
# Universidad Tecnológica de la Mixteca

## PROCESAMIENTO PARALELO DE IMÁGENES DE COLOIDES Y RONCHIGRAMAS.

TESIS

PARA OBTENER EL TÍTULO PROFESIONAL DE:

**INGENIERO EN COMPUTACIÓN**

PRESENTA:

**MOISÉS EMMANUEL RAMÍREZ GUZMÁN**

ASESOR:

**DR. MARIO AURELIO RODRÍGUEZ PINEDA**

JULIO DE 2002

# AGRADECIMIENTOS

**A mis padres por haberme apoyado incondicionalmente para llegar a ser lo que soy.**

**A mi familia por su amor y paciencia en tiempos buenos y adversos.**

**A mi asesor, Dr. Mario Aurelio Rodríguez Pineda, por su dirección y apoyo.**

# CONTENIDO

# INTRODUCCIÓN

El presente trabajo propone herramientas para el procesamiento de dos tipos de imágenes digitales: por un lado, imágenes de muestras de suspensiones coloidales[1] y por el otro, de imágenes llamadas ronchigramas. Las primeras surgen durante el estudio de las propiedades físicas de tales suspensiones. Las segundas son necesarias en el campo de las pruebas ópticas y sirven para determinar la calidad de los dispositivos ópticos.

Estudiar modelos de suspensiones coloidales, es de la mayor importancia en el estudio de sistemas como la sangre, las pinturas, los pegamentos, los productos lácteos, etc. Por esa razón, el estudio de las propiedades de las suspensiones coloidales ha tenido un gran desarrollo en los últimos cinco años, usando para ello técnicas de videomicroscopía.

Cabe decir que en algunas aplicaciones se requiere procesar al menos diez mil imágenes casi en tiempo real. ***El uso del procesamiento paralelo corresponde a la búsqueda de una opción para mejorar los tiempos de análisis de dichas imágenes***, los ***resultados obtenidos*** en este trabajo siguen esta estrategia.

Para llevar a cabo el estudio experimental de (algunos tipos de) suspensiones coloidales, se requiere el procesamiento de imágenes y la extracción automática de las funciones de distribución radial y el desplazamiento cuadrático medio de las partículas suspendidas. Para ello, es necesario tener el ***número de partículas presentes en cada centro de la misma imagen*** (fig. 1.1)**.**

Para poder obtener la distribución promedio de las partículas alrededor de una dada, se requiere conocer no sólo la ubicación de ésta, sino también la posición de las demás partículas dentro de la imagen y su relación con las otras partículas.

Con el fin de calcular la distribución promedio de partículas alrededor de una dada, primero se le calcula su diámetro. Posteriormente se trazan anillos concéntricos a partir del centroide de la partícula, de anchura igual a $\frac{1}{10}$ del diámetro calculado. Para cada anillo así generado, se determina el número de partículas que teniendo el mismo tamaño que la dada, tienen su centroide dentro del anillo considerado. Este proceso se repite con todas y cada una de las partículas presentes en la imagen.

En el presente trabajo, se ha realizado esta tarea de conteo. Este proceso se describe en el capítulo II. Para ello se recurrió a un sistema de cálculo rápido usando hardware y software del tipo multiprocesamiento simétrico. El programa fue desarrollado usando la librería MPICH [24]. Esta librería es una implementación de MPI (*Message Passing Interface*, en español, Interfaz de Paso de Mensajes) en C.

Por otra parte, como ya se mencionó al principio de esta introducción, en el presente trabajo también han sido consideradas imágenes digitales llamadas ronchigramas (fig. 3.13), resultado de pruebas realizadas durante la fabricación de dispositivos ópticos para el TELESCOPIO INFRARROJO MEXICANO que se está diseñando y construyendo

[1] Una suspensión coloidal es una dispersión de pequeñas partículas en un medio continuo.

en Ensenada, B.C. por investigadores del Instituto de Astronomía de la UNAM, así como especialistas en Óptica de la BUAP.

Al igual que en el caso de imágenes de coloides, la imagen original del ronchigrama es preprocesada hasta obtener una imagen binaria (fig. 3.20), ver capítulo III.

En este caso, lo que se busca es encontrar el número de franjas blancas y negras que existen en una imagen dada. En lugar de obtener centroides, aquí lo que importa es la obtención de la "línea media" de cada región tanto blanca como negra. En fig. 3.24 se muestran en negro las líneas medias de las franjas negras y en gris las de las blancas.

Una vez obtenidas tales "líneas medias", se les asigna una etiqueta de identificación llamada *orden de interferencia* (ver sección 2.3.2). Esta información será utilizada por especialistas para medir la calidad de diferentes dispositivos ópticos.

La organización de este trabajo es la siguiente:
En la sección 1.1 del capítulo I se describen los problemas a resolver; en la sección 1.2 se presentan los distintos paradigmas de programación paralela y en la sección 1.3 se presenta un esbozo de la historia de las computadoras paralelas.

En la sección 2.1 del Capítulo II se describe la arquitectura del hardware y el software (sistema operativo y compilador) utilizados en el desarrollo de este trabajo; en la sección 2.2 se presentan los conceptos de topología digital necesariospara el mismo y la sección 2.3 se describen las partes más importantes de los los programas desarrollados.

Finalmente, en el capítulo III se presentan los resultados más importantes obtenidos al procesar las imágenes de coloides, se muestran resultados de la comparación de la implementación en uno y dos procesadores de la aplicación para el procesamiento de las imágenes de coloides (esto sugiere que se puede dar como una constante de invariancia el valor de 1.7 [5]).
En la parte final del mismo capítulo se muestran los resultados más importantes obtenidos con la aplicación desarrollada para el procesamiento de los ronchigramas.

El lector interesado podrá consultar todos los resultados en los directorios *resultados_coloides* y *resultados_ronchigramas* del disco

# CAPÍTULO I
## PROBLEMAS A RESOLVER Y CONCEPTOS DE PARALELISMO

### 1.1. Descripción general del problema

La importancia de estudiar los modelos de suspensiones coloidales, se debe a su aplicación en sistemas reales como la sangre, la industria de las pinturas, pegamentos, productos lácteos, etc. Por esa razón, el estudio de las propiedades de las suspensiones coloidales monodispersa y bidispersa ha tenido un gran desarrollo en los últimos diez años. Particularmente en los recientes cinco años se ha generado una gran cantidad de trabajo en sistemas cuasibidimensionales, usando para ello la videomicroscopía. Estos estudios se han centrado en el análisis del efecto de las interacciones electrostáticas sobre las mencionadas propiedades estáticas y dinámicas en el rango de tiempos cortos del orden de un milisegundo, y hasta donde se sabe, no se ha trabajado en la dirección de las funciones termodinámicas. Esta línea de investigación conlleva una contribución teórica importante porque hasta el momento se conoce la mecánica estadística de las propiedades de bulto, no así las correspondientes a sistemas confinados.

Para llevar a cabo el estudio experimental de suspensiones coloidales cuasibidimensionales, se requiere el procesamiento de imágenes (fig. 1.1) y la extracción automática de las funciones de distribución radial y el desplazamiento cuadrático medio de las partículas suspendidas; esto desde el punto de vista de la Física Estadística y Mecánica de Fluidos.

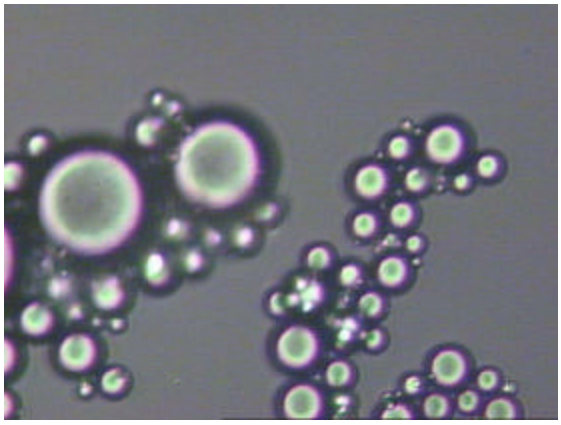


Figura 1.1. Muestra de una suspensión coloidal

Con el fin de calcular la distribución promedio de partículas alrededor de una dada, a ésta se le calcula su centroide, área y diámetro. Posteriormente se trazan anillos concéntricos a tal partícula, de anchura igual a $\frac{1}{10}$ de su diametro. Para cada anillo así generado, se determina el número de partículas que teniendo el mismo tamaño que la dada, tienen centroide dentro del anillo considerado. Este proceso se repite con todas y cada una de las partículas presentes en la imagen.

En definitiva, y ***relacionado con las imágenes de coloides, nuestro trabajo ha tenido como objetivo realizar el conteo anteriormente citado, utilizando programación paralela.***

Cabe decir, que en el caso general es necesario también seguir a una partícula a medida que transcurre el tiempo. Para esto, se debe tomar video de lo que está pasando en la muestra, a fin de tener imágenes secuenciales en el tiempo e igualmente espaciadas.

A través de la cámara se observa sólo una porción pequeña de la muestra, a la cual llamaremos "el sistema". Las partículas estarán entrando y saliendo del sistema, por lo que su número no permanecerá constante. Cada imagen estará contribuyendo al promedio del número de partículas en el sistema, el cual debe coincidir con el número promedio de partículas en la muestra. Este número promedio se conoce desde la misma preparación de la muestra. Hasta aquí sólo es verificar que el procedimiento realizado con las imágenes es correcto.

Hasta el momento los investigadores no se habían interesado en las propiedades termodinámicas de los sistemas confinados, a pesar de la importancia de tales propiedades. Desde el punto de vista de la mecánica estadística esto implica que en el cálculo de tales propiedades se realizaría en el ensamble Gran Canónico, en el cual entra en juego cuántas partículas tiene cada imagen.

Hay otro motivo por el cual es conveniente saber cuantas partículas tiene una imagen.

Como un comentario final en cuanto al proceso de conteo, debemos mencionar que otra propiedad promedio a obtener de las imágenes, es la distribución promedio de partículas dependiente del tiempo. Colocamos $t_1=0$ y tomamos la primera imagen. Se coloca el origen de coordenadas en una de las partículas y se hace el conteo de las partículas que la rodean de acuerdo al procedimiento dado líneas arriba Enseguida se toma la segunda imagen (tiempo igual a $t_2$) y se proyectará la posición de la partícula colocada en el origen en la primera imagen, se hace el conteo nuevamente; con la tercera imagen se hace lo mismo y así hasta terminar con todas las imágenes.

Este procedimiento se repite para la siguiente partícula de la primera imagen y así sucesivamente.

De esta última parte, se ocupan ya otras personas.

Por otra parte, el campo de las pruebas ópticas concierne a la medición de componentes y sistemas ópticos para determinar sus propiedades. Debido a la extrema exactitud que se requiere en los componentes ópticos, la mayoría de las pruebas se realizan durante su fabricación. La necesidad de tales pruebas incrementa el costo de la óptica y abastece fuertemente la motivación para el desarrollo de nuevos procedimientos de prueba exactos y fáciles de seguir. Actualmente los especialistas en el manejo y aplicación de las pruebas ópticas han mejorado la forma de evaluación, a tal grado que ahora se realiza de manera rápida y precisa, brindando una información amplia sobre el sistema óptico probado, determinando no sólo si una componente o el sistema en sí realiza su funcionamiento, sino además se detectan pequeños decaimientos de su nivel, llevando a realizar correcciones.

La prueba de Ronchi es una de las pruebas que más se emplea para evaluar la calidad de la imagen de un sistema óptico. Además es muy útil para la prueba de superficies esféricas, incluyendo sistemas más complejos, como los telescopios

astronómicos. Este método consiste básicamente en colocar una fuente puntual de luz frente a la superficie óptica bajo prueba, así como una rejilla de Ronchi, detrás de la cual se ubica un detector para observar un patrón de franjas, cuyo análisis permite conocer la forma del frente de onda proveniente de la superficie bajo prueba. Para tal análisis es necesario conocer el orden de interferencia de cada franja. Durante el desarrollo y cambios que ha ido teniendo la forma de evaluación, se han propuesto nuevas técnicas para efectuarla, necesitándose de dos patrones para el análisis. En los modernos laboratorios de talleres de óptica, se utiliza en calidad de detector un CCD lo cual permite digitalizar imágenes de interferogramas, ronchigramas y birronchigramas [9].

En el presente trabajo, también hemos considerado imágenes digitales llamadas ronchigramas (fig. 3.13), resultado de pruebas realizadas durante la fabricación de dispositivos ópticos para el TELESCOPIO INFRARROJO MEXICANO que se está diseñando y construyendo en Ensenada, B.C. por investigadores del Instituto de Astronomía de la UNAM, así como especialistas en Óptica de la BUAP.

Al igual que en el caso de imágenes de coloides, la imagen original del ronchigrama es preprocesada (ver capítulo III) hasta obtener una imagen binaria (fig. 3.20).

En este caso, de lo que se trata es de hallar el número de regiones blancas y también cuántas regiones negras existen en la imagen. En lugar de obtener centroides, lo que importa es la obtención de la "línea media" de cada región tanto blanca como negra (fig. 3.24).

Una vez obtenidas las "líneas medias", a éstas se les debe asignar una etiqueta de identificación llamada *orden de interferencia*. ***En este punto, el ha sido la asignación automática de los órdenes de interferencia.***

Como se puede observar de la anterior exposición, se requiere del manejo rápido de gran cantidad de información, razón por la cual se hace indispensable contar con Sistemas de Cálculo Rápido, tanto en hardware como en software.

Más aún, para dar una idea de los requerimientos, cabe decir que en el análisis de sistemas para tiempos casi realeales son necesarias al menos diez mil imágenes de coloides; mientras que para el caso de ronchigramas, la obtención rápida de información es indispensable por la naturaleza de los dispositivos ópticos.

## 1.2. Procesamiento en paralelo

En esta sección se describen los elementos de hardware necesarios para el procesamiento paralelo llevado a cabo en el presente trabajo.
Para esto se inicia con una breve reseña histórica de la programación paralela.

### 1.2.1. Antecedentes históricos del cómputo paralelo [7]

El cómputo paralelo se ha venido desarrollando con el objetivo de hacer equipos cada vez más rápidos. A pesar de los grandes avances logrados para incrementar la velocidad de los equipos electrónicos existen límites físicos que serán difíciles de superar por los sistemas con un único procesador, cuando se llegue a estos límites no habrá otra opción mas que apostar por un mayor desarrollo del cómputo paralelo.

Las tecnologías innovadoras pasan principalmente por tres etapas: la primera es cuando están en etapa experimental, que es cuando investigadores buscan la forma de aplicar estas tecnologías a ciertos problemas. La segunda fase es cuando las empresas se empiezan a interesar en la tecnología y comienzan a comercializarla. La tercer etapa es cuando dicha tecnología es de aceptación general. En nuestros días el cómputo paralelo está en los inicios de la tercer fase, cada vez existen más personas que aplican esta tecnología para diversos propósitos.

### 1.2.2 Primeros prototipos

Los primeros prototipos de investigación eran equipos que variaban desde arquitecturas SIMD[2], SIMD/MIMD[3] o completamente MIMD. Los principios usados para la construcción de estos equipos fueron útiles para las siguientes generaciones. Un listado de los principales equipos se muestra en la Tabla 1.1.

| Máquina | Control | Topología | Memoria | CPU | # elementos de procesamiento | Lugar y Fecha |
|---|---|---|---|---|---|---|
| Illiac IV | (Múltiple)SIMD | Malla 2D | Distribuida | 64 bits | 256(64) | Universidad de Illinois, 1968 |
| MPP | SIMD | Malla 2D reconfigurable | Distribuida | 10 Mhz | 16384 | Goodyear AeroSpace, 1980 |
| HEP | MIMD | Segmentación | Compartida | Multihilos | 4 | Finales de los 70's |
| PASM | SIMD/MIMD | Cubo extra-etapas | Distribuida | MC68010 | 1024(16) | Universidad Pardue, 1980 |
| TRAC | SIMD/MIMD | SW-Banyan | Compartida / Distribuida | 8 bits byte slice | 16 | U.T. Austin, 1977 |
| Manchester | Data-Flow | Red en anillo | Compartida | - | 12 | Universidad de Manchester, 1981 |
| NYU Ultra | MIMD | Red Omega | Compartida | MC68010 | 4096 (8) | NYU, 1983 |
| RP3 | MIMD | Red Omega | Compartida /Distribuida | - | 512 | IBM, 1983 |
| Cosmic Cube | MIMD | Hipercubo en 6F | Distribuida | 8086 | 64 | Caltech, 1980 |

Tabla 1.1. Características de los primeros prototipos de investigación [7].

A continuación se describen solamente los equipos más sobresalientes de la tabla anterior.

#### 1.2.2.1. Illiac IV

En el año de 1966, la Agencia de Proyectos de Investigación del Departamento de Defensa de los Estados Unidos contrató a la Universidad de Illinois para construir la Illiac IV, que comenzó a operar hasta 1972 en el Centro de Investigación Ames de la NASA. Esta

[2] SIMD: Single-Instruction, Múltiple-Data
[3] MIMD: Múltiple-Instruction-Multiple-Data

computadora mejoró la velocidad de procesamiento de 200 millones de operaciones por segundo en un 50% con un billón de bits por segundo de transferencia de entrada/salida(I/O) [12]. En la figura 1.2 se muestra la arquitectura de la Illiac IV.

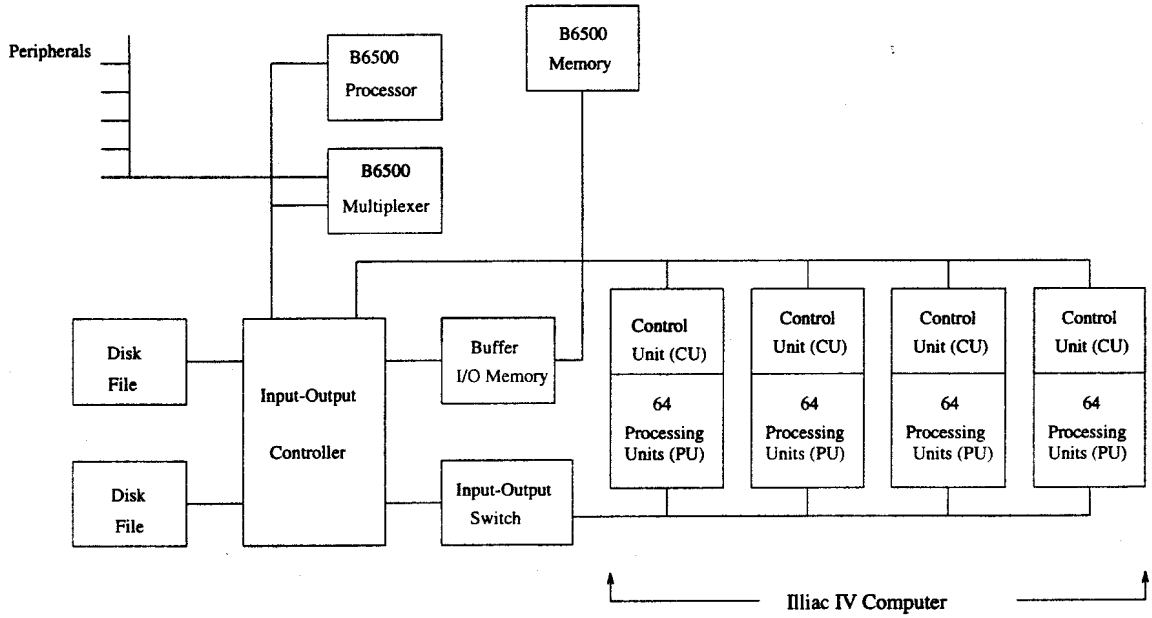



Figura 1.2. Arquitectura de la computadora Illiac IV.

La arquitectura de esta computadora consistía en cuatro arreglos de 64 unidades de procesamiento (PU), cada arreglo era controlado por una unidad de control (CU). Los cuatro arreglos de 64 PU's dan un total de 256 procesadores. Las CU's compartían datos e instrucciones y realizaban instrucciones de control de ciclos. Cada PU podía acceder a 2048 palabras de una memoria RAM con tiempos de acceso de 250ns. La I/O era controlada por el sistema Borroughts B6500.

Cabe notar que antes de la aparición de la Illiac IV ya habían computadoras paralelas, pero la Illiac IV fue la primera en llamar la atención de una gran cantidad de científicos. A pesar de la dificultad de su programación, su rendimiento moderado y su alto costo fue una señal del advenimiento de las supercomputadoras paralelas. La Illiac IV fue sólo un proyecto de investigación, pero hubo mucha gente que se dedicó a desarrollar herramientas de software y lenguajes de programación como Glypnyr y Fortran, estas personas hicieron contribuciones significativas que permitieron el desarrollo de otros equipos posteriores [16].

### 1.2.2.2. PASM (Partitionable SIMD/MIMD)

Construida por Siegel, H. J., Schwederski, T., Kuehn, J. y Davis, N. J. en la Universidad de Pardue. Trabajaba con los esquemas de paralelismo SIMD y MIMD (sección 2.2). Consistía en un arreglo de procesadores que podían ser reconfigurados para trabajar como SIMD y/o MIMD para trabajar como máquinas independientes. La arquitectura de esta computadora se muestra en la figura 1.3.

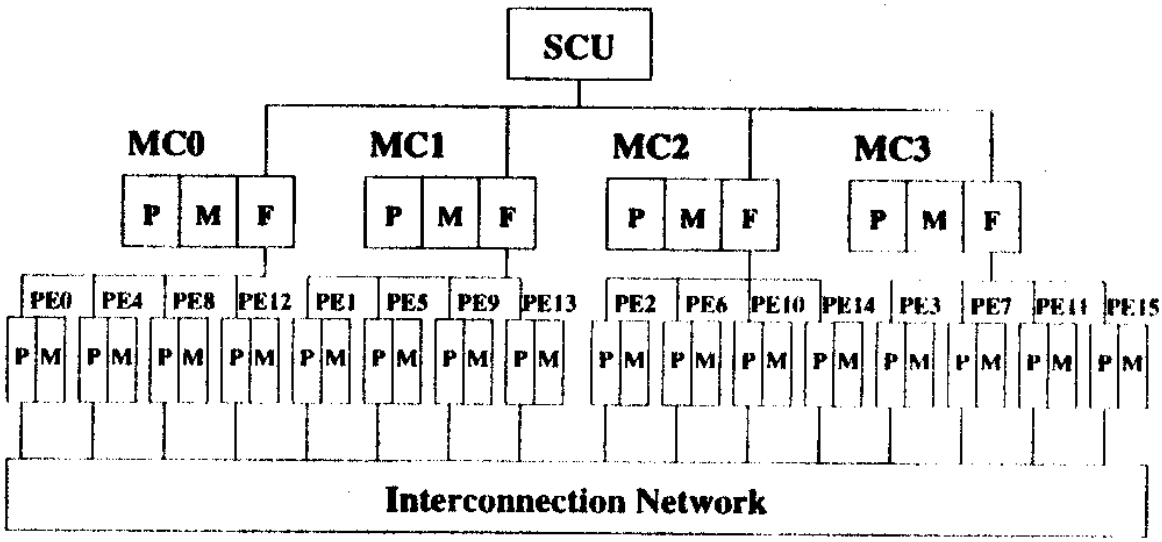



Figura 1.3. Arquitectura de la PASM

En la figura 1.3 se tiene una unidad de control del sistema (SCU) que daba instrucciones a los microcontroladores (MC), que a su vez controlaban a las unidades de procesamiento (PE) para trabajar en modo SIMD o MIMD. Cada microcontrolador actuaba como unidad de control para las PE's y a su vez permitía particionar la máquina [7].

Esta máquina tenía 1024 PE's y 32 MC's (el prototipo inicial tenía 16 PE's y 4 MC's). Las ventajas que ofrecía esta máquina eran flexibilidad en compartición de recursos, tolerancia a fallos, esquemas más fáciles de direccionamiento. Las aplicaciones para las que fue usada fueron: procesamiento de imágenes, procesamiento de señales biomédicas y entendimiento de lenguaje, entre otras [1].

Otra computadora no paralela pero de muy alta velocidad que se introdujo en la década de los 70's fue la CRAY-1, computadora vectorial de un solo procesador que entra en la clasificación de SIMD.

Es importante resaltar la personalidad de Seymour Cray de Cray Research quien diseñó la CRAY-1 que fue la primera capaz de ejecutar más de 100 millones de operaciones de punto flotante por segundo que en su tiempo fuese la más rápida [20].

La computadora paralela HEP (*Denelcor Heterogeneous Element Processor*) era una máquina que a pesar de su alto costo y bajo rendimiento fue adquirida por varias instituciones ya que era más fácil de programar. Algunas instituciones como Los Álamos, Argonne National Laboratory, Ballistic Research Laboratory, y Messerschmidt en Alemania adquirieron estas computadoras. Messerschmidt la usó para aplicaciones reales, mientras que las otras las usaron para realizar investigación en algoritmos paralelos. Estas computadoras soportaban problemas de grande y pequeña granularidad (sección 2.1.4) [10].

La importancia de la NYU Ultracomputer fue en investigación sobre la escalabilidad de sistemas con procesadores paralelos de memorias compartidas.

### 1.2.3 Primera generación

Estas computadoras fueron en principio de empresas comerciales que arriesgaron capital pero que en muchos de los casos resultaron económicamente exitosas [7]. Algunas de las máquinas que fueron desarrolladas en esta época se muestran en la tabla 1.2.

| Máquina | Control | Topología | Memoria | CPU | # elementos de procesamiento | Lugar y Fecha |
|---|---|---|---|---|---|---|
| NCUBE-1 | MIMD | Hipercubo | Distribuida | Propia | 1024 | NCUBE, 1988 |
| GP1000 | MIMD | Comnutador multietapas de cubo. | Compartida/distribuida | MC68020 | 64 | BBN, 1985 |
| Balance | MIMD | Bus compartido | Compartida | NS32032 | 24 | Sequent, 1985 |
| Monsoon | Sin control (Data-Flow) | Paquete multietapas | Compartida | Segmentación propia | 8 | MIT, 1991 |
| TRACE | VLIW | Bus | Compartida | Propia | 4 enteros | Multiflow, 1987 |
| FX/8 | Propio | Bus compartido | Compartida | Propia | 8 | Alliant, 1985 |
| iPSC/1 | MIMD | Hipercubo de 7D | Distribuida | i286 | 128 | Intel, 1985 |
| SUPRENUM | MIMD | Malla toroidal agrupado | Distribuida | - | 256 | GMD FIRST, 1985 |
| MP-1 | SIMD | Malla Xnet de 3 etapas | Distribuida | Propia de 4 bits | 16K | MasPar, 1985 |
| CM-2 | SIMD | Hipercubo de 12D de mallas de 4x4 | Distribuida | Propia de 1 bit | 64K | TMC, 1985 |
| GF11 | SIMD | Conmutador Benes | Distribuida | Propia | 566 | IBM, 1986 |

Tabla 1.2 Características de las computadoras paralelas de la primera generación [7].

#### 1.2.3.1 NCUBE-1

La familia de computadoras NCUBE-1 está basada en el procesador VAX, con capacidad de realizar operaciones de punto flotante. Los procesadores están interconectados en una topología de hipercubo ideada por Charles Seitz y Geoffrey Fox en Caltech [4] (sección 2.3.2).

Las configuraciones de estas computadoras eran desde 4 a 1024 nodos. El sistema I/O llamado *Nchannel* permitía comunicación directa de los canales de I/O con los nodos de la máquina. Cada uno de estos canales permitía conectar de 1 a 7 nodos de I/O, donde cada nodo contenía un procesador NCUBE y 16 canales seriales de I/O. Los nodos de I/O estaban conectados a un controlador periférico localizado junto a la máquina principal [7]. Un diagrama a bloques del procesador se muestra en la figura 1.4.

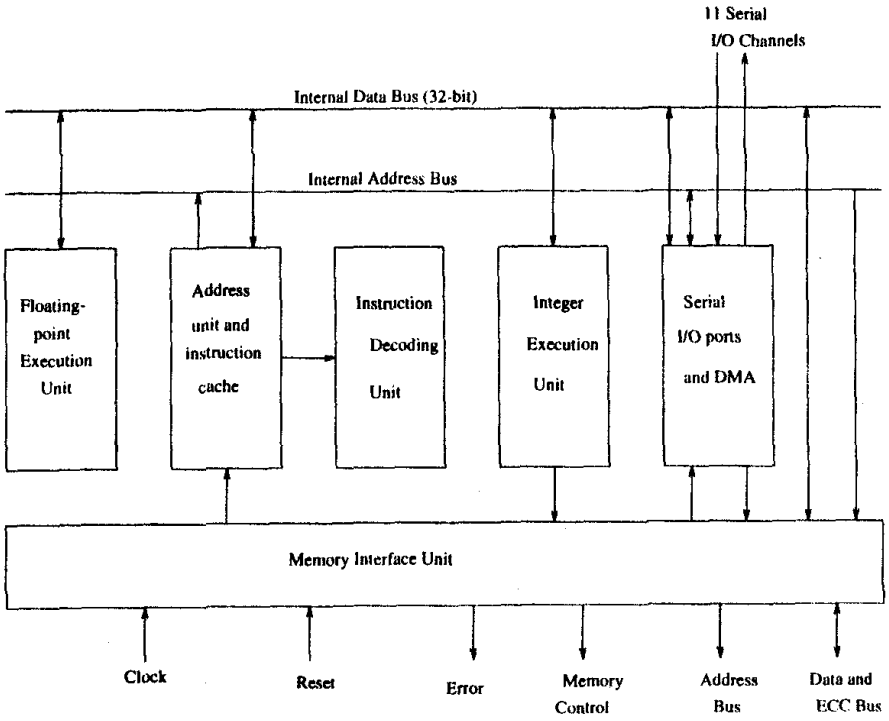


Figura 1.4. Procesador VAX para el NCUBE-1

### 1.2.4. Segunda generación

La mayoría de estas computadoras empezaron a cambiar la memoria compartida por la memoria distribuida. Gracias a la integración a gran escala que permitió, entre otras cosas el poder introducir equipos cada vez más veloces y pequeños, por lo tanto se pudieron construir equipos con más unidades de procesamiento para dar velocidades mayores. En esta generación se puede apreciar una tendencia al control MIMD.

| Máquina | Control | Topología | Memoria | CPU | # elementos de procesamiento | Lugar y Fecha |
|---|---|---|---|---|---|---|
| AP1000 | MIMD | 3 Redes Torus | Distribuida | SPARC | 1024 | Fujitsu, 1991 |
| Cedar | MIMD | Red Omega | Compartida | Vector | 32 | Universidad de Illinois |
| IPSC/2 iPSC/860 | MIMD | Hipercubo | Distribuida | i386/i860 | 128 | Intel, 1988/89 |
| NCUBE-2 | MIMD | Hipercubo | Distribuida | Propia de 64 bits | 8192 | NCUBE, 1992 |
| CM-5 | MIMD | Árbol | Distribuida | Sparc 2 | 16382 | TMC 1992 |
| TC2000 | MIMD | Comnutador multietapas de cubo | Compartida/ Distribuida | MC88000 | 512 | BBN, 1989 |
| Symmetry | MIMD | Bus compartido | Compartida | I386 | 30 | Sequent, 1990 |
| FX/2800 | MIMD | Bus compartido | Compartida | i860 | 28 | Alliant, 1990 |
| MP-2 | SIMD | Malla Xnet | Distribuida | Propio de 32 bits | 16K | MasPar, 1992 |
| KSR1 | MIMD | Red | Compartida | Propio de 64 bits | 1088 | Kendall Square 1989 |

Tabla 1.3. Características de las computadoras paralelas de la segunda generación [7].

### 1.2.5. Tercera generación

Para estas computadoras los avances hechos en varias ramas de la tecnología provocan una variedad mayor de equipos cada vez más complejos y también más comerciales. Los sistemas con memoria compartida prácticamente han desaparecido para optar por la memoria distribuida, dispuesta para cada unidad de procesamiento. La arquitectura MIMD sigue predominando sobre la SIMD. En esta época se comienza a dar un auge mayor al desarrollo de aplicaciones y lenguajes de programación para operar en estos equipos.

| Máquina | Procesador | Topología | Fabricante | MHz | # elementos de procesamiento | Memoria (MB) |
|---|---|---|---|---|---|---|
| T3D | Dec Alpha | 3D Torus | Cray | 150 | 128-2048 | 16-64 |
| VPP500 | Propio GAs, Vector | Barra | Fujitsu | 100 | 4-222 | 128-256 |
| SP1 (SP2) | RS/6000 | Conmutador multietapa. | IBM | 62.5 | 8-64 | 64-256 (64M – 2GB) |
| Paragon XP/S | i860XP | Malla 2D | Intel | 50 | 4-2048 | 64-128 |
| KSR2 | Propia VLSI | Anillo | Kendall Square | 40 | 2-5120 | 32 |
| Cenju-3 | NEC/MIPS CR4400SC | Barra multietapas | NEC | 75 | 8-256 | 32-64 |
| GC | Motorola Pwr PC 601 | Malla 2D | Parsytec | 50 | 16-128 | 2-32 |
| CS-2 | Sparc | Conmutador multietapas | Meiko Limited | 40 | 4-1024 | 32-128 |
| Alewife | SPARCLE | Malla 2D | MIT | 33 | 512 | 8 |
| DASH | MIPS R3000 | Malla 2D | Stanford | 33 | 64 | 256 |
| Tera | Propio GAs | Torus 3D | Tera | 333 | 256 | 256GB |
| DDM | Motorola MC88100 | Árbol agrupado | Swedish Inst. of CS | 20 | 256 | 8-32 |
| SPP | PA-RISC | Anillo múltiple SCI | Convex | 200 | 8-128 | 32-256 |
| J-Machine | Propio | Torus 3D | MIT, Intel | 20 | 65536 | 1 |

Tabla 1.4 Características de las computadoras paralelas de la tercera generación [7].

### 1.2.6. Las computadoras más rápidas del mundo

La siguiente tabla muestra las 25 supercomputadoras más rápidas del mundo, (junio de 2002) [25]. Cabe notar el uso de los procesadores paralelos en estos equipos.

Estos equipos son usados principalmente para fines académicos y de investigación. Los países que tienen las supercomputadoras más rápidas son Japón y los Estados Unidos.

| Posición mundial | Fabricante | Computadora | $R_{max}$ (GFlops) | Lugar en donde opera | País | Año | Fin | Número de procesadores |
|---|---|---|---|---|---|---|---|---|
| 1 | NEC | Earth-Simulator | 35860.00 | Earth Simulator Center | Japan | 2002 | Investigación | 5120 |
| 2 | IBM | ASCI White, SP Power3 375 MHz | 7226.00 | Lawrence Livermore National Laboratory | USA | 2000 | Investigación | 8192 |
| 3 | Hewlett-Packard | AlphaServer SC ES45/1 GHz | 4463.00 | Pittsburgh Supercomputing Center | USA | 2001 | Académico | 3016 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4 | Hewlett-Packard | AlphaServer SC ES45/1 GHz | 3980.00 | Commissariat a l'Energie Atomique (CEA) | France | 2001 | Investigación | 2560 |
| 5 | IBM | SP Power3 375 MHz 16 way | 3052.00 | NERSC/LBNL | USA | 2001 | Investigación | 3328 |
| 6 | Hewlett-Packard | AlphaServer SC ES45/1 GHz | 2916.00 | Los Alamos National Laboratory | USA | 2002 | Investigación | 2048 |
| 7 | Intel | ASCI Red | 2379.00 | Sandia National Laboratories | USA | 1999 | Investigación | 9632 |
| 8 | IBM | pSeries 690 Turbo 1.3GHz | 2310.00 | Oak Ridge National Laboratory | USA | 2002 | Investigación | 864 |
| 9 | IBM | ASCI Blue-Pacific SST, IBM SP 604e | 2144.00 | Lawrence Livermore National Laboratory | USA | 1999 | Investigación | 5808 |
| 10 | IBM | pSeries 690 Turbo 1.3GHz | 2002.00 | IBM/US Army Investigación Laboratory (ARL) | USA | 2002 | Ventas | 768 |
| 11 | IBM | SP Power3 375 MHz 16 way | 1910.00 | Atomic Weapons Establishment | UK | 2002 | Clasificado | 1920 |
| 12 | IBM | pSeries 690 Turbo 1.3GHz | 1840.00 | IBM/ECMWF | USA | 2002 | Vendor | 704 |
| 13 | Hitachi | SR8000/MPP | 1709.10 | University of Tokyo | Japan | 2001 | Académico | 1152 |
| 14 | Hitachi | SR8000-F1/168 | 1653.00 | Leibniz Rechenzentrum | Germany | 2002 | Académico | 168 |
| 15 | SGI | ASCI Blue Mountain | 1608.00 | Los Alamos National Laboratory | USA | 1998 | Investigación | 6144 |
| 16 | IBM | SP Power3 375 MHz | 1417.00 | Naval Oceanographic Office (NAVOCEANO) | USA | 2000 | Investigación | 1336 |
| 17 | IBM | SP Power3 375 MHz 16 way | 1293.00 | Deutscher Wetterdienst | Germany | 2001 | Investigación | 1280 |
| 18 | IBM | SP Power3 375 MHz 16 way | 1272.00 | NCAR (National Center for Atmospheric Investigación) | USA | 2001 | Investigación | 1260 |
| 19 | NEC | SX-5/128M8 3.2ns | 1192.00 | Osaka University | Japan | 2001 | Académico | 128 |
| 20 | IBM | SP Power3 375 MHz | 1179.00 | National Centers for Environmental Prediction | USA | 2000 | Investigación | 1104 |
| 21 | IBM | SP Power3 375 MHz | 1179.00 | National Centers for Environmental Prediction | USA | 2001 | Investigación | 1104 |
| 22 | Cray Inc. | T3E1200 | 1127.00 | Government | USA | 2001 | Clasificado | 1900 |
| 23 | IBM | SP Power3 375 MHz 16 way | 1100.00 | Lawrence Livermore National Laboratory | USA | 2001 | Investigación | 1088 |
| 24 | NEC | SX-6/128M16 | 982.00 | NEC Fuchu Plant | Japan | 2002 | Ventas | 128 |
| 25 | IBM | SP Power3 375 MHz 8 way | 929.00 | UCSD/San Diego Supercomputer Center | USA | 2000 | Académico | 1152 |

**Tabla 1.5.** Las 25 supercomputadoras más rápidas del mundo [25]

## 1.3. Cómputo paralelo [14]

El hacer cómputo paralelo ha cambiado el paradigma de programación tradicional, y como consecuencia la forma de plantear los problemas. En este nuevo paradigma hay que tomar en cuenta todas las partes que pueden cooperar para poder llegar a la meta común. En este modelo se debe hacer un uso adecuado de los recursos de manera tal que sean aprovechados eficazmente, y así poder hacer uso pleno de la ventaja de tener múltiples unidades de procesamiento. El cambio en el paradigma ha traído también nuevos problemas tanto en hardware como en software como es el caso del cuello de botella de Von Neumann (véase sección 1.3.4.1).

La escritura de programas paralelos se dificulta a medida que el número de procesadores aumenta. Para ilustrar esto supóngase la siguiente analogía: se tiene un trabajo que realizar y para ello se cuenta por un lado con una sola persona y por otro lado con un grupo de 20 personas. La persona que labora sola hará el trabajo en un tiempo determinado y entregará sus resultados sin tener que ponerse de acuerdo con alguien más. El grupo de personas tendrá primero que ponerse de acuerdo para distribuirse las actividades, establecer mecanismos de control y una forma de entregar los resultados de la actividad. La parte de "ponerse de acuerdo" es la que dificulta la programación de aplicaciones que funcionen en computadoras con múltiples procesadores. Ya que si suponemos que el número se incrementa, por ejemplo a 200, la dificultad de "ponerse de acuerdo" aumentará y lo seguirá haciendo cada vez que se aumenten personas al grupo.

### 1.3.1. Paradigmas de paralelismo [14]

**Paralelismo**: es el proceso de realizar tareas concurrentemente [14].

**Paradigma**: es un modelo del mundo que se usa para formular una solución computacional a un problema [14].

A partir de las dos definiciones anteriores se puede entender que un paradigma es un modelo de pensamiento que puede ayudar a resolver un problema real. Usando el modelo se puede abstraer los detalles de un problema que permitan resolverlo de manera más fácil. Debido a que un paradigma es una aproximación a la realidad, la solución no es exacta ya que siempre habrán detalles no cubiertos.

Algunos de los paradigmas de paralelismo más importantes se muestran en la figura 1.5.

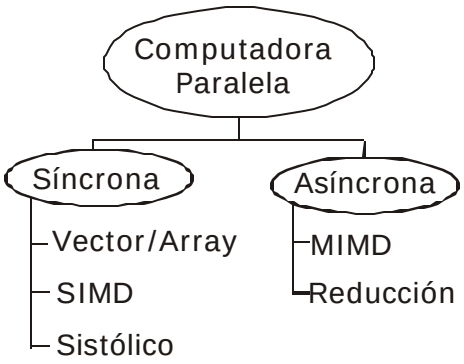



Figura 1.5 Paradigmas de paralelismo [14].

Como se puede apreciar, los paradigmas se pueden dividir en dos enfoques principales, para computadoras síncronas y para asíncronas.

Cuando las tareas que se están realizando son dependientes entre sí, es necesario un mecanismo que permita coordinarlas. El enfoque síncrono coordina las acciones desde el hardware, forzando a que todas las operaciones sean hechas al mismo tiempo eliminando las dependencias de una tarea con otra. En el segundo enfoque se cuenta con mecanismos de coordinación llamados *cerraduras* para coordinar a los procesadores, es la forma más general de paralelismo ya que los procesadores trabajan sobre sus tareas "libremente" sin necesidad de un mecanismo global de sincronización.

#### 1.3.1.1. Paradigma Vector/Array

En este paradigma a cada procesador se le asigna una tarea específica, es un caso muy similar al de una línea de ensamble. El primer procesador recibe los datos iniciales, la salida de éste es recibida por otro para realizar la siguiente tarea, éste último genera otra salida que será recibida por otro procesador y así sucesivamente hasta concluir el proceso como se muestra en la figura 1.6.

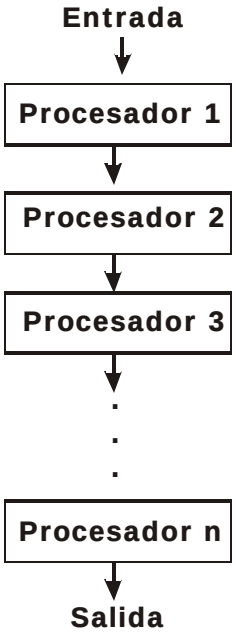



Figura 1.6 Paradigma Vector/Array

Este paradigma es útil cuando una tarea puede ser divida en etapas. Existen muchas aplicaciones numéricas que involucran el uso de matrices y vectores que usan este paradigma, por eso es llamado *vector/array (vector/matriz).*
Este modelo puede aplicarse por ejemplo cuando a una imagen (o algún conjunto de datos) se le quiere aplicar varios filtros, un procesador puede leer la imagen y aplicar un filtro, el siguiente procesador puede tomar la imagen a la que se le aplicó el primer filtro y aplicarle un segundo filtro y así sucesivamente. En este esquema, se pueden estar procesando varias imágenes al mismo tiempo, dependiendo del número de etapas.

Otro ejemplo es cuando una imagen se introduce a un procesador, éste le hace algún proceso y obtiene ciertas características de interés, como puede ser la posición de un objeto. Posteriormente las características obtenidas son transferidas a otro procesador cuyo único fin es hacer un seguimiento de la partícula detectada en el primer procesador. Como se puede ver aquí, la programación es más transparente ya que se puede programar a cada procesador para realizar una tarea específica, sólo hay que preocuparse de recibir y enviar bien las variables necesarias para la comunicación. Un aspecto que se debe cuida en este paradigma es que haya un equilibrio de carga en los procesadores para aprovechar el tiempo de procesamiento al máximo y no desperdiciarlo al tener algún procesador ocioso.

#### 1.3.1.2. Paradigma SIMD

En este paradigma todos los procesadores realizan la misma tarea al mismo tiempo, en caso de no haber suficientes tareas para los procesadores, quedan en estado ocioso (*idle*). Las tareas realizadas afectan a múltiples datos, por eso el nombre SIMD (*Single-Instruction, Múltiple-Data*), este paradigma también es conocido como programación de datos paralelos (*data-parallel programming*). La figura 1.7 muestra de manera gráfica este paradigma.

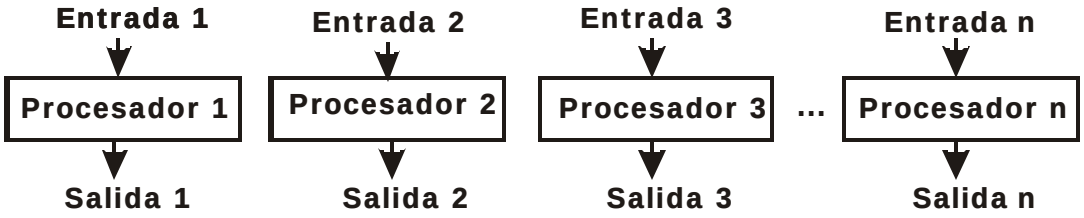



Figura 1.7. Paradigma SIMD

En este modelo la parte medular es el reparto de los datos que deben ser procesados en las diferentes unidades de procesamiento. Por ello, los problemas son divididos en dos fases:

1) Partición y distribución de los datos
2) Procesamiento de los datos en paralelo.

En el primer proceso se aplican criterios que permitan dividir los datos para ser procesados, estos datos en general deben ser uniformes. En la segunda etapa se aplica el mismo proceso a cada flujo de datos. Un ejemplo de la aplicación de este modelo se puede dar cuando se leen señales de varios dispositivos iguales al mismo tiempo, pero todas las señales son del mismo tipo, y por lo tanto el proceso que se les debe aplicar es el mismo para obtener ciertos resultados.

Para aprovechar eficientemente SIMD es necesario emparejar el número de procesadores con los flujos de datos a procesar.
La *MasPar Parallel Computer* y la *Thinking Machines Connection Machine* (véanse las tablas 1.2 y 1.3) son ejemplos de computadoras que usan este paradigma.

Como un ejemplo más de este paradigma, en los laboratorios de software para microcomputadoras de Intel Corp [3] se hicieron pruebas al Intel C++/Fortran Compiler desarrollado en dicho lugar. Este compilador optimiza código para aplicaciones SIMD de acuerdo a ciertos tipos de datos que ofrece. Una de las aplicaciones de prueba fue la factorización LU[4] sin pivoteo, esta aplicación es de granularidad pequeña.
En la figura 1.8 se muestra el incremento de velocidad obtenido al usar un equipo con procesadores duales contra el uso de un equipo con procesadores quad para dicha aplicación.

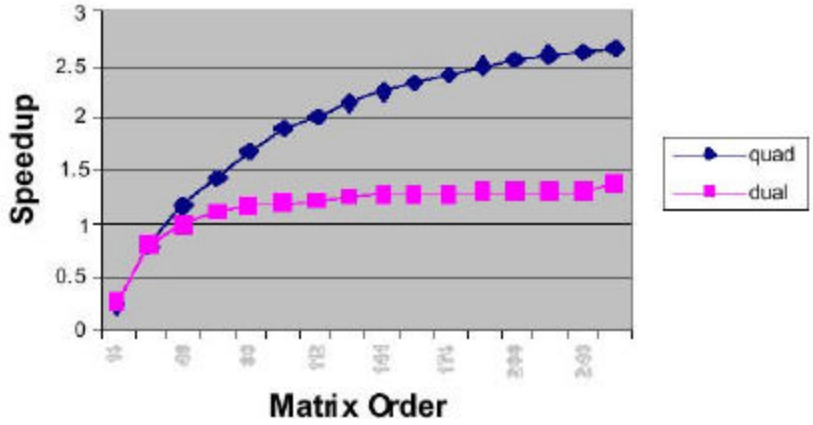


Figura 1.8. Curvas de rapidez *dual* vs. *Quad* para la Factorización **LU** [3]

### 1.3.1.3. Paradigma Sistólico

Este paradigma fue inventado en los inicios de los 80's por H. T. Kung de la Carnegie Mellon University. Una computadora paralela sistólica es un sistema de múltiples procesadores interconectados en el cual los datos son distribuidos de memoria a un arreglo de procesadores. Este paradigma hace la combinación de los dos tipos de paradigmas anteriores. Además de que busca reducir el tiempo de retardo ocasionado por los accesos a memoria.

[4] Factorización de una matriz como el producto de una matriz triangular superior y una matriz triangular inferior.

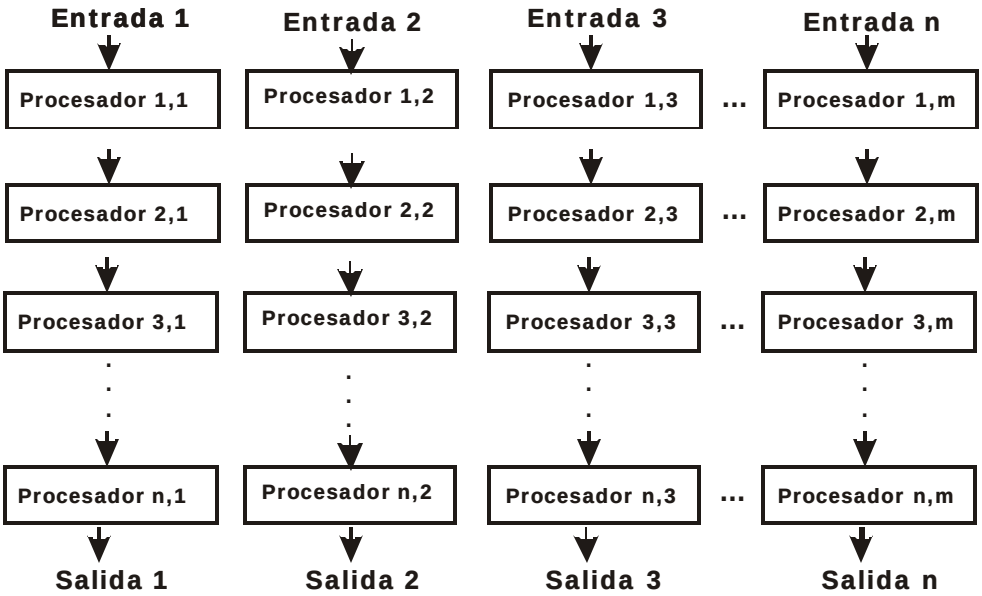


Figura 1.9. Paradigma sistólico

Este modelo se puede aplicar para el primer ejemplo de la sección 1.3.1.2 donde se tienen *m* señales (según la figura 1.9) que pueden ser tratadas por *m* vectores de procesadores, y cada vector de procesadores tiene *n* elementos, por lo tanto el procesamiento de la señal tiene *n* etapas. Esto implica que la señal *i* es tratada por los procesadores (1,*i*), (2,*i*), (3,*i*), ... (*n*,*i*).

### 1.3.1.4. Paradigma MIMD.

Antes de describir este paradigma, es necesario recordar el concepto de *granularidad.* Ésta se refiere al número de operaciones de cómputo realizadas en cada módulo del algoritmo paralelo [26]. La granularidad puede dividirse en dos:

- **Granularidad gruesa.-** ocurre cuando en el algoritmo hay un número elevado de operaciones de cómputo y como consecuencia tiene pocas operaciones de comunicación.
- **Granularidad fina.-** ocurre cuando el algoritmo tiene pocas operaciones de cómputo y un elevado número de comunicaciones.

En este modelo, múltiples procesadores pueden realizar varias instrucciones sobre diferentes datos (MIMD, *Múltiple-Instruction, Múltiple-Data*). Los procesadores pueden operar de manera autónoma, existen mecanismos de sincronización para cumplir con la exclusión mutua, esto es que sólo se pueda modificar un solo dato a la vez para evitar inconsistencias en los mismos.
Este paradigma se puede aplicar en problemas que presentan granularidad gruesa debido al sobreflujo de información que se presentaría al pasarse datos de una tarea a otra.

Para el presente trabajo se utilizó este paradigma, ya que la imagen es dividida en dos submatrices, a cada una de las cuales se le aplica el mismo tratamiento; después de cada cierto tiempo uno de los dos procesadores debe reunir los resultados de ambos para

producir un resultado que posteriormente comparte con el otro procesador que espera dichos resultados para continuar con otra tarea. El hecho de que en algunas partes se lleva a cabo un proceso diferente hace la diferencia entre este modelo y los paradigmas síncronos.

### 1.3.2. Taxonomía de Flynn

Debido a que el paralelismo puede presentarse a muchos niveles, es necesario hacer una clasificación. Flynn [17] en 1996 propuso un modelo para establecer una clasificación, misma que hizo en base a los componentes que la integran, número de instrucciones paralelas y los flujos de datos. La clasificación es la siguiente:

- Flujo único de instrucciones, flujo único de datos (SISD).
- Flujo único de instrucciones, flujo múltiple de datos (SIMD).
- Flujos múltiples de instrucciones, flujo único de datos (MISD).
- Flujos múltiples de instrucciones, flujos múltiples de datos (MIMD).

Esta es una clasificación primaria ya que existen máquinas híbridas de estas categorías.

Enseguida describimos las clases anteriores.

#### 1.3.2.1. Arquitectura SISD

Las computadoras que entran en la clasificación SISD tienen un único procesador y ejecutan una sola instrucción a la vez [26]. Las computadoras de escritorio entran en esta clasificación. Existe un modelo donde se puede ejecutar más de una instrucción al mismo tiempo utilizando segmentación [17]. Los equipos con estas características están incluidos en esta arquitectura.

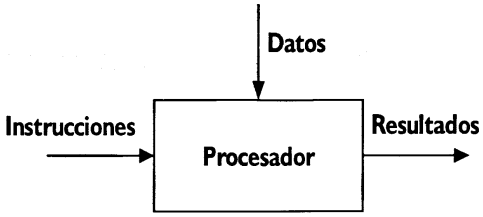



Figura 1.10. Arquitectura SISD [26]

#### 1.3.2.2. Arquitectura SIMD

En esta arquitectura se tienen *p* procesadores idénticos, los cuales poseen una memoria local. Trabajan bajo un solo flujo de instrucciones originado por una unidad central de control, por lo que se tienen *p* flujos de datos [26].

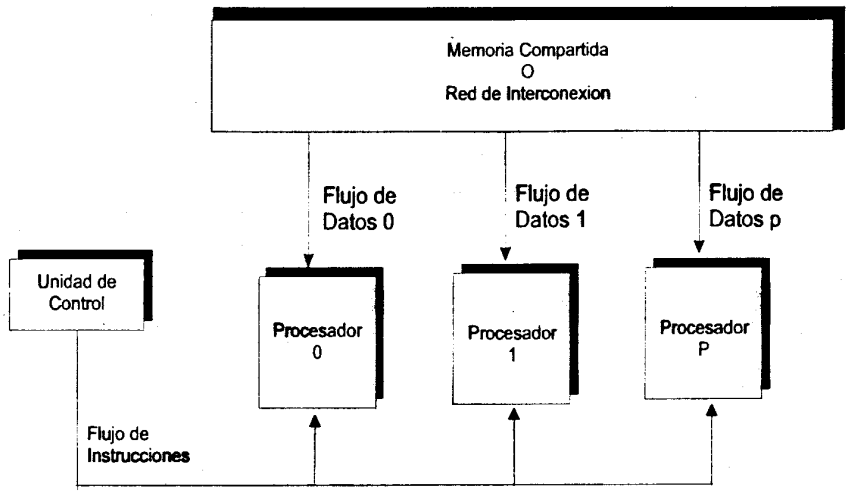


Figura 1.11. Arquitectura SIMD [26]

Los sistemas SIMD operan sobre vectores de datos. La diferencia entre las SISD y las SIMD es que en las SIMD cada unidad de ejecución tiene sus propios registros de direcciones y por ello cada unidad puede tener diferentes direcciones para los datos[16].

Una ventaja sobre las SISD es que necesitan solamente una copia del código a ejecutar y por lo tanto sus requerimientos de memoria son menores (no necesitan una copia del código para cada unidad de ejecución). Esta ventaja ha venido siendo menos significativa debido a que los costos de memoria han estado reduciéndose.

La fortaleza de los equipos SIMD está en el manejo de arreglos con ciclos repetitivos, debido a esto estas máquinas son muy eficientes para aplicaciones con paralelismo de datos. Su debilidad está en la ejecución de sentencias de selección múltiple donde su rendimiento se disminuye a un *1/n* donde n es el número de casos.

Ejemplos de estas máquinas son la Connection Machine 2 (véase tabla 1.3) de la empresa Thinking Machines con 65535 procesadores de un bit, 64 Kbits de memoria y conexiones que permiten que los procesadores se comuniquen entre sí [17]. Otras máquinas conocidas son: Illiac IV de la Universidad de Illinois, la DAP de ICL, la MPP de Goodyear y la MP-1216 de Maspar.

### 1.3.2.3. Arquitectura MISD

Esta arquitectura es solamente un modelo teórico, ya que hasta el momento no se ha diseñado ninguna computadora que efectúe múltiples instrucciones para un solo conjunto de datos. La figura 1.12 muestra esta arquitectura.

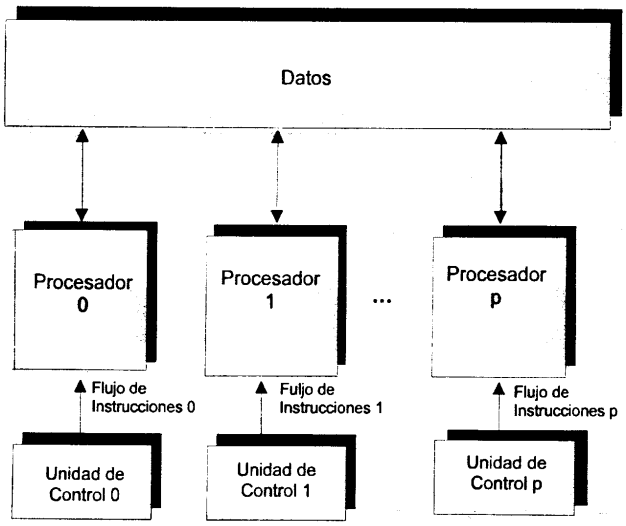


Figura 1.12. Arquitectura MISD [26]

### 1.3.2.4. Arquitectura MIMD

La arquitectura MIMD está conformada por *p* procesadores, *p* flujos de instrucciones y *p* flujos de datos. Cada procesador trabaja de modo asíncrono bajo el control de un flujo de instrucciones proveniente de su propia unidad de control [26].

Para el presente trabajo, el equipo con que se utilizó tiene *p=2*, es decir 2 procesadores que trabajan sobre 2 flujos de instrucciones que operan sobre 2 flujos de datos distintos.

Los equipos MIMD tienen como objetivo crear computadores muy potentes utilizando muchos procesadores de poca capacidad, pretendiendo obtener un rendimiento proporcional al número de procesadores usados. En hardware estos sistemas son muy escalables. Los sistemas MIMD son útiles cuando el problema a resolver permite ejecutar al mismo tiempo varias tareas múltiples y heterogéneas que necesitan sincronización muy ocasional [17].

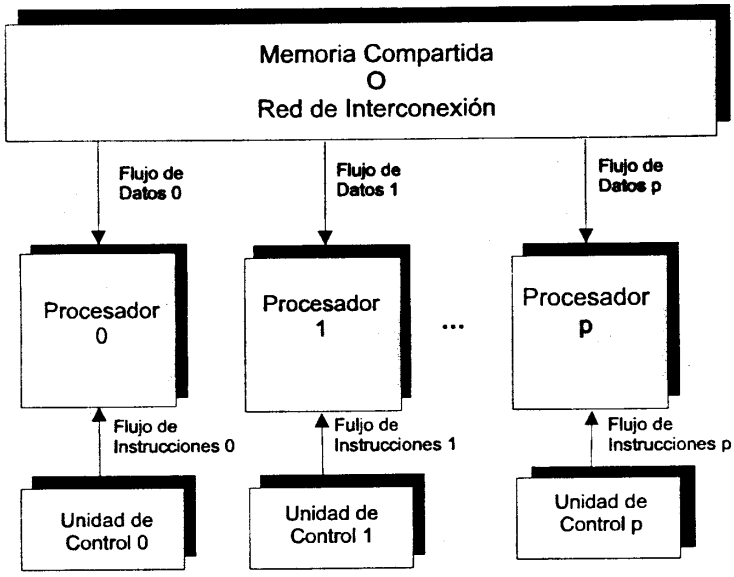



Figura 1.13. Arquitectura MIMD [26]

Existen dos modelos principales de MIMD para compartir información: los de memoria compartida y los de paso de mensajes. En el modelo de paso de mensajes se utilizan emisores de mensajes como medio de comunicación, en este modelo la optimización de la comunicación es más fácil, pero el costo de procesamiento para efectuar la comunicación es mayor. En el modelo de memoria compartida la comunicación se hace a través de variables compartidas pero aunque la carga y almacenamiento de las variables es menos costosa la planeación de la comunicación para el programador es más difícil.

Hay tres características que se busca maximizar en los sistemas MIMD: buen ancho de banda, baja latencia y poder tener un número grande de procesadores interconectados. Existen dos caminos a seguir: construir máquinas tales que sus procesadores compartan un mismo bus de datos y tener máquinas que estén interconectadas por una red. La primer opción tiene un buen ancho de banda, baja latencia pero la cantidad de procesadores está limitada por la velocidad del bus en estas máquinas se encuentran las máquinas con procesadores paralelos como la usada para realizar el presente trabajo. La otra opción está menos limitada al número de equipos interconectados a través de una red, pero presenta deficiencias en la latencia y ancho de banda, además de la necesidad de usar protocolos para comunicación a través de la red. Los clústers son ejemplos de sistemas que quedan dentro de la segunda opción.

La arquitectura MIMD es de particular interés para el desarrollo del presente trabajo ya que el equipo usado tiene esta arquitectura. Los flujos de datos son independientes pues cada procesador tiene su propia unidad de control y los datos procesados son independientes entre sí; cada procesador tiene una copia del programa, pero independientemente de esto cada procesador puede trabajar sobre un flujo de instrucciones distintas.

### 1.3.3. Clasificación de acuerdo a la memoria

De acuerdo al manejo de memoria compartida o local, se tiene la siguiente clasificación.

#### 1.3.3.1. Arquitecturas con memoria compartida

En la arquitectura PRAM (Parallel Random Access Machine) se tiene un conjunto de *p* procesadores que comparten un espacio de memoria común; además, cada uno cuenta con una memoria local. El espacio de memoria común tiene la función de permitir la comunicación entre los procesadores. Cada procesador puede, en un momento dado acceder a la memoria compartida con el fin de leer o escribir un dato en una cierta dirección de la memoria [26].

Este modelo da lugar a cuatro categorías de computadoras según el grado de restricción del acceso simultáneo a una misma dirección de memoria por dos o más procesadores [26]:

- EREW, Lectura exclusiva, escritura exclusiva (*Exclusive-Read, Exclusive-Write*).
- CREW, Lectura concurrente, escritura exclusiva (*Concurrent-Read, Exclusive-Write*).
- ERCW, Lectura exclusiva, escritura concurrente (*Exclusive-Read, Concurrent-Write*).
- CRCW, Lectura concurrente, escritura concurrente (*Concurrent-Read, Concurrent-Write*)

El equipo usado para el presente trabajo tiene memoria compartida bajo el esquema EREW. Las librerías de paso de mensajes hacen que la lectura y escritura en la memoria compartida sea transparente para el programador, en el sentido de que es el mismo programador quien determina en qué momento y qué tareas le asigna a cada procesador.

#### 1.3.3.2. Arquitecturas con memoria distribuida

Estas arquitecturas se basan en que la distribución de la memoria se puede representar por grafos de procesadores interconectados, entre las principales arquitecturas están [26]:

a) Arreglo lineal
b) Anillo
c) Anillo con cuerda
d) Árbol binario
e) Malla
f) Toroide
g) Hipercubo

En particular, la arquitectura de hipercubo es de las más eficientes; además tiene la capacidad de simular otras topologías (que se muestran en la lista anterior). En esta arquitectura se tienen $n=2^d$ procesadores trabajando con un número de vecinos igual a $d$, esto implica los datos que se estén comunicando recorrerán distancias menores y por tanto

se obtendrán mayores velocidades de transferencia de datos. En la figura 1.14 se muestra un ejemplo de un hipercubo.

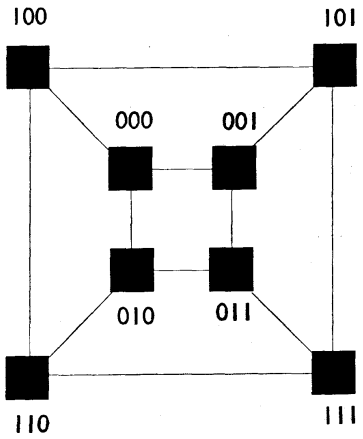



Figura 1.14. Hipercubo de dimensión 3 con 8 procesadores.

### 1.3.4. El papel del rendimiento

Las medidas de rendimiento permiten saber desde diferentes puntos de vista la mejora que se hace al modificar un programa secuencial e implementarlo en paralelo; sin embargo, existen varios factores que pueden afectar el rendimiento.

La medida más común es la curva de rapidez [14], ésta se calcula dividiendo el tiempo usado en computar una solución usando un solo procesador, entre el tiempo empleado en la implementación con *N* procesadores en paralelo.

#### 1.3.4.1. El cuello de botella de Von Neumann

Las máquinas secuenciales han sido creadas usando el modelo de Von Neumann, quien desarrolló también ciertas ideas sobre algunos obstáculos que tendría la construcción de computadoras paralelas (hizo sus consideraciones basado en cuestiones prácticas de su época).

El modelo de Von Neumann consiste de una única unidad de control conectando una memoria a la unidad de procesamiento como se muestra en la figura 1.15. Las instrucciones y datos son obtenidos uno a la vez de la memoria para la unidad de procesamiento. La velocidad de procesamiento del sistema está limitada entonces por la rapidez en que los datos e instrucciones pueden ser transportados desde la memoria hasta la unidad de procesamiento. Esta conexión entre la memoria y la unidad de procesamiento es conocida como el cuello de botella de Von Neumann [14].

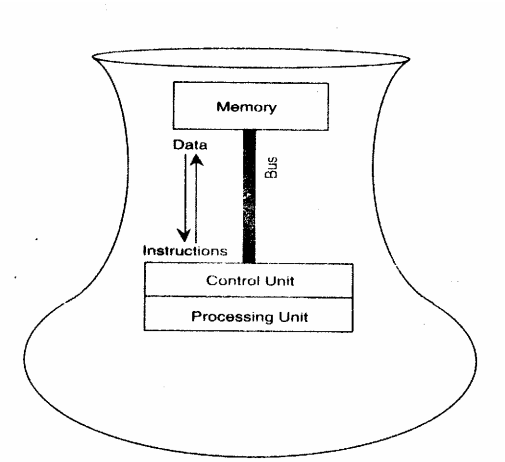



Figura 1.15. Cuello de botella de Von Neumann

Este cuello de botella puede ser evitado empleando más de una unidad de procesamiento y muchas memorias, así varios flujos de datos pueden estar activos al mismo tiempo. Finalmente todos los elementos anteriores se pueden interconectar usando una red de interconexión dando lugar a una computadora del tipo 'no-Von Neuman' (*non-von*) ya que no sigue el modelo de Von Neumann [14].

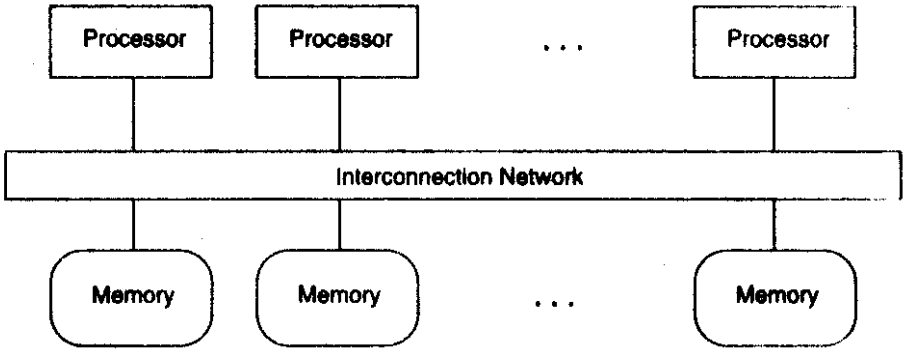



Figura 1.16 Estructura general de una *computadora no Von Neumann*

El equipo usado en el presente trabajo está incluido en el tipo de tecnología llamado de multiprocesamiento simétrico (SMP, *Symmetric Multiprocessing*) y es del tipo no-Von Neumann ya que cada procesador tiene una memoria local (caché), y cuando es necesario compartir datos hace uso de una memoria global compartida como se muestra en la figura 1.17.

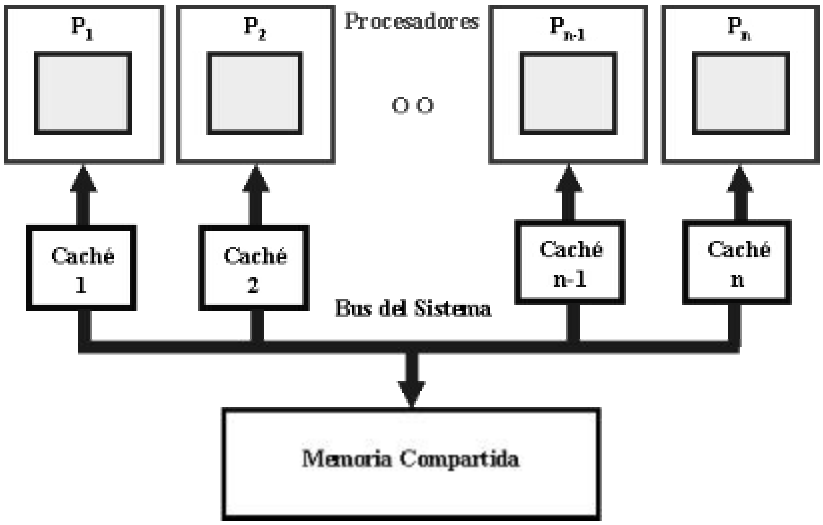



Figura 1.17. Arquitectura de un sistema SMP [15]

En la figura 1.17 se ilustra un caso de computadoras no-Von Neumann, una computadora con multiprocesamiento simétrico. Se puede apreciar que la única diferencia entre la figura 1.16 y la 1.17 es la memoria compartida, a ésta sólo se accede cuando se requiere compartir datos, de otra forma se hace uso del caché.

En la figura 1.17 se puede apreciar que los procesadores comparten la memoria RAM y el bus del sistema, esto en la práctica simplifica la programación de aplicaciones y la sincronización de los datos. Estos dispositivos compartidos tienen una limitante ya que al aumentar el número de procesadores el tráfico se incrementa y por lo tanto la escalabilidad se puede ver limitada, estos problemas pueden reducirse al implementar programas que tengan granularidad grande ya que en estos problemas el uso del bus sería menor.

**1.3.4.2. La ley de Amdahl**

Un número pequeño de operaciones secuenciales puede limitar el factor de aceleración de un algoritmo paralelo. Sea $f$ la fracción de operaciones de un algoritmo que deben ser ejecutadas secuencialmente, en donde $0<f<1$. La máxima aceleración $S_p$ alcanzada usando $p$ procesadores está dada por [26]:

$$S_p \leq \frac{1}{f + \frac{(1-f)}{p}} \qquad (1)$$

A partir de esta ley se puede deducir que un pequeño número de operaciones secuenciales en un algoritmo paralelo puede limitar de manera significativa la aceleración alcanzada por el algoritmo [26].

En la figura 1.18 se puede apreciar la aplicación de la ley de Amdahl para f=0.10, note que la curva no puede atravesar el límite de $S_p$=10. Por lo tanto, independientemente del número de procesadores que se utilicen la mayor aceleración que se le puede dar al algoritmo es 10.

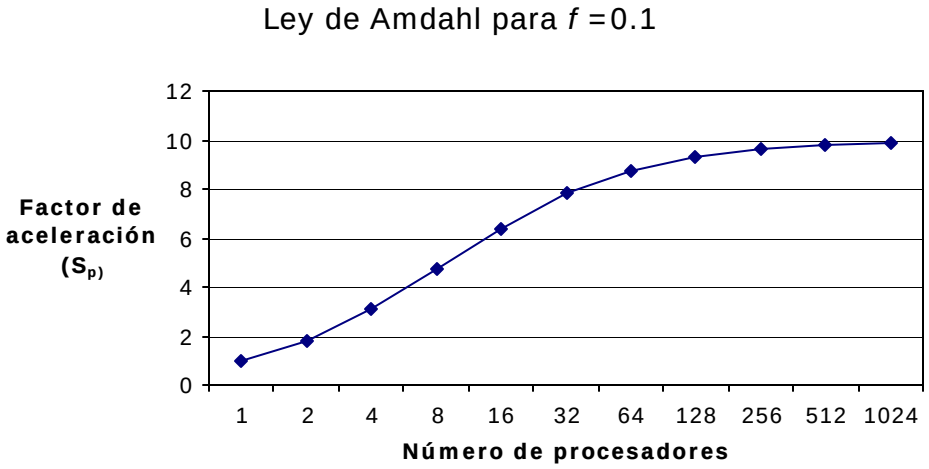



Figura 1.18 Aplicación de la Ley de Amdahl

### 1.3.4.3. Curvas de velocidad (Speedup curves)

El propósito de estas curvas es comparar el tiempo del programa serial más rápido $T^1$ contra el tiempo del programa paralelo equivalente T(*N*), donde *N* es el número de procesadores usados. En la práctica se utiliza T(1) como una aproximación a $T^1$ debido a la dificultad para obtenerlo. Entonces se obtiene *S* como el cociente de T(1) y T(*N*):

$$S \leq \frac{T(1)}{T(N)} \qquad (2)$$

En la figura 1.19 se muestran las curvas de incremento de velocidad para un problema del tipo trivialmente paralelo (a 45º), la siguiente curva es para un programa que usa comunicación intensiva y la curva que crece menos es de un programa que usa la estrategia divide y vencerás, todos ellos usando desde 1 hasta 10 procesadores que se muestran en el eje X, el eje Y muestra el valor de S.

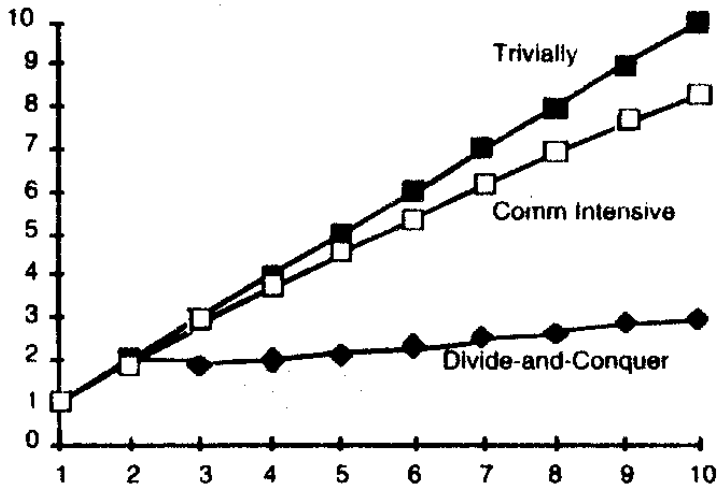


Figura 1.19. Curvas de incremento de velocidad

El incremento del número de procesadores no siempre garantiza la disminución del tiempo en el que es realizada una determinada tarea. Esto se muestra en el siguiente ejemplo aportado por Instituto de Supercómputo de Minesota [13] se hicieron pruebas al modelo de paralelización JavaSpMT (Java Speculative MultiThreading), que son librerías específicas para la programación de hilos. Una de las aplicaciones de prueba que se hicieron fue para resolver sistemas de ecuaciones lineales de *nxn* usando el método de eliminación gausiana. Las pruebas fueron hechas en un sistema multiprocesador IP25 SGI Challenge a 196 MHz con memoria compartida y 8 procesadores MIPS R10000.

Los resultados arrojaron las siguientes curvas de incremento de velocidad:

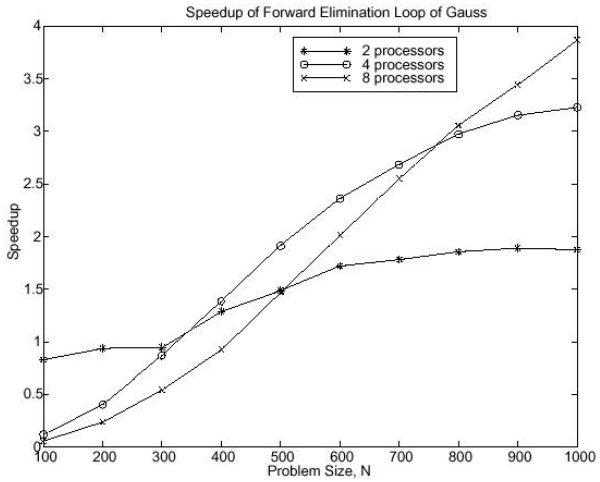


Figura 1.20. Curvas de rapidez al utilizar 2, 4, y 8 procesadores para resolver un sistema de ecuaciones lineales de $N = n$x$n$, por el método de Gauss [13].

Como puede observarse, si $N \leq 300$ el utilizar dos procesadores resulta más rápido que usar más; mientras que si $300 < N \leq 750$, es mejor utilizar cuatro procesadores. Sin embargo, para un sistema con más de 750 ecuaciones, resultan mejor ocho procesadores.

En el presente trabajo, se construyó la correspondiente curva de rapidez para comparar el proceso secuencial y el que se programó en paralelo. Tal curva será mostrada en el capítulo III.

# CAPÍTULO II
## IMPLEMENTACIÓN

En este capítulo se encuentra la parte medular del presente trabajo.
Empezamos con una breve descripción de los elementos que hemos utilizado.

### 2.1. Descripción del hardware y software utilizado

El hardware corresponde a un equipo facilitado por la Facultad de Ciencias Físico Matemáticas de la Benemérita Universidad Autónoma de Puebla. El Software, particularmente en cuanto a Sistema Operativo y Compilador inicial, también fue proporcionado por tal institución.

#### 2.1.1. Hardware utilizado

Para el desarrollo del presente trabajo se utilizó una computadora paralela que consta de las siguientes características:

- 1 motherboard para dos procesadores .
- 2 procesadores Intel Pentium II a 366 MHZ.
- 2 discos duros SCSI de 8 y 10 Gigabytes (marca Quantum).
- Lector de CD's (Pionner).
- Tarjeta de sonido.
- Tarjeta digitalizadora de imágenes.

La arquitectura de esta computadora es del tipo multiprocesamiento simétrico, tiene un esquema como el que se muestra en la figura 1.7. Esta arquitectura apareció en los supercomputadores CRAY X-MP y en sistemas similares en 1983 [23].

Las computadoras que entran en esta arquitectura permiten obtener un alto rendimiento al permitir que sus CPU's puedan realizar procesos individuales de manera concurrente. Los sistemas operativos como UNIX, varias versiones de Linux y Windows NT soportan SMP (Symmetric multiprocessing) [22].

Cabe mencionar que existen otras tecnologías similares que pudieron ser usadas para el desarrollo del presente proyecto, como lo son los transputers que hace uso del lenguaje de programación OCCAM y los clústers.

Los clústers de computadoras son otra tecnología que está teniendo auge, siendo ésta una nueva opción para las necesidades de cómputo intensivo y de desarrollo debido a que proveen una gran capacidad de procesamiento a un costo relativamente bajo (en comparación con las supercomputadoras).

Existen empresas como IBM, Intel, Qwest, Myricom, Oracle y Sun que apoyan a la investigación de esta tecnología. Actualmente colaboran para el proyecto Distributed Terascale Facility (DFT), un clúster capaz de realizar 11.6 Teraflops y almacenar 450 Terabytes en datos.

En la Universidad Nacional Autónoma de México (UNAM) se está desarrollando un clúster llamado Zapacluster a través del Departamento de Supercómputo, DGSCA-UNAM [8]

### 2.1.2. Software utilizado

Para la implementación del programa creado en el presente trabajo, fueron investigados tres compiladores distintos, el Portland Group Compiler, el Intel® C++ Compiler for Linux 6.0, y una implementación de MPI llamada MPICH [24].

El Portland Group Compiler y el Intel® C++ Compiler for Linux 6.0 fueron desechados porque el primero a pesar de tener características muy parecidas al MPI, no permite mucha escalabilidad a sistemas con varios procesadores y el problema a resolver tiene miras a una implementación con un número considerable de procesadores. EL Intel® C++ Compiler for Linux 6.0 fue desechado porque sus características se adaptan más a problemas del paradigma SIMD, mientras que nuestro problema se adapta más al modelo MIMD. Para este último se ha utilizado la biblioteca MPI, misma que a continuación se describe.

#### 2.1.2.1. MPI

MPI es la primera biblioteca de paso de mensajes estándar que ofrece portabilidad, estandarización y una implementación eficiente de la ejecución paralela, está formada por una colección de rutinas que facilitan la comunicación entre procesadores en programas paralelos [7].

La estandarización asegura que las llamadas hechas en algún equipo se comportan de igual manera en otras máquinas, independientemente de la implementación usada. La portabilidad permite la implementación de programas en diferentes plataformas como Linux, Solaris, UNIX, Windows NT, entre otras.

MPI nació en el taller de Estándares para el Paso de Mensajes [26] en un ambiente de memoria distribuida patrocinado por el Centro para la Investigación de Computación Paralela en abril de 1992; en ese mismo año surgió la primera y segunda versión de MPI. Estas versiones no fueron aceptadas porque no contenían un manejo de grupos de procesos y tampoco manejaban contextos de comunicación seguros.

MPI ha sido fuertemente influenciado por las siguientes entidades:
Centro de Investigaciones Watson de IBM, Intel's NX/2, Express, nCUBES's Vértex p4 y PARMACS.

Otras contribuciones importantes han sido de Zipcode, Chimp, PVM, PICL. y Chameleon, ésta última tiene una versión libre llamada MPICH [24] y es la versión de MPI usada en el presente trabajo.

La estructura general de un programa MPI es la siguiente:

1) Inicialización de la comunicación
2) Comunicar para compartir datos entre procesos cada vez que sea necesario.
3) Finalizar el ambiente paralelo.

La implementación de MPI usada es para el lenguaje de progamación C, aunque existen también implementaciones para Fortran, éstas no se usaron debido a los antecedentes que se tienen en el lenguaje.

En la siguiente sección se expondrán los conceptos de Topología Digital necesarios para la comprensión de este trabajo.

## 2.2. Conexidad en puntos lattice

A las imágenes originales mostradas en las figuras 1.2. y 3.20 se les aplicaron bs filtros descritos en la sección 2.2.1**,** hasta obtener las correspondiente imágenes binarias $M_{nxm}$ (figs. 3.6. y 3.20, respectivamente). Estas últimas se representan por medio de matrices rectangulares cuyas entradas son solamente 0's o 1's. En la figura 2.1 a). se muestra una parte de la matriz binaria de la imagen mostrada en figura 3.20.

Siguiendo los métodos de topología digital, a cada entrada de esta imagen binaria $M(i,j)$ se le asocia un punto de coordenadas enteras en el plano (puntos lattice). Si $M(i, j) = 0$ entonces el punto asociado será llamado *blanco*; mientras que si $M(i, j) = 1$, el punto será llamado negro. Con tal regla de asociación se obtiene una imagen binaria formada por puntos *lattice* (fig. 2.1b).

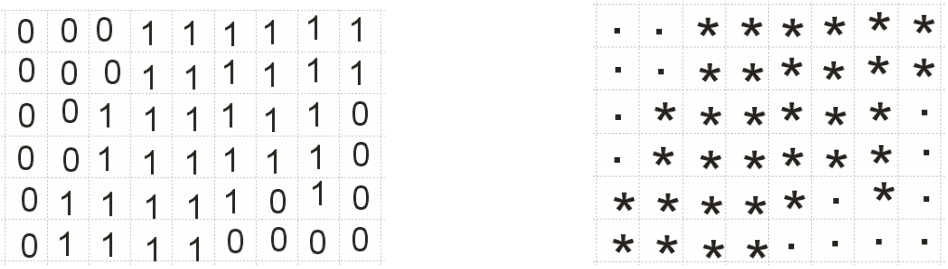



Figura 2.1. (a) Imagen binaria como matriz de 0's o 1's,
(b) Imagen binaria en puntos lattice.

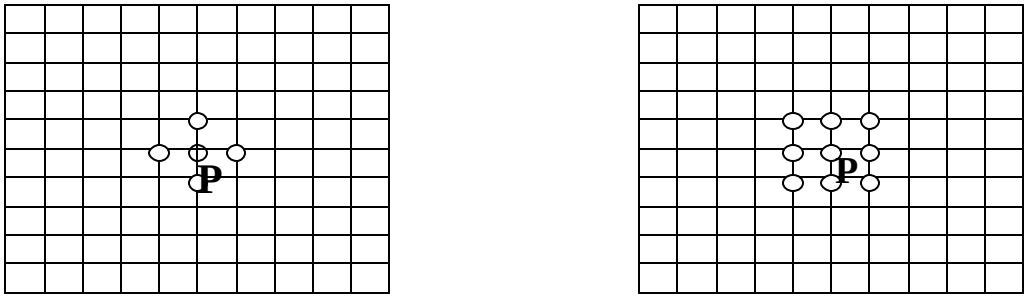
Para fijar ideas relacionadas con el cálculo del número de componentes conexas realizado en el presente trabajo, enseguida se dan los conceptos necesarios con el fin de precisar el tipo de conexidad asumido.

### 2.2.1. Conexidad en conjuntos de puntos lattice [19]

**Definición 2.2.1.1** Los elementos $(x, y) \in \mathbb{Z}^2$ corresponden a los puntos con coordenadas enteras en el plano euclideano y se llaman *puntos lattice.*

**Definición 2.2.1.2** Dos *puntos lattice* $p, q \in \mathbb{Z}^2$ se dicen *8-adyacentes* si son distintos, y cada coordenada de uno de ellos difiere en a lo más uno, de la respectiva coordenada del otro, es decir, para $p = (x_1, x_2)$ y $q = (y_1, y_2)$, se tiene $|x_i - y_i| = 0$ ó $|x_i - y_i| = 1$, i=1,2.

Dos puntos lattice $p, q \in \mathbb{Z}^2$ se llaman *4-adyacentes* si ellos son *8-adyacentes* y difieren en a lo más una de sus coordenadas. Para *n* =4 u 8, un *n-vecino* del punto *p*, es un punto *q* que es *n-adyacente* a *p* (fig. 2.2).




**a)** **b)**

Figura 2.2. a) *p* y sus 4 vecinos; b) *p* y sus 8 vecinos.

**Definición 2.2.1.3** Un conjunto *S* de puntos lattice se dice *m- adyacente a un conjunto T de puntos lattice* si algún punto en *S* es *m-adyacente* a algún punto de *T* (fig. 2.3).

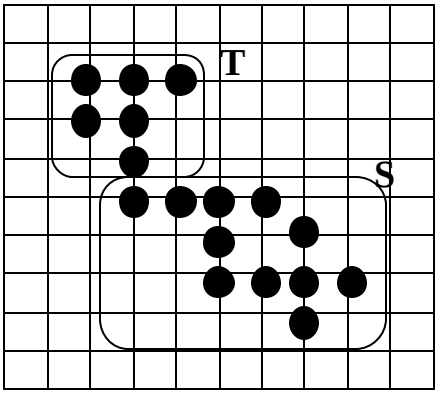



Figura 2.3. El conjunto S es a la vez 4 y 8-adyacente al conjunto T

En particular, un punto *p* es *adyacente* a un conjunto de puntos *S* si *p* es adyacente a algún punto de *S* (fig. 2.4)

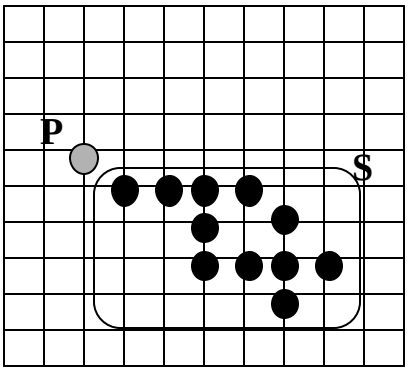



Figura 2.4. *p* es *8-adyacente* al conjunto S, pero no es *4-adyacente* a S

**Definición 2.2.1.4** Un conjunto de puntos lattice *S*, $S \subseteq \mathbb{Z}^2$, es *m-conexo* si *S* no puede particionarse en dos subconjuntos *A* y *B no m-adyacentes* el uno del otro (fig. 2.5)

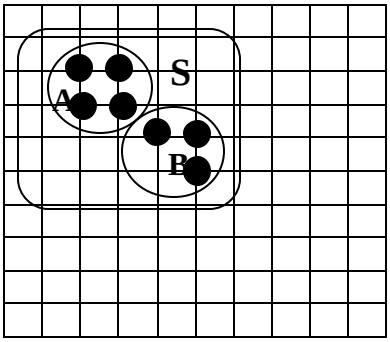



Figura 2.5. S es *8-conexo*, pero no *4-conexo*.

**Definición 2.2.1.5** Una *m-componente* de un conjunto de puntos lattice *S*, es un *subconjunto* de *S, no vacío m-conexo,* que no es *m-adyacente* a algún otro punto de *S* (fig. 2.6)

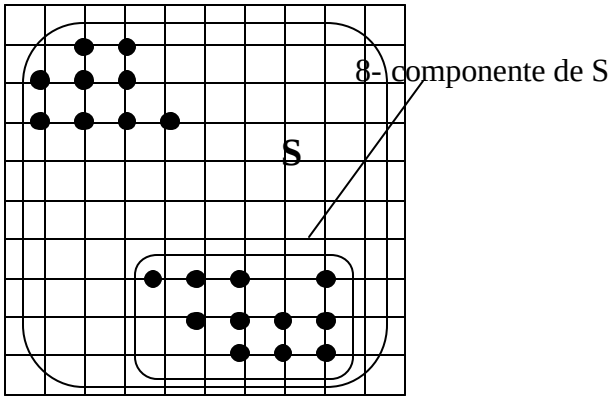



Figura 2.6. Una 8-componente de S

Sin duda una de las grandes ideas de la topología digital, es la de considerar un tipo de adyacencia para el conjunto de puntos negros, y otro para el conjunto de puntos blancos (Duda, Hart, Munson [19] para evitar paradojas.

En el presente trabajo se ha asumido *8-adyacencia* para puntos negros y *4-adyacencia* para puntos blancos.

### 2.2.2. Imágenes digitales Binarias

Ahora podemos dar la definición principal.

**Definición 2.2.2.1** *Una imagen digital bidimensional* $P$ es una tetrada $P=\left(\mathbb{Z}^2, m, n, B\right)$ *donde* $m,n$ *son tales que* $(m,n)\in\{(4,8),(8,4)\}$ *y* $B\subseteq\mathbb{Z}^2$. Los puntos de $B$ se llaman puntos negros, mientras que los puntos de $\mathbb{Z}^2-B$ se llaman puntos blancos de la imagen y constituyen el fondo de la misma. Cuando B es un conjunto finito, la imagen $P$ se denomina finita. En lugar de $\mathbb{Z}^2$, vamos a considerar un conjunto finito $S\subset\mathbb{Z}^2$, el cual contiene a todas las imágenes. En la práctica, $S$ puede considerarse como la colección de todos los puntos de la pantalla de un monitor de tal manera que, nos referimos a la imagen $P=(S, m, n, B)$.

**Definición 2.2.2.2** Dos puntos negros en una imagen digital $P=(S,m,n,B)$, se denominan *adyacentes* si son *m-adyacentes*, y dos puntos blancos o bien dos puntos blancos y uno negro se dicen *adyacentes* si ellos son *n-adyacentes*. Una imagen digital *(S, m, n, B)* también se llamará una *(m , n)-imagen digital* (fig.2.7).

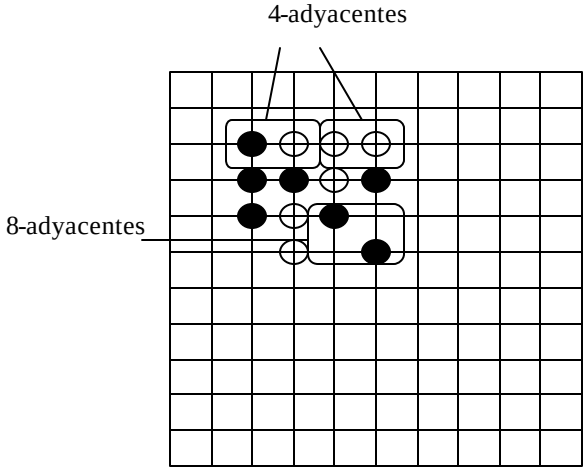



Figura 2.7. Adyacencias entre puntos

**Definición 2.2.2.3** Un conjunto S de puntos negros (blancos) en una *(m ,n)*-imagen digital es *conexo* si *S es m-conexo* (respectivamente *n-conexo*) (fig. 2.8).

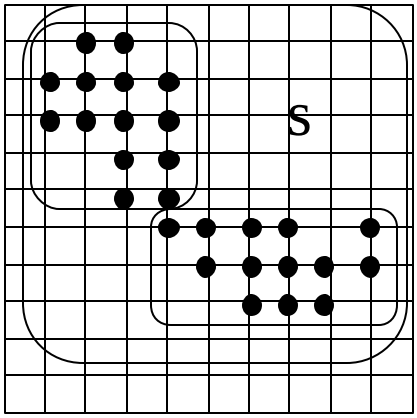


Figura 2.8. El conjunto S de puntos negros es conexo

**Definición 2.2.2.4** En una *(m, n)*-imagen digital una componente de un conjunto S de puntos negros es una *m-componente*, mientras que una componente de un conjunto de puntos blancos es una *n-componente* de este conjunto.

**Definición 2.2.2.5** Una componente del conjunto de todos los puntos negros de una imagen digital se llama *componente negra*, y una componente de puntos blancos se llama *componente blanca*.

Esta definición es de particular importancia para el presente trabajo, pues como parte del mismo, se localizan componentes negras y se les calcula sus centroides, entre otras propiedades. En el caso de los ronchigramas se hallan también las 4-componentes blancas y se esquelatizan (figs. 3.21, 3.23).

A continuación, se describe el algoritmo adoptado como base para hallar los centroides de las componentes negras, y realizar el conteo de las mismas.

### 2.2.3. Algoritmo de Etiquetación de Componentes Conexas [11]

Este algoritmo sirve para la etiquetación de 8-componentes, la entrada para este algoritmo es una matriz que contiene 1's o 0's. Los 1's son los 8-componentes que serán etiquetados con valores mayores que 1, y los 0's son el fondo de la imagen. Al finalizar el proceso la matriz contendrá 0's y valores mayores que 1, donde cada valor distinto asignado representa una 8-componente diferente.

Hágase un barrido de la imagen pixel a pixel de izquierda a derecha y de arriba hacia abajo. Para este algoritmo refierase como etiqueta al valor mayor que 1 asignado a algún píxel *p* en caso contrario cuando el píxel tiene valor 0 significa que no ha sido etiquetado

Sea *p* el pixel en cada paso del proceso de barrido y sean *r, s, t* y *u* los vecinos superior, superior derecho, superior izquierdo e izquierdo respectivamente.

**Paso 1.** Buscar un pixel p con valor 1

**Paso 2.** Se examinan *r, s, t* y *u*. Si al menos uno de ellos tiene etiqueta:

Si la etiqueta de todos los vecinos de *p* es mayor que 1, asignar a *p* dicha etiqueta.

Si existe más de una etiqueta diferente para todos los vecinos de *p*, se elige alguna etiqueta de dichos vecinos, y se procede a reemplazar las coincidencias de dichas etiquetas por la seleccionada en toda la matriz.

**Paso 3.** Dar una segunda pasada y reemplazar las etiquetas equivalentes

Fin del algoritmo.

Esta es la versión secuencial de tal algoritmo. En la subsección 2.2.1.7 se da la versión paralela, resultado del presente trabajo de tesis.

### 2.2.4. Procesamiento de ronchigramas

Como ya se ha dicho al final de la sección 1.1 del capítulo I, el objetivo de procesar ronchigramas es obtener "líneas medias" de las franjas blancas o negras, y posteriormente se les debe asignar una etiqueta de identificación llamada *orden de interferencia*. Esto último se explica en la sección 2.3.2.

Para obtener las "líneas medias", a la imagen binaria correspondiente se le aplica un algoritmo de esqueletización.

El concepto clave de tal algoritmo es el *índice de un píxel* (Shchepin [21], Rodríguez [18]).

Con el fin de precisar tal concepto, supóngase que ya se tiene una imagen binaria en puntos lattice (fig. 2.1 b).

A cada punto lattice le asignamos un complejo cúbico bidimensional (o cuadrado) formado por

- Cuatro 0-*celdas* (o vértices)
- Cuatro *1-celdas* (o aristas)
- Una *2-celda* (el cuadrado en sí)

Este complejo cúbico bidimensional puede interpretarse como un cuadrado unitario cerrado y se llama *píxel* (fig. 2.9) cuyos lados son paralelos a los ejes coordenados, y centrado en dicho punto lattice.

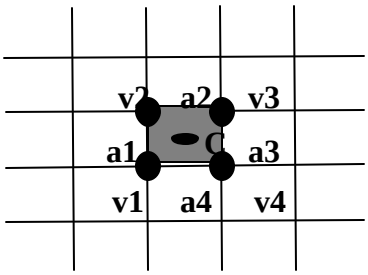



Figura 2.9. *p* con sus vértices y aristas

Un pixel se llamará negro si su correspondiente punto lattice es negro, y se llamará blanco si así lo es su respectivo punto lattice. De tal manera, la imagen binaria en puntos lattice ahora adquiere una nueva representación llamada *realización geométrica* (fig. 2.10).

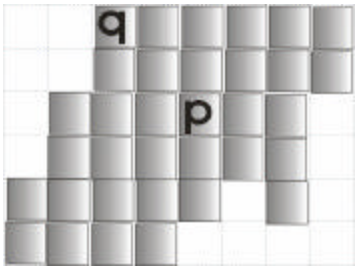



Figura 2.10. Realización geométrica de la imagen binaria bidimensional mostrada en la figura 2.1.

Teniendo la imagen así representada, puede aplicársele ya la herramienta de Topología Algebraica, disciplina matemática que estudia ciertos aspectos de la forma de los objetos.

**Definición 2.2.4.1.** Sea *R= (S, m, n, B)* una *(m ,n)*- imagen digital binaria bidimensional. Un subconjunto *C* de *S* es una *m-componente* de *R* si su correspondiente conjunto de puntos lattice es una *m- componente.* Un subconjunto *H* de *S* es una *n-componente* si su correspondiente conjunto de puntos lattice es una *n-componente.*

Ahora sí, se está en condiciones de definir el concepto de índice de un píxel.

**Definición 2.2.4.2.** *Indice del pixel p* con relación al conjunto **B** de píxeles negros, se llama a la diferencia entre el número de vértices de *p* comunes a **B**, y el número de aristas de *p* comunes a **B.**

Por ejemplo, en la figura 2.10 el píxel ***p*** tiene ***índice cero***, mientras que el píxel ***q*** tiene ***índice uno***. Nótese que la eliminación del píxel *p* aumentaría en uno la cantidad de agujeros (o 4-componentes blancas) de la imagen; mientras que la eliminación del píxel *q* no causaría modificación alguna en las propiedades homotópicas de la imagen (número de 8-componentes y número de 4-componentes, entre otras). Este hecho fue demostrado en Shchepin [21].

### 2.2.5. Algoritmo de adelgazamiento (obtención del esqueleto)

La entrada de este algoritmo es una matriz binaria (con 0's o 1's). La salida es otra matriz binaria donde las 8-componentes tienen grosor de un píxel.

**Paso 1.** Hagamos la bandera de "adelgazamiento" *falso*.

**Paso 2.** Fijemos la dirección en torno de un plano discreto Y-conexo en correspondencia con el esquema de prioridades elegido. Si las direcciones se terminan, traslademos al paso 6, en caso contrario ir al paso 3.

**Paso 3.** Busquemos un pixel negro *p* (marca 1) que no tenga vecino en la dirección dada. Si no existen tales, entonces ir al paso 5; si los hay, entonces al siguiente paso.

**Paso 4.** Si el índice del pixel *p* no es 1 o su valencia es igual a 1, entonces se marcará con 2, en caso contrario con 3. Por esto para el cálculo del índice se toman todos los pixeles con marca positiva, ir al paso 3.

**Paso 5.** Anular las marcas de todos los pixeles de marca 3. Si en este paso aconteció aunque sea una anulación, se establecerá la bandera "adelgazamiento" en *verdadero*. Ir al paso 2.

**Paso 6.** Si la bandera "adelgazamiento" es *verdadera*, ir al paso 1. Si no es así se termina el algoritmo.

Fin del algoritmo.

***Nota.*** El orden de recorrido para la matriz binaria, según las prioridaes escogidas es el mostrado en la figura 2.11.

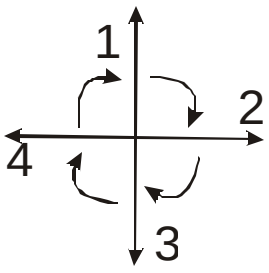


Figura 2.11. Orden de prioridad de dirección.

## 2.3. Descripción del programa

En esta sección se describen los módulos más importantes del software desarrollado para el presente trabajo.

### 2.3.1. Preprocesamiento y procesamiento de las imágenes de coloides

Debido a que en la implementación se contó con un equipo con dos procesadores, se dividió la matriz en dos submatrices del mismo tamaño. Por ello todas las funciones presentan los valores **coli, colf,** que sirven para indicar desde qué columna se va a trabajar hasta cuál. Cada procesador tiene una copia del código.

Una vez que se ha dividido la matriz original que representa a la imagen digital original (tamaño 640x480, en formato PCX), se le aplican diversos filtros hasta llevarla a una imagen binaria.

Enseguida se describen los filtros utilizados.

#### 2.3.1.1. Inversión de la imagen

La primera transformación realizada a la imagen fue la inversión del orden de blanco a negro. La imagen tratada está en modo de escala de grises, cada píxel tiene valores entre 0 y 255. Entonces el negativo de la imagen se obtiene restando al valor 255 el valor del píxel en ese mismo punto [11]. El siguiente fragmento de código muestra la implementación de dicho proceso:

```
void invertir(int coli, int colf)
{       int i, j;
        for (i=1 ; i<alto+2; i++)
                for(j=coli; j<=colf; j++)
                {       m[i][j]=255-m[i][j];
                }

}
```

En esta parte, la matriz que contiene los datos de la imagen es ***m***, esta matriz tiene dimensión 2 y es de tamaño *ancho+2* por *alto+2*. Donde *ancho* y *alto* son valores correspondientes al ancho y alto de la imagen digital original.

#### 2.3.1.2. Cambio del contraste

Consiste en incrementar el rango dinámico de los niveles de gris de la imagen procesada. Esta transformación permite de alguna manera distinguir más claramente a un objeto de otro [11].

Una forma de hacer el cambio de contraste por niveles es seleccionar un rango entre 0 y 255, sean *superior* e *inferior* los valores que determinan dicho rango, entonces se hace una proporción haciendo que *inferior* sea igual a 0 y *superior* igual a 255, entonces los valores que quedan en dicho rango se distribuyen uniformemente en 256 partes iguales o pueden aproximarse con una función que permita distribuirlos sobre una curva que permita que los tonos más oscuros lo sean más y los más claros se aclaren más. Los valores de

entrada que no quedan dentro de este rango son convertidos entonces a un valor determinado, por ejemplo blanco.

El código de este proceso se muestra a continuación.

```
void nivel(int coli, int colf, int inferior, int superior)
{       float maximo=3.996088;
        float salida;
        int i, j;

        for (i=1 ; i<alto+1; i++)
                for(j=coli; j<=colf; j++)
                {       if(m[i][j]<inferior || m[i][j]>superior)
                        {       m[i][j]=255;     //Blancos los que no están en el rango
                        }
                        else
                        {       salida=m[i][j]-inferior;
                                if((superior-inferior)>0)
                                {       salida=maximo*salida/(superior-inferior);
                                        m[i][j]=(int)(salida*salida*salida*salida);
                                }
                        }
                }
}
```

Otra forma de modificar el contraste es cambiar los valores de entrada usando una función como la que se muestra en la figura 2.12, con el mismo objetivo de la curva usada para el cambio de contraste por niveles, su implementación se muestra en el siguiente fragmento de código.

```
void  curva(int coli, int colf )
{       int i, j;
        for (i=1 ; i<alto+1; i++)
                for(j=coli; j<=colf; j++)
                {       if(m[i][j]<107)
                        {       m[i][j]=(28*m[i][j])/107;
                        }
                        else if(m[i][j]<148)
                        {       m[i][j]=(203*(m[i][j]-107))/41+28;
                        }
                        else
                        {       m[i][j]=(24*(m[i][j]-255))/107+255;
                        }
                }
        }
}
```

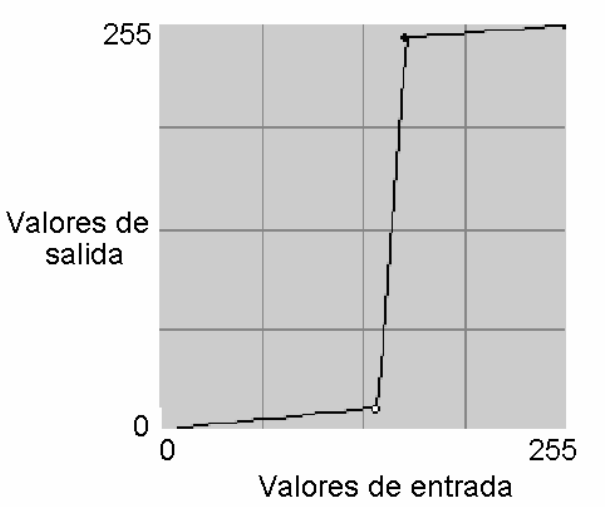


Figura 2.12. Cambio de contraste.

En esta parte se busca hacer que los valores más claros lo sean más, y los obscuros se vuelvan también más obscuros. La función de utilizada se muestra en la figura 2.12.

### 2.3.1.3. Filtro suavizante pasa bajo

Este filtro se usa para hacer que la imagen parezca algo borrosa y también reducir el ruido. Esto elimina pequeños detalles y rellena espacios vacíos.
Este filtro consiste en realizar la suma de los valores de los píxeles vecinos al píxel que se está analizando y dividirla por el número de partes analizadas (incluso el píxel analizado) [11]. Para la aplicación del filtro nosotros hemos utilizado máscaras de 5x5 .

1/25 x

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Figura 2.13. Máscara para la implementación del filtro suavizante.

Código de la aplicación de la máscara de 5x5 a la matriz ***m:***

```
void masc5x5(int coli, int colf, int divisor, int desplazamiento)
{       int i, j, k, l, mx, my, sum;
        for (i=2 ; i<alto; i++)
                for(j=coli; j<=colf; j++)
                {       sum=0;
                        for(k= i-2, my=0; k<= i+2; k++, my++)
                                for(l= j-2, mx=0; l<= j+2; l++, mx++)
                                        sum+=m[k][l]*masc[my][mx];
                        sum=sum/divisor;
                        n[i][j]=sum+desplazamiento;
                }
```

```
        copianm(coli, colf);
}
```

Debido a que en este programa se trabaja con dos matrices (***m*** y ***n***), donde ***m*** contiene los valores originales y ***n*** contiene el resultado de la aplicación del filtro, es necesario regresar a ***m*** los nuevos valores, para ello se usa la función **copianm**.

***void copianm(int coli, int colf)***

```
{       //copia la matriz n a la m
        int i, j;
        for (i=1 ; i<alto+2; i++)
                for(j=coli; j<=colf; j++)
                        m[i][j]=n[i][j];
}
```

### 2.3.1.4. Laplaciano

Este filtro se usa para detectar bordes [11], es aplicado solamente en las imágenes de ronchigramas.

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

Figura 2.14. Máscara usada para calcular el filtro laplaciano

### 2.3.1.5. Equalización

Este algoritmo se usa para realzar el contraste en las imágenes de ronchigramas.

Durante la aplicación de los anteriores filtros, no es necesario tener comunicación entre los procesadores para compartir algún tipo de datos.

En el caso del algoritmo de equalización [2] no ocurre esto. En esta parte se tienen dos funciones específicas, equalizaEsclavo y equalizaMaestro.
Se tienen estos dos módulos porque cada procesador obtiene datos que necesita el otro, el procesador esclavo envía los datos que necesita el maestro, este obtiene los nuevos niveles y finalmente el maestro después de procesarlos envía de regreso los nuevos niveles que se obtuvieron que deberán aplicar ambos para la actualización de los niveles de la imagen.

***void equalizaMaestro(int coli, int colf, int esclavo)***

```
{       int vecEQ[256];
        int vecEQ2[256];
        int vecC[256];
        int vecN[256];
        int i, j, tmp;
        MPI_Status estado;
```

```
	//se inicializan todos los vectores a 0
	for(i=0; i<256; i++)
	{	vecEQ[i]=0;
		vecC[i]=0;
		vecN[i]=0;
	}

	//Se buscan cuántos niveles hay con valor 1, 2, ...255
	for (i=1 ; i<alto+1; i++)
	{	for(j=coli; j<=colf; j++)
		{	if(m[i][j]>=0 && m[i][j]<256)
				vecEQ[m[i][j]]++;
			else
				vecEQ[255]++;
		}
	}

	//Se recibe el vector VecEQ del otro procesador en VEQ2.
	MPI_Recv(vecEQ2, 256, MPI_INT, esclavo, 12, MPI_COMM_WORLD, &estado);

	for(i=0; i<256; i++)
		vecEQ[i]+=vecEQ2[i];

	//acumulado
	vecC[0]=vecEQ[0];
	for(i=1; i<256; i++)
	{	vecC[i]=vecC[i-1]+vecEQ[i];
	}

	//obtencion de los nuevos niveles de gris
	for(i=0; i<256; i++)
	{	tmp=(int)((256*vecC[i])/(alto*ancho))-1;
		if(tmp>0)
			vecN[i]=tmp;
		else
			vecN[i]=0;
	}

	//Se envían los nuevos niveles de gris
	MPI_Send(vecN, 256, MPI_INT, esclavo, 13, MPI_COMM_WORLD);

	//Se aplica la equalización
	for (i=1 ; i<alto+1; i++)
	{	for(j=coli; j<=colf; j++)
		{	if(m[i][j]>=0 && m[i][j]<256)
				m[i][j]=vecN[m[i][j]];
			else
				m[i][j]=255;
		}
	}
}
```

***void equalizaEsclavo(int coli, int colf, int maestro)***

```
{       int vecEQ[256];
        int vecN[256];
        int i, j;
        MPI_Status estado;

        //inicializo todos los vectores a 0
        for(i=0; i<256; i++)
        {       vecEQ[i]=0;
        }

        //Se buscan cuántos niveles hay con valor 1, 2, ...255
        for (i=1 ; i<alto+1; i++)
        {       for(j=coli; j<=colf; j++)
                {       if(m[i][j]>=0 && m[i][j]<256)
                                vecEQ[m[i][j]]++;
                        else
                                vecEQ[255]++;
                }
        }

        //Se envía el vector vecEQ
        MPI_Send(vecEQ, 256, MPI_INT, maestro, 12, MPI_COMM_WORLD);
        //Se recibe la nueva lista de niveles de gris
        MPI_Recv(vecN,  256, MPI_INT, maestro, 13, MPI_COMM_WORLD, &estado);

        //Se reemplazan los niveles de gris por los nuevos de vecN
        for (i=1 ; i<alto+1; i++)
        {       for(j=coli; j<=colf; j++)
                {       if(m[i][j]>=0 && m[i][j]<256)
                                m[i][j]=vecN[m[i][j]];
                        else
                                m[i][j]=255;
                }
        }
}
```

### 2.3.1.6. Umbralización

Consiste en obtener una matriz con valores 1's o 0's a partir de valores entre 0 y 255. La idea consiste en que si el valor de entrada sobrepasa cierto umbral se le aplica un valor, en caso contrario se le aplica el otro [11]. Para el caso de este programa, si el valor de entrada es mayor que umbral, el nuevo valor es 0, en caso contrario es 1.

***void binarizador(int coli, int colf, int umbral)***

```
{       int i, j;
        for(i=1; i<alto+1; i++)
                for(j=1; j<ancho+1; j++)
                {       if(m[i][j]>umbral)
                                m[i][j]=0;
```

```
                else
                        m[i][j]=1;
        }
}
```

Para poder identificar a los hoyos y distinguirlos del fondo se insertaron columnas y filas alrededor de la imagen original para poder detectar como parte del fondo a imágenes que pudieran tener un hoyo en algún extremo de la imagen.

En la siguiente sección se tienen variables que permiten contener características útiles en el algoritmo:

| | |
|---|---|
| Hcont | Contador del número de hoyos |
| H | Vector con identificadores de los hoyos |
| Hx | Vector para centroide coord x |
| Hy | Vector para centroide coord y |
| Hs | Vector para el área |
| Hsup | Vector que tiene la posición en y mas alta del hoyo dentro de la imagen |

A continuación se muestra el código para la detección de los hoyos.

Note que aquí se usa la 4-conexidad para detectar los hoyos.

***void eliminaHoyos(int coli, int colf)***

```
{       int i, j;
        HCont=0;
        for(i=coli; i<colf; i++)
                m[0][i]=-1;     //parte superior de la matriz =-1
        for(i=0; i<alto+1; i++)
                m[i][0]=-1;     //parte izquierda de la matriz= -1
        if(coli==1)
                for(i=0; i<alto+1; i++)
                        m[i][colf+1]=-1;

        //Buscando hoyos
        for(i=1; i<alto+1; i++)
        {       for(j=coli; j<colf; j++)
                {       if(m[i][j]==0)
                        {       if(m[i-1][j]==-1)       //vecino superior
                                {       if(m[i][j-1]==1 || m[i][j-1]==-1 )      //vecino izquierdo es negro
                                        {       m[i][j]=-1;
                                        }
                                        else if(m[i][j-1]<-1)//el bit a la izquierda es de otro grupo de hoyos
                                        {       m[i][j]=-1;
                                                cambiaHoyo(m[i][j-1], -1, i, coli, colf);

                                        }
                                }
                                else if(m[i][j-1]==-1)  //vecino izquierdo
```

```
                    {       if(m[i-1][j]==1 || m[i-1][j]==-1  )
                            {       m[i][j]=-1;
                            }
                            else if(m[i-1][j]<-1) //el bit superior es de otro grupo de
hoyos
                            {       m[i][j]=-1;
                                    cambiaHoyo(m[i-1][j], -1, i, coli, colf);
                            }
                    }

                    else if(m[i-1][j]<-1)       //sup
                    {       if(m[i][j-1]==1)//izq es negro
                                    m[i][j]=m[i-1][j];
                            else if(m[i][j-1]<-1)//izq de otro grupo
                            {       if(m[i-1][j]!=m[i][j-1])
                                    {       m[i][j]=m[i-1][j];
                                            cambiaHoyo(m[i][j-1], m[i][j], i, coli, colf);
                                    }
                                    else
                                    {       m[i][j]=m[i-1][j];
                                    }
                            }
                            else
                                    m[i][j]=m[i-1][j];

                    }
                    else if(m[i][j-1]<-1)       //izquierdo
                    {       if(m[i-1][j]==1)
                                    m[i][j]=m[i][j-1];

                            else if(m[i-1][j]<-1)
                            {       if(m[i-1][j]!=m[i][j-1])
                                    {       m[i][j]=m[i][j-1];
                                            cambiaHoyo(m[i-1][j], m[i][j], i, coli, colf);
                                    }
                                    else
                                    {       m[i][j]=m[i][j-1];
                                    }
                            }
                            else
                                    m[i][j]=m[i][j-1];

                    }
                    if(m[i][j]==0) //no ha sido cambiado
                    {       m[i][j]=agregaHoyo(i); //se obtiene un nuevo identificador
para el hoyo.
                    }
            }
    }
}
```

```
}
```

En esta función se cambian las coincidencias de *v1* por *v2*, empezando verticalmente desde la parte superior del grupo (Hsup)

```
 void cambiaHoyo(int v1, int v2, int y, int coli, int colf)
{       int i, j;
        int inicio=-v1;
        inicio=Hsup[inicio];
        for(i=inicio; i<=y; i++)
                for(j=coli;j<colf; j++)
                {       if(m[i][j]==v1)
                                m[i][j]=v2;
                }

        //Si la parte mas alta de v1 es mas alta que la de v2
        //la parte mas alta de v2 queda como mas alta ya que
        //v1 va a ser eliminada
        if(Hsup[-v1]<Hsup[-v2])
                Hsup[-v2]=Hsup[-v1];
        borraHoyo(v1);
}
```

Este módulo tiene la función de rellenar los hoyos y así obtener la nueva matriz binarizada con los objetos rellenos.

```
void nuevaMatrizHoyos(int coli, int colf)
{       //Regresa los datos a una matriz binaria
        for(i=0; i<alto+2; i++)
        {       for(j=coli; j<colf; j++)
                {       if(m[i][j]!=1)
                        {       if(m[i][j]<-1 && H[-m[i][j]]!=0)
                                        m[i][j]=1; //los hoyos se rellenan
                                else
                                        m[i][j]=0;
                        }
                }
        }
}
```

En este módulo se detectan los centroides y el área de los hoyos (cuantos píxeles forman el objeto).

```
void caracteristicasHoyos(int coli, int colf)
{       int i, j;
        for (i=1 ; i<alto+1; i++)
                for(j=coli; j<colf; j++)
                {       if(m[i][j]<0)     //cuenta los identificadores menores que cero
                        {       Hx[-m[i][j]]+=j;
```

```
                    Hy[-m[i][j]]+=i;
                    Hs[-m[i][j]]++;
                }
            }
}
```

### 2.3.1.7. Conteo de componentes conexas

En el siguiente módulo se etiquetan las 8-componentes conexas dentro de una imagen binarizada. Para este módulo se rellena el contorno de la imagen con valores igual a 0, es decir, el contorno es fondo.

***void cuentaObjetos(int coli, int colf)***

```
{       int i, j;
        VCont=0;
        //Se pone el contorno de la imagen como fondo
        for(i=0; i<alto+1; i++)
        {       m[i][coli-1]=0;  m[i][colf]=0;
        }
        for(i=coli; i<colf; i++)
                m[0][i]=0;


        for( i=1; i<alto+1; i++)
                for( j=coli; j<colf; j++)
                {       if(m[i][j]==1)   //punto negro
                        {       if(m[i][j-1]>0)  //punto izq
                                {       m[i][j]=m[i][j-1];
                                        actualizar(i, j, m[i][j], coli, colf);//y,x, val
                                }
                                if(m[i-1][j-1]>0)          //izq - sup
                                {       if(m[i][j]!=m[i-1][j-1])
                                        {       m[i][j]=m[i-1][j-1];
                                                actualizar(i,j, m[i][j], coli, colf);
                                        }
                                }
                                if(m[i-1][j]>0)            //sup
                                {       if(m[i][j]!=m[i-1][j])
                                        {       m[i][j]=m[i-1][j];
                                                actualizar(i,j, m[i][j], coli, colf);
                                        }
                                }
                                if(m[i-1][j+1]>0)          //sup - der
                                {       if(m[i-1][j+1]!=m[i][j])
                                        {       m[i][j]=m[i-1][j+1];
                                                actualizar(i,j, m[i][j], coli, colf);
                                        }
                                }
                                if(m[i][j]==1)
```

```
                                        {       m[i][j]=agregaAVector(i);//Se obtiene un nuevo
identificador
                                        }
                                }
                        }
        caracteristicas(coli, colf);
}
```

En esta función se cambian las coincidencias de *v1* por *v2* dentro de la matriz **m** empezando verticalmente desde la parte superior del grupo (*Vsup*)

***void cambia(int v1, int v2, int y, int coli, int colf)***

```
{
        for(int i=Vsup[v1]; i<=y; i++)
                for(int j=coli; j<colf; j++)
                {       if(m[i][j]==v1)
                                m[i][j]=v2;
                }

        if(Vsup[v1]<Vsup[v2])
                Vsup[v2]=Vsup[v1];

        borraEnVector(v1);
}
```

En este módulo se detecta si existen vecinos que tienen etiquetas diferentes, en caso de haberlos, se elimina una etiqueta y ambos grupos quedan con la misma etiqueta.

***void actualizar(int y, int x,int val, int coli, int colf)***

```
{       if(m[y][x-1]!=val && m[y][x-1]>1)      //izq
        {       cambia(m[y][x-1], val, y, coli, colf);
        }
        if(m[y-1][x-1]!=val && m[y-1][x-1]>1)//sup - izq
        {       cambia(m[y-1][x-1], val, y, coli, colf);
        }
        if(m[y-1][x]!=val && m[y-1][x]>1)      //sup
        {       cambia(m[y-1][x], val, y, coli, colf);
        }
        if(m[y-1][x+1]!=val && m[y-1][x+1]>1)          //sup - der
        {       cambia(m[y-1][x+1], val, y, coli, colf);
        }
}
```

En este módulo se obtiene el centroide y área de cada 8-componente (objetos), quedando en los vectores *Vx*, *Vy* y *Vs*.

***void caracteristicas(int coli, int colf)***

```
{       int i, j;
        for (i=1 ; i<alto+1; i++)
                for(j=coli; j<colf; j++)
                {       if(m[i][j]>1)
```

```
                    {       Vx[m[i][j]]+=j;
                            Vy[m[i][j]]+=i;
                            Vs[m[i][j]]++;
                    }
            }

}
```

Como se mencionó anteriormente, para procesar la matriz se dividió en dos partes iguales. Para poder identificar las componentes conexas se han identificado los casos mostrados en la imagen 2.15. La etiquetación de las 4-componentes (hoyos) da como resultado una matriz etiquetada como la que se muestra en la figura 2.16, la etiquetación de las 8-componentes (partículas) produce un resultado como el mostrado en la figura 2.17.

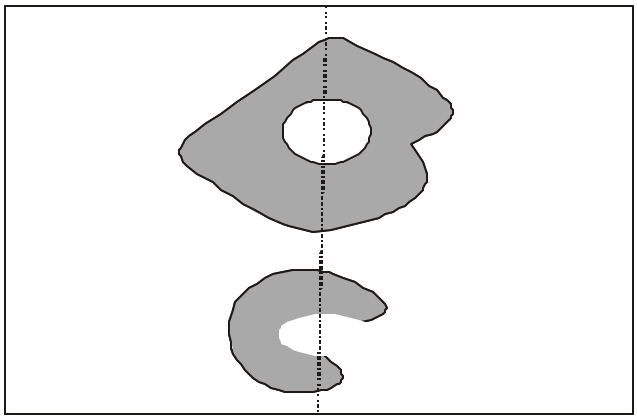


Figura 2.15. Imagen dividida

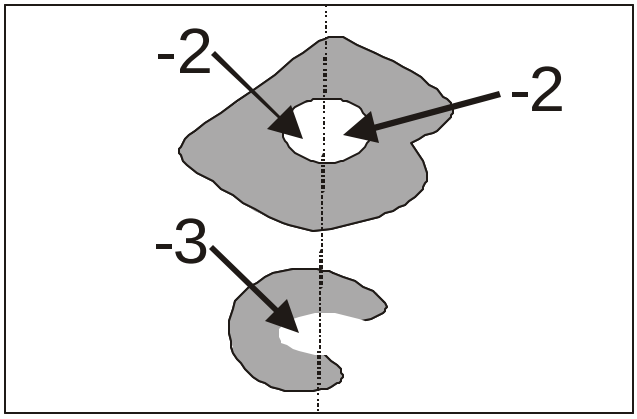



Figura 2.16. Hoyos que han sido etiquetados en cada parte de la imagen

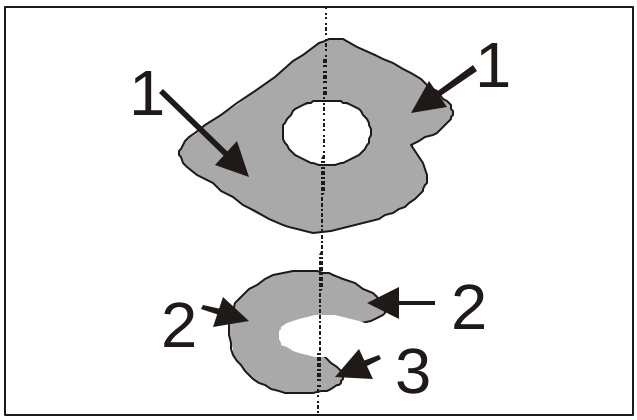


Figura 2.17. Objetos que han sido etiquetados
en cada parte de la imagen

De las figuras 2.16 y 2.17 se puede observar que no se tiene una relación entre las etiquetas de una parte de la matriz a la otra. Por lo tanto, para identificar que un hoyo (u objeto) es parte de otro en la misma imagen, se tiene que hacer un proceso que consiste en compartir las columnas adyacentes entre los procesadores e identificar las relaciones entre ellos.

En la figura 2.16 se tiene además que un hoyo (con etiqueta –3 en la figura) es en realidad fondo, entonces es necesario identificarlo como tal.

Para poder identificar cuándo dos objetos (u hoyos) adyacentes son el mismo para ambos procesadores, se siguen los puntos que se explican a continuación:

A partir de la figura 2.18, se puede observar que existen 5 grupos en cada lado de la imagen, pero como se menciona anteriormente las etiquetas de cada lado son independientes unas de las otras.

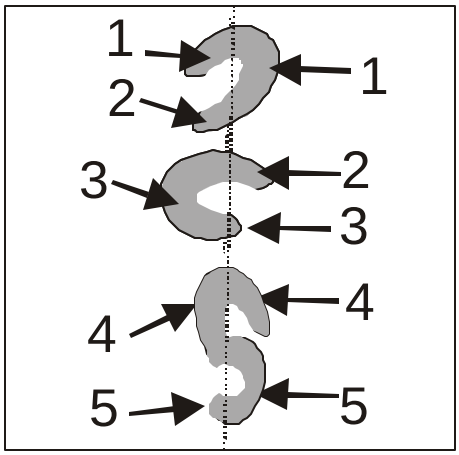


Figura 2.18. Objetos que han sido etiquetados en cada parte de la imagen

Supongamos que los vectores adyacentes se unen en una matriz ADY como se muestra a continuación:

| | |
|---|---|
| 1 | 1 |
| 0 | 1 |
| 2 | 0 |
| 0 | 0 |
| 3 | 2 |
| 0 | 0 |
| 3 | 2 |
| 0 | 0 |
| 4 | 4 |
| 4 | 0 |
| 0 | 5 |
| 5 | 0 |

Figura 2.19. Matriz ADY

A este vector se le aplica el algoritmo de etiquetación de 8-componentes, dando la matriz ETQ

| | |
|---|---|
| 1 | 1 |
| 0 | 1 |
| 1 | 0 |
| 0 | 0 |
| 2 | 2 |
| 0 | 0 |
| 3 | 3 |
| 0 | 0 |
| 4 | 4 |
| 4 | 0 |
| 0 | 4 |
| 4 | 0 |

Figura 2.20. Matriz ETQ después de hacer etiquetación.

La etiquetación sobre ADY se obtiene usando el siguiente código:

***int gruposIguales(int pos)***

```
{       int  gruposIguales(int pos)
{       int i, j;
        GCont=0;
        for(i=0; i<alto+1; i++)
        {       m[i][0]=0;
                if(m[i][1]!=0)
                        m[i][1]=1;          //matriz del procesador de la izquierda
                else
                        m[i][1]=0;

                if(m[i][2]!=0)
                        m[i][2]=1;          //matriz del procesador de la derecha.
                else
                        m[i][2]=0;
```

```
                m[i][3]=0;
                G[i]=0;
        }
                //alto+1
        for(i=1; i<alto+1; i++)
        {       for(j=1; j<3; j++)
                {       if(m[i][j]==1)
                        {       if(m[i][j-1]!=0)
                                {       m[i][j]=m[i][j-1];
                                }
                                else if(m[i-1][j-1]!=0)
                                {       m[i][j]=m[i-1][j-1];
                                }
                                else if(m[i-1][j]!=0)
                                {       m[i][j]=m[i-1][j];
                                }
                                else if(m[i-1][j+1]!=0)
                                {       m[i][j]=m[i-1][j+1];
                                }
                                if(m[i][j]==1)
                                        m[i][j]=agregaAGrupo(i);
                        }
                }
        }
        cuantosIguales(pos);
        return 1;
}
```

Nótese que ADY corresponde a las columnas 1 y 2 de la matriz **m**, los 0's en las columnas 0 y 3 sirven para aislar a estos objetos de cualquier basura que pudiera haber alrededor de ADY.

A cada nuevo 8-componente detectado se le asocia un conjunto de vectores inicializados en cero que son:

| | |
|---|---|
| Gcont | Contador del número de objetos |
| G | Vector con identificadores de los objetos |
| Gx | Vector para centroide coord x |
| Gy | Vector para centroide coord y |
| Gs | Vector para el área |
| Gsup | Vector que tiene la posición mas alta del objeto dentro de la imagen |

Usando el proceso anterior se obtiene la matriz ETQ modificada de la siguiente forma:

| | |
|---|---|
| 1 | 1 |
| 0 | 1 |
| 1 | 0 |
| 0 | 0 |
| 2 | 2 |
| 0 | 0 |
| 2 | 2 |
| 0 | 0 |
| 3 | 3 |
| 3 | 0 |
| 0 | 3 |
| 3 | 0 |

Figura 2.21. Matriz ETQ después de identificar
a los objetos que son iguales.

Para identificar los objetos que son adyacentes y por lo tanto son uno solo, se sigue el siguiente algoritmo:

1. Hacer para cada columna en ADY
   1.1. Recorrer la columna hasta encontrar algún elemento con valor diferente de 0, sea (x1, y1) la posición de ese elemento en ADY.
   1.2. Buscar otro elemento sobre la misma columna diferente del encontrado en el paso 1.1, si no existe otro elemento ir al paso 1.4 en caso contrario continuar con el paso 1.3
   1.3. Sea (x2, y2) la posición del siguiente elemento encontrado, si ETQ[x2,y2] ≠ ETQ[x1,y1], hacer
      1.3.1. Hacer tmp= ETQ[x1,y1]
      1.3.2. Buscar en ETQ todas las coincidencias de ETQ[x2,y2] y reemplazarlas por tmp.
   1.4. Fin

Lo anterior se consigue con el siguiente código:

```
int cuantosIguales(int pos)
{       int cuantos=0, i,j,k, tmp;
        for(i=1; i<alto+1; i++)
        {       if(m[i][4]!=0)
                {       for(j=i+1; j<alto+1; j++)
                        {       //if(m[i][1]==m[j][1])
                                if(m[i][4]==m[j][4])
                                {       if(m[j][1]!=m[i][1] && m[i][1]!=0 && m[j][1]!=0)
                                        {       //eliminar m[j][1] de la lista de grupos
                                                if(borraEnGrupo(m[j][1])==1)
                                                {       //reemplazar las ocurrencias de mj1 por mi1
                                                        tmp=m[j][1];
                                                        for(k=1; k<alto+1; k++)
                                                        {       if(m[k][1]==tmp)
                                                                        m[k][1]=m[i][1];
```

```
                                                        if(m[k][2]==tmp)
                                                                m[k][2]=m[i][1];
                                                }
                                        }
                                }
                        }
                }//fin if m[i][pos]

                if(m[i][5]!=0)
                {       for(j=i+1; j<alto+1; j++)
                        {       //(m[i][2]==m[j][2])
                                if(m[i][5]==m[j][5])
                                {       if(m[j][2]!=m[i][2] && m[i][2]!=0 && m[j][2]!=0)
                                        {       //eliminación de m[j][1] de la lista de grupos
                                                if(borraEnGrupo(m[j][2])==1)
                                                {       //reemplazar las ocurrencias de mj1 por mi1
                                                        tmp=m[j][2];
                                                        for(k=1; k<alto+1; k++)
                                                        {       if(m[k][1]==tmp)
                                                                        m[k][1]=m[i][2];

                                                                if(m[k][2]==tmp)
                                                                        m[k][2]=m[i][2];
                                                        }
                                                }
                                        }
                                }
                        }
                }//fin if m[i][pos]


        }
        agregaIzquierdaADerecha(pos);
        printf("\nValores de los grupos\n");
        for(i=1, j=1; i<alto+1; i++)
        {       if(G[i]!=0)
                {       Gx[i]/=Gs[i];
                        Gy[i]/=Gs[i];
                        printf("\nc.%d\t%d\t%d\t%d",j, Gx[i], Gy[i], Gs[i]);
                        j++;
                }
        }
        return GCont;
}
```

Usando la matriz ETQ y ADY se pueden obtener las características de los objetos a partir de los vectores V, Vx, Vy y Vs de cada procesador (o en el caso de los hoyos con H, Hx, Hy y Hs). Al sumar por ejemplo el área de cada objeto adyacente se puede obtener el área del objeto total que los contiene usando el siguiente proceso:

Para cada elemento $\neq 0$ en la matriz ETQ con posición (x,y) hacer

1. Actualizar Gs, Gx y Gy como

   Gs[ETQ [x][y]]=Gs[ETQ[x][y]]+Vs[ADY[x][y]]
   Gx[ETQ [x][y]]=Gx[ETQ[x][y]]+Vx[ADY[x][y]]
   Gy[ETQ [x][y]]=Gy[ETQ[x][y]]+Vy[ADY[x][y]]
2. Eliminar todas las coincidencias de ETQ[x][y] en ETQ

Al terminar todo este proceso en Gs, Gx y Gy se tienen las propiedades de los objetos que han quedado en el área de corte de la matriz.

**2.3.1.8. Filtrado de discriminación**

Después de tener en Gx, Gy y Gs los valores de las áreas, y centroide de cada figura, que está en el área de corte, también en Vx, Vy, y Vs se tienen los valores correspondientes de los grupos para cada procesador; se procede al filtrado de los objetos de cada vector, para facilitar lo anterior se crea un nuevo vector aH que contiene la información de todos los objetos anteriores.

Se procede a eliminar todos los objetos que no tienen un área dentro de un rango determinado. Sea R el radio del área deseada, pero como es muy difícil encontrar dos objetos con la misma área, se procede a tener un rango de error se d1 dicho rango, en las imágenes procesadas aparecía gran heterogeneidad en tamaños, por eso se hace antes un proceso para detectar los objetos que están fuera de ese rango con un rango un poco más grande. Sea $d_2$ dicho rango, entonces se procede a eliminar directamente del vector aH todos los elementos que no están dentro del rango $R \pm d_2$.

Debido a que todos los objetos tienen formas cercanas a discos, no son perfectos, (véase figura 2.22) se debe obtener de cada objeto el radio más grande a partir del centroide y ese valor será comparado con R.

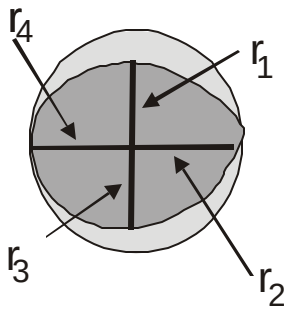



Figura 2.22. Diferentes radios de un objeto no uniforme.

Para calcular los diferentes radios se coloca en el centroide de la figura y, por ejemplo para el radio $r_1$, se recorre una posición sobre el eje vertical (hacia arriba). Si no se encuentra un valor 0 (fondo) se incrementa en 1 un contador, en caso contrario se ha llegado al borde del objeto y el radio $r_1$ es el valor del contador.

El siguiente código ilustra la búsqueda del radio superior.

```
int perArriba(int vx, int vy)
{       int i, cont=1;
        for(i=vy; i>0; i--)
        {       if(m[i][vx]==0)
                        return cont;
                else
                        cont++;
        }
}
```

El siguiente código ilustra la obtención del radio mayor:

```
int calculaRadio(int vx, int vy)
{       //Cálculo del radio hacia arriba
        int maximo=perArriba(vx, vy);
        //Cálculo del radio hacia abajo
        int tmp=perAbajo(vx,vy);
        //Búsqueda del máximo
        if(tmp>maximo)
                maximo=tmp;
        //cálculo del radio a la  izquierda
        tmp=perIzquierda(vx,vy);
        if(tmp>maximo)
                maximo=tmp;
        //Cálculo del radio a la  derecha
        tmp=perDerecha(vx,vy);
        if(tmp>maximo)
                maximo=tmp;

        return maximo;
}
```

De los radios obtenidos se busca el mayor, y si dicho radio no está en el rango $R \pm d_1$, se elimina el objeto correspondiente a dicho radio.

La última parte de la implementación consiste en trazar anillos alrededor de la imagen e identificar cuantos objetos se localizan en cada anillo. El proceso se ilustra en la figura 2.23.

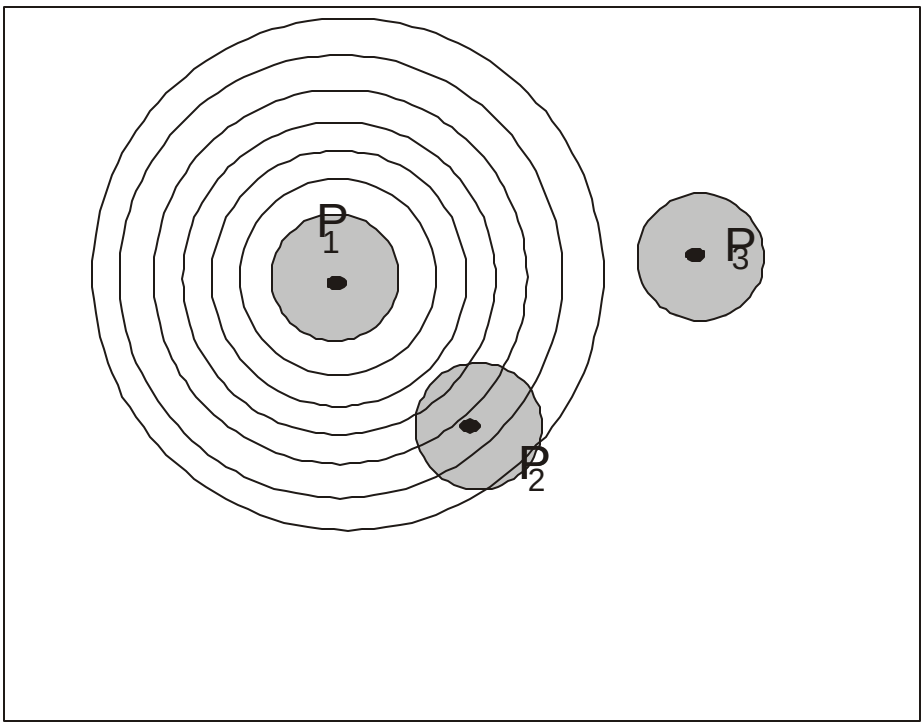


Figura 2.23. Proceso de trazado de anillos.

Como se describe en la sección 1.1, los anillos son de un grosor de r/5, donde r es el radio de la partícula, obtenido según se muestra en la figura 2..
Para trazar los anillos existe la limitante de que sólo se pueden trazar mientras no se encuentre un anillo que toque un extremo de la imagen una ilustración de lo anterior se observa en la figura 2.20.

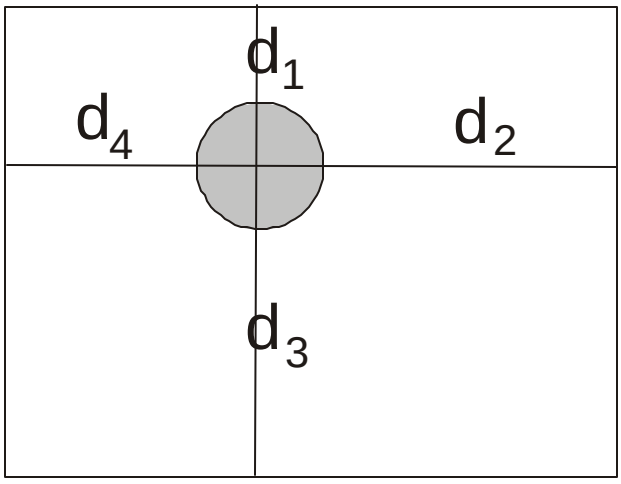


Figura 2.24. Identificación de la distancia menor del centroide a los extremos de la imagen.

Se puede identificar en la imagen 2.24 que la distancia del centroide (la intersección de las líneas vertical y horizontal) a los bordes de la imagen son $d_1$,$d_2$,$d_3$ y $d_4$, pero de ellos $d_1$ es el menor. Por lo tanto radios de los anillos trazados en la figura 2.23 no deberán exceder la distancia $d_1$.

El valor de $d_1$ se obtiene con el siguiente código

```
int radios(int vx, int vy)
{     int radio1=radio;
      int radio2=radio1+radio/10;
      //Búsqueda del radio máximo al que se hará la búsqueda de objetos
      //dentro de los anillos
      int maximo=vx;
      if(vy<maximo)
            maximo=vy;
      if((alto-vy)<maximo)
            maximo=alto-vy;
      if((ancho-vx)<maximo)
            maximo=ancho-vx;

      while(radio2<=maximo)
      {
            radio1=radio2;
            radio2=radio1+radio/10;
      }
      return 0;
}
```

donde *vx* y *vy* corresponden al centroide de la figura.

El siguiente módulo permite discriminar los objetos en base a dos criterios:
El primer criterio elimina todos los objetos que están fuera del rango radio±$delta_2$, donde $delta_2$ > $delta_1$ y $delta_1$, $delta_2$>0. Este criterio elimina los valores que están muy fuera del rango radio±$delta_1$, si se considera $delta_1$ suficientemente menor que $delta_2$, esto implica reducir el número de búsquedas de radios usando el criterio ilustrado en la figura 2.18.

```
void dejarObjetos(float radio, float delta1, float delta2)
{     int i;
      int radioMaximo;
      float a1=( 3.141592*(radio+delta2)*(radio+delta2));
      float a2=( 3.141592*(radio-delta2)*(radio-delta2));

      printf("\n\nÁreas: %d %d \n", a1,a2);

      for(i=2; i<VTam; i++)
            if(V[i]!=0 && V[i]!=1)
            {     if(!(Vs[i]>a2 && Vs[i]<a1))
                        V[i]=0;//Se descarta el objeto para el analisis
                  else
                  {     //Búsqueda del radio más grande de la figura
                        radioMaximo=calculaRadio(Vx[i], Vy[i]);
                        //si el radio está en el rango
                        V[i]=0;
                        if(radioMaximo<=radio+delta1 && radioMaximo>=radio-delta1)
                        {                    printf("\n radio: %d\t radio+d: %d\t radio-d: %d\t (%d,
%d)",radioMaximo,radio+delta1,radio-delta1, Vx[i], Vy[i]);
                              V[i]=2;
                        }
```

```
                }
            }
}
```

El segundo criterio elimina los radios que no están dentro del rango $radio \pm delta_1$ note que en el código la eliminación consiste en colocar un valor 0 en *V[i]* y el caso contrario en asignar un valor 2 a *V[i]*.

El siguiente fragmento de código hace el trazado de los anillos para un objeto que se encuentra en la posición *aHx[ind], aH[ind]* que es el arreglo que contiene todos los objetos de igual tamaño que fueron identificados en la imagen .

```
void linea(int ind, int maximo, FILE *arch)
{       float radio=(float)aHsup[ind];
        float radio2=radio+radio/10;
        float dist, delta= (float)aHsup[ind]/10;
        int i;
        int cont=0;
        int cont2;
        //X Y Radio Área AnchoAnillo radio_analizado
        fprintf(arch, "\n%d %d %d %d %d %3.2f", aHx[ind], aHy[ind], (int)(radio*factor),
(int)(radio*radio*3.141592*factor*factor), maximo, delta );

        while(radio2<=maximo)
        {       cont2=0;
                for(i=0; i<VTam; i++)
                        if(i!=ind && aH[i]==2)
                        {       dist=distancia(aHx[i],aHy[i], aHx[ind], aHy[ind]);
                                if(dist>=radio && dist<=radio2)
                                {       //Pn[cont]++;
                                        cont2++;
                                }
                        }

                Pn[cont]+=cont2;
                fprintf(arch, " %d", cont2);

                Pv[cont]++;
                cont++;
                radio=radio2;
                radio2=radio+delta;

        }//while

        fprintf(arch, " %d", cont);

        if(cont>PCont)
        {       PCont=cont;
        }
}
```

El siguiente código recorre todos los objetos que son de igual tamaño dentro de la imagen, y al encontrar alguno llama a la función *línea*, que hace el proceso de trazar los anillos.

Usando este módulo y el anterior se crea una tabla de datos (véase tabla 3.1) que se explica en el capítulo III.

```
int tabla( )
{	PCont=0;
	int i;
	FILE *arch;

	if((arch=fopen("tabla.dat", "w"))==NULL)
	{	printf("\n    No se pudo crear la tabla \n");
		return 0; //no exito
	}
	fprintf(arch, "Analisis de las distancias \n");
	fprintf(arch, "PosX PosY Radio Área Ancho_del_anillo Radio_analizado Anillos");


	for(i=0; i<VTam; i++)
	{	Pn[i]=0;
		Pv[i]=0;
	}
	PCont=0;

	for(i=0; i<VTam; i++)
	{	if(aH[i]==2)
		{	linea(i, radios(aHx[i], aHy[i]), arch);
		}
	}

	fprintf(arch, "\nPartículas: ");
	for(i=0; i<PCont; i++)
	{	fprintf(arch," %d", Pn[i]);
	}
	fprintf(arch, "\nNo.Analisis ");
	for(i=0; i<PCont; i++)
	{	fprintf(arch," %d", Pv[i]);
	}

	fprintf(arch, "\nPromedios ");
	for(i=0; i<PCont; i++)
	{	if(Pv[i]==0)
			fprintf(arch, " NE");
		else
			fprintf(arch, " %4.4f", (float)Pn[i]/(float)Pv[i]);
	}
	fprintf(arch, "\n");
	return 1;
}
```

### 2.3.1.9. Programa principal

El programa principal consiste en hacer las llamadas correspondientes a los módulos y hacer la comunicación entre los procesadores.

```
void main(int argc, char **argv)
{	//En este arreglo se define de qué a qué columna procesará cada procesador
	int iniFin[2]={ancho/2+1, ancho+1};
```

```
float radio, delta1, delta2;


//Variable para saber qué procesador es
int soy;
//Variable para conocer el estado de los mensajes
MPI_Status estado;


//Inicializo MPI
MPI_Init(&argc, &argv);
//Obtencion del identificador del procesador
MPI_Comm_rank(MPI_COMM_WORLD, &soy);

if(soy==0)
{       int tmp, i,j;

        //Abrir un archivo de imagen
        tmp=abreGrises();
        MPI_Send(&tmp, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
        //Si el archivo se abrio exitosamente, continuo, si no salgo
        if(tmp!=0)
        {       //guardaGrises("entrada.pcx");
                //Deben enviarse dos columnas más para que se pueda aplicar el filtro suavizante.
                enviaMatriz(ancho/2-1, ancho+1, 1);

                //Se procesan dos columnas posteriores y se invierten, ya que el filtro P.B.
                //usa esas dos columnas para aplicar la máscara
                invertir(0,ancho/2+2);
                //Se aplica el filtro P.B.
                filtroPB();
                masc5x5(2, ancho/2, 25,0);

                //Se hacen los tonos negros más obscuros, y los más claros se vuelven blancos
completamente
                nivel(1, ancho/2, 120,255);
                invertir(0, ancho/2);

                recibeMatriz(ancho/2+1, ancho+1, 1);
                guardaGrises("inv.pcx");

                //Se binariza la imagen (se vueve 1's y 0's) para proceder a la
                //etiquetación de componentes.
                binarizador(0, ancho/2, 160);

                ////.....////
                //  COMIENZA EL PROCESO DE ELIMINACIÓN DE LOS HOYOS
                ////.....////
                eliminaHoyos(1, iniFin[0]+1);

                //Se envía la columna más a la derecha.
                for(j=1; j<alto+1; j++)
                        vectorV[j]=m[j][iniFin[0]];
                MPI_Send(vectorV, alto+2, MPI_INT, 1, 17, MPI_COMM_WORLD);
```

```
//Hasta aquí los hoyos ya están etiquetados
//Ahora hay que sacar las características de los hoyos
//después enviar esas características al otro procesador
//y recibir un vector donde diga si algunos hoyos en realidad no son
//fondo, y en este caso tomar esos hoyos como fondos para obtener la
//nueva matriz binarizada.
característicasHoyos(1, iniFin[0]+1);
//Se envía la lista de Hoyos

        //enviar la lista de los identificadores de los Hoyos
        MPI_Send(H, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

        //enviar el vector con el centroide x
        MPI_Send(Hx, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

        //enviar el vector con el centroide y
        MPI_Send(Hy, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

        //enviar el vector con el área de los pixeles (cuantos son)
        MPI_Send(Hs, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);


//Se recibe el vector modificado que sirve para saber si el hoyo es en realidad fondo
o no.
MPI_Recv(vectorV,alto+2,MPI_INT, 1, 17, MPI_COMM_WORLD, &estado);

//Se quitan los grupos que son adyacentes y los que sí lo son, pongo en H[i]=1
quitarGposIzquierda(iniFin[0]);

//Se hace la nueva matriz binarizada.
nuevaMatrizHoyos(1, iniFin[0]);

recibeMatriz(ancho/2+1, ancho+1, 1);
matrizPCX("bin.pcx");


////.....////
//  COMIENZA EL PROCESO DE CONTEO DE LOS OBJETOS YA
RELLENOS
////.....////
cuentaObjetos(1, iniFin[0]+1);

//Enviar la columna más a la derecha.
for(j=1; j<alto+1; j++)
        vectorV[j]=m[j][iniFin[0]];
MPI_Send(vectorV, alto+2, MPI_INT, 1, 17, MPI_COMM_WORLD);

//Enviar los vectores con caracteristicas
        //enviar la lista de los identificadores de los Hoyos
        MPI_Send(V, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

        //enviar el vector con el centroide x
        MPI_Send(Vx, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

        //enviar el vector con el centroide y
        MPI_Send(Vy, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);
```

```
                        //enviar el vector con el área de los pixeles (cuántos son)
                        MPI_Send(Vs, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

                quitarGpos2Izquierda(iniFin[0]);

                nuevaMatrizObjetos(1, iniFin[0]+1);

                //Introducción de los valores para dis criminar
                printf("\nRadio: ");
                scanf("%f", &radio);

                printf("\nDelta 1: ");
                scanf("%f", &delta1);

                printf("\nDelta 2: ");
                scanf("%f", &delta2);

                //Se toma el radio según la proporción real con respecto a las imágenes originales

                radio=radio/factor;

                MPI_Send(&radio, 1, MPI_FLOAT, 1, 17, MPI_COMM_WORLD);
                MPI_Send(&delta1, 1, MPI_FLOAT, 1, 17, MPI_COMM_WORLD);
                MPI_Send(&delta2, 1, MPI_FLOAT, 1, 17, MPI_COMM_WORLD);


                dejarObjetosV(radio, delta1, delta2);
                //Enviar los vectores con caracteristicas
                        //enviar la lista de los identificadores de los Hoyos
                        MPI_Send(V, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

                        //enviar el vector con el centroide x
                        MPI_Send(Vx, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

                        //enviar el vector con el centroide y
                        MPI_Send(Vy, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

                        //enviar el vector con el área de los pixeles (cuantos son)
                        MPI_Send(Vs, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);

                        //enviar el vector con el radio de cada objeto
                        MPI_Send(Vsup, VTam, MPI_INT, 1, 17, MPI_COMM_WORLD);


                enviaMatriz(1,iniFin[0], 1);

        }
}else if(soy==1)
{       int tmp, i,j;
        MPI_Recv(&tmp,1, MPI_INT, 0, 10, MPI_COMM_WORLD, &estado);
        //tmp tiene el valor 1 si el archivo de imagen se abrio exitosamente
        if(tmp!=0)
        {       recibeMatriz(ancho/2-1,ancho+1, 0);
```

```
//Se procesan dos columnas anteriores y se invierten ya que el filtro P.B.
//usa esas dos columnas para aplicar la máscara
invertir(ancho/2-1, ancho+1);

//Se aplica el filtro P.B.
filtroPB();
masc5x5(ancho/2+1,ancho-1 ,25,0);

//Se hacen  los tonos negros más obscuros, y los más claros se vuelven blancos
completamente
nivel(ancho/2+1, ancho+1, 120,255);
invertir(ancho/2+1, ancho+1);

//envia la parte de la martiz que procesó
enviaMatriz(ancho/2+1, ancho+1, 0);

//Se binariza la matriz para el siguiente proceso que es la etiquetación
binarizador(ancho/2+1, ancho+1, 160);

////.....////
//  COMIENZA EL PROCESO DE ELIMINACIÓN DE LOS HOYOS
////.....////


//Se pone en uno al extremo izquierdo para poder calcular los hoyos.
for(i=0; i<alto+2; i++)
        m[i][iniFin[0]-1]=1;

//Se realiza el conteo de los hoyos
eliminaHoyos(iniFin[0],iniFin1]+1);

//Se recibe la columna más a la derecha de la matriz del otro procesador
MPI_Recv(vectorV, alto+2, MPI_INT, 0, 17, MPI_COMM_WORLD, &estado);
for(j=1; j<alto+1; j++)
        m[j][iniFin[0]-1]=vectorV[j];


//Cálculo las características de cada uno de los hoyos
caracteristicasHoyos(iniFin[0], iniFin1]+1);

//Se recibe la lista de Hoyos
        //enviar la lista de los identificadores de los Hoyos
        MPI_Recv(aH, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el centroide x
        MPI_Recv(aHx, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el centroide y
        MPI_Recv(aHy, VTam, MPI_INT,0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el área de los pixeles (cuántos son)
        MPI_Recv(aHs, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);
```

```
//Se arreglan  los hoyos y  se eliminan los que en realidad son fondos
arreglaHoyos(iniFin[0]);


//Enviar vector modificado que sirve para saber si el hoyo es en realidad fondo o
no.
MPI_Send(vectorV, alto+2, MPI_INT, 0, 17, MPI_COMM_WORLD);

//Se quitan los grupos que son adyacentes a los de la otra matriz
quitarGposDerecha(iniFin[0]);


//Cuenta los grupos adyacentes y los muestra.
gruposIguales(1);


//Se hace la nueva matriz binarizada.
nuevaMatrizHoyos(iniFin[0], iniFin1]+1);

enviaMatriz(ancho/2+1, ancho+1, 0);

////.....////
//  COMIENZA EL PROCESO DE CONTEO DE LOS OBJETOS YA
RELLENOS
////.....////


cuentaObjetos(iniFin[0], iniFin1]);

//Se recibe la columna más a la derecha de la matriz del otro procesador
MPI_Recv(vectorV, alto+2, MPI_INT, 0, 17, MPI_COMM_WORLD, &estado);
for(j=1; j<alto+1; j++)
        m[j][iniFin[0]-1]=vectorV[j];

//Se recibe la lista de características de los objetos
        //enviar la lista de los identificadores de los Hoyos
        MPI_Recv(aH, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el centroide x
        MPI_Recv(aHx, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el centroide y
        MPI_Recv(aHy, VTam, MPI_INT,0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el área de los pixeles (cuantos son)
        MPI_Recv(aHs, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

arreglaObjetos(iniFin[0]-1);

quitarGpos2Derecha(iniFin[0]);
```

```
gruposIguales(2);

nuevaMatrizObjetos(iniFin[0], iniFin1]+1);
MPI_RecvSend(&radio, 1, MPI_FLOAT,  0, 17, MPI_COMM_WORLD,
&estado);
MPI_Recv (&delta1, 1, MPI_FLOAT,  0, 17, MPI_COMM_WORLD, &estado);
MPI_Recv (&delta2, 1, MPI_FLOAT,  0, 17, MPI_COMM_WORLD, &estado);


//Se recibe la lista de características de los objetos
        //enviar la lista de los identificadores de los Hoyos
        MPI_Recv(aH, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el centroide x
        MPI_Recv(aHx, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el centroide y
        MPI_Recv(aHy, VTam, MPI_INT,0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el área de los pixeles (cuantos son)
        MPI_Recv(aHs, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);

        //enviar el vector con el radio de cada objeto
        MPI_Recv(aHsup, VTam, MPI_INT, 0, 17, MPI_COMM_WORLD,
&estado);


recibeMatriz(1,iniFin[0], 0);

//dejarObjetosV(15, 3, 4);
for(i=0; i<VTam; i++)
{       if(aH[i]>1)
        {       printf("\naH: %d\t%d\t%d\t%d", aH[i], aHx[i], aHy[i], aHs[i]);
                aceptarObjeto(aHx[i], aHy[i]);
        }
}

dejarObjetosV(15, 3, 4);
for(i=0; i<VTam; i++)
{       if(V[i]>1)
        {       printf("\nV: %d\t%d\t%d\t%d", V[i], Vx[i], Vy[i], Vs[i]);
                aceptarObjeto(Vx[i], Vy[i]);
                agregarVaH(0,i);
        }
}
printf("\n");

dejarObjetosG(15,3,4);
for(i=0; i<alto; i++)
{       if(G[i]>0)
        {       printf("\nG: %d\t%d\t%d\t%d", G[i], Gx[i], Gy[i], Gs[i]);
```

```
                                        aceptarObjeto(Gx[i], Gy[i]);
                                        agregarGaH(0,i);
                                }
                        }
                        printf("\n");

                        tabla();
                        eliminarObjetosMatriz(iniFin[0]+1, iniFin1]);
                        eliminarObjetosFinal();
                        matrizPCX("redux.pcx");
                }
        }

        MPI_Finalize();
        printf("\n");
}
```

### 2.3.2. Análisis de ronchigramas

El programa desarrollado para la obtención de los esqueletos de las imágenes de los ronchigramas hace un preprocesamiento similar al de las imágenes de coloides hasta el punto de obtener la imagen binarizada (véase figura 3.20). Para el preprocesamiento de las imágenes de ronchigramas se usa el filtro laplaciano para la detección de bordes, el algoritmo de equalización del histograma para mejorar el contraste de la imagen y los dos agoritmos de cambio de contraste.

A partir de la imagen binarizada, se obtiene el esqueleto de las franjas aplicando el algoritmo de adelgazamiento que se muestra en la sección 2.2.4, éste se ilustra en el siguiente fragmento de código:

```
void esqueleto()
{  int dir=0;
  int i,j,p, continuar=0;
        do      // Las direcciones no terminan
        {       for (i=2 ; i<alto+2; i++)
                        for(j=2; j<ancho+2; j++)
                        {       p=buscaPixel(dir,i,j);
                                if (p==1)
                                        if ((IndiceE(i,j)==1)&&(valencia(i,j)!=1))
                                        {  m[i][j]=3;
                                        }
                        }

                continuar=0;
                for (i=2 ; i<alto+2; i++)
                        for(j=2; j<ancho+2; j++)
                        {       if (m[i][j]==3)
                                {       m[i][j]=0;
                                        continuar=1;
                                }
                        }
                dir++;
                if(dir==4)
                        dir=0;
        }
        while (continuar==1);
}
```

Obtenido el esqueleto para la imagen binaria y binaria invertida se debe asignar un orden a cada franja, para eso se recorre la matriz *filtrada* (que contiene la información de los esqueletos de las franjas blancas y negras) sobre una columna en la posición ancho/2 (la mitad de la imagen horizontalmente) y cada vez que se localiza una franja, a ésta se le asigna el valor de un contador (orden) que se va incrementando en 0.5 cada vez que se encuentra el esqueleto de una franja. Después de localizado dicho píxel se utiliza un algoritmo recursivo que tiene la función de etiquetar a todos los elementos de esa franja, dichos módulos se encuentran enseguida.

Código para asignar órdenes de interferencia a las franjas.

```
//Reetiqueta lineas horizontales.
void reetiquetaxDoble()
{       int i, j;
        float cont=0;

        printf("\n");
        for(i=0; i<alto+4; i++)
        {               if(filtrada[i][ancho/2]!=0)
                        {       cont+=0.5;
                                vecinos(ancho/2, i, cont);
                        }

        }

}
```

Módulo para asignar la etiqueta *valor* al esqueleto de una franja que tiene un elemento localizado en la posición *x*, *y* de la matriz *filtrada*.

```
int vecinos(int x, int y, float valor)
{       if(filtrada[y][x]==0 || filtrada[y][x]==valor)
                return 0;
        else
        {       filtrada[y][x]=valor;
                vecinos(x-1,y-1, valor);
                vecinos(x,y-1, valor);
                vecinos(x+1,y-1, valor);
                vecinos(x+1,y, valor);
                vecinos(x+1,y+1, valor);
                vecinos(x,y+1, valor);
                vecinos(x-1,y+1, valor);
                vecinos(x-1,y, valor);
        }
}
```

Teniendo la matriz con órdenes asignados a cada esqueleto de cada franja, los datos de la matriz *filtrada* son escritos en un archivo mediante la función *generarArchivo* que se muestra a continuación.

```
void generarArchivo()
{       FILE *arch;
        int i,j;
        char nArch[100];
```

```
        printf("\nNombre del archivo a generar:  ");
        scanf("%s", nArch);

        if((arch=fopen(nArch, "w"))==NULL)
        {       printf("No se pudo guardar la matriz....");
        }else
        {       for (i=2 ; i<alto+2; i++)
                        for(j=2; j<ancho+2; j++)
                        {       fprintf(arch, "%d %d %2.1f\n", j-2, i-2, filtrada[i][j]);
                        }
                fclose(arch);
        }
}
```

El formato de salida del archivo es el siguiente para cada píxel de la matriz:

Posición_en_X, posición_en_Y, filtrada[y][j]

Posición_en_X es el índice de la columna sobre la que se encuentra el píxel en la matriz.
posición_en_Y es el índice de del renglón sobre el que se encuentra el píxel en la matriz.

filtrada[y][j] es valor del orden de interferencia asociado al píxel.

En el anexo I se muestra un fragmento de un archivo de salida correspondiente a los órdenes de interferencia para la figura 3.24.

# CAPÍTULO III
Resultados

## 3.1. Resultados del análisis de las imágenes de coloides.

En el presente trabajo se analizaron imágenes de coloides como la que se presenta en la figura 3.1, dicha imagen es transformada a escala de grises (fig. 3.2).

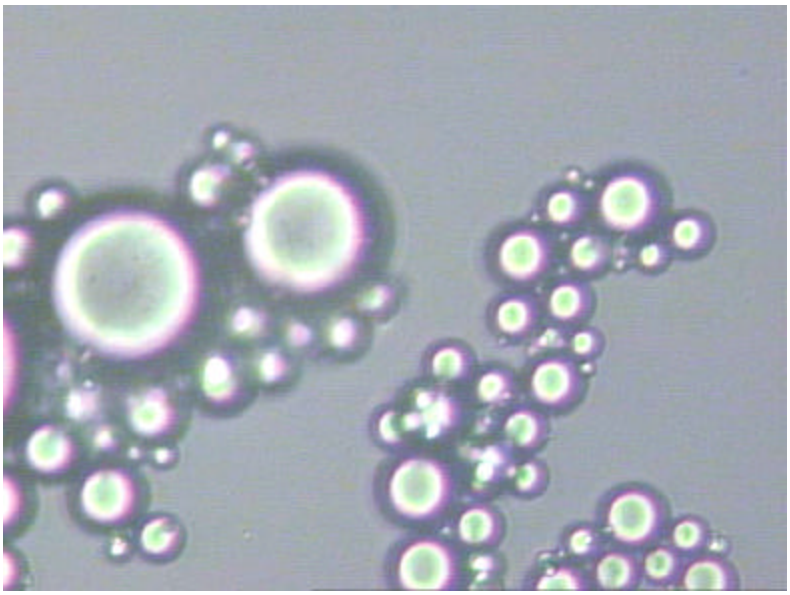


Figura 3.1 Imagen original

La aplicación desarrollada parte de una imagen en escala de grises (fig. 3.2) generada usando el programa Corel versión 10.0 a la que se le aplican algoritmos de preprocesamiento para obtener una imagen binarizada (fig. 3.6).

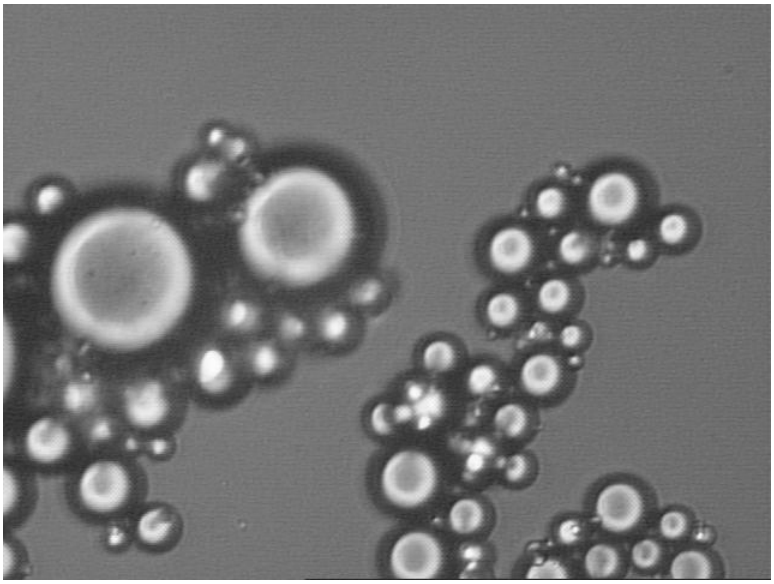


Figura 3.2. Imagen original en escala de grises.

La idea inicial consiste en identificar a los objetos y separarlos del fondo. Como se puede ver en la figura 3.2, los conjuntos de partículas están rodeados de zonas obscuras, esto significa que al aplicarle un algoritmo para obtener el negativo de la imagen, las zonas más claras (que son las partículas) tomarán un color muy oscuro y las zonas oscuras serán muy blancas; por ello, el algoritmo de inversión de la imagen (primer filtro aplicado) hace que las partículas contrasten con el fondo. Lo anteriormente mencionado se ilustra en la figura 3.3.

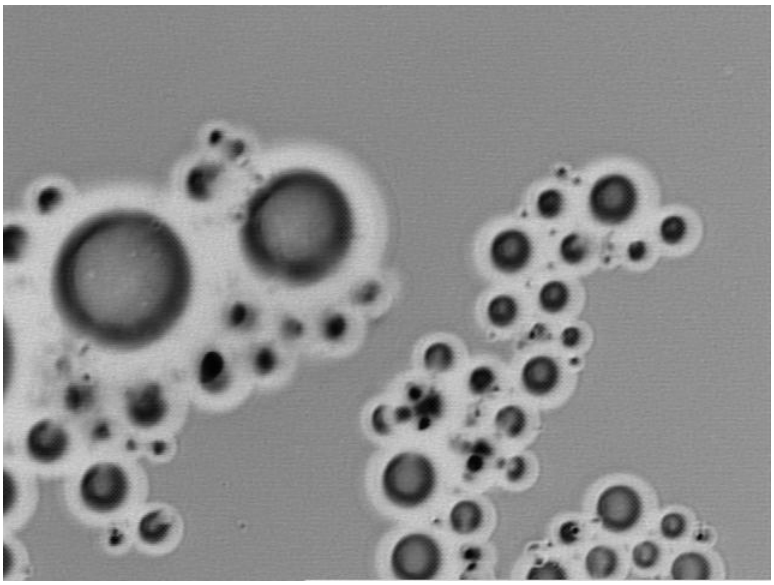


Figura 3.3. Imagen después de invertirla

En las imágenes generalmente existen rugosidades y pequeños detalles. Para poder eliminar lo anterior, se aplica un segundo preproceso a la imagen, que consiste en el filtro suavizante (ver sección 2.3.1.3), el resultado se aprecia en la figura 3.4.

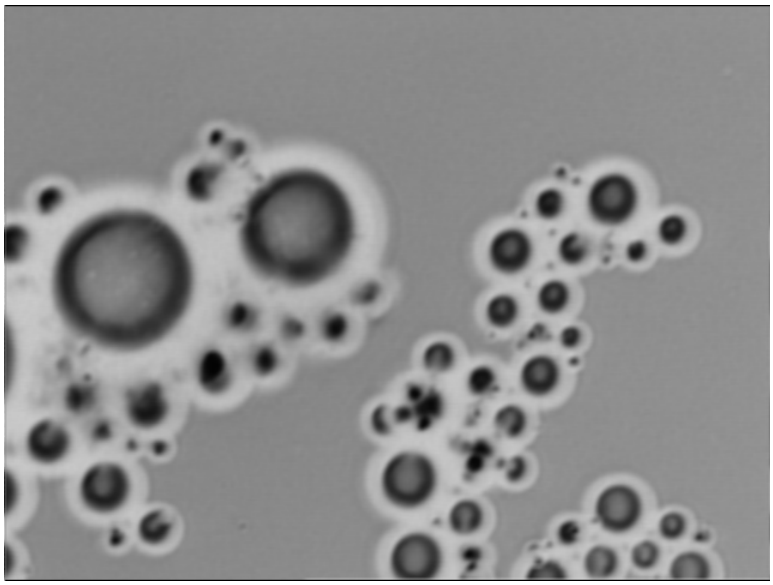


Figura 3.4. Imagen después de aplicarle el filtro suavizante

A la imagen mostrada en la figura 3.4 se le aplica el algoritmo de cambio de contraste por niveles, que permite resaltar las partes más oscuras de las más claras de la imagen, este es el tercer proceso aplicado a la imagen, su resultado se muestra en la figura 3.5.

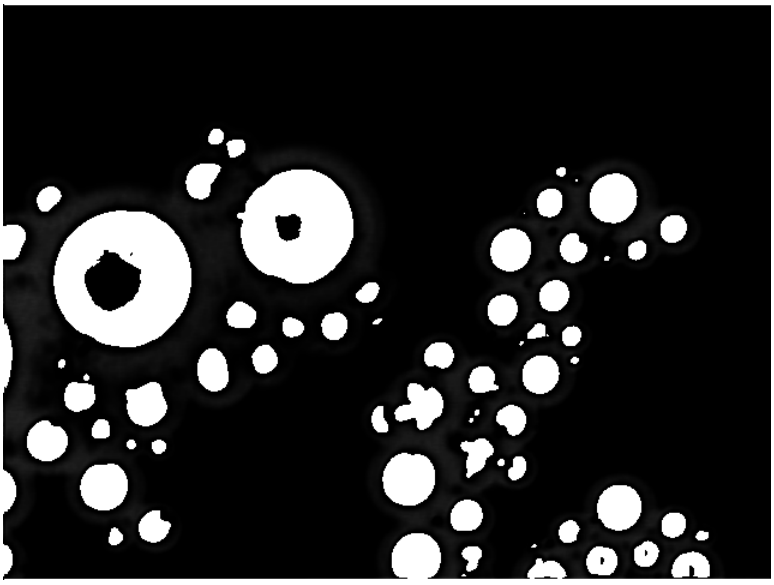


Figura 3.5. Imagen con aumento de contraste.

Después de la mejora obtenida por el aumento del contraste, debido a que los valores de los píxeles están todavía en el rango de 0 a 255, es necesario aplicar el algoritmo de umbralización para obtener una imagen binarizada, con este algoritmo termina el preprocesamiento, sus resultados se pueden apreciar en la figura 3.6. El valor del umbral usado en las imágenes procesadas fue de 160. La aplicación de otros valores para el umbral arrojaba imágenes con mucha basura.

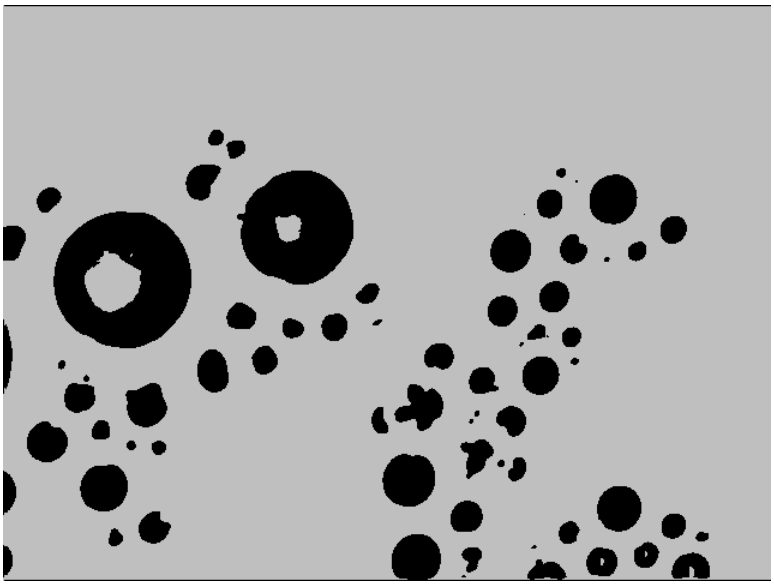


Figura 3.6. Imagen después de aplicarle la umbralización.

Después de obtener una matriz que contiene valores de ceros o unos, se procede a aplicar el algoritmo de etiquetación de componentes para detectar los posibles 4-componentes (hoyos) que pueden haber en la imagen como los que aparecen en la figura 3.6. Después de detectar los 4-componentes, los valores de los pixeles que los conforman son puestos a 1 para rellenarlos y obtener las partículas completas. La aplicación de dicho proceso genera la imagen que se muestra en la figura 3.7.

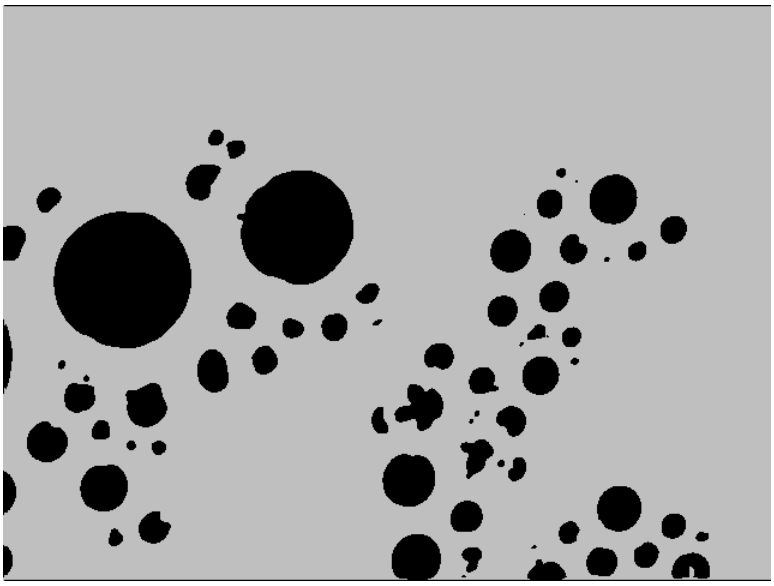


Figura 3.7 Imagen después de la eliminación de hoyos.

Con los 8-componentes (partículas) rellenos, se procede a eliminar los 8-componentes que no están en un cierto rango de tamaño, como en la figura 3.8 que es el resultado de seleccionar solamente las partículas que tienen un radio de 15±3. Las figuras 3.9 y 3.10 muestran los resultados para la discriminación con radios de 20±3 y 10±3, respectivamente.

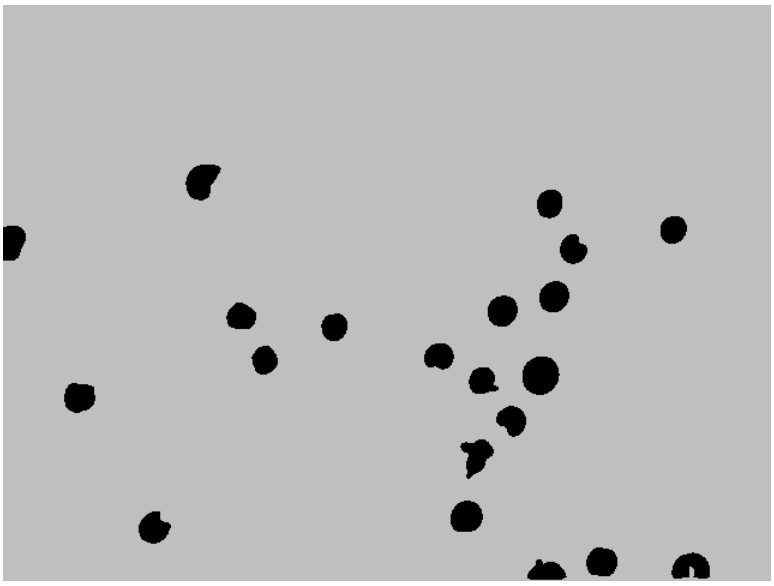


Figura 3.8. Imagen después de discriminar solo las partículas que tienen radio 15± 3

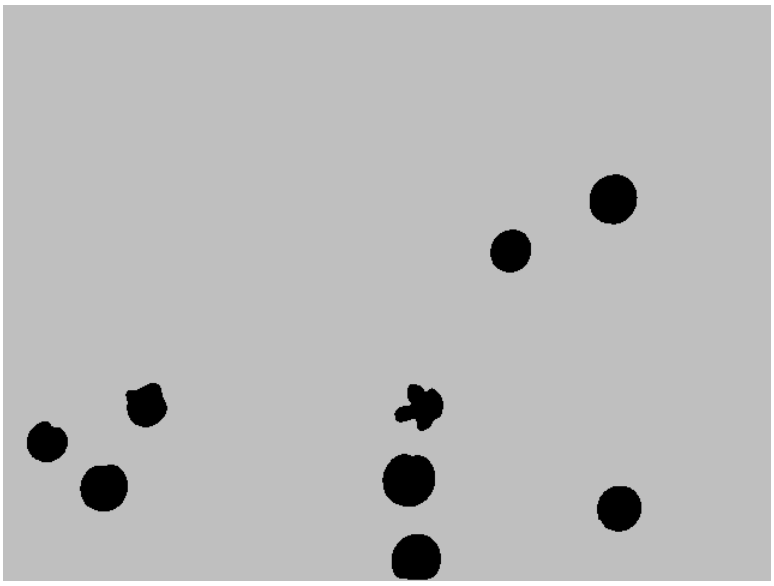


Figura 3.9. Imagen después de discriminar solo las partículas que tienen radio 20± 3

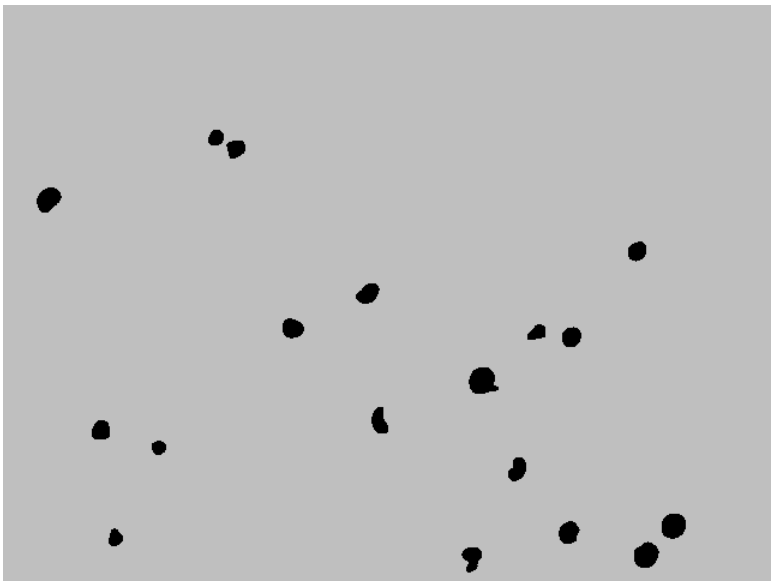


Figura 3.10. Imagen después de discriminar las partículas que tienen radio 10± 3

La parte final del proceso consiste en generar una tabla a partir de la identificación de las partículas que quedan dentro de los anillos generados para cada partícula. La tabla 3.1 es el resultado de aplicar tal proceso. Para el caso especial de los radios, el valor que se muestra es el valor del radio obtenido multiplicado por un factor, éste es una proporción del tamaño real con el tamaño obtenido del preprocesamiento, cabe notar que existe una pérdida del tamaño durante el preprocesamiento de la imagen. Dicho factor tiene un valor de 1.85. El valor fue obtenido ejecutando el programa para 10 partículas aisladas

(sobre un fondo blanco) de difetentes tamaños de la imagen original, se obtuvieron los tamaños para dichas partículas y se obtuvo la proporción con respecto a los tamaños obtenidos para la corrida con las imágenes originales.

```
Analisis de las distancias
PosX PosY Radio Area Radio_del_area_analizada Ancho_del_anillo Anillos:
509 162 23 1661 131 4.60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 23
423 205 20 1256 205 4.00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 46
448 309 18 1017 171 3.60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 42
348 335 22 1520 145 4.40 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
0 0 0 27
514 420 21 1385 60 4.20 0 0 0 0 0 0 0 0 0 9
344 461 23 1661 19 4.60 0
175 305 20 1256 175 4.00 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 0 0 0 0 38
120 334 20 1256 120 4.00 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
0 25
37 365 19 1134 37 3.80 0 0 0 0 4
84 402 22 1520 78 4.40 0 0 0 0 0 0 0 0 1 0 0 0 12
Partículas:  0 0 0 0 0 0 0 0 1 0 2 0 0 0 1 1 0 1 1 1 0 1 0 2 1 0 0 0 1 0
1 0 2 0 0 0 0 0 0 1 0 0 0 0 0 0
No.Analisis  9 9 9 9 8 8 8 8 8 7 7 7 6 6 6 6 6 6 6 6 6 6 5 5 4 4 3 3 3
3 3 3 3 3 3 3 3 2 2 2 2 1 1 1 1
Promedios  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.1250
0.0000 0.2857 0.0000 0.0000 0.0000 0.1667 0.1667 0.0000 0.1667 0.1667
0.1667 0.0000 0.1667 0.0000 0.4000 0.2000 0.0000 0.0000 0.0000 0.3333
0.0000 0.3333 0.0000 0.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.5000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Tabla 3.1.- Ubicaciones relativas entre las partículas que tienen un radio de 20± 3

### 3.2. Comparación de la implementación paralela con la secuencial

En la siguiente tabla se hace la comparación de los tiempos que tomó la implementación secuencial y la paralela para procesar desde 50 hasta 500 imágenes. En estos datos se observa una mejora de entre el 60 y 70% en la implementación paralela, dicha tendencia se muestra gráficamente en la figura 3.11.

| Número de Imágenes procesadas | Tiempo del programa secuencial, T(1) (min) | Tiempo del programa en paralelo, T(2) (min) | T(1)/T(2) |
|---|---|---|---|
| 50 | 1 | 0.6166 | 1.6166 |
| 100 | 1.9833 | 1.1666 | 1.7 |
| 150 | 3 | 1.76 | 1.698 |
| 200 | 4.02 | 2.366 | 1.697 |
| 250 | 5.033 | 2.9333 | 1.7159 |
| 350 | 6.1 | 3.533 | 1.7264 |
| 400 | 8.166 | 4.7833 | 1.7073 |
| 450 | 9.1166 | 5.333 | 1.6475 |
| 500 | 10.1666 | 5.9666 | 1.7039 |

Tabla 3.2. Tiempos de procesamiento de la imagen en paralelo y secuencial.

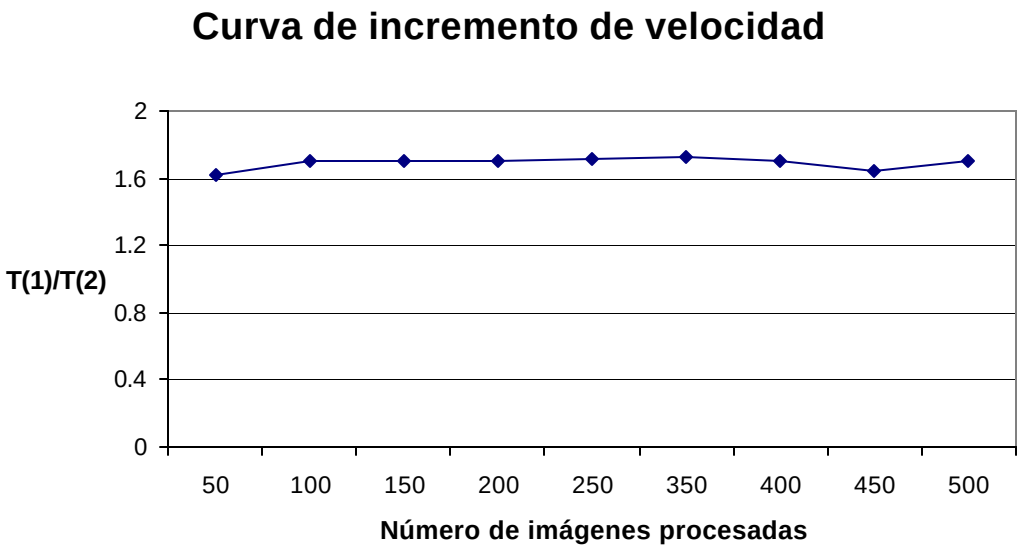



Figura 3.11. Razón del tiempo de ejecución del programa secuencial sobre el paralelo.

En la figura 3.12 se muestran los tiempos que se toma el procesar desde 50 hasta 500 imágenes con el programa paralelo y secuencial, se nota que la mejora no es muy grande para cantidades pequeñas de imágenes procesadas, pero para números grandes la mejora se aprecia más. En el caso de la aplicación real, se deberán procesar por lo menos 10,000 imágenes, y el hecho de reducir en poco menos de la mitad del tiempo total usado para procesarlas, significa una gran mejora.

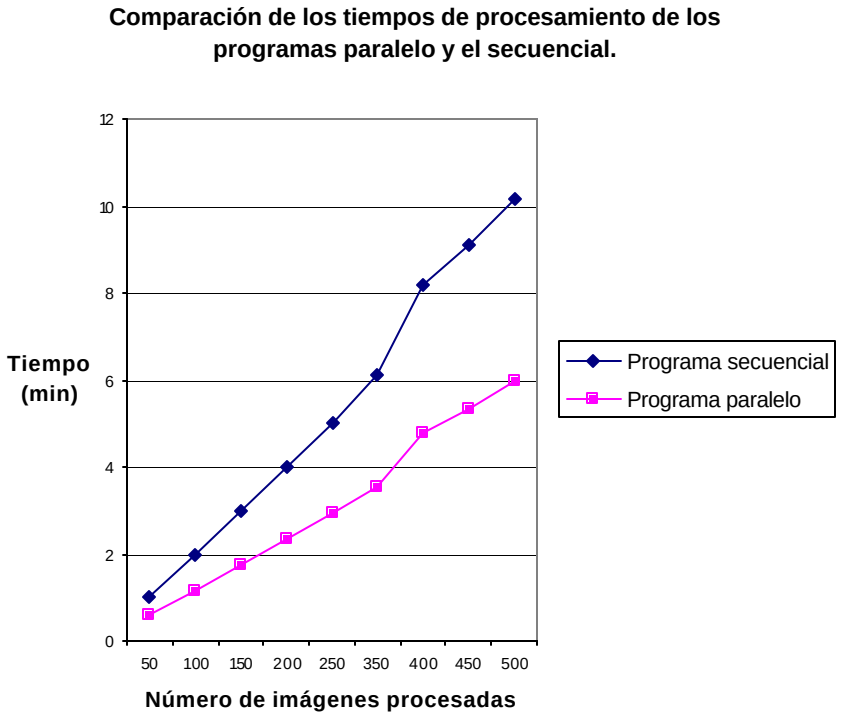



Figura 3.12. Tiempo tomado para la ejecución de la implementación paralela contra la secuencial.

### 3.3. Procesamiento de los ronchigramas

A estas imágenes se les aplicó la siguiente secuencia de algoritmos para obtener las imágenes binarizadas:

- Inversión
- Ecualización
- Filtro laplaciano
- Variación del contraste (curvas)
- Filtro suavizante pasa bajo (3 veces).
- Variación del contraste (niveles)
- Umbralización global

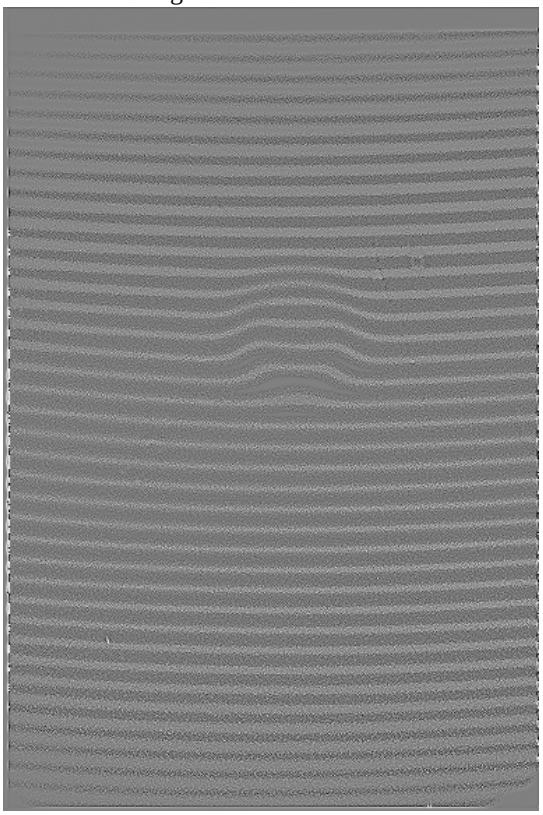
Después de tener la imagen binarizada, se aplicó el algoritmo de esqueletización.

A continuación se muestra la secuencia de imágenes procesadas.

Figura 3.13 Imagen del ronchigrama original

Figrura 3.14.Ronchigrama después de aplicarle el algoritmo de inversión

Figrura 3.15. Ronchigrama después del algoritmo de equalización de componentes.



Figrura 3.16 Ronchigrama después de aplicarle la máscara del laplaciano.

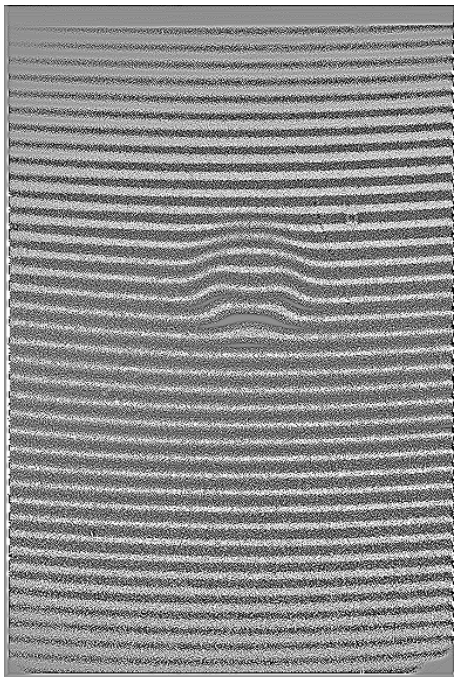
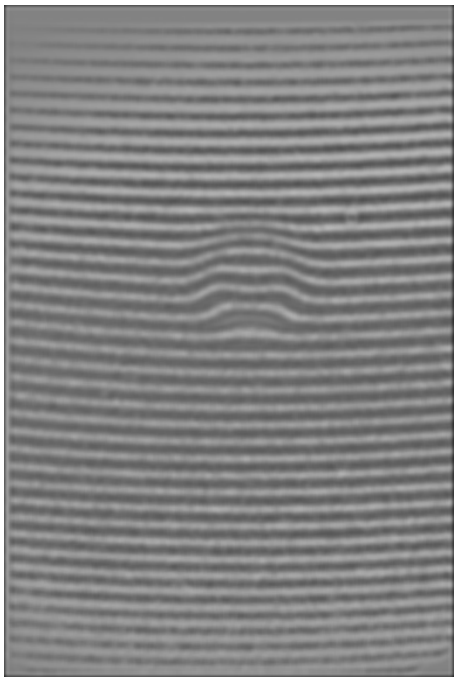
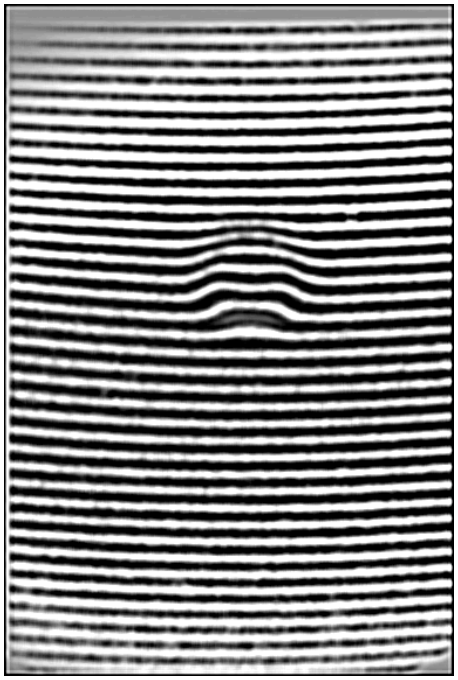


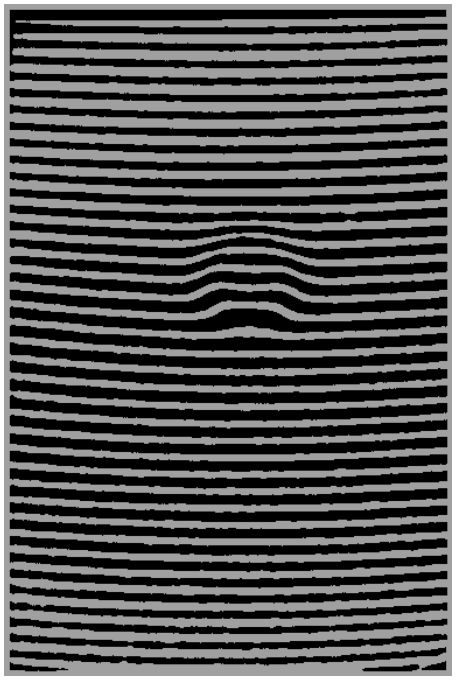
Figura 3.17 Ronchigrama después de aplicarle el algoritmo de cambio de contraste por curvas



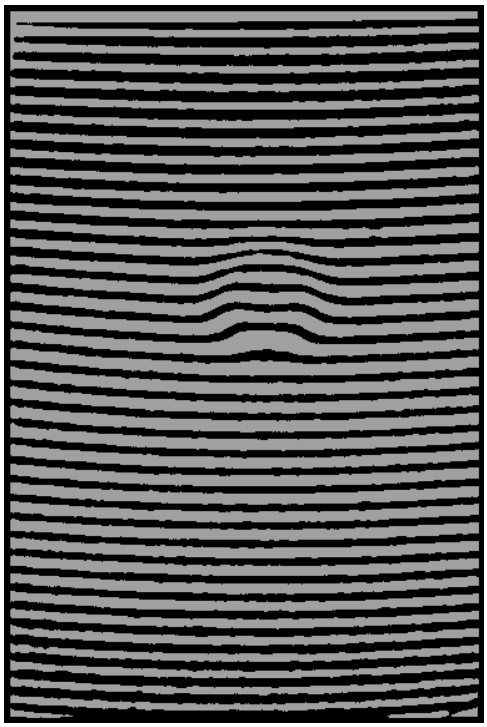
Figrura 3.18. Ronchigrama después de aplicarle un filtro suavizante



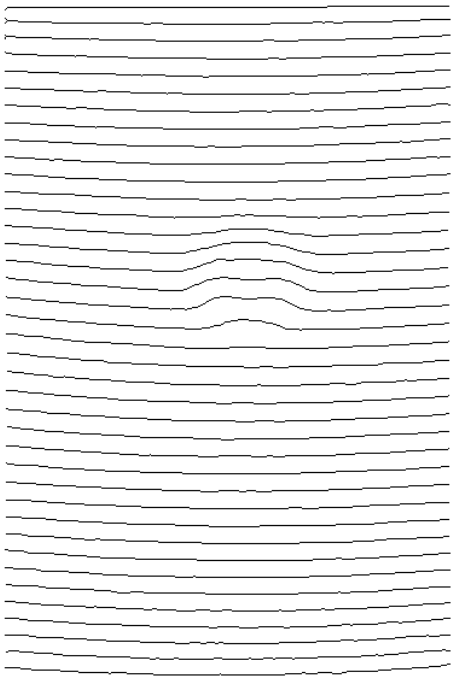
Figrura 3.19. Ronchigrama después de aplicarle el algoritmo de cambio de contraste por niveles.



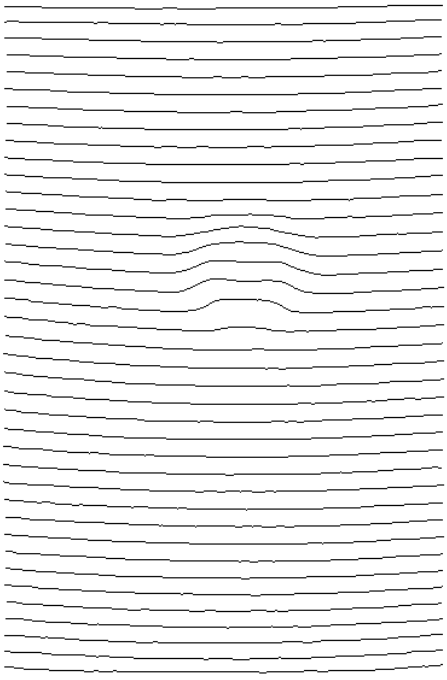
Figrura 3.20. Imagen binaria del ronchigrama (su matriz tiene sólo 0's o 1's). Esta imagen es el resultado de la aplicación del algoritmo de umbralización.



Figrura 3.21. Imagen binaria invertida.



Figrura 3.22. Componentes oscuras del ronchigrama esqueletizadas.



Figrura 3.23. Componentes blancas del ronchigrama esqueletizadas.

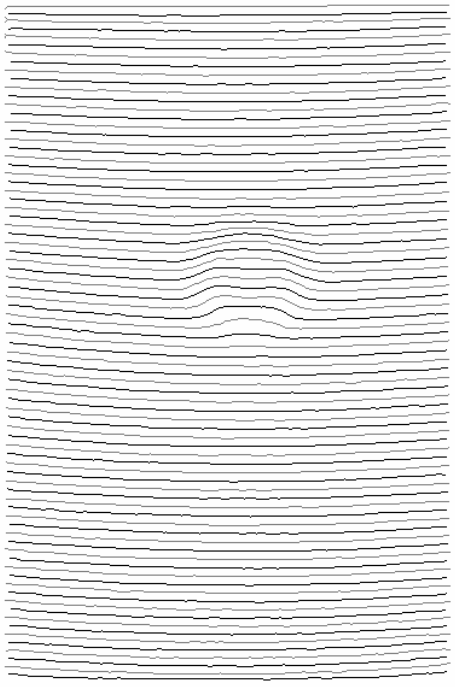


Figura 3.24. Imagen con las componentes blancas y oscuras esqueletizadas.

Un fragmento del archivo que contiene la asignación de de los órdenes de interferencia para la figura 3.24 se muestra en el anexo I.

Figura 3.25 Un ronchigrama con ruido

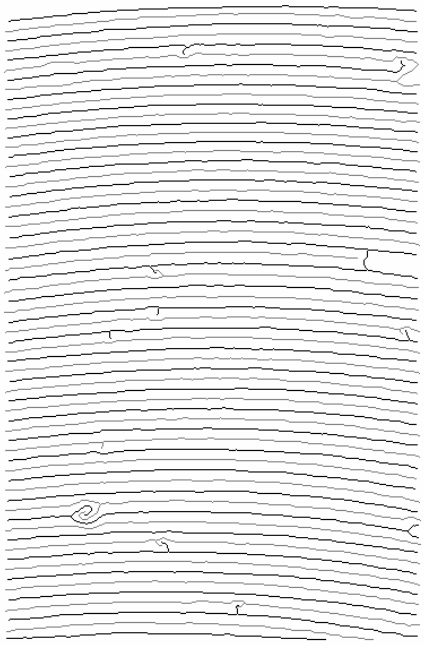


Figura 3.26 Esqueleto de la figura 3.25

Figura 3.27 Un segundo ronchigrama con ruido

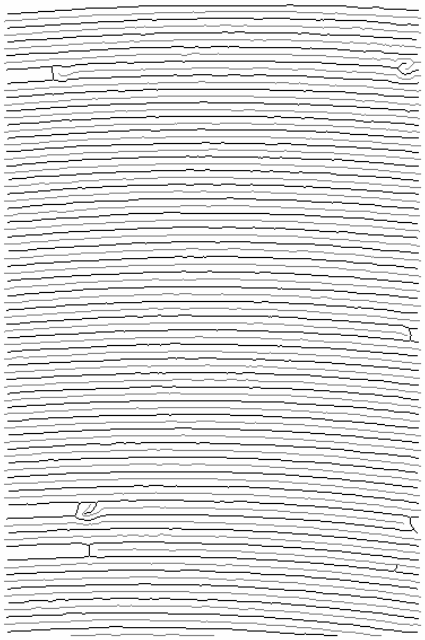


Figura 3.28. Esqueleto de la imagen de la figura 3.27

# CONCLUSIONES

Los objetivos inicialmente planteados han sido cubiertos, pues se ha desarrollado una herramienta que realiza el conteo de las partículas presentes en cada imagen de coloides, el cálculo del tamaño de cada una de ellas y su posición relativa respecto de otras. Tal conteo se hizo mediante programación paralela utilizando MPI, lo cual establece un camino a seguir para la obtención en tiempo real, de información acerca del coloide analizado. Más aún, el paradigma seguido permitirá escalar el software creado para su implementación en un clúster. En ese sentido, se ha rebasado con mucho los objetivos originalmente planteados.

El incremento de velocidad obtenido fue de entre 1.6 y 1.7 con respecto a la implementación secuencial, ello implica una mejora que servirá como referencia a próximas investigaciones en el área. Esto sugiere que se puede dar como una constante de invariancia el valor de 1.7.

**La implementación hecha da nuevas opciones sobre cómo hacer cómputo paralelo para el procesamiento de imágenes en general.**

La otra herramienta desarrollada lleva a cabo la asignación automática de los órdenes de interferencia para cierto tipo de ronchigramas. Esto permitirá agilizar el proceso de control de calidad de los dispositivos ópticos durante su fabricación.

Por otra parte, cabe aclarar que el programa para el análisis de imágenes de coloides fue probado con algunas imágenes pero no fue posible adquirir más por razones ajenas al tesista.

Una limitante a resolver será la capacidad de almacenamiento que deberá permitir trabajar con al menos 10,000 imágenes en tiempo real. Esta limitante podría ser resuelta utilizando algoritmos de compresión, lo que se traduciría en una cantidad de procesamiento adcional.

También cabe decir, que para imágenes de ronchigramas con mucho ruido no se pudo llegar a la asignación automática de los órdenes de interferencia, debido a la forma de los esqueletos, para ello es necesario mejorar las técnicas de adquisición de las imágenes.

En cuanto a la implementación paralela utilizada, podría usarse el paradigma SIMD, esto es, que cada procesador trabaje sobre una imagen diferente evitando la parte de comunicación que implica un tiempo de procesamiento extra para poder realizar la sincronización.

No obstante, a pesar de las limitaciones antes mencionadas, se han cubierto íntegramente los objetivos señalados al inicio del trabajo; habiendo contribuido a la solución de problemas reales enmarcados en el Proyecto IG26-A cofinanciado por CONACYT-BUAP.

## BIBLIOGRAFÍA


1. *A survey of heteregeneous processing concepts and systems,* URL: http://galeb.etf.bg.ac.yu/~vm/tutorial/multi/hetero/system.htm
2. Awcock G. J. *Applied image processing,* Editorial McGraw Hill, 106-108
3. Bick, Aart, Girkar, Milind, Grey, Paul, Tian, Xinmin, *Efficient Exploitation of Parallelism on Pentium III and Pentium 4 Processor-Based Systems* , Microcomputer Software Laboratories, Intel Corp.
4. *Birth of the Hypercube,* URL: http://www.npac.syr.edu/copywrite/pcw/node13.html
5. Brassard, Pilles y Bratley, Paul, *Fundamentals of algorithmics*, Prentice Hall, página 60.
6. Bykov A., Zerkalov L., *Algorithm for Homotopy Classification ob binary images, Pattern Recognition* V. 29, N.4, 1996.
7. Casavant , Thomas L. / Turdik Pavel, *Parallel Computers: Theory and Practice*, IEEE Computer Society Press
8. *Clústers en México,* URL: http://clusters.unam.mx/Historia
9. Cordero D. Alberto, Luna Aguilar E., Vázquez Montiel S., Zárate Vázquez S., Percino Zacarías M., *Ronchi Test with a square grid, aplied Optics*, Vol. 37, No. 4, pp. 1-4, 1998.
10. *Early 1980's* URL: http://www.npac.syr.edu/copywrite/pcw/node12.html
11. González C. Rafael, Woods, *Richard, Tratamiento digital de imágenes,* Addison-Wesley/Díaz de Santos
12. *ILLIAC IV*, URL: http://ed-thelen.org/comp-hist/vs-illiac-iv.html
13. Kazi, Iffat H., Lilja, David J., *JavaSpMT: A Speculative Thread Pipelining Parallelization Model for Java Programs,* Department of Electrical & Computer Engineering, Minnesota Supercomputing Institute
14. Lewis, Ted G./ El-Rewini Heshan, *Introduction to Parallel Computing*, Prentice Hall 1992.
15. *Monografias.com,* URL: http://www.monografias.com
16. *Parallel Scientific Computers Before 1980,* URL: http://www.npac.syr.edu/copywrite/pcw/node11.html
17. Patterson, David A / Hennessy John, *Organización y diseño de computadoras, La interfaz hardware/software*, McGraw Hill 2ª Edición.
18. Rodríguez P. Mario, Bykov A., Zerkalov L., *Index of a point of 3-D digital binary image and algorithm for computing its Euler characteristic, Pattern Recongnition*, V. 32., N.5, pp. 845-850, 1999.
19. Rosenfeld A., Kong T. Y., *Digital Topology: Introduction and Survey, Computer Vision, Graphics and Image Processing*, 48, 357-393, 1989.
20. *Seymour Cray: El padre de la supercomputadora,* URL: http://www.lania.mx/~ccoello/historia/papers/cray.htm

21. Shchepin E. V., Nepomnyashchii G. M., *On Topology Analysis of Images*, Mezvuzovskii Abornik nauchinh trudo, Moscú, MIR, 1990.
22. *SMP*,
URL: http://www.webopedia.com/TERM/S/SMP.html
23. *Symmetric multiprocessing (SMP)*
URL: http://www.monografias.com/trabajos6/symu/symu.shtml
24. The Message Passing Interface (MPI) standard,
URL: http://www-unix.mcs.anl.gov/mpi/
25. *TOP500 Sublist for JUN2002*,
URL: http://www1.top500.org
26. Torres Jiménez José, *Conceptos de Cómputo Paralelo*, Editorial Trillas

ANEXO I

Fragmento del archivo de salida del programa de análisis de ronchigramas, este fragmento muestra los ordenes asignados a las franjas blancas y negras de la figura 3.24:

```
0 20 0.0        64 20 0.0       128 20 0.0      192 20 0.0      256 20 0.0
1 20 0.0        65 20 0.0       129 20 0.0      193 20 0.0      257 20 0.0
2 20 0.0        66 20 0.0       130 20 0.0      194 20 0.0      258 20 0.0
3 20 0.0        67 20 0.0       131 20 0.0      195 20 0.0      259 20 0.0
4 20 0.0        68 20 0.0       132 20 0.0      196 20 0.0      260 20 0.0
5 20 0.0        69 20 0.0       133 20 0.0      197 20 0.0      261 20 0.0
6 20 0.0        70 20 0.0       134 20 0.0      198 20 0.0      262 20 0.0
7 20 0.0        71 20 0.0       135 20 0.0      199 20 0.0      263 20 0.0
8 20 0.0        72 20 0.0       136 20 0.0      200 20 0.0      264 20 0.0
9 20 0.0        73 20 0.0       137 20 0.0      201 20 0.0      265 20 0.0
10 20 1.5       74 20 0.0       138 20 0.0      202 20 0.0      266 20 0.0
11 20 1.5       75 20 0.0       139 20 0.0      203 20 0.0      267 20 0.0
12 20 1.5       76 20 0.0       140 20 0.0      204 20 0.0      268 20 0.0
13 20 1.5       77 20 0.0       141 20 0.0      205 20 0.0      269 20 0.0
14 20 1.5       78 20 0.0       142 20 0.0      206 20 0.0      270 20 0.0
15 20 1.5       79 20 0.0       143 20 0.0      207 20 0.0      271 20 0.0
16 20 1.5       80 20 0.0       144 20 0.0      208 20 0.0      272 20 0.0
17 20 1.5       81 20 0.0       145 20 0.0      209 20 0.0      273 20 0.0
18 20 1.5       82 20 0.0       146 20 0.0      210 20 0.0      274 20 0.0
19 20 0.0       83 20 0.0       147 20 0.0      211 20 0.0      275 20 0.0
20 20 0.0       84 20 0.0       148 20 0.0      212 20 0.0      276 20 0.0
21 20 0.0       85 20 0.0       149 20 0.0      213 20 0.0      277 20 0.0
22 20 0.0       86 20 0.0       150 20 0.0      214 20 0.0      278 20 0.0
23 20 0.0       87 20 0.0       151 20 0.0      215 20 0.0      279 20 0.0
24 20 0.0       88 20 0.0       152 20 0.0      216 20 0.0      280 20 0.0
25 20 0.0       89 20 0.0       153 20 0.0      217 20 0.0      281 20 0.0
26 20 0.0       90 20 0.0       154 20 0.0      218 20 0.0      282 20 0.0
27 20 0.0       91 20 0.0       155 20 0.0      219 20 0.0      283 20 0.0
28 20 0.0       92 20 0.0       156 20 0.0      220 20 0.0      284 20 0.0
29 20 0.0       93 20 0.0       157 20 0.0      221 20 0.0      285 20 0.0
30 20 0.0       94 20 0.0       158 20 0.0      222 20 0.0      286 20 0.0
31 20 0.0       95 20 0.0       159 20 0.0      223 20 0.0      287 20 0.0
32 20 0.0       96 20 0.0       160 20 0.0      224 20 0.0      288 20 0.0
33 20 0.0       97 20 0.0       161 20 0.0      225 20 0.0      289 20 0.0
34 20 0.0       98 20 0.0       162 20 0.0      226 20 0.0      290 20 0.0
35 20 0.0       99 20 0.0       163 20 0.0      227 20 0.0      291 20 0.0
36 20 0.0       100 20 0.0      164 20 0.0      228 20 0.0      292 20 0.0
37 20 0.0       101 20 0.0      165 20 0.0      229 20 0.0      293 20 0.0
38 20 0.0       102 20 0.0      166 20 0.0      230 20 0.0      294 20 0.0
39 20 0.0       103 20 0.0      167 20 0.0      231 20 0.0      295 20 0.0
40 20 0.0       104 20 0.0      168 20 0.0      232 20 0.0      296 20 0.0
41 20 0.0       105 20 0.0      169 20 0.0      233 20 0.0      297 20 0.0
42 20 0.0       106 20 0.0      170 20 0.0      234 20 0.0      298 20 0.0
43 20 0.0       107 20 0.0      171 20 0.0      235 20 0.0      299 20 0.0
44 20 0.0       108 20 0.0      172 20 0.0      236 20 0.0      300 20 0.0
45 20 0.0       109 20 0.0      173 20 0.0      237 20 0.0      301 20 0.0
46 20 0.0       110 20 0.0      174 20 0.0      238 20 0.0      302 20 0.0
47 20 0.0       111 20 0.0      175 20 0.0      239 20 0.0      303 20 0.0
48 20 0.0       112 20 0.0      176 20 0.0      240 20 0.0      304 20 0.0
49 20 0.0       113 20 0.0      177 20 0.0      241 20 0.0      305 20 0.0
50 20 0.0       114 20 0.0      178 20 0.0      242 20 0.0      306 20 0.0
51 20 0.0       115 20 0.0      179 20 0.0      243 20 0.0      307 20 0.0
52 20 0.0       116 20 0.0      180 20 0.0      244 20 0.0      308 20 0.0
53 20 0.0       117 20 0.0      181 20 0.0      245 20 0.0      309 20 0.0
54 20 0.0       118 20 0.0      182 20 0.0      246 20 0.0      310 20 0.0
55 20 0.0       119 20 0.0      183 20 0.0      247 20 0.0      311 20 0.0
56 20 0.0       120 20 0.0      184 20 0.0      248 20 0.0      312 20 0.0
57 20 0.0       121 20 0.0      185 20 0.0      249 20 0.0      313 20 0.0
58 20 0.0       122 20 0.0      186 20 0.0      250 20 0.0      314 20 0.0
59 20 0.0       123 20 0.0      187 20 0.0      251 20 0.0      315 20 0.0
60 20 0.0       124 20 0.0      188 20 0.0      252 20 0.0      316 20 0.0
61 20 0.0       125 20 0.0      189 20 0.0      253 20 0.0      317 20 0.0
62 20 0.0       126 20 0.0      190 20 0.0      254 20 0.0      318 20 0.0
63 20 0.0       127 20 0.0      191 20 0.0      255 20 0.0      319 20 0.0
```

320 20 0.0
321 20 0.0
322 20 0.0
323 20 0.0
324 20 0.0
325 20 0.0
326 20 0.0
327 20 0.0
328 20 0.0
329 20 1.5
330 20 1.5
331 20 1.5
332 20 1.5
333 20 1.5
334 20 1.5
335 20 1.5
336 20 1.5
337 20 1.5
338 20 1.5
339 20 1.5
340 20 1.5
341 20 1.5
342 20 1.5
343 20 1.5
344 20 1.5
345 20 1.5
346 20 1.5
347 20 1.5
348 20 1.5
349 20 1.5
350 20 1.5
351 20 1.5
352 20 0.0
353 20 0.0
354 20 0.0
355 20 0.0
356 20 0.0
357 20 0.0
358 20 0.0
359 20 0.0
360 20 0.0
361 20 0.0
362 20 0.0
363 20 0.0
364 20 0.0
365 20 0.0
366 20 0.0
367 20 0.0
368 20 0.0
369 20 0.0
370 20 0.0
371 20 0.0
372 20 0.0
373 20 0.0
374 20 0.0
375 20 0.0
376 20 0.0
377 20 0.0
378 20 0.0
379 20 0.0
380 20 0.0
381 20 0.0
382 20 0.0
383 20 0.0
0 21 0.0
1 21 0.0
2 21 0.0
3 21 0.0
4 21 0.0
5 21 0.0
6 21 0.0
7 21 0.0
8 21 0.0
9 21 1.5
10 21 0.0
11 21 0.0
12 21 0.0
13 21 0.0
14 21 0.0
15 21 0.0
16 21 0.0
17 21 0.0
18 21 0.0
19 21 1.5
20 21 1.5
21 21 1.5
22 21 1.5
23 21 1.5
24 21 1.5
25 21 1.5
26 21 1.5
27 21 1.5
28 21 1.5
29 21 1.5
30 21 1.5
31 21 1.5
32 21 1.5
33 21 1.5
34 21 1.5
35 21 1.5
36 21 1.5
37 21 1.5
38 21 1.5
39 21 1.5
40 21 1.5
41 21 1.5
42 21 1.5
43 21 1.5
44 21 1.5
45 21 1.5
46 21 1.5
47 21 0.0
48 21 0.0
49 21 0.0
50 21 0.0
51 21 0.0
52 21 0.0
53 21 0.0
54 21 0.0
55 21 0.0
56 21 0.0
57 21 0.0
58 21 0.0
59 21 0.0
60 21 0.0
61 21 0.0
62 21 0.0
63 21 0.0
64 21 0.0
65 21 0.0
66 21 0.0
67 21 0.0
68 21 0.0
69 21 0.0
70 21 0.0
71 21 0.0
72 21 0.0
73 21 0.0
74 21 0.0
75 21 0.0
76 21 0.0
77 21 0.0
78 21 0.0
79 21 0.0
80 21 0.0
81 21 0.0
82 21 0.0
83 21 0.0
84 21 0.0
85 21 0.0
86 21 0.0
87 21 0.0
88 21 0.0
89 21 0.0
90 21 0.0
91 21 0.0
92 21 0.0
93 21 0.0
94 21 0.0
95 21 0.0
96 21 0.0
97 21 0.0
98 21 0.0
99 21 0.0
100 21 0.0
101 21 0.0
102 21 0.0
103 21 0.0
104 21 0.0
105 21 0.0
106 21 0.0
107 21 0.0
108 21 0.0
109 21 0.0
110 21 0.0
111 21 0.0
112 21 0.0
113 21 0.0
114 21 0.0
115 21 0.0
116 21 0.0
117 21 0.0
118 21 0.0
119 21 0.0
120 21 0.0
121 21 0.0
122 21 0.0
123 21 0.0
124 21 0.0
125 21 0.0
126 21 0.0
127 21 0.0
128 21 0.0
129 21 0.0
130 21 0.0
131 21 0.0
132 21 0.0
133 21 0.0
134 21 0.0
135 21 0.0
136 21 0.0
137 21 0.0
138 21 0.0
139 21 0.0
140 21 0.0
141 21 0.0
142 21 0.0
143 21 0.0
144 21 0.0
145 21 0.0
146 21 0.0
147 21 0.0
148 21 0.0
149 21 0.0
150 21 0.0
151 21 0.0
152 21 0.0
153 21 0.0
154 21 0.0
155 21 0.0
156 21 0.0
157 21 0.0
158 21 0.0
159 21 0.0
160 21 0.0
161 21 0.0
162 21 0.0
163 21 0.0
164 21 0.0
165 21 0.0
166 21 0.0
167 21 0.0
168 21 0.0
169 21 0.0
170 21 0.0
171 21 0.0
172 21 0.0
173 21 0.0
174 21 0.0
175 21 0.0
176 21 0.0
177 21 0.0
178 21 0.0
179 21 0.0
180 21 0.0
181 21 0.0
182 21 0.0
183 21 0.0
184 21 0.0
185 21 0.0
186 21 0.0
187 21 0.0
188 21 0.0
189 21 0.0
190 21 0.0
191 21 0.0
192 21 0.0
193 21 0.0
194 21 0.0
195 21 0.0
196 21 0.0
197 21 0.0
198 21 0.0
199 21 0.0
200 21 0.0
201 21 0.0
202 21 0.0
203 21 0.0
204 21 0.0
205 21 0.0
206 21 0.0
207 21 0.0
208 21 0.0
209 21 0.0
210 21 0.0
211 21 0.0
212 21 0.0
213 21 0.0
214 21 0.0
215 21 0.0
216 21 0.0
217 21 0.0
218 21 0.0
219 21 0.0
220 21 0.0
221 21 0.0
222 21 0.0
223 21 0.0
224 21 0.0
225 21 0.0
226 21 0.0
227 21 0.0
228 21 0.0
229 21 0.0
230 21 0.0
231 21 0.0
232 21 0.0
233 21 0.0
234 21 0.0
235 21 0.0
236 21 0.0
237 21 0.0
238 21 0.0
239 21 0.0
240 21 0.0
241 21 0.0
242 21 0.0
243 21 0.0
244 21 0.0
245 21 0.0
246 21 0.0
247 21 0.0
248 21 0.0
249 21 0.0
250 21 0.0
251 21 0.0
252 21 0.0
253 21 0.0
254 21 0.0
255 21 0.0
256 21 0.0
257 21 0.0
258 21 0.0
259 21 0.0
260 21 0.0
261 21 0.0
262 21 0.0
263 21 1.5
264 21 1.5
265 21 1.5
266 21 1.5
267 21 1.5
268 21 1.5
269 21 1.5
270 21 1.5
271 21 0.0
272 21 0.0
273 21 1.5
274 21 1.5
275 21 1.5
276 21 1.5
277 21 1.5
278 21 1.5
279 21 1.5
280 21 0.0
281 21 0.0
282 21 0.0
283 21 0.0
284 21 0.0
285 21 0.0
286 21 0.0
287 21 0.0
288 21 1.5
289 21 1.5
290 21 1.5

```
291 21 1.5
292 21 1.5
293 21 1.5
294 21 1.5
295 21 1.5
296 21 1.5
297 21 1.5
298 21 1.5
299 21 1.5
300 21 1.5
301 21 1.5
302 21 1.5
303 21 1.5
304 21 1.5
305 21 1.5
306 21 1.5
307 21 1.5
308 21 1.5
309 21 1.5
310 21 1.5
311 21 1.5
312 21 1.5
313 21 1.5
314 21 1.5
315 21 1.5
316 21 1.5
317 21 1.5
318 21 1.5
319 21 1.5
320 21 1.5
321 21 1.5
322 21 1.5
323 21 1.5
324 21 1.5
325 21 1.5
326 21 1.5
327 21 1.5
328 21 1.5
329 21 0.0
330 21 0.0
331 21 0.0
332 21 0.0
333 21 0.0
334 21 0.0
335 21 0.0
336 21 0.0
337 21 0.0
338 21 0.0
339 21 0.0
340 21 0.0
341 21 0.0
342 21 0.0
343 21 0.0
344 21 0.0
345 21 0.0
346 21 0.0
347 21 0.0
348 21 0.0
349 21 0.0
350 21 0.0
351 21 0.0
352 21 0.0
353 21 0.0
354 21 0.0
355 21 0.0
356 21 0.0
357 21 0.0
358 21 0.0
359 21 0.0
360 21 0.0
361 21 0.0
362 21 0.0
363 21 0.0
364 21 0.0
365 21 0.0
366 21 0.0
367 21 0.0
368 21 0.0
369 21 0.0
370 21 0.0
371 21 0.0
372 21 0.0
373 21 0.0
374 21 0.0
375 21 0.0
376 21 0.0
377 21 0.0
378 21 0.0
379 21 0.0
380 21 0.0
381 21 0.0
382 21 0.0
383 21 0.0
0 22 0.0
1 22 0.0
2 22 0.0
3 22 0.0
4 22 0.0
5 22 0.0
6 22 0.0
7 22 0.0
8 22 1.5
9 22 0.0
10 22 0.0
11 22 0.0
12 22 0.0
13 22 0.0
14 22 0.0
15 22 0.0
16 22 0.0
17 22 0.0
18 22 0.0
19 22 0.0
20 22 0.0
21 22 0.0
22 22 0.0
23 22 0.0
24 22 0.0
25 22 0.0
26 22 0.0
27 22 0.0
28 22 0.0
29 22 0.0
30 22 0.0
31 22 0.0
32 22 0.0
33 22 0.0
34 22 0.0
35 22 0.0
36 22 0.0
37 22 0.0
38 22 0.0
39 22 0.0
40 22 0.0
41 22 0.0
42 22 0.0
43 22 0.0
44 22 0.0
45 22 0.0
46 22 0.0
47 22 1.5
48 22 1.5
49 22 1.5
50 22 1.5
51 22 1.5
52 22 1.5
53 22 1.5
54 22 1.5
55 22 1.5
56 22 1.5
57 22 1.5
58 22 1.5
59 22 1.5
60 22 1.5
61 22 1.5
62 22 1.5
63 22 1.5
64 22 1.5
65 22 1.5
66 22 1.5
67 22 1.5
68 22 1.5
69 22 1.5
70 22 1.5
71 22 1.5
72 22 1.5
73 22 1.5
74 22 1.5
75 22 1.5
76 22 1.5
77 22 1.5
78 22 1.5
79 22 1.5
80 22 1.5
81 22 1.5
82 22 1.5
83 22 1.5
84 22 1.5
85 22 1.5
86 22 1.5
87 22 1.5
88 22 1.5
89 22 1.5
90 22 1.5
91 22 1.5
92 22 1.5
93 22 1.5
94 22 1.5
95 22 1.5
96 22 1.5
97 22 1.5
98 22 1.5
99 22 1.5
100 22 1.5
101 22 1.5
102 22 1.5
103 22 1.5
104 22 1.5
105 22 1.5
106 22 1.5
107 22 1.5
108 22 1.5
109 22 1.5
110 22 1.5
111 22 1.5
112 22 1.5
113 22 1.5
114 22 1.5
115 22 1.5
116 22 1.5
117 22 1.5
118 22 1.5
119 22 1.5
120 22 1.5
121 22 1.5
122 22 1.5
123 22 0.0
124 22 0.0
125 22 0.0
126 22 0.0
127 22 0.0
128 22 0.0
129 22 1.5
130 22 1.5
131 22 1.5
132 22 1.5
133 22 1.5
134 22 1.5
135 22 1.5
136 22 1.5
137 22 1.5
138 22 1.5
139 22 0.0
140 22 0.0
141 22 0.0
142 22 0.0
143 22 0.0
144 22 0.0
145 22 0.0
146 22 0.0
147 22 0.0
148 22 0.0
149 22 0.0
150 22 0.0
151 22 0.0
152 22 0.0
153 22 0.0
154 22 0.0
155 22 0.0
156 22 0.0
157 22 0.0
158 22 0.0
159 22 0.0
160 22 0.0
161 22 0.0
162 22 0.0
163 22 0.0
164 22 0.0
165 22 0.0
166 22 0.0
167 22 0.0
168 22 0.0
169 22 0.0
170 22 0.0
171 22 0.0
172 22 0.0
173 22 0.0
174 22 0.0
175 22 0.0
176 22 0.0
177 22 0.0
178 22 0.0
179 22 0.0
180 22 0.0
181 22 0.0
182 22 0.0
183 22 0.0
184 22 0.0
185 22 0.0
186 22 0.0
187 22 0.0
188 22 0.0
189 22 0.0
190 22 0.0
191 22 0.0
192 22 0.0
193 22 0.0
194 22 0.0
195 22 0.0
196 22 0.0
197 22 0.0
198 22 0.0
199 22 0.0
200 22 0.0
201 22 0.0
202 22 0.0
203 22 0.0
204 22 0.0
205 22 0.0
206 22 0.0
207 22 0.0
208 22 0.0
209 22 0.0
210 22 0.0
211 22 0.0
212 22 0.0
213 22 0.0
214 22 0.0
215 22 0.0
216 22 0.0
217 22 0.0
218 22 0.0
219 22 0.0
220 22 0.0
221 22 0.0
222 22 0.0
223 22 0.0
224 22 0.0
225 22 0.0
226 22 0.0
227 22 0.0
228 22 1.5
229 22 1.5
230 22 1.5
231 22 1.5
232 22 1.5
233 22 1.5
234 22 1.5
235 22 1.5
236 22 1.5
237 22 1.5
238 22 1.5
239 22 1.5
240 22 1.5
241 22 1.5
242 22 1.5
243 22 1.5
244 22 1.5
245 22 1.5
246 22 1.5
247 22 1.5
248 22 1.5
249 22 1.5
250 22 1.5
251 22 1.5
252 22 1.5
253 22 1.5
254 22 1.5
255 22 1.5
256 22 1.5
257 22 1.5
258 22 1.5
259 22 1.5
260 22 1.5
261 22 1.5
```

262 22 1.5
263 22 0.0
264 22 0.0
265 22 0.0
266 22 0.0
267 22 0.0
268 22 0.0
269 22 0.0
270 22 0.0
271 22 1.5
272 22 1.5
273 22 0.0
274 22 0.0
275 22 0.0
276 22 0.0
277 22 0.0
278 22 0.0
279 22 0.0
280 22 1.5
281 22 1.5
282 22 1.5
283 22 1.5
284 22 1.5
285 22 1.5
286 22 1.5
287 22 1.5
288 22 0.0
289 22 0.0
290 22 0.0
291 22 0.0
292 22 0.0
293 22 0.0
294 22 0.0
295 22 0.0
296 22 0.0
297 22 0.0
298 22 0.0
299 22 0.0
300 22 0.0
301 22 0.0
302 22 0.0
303 22 0.0
304 22 0.0
305 22 0.0
306 22 0.0
307 22 0.0
308 22 0.0
309 22 0.0
310 22 0.0
311 22 0.0
312 22 0.0
313 22 0.0
314 22 0.0
315 22 0.0
316 22 0.0
317 22 0.0
318 22 0.0
319 22 0.0
320 22 0.0
321 22 0.0
322 22 0.0
323 22 0.0
324 22 0.0
325 22 0.0
326 22 0.0
327 22 0.0
328 22 0.0
329 22 0.0
330 22 0.0
331 22 0.0
332 22 0.0
333 22 0.0
334 22 0.0
335 22 0.0
336 22 0.0
337 22 0.0
338 22 0.0
339 22 0.0
340 22 0.0
341 22 0.0
342 22 0.0
343 22 0.0
344 22 0.0
345 22 0.0
346 22 0.0
347 22 0.0
348 22 0.0
349 22 0.0
350 22 0.0
351 22 0.0
352 22 0.0
353 22 0.0
354 22 0.0
355 22 0.0
356 22 0.0
357 22 0.0
358 22 0.0
359 22 0.0
360 22 0.0
361 22 0.0
362 22 0.0
363 22 0.0
364 22 0.0
365 22 0.0
366 22 0.0
367 22 0.0
368 22 0.0
369 22 0.0
370 22 0.0
371 22 0.0
372 22 0.0
373 22 0.0
374 22 0.0
375 22 0.0
376 22 0.0
377 22 0.0
378 22 0.0
379 22 0.0
380 22 0.0
381 22 0.0
382 22 0.0
383 22 0.0
0 23 0.0
1 23 0.0
2 23 0.0
3 23 0.0
4 23 0.0
5 23 0.0
6 23 0.0
7 23 0.0
8 23 0.0
9 23 0.0
10 23 0.0
11 23 0.0
12 23 0.0
13 23 0.0
14 23 0.0
15 23 0.0
16 23 0.0
17 23 0.0
18 23 0.0
19 23 0.0
20 23 0.0
21 23 0.0
22 23 0.0
23 23 0.0
24 23 0.0
25 23 0.0
26 23 0.0
27 23 0.0
28 23 0.0
29 23 0.0
30 23 0.0
31 23 0.0
32 23 0.0
33 23 0.0
34 23 0.0
35 23 0.0
36 23 0.0
37 23 0.0
38 23 0.0
39 23 0.0
40 23 0.0
41 23 0.0
42 23 0.0
43 23 0.0
44 23 0.0
45 23 0.0
46 23 0.0
47 23 0.0
48 23 0.0
49 23 0.0
50 23 0.0
51 23 0.0
52 23 0.0
53 23 0.0
54 23 0.0
55 23 0.0
56 23 0.0
57 23 0.0
58 23 0.0
59 23 0.0
60 23 0.0
61 23 0.0
62 23 0.0
63 23 0.0
64 23 0.0
65 23 0.0
66 23 0.0
67 23 0.0
68 23 0.0
69 23 0.0
70 23 0.0
71 23 0.0
72 23 0.0
73 23 0.0
74 23 0.0
75 23 0.0
76 23 0.0
77 23 0.0
78 23 0.0
79 23 0.0
80 23 0.0
81 23 0.0
82 23 0.0
83 23 0.0
84 23 0.0
85 23 0.0
86 23 0.0
87 23 0.0
88 23 0.0
89 23 0.0
90 23 0.0
91 23 0.0
92 23 0.0
93 23 0.0
94 23 0.0
95 23 0.0
96 23 0.0
97 23 0.0
98 23 0.0
99 23 0.0
100 23 0.0
101 23 0.0
102 23 0.0
103 23 0.0
104 23 0.0
105 23 0.0
106 23 0.0
107 23 0.0
108 23 0.0
109 23 0.0
110 23 0.0
111 23 0.0
112 23 0.0
113 23 0.0
114 23 0.0
115 23 0.0
116 23 0.0
117 23 0.0
118 23 0.0
119 23 0.0
120 23 0.0
121 23 0.0
122 23 0.0
123 23 1.5
124 23 1.5
125 23 1.5
126 23 1.5
127 23 1.5
128 23 1.5
129 23 0.0
130 23 0.0
131 23 0.0
132 23 0.0
133 23 0.0
134 23 0.0
135 23 0.0
136 23 0.0
137 23 0.0
138 23 0.0
139 23 1.5
140 23 1.5
141 23 1.5
142 23 1.5
143 23 1.5
144 23 1.5
145 23 1.5
146 23 1.5
147 23 1.5
148 23 1.5
149 23 1.5
150 23 1.5
151 23 1.5
152 23 1.5
153 23 1.5
154 23 1.5
155 23 1.5
156 23 1.5
157 23 1.5
158 23 1.5
159 23 1.5
160 23 1.5
161 23 1.5
162 23 1.5
163 23 1.5
164 23 1.5
165 23 1.5
166 23 1.5
167 23 1.5
168 23 1.5
169 23 1.5
170 23 1.5
171 23 1.5
172 23 1.5
173 23 1.5
174 23 1.5
175 23 1.5
176 23 1.5
177 23 1.5
178 23 1.5
179 23 1.5
180 23 1.5
181 23 1.5
182 23 1.5
183 23 1.5
184 23 1.5
185 23 1.5
186 23 1.5
187 23 1.5
188 23 1.5
189 23 1.5
190 23 1.5
191 23 1.5
192 23 1.5
193 23 1.5
194 23 1.5
195 23 1.5
196 23 1.5
197 23 1.5
198 23 1.5
199 23 1.5
200 23 1.5
201 23 1.5
202 23 1.5
203 23 1.5
204 23 1.5
205 23 1.5
206 23 1.5
207 23 1.5
208 23 1.5
209 23 1.5
210 23 1.5
211 23 1.5
212 23 1.5
213 23 1.5
214 23 1.5
215 23 1.5
216 23 1.5
217 23 1.5
218 23 1.5
219 23 1.5
220 23 1.5
221 23 1.5
222 23 1.5
223 23 1.5
224 23 1.5
225 23 1.5
226 23 1.5
227 23 1.5
228 23 0.0
229 23 0.0
230 23 0.0
231 23 0.0
232 23 0.0

233 23 0.0
234 23 0.0
235 23 0.0
236 23 0.0
237 23 0.0
238 23 0.0
239 23 0.0
240 23 0.0
241 23 0.0
242 23 0.0
243 23 0.0
244 23 0.0
245 23 0.0
246 23 0.0
247 23 0.0
248 23 0.0
249 23 0.0
250 23 0.0
251 23 0.0
252 23 0.0
253 23 0.0
254 23 0.0
255 23 0.0
256 23 0.0
257 23 0.0
258 23 0.0
259 23 0.0
260 23 0.0
261 23 0.0
262 23 0.0
263 23 0.0
264 23 0.0
265 23 0.0
266 23 0.0
267 23 0.0
268 23 0.0
269 23 0.0
270 23 0.0
271 23 0.0
272 23 0.0
273 23 0.0
274 23 0.0
275 23 0.0
276 23 0.0
277 23 0.0
278 23 0.0
279 23 0.0
280 23 0.0
281 23 0.0
282 23 0.0
283 23 0.0
284 23 0.0
285 23 0.0
286 23 0.0
287 23 0.0
288 23 0.0
289 23 0.0
290 23 0.0
291 23 0.0
292 23 0.0
293 23 0.0
294 23 0.0
295 23 0.0
296 23 0.0
297 23 0.0
298 23 0.0
299 23 0.0
300 23 0.0
301 23 0.0
302 23 0.0
303 23 0.0
304 23 0.0
305 23 0.0
306 23 0.0
307 23 0.0
308 23 0.0
309 23 0.0
310 23 0.0
311 23 0.0
312 23 0.0
313 23 0.0
314 23 0.0
315 23 0.0
316 23 0.0
317 23 0.0
318 23 0.0
319 23 0.0
320 23 0.0
321 23 0.0
322 23 0.0
323 23 0.0
324 23 0.0
325 23 0.0
326 23 0.0
327 23 0.0
328 23 0.0
329 23 0.0
330 23 0.0
331 23 0.0
332 23 0.0
333 23 0.0
334 23 0.0
335 23 0.0
336 23 0.0
337 23 0.0
338 23 0.0
339 23 0.0
340 23 0.0
341 23 0.0
342 23 0.0
343 23 0.0
344 23 0.0
345 23 0.0
346 23 0.0
347 23 0.0
348 23 0.0
349 23 0.0
350 23 0.0
351 23 0.0
352 23 0.0
353 23 0.0
354 23 0.0
355 23 0.0
356 23 0.0
357 23 0.0
358 23 0.0
359 23 0.0
360 23 0.0
361 23 0.0
362 23 0.0
363 23 0.0
364 23 0.0
365 23 0.0
366 23 0.0
367 23 0.0
368 23 0.0
369 23 0.0
370 23 0.0
371 23 0.0
372 23 0.0
373 23 0.0
374 23 0.0
375 23 0.0
376 23 0.0
377 23 0.0
378 23 0.0
379 23 0.0
380 23 0.0
381 23 0.0
382 23 0.0
383 23 0.0
0 24 0.0
1 24 0.0
2 24 0.0
3 24 0.0
4 24 0.0
5 24 0.0
6 24 0.0
7 24 0.0
8 24 0.0
9 24 0.0
10 24 0.0
11 24 0.0
12 24 0.0
13 24 0.0
14 24 0.0
15 24 0.0
16 24 0.0
17 24 0.0
18 24 0.0
19 24 0.0
20 24 0.0
21 24 0.0
22 24 0.0
23 24 0.0
24 24 0.0
25 24 0.0
26 24 0.0
27 24 0.0
28 24 0.0
29 24 0.0
30 24 0.0
31 24 0.0
32 24 0.0
33 24 0.0
34 24 0.0
35 24 0.0
36 24 0.0
37 24 0.0
38 24 0.0
39 24 0.0
40 24 0.0
41 24 0.0
42 24 0.0
43 24 0.0
44 24 0.0
45 24 0.0
46 24 0.0
47 24 0.0
48 24 0.0
49 24 0.0
50 24 0.0
51 24 0.0
52 24 0.0
53 24 0.0
54 24 0.0
55 24 0.0
56 24 0.0
57 24 0.0
58 24 0.0
59 24 0.0
60 24 0.0
61 24 0.0
62 24 0.0
63 24 0.0
64 24 0.0
65 24 0.0
66 24 0.0
67 24 0.0
68 24 0.0
69 24 0.0
70 24 0.0
71 24 0.0
72 24 0.0
73 24 0.0
74 24 0.0
75 24 0.0
76 24 0.0
77 24 0.0
78 24 0.0
79 24 0.0
80 24 0.0
81 24 0.0
82 24 0.0
83 24 0.0
84 24 0.0
85 24 0.0
86 24 0.0
87 24 0.0
88 24 0.0
89 24 0.0
90 24 0.0
91 24 0.0
92 24 0.0
93 24 0.0
94 24 0.0
95 24 0.0
96 24 0.0
97 24 0.0
98 24 0.0
99 24 0.0
100 24 0.0
101 24 0.0
102 24 0.0
103 24 0.0
104 24 0.0
105 24 0.0
106 24 0.0
107 24 0.0
108 24 0.0
109 24 0.0
110 24 0.0
111 24 0.0
112 24 0.0
113 24 0.0
114 24 0.0
115 24 0.0
116 24 0.0
117 24 0.0
118 24 0.0
119 24 0.0
120 24 0.0
121 24 0.0
122 24 0.0
123 24 0.0
124 24 0.0
125 24 0.0
126 24 0.0
127 24 0.0
128 24 0.0
129 24 0.0
130 24 0.0
131 24 0.0
132 24 0.0
133 24 0.0
134 24 0.0
135 24 0.0
136 24 0.0
137 24 0.0
138 24 0.0
139 24 0.0
140 24 0.0
141 24 0.0
142 24 0.0
143 24 0.0
144 24 0.0
145 24 0.0
146 24 0.0
147 24 0.0
148 24 0.0
149 24 0.0
150 24 0.0
151 24 0.0
152 24 0.0
153 24 0.0
154 24 0.0
155 24 0.0
156 24 0.0
157 24 0.0
158 24 0.0
159 24 0.0
160 24 0.0
161 24 0.0
162 24 0.0
163 24 0.0
164 24 0.0
165 24 0.0
166 24 0.0
167 24 0.0
168 24 0.0
169 24 0.0
170 24 0.0
171 24 0.0
172 24 0.0
173 24 0.0
174 24 0.0
175 24 0.0
176 24 0.0
177 24 0.0
178 24 0.0
179 24 0.0
180 24 0.0
181 24 0.0
182 24 0.0
183 24 0.0
184 24 0.0
185 24 0.0
186 24 0.0
187 24 0.0
188 24 0.0
189 24 0.0
190 24 0.0
191 24 0.0
192 24 0.0
193 24 0.0
194 24 0.0
195 24 0.0
196 24 0.0
197 24 0.0
198 24 0.0
199 24 0.0
200 24 0.0
201 24 0.0
202 24 0.0
203 24 0.0

204 24 0.0
205 24 0.0
206 24 0.0
207 24 0.0
208 24 0.0
209 24 0.0
210 24 0.0
211 24 0.0
212 24 0.0
213 24 0.0
214 24 0.0
215 24 0.0
216 24 0.0
217 24 0.0
218 24 0.0
219 24 0.0
220 24 0.0
221 24 0.0
222 24 0.0
223 24 0.0
224 24 0.0
225 24 0.0
226 24 0.0
227 24 0.0
228 24 0.0
229 24 0.0
230 24 0.0
231 24 0.0
232 24 0.0
233 24 0.0
234 24 0.0
235 24 0.0
236 24 0.0
237 24 0.0
238 24 0.0
239 24 0.0
240 24 0.0
241 24 0.0
242 24 0.0
243 24 0.0
244 24 0.0
245 24 0.0
246 24 0.0
247 24 0.0
248 24 0.0
249 24 0.0
250 24 0.0
251 24 0.0
252 24 0.0
253 24 0.0
254 24 0.0
255 24 0.0
256 24 0.0
257 24 0.0
258 24 0.0
259 24 0.0
260 24 0.0
261 24 0.0
262 24 0.0
263 24 0.0
264 24 0.0
265 24 0.0
266 24 0.0
267 24 0.0
268 24 0.0
269 24 0.0
270 24 0.0
271 24 0.0
272 24 0.0
273 24 0.0
274 24 0.0
275 24 0.0
276 24 0.0
277 24 0.0
278 24 0.0
279 24 0.0
280 24 0.0
281 24 0.0
282 24 0.0
283 24 0.0
284 24 0.0
285 24 0.0
286 24 0.0
287 24 0.0
288 24 0.0
289 24 0.0
290 24 0.0
291 24 0.0
292 24 0.0
293 24 0.0
294 24 0.0
295 24 0.0
296 24 0.0
297 24 0.0
298 24 0.0
299 24 0.0
300 24 0.0
301 24 0.0
302 24 0.0
303 24 0.0
304 24 0.0
305 24 0.0
306 24 0.0
307 24 0.0
308 24 0.0
309 24 0.0
310 24 0.0
311 24 0.0
312 24 0.0
313 24 0.0
314 24 0.0
315 24 0.0
316 24 0.0
317 24 0.0
318 24 0.0
319 24 0.0
320 24 0.0
321 24 0.0
322 24 0.0
323 24 0.0
324 24 0.0
325 24 0.0
326 24 0.0
327 24 0.0
328 24 0.0
329 24 0.0
330 24 0.0
331 24 0.0
332 24 0.0
333 24 0.0
334 24 0.0
335 24 0.0
336 24 0.0
337 24 0.0
338 24 0.0
339 24 0.0
340 24 0.0
341 24 0.0
342 24 0.0
343 24 0.0
344 24 0.0
345 24 0.0
346 24 0.0
347 24 0.0
348 24 0.0
349 24 0.0
350 24 0.0
351 24 0.0
352 24 0.0
353 24 0.0
354 24 0.0
355 24 0.0
356 24 0.0
357 24 0.0
358 24 0.0
359 24 0.0
360 24 0.0
361 24 0.0
362 24 0.0
363 24 0.0
364 24 0.0
365 24 0.0
366 24 0.0
367 24 0.0
368 24 0.0
369 24 0.0
370 24 0.0
371 24 0.0
372 24 0.0
373 24 0.0
374 24 0.0
375 24 0.0
376 24 0.0
377 24 0.0
378 24 0.0
379 24 0.0
380 24 0.0
381 24 0.0
382 24 0.0
383 24 0.0
0 25 0.0
1 25 0.0
2 25 0.0
3 25 0.0
4 25 0.0
5 25 0.0
6 25 0.0
7 25 0.0
8 25 0.0
9 25 0.0
10 25 0.0
11 25 0.0
12 25 0.0
13 25 0.0
14 25 0.0
15 25 0.0
16 25 0.0
17 25 0.0
18 25 0.0
19 25 0.0
20 25 0.0
21 25 0.0
22 25 0.0
23 25 0.0
24 25 0.0
25 25 0.0
26 25 0.0
27 25 0.0
28 25 0.0
29 25 0.0
30 25 0.0
31 25 0.0
32 25 0.0
33 25 0.0
34 25 0.0
35 25 0.0
36 25 0.0
37 25 0.0
38 25 0.0
39 25 0.0
40 25 0.0
41 25 0.0
42 25 0.0
43 25 0.0
44 25 0.0
45 25 0.0
46 25 0.0
47 25 0.0
48 25 0.0
49 25 0.0
50 25 0.0
51 25 0.0
52 25 0.0
53 25 0.0
54 25 0.0
55 25 0.0
56 25 0.0
57 25 0.0
58 25 0.0
59 25 0.0
60 25 0.0
61 25 0.0
62 25 0.0
63 25 0.0
64 25 0.0
65 25 0.0
66 25 0.0
67 25 0.0
68 25 0.0
69 25 0.0
70 25 0.0
71 25 0.0
72 25 0.0
73 25 0.0
74 25 0.0
75 25 0.0
76 25 0.0
77 25 0.0
78 25 0.0
79 25 0.0
80 25 0.0
81 25 0.0
82 25 0.0
83 25 0.0
84 25 0.0
85 25 0.0
86 25 0.0
87 25 0.0
88 25 0.0
89 25 0.0
90 25 0.0
91 25 0.0
92 25 0.0
93 25 0.0
94 25 0.0
95 25 0.0
96 25 0.0
97 25 0.0
98 25 0.0
99 25 0.0
100 25 0.0
101 25 0.0
102 25 0.0
103 25 0.0
104 25 0.0
105 25 0.0
106 25 0.0
107 25 0.0
108 25 0.0
109 25 0.0
110 25 0.0
111 25 0.0
112 25 0.0
113 25 0.0
114 25 0.0
115 25 0.0
116 25 0.0
117 25 0.0
118 25 0.0
119 25 0.0
120 25 0.0
121 25 0.0
122 25 0.0
123 25 0.0
124 25 0.0
125 25 0.0
126 25 0.0
127 25 0.0
128 25 0.0
129 25 0.0
130 25 0.0
131 25 0.0
132 25 0.0
133 25 0.0
134 25 0.0
135 25 0.0
136 25 0.0
137 25 0.0
138 25 0.0
139 25 0.0
140 25 0.0
141 25 0.0
142 25 0.0
143 25 0.0
144 25 0.0
145 25 0.0
146 25 0.0
147 25 0.0
148 25 0.0
149 25 0.0
150 25 0.0
151 25 0.0
152 25 0.0
153 25 0.0
154 25 0.0
155 25 0.0
156 25 0.0
157 25 0.0
158 25 0.0
159 25 0.0
160 25 0.0
161 25 0.0
162 25 0.0
163 25 0.0
164 25 0.0
165 25 0.0
166 25 0.0
167 25 0.0
168 25 0.0
169 25 0.0
170 25 0.0
171 25 0.0
172 25 0.0
173 25 0.0
174 25 0.0

175 25 0.0
176 25 0.0
177 25 0.0
178 25 0.0
179 25 0.0
180 25 0.0
181 25 0.0
182 25 0.0
183 25 0.0
184 25 0.0
185 25 0.0
186 25 0.0
187 25 0.0
188 25 0.0
189 25 0.0
190 25 0.0
191 25 0.0
192 25 0.0
193 25 0.0
194 25 0.0
195 25 0.0
196 25 0.0
197 25 0.0
198 25 0.0
199 25 0.0
200 25 0.0
201 25 0.0
202 25 0.0
203 25 0.0
204 25 0.0
205 25 0.0
206 25 0.0
207 25 0.0
208 25 0.0
209 25 0.0
210 25 0.0
211 25 0.0
212 25 0.0
213 25 0.0
214 25 0.0
215 25 0.0
216 25 0.0
217 25 0.0
218 25 0.0
219 25 0.0
220 25 0.0
221 25 0.0
222 25 0.0
223 25 0.0
224 25 0.0
225 25 0.0
226 25 0.0
227 25 0.0
228 25 0.0
229 25 0.0
230 25 0.0
231 25 0.0
232 25 0.0
233 25 0.0
234 25 0.0
235 25 0.0
236 25 0.0
237 25 0.0
238 25 0.0
239 25 0.0
240 25 0.0
241 25 0.0
242 25 0.0
243 25 0.0
244 25 0.0
245 25 0.0
246 25 0.0
247 25 0.0
248 25 0.0
249 25 0.0
250 25 0.0
251 25 0.0
252 25 0.0
253 25 0.0
254 25 0.0
255 25 0.0
256 25 0.0
257 25 0.0
258 25 0.0
259 25 0.0
260 25 0.0
261 25 0.0
262 25 0.0
263 25 0.0
264 25 0.0
265 25 0.0
266 25 0.0
267 25 0.0
268 25 0.0
269 25 0.0
270 25 0.0
271 25 0.0
272 25 0.0
273 25 0.0
274 25 0.0
275 25 0.0
276 25 0.0
277 25 0.0
278 25 0.0
279 25 0.0
280 25 0.0
281 25 0.0
282 25 0.0
283 25 0.0
284 25 0.0
285 25 0.0
286 25 0.0
287 25 0.0
288 25 0.0
289 25 0.0
290 25 0.0
291 25 0.0
292 25 0.0
293 25 0.0
294 25 0.0
295 25 0.0
296 25 0.0
297 25 0.0
298 25 0.0
299 25 0.0
300 25 0.0
301 25 0.0
302 25 0.0
303 25 0.0
304 25 0.0
305 25 0.0
306 25 0.0
307 25 0.0
308 25 0.0
309 25 0.0
310 25 0.0
311 25 0.0
312 25 0.0
313 25 0.0
314 25 0.0
315 25 0.0
316 25 0.0
317 25 0.0
318 25 0.0
319 25 0.0
320 25 0.0
321 25 0.0
322 25 0.0
323 25 0.0
324 25 0.0
325 25 0.0
326 25 0.0
327 25 0.0
328 25 0.0
329 25 0.0
330 25 0.0
331 25 0.0
332 25 0.0
333 25 0.0
334 25 0.0
335 25 0.0
336 25 0.0
337 25 0.0
338 25 0.0
339 25 0.0
340 25 0.0
341 25 0.0
342 25 0.0
343 25 0.0
344 25 0.0
345 25 0.0
346 25 0.0
347 25 0.0
348 25 0.0
349 25 0.0
350 25 0.0
351 25 0.0
352 25 0.0
353 25 0.0
354 25 0.0
355 25 0.0
356 25 0.0
357 25 0.0
358 25 0.0
359 25 0.0
360 25 0.0
361 25 0.0
362 25 0.0
363 25 0.0
364 25 0.0
365 25 0.0
366 25 0.0
367 25 0.0
368 25 0.0
369 25 0.0
370 25 0.0
371 25 0.0
372 25 0.0
373 25 0.0
374 25 0.0
375 25 0.0
376 25 0.0
377 25 0.0
378 25 0.0
379 25 0.0
380 25 0.0
381 25 0.0
382 25 0.0
383 25 0.0