características del procesador en la que el resultado será ejecutado. Tiene la ventaja de que el código no se altera para operar con la unidad SIMD. Su desempeño depende del compilador que se utilice, compiladores avanzados (como el Intel C++ Compiler [21]) proveen un sistema de optimizaciones que hace uso extensivo de las optimizaciones SIMD cuando encuentra un bucle vectorizable, sin embargo el tiempo de compilación se incrementa y el resultado es incierto. No siempre se obtiene un buen resultado con el uso de estas optimizaciones. El código es protable, debido a que no ha sufrido modificaciones para usar la unidad SIMD.

Funciones Intrínsecas Las funciones intrínsecas son llamados a funciones predefinidas en el compilador, se utilizan principalmente en la vectorización y paralelización. Para el caso de los procesadores Intel, estas funciones realizan llamados directos a las instrucciones en ensamblador respectivas, pero los compiladores mantienen el control de como son utilizadas [21][22]. Obtienen un buen rendimiento, ya que se indica explítamente como será el algoritmo. La portabilidad se ve limitada dado que los intrínsecos estan relacionados directamente con el lenguaje ensamblador, lo cual lo hace dependiente del procesador.

Lenguaje Ensamblador Se obtiene control de las operaciones que se realizan y de los registros con los que se opera, de modo que es la forma en la que se consigue el mejor desempeño de un algoritmo. Programar en lenguaje ensamblador vuelve al código en dependiente del procesador en el que se programa. La dificultad de utilizar este método es la más alta de los tres métodos expuestos.

2.4.2. Proyectos que utilizan la Unidad SIMD

FFTW *Fastest Fourier Transform in the West* [23], es una librería para calcular la Transformada Discreta de Fourier, además de la Transformada Discreta de Coseno y Seno. Fue desarrollada en el *Massachusetts Institute of Technology* y, de acuerdo a sus continuas pruebas de desempeño, es la implementación libre mas rápida de la Transformada Rápida de Fourier. Entre sus características podemos observar que a partir de su versión 3.0 esta librería utiliza las Unidades SIMD de los procesadores Intel, AMD y PowerPC (SSE/SSE2, 3DNow! y Altivec respectivamente). Su eficiente desempeño ha sido reconocido en 1999 con el premio *J. H. Wilkinson Prize for Numerical Software*[24]. Esta librería es utilizada comercialmente por el entorno MatLab desde su versión 6 [25][26].

ATLAS *Automatically Tuned Linear Algebra Software* [27], es una librería que implementa la interfaz de programación de aplicaciones *BLAS* (*Basic Linear Algebra Subprogramas* - Subrogramas Básicos de Álgebra Lineal), la cual es un estándar que define un conjunto de funciones para realizar operaciones entre vectores y matrices. El proyecto ATLAS utiliza la unidad SIMD de Intel para implementar las funciones requeridas por el estándar BLAS. Programas como el MatLab [26] y GNU Octave hace uso de esta librería.

Adicionalmente a estos proyectos, existen otros que hace uso de la Unidad SIMD para realizar diversas operaciones, por ejemplo en aplicaciones de criptografía[28], y diversas aplicaciones de audio y video [29][30][31].

2.5. Unidad SIMD y el Filtro mediano

El algoritmo mas difundido para ejecutar el filtro mediano es el propuesto por T. Huang [10], el cual se utiliza en la mayoría de los programas de edición de imágenes (Adobe Photoshop por citar un ejemplo [13]). Huang reduce la complejidad del algoritmo original al disminuir los procesos que se deben ejecutar para obtener el resultado, y esto lo consigue al mantener en memoria el histograma del área de análisis para que cuando se analice el siguiente sector adyacente solo se deban retirar del histograma los datos que no corresponden al histograma del nuevo sector de análisis y agregar los datos nuevos. En base a este algoritmo se han realizado múltiples optimizaciones para mejorar su desempeño, Gil[32] propone el uso de un árbol para obtener la mediana, mientras Chaudhuri[14] propone utiliza listas enlazadas en vez del histograma para mantener la data entre sectores adyacentes. El estudio de Wiess[33] consigue disminuir la complejidad de manera notable, pero la implementación de su algoritmo pierde simplicidad.

Si bien estos algoritmos presentan optimizaciones válidas, mantienen un razonamiento basado en la ejecución del programa en un procesador SISD. Para aprovechar las ventajas de procesamiento que ofrece SIMD se debe hacer un cambio en el paradigma de diseño del algoritmo. Se debe tener en cuenta que los datos se tratan como vectores datos, de modo que las operaciones que se realizan son vectoriales y tambien se pueden realizar arreglos

matriciales. De esta manera se deben rediseñar los algoritmos que obtienen la mediana de una matriz y el algoritmo que actualiza la matriz con los nuevos datos a ser procesados. En la actualidad ya existen trabajos en los que se utiliza el paradigma de programación SIMD para mejorar el filtro mediano.

Kolte *et al.*[1] propone un método para obtener el valor medio de una matriz al aplicar ordenamiento de datos, la implementación de su trabajo la realizó en la arquitectura SIMD. Primero se leen un conjunto de datos de la matriz (imagen), los primero de los cuales se almacenan en los registros de la arquitectura SIMD, luego se procede a ordenar las columnas de mayor a menor y se repite el proceso con las filas. El siguiente paso es tomar tres vectores que contienen las diagonales centrales de la matriz y a éstos se aplica el algoritmo de ordenamiento nuevamente. Del vector que contienen los máximos se toma el menor valor, del vector que contiene los mínimos se toma el máximo y del vector que contiene las medianas se toma la mediana. Finalmente estos tres valores son ordenados y se toma el valor central, que viene a ser la mediana de la matriz. Se puede revisar el algoritmo 3.

Algoritmo 3: Cálculo de la mediana de Kolte [1]

Entrada: A es un arreglo de NxN de data desordenada
Salida : m es la mediana de los valores del arreglo
sea $M = (N-1)/2$
/* se ordenan las columnas */
Para $c = 0$ *hasta* $N-1$ **hacer**
  ordenar columna c tal que $A[r-1, c] <= A[r, c]$
/* se ordenan las filas */
Para $r = 0$ *to* $N-1$ **hacer**
  ordenar filas r tal que $A[r, c-1] <= A[r, c]$
Para $k = 1$ *hasta* M **hacer**
  /* ordenamos las diagonales con pendiente k */
  Para $s = k*(M+1)$ *hasta* $k*(M-1)-(N+1)$ **hacer**
    ordenar linea $(k*r+c=s)$ tal que $A[r-1, s-k*(r-1)] <= A[r, s-k*r]$
sea $m = A[M, M]$

De manera similar, Furtak *et al.*[34] se basa en el paradigma de programación que plantea la arquitectura SIMD para diseñar un método para ordenar datos. Dado un vector de datos, se realizan movimientos de los registros para mejorar la eficiencia de la aplicación del algoritmo de comparacion mayor menor que se necesita para ordenar los datos. De esta manera evita el uso de saltos condicionales que disminuyen la eficiencia del programa.

2.5.1. Problemática

El uso de los algoritmos tradicionales, con sus respectivas implementaciones, se pueden considerar ineficientes dadas las nuevas formas de procesamiento de datos que brindan los procesadores actuales. Por esto es necesario el planteamiento de nuevos algoritmos que aprovechen las nuevas funcionalidades que poseen dichos procesadores.

Bajo este ámbito se puede decir que el filtro mediano, dadas sus características y a los diversos estudios realizados, es uno de los algoritmos que mayor beneficio obtiene del nuevo paradigma de programación que ha generado la arquitectura SIMD. La posibilidad de operar arreglos de datos permitiría que el tiempo de procesamiento del filtro mediano disminuya de

gran manera, convirtiéndolo en una operación de uso frecuente junto con los tradicionales filtros lineales como son el filtro gaussiano o las operaciones con la transformada de Fourier.

En las implementaciones tradicionales del algoritmo del filtro mediano se pueden observar dos problemas fundamentales. El primero es el acceso a la misma información en repetidas ocasiones y el segundo es el tiempo que se requiere para realizar el ordenamiento de los datos para obtener la mediana.

Para minimizar el impacto en el desempeño que tiene la lectura de los datos de la imagen se plantea almacenar resultados parciales en los registros del procesador, con lo que se logra reducir notablemente el tiempo de ejecución. Una de las limitaciones de trabajar con las unidades SIMD es su número limitado de registros: En los procesadores Intel de 32 bits (IA-32) la cantidad de registros es de 8, mientras que en la versión de 64 bits (EM64T) se cuenta con 16 registros. En los procesadores PowerPC, y en general todos los procesadores que posean la tecnología AltiVec, poseen 32 registros SIMD. Se tendrá en cuenta la jerarquía de memoria, al tener un acceso ordenado a los datos, para aprovechar los beneficios de la memoria caché.

Con el fin de evitar el proceso de ordenamiento de datos se consideró obtener la mediana con el cálculo del histograma del arreglo de datos, sin embargo el separar registros para calcular el histograma perjudica al primer problema mencionado. Como una alternativa, se considera el uso de listas enlazadas, lo cual reduce la memoria que se utiliza, pero el agregar y quitar elementos de ésta la convierte en poco práctica. Una tercera opción es el uso de funciones recursivas, las cuales disminuyen la complejidad del algoritmo. De éstas alternativas ninguna es aplicable en una unidad SIMD por las limitaciones que esta posee. No se puede realizar accesos indexados a los registros, por lo que el uso de una lista enlazada o el cálculo directo del histograma no es factible, además que al contar con un número limitado de registros no son recomendables estos métodos.

Para realizar una implementación eficiente del filtro mediano se requiere un procesador que posea una implementación de una unidad SIMD, como son la tecnología MMX, SSE, SSE2, SSE3, SSSE3 en los procesadores Intel. El sistema operativo debe permitir el acceso a esta unidad a los programas que se ejecuten. La implementación del algoritmo de filtro mediano se realiza en lenguaje de bajo nivel *(lenguaje ensamblador)*, por lo que es necesario un módulo que permita el uso de la implemtación realizada en lenguajes de mayor nivel *(C/C++, MatLab)*. De esta manera se puede utilizar un filtro mediano eficiente en programas de análisis de imágenes (Figura 2.4).

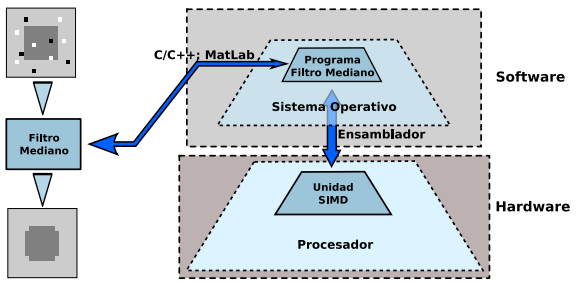


Figura 2.4: Representación Gráfica del Modelo Teórico

Capítulo 3

Diseño del Algoritmo Vectorial para el Filtro Mediano

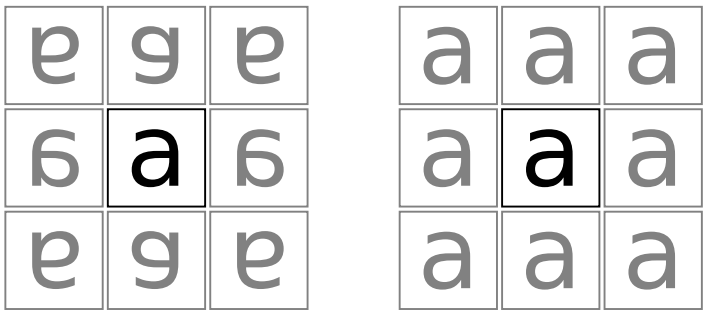
3.1. Consideraciones de Diseño

Número de Registros SIMD Para el diseño del algoritmo se asume que se posee un infinito número de registros SIMD con los que se puede trabajar. En la realidad se cuentan con 8 registros SIMD en la implementación de Intel de 32 bits (IA-32), 16 para los procesadores Intel de 64 bits EM64T y 32 registros SIMD en la implementación de Motorola en los procesadores PowerPC.

Tamaño de la Imagen El Ancho de la Imagen, o número de columnas del arreglo, será múltiplo de la cantidad de datos que se pueden almacenar en un vector en los registros SIMD. Además el alto de la imagen es mayor o igual al tamaño del área de análisis.

Instrucciones Universales El algoritmo podría ser implementado en cualquier unidad SIMD, por lo que no se utilizan instrucciones específicas de un procesador en la etapa de diseño.

Bordes Se consideran los bordes del tipo reflexivo [35](figura 3.1). Se elige esta consideración de borde ya que permite mantener un orden en la lectura de los datos.



(a) Reflexivo (b) Periódico

Figura 3.1: Dos condiciones de borde.

3.2. Cálculo de la Mediana

El método que se utilizará para calcular la mediana será el de ordenar los datos y obtener el elemento central del arreglo. Como operación pincipal se utiliza una Red de Ordenamiento *(Sorting Network)*, la cual es la base del algoritmo de Kolte[1], con el que se calculará la mediana.

3.2.1. Red de Ordenamiento

Una Red de Ordenamiento reduce el número de comparaciones que se realizan al ordenar un conjunto de datos. Algoritmos clásicos basados en comparación suelen repetir comparaciones en sus bucles, una red de clasificación realiza el numero de comparaciones mínimo para obtener el arreglo ordenado de datos. Adicionalmente no hace uso de sentencias condicionales y saltos, por lo que su ejecución es rápida y fácilmente vectorizable [12].

El principio de funcionamiento de una red de clasificación es que dadas dos entradas a_1 y a_2 la salida de la red es $min(a_1, a_2)$ y $max(a_1, a_2)$ (Figura 3.2). Esta operación de dos elementos se extiende a arreglos de mayor longitud (Figura 3.3).

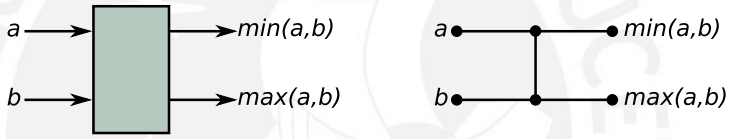


Figura 3.2: Funcionamiento de una Red de Clasificación de dos elementos, se muestran dos formas de representación.

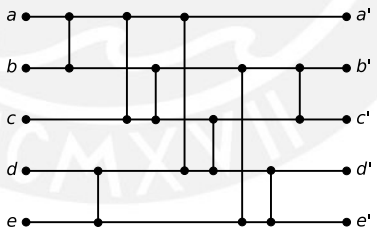


Figura 3.3: Funcionamiento de una Red de Clasificación de cinco elementos.

La función básica, una red de clasificación de dos elementos, se implementa con el uso de un registro temporal y las operaciones máximo y mínimo. Si tenemos los elementos a y b, asignamos el valor de a a una variable temporal $temp$, entonces $a = max(a, b)$ y $b = min(temp, b)$.

3.2.2. La Mediana de Kolte

Kolte *et al.* [1] propone un método para obtener la mediana por medio de una serie Redes de Clasificación, además, este método ha sido diseñado para ser implementado en una unidad SIMD. Como primer paso se debe ordenar de manera ascendente las columnas del

arreglo, y luego se ordena de manera ascendente las filas del mismo arreglo. De esta manera se obtiene en la esquina superior izquierda del arreglo los máximos y en la esquina inferior derecha del arreglo, lo mínimos. Ahora se toman los elementos con pendiente $k = 1$, los elementos centrales que pertenecen a la recta con dicha pendiente, y se procede a ordenar los datos. Esta última operación se repite incrementando el valor de k hasta que solo quede un elemento en el centro del arreglo, el cual es la mediana del arreglo (Figura 3.4).

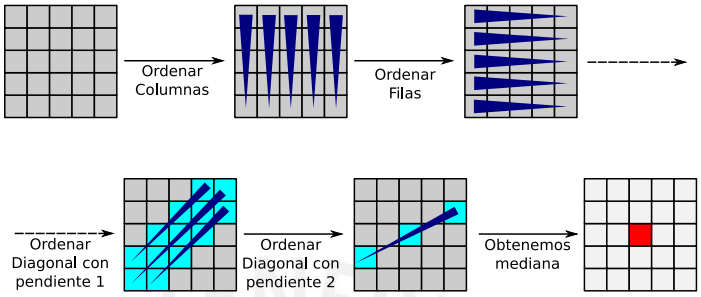


Figura 3.4: Algoritmo de Kolte para hallar la mediana para el caso de un kernel de 5×5.

3.3. Accesos a la Memoria

Para agilizar los accesos a memoria se leerán los datos de manera ordenada, de modo que la memoria caché pueda adelantarse a la próxima lectura. Esto se consigue si tenemos en cuenta que una imagen se almacena en memoria como un arreglo unidimensional. Por lo tanto, si una imagen tiene dimensiones $M \times N$, podemos afirmar que en la memoria la imagen esta almacenada como un arreglo unidireccional de $M \times N$, con índices de 1 a $M \times N$. En este arreglo unidireccional, cada bloque de M datos respresenta una fila de la imagen. De esta manera si recorremos este arreglo con punteros distanciados por M datos, podemos leer cada fila de manera ordenada y con un patrón que incrementaría las probabilidades de acierto de la cache (Figura 3.5).

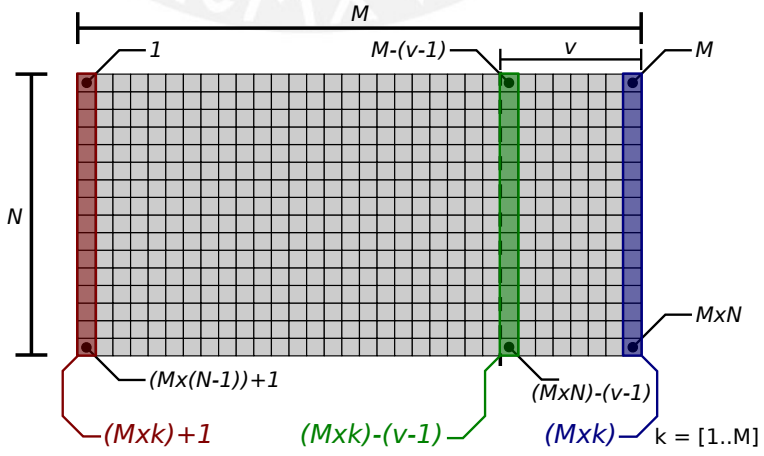


Figura 3.5: Representación de los índices del arreglo unidireccional que corresponde a una imagen de $M \times N$.

3.4. Descripción del Algorimo

3.4.1. Algoritmo General

La imagen será tratada con un algoritmo general para todos los datos, excepto para los que corresponden a los bordes, los cuales tendrán una implementación especial. Con la imagen I de $M \times N$ como entrada y un área de análisis de $k \times k$, se procede a calcular la mediana de la imagen por bloques. Cada uno de estos esta conformado por $k \times v$ datos, donde v es el número del pixeles que se pueden almacenar en un registro SIMD. Para aplicar el filtro mediano a un bloque B_n es necesario tener en los registros los datos de los bloques B_{n-1} y B_{n+1}. Esto es debido a que para obtener la mediana de los pixeles de los extremos del bloque, para el primer y el último elemento del vector, es necesario tener los $\frac{k-1}{2}$ elementos anteriores y posteriores, respectivamente (Figura 3.6).

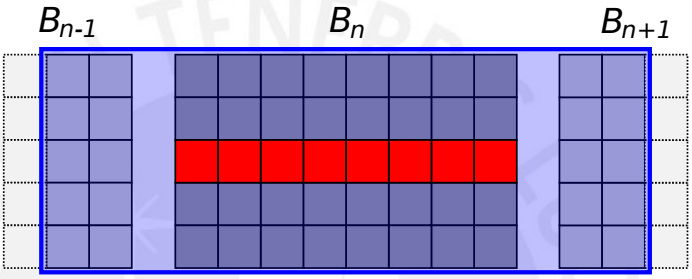
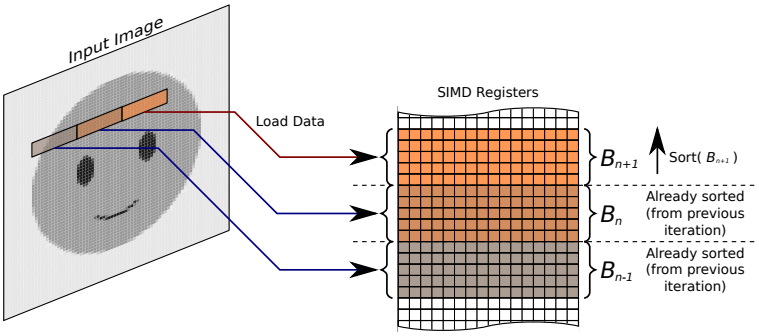


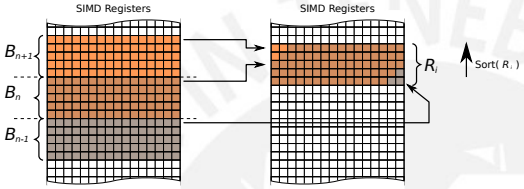
Figura 3.6: Para obtener la mediana del bloque B_n (en rojo), son necesarios los datos de algunos datos de los bloques B_{n-1} y B_{n+1}

Los datos correspondientes a B_{n-1} y B_n ya están ordenados de manera decreciente de la iteración anterior (esto quiere decir que, siendo $R_0, R_1, \ldots, R_{v-1}$ los vectores que conforman el bloque B_n, cada elemento del vector R_i es mayor que el correspondiente elemento en el vector R_{i+1}) de modo que es necesario ordenar los datos del bloque B_{n+1}, que recién han sido leídos de memoria. Para esto se utilizan las redes de ordenamiento en su forma vectorial, es así como obtenemos los tres bloques ordenados de manera decreciente. Cabe mencionar que al final de la iteración, los registros que contienen los datos de B_{n-1} serán reemplazados con los datos de B_n, y los datos de éste último, lo serán con los datos de B_{n+1}, de esta manera para la siguiente iteración se tienen ordenados los registros de los dos primeros bloques necesarios, y sólo se tendría que realizar el ordenamiento del nuevo bloque leído B_{n+2}.

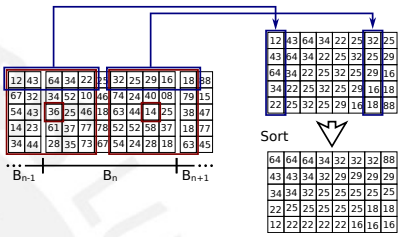
Para la siguiente parte del algoritmo, es necesario ordenar los elementos de cada vector en grupos de k elementos, de modo que de los $\frac{k-1}{2}$ vectores con los datos mayores se obtengan los datos de menor valor; de los $\frac{k-1}{2}$ vectores con los datos menores, los de menor valor, y de los vectores restantes de obtienen los valores centrales. Se repite esta operación a cada nuevo grupo de datos que se obtengan hasta que el grupo de elementos se reduzca a tres elementos, los cuales se ordenan y el elemento central que resulte de éste último proceso correspondería al valor de la mediana (figura 3.7).



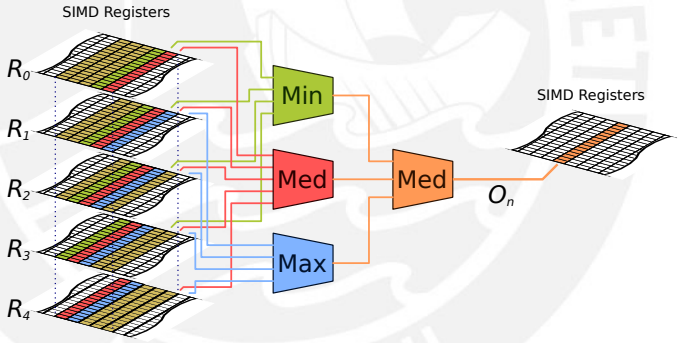
(a) Paso 1: Cargar Datos y Ordenar Columnas



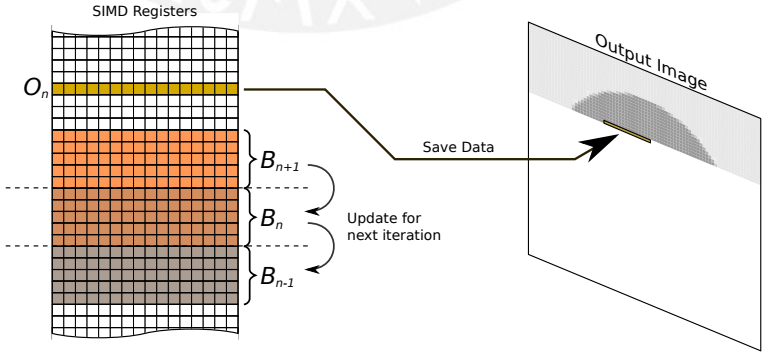
(b) Paso 2: Ordenar Filas



(c) Paso 2: Ordenar Filas (ejemplo numérico)



(d) Paso 3: Ordenar Diagonales



(e) Paso 4: Almacenar Datos

Figura 3.7: Esquema de gráfico del algoritmo.

3.4.2. Modificación del algoritmo general debido a los bordes

Hay ocho casos especiales en los que el algoritmo general se debe modificar debido a los bordes: las cuatro esquinas de la imagen y los bordes superior, inferior, izquierdo y derecho (figura 3.8). Las modificaciones necesarias para adaptarse a las esquinas se pueden descomponer en las modificaciones que se realizan a los bordes a los que pertenece la esquina. Por ejemplo, para la esquina superior derecha, se deben aplicar las modificaciones correspondientes a los bordes superior y derecho.

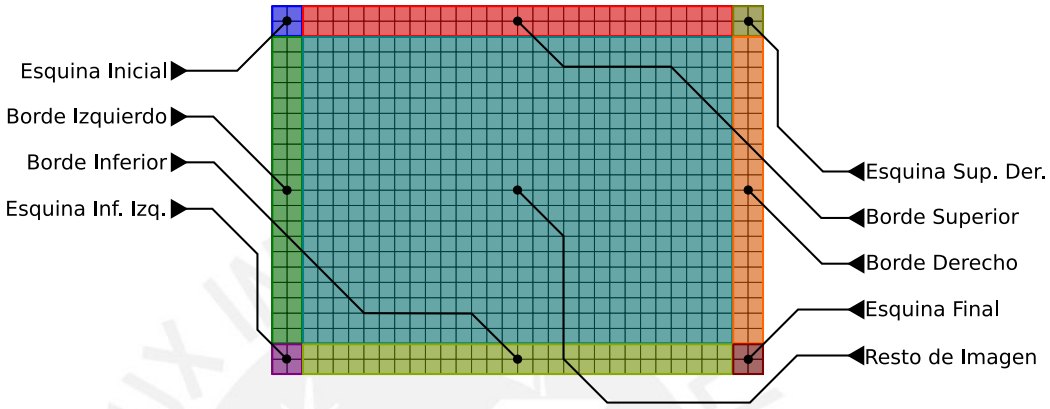


Figura 3.8: Sectores de una imagen analizada con kernel de 3×3

Para el borde superior e inferior de la imagen solo se cargan en los registros SIMD los $\frac{k+1}{2}$ primeras filas de la imagen. Se completan los k vectores necesarios con una copia de los $\frac{k-1}{2}$ vectores inferiores. Por ejemplo, para $k = 5$ y se desea obtener la mediana del vector A_0, se cargan los vectores A_0,A_1, y A_2. El cuarto y quinto vector necesario serían una copia de los vectores A_1 y A_2. Si seguimos con este ejemplo, si ahora se desea obtener la mediana que corresponde al vector A_1, los datos que se cargan en los registros SIMD serían A_0,A_1,A_2 y A_3; el quinto vector necesario sería la copia del vector A_1. Una vez que se completan los cinco vectores se procede con el cálculo de la mediana ya descrito en este documento.

La modificación necesario al borde izquierdo de la imagen consiste en, para analizar el bloque B_n, reemplazar los valores que corresponderían al bloque B_{n-1} con el bloque B_n, de modo que todos los valores que intervienen en el cálculo de la mediana sean los que corresponden a la segunda columna del bloque B_n. Por ejemplo, para un kernel de 3×3, el valor de la última columna del bloque B_{n-1} deben ser los mismos que la segunda columna del bloque B_n. Para el borde derecho se opera de manera análoga, pero en vez de trabajar con el bloque B_{n-1}, se hace con el bloque B_{n+1}, y la primera columna de éste se reemplaza por los valores de la penúltima columna del bloque B_n.

Capítulo 4

Implementación y Resultados Computacionales

Los programas para sistema operativo GNU/Linux de 64 bits se adjuntan en el CD adjunto. El código fuente de todos los programas también se encuentran en el mencionado CD. Adicionalmente se adjunta, en el CD, la interface MEX para que el programa desarrollado pueda utilizarse en MatLab.

4.1. Consideraciones de la Implementación

Plataforma de implementación Se utilizará la Unidad SIMD de los procesares Intel EM64T, con su set de instrucciones hasta el SSSE3, en un sistema operativo Linux de 64 bits. Esto da acceso a 16 registros SIMD.

Herramientas de desarrollo Se utilizan las herramientas de desarrollo propias del compilador GCC 4.4. Los lenguajes en los que se implementará serán C y Assembler.

Tamaño del Kernel El área de análisis será de 3×3 y 5×5. Esto se debe a la cantidad limitada de registros con los que se cuenta.

Características de las Imágenes Las imágenes serán de 256 colores, lo que nos indica que cada pixel ocupará 8 bits en el registro SIMD. Esto nos da un total de 16 datos por registro. El ancho de la imagen debe ser múltiplo de este número para asegurarnos que los datos estén alineados y se pueda utilizar la Unidad SIMD a toda su capacidad.

4.2. Descripción de la Implementación

Se realizó una implementación inicial para verificar el algoritmo de Kolte para obtener la mediana. La implementacíon final del filtro mediano se realiza como una librería estática, de la cual hace uso un programa que permite obtener resultados del desempeño computacional. Éste programa utiliza imágenes en formato PGM [36].

4.2.1. Implementación del Algoritmo de Kolte

Para probar que el método propuesto de Kolte obtiene, de manera correcta, el valor de la mediana de un conjunto de datos, se realizó la implementación de su algoritmo con la Unidad SIMD y se sometió a una prueba con un grupo de datos de los cuales se conoce el valor de la mediana. Estos datos consisten en todas las posibles combinaciones que se pueden realizar con los nueve primeros números naturales, los cuales se almacenan en sectores de memoria que se han asignado de manera que estén alineados. Esta asignación se realiza con la funcion *posix_memalign*, la cual nos asegura que los arreglos en memoria serán múltiplos de la capacidad de los registros SIMD.

Adicionalmente, el programa que implementa el algoritmo de Kolte, también implementa otros algoritmos para realizar una comparación de desempeño entre éstos. El indicador que se utiliza es el número de ciclos del reloj del procesador que se necesitan para obtener el resultado [37].

4.2.2. Implementación del Filtro Mediano

Este programa recibe como parámetros de entrada el nombre del archivo al cual se desea aplicar el filtro mediano, y el nombre del archivo en el que se almacenará la imagen resultante. El formato de archivo que se utiliza es el PGM, el cual se caracteriza por almacenar imágenes solo en escala de grises, además de no utilizar algun método de compresión, el cual lo hace ideal para realizar pruebas a algoritmos. Para leer y escribir en este formato se escribió una serie de funciones que leen los datos del archivo y los almacenan en memoria que ha sido asignada con la funcion *posix_memalign*, de modo que pueda ser cargada directamente a los registros SIMD.

Esta implementación tiene el mismo mecanismo para contar los ciclos del reloj del procesador, y será lo que se utlizará para medir el desempeño del algoritmo. Para tener una referencia de desempeño, se ha incorporado al programa el algoritmo diseñado e implementado por Simon Perreault y Patrick Hébert, el cual afirma tener un mejor desempeño que los algoritmos actuales del filtro mediano [2]. Su algoritmo es llamado *CTMF* (*Constant Time Median Filter* - Filtro Mediano de Tiempo Constante). Según los resultados presentados en [2], el CTMF posee un desempeño computacional superior a la implementación propuesta por Huang [10].

4.3. Resultados Computacionales

Las pruebas se realizaron con un procesador Pentium Dual-Core T2330 de 1.60GHz de frecuencia del reloj máxima, 1024 KB de memoria caché y 1 GB de memoria RAM. El sistema operativo es Fedora 11 de 64 bits, kernel 2.6.30 y con escritorio KDE.

4.3.1. Prueba al algoritmo de Kolte.

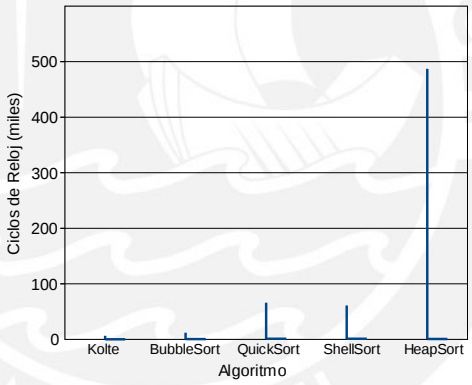
Para probar el algoritmo de Kolte, una vez demostrada su eficacia, se generan 9 datos aleatorios de 8 bits y se obtiene la mediana de este conjunto de datos. Esta operación se re-

Tabla 4.1: Resultados de diversos algoritmos de ordenamiento para nueve elementos.

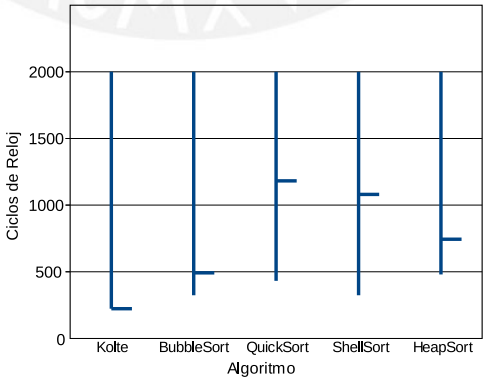
Algoritmo	Ciclos de Reloj			
	Promedio	Mediana	Mínimo	Máximo
Kolte	222.44	216	204	5844
BubbleSort	507,02	492	324	12012
QuickSort	1144,84	1182	432	66168
ShellSort	1050,71	1080	324	61020
HeapSort	1113,21	744	480	487152

pite 2000 veces con datos diferentes. La misma prueba se realiza con otra implementaciones que nos permiten hallar el valor mediano mediante el ordenamiento de datos.

Los resultados (Tabla 4.3.1) indican que en el mejor de los casos, el algoritmo de kolte utiliza 204 ciclos del reloj para obtener la mediana, mientras que tanto el algoritmo *ShellSort* como el *BubbleSort* obtienen 324 ciclos. Sin embargo, el valor medio de los resultados obtenidos indican que el algoritmo de Kolte utiliza 216 ciclos del reloj, mientras que el *BubbleSort* consigue 492 ciclos, aproximadamente 2.3 veces mas ciclos de reloj que el algoritmo de Kolte. Los resultados se pueden observar en la gráfica de la Figura 4.1. Nótese que los valores de ciclos máximos no es confiable, pues hay factores que afectan ese resultado: Otras tareas o procesos que el sistema operativo maneja.



(a) Gráfica completa



(b) Acercamiento a la zona donde se ubican los valores medianos de las pruebas.

Figura 4.1: Comparación del desempeño de diferentes algoritmos para obtener la mediana.

4.3.2. Prueba al algoritmo de Filtro Mediano.

Para esta prueba se lee una imagen en escala de grises, de 512×512 pixeles, a la cual se la ha agregado ruido *sal y pimienta*. Se muestran los resultados para distintos niveles de ruido con filtrajes con kernels de 3×3 y 5×5. Se debe notar el incremento de la SNR en las imágenes que se muestran, un mayor SNR indica menor presencia de ruido (Figura 4.2).

(a) 10 % Ruido Impulsivo (SNR=13.00dB)
(b) Filtro Mediano 3×3 (SNR=14.13dB)
(c) Filtro Mediano 5×5 (SNR=12.10dB)
(d) 20 % Ruido Impulsivo (SNR=10.00dB)
(e) Filtro Mediano 3×3 (SNR=13.18dB)
(f) Filtro Mediano 5×5 (SNR=11.72dB)
(g) 30 % Ruido Impulsivo (SNR=8.23dB)
(h) Filtro Mediano 3×3 (SNR=12.16dB)
(i) Filtro Mediano 5×5 (SNR=11.34dB)

Figura 4.2: Aplicación del filtro mediano de 3×3 y 5×5 en imágenes con 10 %, 20 % y 30 % de ruido impulsivo. La imagen original es "Lena" de 512×512 pixeles

En lo que respecta a su desempeño computacional (Tabla 4.2), en el mejor de los casos obtiene 473220 ciclos del reloj, en el peor, 4822872 y su desempeño medio es de 478398 ciclos. El algoritmo de referencia, el CTMF [2], obtiene en el mejor de los casos 41202828 ciclos, en el peor, 87691932, y como media su resultado fue de 41442558 ciclos de reloj. Estas pruebas nos indican que el algoritmo desarrollado es aproximadamente 87 veces mas rápido que el algoritmo de referencia. La imagen que se utilizó para esta prueba es de 1600×1200 y 256 colores.

Tabla 4.2: Resultados computacionales de la aplicación del filtro mediano del algoritmo propuesto y el de referencia. El kernel utilizado es de 3×3.

	Ciclos de Reloj			
Algoritmo	Mínimo	Promedio	Mediana	Máximo
Vector	473220	507608,20	478398	4822872
CTMF	41202828	42208676,35	41442558	87691932

Para tener una referencia de los tiempos de ejecución es necesario conocer la frecuencia del reloj del procesador. Este valor se obtiene al contar los ciclos de reloj en el lapso de un segundo. Para el caso del procesador T2660, se obtuvo que se realizan 1596210240 ciclos en un segundo, lo que nos indica que el procesador esta trabajando cercano a su frecuencia máxima recomendable (1.6GHz). Al realizar el cálculo apropiado, podemos afirmar que el tiempo medio de ejecución el filtro mediano es de 0.2997 mseg, Si tenemos en cuenta que un video tiene una velocidad de 30 cuadros por segundo, cada uno de estos cuadros debe ser procesador en menos de 33,3 mseg. El tiempo que lleva a cabo el aplicar el filtro mediano a un cuadro sería solo el 1 % del tiempo total disponible para procesar la imagen en tiempo real. En el caso del CTMF, son necesarios 25.96 mseg para aplicar el filtro.

4.3.3. Análisis de los Resultados.

Una vez demostrada la mejora en el desempeño computacional que presenta la implementación realizada, se procede a realizar un conjunto de pruebas para obtener datos concisos de comparación. Para esto se ejecutan ambas implementaciones del filtro a imágenes de diferentes tamaños. Los resultados se muestran en las Tablas 4.3 y 4.4. [3].

Tabla 4.3: Resultados computacionales del procesador EM64T y kernel 3×3

	Ciclos de Reloj		$\frac{Ciclos}{Pixel}$	
Imagen	Vector	CTMF	Vector	CTMF
128×128	33971	2243694	2,07	136,94
256×256	127698	9855750	1,95	150,39
512×512	508074	40790580	1,94	155,60
768×768	1212594	97932858	2,06	166,04
1024×1024	2200506	172130670	2,10	154,63

Tabla 4.4: Resultados computacionales del procesador EM64T y kernel 5×5

	Ciclos de Reloj		$\frac{Ciclos}{Pixel}$	
Imagen	Vector	CTMF	Vector	CTMF
128×128	149130	2254866	9,10	137,63
256×256	559146	9928476	8,53	151,50
512×512	2193846	41365668	8,37	157,80
768×768	4924470	91597956	8,35	155,30
1024×1024	8713950	167149968	8,31	159,41

De éstas tablas se puede observar que promedio se utilizan 2,02 ciclos de reloj por píxel para aplicar el filtro mediano de 3×3, los cuales equivalen el $1,3\,\%$ de los ciclos requeridos por el algoritmo de referencia. En el caso del kernel de 5×5, son necesarios un promedio de 8,53 ciclos de reloj, es decir, el $5,6\,\%$ de los utilizados por el otro algoritmo. Los resultados

obtenidos por Kolte en un procesador PowerPC [1] indican que su implementación requiere, en promedio, 1.15 ciclos de reloj por píxel para el filtro de kernel 3×3, y 6.6 ciclos para el filtro de kernel 5×5. Esta diferencia se debe a que un procesador PowerPC cuenta con 32 registros SIMD [38] e instrucciones que permiten una implementación mas rápida del algoritmo.

Conclusiones

- La implementación vectorial del filtro mediano tiene el mismo efecto en la imagen que su implementación escalar. Esta verificación del funcionamiento del algoritmo se realizó gracias a la interface MEX. La misma imagen es filtrada por el filtro mediano propio de MatLab y por el filtro vectorial.
- El uso de la Unidad SIMD mejora el rendimiento computacional del filtro mediano. Se reduce notablemente la cantidad de ciclos del procesador que son necesarios para aplicar este filtro a una imagen, comparados con implementaciones que no utilizan de manera adecuada esta unidad.
- Se obtienen mejores resultados cuando se diseña un algoritmo que utilice las nuevas tecnologías, en este caso las unidades SIMD, que cuando se paraleliza un algoritmo que ha sido creado utilizando las metodologías tradicionales. Esto se puede corroborar al revisar el código de la implementación del CTMF. Esta usa la unidad SIMD para manipular los histogramas de una variación del algoritmo de Huang. La diferencia entre la mencionada implementación y la que se propone en este documento es considerable.
- El tiempo de ejecución obtenido al aplicar el filtro con esta implementación es lo suficientemente bajo como para que el filtro pueda ser utilizado en aplicaciones de procesamiento de imágenes en tiempo real.

Recomendaciones y Observaciones

- La implementación del algoritmo CTMF [2], con el cual se hizo la comparación del desempenño del algoritmo desarrollado, se tomó directamente de la página de su autor. Sin embargo, la imagen resultante, al utlizar esta implementación, solo contiene las tres primeras columnas con el filtro aplicado de manera correcta. El resto de la imagen es de color negro.
- Es recomendable encontrar otras implementaciones ya probadas del filtro mediano para tener puntos de referencia mas confiables. Sería ideal comparar esta implementación con la que propone Ben Weiss [33].
- Para implementar el algoritmo para áreas de análisis mayores 5×5 (por ejemplo 7×7, 9×9) en procesadores Intel se tendrá que modificar el algoritmo para adaptarlo al reducido número de registros SIMD que estos procesadores poseen.

Bibliografía

[1] P. Kolte, R. Smith, and W. Su, "A fast median filter using altivec," Motorola Inc., 1999.

[2] S. Perreault and P. Hébert, "Median filter in constant time," *IEEE Trans. on Image Processing*, vol. 16, no. 9, Sep. 2007.

[3] R. Sánchez and P. Rodriguez, "Filtro mediano bidimensional rápido implementado con la arquitectura simd," *Conferencia Iberoamericana sobre Tendencias en Educacin y Colaboracin en Ingeniera*, 2009.

[4] *Intel 64 and IA-32 Architectures. Software Developers Manual*, Intel Corporation, Abril 2008.

[5] J. W. Tukey, "Nonlinear (nonsuperimposable) methods for smoothing data," in *Conf. Rec. (EASCON)*, 1974.

[6] A. C. Bovik, *Handbook of Image and Video Processing*. Academic Press, 2000.

[7] *Intel 64 and IA-32 Architectures - Optimization Reference Manual*, Intel Corporation, pp. 73–77, Noviembre 2007.

[8] *3DNow! Technology Manual*, Advanced Micro Devices, Inc., 2000.

[9] L. Null and J. Lobur, *The Essentials of Computer Organization and Architecture*. Jonas and Bartlett Publishers, 2003.

[10] T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filter algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 27, no. 2, pp. 13–18, Feb. 1979.

[11] W. K. Pratt, *Digital Image Processing: PIKS Inside*. John Wiley & Sons, Inc., 2001.

[12] D. E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Professional, May 1998, vol. Vol 3 / Sort and Searching.

[13] *Using ADOBE®PHOTOSHOP®CS4*, Adobe Systems Incorporated, 2008.

[14] B. Chaudhuri, "An efficent algoritm for running window pel gray level ranking 2-d images," *Pattern Recognition Lett.*, vol. 11, no. 2, pp. 77–80, 1990.

[15] M. Flynn, "Some computer organizations and their effectiveness," *IEEE Trans. Comput.*, vol. C, no. 22, p. 948, 1972.

[16] *AltiVec™ Technology Training*, Motorola Inc., 2002.

[17] *Blackfin Processor Programming Reference*, Analog Devices, Inc, Sep. 2008.

[18] *ADSP-2126x SHARC DSP Core Manual*, Analog Devices, Inc, Feb. 2004.

[19] *NEON™ Techonology DataSheet*, ARM Limited.

[20] *Intel 64 and IA-32 Architectures - Optimization Reference Manual*, Intel Corporation, pp. 165–267, Noviembre 2007.

[21] *Intel C++ Compiler for Linux System User's Guide*, Intel Corporation, pp. 153–170 2004.

[22] R. M. Stallman *et al.*, *Using the GNU Compiler Collection*. GNU Press, 2008, pp. 458 – 474.

[23] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[24] J. H. Wilkinson Prize for Numerical Software. [Online]. Available: http://www.mcs.anl.gov/research/opportunities/wilkinsonprize/index.php

[25] Faster finite fourier transforms matlab 6 incorporates fftw. [Online]. Available: http://www.mathworks.com/company/newsletters/news_notes/clevescorner/winter01_cleve.html

[26] Matlab software acknowledgments. [Online]. Available: http://www.mathworks.com/access/helpdesk_r13/help/base/relnotes/software.html

[27] Automatically tuned linear algebra software (atlas). [Online]. Available: http://math-atlas.sourceforge.net/

[28] Crypto++ library 5.6.0. [Online]. Available: http://www.cryptopp.com/

[29] The lame project. [Online]. Available: http://lame.sourceforge.net/index.php

[30] virtualdub.org. [Online]. Available: http://www.virtualdub.org/

[31] Pcsx2. [Online]. Available: http://www.pcsx2.net/

[32] J. Gil, "Computing 2-d min, median and max filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 5, pp. 504–507, May 1993.

[33] B. Weiss, "Fast median and bilateral filtering," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 519–526, 2006.

[34] T. Furtak *et al.*, "Using simd register and instructions to enable instruction-level parallelism in sorting algorithms," *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 348–357, 2007.

[35] S. Serra-Capizzano, "A note on antireflective boundary condition and fast deblurring models," *Society for Industrial an Applied Mathematics*, vol. 25, no. 4, pp. 1307–1325, 2003.

[36] Pgm format specification. [Online]. Available: http://netpbm.sourceforge.net/doc/pgm.html

[37] *Using the RDTSC Instruction To Performance Monitoring*, Intel Corporation, 1997.

[38] *Power ISA Version 2.06*, International Business Machines Corporation, Enero 2009.