"IMPLEMENTACIÓN, VALIDACIÓN Y VERIFICACIÓN DE LOS ALGORITMOS DE CONVOLUCIÓN Y TRANSFORMADA RÁPIDA DE FOURIER EN DOS DIMENSIONES SOBRE UN CLÚSTER DE COMPUTADORES MAC"

DIEGO ERLEY DURÁN DURÁN

MANUEL ARNULFO PLATA ESPINOSA

ESCUELA DE INGENIERÍAS
ELÉCTRICA, ELECTRÓNICA
Y DE TELECOMUNICACIONES

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2009

UNIVERSIDAD INDUSTRIAL DE SANTANDER
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
Perfecta Combinación entre Energía e Intelecto

"IMPLEMENTACIÓN, VALIDACIÓN Y VERIFICACIÓN DE LOS ALGORITMOS DE CONVOLUCIÓN Y TRANSFORMADA RÁPIDA DE FOURIER EN DOS DIMENSIONES SOBRE UN CLÚSTER DE COMPUTADORES MAC"

DIEGO ERLEY DURÁN DURÁN

MANUEL ARNULFO PLATA ESPINOSA

Este trabajo se presenta como requisito para optar al título de Ingeniero Electrónico

Directora

M.SC. ANA BEATRÍZ RAMÍREZ SILVA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2009

AGRADECIMIENTOS

Muchas gracias a todos los que nos han apoyado en todo nuestro aprendizaje y en cada situación de la vida, que nos represento este ciclo de sabiduría y preparación, no solo para ser excelentes ingenieros, también para ser personas con grandes cualidades interpersonales. Fue una gran experiencia el haber estudiado en esta gran universidad y gran comunidad, que para nosotros fue como nuestro segundo hogar y siempre recordaremos lo bueno que hemos aprendido en esta institución.

Gracias a nuestra familia que siempre nos ha apoyado, no solo económicamente sino también con gran entusiasmo y comprensión en lo que fue un ciclo de aprendizaje en nuestra vida. Siempre hemos dado lo mejor de nosotros, a veces encontramos obstáculos debido a muchas cosas en el entorno, pero las hemos superado y esta es la prueba de que somos capaces de todo, solo nuestra imaginación es la que nos limita.

Gracias a las personas que nos enseñaron, que siempre nos exigieron y de ellos aprendimos lo grande que es el mundo y que no importa las dificultades en el camino del aprendizaje, solo falta la actitud, estar dispuestos a abrir la mente al mundo y al nuevo conocimiento que constantemente se renueva y requiere de más capacitación y de mayor comprensión.

TABLA DE CONTENIDO

Pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

pág.

GLOSARIO

API: interfaz de programación de aplicaciones del inglés Application Programming Interface es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software, es una interfaz de comunicación entre componentes software.

APPLE COMPUTER: empresa encargada de desarrollar el Hardware y Software del sistema operativo Macintosh.

BIT REVERSAL: procedimiento utilizado con el fin de reacomodar la secuencias discretas para poder obtener resultados en orden correcto al implementar la FFT.

CLUSTER: conjunto o conglomerado de computadores que se interconectan entre sí mediante una red y software especializado para que se comporten como si fuesen una solo maquina trabajando en un proceso.

CONJUGADO: en números complejos dos números son conjugados cuando tienen igual magnitud pero fase opuesta (una negativa de la otra).

CONVOLUCIÓN: la convolución de f y g se denota $f * g$. Se define como la integral del producto de ambas funciones después de que una sea invertida y desplazada una distancia τ.

$$(f * g)(t) = \int f(\tau) g(t - \tau) d\tau$$

La convolución de dos funciones invierte una función y la desplaza a medida que la va multiplicando con la otra función, y en todo este proceso se suman los resultados teniendo en definitiva una integral, así pues vemos que la convolución es un tipo de suma de multiplicación de funciones, se puede decir que es un promedio móvil.

Para las funciones discretas se puede usar una forma discreta de la convolución:

$$f[m] * g[m] = \sum_n f[n]g[m-n]$$

Estas funciones pueden también ser bidimensionales, como las Matrices.

CONVOLUCIÓN CÍCLICA: cuando se tienen señales discretas en el tiempo con periodos idénticos, la operación de convolución entre ellas se denomina convolución cíclica, estas pueden ser funciones bidimensionales.

CONVOLUCIÓN CÍCLICA 2D: convolución cíclica bidimensional.

COMPILADOR: un compilador es un programa que traduce un lenguaje de programación a otro, generando un código valido para otros lenguajes de programación o para que lo interprete la maquina. Generalmente se traduce a lenguaje de máquina.

COMUNICADOR: en la interfaz de paso de mensajes (MPI) se define un comunicador como una colección de procesos, los cuales pueden enviar mensajes entre ellos.

DECIMACIÓN: significa descomponer. Decimar una secuencia es descomponerla en pequeñas subsecuencias.

DFT: transformada discreta de Fourier, del ingles Discrete Fourier Transform.

DIF: decimación en frecuencia del ingles Decimation in Frequency, una forma de implementar el algoritmo de la FFT.

DIT: decimación en Tiempo del ingles Decimation in Time, una forma de implementar el algoritmo de la FFT.

ETHERNET: ethernet es un estándar de redes de computadoras de área local.

FFT: transformada rápida de Fourier del ingles Fast Fourier Transform.

FFT 2D: transformada rápida de Fourier en dos dimensiones.

GCC: colección de compiladores de software libre, es un conjunto de compiladores creados por el proyecto GNU.

IFFT: transformada inversa de Fourier del ingles Inverse Fast Fourier transform.

IFFT 2D: transformada inversa de Fourier en dos dimensiones.

LAN: una red de área local, red local o LAN (del inglés Local Area Network) es la interconexión de varios ordenadores y periféricos.

LENGUAJE C: lenguaje de programación multiplataforma caracterizado por la eficiencia de su código, generalmente es usado para la creación de software, aplicaciones y librerías.

MAC MINI: computador personal de escritorio construido por la empresa Apple viene preinstalado con el sistema operativo MAC OS X 10.5

MAC OS X 10.5: conocido como “Leopard” es el sistema operativo que usa el Mac Mini.

Mbps: rata de transmisión de datos medida en Mega Bits por segundo.

MIMD: múltiples instrucciones múltiples datos, es una técnica de paralelización de algoritmos que consiste en partir las instrucciones y también los datos.

MISD: múltiples instrucciones datos sencillos, es una técnica de paralelización de algoritmos que consiste en partir las instrucciones para aplicársela a los mismos datos. *(*Multiple Instruction, Single Data*).*

MPI: MPI es un protocolo estándar para programar aplicaciones que ejecutan sus procesos en paralelo y permitir que diferentes procesadores se comuniquen por medio de mensajes para administrar los procesos que se llevan a cabo en paralelo.

MPICC: comando para compilar el código creado en lenguaje C, utilizando el estándar MPI y la librería de Open MPI.

OPEM MPI: Open MPI es una implementación de código abierto de los estándares de MPI.

POOCH: Pooch (Parallel Operation and Control Heuristic application). Software que provee interfaz de usuario, para la distribución e inicialización de una aplicación paralela numéricamente intensiva en una red de computadores Macintosh.

RADIX 2: un tipo de base para implementar la FFT que limita a que la longitud N de la señal sea potencia de 2.

RANK: en MPI a cada proceso se le asigna una variable que se denomina rank, la cual identifica cada proceso en el rango de 0 a p-1, donde p es el número total de procesos.

RAM: la memoria de acceso aleatorio, en inglés Random Access Memory cuyo acrónimo es RAM, es la memoria donde el procesador recibe las instrucciones y guarda los resultados.

SIMD: instrucciones sencillas múltiples datos, es una técnica de paralelización de algoritmos que consiste en partir los datos para aplicarle las mismas instrucciones (Single Instruction, Multiple Data).

TERMINAL: consola del Mac mini para ejecutar los comandos, análogo al intérprete de comandos de Windows.

TWIDDLE FACTORS: los Coeficientes W de la FFT radix 2 son llamados los factores "Twiddles" los cuales representan la fase por:

$$W = e^{-j2\pi/N}$$

Estos factores son función de la longitud N.

XCODE: es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con Mac OS X. Xcode incluye la

colección de compiladores del proyecto GNU (GCC), y puede compilar código C.

RESUMEN

TITULO: IMPLEMENTACIÓN, VALIDACIÓN Y VERIFICACIÓN DE LOS ALGORITMOS DE CONVOLUCIÓN Y TRANSFORMADA RÁPIDA DE FOURIER EN DOS DIMENSIONES SOBRE UN CLÚSTER DE COMPUTADORES MAC. *

AUTORES: DIEGO ERLEY DURÁN DURÁN y MANUEL ARNULFO PLATA ESPINOSA. **

PALABRAS CLAVES: Transformada rápida de Fourier bidimensional, Convolución cíclica bidimensional, Interfaz de paso de mensajes, Macintosh, clusters, radix 2, MISD, SIMD.

DESCRIPCIÓN

Hoy en día el procesamiento digital de datos, por ejemplo de imágenes, requiere obtener resultados en el menor tiempo posible. Cada vez la información captada por los dispositivos sensores es más grande y obtenida en tiempo real, lo cual requiere tener un procesamiento más sofisticado con *hardware* y *software* con capacidad de ejecutar este tipo de procesos adecuadamente.

Como alternativa a esto surgieron los llamados "Clusters" conjuntos o conglomerados de computadoras, construídos mediante la utilización de componentes de *hardware* comunes y que se comportan como si fuesen una única computadora capaz de ejecutar grandes procesos. Con esto se evita el uso de supercomputadoras las cuales no aprovechan la capacidad de paralelización de los algoritmos y no se tienen al alcance en muchas ocasiones.
En este trabajo se presenta el procesamiento de la convolución en paralelo sobre grandes matrices de números complejos, la cual se implementa mediante el algoritmo de la transformada rápida de Fourier bidimensional.
Por otra parte se presenta una descripción teórica de los fundamentos en los cuales se basó la implementación de los algoritmos, por citar algunos tales como la trasformada rápida de Fourier en dos dimensiones, la transformada inversa de Fourier en dos dimensiones, convolución cíclica en dos dimensiones, construcción del cluster y descripción del *hardware* y *software* sobre el cual se implementaron los algoritmos.

La implementación del algoritmo paralelo se realiza sobre los computadores Mac mini del laboratorio de cómputo de la escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander. Este trabajo permite obtener una comparación del tiempo de procesamiento del algoritmo en forma paralela con el mismo algoritmo en forma secuencial, e incluso con el mismo algoritmo implementado en diferentes tipos de formas paralelas (MISD y SIMD).

---

* Proyecto de grado.

** Facultad de Ingenierías Físico-Mecánicas, Escuelas de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. M.Sc. Ana Beatriz Ramírez Silva.

ABSTRACT

TITLE: IMPLEMENTATION, VALIDATION AND VERIFICATION OF ALGORITHMS OF CONVOLUTION AND FAST FOURIER TRANSFORM INTO TWO DIMENSIONS ON A MAC COMPUTER CLUSTER. *

AUTHORS: DIEGO ERLEY DURÁN DURÁN and MANUEL ARNULFO PLATA ESPINOSA. **

Keywords: Two-dimensional Fast Fourier Transform, Two-dimensional cyclical convolution, message passing interface, Macintosh, clusters, radix 2, MISD, SIMD.

DESCRIPTION

Today, the digital data processing, for example, of images, requires to obtain results as fast as possible. The advance of technology, in terms of data collection, grows by leaps and bounds; the information picked up by the sensor devices is larger and obtained in real time, which requires to have a more sophisticated processing with hardware and software capable of running those processes properly.

"Clusters" or conglomerates of computers emerged as an alternative for these processes. They are built using common hardware components which behave as if they were a single computer able to run large processes. This avoids the use of supercomputers which do not exploit the parallel processing of algorithms and they are often not available. This work shows the cyclical convolution processing in parallel on large matrixes of complex numbers, which is implemented by the algorithm of two-dimensional fast Fourier transform, and it makes a comparison with the obtained results for the same algorithm implemented sequentially in this work. On the other hand , it is presented a theoretical description of the basics on which it is based the implementation of algorithms, such as the two-dimensional fast Fourier transform, two-dimensional inverse fast Fourier transform, two-dimensional cyclical convolution, cluster construction , and description of hardware and software on which the algorithms were implemented.

The parallel algorithm implementation is done on Mac mini computers of the computer lab of the school of Electrical, Electronics, and Telecommunications Engineering of the Universidad Industrial de Santander. These computers are connected through an Ethernet network of 100 Mbit / s. This work allows a comparison of the processing time of the algorithm in parallel way with the same algorithm sequentially, and even , with the same algorithm implemented in different types of parallels forms (MISD and SIMD).

---

* Proyecto de grado.

** Facultad de Ingenierías Físico-Mecánicas, Escuelas de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. M.Sc. Ana Beatriz Ramírez Silva.

INTRODUCCIÓN

La transformada rápida de Fourier y la convolución en dos dimensiones son algoritmos ampliamente utilizados en el procesamiento digital de datos, una aplicación principal es la de tratamiento de imágenes, solo por citar uno de tantos ejemplos.
La enorme cantidad de datos requeridos para ciertas aplicaciones limita los tiempos de procesamiento si no se tienen equipos de alto rendimiento disponibles para esto, lo cual es una gran desventaja ya que para aplicaciones de tiempo real estos retardos pueden significar incluso pérdidas económicas muy relevantes.

En este trabajo se construye un Cluster Mac, con equipos pertenecientes a la serie Mac Mini de Apple Inc. Estos equipos son conectados mediante una red Ethernet de 100 Mbits/s y configurados mediante software especializado para que formen un conglomerado que trabaje en conjunto en el procesamiento de matrices de números complejos de gran tamaño. Sobre estos datos se realizarán operaciones frecuentemente usadas en procesamiento de imágenes: la transformada rápida directa de Fourier, la transformada rápida inversa de Fourier y la operación de convolución cíclica que se basa en las dos anteriores usando una multiplicación punto a punto bidimensional.

En la primera parte del trabajo se dan todos los fundamentos o bases teóricas para la construcción de los algoritmos, tales como la FFT 2D, la IFFT 2D y la convolución cíclica 2D. Se presenta una descripción detallada del *hardware* y *software* utilizado para la elaboración de los algoritmos y la construcción del cluster, y se presenta una descripción de la etapa de diseño y construcción del cluster.

Luego se presenta una descripción de los algoritmos y la técnica utilizada para su implementación en el cluster, se obtienen medidas de desempeño y se analizan los resultados obtenidos mediante una comparación entre los diferentes tipos de algoritmos implementados.

Al final se presentan las conclusiones de este trabajo y algunas recomendaciones para quienes deseen implementar futuros trabajos de este tipo.

En los anexos se pueden encontrar los algoritmos secuenciales y paralelos realizados en lenguaje C con la librería estándar de Open MPI.

1. MARCO TEÓRICO Y ESTADO DEL ARTE

1.1 TRANSFORMADA DE FOURIER

Aún funciones que no son periódicas (pero que son absolutamente integrables) pueden ser expresadas como la integral de senos y/o cosenos multiplicada por una función de ponderación. Esta es la transformada de Fourier, y su utilidad es aún más grande que la de las series de Fourier en muchos problemas prácticos.

Las 2 representaciones comparten la importante característica de que una función, expresada en series de Fourier o la transformada, pueden ser reconstruídas completamente por un proceso inverso con mínimas pérdidas de información.

El advenimiento de la computación digital y el "descubrimiento" del algoritmo de la transformada rápida de Fourier (FFT) a finales de los cincuentas revolucionaron el campo del procesamiento de señales. **[4]**

La transformada de Fourier F(u) de una función continua de una sola variable, f(x), se define con la ecuación

$$F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-j2\pi ux}\, dx \qquad (1)$$

$$j = \sqrt{-1}$$

De manera correspondiente, dada F(u), se puede obtener f(x) por medio de la transformada de Fourier inversa

$$f(x) = \int_{-\infty}^{\infty} F(u)\, e^{j2\pi ux}\, du \qquad (2)$$

Estas 2 ecuaciones comprenden el par de transformadas de Fourier. Es decir que una función puede ser recuperada a partir de su transformada. La transformada de Fourier de una función discreta de una variable, f(x), cuando x = 0, 1, 2,..., M-1, está dada por la ecuación

$$F(u) = {}^{1}/_{M} \sum_{x=0}^{M-1} f(x)\, e^{-j2\pi ux/M} \qquad (u = 0, 1, ..., M-1) \qquad (3)$$

Esta es la transformada de Fourier discreta o DFT por sus siglas en ingles. Similarmente, dada F(u), podemos obtener la función original usando la DFT inversa:

$$f(x) = \sum_{x=0}^{M-1} F(u)\, e^{j2\pi ux/M} \qquad (x = 0, 1, ..., M-1) \qquad (4)$$

Para obtener F(u) se sustituye u = 0 en el término exponencial y después sumamos para todos los valores de x, después sustituimos u = 1 en la exponencial y repetimos la suma para todos los valores de x. Se repite este proceso para los M valores de u y de esta manera se obtiene la DFT.

Al igual que f(x) la transformada es una cantidad discreta, y tiene el mismo número de componentes que f(x). El mismo proceso aplica para calcular la DFT inversa. **[4]**

1.2 LA DFT BIDIMENSIONAL Y SU INVERSA

La transformada discreta de Fourier se extiende fácilmente a 2 dimensiones:

$$F(u,v) = {}^{1}/_{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi(ux/M+vy/N)} \quad (u=0,1,...,M-1;\ v=0,1,...,N-1) \tag{5}$$

Y su inversa:

$$f(x,y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F(u,v)\, e^{j2\pi(ux/M+vy/N)} \quad (x=0,1,...,M-1;\ y=0,1,...,N-1) \tag{6}$$

Donde u y v son las variables de transferencia o frecuencia y **x** e **y** son las variables espaciales o de imagen. **[4]**

1.3 LA TRANSFORMADA RÁPIDA DE FOURIER "FFT" RADIX 2 CON DECIMACIÓN O DESCOMPOSICIÓN EN TIEMPO Y FRECUENCIA

La transformada rápida de Fourier (FFT) es un algoritmo eficiente usado para convertir una señal en el dominio del tiempo a su equivalente señal en el dominio de la frecuencia basado en la transformada discreta de Fourier (DFT).**[2]**

1.3.1 Desarrollo del algoritmo FFT con radix 2.

La FFT reduce considerablemente el requerimiento computacional de la DFT.

La DFT de una señal discreta en el tiempo x(nT) es:

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{nk} \qquad k = 0, 1, \ldots, N-1 \tag{7}$$

Donde el periodo de muestreo T está implícito en x(n) y N es la longitud de la señal, las constantes W son llamadas los factores "Twiddles" los cuales representan la fase por:

$$W = e^{-j2\pi/N} \quad (8)$$

Estos factores son función de la longitud N.

La ecuación para la transformada de Fourier puede ser reescrita para k=0, 1, 2... (N-1) como:

$$X(k) = x(0) + x(1)W^{k} + x(2)W^{2k} + \cdots + x(N-1)W^{(N-1)k} \quad (9)$$

La anterior ecuación representa una matriz de N*N términos, porque X(k) necesita ser calculada para N valores K. Como es una ecuación en términos de exponenciales complejas para cada K específico hay (N-1) sumas complejas y N multiplicaciones complejas. Esto resulta en un total de $(N^2 - N)$ sumas complejas y N^2 multiplicaciones complejas.

Para un valor de N grande los requerimientos computacionales serán muy intensos, la FFT reduce la complejidad computacional desde N^2 a **N*log(N).**

El algoritmo de la FFT toma ventaja de la periodicidad y simetría de los factores "Twiddle" para reducir los requerimientos de la DFT ya que:

$$W^{k+N} = W^{k} \quad (10)$$

$$W^{k+N/2} = -W^{k} \quad (11)$$

La Figura 1 ilustra las propiedades de periodicidad y simetría de las constantes "Twiddle" para N=8.

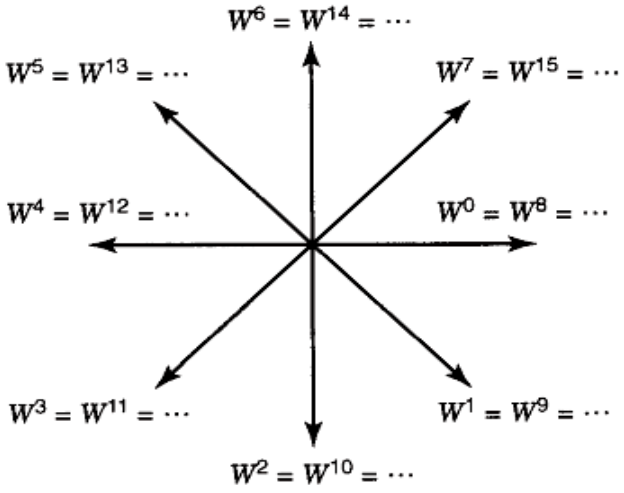

Figura 1. Periodicidad y simetría de las constantes Twiddle con N=8.
Fuente: referencia [2]

Para un radix 2 la FFT descompone una DFT de N puntos en dos DFTs de (N/2) puntos cada una. Cada DFT de (N/2) puntos es descompuesta aun mas en dos DFTs de (N/4) puntos y así sucesivamente. La última descomposición consiste de (N/2) DFTs de 2 puntos cada una. La transformada más pequeña es determinada por el radix de la FFT, para un radix 2, N debe ser una potencia o base de 2 y la más pequeña descomposición o transformada es la DFT de 2 puntos. Para un radix 4 la última descomposición es una DFT de cuatro puntos. **[2]**

1.3.2 Decimación en frecuencia, algoritmo de la FFT con radix 2.

Se separa una secuencia discreta en el tiempo:

$$x(0), x(1), \ldots, x\left(\frac{N}{2}-1\right) \quad (12)$$

Y

$$x\left(\frac{N}{2}\right),\ x\left(\frac{N}{2}+1\right), \ldots, x(N-1) \quad (13)$$

Aplicando la DFT a la ecuación 12 y 13:

$$X(k)=\sum_{n=0}^{(N/2)-1}x(n)W^{nk}+\sum_{n=N/2}^{N-1}x(n)W^{nk} \quad (14)$$

Al hacer cambio de índice en la segunda sumatoria **n= n+(N/2)** se obtiene:

$$X(k)=\sum_{n=0}^{(N/2)-1}x(n)W^{nk}+W^{kN/2}\sum_{n=0}^{(N/2)-1}x\left(n+\frac{N}{2}\right)W^{nk} \quad (15)$$

El término $W^{\frac{KN}{2}}$ es sacado de la sumatoria ya que no depende de n.

Aplicando:

$$W^{kN/2}=e^{-jk\pi}=\left(e^{-j\pi}\right)^{k}=(\cos\pi-j\sin\pi)^{k}=(-1)^{k} \quad (16)$$

Se obtiene:

$$X(k)=\sum_{n=0}^{(N/2)-1}\left[x(n)+(-1)^{k}x\left(n+\frac{N}{2}\right)\right]W^{nk} \quad (17)$$

Debido a que $(-1)^{k}$ es (1) para k par y (-1) para k impar se separa la sumatoria como sigue:

Para k par:

$$X(k)=\sum_{n=0}^{(N/2)-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W^{nk} \quad (18)$$

Y para k impar:

$$X(k)=\sum_{n=0}^{(N/2)-1}\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W^{nk} \quad (19)$$

Sustituyendo k=(2k) para K par y k=(2k+1) para k impar podemos obtener para k=0,1,2... Hasta (N/2)-1 lo siguiente:

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n + \frac{N}{2} \right) \right] W^{2nk} \quad (20)$$

$$x(2k+1) = \sum_{n=0}^{(N/2)-1} \left[x(n) - x\left(n + \frac{N}{2} \right) \right] W^{n} W^{2nk} \quad (21)$$

Como los factores W (Twiddle) son una función de la longitud N, estos pueden ser representados como W_N , entonces W_N^2 puede ser reescrito como $W_{N/2}$.

Si se define:

$$a(n) = x(n) + x(n + N/2) \quad (22)$$

$$b(n) = x(n) - x(n + N/2) \quad (23)$$

Las anteriores sumatorias para k par e impar pueden ser reescritas más claramente como dos DFTs de (N/2) puntos cada una:

$$X(2k) = \sum_{n=0}^{(N/2)-1} a(n) W_{N/2}^{nk} \quad (24)$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} b(n) W_N^n W_{N/2}^{nk} \quad (25)$$

Este algoritmo se llama decimación en frecuencia por que la secuencia de salida es descompuesta (decimada) en pequeñas subsecuencias y este proceso continua durante M etapas donde $N=2^M$. La secuencia de salida es compleja con su parte real e imaginaria. **[2]**

La FFT no es una aproximación de la DFT, esta arroja los mismos resultados pero con menos requerimientos computacionales lo cual se hace importante para un valor de N elevado.

En la Figura 2 se observa la descomposición DIF de una DFT de N=8 puntos en dos DFT's de (N/2) puntos cada una, a su vez cada DFT de (N/2) puntos es descompuesta en dos DFT's de (N/4) puntos cada una.

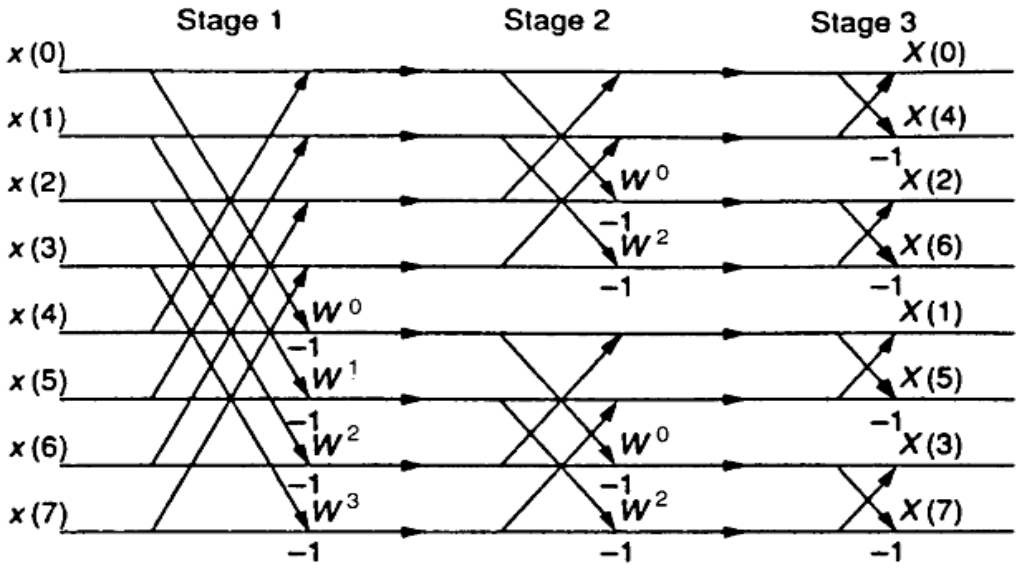

Figura 2. Decimación en frecuencia con N=8.

Fuente: referencia [2]

1.3.3 Decimación en tiempo, algoritmo de la FFT con radix 2.

Mientras que la decimación en frecuencia (DIF) descompone la secuencia de salida en pequeñas subsecuencias, la decimación en tiempo (DIT) descompone la secuencia de entrada.

Al descomponer la secuencia de entrada en una secuencia par y en otra impar:

$$x(0), x(2), x(4), \ldots, x(2n) \qquad (26)$$

$$x(1), x(3), x(5), \ldots, x(2n+1) \quad (27)$$

Aplicando la DFT se obtiene:

$$X(k) = \sum_{n=0}^{(N/2)-1} x(2n) W^{2nk} + \sum_{n=0}^{(N/2)-1} x(2n+1) W^{(2n+1)k} \quad (28)$$

Ahora usando $W_N^2 = W_{N/2}$ se obtiene:

$$X(k) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{(N/2)-1} x(2n+1) W_{N/2}^{nk} \quad (29)$$

Lo cual representa dos DFTs de (N/2) cada una, y si:

$$C(k) = \sum_{n=0}^{(N/2)-1} x(2n) W_{N/2}^{nk} \quad (30)$$

$$D(k) = \sum_{n=0}^{(N/2)-1} X(2n+1) W_{N/2}^{nk} \quad (31)$$

Entonces:

$$X(k) = C(k) + W_N^k D(k) \quad (32)$$

La ecuación anterior necesita ser interpretada para k > (N/2) -1. Usando la propiedad de simetría de los factores "Twiddle" $W^{k+\frac{N}{2}} = -W^k$ se obtiene:

$$X(k + N/2) = C(k) - W^k D(k) \qquad k = 0, 1, \ldots, (N/2) - 1 \quad (33)$$

Por ejemplo para N=8

$$X(k) = C(k) + W^k D(k) \qquad k = 0, 1, 2, 3 \quad (34)$$

$$X(k+4) = C(k) - W^k D(k) \qquad k = 0, 1, 2, 3 \tag{35}$$

La figura 3 muestran el proceso completo de DIT para N=8

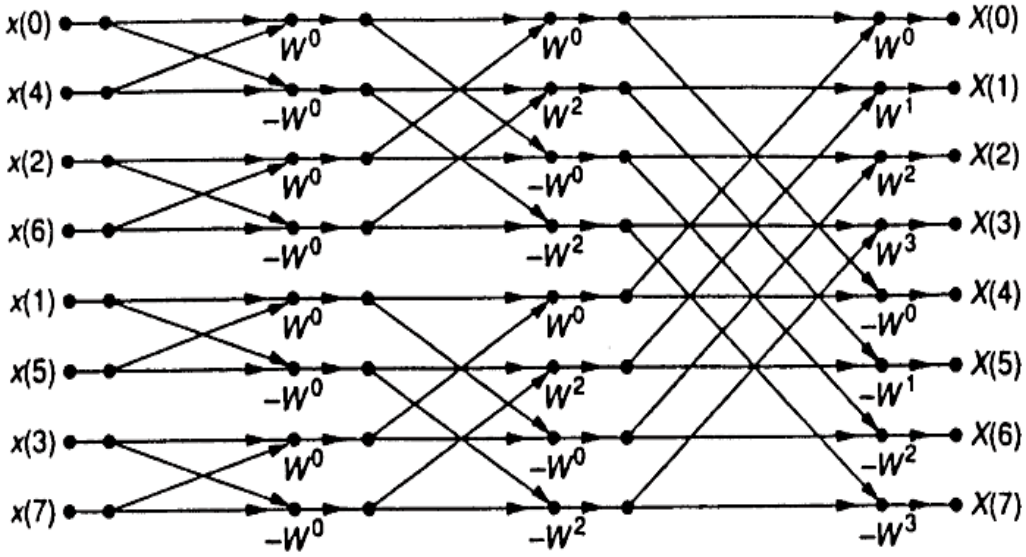

Figura 3. Decimación en tiempo con N=8.

Fuente: referencia [2]

Para aplicar la FFT radix 2 con DIT o DIF en 2 dimensiones se aplica primero la FFT unidimensional a las filas de la matriz y después se aplica la FFT unidimensional radix 2 a las columnas, teniendo en cuenta que el número de elementos de las filas y columnas debe ser potencia de 2 para poder usar la simetría analizada para la FFT.

1.4 TRANSFORMADA RÁPIDA INVERSA DE FOURIER IFFT RADIX 2

La transformada inversa de Fourier 1D convierte una secuencia en el dominio de la frecuencia a su equivalente en el domino del tiempo, esta es definida como:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W^{-nk} \qquad n = 0, 1, \ldots, N-1 \tag{36}$$

Comparando esta ecuación con la DFT podemos ver que los algoritmos de la FFT anteriormente tratados pueden ser usados para calcular la transformada inversa rápida de Fourier "IFFT" con los siguientes cambios:

1. Añadiendo el factor de escala (1/N).

2. Reemplazando W^{nk} por su conjugado complejo W^{-nk} .

Con estos cambios los mismos gráficos de flujo para la FFT pueden ser usados para calcular la IFFT. **[2]**

Para calcular IFFT radix 2 en 2D, simplemente aplicamos primero la IFFT 1D a las filas de la matriz y luego a la resultante se la aplica la IFFT 1D a las columnas.

1.5 BIT REVERSAL

Al aplicar los algoritmos de la transformada rápida de Fourier hay que reacomodar la secuencia de entrada en el DIT y la secuencia de salida en el DIF para obtener los resultados en el orden correcto, para esto existe un procedimiento llamado bit reversal.

Para ilustrar el procedimiento del bit reversal sea N=8, el primer y el tercer bit son intercambiados. Por ejemplo (100) es reemplazado por (001) o sea que (100) que define la dirección X(4) es remplazado por (001) que define la dirección X(1).

De igual forma (110) es remplazado con (011), o sea que (110) que definía la dirección 6 es remplazado con (011) que define la dirección 3.

El procedimiento del bit reversal puede ser aplicado para grandes valores de N por ejemplo para N=64 que es representado por 6 bits, el primero y sexto bit, el quinto y segundo bit y el tercero y el cuarto bit son intercambiados. **[2]**

1.6 CONVOLUCIÓN CÍCLICA EN DOS DIMENSIONES

Cuando se tienen señales discretas en el tiempo con periodos idénticos, la operación de convolución entre ellas se denomina convolución cíclica o circular representada matemáticamente como:

$$y[n] = x[n] \otimes h[n] \quad (37)$$

Donde y[n] es el resultado de la convolución cíclica de x[n] con h[n]. Si las señales son de longitud N la operación de convolución cíclica también será de longitud N. La convolución cíclica de dos matrices por el método de las transformadas rápidas de Fourier directas e inversas, se obtiene como resultado de computar la IFFT 2D de la matriz resultante de la multiplicación elemento a elemento de las FFT 2D de las matrices que son entrada a la operación de convolución. En algunas aplicaciones aunque los datos no sean de igual longitud se permite hacer una aproximación de uno de ellos al tamaño llenando con ceros la longitud restante, con el fin de utilizar los algoritmos rápidos de la transformada de Fourier. De lo contrario, sería necesario implementar la convolución en tiempo que es un proceso más lento. **[5]**

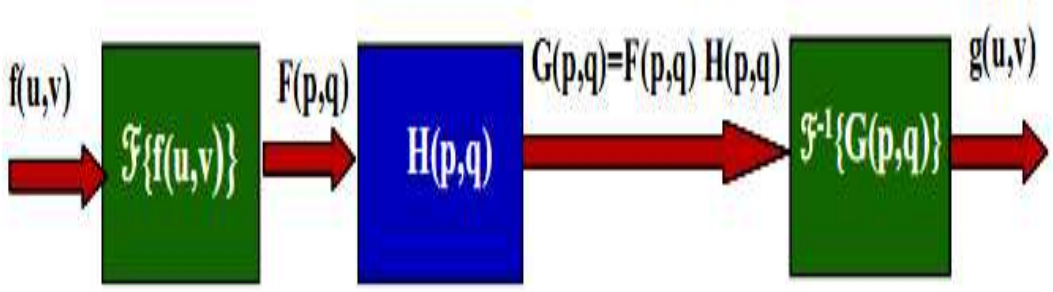

Figura 4. Convolución cíclica en 2 dimensiones.
Fuente: referencia [6]

Para ilustrar más claramente el proceso de convolución cíclica bidimensional se observa la Figura 4, se tienen dos matrices en el dominio del tiempo f(u,v) y h(u,v) cuyas FFT son respectivamente F(p,q) y H(p,q), se convolucionan en el tiempo para obtener la matriz g(u,v). El diagrama ilustra todo el proceso a seguir y previamente ya se debe haber calculado la FFT de h(u,v).

Este proceso descrito anteriormente por la figura 4 en procesamiento digital de imágenes también se puede llamar filtrado considerando a la matriz h(u,v) como un filtro.

1.7 LENGUAJE DE PROGRAMACIÓN C

C es un lenguaje de alto nivel, que permite programar con instrucciones de lenguaje de propósito general. También, C se define como un lenguaje de programación estructurado de propósito general; aunque en su diseño también primó el hecho de que fuera especificado como un lenguaje de programación de sistemas, lo que proporciona una enorme cantidad de potencia y flexibilidad.

El estándar ANSI C formaliza construcciones no propuestas en la primera versión de C, en especial, asignación de estructuras y enumeraciones. Entre otros aportes, se define esencialmente una nueva forma de declaración de funciones (prototipos), pero es esencialmente la biblioteca estándar de funciones otro de los grandes aportes. **[1]**

1.7.1 Ventajas de C.

El lenguaje C tiene una gran cantidad de ventajas sobre otros lenguajes, y son precisamente la razón fundamental de que después de casi tres décadas de uso, C siga siendo uno de los lenguajes más populares y utilizados en empresas, organizaciones y fábricas de software de todo el mundo. Algunas ventajas que justifican el uso todavía creciente del lenguaje C en la programación de computadoras son:

- El lenguaje ANSI C posee un set de instrucciones básicas y fáciles de utilizar, además con estas funciones se pueden crear librerías y funciones más complejas, que pueden manejar matrices y algoritmos más complejos.

- Se puede utilizar C para desarrollar sistemas operativos, compiladores, sistemas de tiempo real y aplicaciones de comunicaciones. Debido a su alta portabilidad también se pueden llevar aplicaciones a diferentes plataformas.

Un programa en C puede ser escrito para un tipo de computadora y trasladarse a otra computadora con pocas o ninguna modificación, propiedad conocida como portabilidad. El hecho de que C sea portable es importante ya que la mayoría de los modernos computadores tienen un compilador C.

C se caracteriza por su velocidad de ejecución, ya que en los primeros días de la informática los problemas de tiempo de ejecución se resolvían escribiendo todo o parte de una aplicación en lenguaje ensamblador (lenguaje muy cercano al lenguaje máquina).

Debido a que existen muchos programas escritos en C, se han creado numerosas bibliotecas en C para programadores profesionales que soportan gran variedad de aplicaciones. Existen bibliotecas del lenguaje C que soportan aplicaciones de bases de datos, gráficos, edición de texto y comunicaciones. **[1]**

1.7.2 Características técnicas de C.

Hay numerosas características que diferencian a C de otros lenguajes y lo hacen eficiente y potente a la vez.

- Una nueva sintaxis para declarar funciones. Una declaración de función puede añadir una descripción de los argumentos de la función, esta información adicional sirve para que los compiladores detecten más fácilmente los errores causados por argumentos que no coinciden.
- Asignación de estructuras (registros) y enumeraciones.
- Una nueva definición de la biblioteca que acompaña a C. Entre otras funciones se incluyen: acceso al sistema operativo (por ejemplo, lectura y escritura de archivos), entrada y salida con formato, asignación dinámica de memoria y manejo de cadenas de carácteres.

- Una colección de cabeceras estándar que proporciona acceso uniforme a las declaraciones de funciones y tipos de datos. **[1]**

1.8 TÉCNICAS DE PARALELIZACIÓN DE ALGORITMOS

Tres arquitecturas se muestran en las figuras 5,6 y 7 en donde cada "PU" (processing unit) es una unidad de procesamiento que para este trabajo corresponde a cada nodo.

1.8.1 SIMD (Single Instruction, Multiple Data).

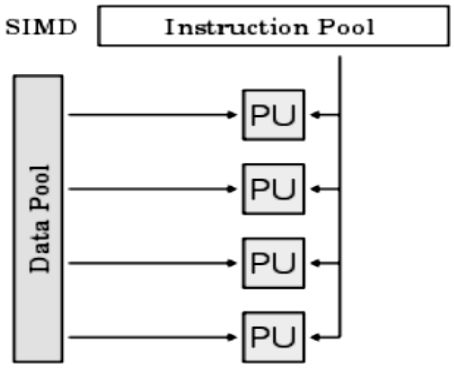

Figura 5. Paralelización de la forma SIMD.
Fuente: referencia [10]

SIMD instrucciones simples, múltiples datos es una técnica empleada para conseguir paralelismo a nivel de datos. **[10]**

En este tipo de paralelización los datos son fragmentados en cantidades más pequeñas y cada parte de los datos se distribuye a un proceso que ejecuta el mismo conjunto de instrucciones. En este tipo de procesamiento se hace evidente que los datos se pueden fragmentar sin afectar el resultado y que cada fragmento se puede operar independientemente.

1.8.2 MISD (Multiple Instruction, Single Data).

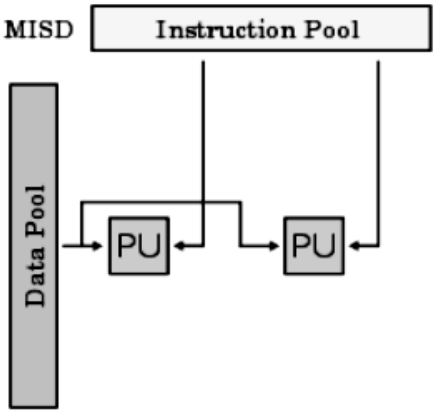

Figura 6. Paralelización de la forma MISD.

Fuente: referencia [10]

MISD Múltiples Instrucciones, Datos sencillos es un tipo de arquitectura de computación paralela donde muchas unidades funcionales realizan diferentes operaciones sobre los mismos datos.

En este tipo de procesamiento cada PU aplica diferentes instrucciones a los mismos datos, cada unidad de procesamiento se encarga de una parte del código fuente. **[10]**

1.8.3 MIMD (Multiple Instruction, Multiple Data).

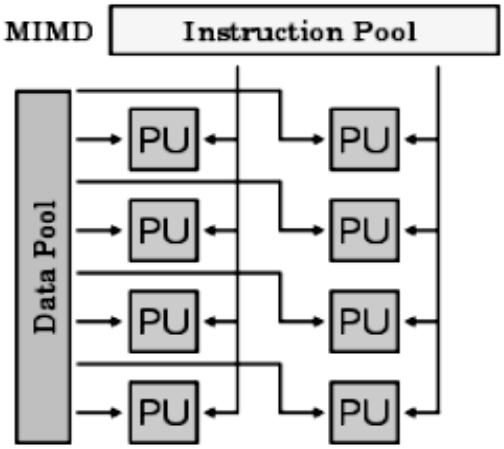

Figura 7. Paralelización de la forma MIMD.

Fuente: referencia [10]

El método MIMD combina los métodos SIMD y MDSI, separando los datos y aplicándole diferentes instrucciones sobre cada parte de ellos. En la Tabla 1 se presenta un esquema de las formas de paralelizar un algoritmo de acuerdo a la taxonomía de Flynn.

Tabla 1. Taxonomía de Flynn.

Taxonomía de Flynn

	Una instrucción	Múltiples instrucciones
Un dato	SISD	MISD
Múltiples datos	SIMD	MIMD

1.9 CLUSTERS

1.9.1 Definición del término Cluster.

El término cluster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Hoy en día juegan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno. La tecnología cluster ha evolucionado en apoyo de actividades que van desde aplicaciones de supercomputación, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos. **[11]**

El cómputo con clusters surge como resultado de la convergencia de varias tendencias actuales, que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así

como la creciente necesidad de potencia computacional para aplicaciones que lo requieran.

Un cluster es un grupo de múltiples ordenadores unidos mediante una red, de tal forma que el conjunto es visto como un único ordenador. Los clusters son generalmente empleados para mejorar el rendimiento y la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

De un cluster se espera que presente combinaciones de los siguientes servicios:

1. Alto rendimiento.

Un cluster de alto rendimiento es un conjunto de ordenadores que está diseñado para dar altas prestaciones en cuanto a capacidad de cálculo. Los motivos para utilizar un cluster de alto rendimiento son el tamaño del problema por resolver y el precio de la máquina necesaria para resolverlo.

2. Alta disponibilidad.

Un cluster de alta disponibilidad es un conjunto de dos o más máquinas que se caracterizan por mantener una serie de servicios compartidos y por estar constantemente monitorizándose entre sí.

No hay que confundir un cluster de alta disponibilidad con un cluster de alto rendimiento. El segundo es una configuración de equipos diseñado para proporcionar capacidades de cálculo mayores que la que proporcionan los equipos individuales, mientras que el primer tipo de cluster está diseñado para garantizar el funcionamiento ininterrumpido de ciertas aplicaciones.

La construcción del cluster es fácil y económica debido a su flexibilidad. Pueden tener todos los nodos la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y

sistemas operativos similares (cluster semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo).

Para que un cluster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.

1.10 PROGRAMACIÓN PARALELA CON MPI

MPI es un protocolo de paso de mensajes, es una biblioteca que ha sido implementada en varios lenguajes de programación pero en este trabajo se utiliza la implementación en el lenguaje C. MPI es un estándar planteado como una biblioteca para software, actualmente no existe una implementación hardware de MPI.

El concepto de paso de mensajes define una red donde varios nodos envían entre sí mensajes donde vienen los datos y otras etiquetas para el tráfico correcto en la red. El programador se tiene que preocupar por incluir en la programación las sentencias para enviar y recibir mensajes y puede controlar el tráfico, el compilador se encarga de enlazar los detalles de la red y del entorno para que este reconozca a cada nodo enumerado. **[17]**

1.10.1 Desarrollo de aplicaciones paralelas para Clusters utilizando MPI (Message Passing Interface).

MPI es una especificación de paso de mensajes, diseñada para ser el estándar de computación paralela de memoria distribuida. Esta interfaz trata de establecer un estándar práctico, eficiente, portátil y flexible para el protocolo de paso de mensajes.

MPI no es un lenguaje de programación, es un conjunto de funciones y macros que conforman una librería estándar de C y C++, y subrutinas en Fortran.

MPI ofrece un API, junto con especificaciones de sintaxis y semántica que explican cómo sus funcionalidades deben añadirse en cada implementación que se realice (tal como almacenamiento de mensajes o requerimientos para entrega de mensajes). MPI incluye operaciones punto a punto y colectivas, todas destinadas a un grupo específico de procesos.

MPI realiza la conversión de datos heterogéneos como parte transparente de sus servicios, por medio de la definición de tipos de datos específicos para todas las operaciones de comunicación. Se pueden tener tipos de datos definidos por el usuario o primitivas.

1.10.2 Fundamentos de MPI.

Con MPI el número de procesos requeridos se asigna antes de la ejecución del programa, y no se crean procesos adicionales mientras la aplicación se ejecuta. A cada proceso se le asigna una variable que se denomina rank, la cual identifica a cada proceso en el rango de 0 a p-1, donde p es el número total de procesos. El control de la ejecución del programa se realiza mediante la variable rank; la variable rank permite determinar qué proceso ejecuta determinada porción de código.

En MPI se define un comunicator como una colección de procesos, los cuales pueden enviar mensajes entre ellos; el comunicator básico se denomina MPI_COMM_WORLD y se define mediante un macro del lenguaje C. MPI_COMM_WORLD agrupa a todos los procesos activos durante la ejecución de una aplicación.

Las llamadas de MPI se dividen en cuatro clases:

- Llamadas utilizadas para inicializar, administrar y finalizar comunicaciones.
- Llamadas utilizadas para transferir datos entre un par de procesos.
- Llamadas para transferir datos entre varios procesos.
- Llamadas utilizadas para crear tipos de datos definidos por el usuario.

2 DESCRIPCIÓN DEL SOFTWARE Y HARDWARE

2.1 DESCRIPCIÓN DEL SOFTWARE

2.1.1 Sistema operativo MAC OS X 10.5 Leopard.

Mac OS X v10.5 “Leopard” fue lanzado el 26 de Octubre de 2007 como la sexta revisión del sistema operativo de Apple Mac OS X para computadoras Macintosh.

Desarrollador**:** Apple Computer Modelo de desarrollo**:** Software no libre
Última versión estable**:** 10.5.6 / 15 de diciembre de 2008
Estos son puntos principales que destaca Apple:

- La posibilidad de poder volver en el tiempo a una versión especifica de los contenidos de una carpeta, del disco duro completo o de un sólo archivo.
- Incluye búsquedas avanzadas, y la posibilidad de buscar en varios computadores Mac en red.
- El sistema operativo esta optimizado para utilizar una arquitectura de 64 bits y manejo eficiente de memoria.
- Instalación de otros sistemas operativos, como Windows XP o Windows Vista, en una partición aparte en Macs con procesador Intel.
- Las mejoras al compartir archivos por la red incluye más controles sobre los permisos, unificación de compartición de AFP, FTP y SMB en un sólo panel de control y la posibilidad de compartir carpetas individuales, una función que no ha estado disponible desde Mac OS 9. **[13]**

2.1.2 Xcode.

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con Mac OS X. Xcode trabaja conjuntamente con Interface Builder, una herramienta gráfica para la creación de interfaces de usuario. **[14]**

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo a Cocoa, Carbon y Java. Otras compañías han añadido soporte para GNU Pascal, Free Pascal, Ada y Perl. En la Tabla 2 se presentan las características del Xcode.

Tabla 2. Características del entorno de programación Xcode.

Xcode	
Desarrollador	Apple Inc.
Última versión	3.0 (26 de octubre de 2007)
S.O.	Mac OS X v10.3 (Versión 1.x) Mac OS X v10.4 (Versión 2.x) Mac OS X v10.5 (Versión 3.x)
Género	IDE
Licencia	Propietaria y libre
Lengua Española	si
Sitio web	www.developer.apple.com/tools/xcode

2.1.3 POOCH (Parallel Operation and Control Heuristic application).

Operación paralela y control heurístico de la aplicación.

"Pooch es la forma más rápida y fácil de iniciar en la computación paralela", según el Dr. Dean Dauger, Presidente de Research Dauger Inc. empresa desarrolladora del Pooch. Pooch provee una interfaz de usuario sencilla para la distribución e inicialización de una aplicación paralela numéricamente intensiva en una red de computadores Macintosh. Este coordina la distribución de los datos, los comandos desde otros Macintosh (que deben tener instalado el

software Pooch) y proporciona una interfaz para supervisar los procesos y la asignación de nodos. Este permite al usuario combinar la supercomputación con un fácil manejo, su tecnología de diseño le permite ser lo suficientemente flexible como para operar en una amplia variedad de entornos de red o configuraciones de grupo y al mismo tiempo apoyar el mayor número de entornos de programación. **[15]**

Pooch soporta el estándar MPI, por lo que es fácil escribir código para supercomputación en un grupo de computadores Macintosh.

No depende del uso compartido de archivos, sistema de archivos de red (NFS), protocolos rsh, ssh, solo requiere el código compilado y enlazado es decir, el ejecutable del código y no requiere modificaciones al hardware o al sistema operativo.

Requerimientos del Pooch.

- Cualquier red que soporte TCP/IP (10BaseT, 100BaseT, Gigabit, Airport, o combinaciones).
- Macintosh conectados a través de una red TCP / IP (100BaseT, 10BaseT, Gigabit o Airport) con Mac OS 9 o posterior. Pooch en OS X 10.2.1 y otras versiones posteriores es totalmente compatible.

Un procesador cuenta como un nodo, pero un procesador doble núcleo cuenta como 2 nodos y Pooch lanza un proceso por cada nodo. Actualmente el pooch soporta 16 nodos trabajando.

2.1.4 Open MPI.

Open MPI es una interfaz de paso de mensajes, está escrita por el MPI Forum. MPI es una API estándar utilizada para la computación paralela o distribuida. El estándar MPI comprende dos documentos: MPI-1 (1994) y MPI-2 (1996). **[12]** Open MPI es una implementación de código abierto de los estándares MPI-1 y MPI-2.

Open MPI es un proyecto que combina las tecnologías y recursos de otros proyectos (FT-MPI, LA-MPI, LAM/MPI y PACX-MPI) para construir la mejor librería MPI.

Características:

- Cumple el estándar MPI-2.
- Distribución de procesos de forma dinámica.
- Alto rendimiento en todas las plataformas.
- Administración de trabajos rápida y fiable.
- Tolerancia a fallos de red y procesos.
- Soporte de redes heterogéneas.
- Una única librería soporta todas las redes.
- Portable y estable.
- Modificable por los instaladores y usuarios finales.
- Licencia: Licencia BSD
  La licencia BSD es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Pertenece al grupo de licencias de software Libre.
- Para el lenguaje C, Open MPI usa el compilador GCC.

2.2 DESCRIPCIÓN DEL HARDWARE

2.2.1 Características de los computadores.

En la Tabla 3 se observa algunas características de los computadores MAC utilizados. **[13]**

Tabla 3. Características de los computadores MAC.

	Mac mini de 120 GB (MB463*/A)
Procesador	Core 2 Duo de Intel a 1.83 GHz
Caché de nivel 2	2 MB (compartida)
Bus del sistema	667 MHz
Memoria	2 GB de SDRAM DDR2 a 667 MHz
Disco duro	Serial ATA de 80 GB a 5.400 rpm.
Unidad óptica	SuperDrive a 8x de carga por ranura compatible con discos de doble capa (DVD±R DL, DVD±RW y CD-RW):
Gráficos	Procesador gráfico GeForce 9400M de NVIDIA con 128 MB de SDRAM DDR3 compartida con la memoria principal
Puertos	Un puerto FireWire 800 (8 vatios); cinco puertos USB 2.0 (hasta 480 Mb/s); salida mini-DVI; salida VGA (con adaptador opcional); Mini DisplayPort
Sonido	Altavoz integrado, entrada combinada de audio de línea y óptico digital, salida combinada de auriculares y óptico digital
Redes	10/100 BASE-T integrada
Conexión inalámbrica	Wi-Fi AirPort Extreme (802.11n) integrada; Bluetooth 2.1 +

2.2.2 Red de Ethernet.

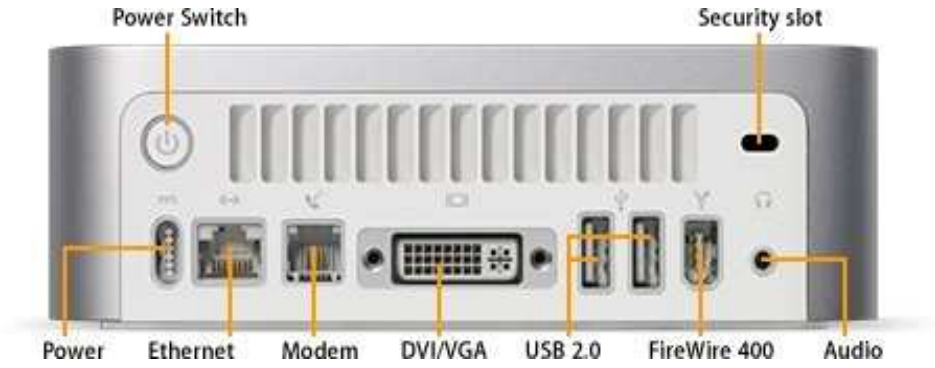

Figura 8. Panel de entrada/salida del Mac mini

Fuente: referencia [16]

El Mac mini dispone de un panel de Entrada/Salida (E/S) en la parte posterior que contiene varios puertos y conectores. En la Figura 8 se puede observar los puertos disponibles en el computador MAC.

Puerto Ethernet (10/100/Base-T).

El Mac mini dispone de un puerto Ethernet 10/100 Mbit/s (megabits por segundo) integrado. El Mac mini se puede conectar a un módem DSL o de cable, un concentrador, un enrutador o incluso directamente a otro ordenador Macintosh (como se propone en este trabajo) utilizando un cable Ethernet. El dispositivo conectado debe ser o bien 10Base-T o 100Base-TX. El Mac mini detectará automáticamente cual es el tipo de red. No es necesario utilizar un cable Ethernet cruzado para conectar el ordenador a otros dispositivos Ethernet. **[16]**

2.2.3 Cables de Red Ethernet.

Cable de Red de Categoría 5e es una de las cinco clases de cableado UTP RJ45 (del inglés Unshielded Twisted Pair, par trenzado no apantallado) que se describen en el estándar TIA/EIA-568-B (tres estándares que tratan el cableado comercial para productos y servicios de telecomunicaciones). El cableado de categoría 5 se usa para ejecutar la Interfaz de Distribución de datos por cobre y puede transmitir datos a velocidades de hasta 1000 Mbps (1000 Mega bits por segundo). El cable es utilizado para unir un Hub o Switch a otros computadores, o como en nuestro caso un computador a otro debido a que la red Ethernet integrada de los Mac ofrece esta ventaja. (Ver Figura 9)

Figura 9. Cable de red Ethernet categoría 5e.
Fuente: autores del proyecto

2.3 DISEÑO E IMPLEMENTACIÓN DEL CLUSTER

Importante**:** En general este diseño e implementación del cluster MAC, libera a los usuarios y administradores de configuraciones extrañas y de modificación o creación de archivos en el sistema operativo.

En particular**:**

• No hay línea de comandos de inicio de sesión: rsh, ssh o keygen.

• No hay almacenamiento compartido: NFS, AFS u otros.

• No hay modificación de archivos del sistema operativo o del PATH de usuario: los archivos de host y los archivos de la máquina.

2.3.1 Diseño del Cluster.

En definitiva el diseño del cluster MAC para este trabajo es el siguiente:

La interfaz de usuario se implementa no mediante consola sino con la herramienta Pooch.

Se utiliza la librería de Open MPI para compilar el código realizado en lenguaje C, el cual esta paralelizado con las rutinas estándar de MPI.

El compilador gcc para el lenguaje C se instala mediante el entorno de desarrollo integrado Xcode que incluye la colección de compiladores del proyecto GNU (GCC). **[18]**

2.3.2 Implementación del Cluster.

Open MPI es una aplicación de código abierto que es desarrollado y mantenido por un consorcio de instituciones académicas, de investigación y socios de la industria.

Cuando Pooch v1.7.6 se ejecuta por primera vez en Leopard, este solicita permiso para instalar los módulos en /usr/lib/openmpi/ que brindan la comunicación entre Pooch y Open MPI y que son necesarios para que Pooch utilice Open MPI.

Una vez que se haya terminado de instalar todos estos módulos, es posible compilar el código C utilizando Open MPI con el comando:

mpicc -o [ejecutable_creado] [codigo_a_compilar.c]

Pooch debe reconocer el ejecutable resultante. **[18]**

Importante: Seleccionar "Open MPI" como tipo de trabajo en las opciones de la ventana de trabajo. Pooch deberá ser capaz de lanzar el ejecutable en un Mac cuyos nodos del clúster se encuentren ejecutando Leopard. Como siempre, el uso de almacenamiento compartido ssh, rsh y configuraciones de archivos inetd.conf o .rhosts no son necesarios, además la ejecución del comando mpirun tampoco es necesaria. Otro aspecto importante es que para ejecutar el Pooch hay que tener abierta la Terminal o consola de Mac.

3 IMPLEMENTACIÓN DE LOS ALGORITMOS

Cabe aclarar que un nodo no es equivalente a un proceso, un nodo puede ejecutar varios procesos pero en este trabajo un nodo ejecuta un solo proceso, o sea que a cada nodo se le asigna un proceso. Los computadores tienen un procesador doble núcleo y pooch reconoce cada núcleo como un nodo.

3.1 IMPLEMENTACIÓN DE LOS ALGORITMOS EN FORMA SECUENCIAL

3.1.1 Algoritmo de la transformada de Fourier unidimensional directa e inversa.

De este algoritmo básico depende el algoritmo de la convolución cíclica, la transformada se aplica a un vector de tamaño N que puede representar una fila o una columna de una matriz.

Pseudocódigo:

a. Lectura del vector
b. Calculo del vector con los valores W
c. Aplicación del algoritmo FFT DIT
d. Algoritmo bit reversal

La transformada inversa tiene similitud con el proceso anterior y consta de los siguientes pasos:

a. Lectura del vector
b. Calculo del vector con los valores W y cambio de signo de la parte imaginaria
c. Aplicación del algoritmo FFT DIT
d. Algoritmo bit reversal y división del resultado entre el número total de puntos.

3.1.2 Algoritmo de la transformada de Fourier bidimensional.

Este algoritmo se encarga de obtener la transformada 2D de Fourier, para esto utiliza la transformada rápida de Fourier aplicada en cada una de las filas de la matriz y luego al resultado le aplica la FFT a cada columna de la matriz, para obtener la representación en el dominio de la frecuencia en 2 dimensiones de la matriz. En la Figura 10 se observa el diagrama de flujo del algoritmo.

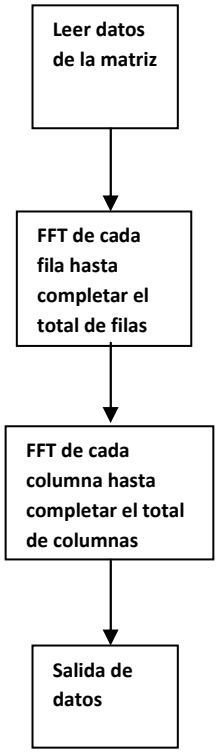

Figura 10. Diagrama de flujo de la FFT 2D.
Fuente: autores del proyecto

Pseudocódigo:

a. Lectura de datos, lectura de la matriz
b. Calculo de los factores W para las filas
c. Calculo de la transformada de Fourier de cada fila
d. Calculo de los factores W para cada columna
e. Calculo de la transformada de Fourier de cada columna

3.1.3 Algoritmo de la transformada de Fourier inversa bidimensional.

Este algoritmo realiza el proceso inverso a la transformada, transforma la matriz que está en el dominio de la frecuencia en una matriz en el dominio del tiempo. La IFFT se puede derivar de la FFT ya que para obtenerla se modifican los factores W cambiando el signo de la parte imaginaria y al final de la transformada se divide cada resultado en el número de puntos totales N. (Ver Figura 11).

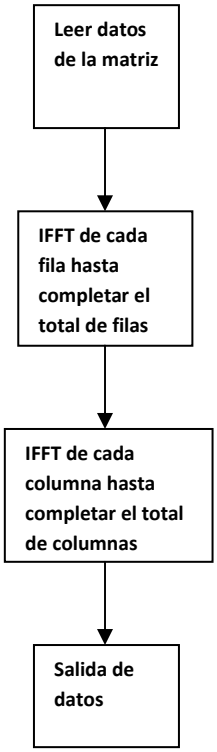

Figura 11. Diagrama de flujo de la IFFT 2D.
Fuente: autores del proyecto

3.1.4 Algoritmo de la convolución cíclica.

Para realizar la convolución cíclica entre 2 matrices, primero las 2 matrices se transforman a su representación en el dominio de la frecuencia y luego se multiplican punto a punto. Cabe decir que las matrices deben tener el mismo tamaño ya que se aplica una convolución cíclica donde el tamaño de la matriz resultado es el mismo que el de las entradas. (Ver Figura 12).

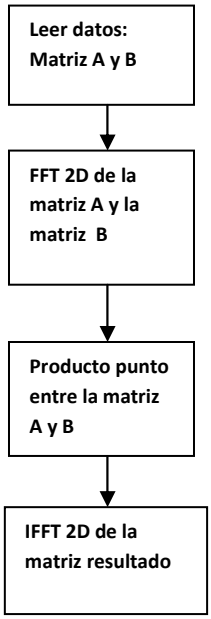

Figura 12. Diagrama de la convolución cíclica 2D.

Fuente: autores del proyecto

3.2 IMPLEMENTACIÓN DE LOS ALGORITMOS EN FORMA PARALELA CON 2 NODOS

3.2.1 Algoritmo de la transformada de Fourier bidimensional con 2 nodos.

En este algoritmo se separa la implementación en 2 procesos, cada nodo ejecuta un proceso, el nodo 0 el proceso 0 y el nodo 1 el proceso 1. Los 2 procesos trabajan simultáneamente sobre una mitad diferente de los datos, donde cada proceso aplica las mismas operaciones. Lo mismo se aplica a la transformada inversa con 2 procesos excepto que la parte imaginaria de los coeficientes W se multiplica por un signo menos y el resultado se divide por el número total de muestras. En la Figura 13 se observa el diagrama de flujo del algoritmo de la FFT 2D para 2 nodos.

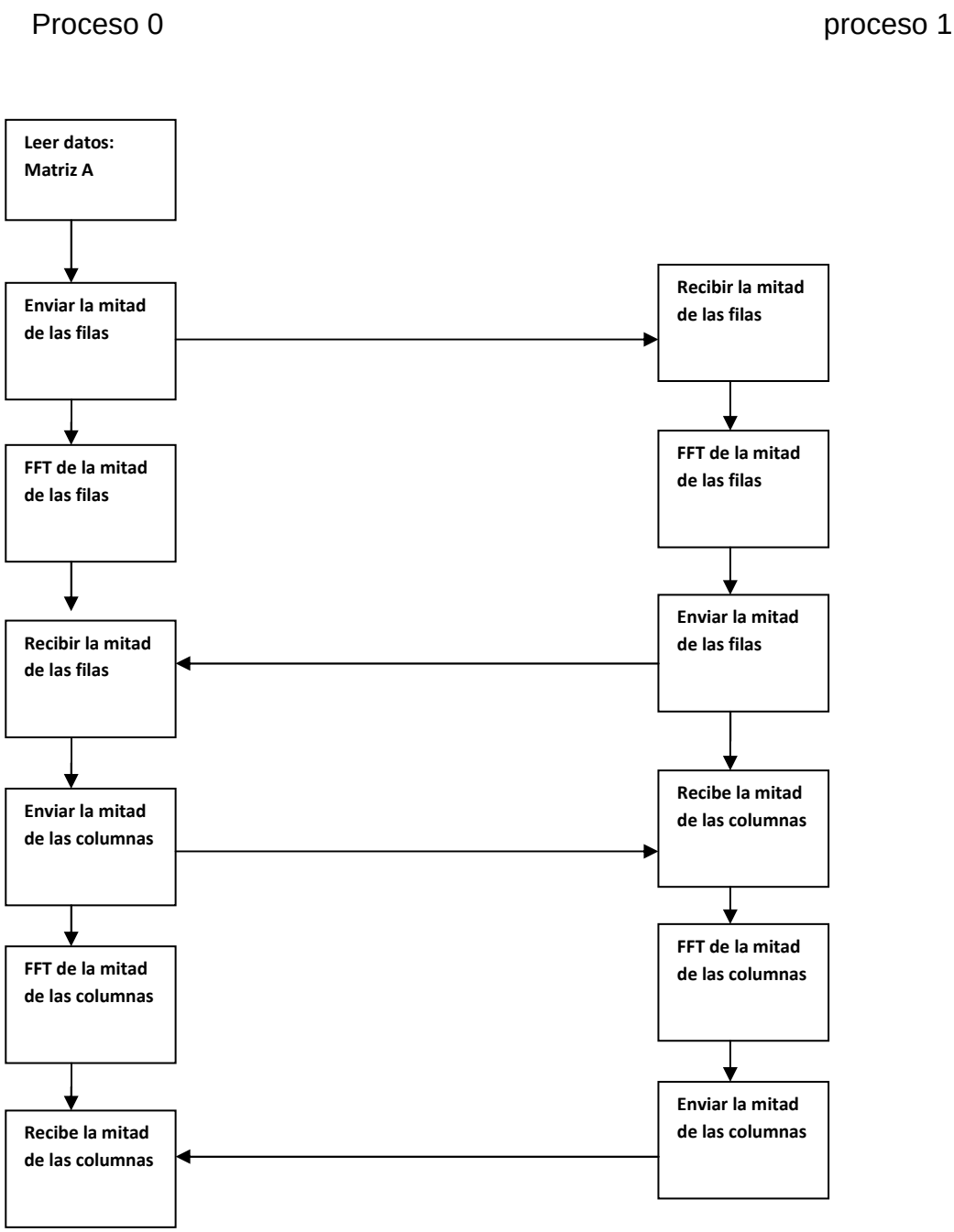

Figura 13. Diagrama de flujo de la FFT 2D con 2 nodos.
Fuente: autores del proyecto

3.2.2 Algoritmo de la convolución paralela por el método de SIMD

En este algoritmo se trabaja con 2 nodos, cada nodo ejecuta un proceso, el nodo 0 el proceso 0 y el nodo 1 el proceso 1. Cada proceso trabaja sobre una parte de los datos aplicando el mismo algoritmo; el proceso 0 selecciona la

matriz de datos B y aplica la 2D FFT, mientras que el proceso 1 realiza la misma operación sobre la matriz A. Al realizar la multiplicación punto a punto un proceso opera sobre la mitad superior de los datos y el otro sobre la mitad inferior, como se puede observar en la tabla 4. Luego se aplica la 2D IFFT, el proceso 0 trabaja sobre la primera mitad de las filas y columnas mientras que el proceso 1 sobre la segunda mitad. Este tipo de procesamiento se denomina SIMD debido a que cada proceso utiliza las mismas instrucciones pero aplicada a diferentes datos.

Este método tiene la ventaja que los mensajes enviados son de tamaños pequeños, y una desventaja al enviar muchos mensajes en la red a cada nodo participando en el procesamiento. (Ver Tabla 4 y Figura 14).

Tabla 4. Algoritmo de la convolución paralela por el método de SIMD.

PROCESO 0	PROCESO 1
Leer datos: Matriz A y B	
Envía la matriz A al proceso 1	**Recibe la matriz A**
2D FFT de la matriz B	**2D FFT de la matriz A**
Recibe la matriz A	**Envía la matriz A**
Envía la mitad superior de las filas de matriz A y B	**Recibe la mitad superior de las filas de matriz A y B**
Producto punto entre la mitad inferior de las filas de matriz A y B	**Producto punto entre la mitad superior de las filas de matriz A y B**
Recibe la mitad de la matriz resultado	**Envía la mitad de la matriz resultado**
Envía la mitad superior de las filas de la matriz resultado	**Recibe la mitad superior de las filas de la matriz resultado**
IFFT de las filas inferiores de la matriz resultado	**IFFT de las filas superiores de la matriz resultado**
Recibe la mitad superior de las filas de la matriz resultado	**Envía la mitad superior de las filas de la matriz resultado**

Envía la primera mitad de las columnas de la matriz resultado	Recibe la primera mitad de las columnas de la matriz resultado
IFFT de la segunda mitad de las columnas	IFFT de la primera mitad de las columnas
Recibe la primera mitad de las columnas de la matriz resultado	Envía la primera mitad de las columnas de la matriz resultado
Fin del proceso	Fin del proceso

CONTINUACION DE LA TABLA 4.

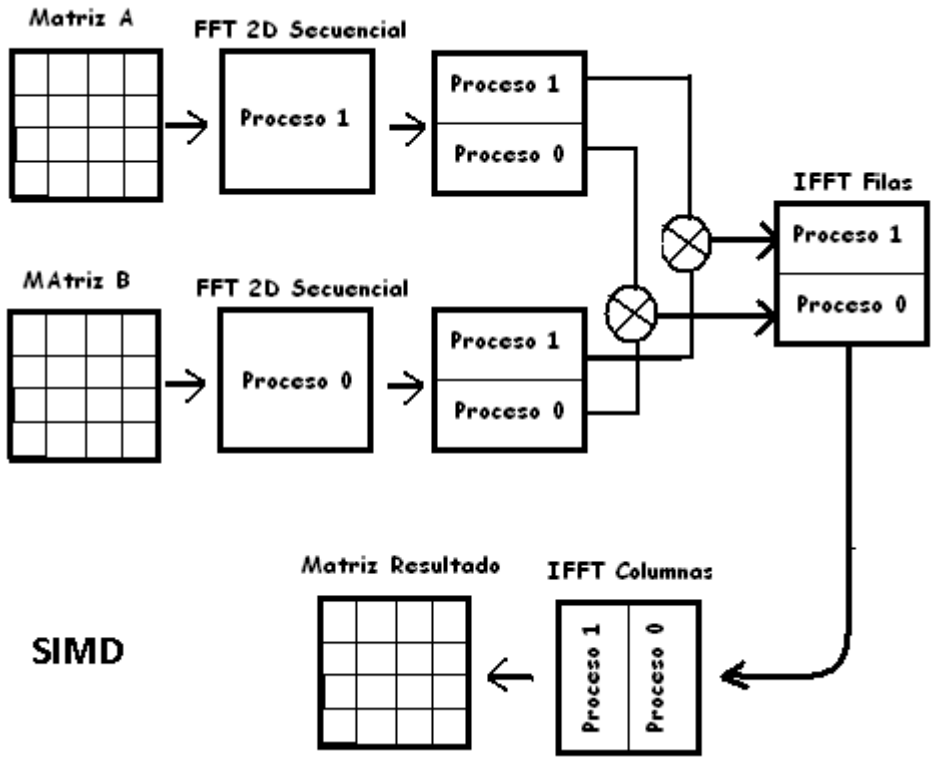

Figura 14. Convolución cíclica 2D de la forma SIMD con 2 nodos.
Fuente: autores del proyecto

3.2.3 Algoritmo de la convolución paralela por el método de MISD.

En este algoritmo se trabaja con 2 nodos, cada nodo ejecuta un proceso, el nodo 0 el proceso 0 y el nodo 1 el proceso 1**.** Los 2 procesos comparten los mismos datos (matriz A y matriz B), pero cada uno aplica un código diferente

sobre estos. El código del proceso 0 se diseñó para realizar la FFT de las filas superiores de la matriz A y excluir las demás filas. El código del proceso 1 se diseño para realizar la FFT de las filas inferiores y excluir las demás. El mismo principio de procesamiento de la FFT realizado a las filas de la matriz A se aplica a las columnas. Luego del procesamiento total de la FFT 2D de la matriz A se hace lo mismo con la matriz B.

Para la multiplicación de las matrices cada uno de los procesos tiene los resultados obtenidos anteriormente, pero el código del proceso 1 se diseño para realizar la operación sobre la mitad superior de las filas y excluir el resto, mientras el código del proceso 0 se diseño para realizar la multiplicación de lo que excluye el proceso 1.

Por último se le aplica la IFFT 2D a la matriz resultado de la multiplicación, cada proceso actúa sobre la misma matriz (datos) pero con diferentes instrucciones. El código del proceso 0 se diseño para operar la primera mitad de columnas y filas y excluir las demás, mientras que el código del proceso 1 se diseño para operar sobre las mitades restantes. Este método MISD se diferencia del método anterior SIMD en que se aplican los procesos sobre los mismos datos pero cada proceso ejecuta una parte distinta del código o un algoritmo distinto, esto implica que cada proceso debe tener almacenado la misma cantidad de datos que el otro.

Este método tiene como ventaja que se envían menos mensajes por la red a cada nodo participando en el procesamiento, pero la desventaja es que se envían bloques de datos grandes. (Ver Tabla 5 y Figura 15).

Tabla 5. Algoritmo de la convolución paralela por el método de MISD.

PROCESO 0	PROCESO 1
Leer datos: Matriz A y B	
Envía la matriz A al proceso 1	**Recibe la matriz A**
FFT de fila N/2 a fila N de matriz A	**FFT de fila 1 a fila N/2 de matriz A**
Recibe la matriz A , construye y	**Envía la matriz A**

reenvía la matriz A	
FFT de columna N/2 a columna N de matriz A	FFT de columna 1 a columna N/2 de matriz A
Recibe la matriz A	Envía la matriz A
Envía la matriz B al proceso 1	Recibe la matriz B
FFT de fila N/2 a fila N de matriz B	FFT de fila 1 a fila N de matriz B
Recibe la matriz B , construye y reenvía la matriz B	Envía la matriz B
FFT de columna N/2 a columna N matriz B	FFT de columna 1 a columna N/2 matriz B
Recibe la matriz B	Envía la matriz B
Envía matrices A y B al proceso 1	Recibe matrices A y B
Multiplica punto a punto las matrices A y B desde la fila N/2 a la fila N	Multiplica punto a punto las matrices A y B desde la fila 1 a la fila N/2
Recibe resultados , construye y reenvía la matriz resultado	Envía resultados
IFFT de la matriz resultado de la fila N/2 a la fila N	IFFT de la matriz resultado de la fila 1 a la fila N/2
Recibe la matriz resultado , construye y reenvía la matriz resultado	Envía la matriz resultado
IFFT de la matriz resultado de la columna N/2 a la columna N	IFFT de la matriz resultado de la columna 1 a la columna N/2
Recibe la matriz resultado y construye	Envía la matriz resultado
Fin del proceso	Fin del proceso

CONTINUACION DE LA TABLA 5.

En este algoritmo el proceso 0 trabaja como el proceso maestro encargado de administrar la transferencia de los datos al tiempo que ejecuta sus funciones.

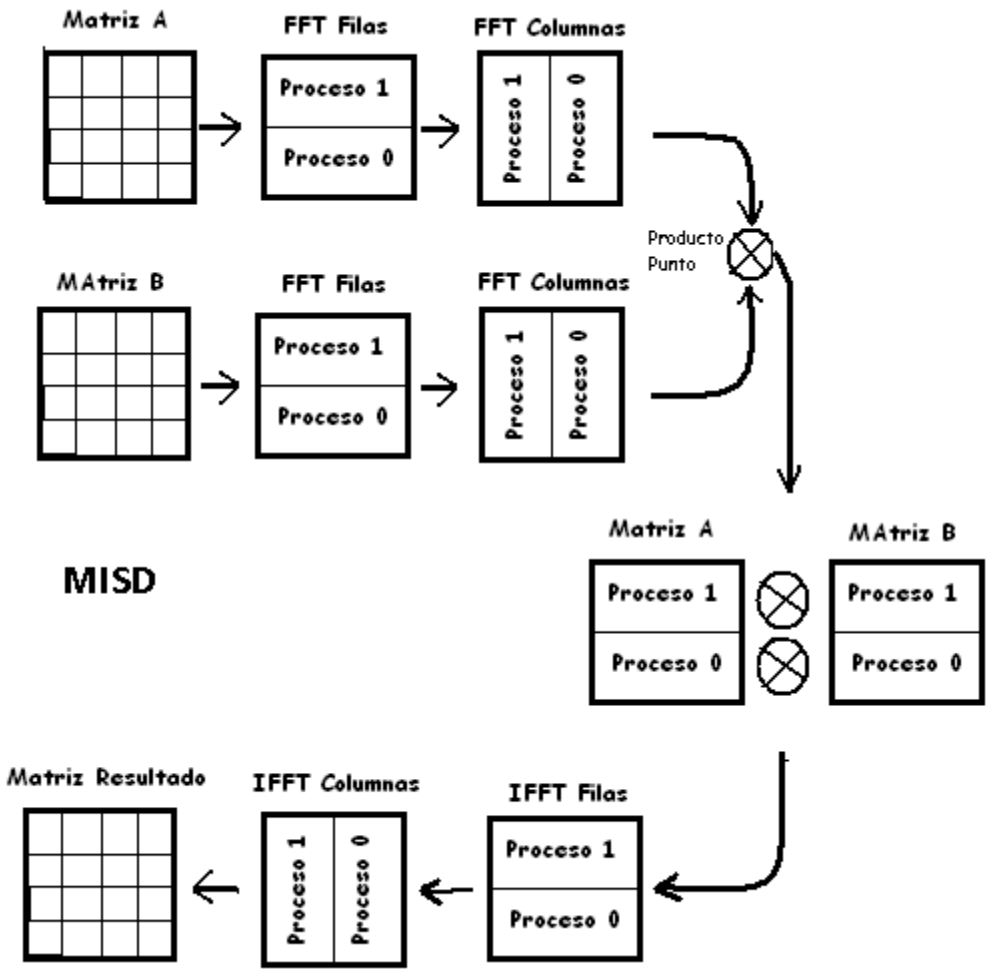

Figura 15. Convolución cíclica 2D de la forma MISD con 2 nodos.
Fuente: autores del proyecto

3.3 IMPLEMENTACIÓN DE LOS ALGORITMOS EN FORMA PARALELA CON 4 NODOS

3.3.1 Transformada de Fourier bidimensional con 4 nodos.

En este algoritmo se trabaja con 4 nodos, cada nodo ejecuta un proceso, el nodo 0 el proceso 0, el nodo 1 el proceso 1, y así respectivamente**.** Cada uno de los procesos recibe una cuarta parte de los datos, luego cada nodo procesa una parte distinta de las matrices aplicando las mismas instrucciones. El

algoritmo primero aplica la FFT a las filas y luego a las columnas recibidas, para así completar la FFT bidimensional. El proceso 0 se encarga de distribuir y unir la información de todos los procesos, transfiriendo la cantidad necesaria de datos a los demás nodos involucrados.

3.3.2 Convolución cíclica bidimensional con 4 nodos.

Este algoritmo tiene un principio similar al SIMD, dividiendo los datos en 4 procesos. Al aplicar primero la transformada bidimensional de Fourier cada proceso ejecuta transformadas unidimensionales a filas y columnas, luego los procesos hacen una multiplicación punto a punto, y al final cada uno aplica la IFFT por filas y columnas a los datos recibidos, así se completa la convolución cíclica.

4 RESULTADOS DE LAS IMPLEMENTACIONES DE CÓDIGO

Estos resultados se tuvieron en cuenta para 30 pruebas, no se hicieron más debido a la convergencia de los resultados y al tiempo de computación que se requiere para más de 30 pruebas. Otro factor es la carga en el procesador que oscila entre 92% y 95%, porque el programa se ejecuta bajo un sistema operativo que requiere memoria y ciclos de procesador para procesos esenciales del sistema. Se minimizó el consumo de recursos del procesador quitando aplicaciones y procesos no esenciales de la memoria, como el protector de pantalla y programas secundarios que consumen recursos. En cada computador se dejaron funcionando la misma cantidad de procesos esenciales del sistema operativo, y con esto se obtienen unas condiciones homogéneas en los equipos utilizados para la implementación.

Cabe aclarar que los datos utilizados son del tipo double, el cual en la versión del compilador utilizado representa una celda de 8 bytes, la cantidad de memoria RAM utilizada por cada proceso se calcula en base a este valor, además hay que decir que los valores a representar son números complejos, los cuales ocupan 2 celdas de memoria para representar la parte real y la imaginaria del valor numérico.

4.1 RESULTADOS DE LAS IMPLEMENTACIONES EN FORMA SECUENCIAL

4.1.1 Resultados de los tiempos de cómputo de la FFT, la IFFT y el cómputo de los factores W.

Tabla 6. Tiempos de cómputo de la FFT, la IFFT y factores W.

Tamaño	Tiempo en realizar la FFT (s)	Tiempo en realizar la IFFT (s)	Tiempo computación de factores W (s)
32	0,000008	0,000008	0,000003
64	0,000016	0,000017	0,000007
128	0,000035	0,000037	0,000013
256	0,000077	0,000081	0,000029
512	0,000163	0,000172	0,000056
1024	0,000342	0,000360	0,000115
2048	0,000728	0,000770	0,000234
4096	0,001624	0,001852	0,000515
8192	0,003384	0,003570	0,000973
16384	0,007229	0,007475	0,001984
32768	0,014902	0,015626	0,004043

Estas operaciones básicas tienen una implementación secuencial, estas no se implementan en forma paralela, pero se utilizan para construir los algoritmos paralelos. En la Tabla 6 se puede observar la progresión en los tiempos de procesamiento de la transformada unidimensional de Fourier y su inversa en forma secuencial. Cabe notar que también se incluye el tiempo que tardan los coeficientes W en ser calculados, para conocer su influencia en el tiempo de cómputo total. Los coeficientes W podrían calcularse u obtenerse de un archivo, debido a que es un conjunto de datos constante para cada conjunto de datos N, pero se calculan debido a que los tiempos para acceder a un archivo en el disco duro son más lentos que el acceso a la memoria RAM, además en la convolución solo es necesario calcular estos coeficientes 2 veces y se pueden reutilizar, esto es porque la matriz tiene 2 dimensiones (filas y columnas) y los coeficientes dependen solo del tamaño del vector y no de su contenido. Hay que observar también la gran similitud entre los tiempos de la

FFT e IFFT ya que son operaciones casi idénticas excepto que la IFFT tiene una operación más.

4.1.2 Resultados de los tiempos de cómputo de la Transformada Rápida de Fourier bidimensional directa e inversa.

Tabla 7. Tiempos de cómputo de la FFT 2D directa e inversa.

Tamaño	Tiempo de computo de la FFT 2D (s)	Tiempo de computo de la IFFT 2D (s)	Memoria usada por el proceso
32x32	0,000304	0,000335	16 KB
64x64	0,001404	0,001492	64 KB
128x128	0,006441	0,006951	256 KB
256x256	0,027371	0,029424	1 MB
512x512	0,157234	0,167201	4 MB
1024x1024	0,661857	0,701845	16 MB
2048x2048	2,790950	2,949228	64 MB
4096x4096	11,928767	12,593175	256 MB
8192x8192	66,914422	69,590888	1 GB

Se observa en la Tabla 7 los tiempos de cómputo de la FFT 2D e IFFT 2D. Además se puede observar el consumo de memoria, que es una limitante importante, ya que los computadores para este trabajo tienen 2 GigaBytes de memoria RAM, eso hace que el tamaño de filas y columnas sea tomado hasta 8192. Para un valor mayor a este, el computador utiliza memoria virtual, haciendo más lento e inestable el proceso debido a que tiene que acceder al disco duro para leer y escribir.

En la Figura 16 se puede observar el comportamiento exponencial de la FFT 2D, que corresponde a la forma que teóricamente debería comportarse: N*logN.

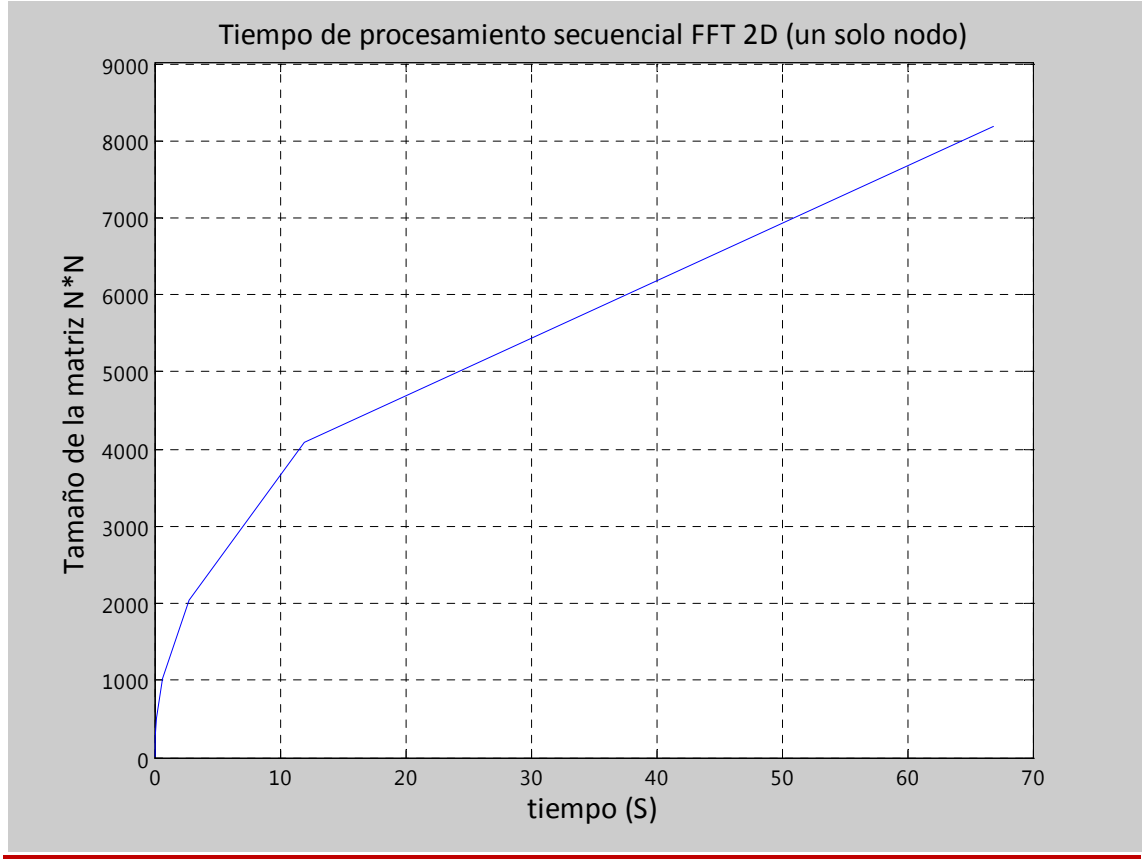

Figura 16. Tiempo de procesamiento FFT 2D secuencial.

Fuente: autores del proyecto

4.1.3 Resultados de los tiempos de cómputo de la convolución cíclica bidimensional.

En la Tabla 8 se presentan los resultados de los tiempos de cómputo requeridos para obtener la convolución secuencial y la cantidad de memoria requerida. Las matrices grandes requieren elevados tiempos de procesamiento porque se procesa una gran cantidad de datos. Cuando el tamaño es muy grande podemos observar el elevado consumo de memoria y para la última cantidad de datos se usa toda la memoria del nodo, lo cual implica que debe estar utilizando memoria virtual para responder a otras tareas, luego la capacidad de la memoria disponible limita a matrices con dimensión 4096x4096.

Tabla 8. Tiempos de cómputo de la convolución cíclica 2D.

Tamaño	tiempo total (s)	Memoria usada por el proceso
32x32	0,000990	32 KB
64x64	0,004356	128 KB
128x128	0,020337	512 KB
256x256	0,085615	2 MB
512x512	0,490791	8 MB
1024x1024	2,081436	32 MB
2048x2048	8,765373	128 MB
4096x4096	37,012683	512 MB
8192x8192	607,140093	2 GB

4.2 RESULTADOS DE LAS IMPLEMENTACIONES EN FORMA PARALELA

4.2.1 Resultados de la implementación de la transformada bidimensional de Fourier en forma paralela con 2 nodos.

Tabla 9. Tiempos de cómputo de la Transformada de Fourier bidimensional con 2 nodos.

Tamaño de la matriz	Tiempo total (s)	Tiempo de comunicación (s)	Tiempo de computo (s)	Memoria máxima usada por el proceso 1	Porcentaje tiempo comunicación (%)	Mejora del tiempo de cómputo respecto al algoritmo secuencial (%)
32x32	0,000743	0,000534	0,000209	8 KB	71,87	31,25
64x64	0,002702	0,001482	0,001220	32 KB	54,85	13,11
128x128	0,034456	0,030502	0,003954	128 KB	88,52	38,61
256x256	0,081888	0,062216	0,019672	0,5 MB	75,98	28,13
512x512	0,105678	0,033089	0,072589	2 MB	31,31	53,83
1024x1024	0,448589	0,054570	0,394020	8 MB	12,16	40,47
2048x2048	2,043112	0,208379	1,834733	32 MB	10,20	34,26
4096x4096	8,531745	0,773960	8,068613	128 MB	9,07	32,36
8192x8192	36,820431	3,724969	33,095462	0,5 GB	10,12	50,54

Se puede observar en la Tabla 9 el tiempo de cómputo para obtener la transformada bidimensional de Fourier paralela. Se encuentra una comparación

de los tiempos que toma el cálculo para 2 nodos, con los cuales hay una mejora promedio del 35,84 %, en comparación con el algoritmo secuencial. El proceso 1 requiere la mitad de memoria RAM que el proceso 0 utiliza y los tiempos de comunicación son menores de 4 segundos debido a la red que se utiliza (el bus de datos de la conexión de un procesador doble núcleo).

A medida que aumenta el tamaño de la matriz el tiempo de procesamiento aumenta y el tiempo de comunicación también, pero el tiempo de procesamiento se hace más grande que el tiempo de comunicación, haciendo que el tiempo de comunicación sea menor del 15 % del tiempo total.

4.2.2 Resultados de la implementación de la transformada bidimensional de Fourier en forma paralela con 4 nodos.

Tabla 10. Tiempos de cómputo de la Transformada bidimensional de Fourier con 4 nodos.

Tamaño de la matriz	Tiempo total (s)	Tiempo de comunicación (s)	Tiempo de computo (s)	Memoria máxima usada por los procesos 1, 2 y 3.	Porcentaje tiempo comunicación (%)	Mejora del tiempo de cómputo respecto al algoritmo secuencial (%)
32x32	0,011822	0,011564	0,000258	4 KB	97,82	15,13
64x64	0,011624	0,010909	0,000715	16 KB	93,85	49,07
128x128	0,010798	0,008117	0,002681	64 KB	75,17	58,38
256x256	0,038144	0,027067	0,011077	0,25 MB	70,96	59,53
512x512	0,150673	0,107039	0,043634	1 MB	71,04	72,25
1024x1024	0,651334	0,368309	0,283024	4 MB	56,55	57,24
2048x2048	2,692598	1,421131	1,271467	16 MB	52,78	54,44
4096x4096	11,018513	5,709276	5,309237	64 MB	51,82	55,49
8192x8192	45,604615	22,941204	22,663411	0,25 GB	50,30	66,13

Para cuatro nodos hay una mejora promedio del 54,18 %, en comparación con el algoritmo secuencial. En la Tabla 10 se observa el consumo de memoria, cada proceso debe utilizar una zona física de la RAM para almacenar los datos, el proceso 0 o nodo maestro se encarga de distribuir los datos, entonces utiliza mas memoria RAM que los otros procesos, al almacenar todos los datos para

tenerlos disponibles. Es obvio que al aumentar la cantidad de procesadores cada nodo requerirá de una fracción de memoria más pequeña, porque procesara una cantidad menor de datos, los procesos 1, 2 y 3 requieren una cuarta parte de la memoria RAM que el proceso 0 utiliza.

Los tiempos de comunicación con 4 procesos son elevados respecto a la implementación con 2 nodos ya que se usa la red Ethernet que para este trabajo tiene una tasa de transferencia máxima de 100Mbits/seg.

4.2.3 Resultados de la implementación de la convolución cíclica bidimensional en forma paralela con 2 nodos.

Convolución cíclica tipo MISD.

En la Tabla 11 se puede observar los tiempos de cómputo de la convolución bidimensional utilizando el método MISD, con el cual se obtiene una mejora promedio de 39,86 %, en comparación con el algoritmo secuencial. Se observa el consumo de memoria, el proceso 0 o nodo maestro se encarga de distribuir los datos, entonces utiliza el doble de memoria RAM que el proceso 1, al almacenar todos los datos para tenerlos disponibles.

Los tiempos de comunicación para las matrices de mayor tamaño no son una parte considerable del tiempo total, debido a que se utiliza el bus de datos del procesador dual core para la comunicación entre los 2 nodos

Tabla 11. Convolución bidimensional con la implementación MISD.

Tamaño de la matriz	Tiempo total (s)	Tiempo de comunicación (s)	Tiempo de computo (s)	Memoria máxima usada por el proceso 1	Porcentaje tiempo comunicación (%)	Mejora del tiempo de cómputo respecto al algoritmo secuencial (%)
32x32	0,002479	0,001840	0,000639	16 KB	74,22	35,45
64x64	0,004847	0,002257	0,002590	64 KB	46,56	40,54
128x128	0,014214	0,003186	0,011029	256 KB	22,41	45,77
256x256	0,062199	0,010692	0,051506	1 MB	17,19	39,84
512x512	0,278172	0,050340	0,227832	4 MB	18,10	53,58
1024x1024	1,580770	0,207001	1,373769	16 MB	13,09	34,00
2048x2048	6,500952	0,727905	5,773046	64 MB	11,20	34,14
4096x4096	26,749452	2,878519	23,870932	256 MB	10,76	35,61

Convolución cíclica tipo SIMD.

En la Tabla 12 se observa los tiempos de cómputo de la convolución bidimensional utilizando el método SIMD, con el cual se obtiene una mejora promedio de 41,60 %, en comparación con el algoritmo secuencial.

El nodo 1 utiliza la mitad de memoria RAM que el nodo maestro (nodo 0).

Tabla 12. Convolución bidimensional con la implementación SIMD.

Tamaño de la matriz	Tiempo total (s)	Tiempo de comunicación (s)	Tiempo de computo (s)	Memoria máxima usada por el proceso 1	Porcentaje tiempo comunicación (%)	Mejora del tiempo de cómputo respecto al algoritmo secuencial (%)
32x32	0,002598	0,001922	0,000677	16 KB	73,98	31,62
64x64	0,005274	0,002698	0,002576	64 KB	51,16	40,86
128x128	0,014195	0,003600	0,010595	256 KB	25,36	47,90
256x256	0,059238	0,011369	0,047869	1 MB	19,19	44,09
512x512	0,305651	0,047447	0,258204	4 MB	15,52	47,39
1024x1024	1,412436	0,170373	1,242064	16 MB	12,06	40,33
2048x2048	5,920931	0,670289	5,250642	64 MB	11,32	40,10
4096x4096	24,627525	2,635826	21,991698	256 MB	10,70	40,58

4.2.4 Resultados de la implementación de la convolución cíclica bidimensional en forma paralela con 4 nodos.

En la Tabla 13 se observa los tiempos de cómputo de la convolución bidimensional utilizando 4 nodos, se obtiene una mejora promedio de 56,84 %, en comparación con el algoritmo secuencial. El proceso 0 o nodo maestro se encarga de distribuir los datos, entonces utiliza 4 veces más memoria RAM que los otros procesos, al almacenar todos los datos para tenerlos disponibles.

Tabla 13. Convolución bidimensional con la implementación de 4 nodos.

Tamaño de la matriz	Tiempo total (s)	Tiempo de comunicación (s)	Tiempo de computo (s)	Memoria máxima usada por el proceso 1, 2 y 3	Porcentaje tiempo comunicación (%)	Mejora del tiempo de cómputo respecto al algoritmo secuencial (%)
32x32	0,003911	0,003362	0,000549	8 KB	85,96	44,55
64x64	0,009148	0,007191	0,001957	32 KB	78,61	55,07
128x128	0,028482	0,019883	0,008598	128 KB	69,81	57,72
256x256	0,131422	0,098855	0,032567	0,5 KB	75,22	61,96
512x512	0,471271	0,335482	0,135789	2 MB	71,19	72,33
1024x1024	2,305039	1,336809	0,968230	8 MB	58,00	53,48
2048x2048	9,359290	5,305511	4,053779	32 MB	56,69	53,75
4096x4096	37,468448	21,151301	16,317147	128 MB	56,45	55,91

4.3 COMPARACIÓN DE RESULTADOS ENTRE LAS IMPLEMENTACIONES SECUENCIAL Y PARALELA

4.3.1 Comparación de tiempos de la transformada bidimensional de Fourier.

En la Figura 17 se observa los tiempos de procesamiento de cada implementación de la FFT 2D. Se observa una mejora en los tiempos a medida que se aumenta el número de nodos (Observar tablas 7, 9 y 10).

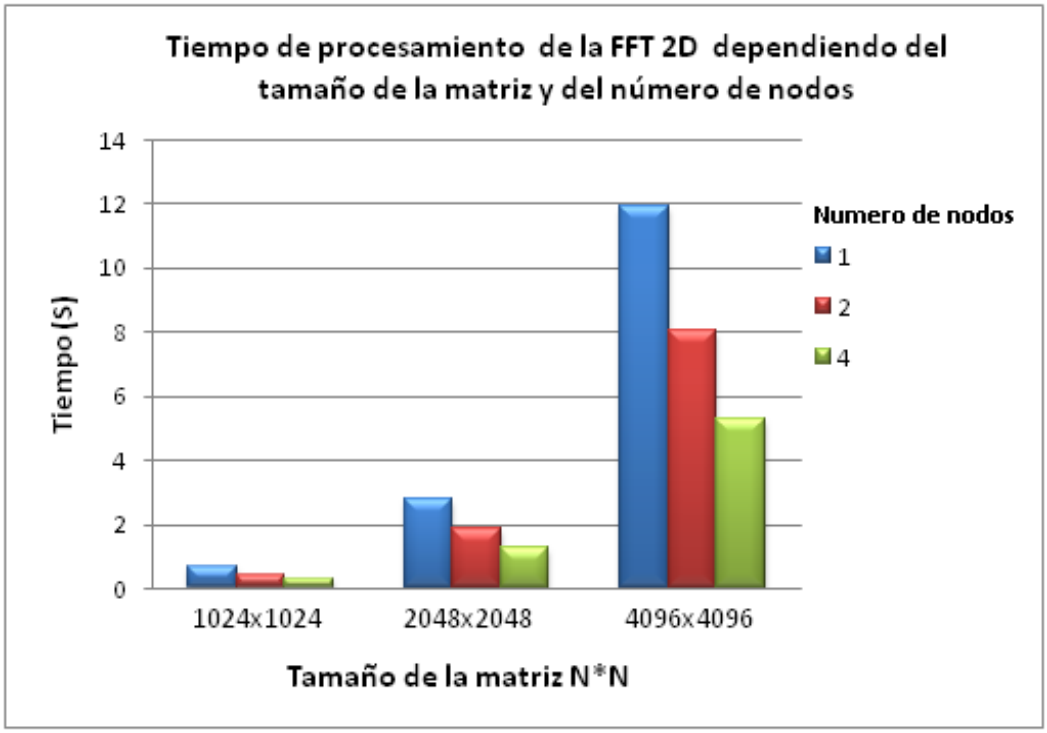

Figura 17. Diagrama de barras de la FFT 2D en función del número de nodos. Fuente: autores del proyecto

En la Figura 18 se observa la comparación de tiempos entre las implementaciones del algoritmo de la FFT 2D, dando mejores resultados la implementación sobre 4 nodos, el algoritmo secuencial es mejorado por las implementaciones del código en paralelo. El eje Y de la grafica en cuestión representa el tamaño tanto de filas como columnas y existe una diferencia considerable entre las 3 graficas, a medida que aumentamos el tamaño de las matrices es más notable la mejora en los tiempos de procesamiento.

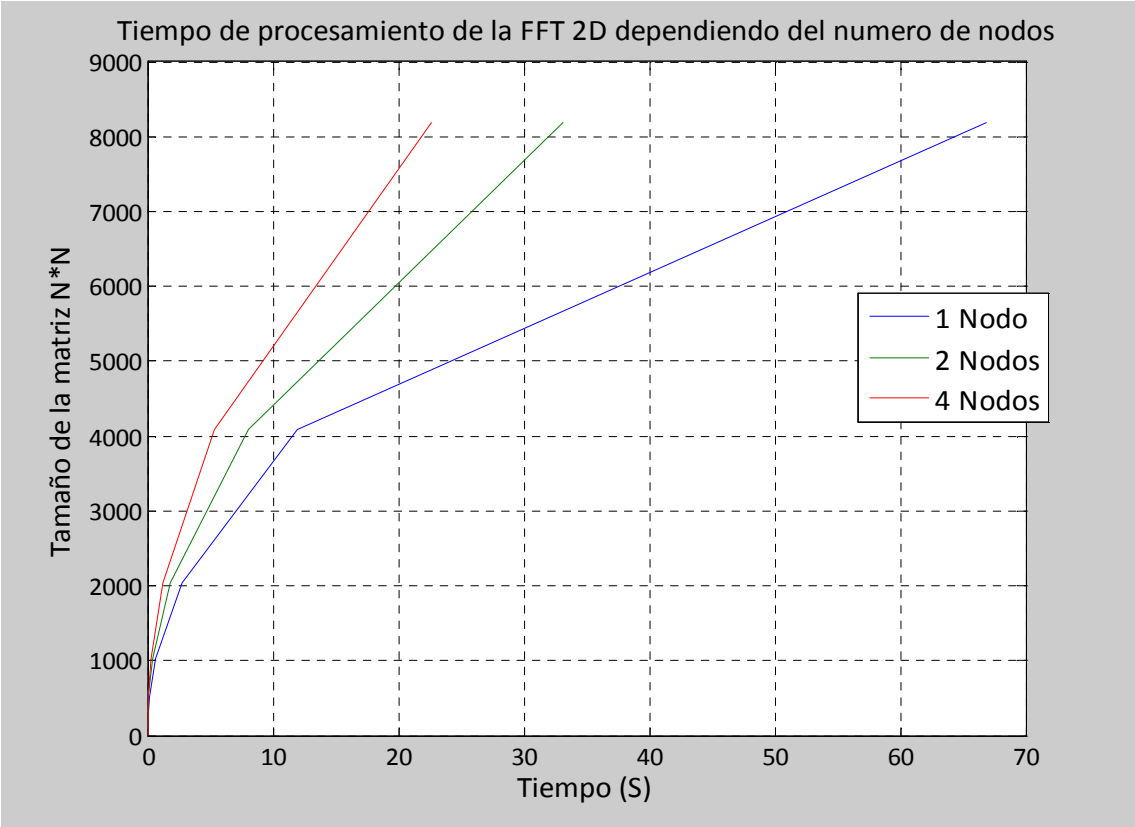

Figura 18. Tiempo de procesamiento de la FFT 2D en función del número de nodos.
Fuente: autores del proyecto

4.3.2 Comparación de tiempos de la convolución cíclica bidimensional.

En las figuras 19 y 20 se observa la comparación de tiempos entre las implementaciones del algoritmo de la convolución cíclica bidimensional, dando mejores resultados la implementación sobre 4 nodos, el algoritmo secuencial es mejorado por las implementaciones del código en paralelo. El eje Y de la Figura 19 representa el tamaño tanto de filas como columnas y existe una diferencia considerable entre las 3 graficas, a medida que se aumenta el tamaño de las matrices es más notable la mejora en los tiempos de procesamiento.

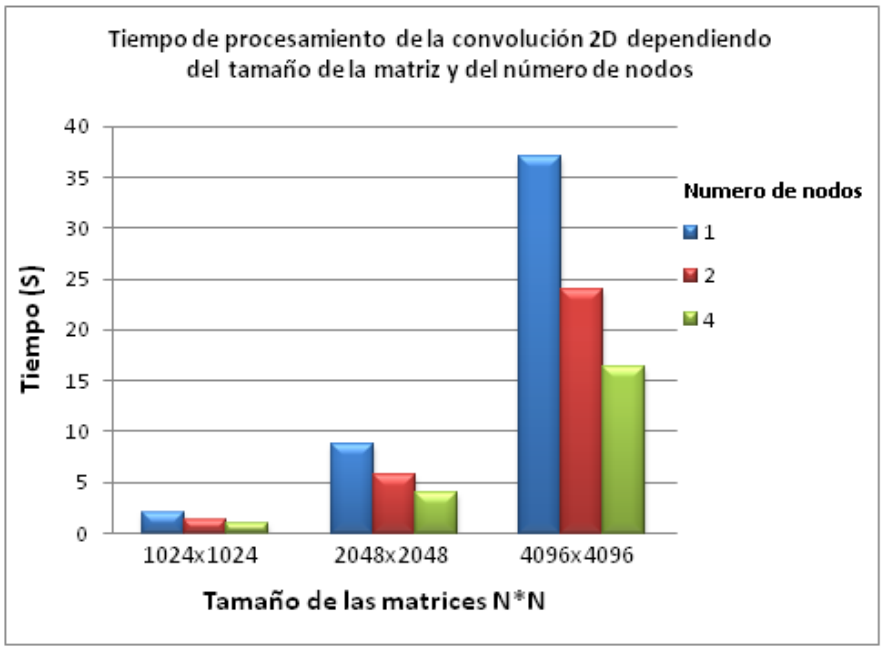

Figura 19.Grafica de barras de la convolución cíclica 2D en función del número de nodos

Fuente: autores del proyecto

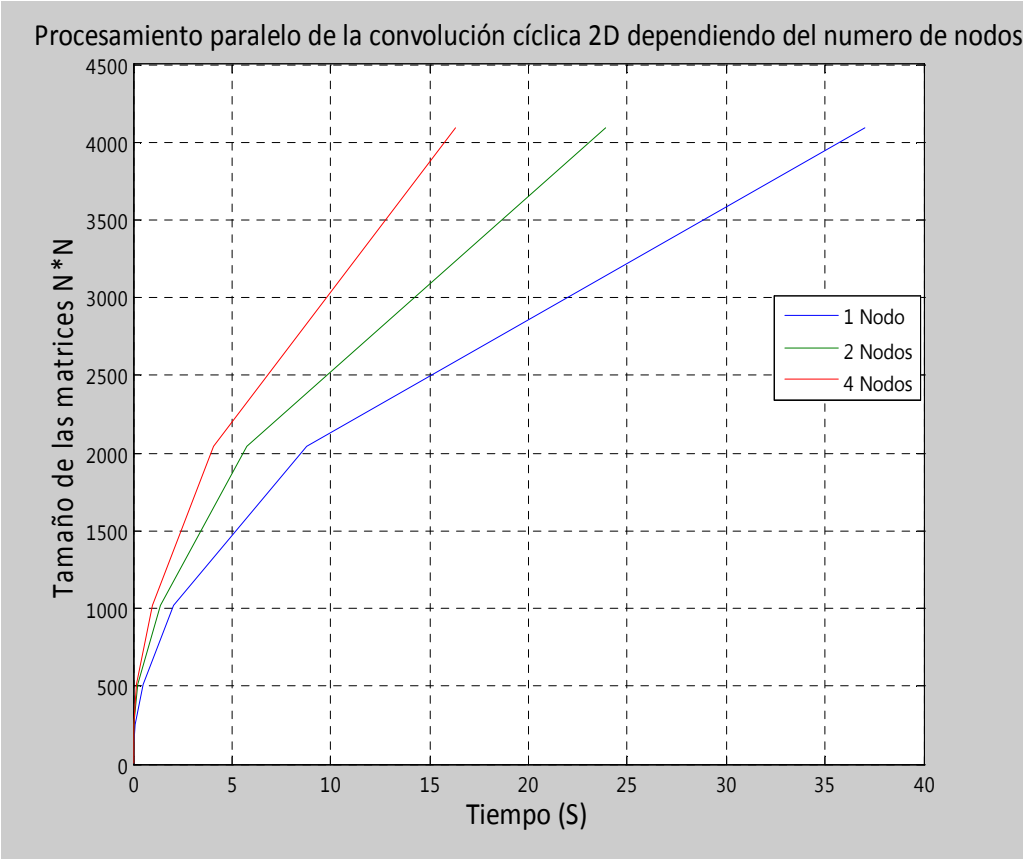

Figura 20. Tiempo de procesamiento de la FFT 2D en función del número de nodos.

Fuente: autores del proyecto

4.4 COMPARACIÓN DE RESULTADOS ENTRE LAS IMPLEMENTACIONES DE LA CONVOLUCIÓN BIDIMENSIONAL EN FORMA PARALELA CON 2 NODOS

En las figuras 21 y 22 se observa la comparación de los 2 métodos de paralelización utilizados en este trabajo para la convolución bidimensional con 2 nodos. Su comportamiento con valores de matrices pequeñas no se diferencian mucho en los tiempos, pero para matrices grandes la forma SIMD presenta mejor desempeño. La figura 22 está compuesta por valores continuos pero es importante recordar que las matrices procesadas tienen los tamaños que se dan en las tablas 11 y 12.

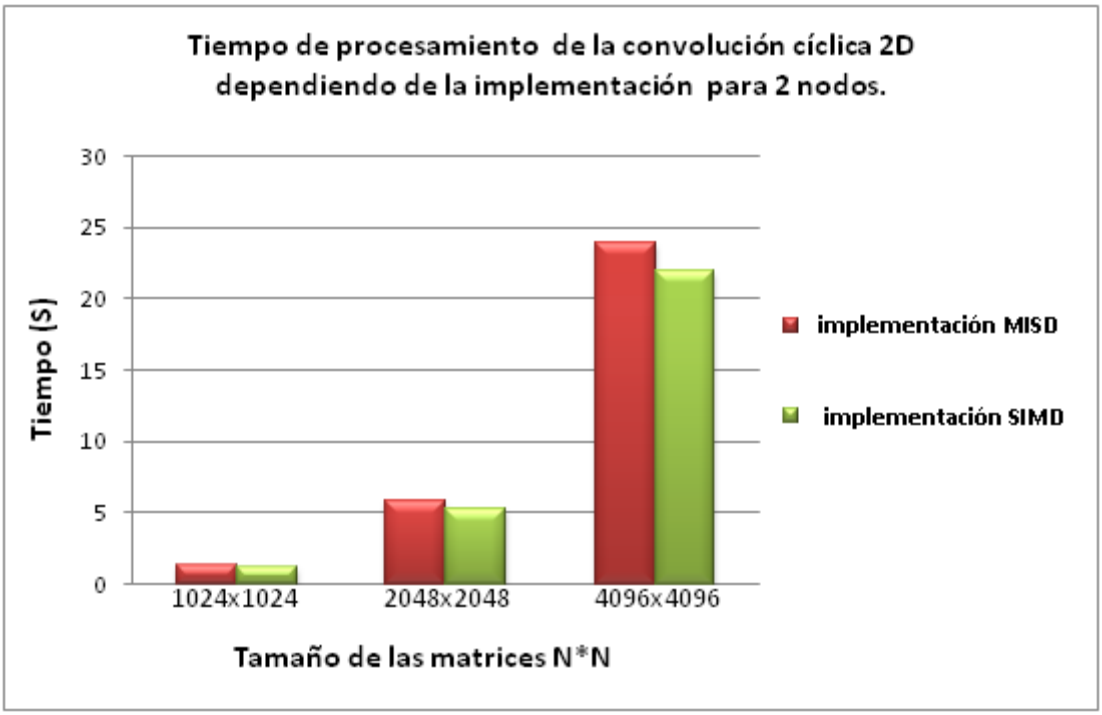

Figura 21. Grafica de barras de la convolución cíclica 2D de la forma MISD y SIMD con 2 nodos.

Fuente: autores del proyecto

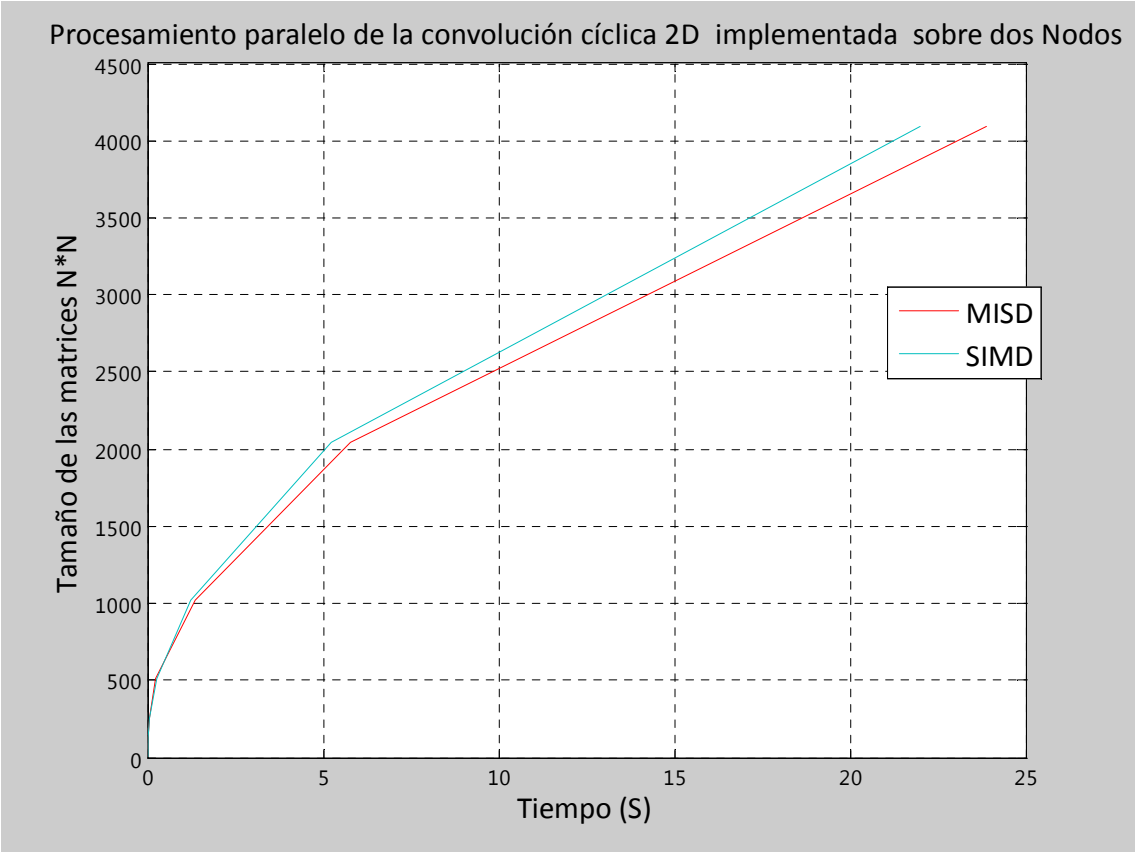

Figura 22. Paralelización de la convolución cíclica 2D de la forma MISD y SIMD con 2 nodos.

Fuente: autores del proyecto

4.5 CONSIDERACIONES DE LA RED USADA

4.5.1 Pruebas de red.

Se hacen pruebas para comprobar el estado de la red, para visualizar el tiempo que tarda la comunicación con diferentes tamaños de paquetes. El objetivo del proyecto no es mejorar el tiempo de comunicación, por tal razón se utiliza una red Ethernet económica para la transmisión de mensajes. Si se utiliza una red de mas alta velocidad (ej: giga Ethernet, infiniband, etc) se pueden reducir los tiempos de comunicación.

En la Figura 23 se observa los resultados de las pruebas de la red Ethernet de 100 Mbits/seg para observar la tasa de transferencia real a medida que se aumenta el tamaño de los paquetes enviados.

Los tiempos de comunicación se deben a las características implícitas de la red como su ancho de banda de 100 MBits/s y el tráfico real en la red, lo cual hace que los tiempos medidos tengan una dispersión considerable.

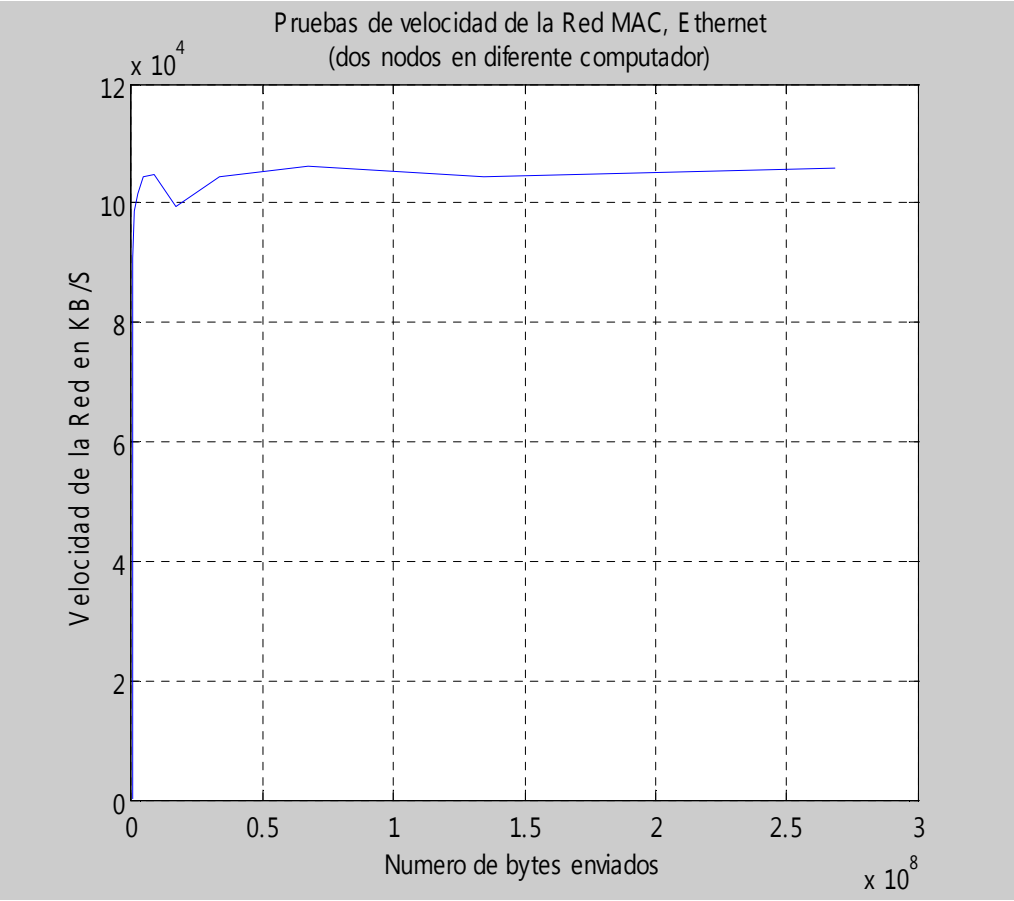

Figura 23. Pruebas de velocidad de la red Ethernet.

Fuente: autores del proyecto

En la Figura 24 se observa los resultados de las pruebas del bus de datos para observar la tasa de transferencia real a medida que se aumenta el tamaño de los paquetes enviados.

Para aprovechar los recursos disponibles se utiliza la capacidad doble núcleo de cada computador, utilizando el bus de datos interno que es mucho más rápido que la red.

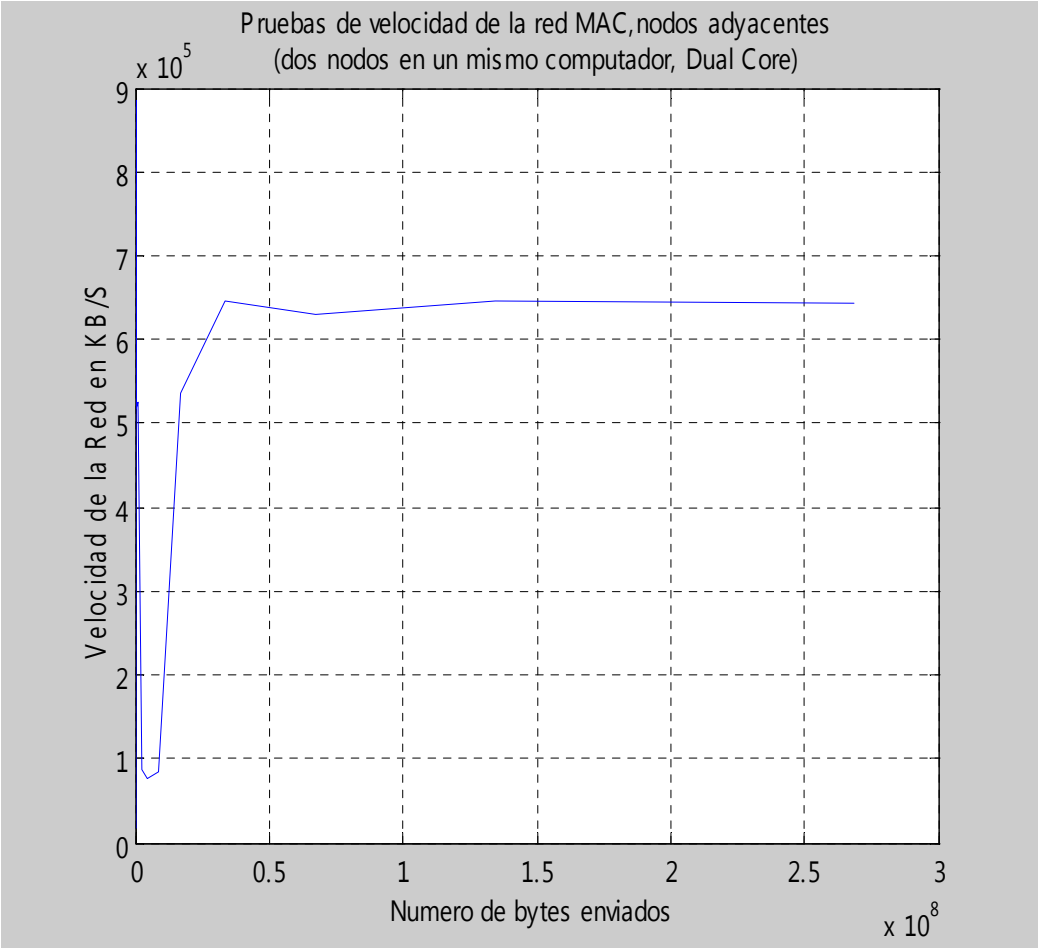

Figura 24. Pruebas de velocidad del bus de datos del procesador dual core.
Fuente: autores del proyecto

4.6 VERIFICACIÓN DE LOS RESULTADOS DE LAS IMPLEMENTACIONES.

Para verificar que el algoritmo de convolución cíclica está funcionando correctamente se validó con Matlab. Se compara una serie de matrices complejas (que son los resultados obtenidos con las implementaciones en el cluster) con las matrices obtenidas en Matlab al efectuarse el algoritmo de convolución cíclica 2D. Las matrices de entrada para la validación de

resultados se obtienen de forma aleatoria por medio de código. En el anexo D se observa el código fuente que se utiliza para comparar los resultados de la implementación paralela y la de Matlab. Se obtuvieron de la comparación resultados satisfactorios ya que las matrices procesadas en Matlab dan resultados idénticos a la implementación en paralelo para la convolución cíclica bidimensional y la transformada rápida de Fourier bidimensional.

5 CONCLUSIONES

- Se implementó exitosamente el algoritmo de la transformada rápida de Fourier bidimensional en paralelo utilizando 2 y 4 nodos, para un conjunto de matrices de dimensiones 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2048x2048, 4096x4096 y 8192x8192.

- Se implementó exitosamente el algoritmo de la convolución cíclica bidimensional en paralelo utilizando 2 y 4 nodos, para un conjunto de matrices de dimensiones 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2048x2048 y 4096x4096.

- Se implementó exitosamente la convolución cíclica bidimensional con 2 métodos de paralelización MISD y SIMD, aplicados a 2 nodos y para un conjunto de matrices de dimensiones 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2048x2048 y 4096x4096.

- Para la convolución cíclica bidimensional implementada sobre 2 nodos se comparó la implementación con el método MISD y el método SIMD. Analizando los resultados obtenidos se concluye que la SIMD presenta un mejor desempeño en tiempo de procesamiento.

- Se validó la implementación de la convolución cíclica bidimensional sobre 2 nodos mediante matrices complejas generadas por código y se compararon los resultados con la implementación realizada en Matlab.

- Para la transformada rápida de Fourier bidimensional se compararon los tiempos de cómputo de la implementación con 1, 2 y 4 nodos y se concluyó que la implementación con 4 nodos presenta el mejor desempeño, además

las implementaciones en paralelo mejoran el tiempo de procesamiento comparado con la implementación secuencial.

- Para la convolución cíclica bidimensional se compararon los tiempos de cómputo de la implementación con 1, 2 y 4 nodos y se concluyó que la implementación con 4 nodos presenta el mejor desempeño, además las implementaciones en paralelo mejoran el tiempo de cómputo comparado con la implementación secuencial.

- A medida que se aumenta el tamaño de las matrices para las implementaciones paralelas de los algoritmos es más notable la mejoría en cuanto a tiempos de cómputo.

- Cuando se trabajan matrices pequeñas se tienen un desempeño limitado por los tiempos de comunicación, que son comparables con los tiempos de cómputo.

- En este trabajo se profundizaron temas de importancia en la ingeniería Electrónica como la transformada de Fourier y la convolución aplicada al procesamiento de los datos, comprendiendo la utilidad de esta herramienta en el ámbito profesional.

- El trabajo se desarrollo en un ambiente de colaboración y apoyo entre los integrantes del proyecto. Se aprendió a trabajar en equipo de forma óptima y a concertar entre distintos puntos de vistas y perspectivas del problema, concluyendo que se debe formar al estudiante con los conocimientos y habilidades necesarios para el trabajo grupal.

6 RECOMENDACIONES

- Se recomienda utilizar en futuras implementaciones equipos con mayor memoria RAM, ya que para la implementación de la convolución se alcanzo a llegar a una matriz de 4096x4096 debido a las limitaciones en la memoria RAM.

- Se recomienda tener una red de comunicación Gigabit Ethernet o superior para implementaciones donde se trabaje mayor cantidad de equipos y el tiempo de comunicación sea un factor crítico.

- Para una futura implementación de los algoritmos con más nodos se recomienda utilizar la forma SIMD debido a su mejor desempeño comparado con la forma MISD.

- Para realizar la convolución cíclica con más nodos se recomienda dividir las filas y columnas en el numero de nodos disponibles P, haciendo que cada nodo procese N/P filas y N/P columnas teóricamente reduciendo el tiempo de procesamiento del algoritmo P veces. Para aplicar este postulado P y N deben ser una potencia de 2.

- Para trabajar MPI sobre los computadores MAC la opción más cómoda y flexible es utilizar el software Pooch, el cual es libre y tiene fácil instalación e interfaz de usuario, lo que nos permite trabajar sin estar utilizando la línea de comandos ni modificar archivos del sistema operativo.

BIBLIOGRAFIA

[1] Aguilar Luis Joyanes, Zahonero Martínez Ignacio, “Programación en C, Metodología, algoritmos y estructura de datos”, Departamento de lenguajes y sistemas infórmaticos e Ingeniería del software Facultad de informática/Escuela universitaria de infórmatica, Universidad Pontificia de Salamanca Campus Madrid, Editorial Mc Graw Hill, (2003).

[2] Chassaing Rulph, “Digital Signal Processing and applications with the C6713 and C6416 DSK”, Wiley interscience, 2005.

[3] García de Jalón de la Fuente Javier, Rodríguez Garrido José Ignacio, Lasheras Goñi Rufino, Brazález Guerra Alfonso, Funes Martínez Patxi y Rodríguez Tamayo Rubén, “Aprenda Lenguaje Ansi C como si estuviera en primero”, Escuela superior de Ingeníeros Industriales Universidad de Navarra, San Sebastian, febrero de 1998.

[4] González R. y Woods R. “Tratamiento digital de imágenes”, Addison–Wesley Iberoamericana S. A. USA, 1996.

[5] Linares V. y Tejedor J. “Implementación del algoritmo de formación de imágenes SAR en paralelo usando procesadores digitales de señales”, Trabajo de Título de Ingeniería Electrónica, Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones, Universidad Industrial de Santander Bucaramanga, Colombia. (2008).

[6] Mazo Manuel y Bergasa Luis, “*Procesamiento* de imágenes en el dominio del espacio y la frecuencia (Aplicaciones en robótica)”, Departamento de Electrónica Universidad de Alcalá, 20 de marzo del 2003.

[7] Mejía Vilet José Ramón, “Apuntes de Procesamiento Digital de Imágenes”, Área de Computación e Informática de la Facultad de Ingeniería de la Universidad Autónoma de San Luis Potosí, 2006.

[8] Oppenheim Alan V, Willsky Alan S. y Nawab S. Hamid, “Señales y sistemas”, segunda edición, Prentice Hall, (1998).

[9] Proakis John G. Y Manolakis Dimitris G. “Digital Signal Processing, Principles, algorithms and applications”, Third edition, Prentice-Hall Inc, (1996).

[10] Quinn Michael J. “Parallel Programming in C with MPI and OpenMPI”, McGraw Hill, 2004.

ENLACES EN LA WEB

[11] Escuela Politécnica Nacional, Desarrollo de aplicaciones paralelas para clusters utilizando MPI, Último acceso: lunes 27 julio del 2009.

http://ciecfie.epn.edu.ec/JIEE/historial/XIXJIEE/17Clusters.pdf

[12] Sitio web de Open MPI, Último acceso: lunes 27 julio del 2009.

http://www.open-mpi.org/

[13] Pagina web de Apple, información sobre el sistema operativo MAC OS X v10.5 , Último acceso: lunes 27 julio del 2009.

http://www.apple.com/e/support/leopard/

[14] Información sobre el software Xcode en Apple Developer, Último acceso: lunes 27 julio del 2009.

http://developer.apple.com/TOOLS/Xcode/

[15] Página web del software Pooch (Dauger Research), Último acceso: lunes 27 julio del 2009.

http://daugerresearch.com/pooch/top.shtml

[16] Pagina web Apple, información sobre puertos de (E/S) para el Mac mini, Último acceso: lunes 27 julio del 2009.

http://support.apple.com/kb/HT3019?viewlocale=es_ES&locale=es_ES

[17] MPI sobre C, teoría y aplicaciones, Último acceso: lunes 27 julio del 2009.

http://www.eead.csic.es/compbio/material/intensiva/node4.html

[18] Información sobre implementación de un cluster Pooch-open mpi, Último acceso: lunes 27 julio del 2009.

http://daugerresearch.com/vault/compilingMPI.shtml

ANEXO A. FUNCIONES SECUENCIALES

- **Algoritmo para calcular los factores W**

```
// Esta función solo requiere la mitad del vector, la otra mitad se coloca en ceros, debido a la  // simetría de los factores W.
// Si la FFT es de n puntos, el vector W a trabajar es de n/2 puntos...


void compute W(int n, double *W_re, double *W_im)
{           int i, br;
            int log2n = log_2(n);
          for (i=0; i<(n/2); i++)
          {           br = bitrev(i,log2n-1);
                    W_re[br] = cos(((double)i*2.0*M_PI)/((double)n));
                    W_im[br] = -sin(((double)i*2.0*M_PI)/((double)n));   }
}
/// Función anexa para aplicar bit reversal
void permutar(int n, double *A_re, double *A_im)
{           int i, bri, log2n;
          double t_re, t_im;
          log2n = log_2(n);
          for (i=0; i<n; i++)
```

```
        {           bri = bitrev(i, log2n);
                if (bri <= i) continue;
                t_re = A_re[i];
                t_im = A_im[i];
                A_re[i]= A_re[bri];
                A_im[i]= A_im[bri];
                A_re[bri]= t_re;
                A_im[bri]= t_im;
        }
}
```

- **Algoritmo de la transformada de Fourier unidimensional**

```
//FFT - DIT decimación en tiempo, utilizando radix-2.
// N debe ser potencia de 2...
void fft(int n, double *A_re, double *A_im, double *W_re, double *W_im)
{           double w_re, w_im, u_re, u_im, t_re, t_im;
        int m, g, b;
        int i, mt, k;
        for (m=n; m>=2; m=m>>1)
        {
                // m = n/2^s; mt = m/2;
                mt = m >> 1;
                // for each group of butterfly
```

```
                for (g=0,k=0; g<n; g+=m,k++)
                {
                        w_re = W_re[k];
                        w_im = W_im[k];

                        //for each butterfly
                        for (b=g; b<(g+mt); b++)
                        {
                                t_re = w_re * A_re[b+mt] - w_im * A_im[b+mt];
                                t_im = w_re * A_im[b+mt] + w_im * A_re[b+mt];
                                u_re = A_re[b];
                                u_im = A_im[b];
                                A_re[b] = u_re + t_re;
                                A_im[b] = u_im + t_im;
                                A_re[b+mt] = u_re - t_re;
                                A_im[b+mt] = u_im - t_im;
                        }
                }
        }
}
```

- **Algoritmo de la transformada inversa de Fourier unidimensional**

//este método para hallar la transformada inversa ya teniendo el vector W inverso, aplica la FFT a los puntos y los divide todos sobre N

```
void ifft(int n, double *A_re, double *A_im, double *W_re, double *W_im)
{           int i;
        fft(n,A_re,A_im,W_re,W_im);
        permutar(n, A_re, A_im);
for (i=0;i<n;i++){
                A_re[i]=A_re[i]/n;
                A_im[i]=A_im[i]/n;
        }
}
```

- **Algoritmo de la FFT bidimensional**

// Función que calcula la FFT bidimensional de una matriz con su matriz compleja calculando los factores W

// Esta optimizada aprovechando el tamaño n/2 del vector W y computa W solo 2 veces y la utiliza recursivamente

// Redimensiona W para columnas y filas y al final lo destruye para ahorrar memoria

```
void fft2Dsecuencial(int f,int col,double *MA_re,double *MA_im)
{           int i,j;
        double *tempi,*tempr,*Wr,*Wi;
```

```
//asigna espacio en memoria para los vectores
tempr=calloc(col,sizeof(double));
tempi=calloc(col,sizeof(double));
Wr=calloc(col/2,sizeof(double));
Wi=calloc(col/2,sizeof(double));
//Comienza con la FFT a cada fila
compute_W(col,Wr,Wi); //calcula una única vez el vector W para las filas
for(i=0;i<f;i++){
        for(j=0;j<col;j++ ){
                tempi[j]=*(MA_im+i*col+j);
                tempr[j]=*(MA_re+i*col+j);
        }

        fft(col,tempr,tempi,Wr,Wi);
        permutar(col,tempr,tempi);
        for(j=0;j<col;j++ ){
                *( MA_im+i*col+j)=tempi[j];
                *( MA_re+i*col+j)=tempr[j];
        }
}
//comienza la FFT por columnas
```

```
//asigna espacio en memoria para los vectores y redimensiona el vector W y el vector temporal

free(Wr);

free(Wi);

free(tempr);

free(tempi);

tempr=calloc(f,sizeof(double));

tempi=calloc(f,sizeof(double));

Wr=calloc(f/2,sizeof(double));

Wi=calloc(f/2,sizeof(double));

compute_W(f,Wr,Wi); //  calcula solo una vez el vector W para columnas

for(i=0;i<col;i++){

        for(j=0;j<f;j++ ){

                tempi[j]=*(MA_im+j*col+i);

                tempr[j]=*(MA_re+j*col+i);

        }

        fft(f,tempr,tempi,Wr,Wi);

        permutar(f,tempr,tempi);

        for(j=0;j<f;j++ ){

                *( MA_im+j*col+i)=tempi[j];

                *( MA_re+j*col+i)=tempr[j];

        }
```

```
        }
        free(Wr);
        free(Wi);
        free(tempr);
        free(tempi); //borra y libera el espacio de las variables temporales
}
```

- **Algoritmo de la multiplicación punto a punto de 2 matrices**

```
//esta función hace el producto punto de la matriz compleja A y B del mismo tamaño y lo almacena en la // matriz B
void multiplica (int f,int col,double *Ar,double *Ai,double *Br, double *Bi)
{  double ax,bx,cx,dx;
        int i,j;
 for (i=0;i<f;i++)
        {       for(j=0;j<col;j++)
                {
                        ax=*(Ar+i*col+j);
                        bx=*(Ai+i*col+j);
                        cx=*(Br+i*col+j);
                        dx=*(Bi+i*col+j);
         *(Br+i*col+j)=ax*cx-bx*dx;
         *(Bi+i*col+j)=bx*cx+ax*dx;
                }
```

```
        }
}
```

- **Algoritmo de la convolución cíclica bidimensional**

/// Esta función realiza la convolución en frecuencia de 2 matrices con igual dimensión y la respuesta la da en B....

```
void convolucionsecuencial(int f,int col,double *Ar,double *Ai,double *Br, double *Bi)
{
        fft2Dsecuencial(f, col, Ar, Ai); //transformada 2d de la primera matriz
        fft2Dsecuencial(f, col, Br, Bi); //transformada 2d de la segunda matriz
        multiplica (f,col,Ar,Ai,Br, Bi); //multiplicación punto a punto entre las matrices
        ifft2Dsecuencial(f, col, Br, Bi); //transformada inversa de la matriz resultado
}
/// funcion transpuesta de matricez cuadradas
void transpuesta(int f, int col,double* A){
    double temp;
    int i,j;
    for(i=0;i<f;i++){
       for(j=i;j<col;j++){
    temp=*(A+i*col+j);
```

```
        *(A+i*col+j)=*(A+j*col+i);

        *(A+j*col+i)=temp;

         }
}
}
```

ANEXO B. FUNCIONES PARALELAS PARA 2 NODOS

//estas funciones incluyen la medición de tiempos de comunicación, para usar estas funciones se debe // en el MAIN inicializar MPI y trabajar solo con el proceso 0 y 1....

//MPI debe ser inicializado en el MAIN, y debe existir la restricción para que solo trabaje con 2 procesos //en el main

//estas funciones son hechas para matrices cuadradas múltiplos de 2

// la variable secs mide el tiempo acumulado de comunicación, rank2 define el procesador que acompaña al proceso 0, y rank es el proceso actual

//el espacio de memoria que consumen las matrices en cada nodo debe ser definido en el main con la función calloc

// a esta función solo se le asigna memoria al proceso 0, y el proceso n se le asigna una matriz temporal

//rank2 es el proceso a trabajar con el rango cero, y rank es el rango del proceso actual

// Esta función solo es usada para matrices cuadradas con dimensión de potencias de 2

- **Algoritmo de la transformada de Fourier bidimensional para 2 nodos**

```
void fft2Dparalelo2(int f,int col,double *MA_re,double *MA_im,int rank,int rank2,double* secs)

{ int p1,p2;
```

```
        MPI_Status stat;
        struct timeval xp1,xp2,xp3;
        *secs=0;
        ////////////////////////
        //envió de datos al otro proceso
        if(rank==0){
   gettimeofday(&xp1, NULL);
        MPI_Send(MA_re, col*f/2, MPI_DOUBLE, rank2, 1, MPI_COMM_WORLD); //se envía la mitad de la matriz
   MPI_Send(MA_im, col*f/2, MPI_DOUBLE, rank2, 2, MPI_COMM_WORLD);
   gettimeofday(&xp2, NULL);
   *secs=*secs+timeval_diff(&xp2, &xp1);
        }
        ///recepción de datos del proceso n
        if(rank==rank2){
        MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
   MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD,&stat);
        }
//comienza con la FFT a cada fila del proceso 0 y n
        if(rank==0){
            p1=f/2;
            p2=f;                }
```

```
        if(rank==rank2){
            p1=0;
            p2=f/2;
            }
 //calcula la FFT a cada fila de cualquier proceso
        filasfft(f,col,MA_re,MA_im,p1,p2);
///ahora el proceso cero reúne la información del proceso n
        //envió de datos al otro proceso
        if(rank==rank2){
        MPI_Send(MA_re, col*f/2, MPI_DOUBLE, 0, 3, MPI_COMM_WORLD);
//se envía la mitad de la matriz
    MPI_Send(MA_im, col*f/2, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD);
    ///recepción de datos del proceso 1
    MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, 0, 5, MPI_COMM_WORLD,&stat);
//recibe la mitad de la matriz
    MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, 0, 6, MPI_COMM_WORLD,&stat);
        }
        ///recepción de datos del proceso 0
        if(rank==0){
     gettimeofday(&xp1, NULL);
    MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, rank2, 3,
MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
```

```
MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, rank2, 4, MPI_COMM_WORLD,&stat);

gettimeofday(&xp2, NULL);

*secs=*secs+timeval_diff(&xp2, &xp1);

//el proceso 0 realiza la transpuesta de la matriz, la parte real y la imaginaria....

transpuesta(f,col,MA_re);

transpuesta(f,col,MA_im);

///envió al proceso n la mitad de la matriz de datos

gettimeofday(&xp1, NULL);

MPI_Send(MA_re, col*f/2, MPI_DOUBLE, rank2, 5, MPI_COMM_WORLD); //se envía la mitad de la matriz

MPI_Send(MA_im, col*f/2, MPI_DOUBLE, rank2, 6, MPI_COMM_WORLD);

gettimeofday(&xp2, NULL);

*secs=*secs+timeval_diff(&xp2, &xp1);

    }

    //comienza la FFT por columnas del proceso 0

    if(rank==0){

        p1=col/2;

        p2=col;

        }

  if(rank==rank2){

        p1=0;
```

```
            p2=col/2;

            }

   filasfft(f,col,MA_re,MA_im,p1,p2);

        ///////proceso para enviar la información al proceso 0

if(rank==rank2){


    MPI_Send(MA_re, col*f/2, MPI_DOUBLE, 0, 6, MPI_COMM_WORLD); //se envía la matriz temporal

   MPI_Send(MA_im, col*f/2, MPI_DOUBLE, 0, 7, MPI_COMM_WORLD);

                }

//aquí el proceso 0 recibe y organiza la matriz

if(rank==0){

        gettimeofday(&xp1, NULL);

        MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, rank2, 6, MPI_COMM_WORLD,&stat); //recibe la mitad de la matrix

        MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, rank2, 7, MPI_COMM_WORLD,&stat);

        gettimeofday(&xp2, NULL);

        *secs=*secs+timeval_diff(&xp2, &xp1);

         //realiza la transpuesta de la matriz

         transpuesta(f,col,MA_re);

         transpuesta(f,col,MA_im);

         }
```

```
}
```

- **Algoritmo de la IFFT bidimensional con 2 nodos**

```
// a esta función solo se le asigna memoria al proceso 0, y el proceso n se le asigna una matriz temporal

//rank2 es el proceso a trabajar con el rango cero, y rank es el rango del proceso actual

// Esta función solo es usada para matrices cuadradas con dimensión de potencias de 2

void ifft2Dparalelo2(int f,int col,double *MA_re,double *MA_im,int rank,int rank2,double *secs)

{ int p1,p2;

        MPI_Status stat;

        *secs=0;

        struct timeval xp1,xp2,xp3;

        //////////////////////////

        //envió de datos al otro proceso

        if(rank==0){

    gettimeofday(&xp1, NULL);

        MPI_Send(MA_re, col*f/2, MPI_DOUBLE, rank2, 12, MPI_COMM_WORLD); //se envía la mitad de la matriz

    MPI_Send(MA_im, col*f/2, MPI_DOUBLE, rank2, 13, MPI_COMM_WORLD);

    gettimeofday(&xp2, NULL);

    *secs=*secs+timeval_diff(&xp2, &xp1);

        }
```

```
        ///recepción de datos del proceso n
        if(rank==rank2){
        MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, 0, 12, MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
      MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, 0, 13, MPI_COMM_WORLD,&stat);
        }
        //comienza con la FFT a cada fila del proceso 0
        if(rank==0){
            p1=f/2;
            p2=f;
            }
        if(rank==rank2){
            p1=0;
            p2=f/2;
            }
    //calcula la IFFT a cada fila de cualquier proceso
        filasifft(f,col,MA_re,MA_im,p1,p2);

///ahora el proceso cero reúne la información del proceso n
        //envió de datos al otro proceso
        if(rank==rank2){
```

```
        MPI_Send(MA_re, col*f/2, MPI_DOUBLE, 0, 14, MPI_COMM_WORLD);
//se envía la mitad de la matriz
   MPI_Send(MA_im, col*f/2, MPI_DOUBLE, 0, 15, MPI_COMM_WORLD);
   ///recepción de datos del proceso n
   MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, 0, 16,
MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
   MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, 0, 17,
MPI_COMM_WORLD,&stat);
   }
        ///recepción de datos del proceso 0
        if(rank==0){
    gettimeofday(&xp1, NULL);
   MPI_Recv(MA_re, col*f/2, MPI_DOUBLE, rank2, 14,
MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
   MPI_Recv(MA_im, col*f/2, MPI_DOUBLE, rank2, 15,
MPI_COMM_WORLD,&stat);
    gettimeofday(&xp2, NULL);
    *secs=*secs+timeval_diff(&xp2, &xp1);
    //transpuesta de la matriz
    transpuesta(f,col,MA_re);
    transpuesta(f,col,MA_im);
///envió al proceso n la mitad de la matriz de datos
   gettimeofday(&xp1, NULL);
```

```
    MPI_Send(MA_re, col*f/2, MPI_DOUBLE, rank2, 16, MPI_COMM_WORLD);
//se envía la mitad de la matriz
    MPI_Send(MA_im, col*f/2, MPI_DOUBLE, rank2, 17, MPI_COMM_WORLD);
     gettimeofday(&xp2, NULL);
      *secs=*secs+timeval_diff(&xp2, &xp1);
          }

          //comienza la FFT por columnas del proceso 0
    if(rank==0){
               p1=col/2;
               p2=col;
               }
    if(rank==rank2){
               p1=0;
               p2=col/2;
               }
  filasifft(f,col,MA_re,MA_im,p1,p2);

          ///////proceso para enviar las mitad de las columnas del proceso n al
proceso 0
if(rank==rank2){
           MPI_Send(MA_re, col*f/2, MPI_DOUBLE, 0, 18, MPI_COMM_WORLD);
//se envía la matriz temporal
```

```
    MPI_Send(MA_im, col*f/2, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD);


            }
//aquí el proceso 0 recibe y organiza la matriz
if(rank==0){
        gettimeofday(&xp1, NULL);
        MPI_Recv(MA_re, col*f, MPI_DOUBLE, rank2, 18, MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
        MPI_Recv(MA_im, col*f, MPI_DOUBLE, rank2, 19, MPI_COMM_WORLD,&stat);
        gettimeofday(&xp2, NULL);
       *secs=*secs+timeval_diff(&xp2, &xp1);
       //transpuesta de la matriz
       transpuesta(f,col,MA_re);
       transpuesta(f,col,MA_im);
        }
}
```

- **Algoritmo de la multiplicación punto a punto en paralelo para 2 nodos**

```
//esta función reserva memoria temporal en el otro proceso para procesar la información
void multiplicaparalelo2(int f,int col,double *Ar,double *Ai, double *Br,double *Bi, int rank,int rank2, double *secs){
    double ax,bx,cx,dx;
```

```
int p1,p2;
    int i,j;
    *secs=0;
    struct timeval xp1,xp2,xp3;
    MPI_Status stat;
    ///el proceso 0 envía la mitad superior de la matriz A y B
    if(rank==0){
  gettimeofday(&xp1, NULL);
  MPI_Send(Ar, col*f/2, MPI_DOUBLE, rank2, 20, MPI_COMM_WORLD); //se envía la mitad de A
  MPI_Send(Ai, col*f/2, MPI_DOUBLE, rank2, 21, MPI_COMM_WORLD);
  MPI_Send(Br, col*f/2, MPI_DOUBLE, rank2, 22, MPI_COMM_WORLD); //se envía la mitad de B
  MPI_Send(Bi, col*f/2, MPI_DOUBLE, rank2, 23, MPI_COMM_WORLD);
   gettimeofday(&xp2, NULL);
   *secs=*secs+timeval_diff(&xp2, &xp1);
  p1=f/2;
  p2=f;
        }
  if(rank==rank2){ //reserva espacio temporal para procesar
    MPI_Recv(Ar, col*f/2, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
    MPI_Recv(Ai, col*f/2, MPI_DOUBLE, 0, 21, MPI_COMM_WORLD,&stat);
```

```
    MPI_Recv(Br, col*f/2, MPI_DOUBLE, 0, 22, MPI_COMM_WORLD,&stat);
//recibe la mitad de la matriz

    MPI_Recv(Bi, col*f/2, MPI_DOUBLE, 0, 23, MPI_COMM_WORLD,&stat);

    p1=0;

    p2=f/2;

          }

 for (i=p1;i<p2;i++)

       {

     for(j=0;j<col;j++)

              {

        ax=*(Ar+i*col+j);

        bx=*(Ai+i*col+j);

        cx=*(Br+i*col+j);

        dx=*(Bi+i*col+j);

                  *(Br+i*col+j)=ax*cx-bx*dx;

        *(Bi+i*col+j)=bx*cx+ax*dx;

              }

  }

       ///el proceso n envía la mitad superior de la matriz B

       if(rank==rank2){

       MPI_Send(Br, col*f/2, MPI_DOUBLE, 0, 24, MPI_COMM_WORLD); //se
envía la mitad de B

  MPI_Send(Bi, col*f/2, MPI_DOUBLE, 0, 25, MPI_COMM_WORLD);
```

```
            }
    if(rank==0){
      gettimeofday(&xp1, NULL);
     MPI_Recv(Br, col*f/2, MPI_DOUBLE, rank2, 24, MPI_COMM_WORLD,&stat); //recibe la mitad de la matriz
     MPI_Recv(Bi, col*f/2, MPI_DOUBLE, rank2, 25, MPI_COMM_WORLD,&stat);
      gettimeofday(&xp2, NULL);
     *secs=*secs+timeval_diff(&xp2, &xp1);
             }
    }
```

- **Algoritmo de la convolución cíclica bidimensional para 2 nodos de la forma MISD**

```
void convolucion3paralelo2(int f,int col,double *Ar,double *Ai,double *Br, double *Bi, int rank,int rank2,double *secs){
        //el segundo proceso necesita reservar medio bloque para A y B
    *secs=0;
        double comu1=0; //el tiempo que gasta cada función en comunicarse con otros procesos
        fft2Dparalelo2(f,col,Ar,Ai,rank,rank2,&comu1);
        *secs=*secs+comu1;
    fft2Dparalelo2(f,col,Br,Bi,rank,rank2,&comu1);
        *secs=*secs+comu1;
    multiplicaparalelo2(f,col,Ar,Ai,Br,Bi,rank,rank2,&comu1);
```

```
        *secs=*secs+comu1;
    ifft2Dparalelo2(f,col,Br,Bi,rank,rank2,&comu1);
        *secs=*secs+comu1;
    }
```

- **Algoritmo de la convolución cíclica bidimensional con 2 nodos de la forma SIMD**

```
void convolucion4paralelo2(int f,int col,double *Ar,double *Ai,double *Br, double *Bi, int rank,int rank2,double *secs){
        MPI_Status stat;
        *secs=0;
        double comu1=0;
        struct timeval xp1,xp2,xp3;
        double *MAi,*MAr,*MBr,*MBi;
        //toca reservar aquí ya que el bloque B del segundo proceso cambia de dimensiones en cada operación


        //envía la matriz B al proceso n y procesa la matriz A
        if(rank==0){
                gettimeofday(&xp1, NULL);
                MPI_Send(Br, col*f, MPI_DOUBLE, rank2, 26, MPI_COMM_WORLD); //se envía la matriz B
                MPI_Send(Bi, col*f, MPI_DOUBLE, rank2, 27, MPI_COMM_WORLD);
                gettimeofday(&xp2, NULL);
```

```
                *secs=*secs+timeval_diff(&xp2, &xp1);

                fft2Dsecuencial(f, col, Ar, Ai); //transformada 2d de la primera
matriz

                ///recibe la matriz B procesada

                gettimeofday(&xp1, NULL);

                MPI_Recv(Br, col*f, MPI_DOUBLE, rank2, 28,
MPI_COMM_WORLD,&stat); //recibe la matriz B procesada

                MPI_Recv(Bi, col*f, MPI_DOUBLE, rank2, 29,
MPI_COMM_WORLD,&stat);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

   }

   ///el proceso n recibe la matriz A y la procesa y la devuelve

   if(rank==rank2){

      if((MBr=malloc(col*f*sizeof(double)))==NULL)

                {printf("\n error, no hay memoria suficiente para Ar....");

                }

                if((MBi=malloc(col*f*sizeof(double)))==NULL)

                {printf("\n error, no hay memoria suficiente para Ai....");

                }

MPI_Recv(MBr, col*f, MPI_DOUBLE, 0, 26, MPI_COMM_WORLD,&stat);
//recibe la matriz B

                MPI_Recv(MBi, col*f, MPI_DOUBLE, 0, 27,
MPI_COMM_WORLD,&stat);
```

```
                fft2Dsecuencial(f, col, MBr, MBi); //transformada 2d de la segunda
matriz

MPI_Send(MBr, col*f, MPI_DOUBLE, 0, 28, MPI_COMM_WORLD); //se envía
la matriz B procesada

                MPI_Send(MBi, col*f, MPI_DOUBLE, 0, 29,
MPI_COMM_WORLD);

                free(MBr);

                free(MBi);

                if((MAr=malloc(col*f/2*sizeof(double)))==NULL)

                {printf("\n error, no hay memoria suficiente para Ar....");

                }

                if((MAi=malloc(col*f/2*sizeof(double)))==NULL)

                {printf("\n error, no hay memoria suficiente para Ai....");

                }

                if((MBr=malloc(col*f/2*sizeof(double)))==NULL)

                {printf("\n error, no hay memoria suficiente para Br....");

                }

                if((MBi=malloc(col*f/2*sizeof(double)))==NULL)

                {printf("\n error, no hay memoria suficiente para Bi....");

                }

  }

  if(rank==0){MAr=Ar;

          MAi=Ai;
```

```
            MBr=Br;

            MBi=Bi;

            }

   multiplicaparalelo2(f,col,MAr,MAi,MBr,  MBi,rank,rank2,&comu1);
//multiplicación punto a punto entre las matrices

        *secs=*secs+comu1;

   ifft2Dparalelo2(f,col,MBr,MBi,rank,rank2,&comu1); //los 2 procesos realizan
la IFFT inversa en 2D

        *secs=*secs+comu1;


        if(rank==rank2){//libera la memoria de los procesos

            free(MBr);

            free(MBi);

            free(MAr);

            free(MAi);

            }

}
```

ANEXO C. FUNCIONES PARALELAS CON 4 NODOS

```
//funciones paralelas para 4 procesos de la convolución, 2Dfft, 2Difft..

//estas funciones incluyen la medición de tiempo de comunicación, para usar estas funciones se debe en el main inicializar MPI y trabajar solo con el proceso 0,n1,n2,n3....

//MPI debe ser inicializado en el main, y debe existir la restricción para que solo trabaje con 4 procesos en el main

//estas funciones son hechas para matrices cuadradas de tamaño múltiplo de 2, ojo con la restricción...

// la variable secs mide el tiempo acumulado de comunicación, rank2,rank3 y rank4 define los procesos que acompañan al proceso 0, y rank es el proceso actual

//el espacio de memoria para el proceso 0 debe ser definido en el main y solo para ese proceso ya que para los otros acá se define su espacio de memoria...

//versión final

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <mpi.h>

#include "funciones.h"

double timeval_diff(struct timeval *a, struct timeval *b);
```

void fft2Dparalelo4(int f,int col,double *MA_re,double *MA_im,int rank,int rank2,int rank3,int rank4,double* secs);

void ifft2Dparalelo4(int f,int col,double *MA_re,double *MA_im,int rank,int rank2,int rank3,int rank4,double* secs);

void convolucionparalelo4(int f,int col,double *Ar,double *Ai,double *Br, double *Bi, int rank,int rank2,int rank3,int rank4,double* secs);

void multiplicaparalelo4(int f,int col,double *Ar,double *Ai, double *Br,double *Bi, int rank,int rank2,int rank3,int rank4, double* secs);

- **Algoritmo de la FFT bidimensional para 4 nodos**

```
void fft2Dparalelo4(int f,int col,double *MA_re,double *MA_im,int rank,int rank2,int rank3,int rank4,double* secs)
{ int p1,p2;
        MPI_Status stat;
        struct timeval xp1,xp2,xp3;
        *secs=0;
                //envió de datos a los otros procesos
        if(rank==0){
                gettimeofday(&xp1, NULL);
                MPI_Send(MA_re, col*f/4, MPI_DOUBLE, rank2, 1, MPI_COMM_WORLD);
                MPI_Send(MA_im, col*f/4, MPI_DOUBLE, rank2, 2, MPI_COMM_WORLD);
```

```
                MPI_Send((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 3,
MPI_COMM_WORLD);

                MPI_Send((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 4,
MPI_COMM_WORLD);

                MPI_Send((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 5,
MPI_COMM_WORLD);

                MPI_Send((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 6,
MPI_COMM_WORLD);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

        }

        ///recepción de datos por los otros 3 procesos

   if(rank==rank2){

                MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 1,
MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 2,
MPI_COMM_WORLD,&stat);

        }

        if(rank==rank3){

                MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 3,
MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 4,
MPI_COMM_WORLD,&stat);

        }

        if(rank==rank4){
```

```
            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 5, MPI_COMM_WORLD,&stat);
            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 6, MPI_COMM_WORLD,&stat);
        }
        //comienza con la FFT a cada fila del proceso 0 y los otros 2
        if(rank==0){
            p1=3*(f/4);
            p2=f;
        }
        if(rank!=0){
            p1=0;
            p2=f/4;
        }
        //calcula la fft a cada fila de cualquier proceso
        filasfft(f,col,MA_re,MA_im,p1,p2);
        ///ahora el proceso cero reúne la información de los otros procesos
        //envió de datos al otro proceso
        if(rank==rank2){
            MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 7, MPI_COMM_WORLD);
            MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 8, MPI_COMM_WORLD);
```

```
                ///recepción de datos del proceso

                MPI_Recv(MA_re,      col*f/4,      MPI_DOUBLE,      0,      13,
MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im,      col*f/4,      MPI_DOUBLE,      0,      14,
MPI_COMM_WORLD,&stat);


        }

        if(rank==rank3){

                MPI_Send(MA_re,      col*f/4,      MPI_DOUBLE,      0,      9,
MPI_COMM_WORLD);

                MPI_Send(MA_im,      col*f/4,      MPI_DOUBLE,      0,      10,
MPI_COMM_WORLD);

                ///recepción de datos del proceso

                MPI_Recv(MA_re,      col*f/4,      MPI_DOUBLE,      0,      15,
MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im,      col*f/4,      MPI_DOUBLE,      0,      16,
MPI_COMM_WORLD,&stat);

        }

        if(rank==rank4){

                MPI_Send(MA_re,      col*f/4,      MPI_DOUBLE,      0,      11,
MPI_COMM_WORLD);

                MPI_Send(MA_im,      col*f/4,      MPI_DOUBLE,      0,      12,
MPI_COMM_WORLD);

                ///recepción de datos del proceso
```

```
                MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 18, MPI_COMM_WORLD,&stat);

        }

        ///recepción de datos del proceso 0

        if(rank==0){

                gettimeofday(&xp1, NULL);

                MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, rank2, 7, MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, rank2, 8, MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 9, MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 10, MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 11, MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 12, MPI_COMM_WORLD,&stat);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

                //el proceso 0 realiza la transpuesta de la matriz, la parte real y la imaginaria....

                transpuesta(f,col,MA_re);
```

```
                transpuesta(f,col,MA_im);

                ///envió al proceso n la mitad de la matriz de datos

                gettimeofday(&xp1, NULL);

                MPI_Send(MA_re, col*f/4, MPI_DOUBLE, rank2, 13,
MPI_COMM_WORLD);

                MPI_Send(MA_im, col*f/4, MPI_DOUBLE, rank2, 14,
MPI_COMM_WORLD);

                MPI_Send((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 15,
MPI_COMM_WORLD);

                MPI_Send((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 16,
MPI_COMM_WORLD);

        MPI_Send((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 17,
MPI_COMM_WORLD);

        MPI_Send((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 18,
MPI_COMM_WORLD);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

        }

        //comienza la fft por columnas del proceso 0

        if(rank==0){

                p1=3*(col/4);

                p2=col;

        }

        if(rank!=0){
```

```
                p1=0;
                p2=col/4;
        }
   filasfft(f,col,MA_re,MA_im,p1,p2);
        ///////proceso para enviar la información al proceso 0
        ///ahora el proceso cero reúne la información de los otros procesos
        //envió de datos al otro proceso
        if(rank==rank2){
                MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD);
                MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD);
        }
        if(rank==rank3){
                MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 21, MPI_COMM_WORLD);
                MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 22, MPI_COMM_WORLD);
        }
        if(rank==rank4){
                MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 23, MPI_COMM_WORLD);
                MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 24, MPI_COMM_WORLD);
```

```
        }

        ///recepción de datos del proceso 0

        if(rank==0){

                gettimeofday(&xp1, NULL);

                MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, rank2, 19,
MPI_COMM_WORLD,&stat);

                MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, rank2, 20,
MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 21,
MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 22,
MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 23,
MPI_COMM_WORLD,&stat);

        MPI_Recv((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 24,
MPI_COMM_WORLD,&stat);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

                //el proceso 0 realiza la transpuesta de la matriz, la parte real y la
imaginaria....

                transpuesta(f,col,MA_re);

                transpuesta(f,col,MA_im);

        }

}
```

- **Algoritmo de la IFFT bidimensional con 4 nodos**

```
void ifft2Dparalelo4(int f, int col, double *MA_re, double *MA_im, int rank, int rank2, int rank3, int rank4, double* secs)

{ int p1,p2;

        MPI_Status stat;

        struct timeval xp1,xp2,xp3;

        *secs=0;

                //envió de datos a los otros procesos

        if(rank==0){

                gettimeofday(&xp1, NULL);

                MPI_Send(MA_re, col*f/4, MPI_DOUBLE, rank2, 25, MPI_COMM_WORLD);

                MPI_Send(MA_im, col*f/4, MPI_DOUBLE, rank2, 26, MPI_COMM_WORLD);

                MPI_Send((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 27, MPI_COMM_WORLD);

                MPI_Send((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 28, MPI_COMM_WORLD);

        MPI_Send((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 29, MPI_COMM_WORLD);

        MPI_Send((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 30, MPI_COMM_WORLD);

                gettimeofday(&xp2, NULL);
```

```
            *secs=*secs+timeval_diff(&xp2, &xp1);
        }
        ///recepción de datos por los otros 3 procesos
    if(rank==rank2){
            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 25,
MPI_COMM_WORLD,&stat);
            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 26,
MPI_COMM_WORLD,&stat);
        }
        if(rank==rank3){
            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 27,
MPI_COMM_WORLD,&stat);
            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 28,
MPI_COMM_WORLD,&stat);
        }
        if(rank==rank4){
            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 29,
MPI_COMM_WORLD,&stat);
            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 30,
MPI_COMM_WORLD,&stat);
        }
        //comienza con la IFFT a cada fila del proceso 0 y los otros 3
        if(rank==0){
            p1=3*(f/4);
```

```
            p2=f;
        }
        if(rank!=0){
            p1=0;
            p2=f/4;
        }
        //calcula la fft a cada fila de cualquier proceso
        filasifft(f,col,MA_re,MA_im,p1,p2);
        ///ahora el proceso cero reúne la información de los otros procesos
        //envió de datos al otro proceso
        if(rank==rank2){
            MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 31, MPI_COMM_WORLD);
            MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 32, MPI_COMM_WORLD);
            ///recepción de datos del proceso
            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 37, MPI_COMM_WORLD,&stat);
            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 38, MPI_COMM_WORLD,&stat);

        }
        if(rank==rank3){
```

```
            MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 33,
MPI_COMM_WORLD);

            MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 34,
MPI_COMM_WORLD);

            ///recepción de datos del proceso

            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 39,
MPI_COMM_WORLD,&stat);

            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 40,
MPI_COMM_WORLD,&stat);

        }

        if(rank==rank4){

            MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 35,
MPI_COMM_WORLD);

            MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 36,
MPI_COMM_WORLD);

            ///recepción de datos del proceso

            MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, 0, 41,
MPI_COMM_WORLD,&stat);

            MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, 0, 42,
MPI_COMM_WORLD,&stat);


        }

        ///recepción de datos del proceso 0

        if(rank==0){

            gettimeofday(&xp1, NULL);
```

```
MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, rank2, 31, MPI_COMM_WORLD,&stat);

MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, rank2, 32, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 33, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 34, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 35, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 36, MPI_COMM_WORLD,&stat);

        gettimeofday(&xp2, NULL);

        *secs=*secs+timeval_diff(&xp2, &xp1);

        //el proceso 0 realiza la transpuesta de la matriz, la parte real y la imaginaria....

        transpuesta(f,col,MA_re);

        transpuesta(f,col,MA_im);

        ///envió al proceso n la mitad de la matriz de datos

        gettimeofday(&xp1, NULL);

MPI_Send(MA_re, col*f/4, MPI_DOUBLE, rank2, 37, MPI_COMM_WORLD);

MPI_Send(MA_im, col*f/4, MPI_DOUBLE, rank2, 38, MPI_COMM_WORLD);
```

```
        MPI_Send((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 39,
MPI_COMM_WORLD);

        MPI_Send((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 40,
MPI_COMM_WORLD);

        MPI_Send((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 41,
MPI_COMM_WORLD);

        MPI_Send((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 42,
MPI_COMM_WORLD);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

        }

        //comienza la ifft por columnas de los procesos

        if(rank==0){

                p1=3*(col/4);

                p2=col;

        }

        if(rank!=0){

                p1=0;

                p2=col/4;

        }

   filasifft(f,col,MA_re,MA_im,p1,p2);

        ///////proceso para enviar la información al proceso 0

        ///ahora el proceso cero reúne la información de los otros procesos
```

```
//envió de datos al otro proceso
if(rank==rank2){
MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 43, MPI_COMM_WORLD);
MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 44, MPI_COMM_WORLD);
}
if(rank==rank3){
MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 45, MPI_COMM_WORLD);
MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 46, MPI_COMM_WORLD);
}
if(rank==rank4){
MPI_Send(MA_re, col*f/4, MPI_DOUBLE, 0, 47, MPI_COMM_WORLD);
MPI_Send(MA_im, col*f/4, MPI_DOUBLE, 0, 48, MPI_COMM_WORLD);
}
///recepción de datos del proceso 0
if(rank==0){
gettimeofday(&xp1, NULL);
MPI_Recv(MA_re, col*f/4, MPI_DOUBLE, rank2, 43, MPI_COMM_WORLD,&stat);
```

```
MPI_Recv(MA_im, col*f/4, MPI_DOUBLE, rank2, 44, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_re+col*f/4), col*f/4, MPI_DOUBLE, rank3, 45, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_im+col*f/4), col*f/4, MPI_DOUBLE, rank3, 46, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_re+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 47, MPI_COMM_WORLD,&stat);

MPI_Recv((MA_im+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 48, MPI_COMM_WORLD,&stat);

        gettimeofday(&xp2, NULL);

        *secs=*secs+timeval_diff(&xp2, &xp1);

        //el proceso 0 realiza la transpuesta de la matriz, la parte real y la imaginaria....

        transpuesta(f,col,MA_re);

        transpuesta(f,col,MA_im);

    }


}
```

- **Algoritmo de la multiplicación punto a punto para 4 nodos**

```
//esta función reserva memoria temporal en el otro proceso para procesar la información

void multiplicaparalelo4(int f,int col,double *Ar,double *Ai, double *Br,double *Bi, int rank,int rank2,int rank3,int rank4, double *secs){

    double ax,bx,cx,dx;
```

```
int p1,p2;

int i,j;

*secs=0;

struct timeval xp1,xp2,xp3;

MPI_Status stat;

if(rank==0){

        gettimeofday(&xp1, NULL);

        MPI_Send(Ar, col*f/4, MPI_DOUBLE, rank2, 49, MPI_COMM_WORLD);

        MPI_Send(Ai, col*f/4, MPI_DOUBLE, rank2, 50, MPI_COMM_WORLD);

        MPI_Send(Br, col*f/4, MPI_DOUBLE, rank2, 51, MPI_COMM_WORLD);

        MPI_Send(Bi, col*f/4, MPI_DOUBLE, rank2, 52, MPI_COMM_WORLD);

        MPI_Send((Ar+col*f/4), col*f/4, MPI_DOUBLE, rank3, 53, MPI_COMM_WORLD);

        MPI_Send((Ai+col*f/4), col*f/4, MPI_DOUBLE, rank3, 54, MPI_COMM_WORLD);

        MPI_Send((Br+col*f/4), col*f/4, MPI_DOUBLE, rank3, 55, MPI_COMM_WORLD);

        MPI_Send((Bi+col*f/4), col*f/4, MPI_DOUBLE, rank3, 56, MPI_COMM_WORLD);

        MPI_Send((Ar+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 57, MPI_COMM_WORLD);
```

```
                MPI_Send((Ai+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 58,
MPI_COMM_WORLD);

                MPI_Send((Br+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 59,
MPI_COMM_WORLD);

                MPI_Send((Bi+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 60,
MPI_COMM_WORLD);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

                p1=3*f/4;

                p2=f;

  }

      ///ahora se reserva memoria para cada proceso auxiliar

  if(rank!=0){ //reserva espacio temporal para procesar

                p1=0;

                p2=f/4;

      }

      /// cada proceso recibe una tercera parte...

      if(rank==rank2){

    MPI_Recv(Ar, col*f/4, MPI_DOUBLE, 0, 49, MPI_COMM_WORLD,&stat);

    MPI_Recv(Ai, col*f/4, MPI_DOUBLE, 0, 50, MPI_COMM_WORLD,&stat);

    MPI_Recv(Br, col*f/4, MPI_DOUBLE, 0, 51, MPI_COMM_WORLD,&stat);

    MPI_Recv(Bi, col*f/4, MPI_DOUBLE, 0, 52, MPI_COMM_WORLD,&stat);

      }
```

```
        if(rank==rank3){

                MPI_Recv(Ar,    col*f/4,    MPI_DOUBLE,    0,    53,
MPI_COMM_WORLD,&stat);

                MPI_Recv(Ai,    col*f/4,    MPI_DOUBLE,    0,    54,
MPI_COMM_WORLD,&stat);

                MPI_Recv(Br,    col*f/4,    MPI_DOUBLE,    0,    55,
MPI_COMM_WORLD,&stat);

                MPI_Recv(Bi,    col*f/4,    MPI_DOUBLE,    0,    56,
MPI_COMM_WORLD,&stat);

        }

        if(rank==rank4){

                MPI_Recv(Ar,    col*f/4,    MPI_DOUBLE,    0,    57,
MPI_COMM_WORLD,&stat);

                MPI_Recv(Ai,    col*f/4,    MPI_DOUBLE,    0,    58,
MPI_COMM_WORLD,&stat);

                MPI_Recv(Br,    col*f/4,    MPI_DOUBLE,    0,    59,
MPI_COMM_WORLD,&stat);

                MPI_Recv(Bi,    col*f/4,    MPI_DOUBLE,    0,    60,
MPI_COMM_WORLD,&stat);

        }

        for (i=p1;i<p2;i++)

        {

      for(j=0;j<col;j++)

                {

         ax=*(Ar+i*col+j);
```

```
            bx=*(Ai+i*col+j);
            cx=*(Br+i*col+j);
            dx=*(Bi+i*col+j);
            *(Br+i*col+j)=ax*cx-bx*dx;
            *(Bi+i*col+j)=bx*cx+ax*dx;
                }
    }
        ///los procesos envían la parte de la matriz B desarrollada
        if(rank==rank2){
                MPI_Send(Br, col*f/4, MPI_DOUBLE, 0, 61, MPI_COMM_WORLD);
                MPI_Send(Bi, col*f/4, MPI_DOUBLE, 0, 62, MPI_COMM_WORLD);
        }
        if(rank==rank3){
                MPI_Send(Br, col*f/4, MPI_DOUBLE, 0, 63, MPI_COMM_WORLD);
                MPI_Send(Bi, col*f/4, MPI_DOUBLE, 0, 64, MPI_COMM_WORLD);
        }
        if(rank==rank4){
                MPI_Send(Br, col*f/4, MPI_DOUBLE, 0, 65, MPI_COMM_WORLD);
```

```
                MPI_Send(Bi, col*f/4, MPI_DOUBLE, 0, 66, MPI_COMM_WORLD);


        }

        /// el proceso 0 recibe los datos

   if(rank==0){     gettimeofday(&xp1, NULL);

                MPI_Recv(Br, col*f/4, MPI_DOUBLE, rank2, 61, MPI_COMM_WORLD,&stat);

                MPI_Recv(Bi, col*f/4, MPI_DOUBLE, rank2, 62, MPI_COMM_WORLD,&stat);

                MPI_Recv((Br+col*f/4), col*f/4, MPI_DOUBLE, rank3, 63, MPI_COMM_WORLD,&stat);

                MPI_Recv((Bi+col*f/4), col*f/4, MPI_DOUBLE, rank3, 64, MPI_COMM_WORLD,&stat);

        MPI_Recv((Br+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 65, MPI_COMM_WORLD,&stat);

        MPI_Recv((Bi+2*col*f/4), col*f/4, MPI_DOUBLE, rank4, 66, MPI_COMM_WORLD,&stat);

                gettimeofday(&xp2, NULL);

                *secs=*secs+timeval_diff(&xp2, &xp1);

        }

}
```

- **Algoritmo de la convolución cíclica bidimensional para 4 nodos**

```
void convolucionparalelo4(int f,int col,double *Ar,double *Ai,double *Br, double *Bi, int rank,int rank2,int rank3,int rank4,double *secs){
```

```
        *secs=0;

        double comu1=0; //el tiempo que gasta cada función en comunicarse con
otros procesos

        fft2Dparalelo4(f,col,Ar,Ai,rank,rank2,rank3,rank4,&comu1);

        *secs=*secs+comu1;

    fft2Dparalelo4(f,col,Br,Bi,rank,rank2,rank3,rank4,&comu1);

        *secs=*secs+comu1;

    multiplicaparalelo4(f,col,Ar,Ai,Br,Bi,rank,rank2,rank3,rank4,&comu1);

        *secs=*secs+comu1;

    ifft2Dparalelo4(f,col,Br,Bi,rank,rank2,rank3,rank4,&comu1);

        *secs=*secs+com
```

ANEXO D. FUNCION PARA VALIDAR EL ALGORITMO DE CONVOLUCIÓN CÍCLICA BIDIMENSIONAL

Este algoritmo tiene la finalidad de comprobar que la matriz resultado de la implementación en 2 o 4 nodos se compare con la implementación que se realiza en matlab. Para comprobar los resultados se comparan las matrices para determinar la igualdad.

Se tiene una matriz A que representa los resultados en matlab de la convolución entre 2 matrices y se tiene la matriz B que es el resultado de alguna de las implementaciones con 2 o 4 nodos:

```
Void comparacionmatrices(double *Ar, double *Ai, double *Br, double *Bi,double  int n)
  {  Int i=0;
     Int j=0;
      for(i=0; i<n; i++) {
    for(j=0; j<n; j++){
    if(*(Ar+i*col+j)!= *(Br+i*col+j)){
        printf("\n error, la matrices son diferentes");
       break;
            } // compara la parte real de las matrices
    if(*(Ai+i*col+j)!= *(Bi+i*col+j)){
        printf("\n error, la matrices son diferentes");
```

```
            break;

                } //compara la parte imaginaria de las matrices

}

}
```