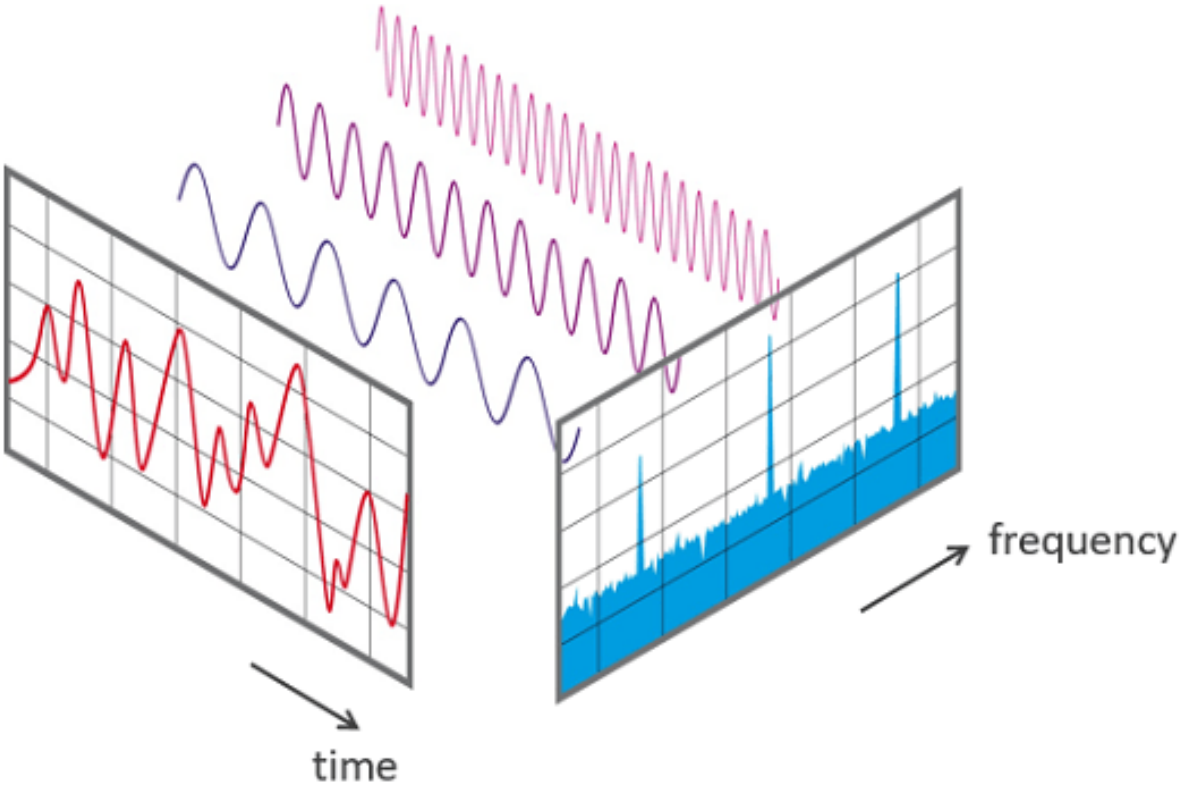
Proyecto 2

Algoritmos paralelos: Transformada de Fourier.



Equipo 4

Calderón Jiménez, David
Hernández Olvera, Humberto Ignacio
Ordiales Caballero, Iñaky

20 de Enero del 2021.

Descripción General del Proyecto

Para este segundo proyecto se deberá realizar un trabajo de investigación acerca de algún algoritmo, alguna arquitectura o alguna forma de implementación de la programación paralela. Así mismo, se deberá realizar un video para la presentación de los puntos mas relevantes del trabajo de investigación.

Objetivo

Que el alumno ponga en práctica los conceptos de la programación paralela a través de la implementación de un algoritmo paralelo, así mismo desarrolle su capacidad para responder preguntas acerca de un concepto analizado a profundidad.

Objetivos personales

Nuestro objetivo como equipo para este proyecto es lograr un gran entendimiento del algoritmo empleado para calcular computacionalmente la transformada de Fourier. Y de este modo poder explicar de forma clara y precisa su implementación y funcionamiento desde una perspectiva paralela.

Introducción

A lo largo del curso de EDA 2 hemos estudiado diversos temas sobre las estructuras de datos y los algoritmos. El último tema que se encuentra en el temario es la *Introducción a los Algoritmos Paralelos*. En la clase de teoría, así como en las prácticas del laboratorio hemos podido aprender los conceptos básicos de la programación paralela, así como hemos desarrollado programas simples que la usen. Sin embargo, hasta ahora no hemos revisado un uso e implementación de la vida real de este tipo de algoritmos. En este trabajo se explorará el algoritmo ***Fast Fourier Transform* (FFT)** o transformada rápida de Fourier. Pero antes de poder llegar al paralelismo del algoritmo, primero se requieren explicar otros temas.

Uno de los aspectos de mayor importancia dentro de la programación es la eficiencia de los algoritmos/programas. Existen diferentes formas de medir la eficiencia de un programa o un procesador, ya sea por espacio de memoria o por tiempo. Viéndolo desde el punto de vista del tiempo la eficiencia depende directamente de cuánto se tarda en ejecutar una instrucción básica y del número de instrucciones básicas que pueden ser ejecutadas concurrentemente. Esta eficiencia puede ser incrementada por avances en la arquitectura como el aumento del tamaño de la palabra del procesador (8, 16, 32, 64 bits) y por avances tecnológicos en transistores y demás componentes electrónicos. El avance en la arquitectura mediante el tamaño de palabra del procesador incrementan la cantidad de trabajo que se puede realizar por ciclo de instrucción. Con los 32 o 64 bits de las computadora modernas realmente ya no se obtiene mucho beneficio de seguirlas aumentando. Al menos por el momento se encuentran en su estado más optimo. Por el otro lado, tenemos los componentes electrónicos. Al mejorar estos, se disminuye el tiempo del ciclo del procesador, es decir el tiempo para ejecutar la operación más básica. Sin embargo, actualmente estos tiempos ya decrecen muy poco con cada mejora y parece que están alcanzando límites físicos como la velocidad de la luz. Esto significa que pronto tampoco se podrá mejorar el tiempo de ejecución mejorando los componentes electrónicos. Esto nos lleva a la pregunta, ¿Cómo podemos seguir eficientizando la programación?

Al no poder depender de procesadores más rápidos para obtener más eficiencia, la solución es un cambio de paradigma en la programación. Más en específico el usar la **Programación Paralela**. La programación paralela parte de la idea del multiprocesamiento. El multiprocesamiento es el usar más de un procesador a la vez (o por lo menos varios núcleos) para realizar en conjunto una serie de instrucciones. Entonces un programa paralelo es aquel formado por varios procesos que se ejecutan en múltiples procesadores conectados entre sí mediante una red de comunicación. Todo esto siguiendo un esquema de paralelismo espacial, el cual se expresa físicamente en varios procesadores operando en simultáneo. Los beneficios que se obtienen con este paradigma es el lograr en muchas ocasiones disminuir los tiempos de ejecución y volver más eficientes los programas. Sin embargo, el paralelismo no siempre es posible, y aún cuando sí lo es, puede no volver más eficiente el programa debido a los tiempos de comunicación entre los procesadores. Todo esto se tiene que estudiar al momento de querer paralelizar algún algoritmo ya existente, o al tratar de crear uno desde cero. En este trabajo se estudia la paralelización del algoritmo ya existente de la transformada rápida de Fourier. Se revisará el algoritmo, su implementación y sus medidas de eficiencia.

Transformada de Fourier

Antes de llegar a ver los algoritmos, primero hay que entender aunque sea superficialmente la trasformada de Fourier. Una transformada es un operador especial matemático que representa una transformación entre dos bases diferentes. Dentro de las transformaciones existen un subconjunto llamado transformaciones integrales, éstas son una función (la cual se centra en el cálculo de una integral) que se aplica sobre alguna otra función para llevarla de un dominio a su expresión equivalente en otro dominio. La transformada de Fourier, es una caso especial de una transformación integral. Denominada así por Joseph Fourier, se emplea comúnmente para llevar funciones o señales desde el dominio del tiempo (o espacial) hasta el dominio de la frecuencia. A pesar de que en un inicio esto tal vez no nos parezca tan relevante, la Transformada de Fourier es una de las más importantes (puede que sea la más importante) que se hayan descubierto en las matemáticas. Algunos de los usos más importantes en la matemática, física e ingeniería los vemos en el procesamiento de señales (electrónicas), comunicaciones, óptica, procesamiento de imágenes y medicina, por mencionar algunos.

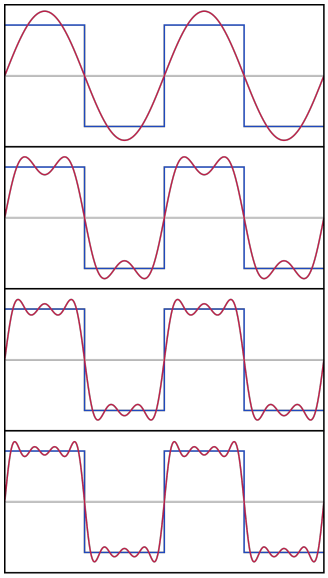


Figura 1: Diagrama de FFT

El que nos ha interesado más desde el punto de vista ingenieril es el de procesamiento de señales electrónicas, ya que esta aplicación fue lo que nos permitió evolucionar de gran forma las telecomunicaciones. Los ejemplos iniciales más grandes del uso de la transformada de Fourier están en la radio, donde a través de ella podemos filtrar las diferentes frecuencias y concentrarnos en sólo una. Pero además de eso la utilizamos en las redes como el wi-fi, el internet, o la telefonía móvil. Éstas no existirían como los conocemos hoy en día si nos fuera por la transformada de Fourier. Fue gracias a su gran cantidad y relevancia de usos que se vio casi natural el computalizar sus cálculos para facilitar su obtención. Esto ya que para cada señal, la transformada es una operación diferente y a pesar de tener tablas que facilitan el cálculo, en muchas ocasiones no es sencillo realizarlo. Por eso se planteó en un inicio un algoritmo secuencial que pudiera realizarla. Pero para poder ver bien el algoritmo antes repasemos qué es y de dónde sale la idea de una Transformada de Fourier.

La transformada de Fourier surge de las series de Fourier. A su vez la serie de Fourier surge de la idea que cualquier función periódica puede ser representada mediante una suma infinita de senos y cosenos. Existen dos tipos de series de Fourier. La trigonométrica y la compleja. La trigonométrica es un termino constante más una suma infinita de funciones coseno más una suma infinita de funciones seno. La serie trigonométrica de Fourier se define para una función $f(t)$ en un intervalo $-L < x < L$ donde la función es periódica de la siguiente manera:

$$f(t) = \frac{a(0)}{2} + \sum_{n=1}^{\infty} \left[a(n) \cos \frac{n\pi t}{L} + b(n) \sin \frac{n\pi t}{L} \right]$$

donde $a(0)$, $a(n)$ y $b(n)$ son los coeficiente de Fourier definidos como:

$$\begin{aligned} a(0) &= \frac{1}{L} \int_{-L}^{L} f(t)\, dx \\ a(n) &= \frac{1}{L} \int_{-L}^{L} f(t) \cos \frac{n\pi t}{L} dx \\ b(n) &= \frac{1}{L} \int_{-L}^{L} f(t) \sin \frac{n\pi t}{L} dx \end{aligned}$$

A su vez de la serie trigonométrica se puede llegar a la serie compleja. La serie compleja de Fourier parte de que el periodo de una función $f(t)$ es T y C_n, w_0 son:

$$w(0) = \frac{2\pi}{T}$$

$$C(n) = \frac{a_n - ib_n}{2} = \frac{1}{T} \int_{\frac{-T}{2}}^{\frac{T}{2}} f(t)\, e^{-inw_0 t} dt$$

Así la serie compleja de Fourier es:

$$f(t) = \sum_{n=1}^{\infty} \left[C(n)\, e^{inw_0 t} \right] + \sum_{n=-1}^{-\infty} \left[C(n)\, e^{inw_0 t} \right]$$

$$f(t) = \sum_{n=-\infty}^{\infty} \left[C(n)\, e^{inw_0 t} \right]$$

Una vez se tiene la serie compleja de Fourier y bajo la idea de que cualquier función es periódica en el intervalo $(\infty, -\infty)$ mediante algunos cálculos que no se muestran en este documento llegamos a la dichosa **Transformada de Fourier** y a su vez a la transformada inversa. El propio término se refiere tanto a la operación de transformación como a la función que produce.

$$\boldsymbol{f_{(t)} = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{j\omega t} d\omega} \quad \rightarrow \quad \textit{Antitransformada de Fourier}$$

$$\boldsymbol{F(\omega) = \int_{-\infty}^{+\infty} f_{(t)} e^{-j\omega t} dt} \quad \rightarrow \quad \textit{Transformada de Fourier}$$

Figura 2: Transformada de Fourier.

Sin embargo la computalización de esta transformada no sea hace directamente, ya que sería un proceso laborioso y costoso computacionalmente hablando. Para mejorar esto se creó la Transformada Discreta de Fourier, de la cual deriva la versión ya programable, la Transformada Rápida de Fourier. En las siguientes secciones se hablará más de esto.

Algoritmo Secuencial Transformada de Fourier

La transformada rápida de Fourier, conocida por la abreviatura FFT (del inglés Fast Fourier Transform) es un algoritmo eficiente que permite calcular la transformada de Fourier discreta (DFT) y su inversa. La FFT es en ocasiones catalogado como el algoritmo más poderoso del último siglo. Esto debido a su de gran importancia en una amplia variedad de aplicaciones, desde el tratamiento y filtrado digital de señales, compresión de imágenes y audio, resolución de ecuaciones diferenciales parciales o inclusive en los algoritmos de multiplicación rápida de grandes enteros. Para poder presentar el algoritmo, lo único que nos falta es ver la transformada discreta de Fourier (que en realidad es una serie) y porqué se usa ésta.

Discrete Fourier Transform (DFT)

En la mayoría de los casos no se cuenta con funciones analíticas desde las cuales se puede calcular una transformación continua, sino que se tienen mediciones experimentales o mediciones de simulaciones. Por esto en realidad lo que se tiene son N medidas puntuales en un intervalo de $[x_0, x_n]$. Por lo que tenemos un vector de datos $[x_0, x_1, x_2, ..., x_n]^T$ con los que queremos obtener la serie de Fourier para cada uno de esos puntos. Y esto es lo que hace la Transformada de Fourier. A continuación se defina la Transformada Discreta de Fourier:

$$\hat{x}_k = \sum_{n=0}^{N-1} \left[x_n e^{\frac{-i2\pi}{N} kn} \right]$$

La Transformada Discreta Inversa de Fourier es:

$$x_n = \sum_{n=0}^{N-1} \left[\hat{x}_n e^{\frac{i2\pi}{N} kn} \right]$$

donde $\hat{x}_k$ es el k-ésimo coeficiente de Fourier y x_n es el n-ésimo dato del intervalo.

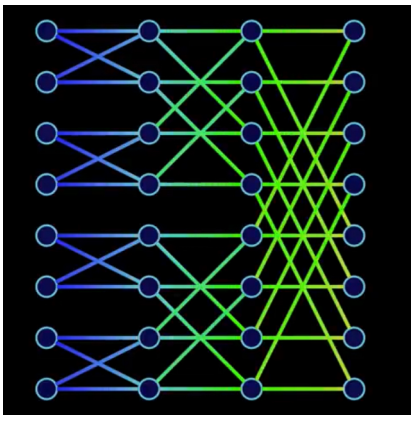


Figura 3: Circuito del cómputo de la transformada de Fourier

Fast Fourier Transform (FFT)

Ahora bien, la Transformada Discreta de Fourier es una transformación matemática que puede ser escrita en términos de una multiplicación de matrices muy grandes. La Transformada Rápida de Fourier es una forma computacionalmente eficiente de calcular la transformada discreta para conjuntos de datos muy grandes. En cierto modo son sinónimos, FFT es cómo computamos la DFT. Cuando se habla del tratamiento digital de señales, el algoritmo FFT impone algunas limitaciones en la señal y en el espectro resultante ya que la señal muestreada y que se va a transformar debe consistir de un número de muestras igual a una potencia de dos.

El algoritmo secuencial de la Fast Fourier Transfrom se basa en el principio de "divide y vencerás". Aprovecha la periodicidad de los datos (siendo el periodo $T = x_n - x_0$) y de la simetría de la transformada al trabajar con conjugados complejos.La transformada discreta de Fourier puede ser reescrita en forma matricial de la siguiente forma:

$$\begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \\ ... \\ \hat{x}_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & w_n & w_n^2 & ... & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & ... & w_n^{2(n-1)} \\ . & . & . & . & . \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & ... & w_n^{(n-1)^2} \end{bmatrix}}_{Matriz\ de\ Transformacion\ Directa\ de\ Fourier\ \mathbf{DFT}} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} \tag{1}$$

Esta puede ser simplificada de la siguiente forma (múltiples veces):

$$\begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \hat{x}_2 \\ ... \\ \hat{x}_n \end{bmatrix} = \begin{bmatrix} [I_{\frac{n}{2}\times\frac{n}{2}}] & -[D_{\frac{n}{2}\times\frac{n}{2}}] \\ [I_{\frac{n}{2}\times\frac{n}{2}}] & -[D_{\frac{n}{2}\times\frac{n}{2}}] \end{bmatrix} \begin{bmatrix} [DFT_{\frac{n}{2}\times\frac{n}{2}}] & 0 \\ 0 & [DFT_{\frac{n}{2}\times\frac{n}{2}}] \end{bmatrix} \begin{bmatrix} [x_{par}] \\ [x_{impar}] \end{bmatrix} \tag{2}$$

La idea general del algoritmo es la siguiente:**Inicio.**

1. Dividir el vector de datos inicial en dos vectores, uno para los índices par y otro de los índices impar. (pasar de la ecuación 1 a la ecuación 2).
2. Obtener las matrices diagonales y sub-matrices de transformación correspondientes.
3. Se puede repetir el paso 1 y 2 las veces que sea deseado. Estas serán las etapas de descomposición.
4. Realizar la multiplicación de matrices resultante.

Fin.

El algoritmo se puede ver de manera gráfica en el diagrama de la figura 4 en la siguiente página.

Algo interesante de notar es que antes de ser paralelizado, el algoritmo FFT ya es bastante más eficiente que implementar directamente la transformada discreta. FFT tiene una complejidad temporal de $O(n \log n)$ mientras que DFT tiene una complejidad temporal de $O(n^2)$, por lo que ya representa una mejora con tan solo usar el algoritmo FFT que realiza una multiplicación de matrices en lugar de varios ciclos for anidados.

**** Como dato curioso, se ha encontrado que el matemático alemán Carl Friedrich Gauss, apodado el "príncipe de las matemáticas, desarrolló y usó un algoritmo muy parecido al FFT moderno en 1805. Inclusive siendo esta fecha anterior a la publicación de los estudios harmónicos de Fourier (1807).*

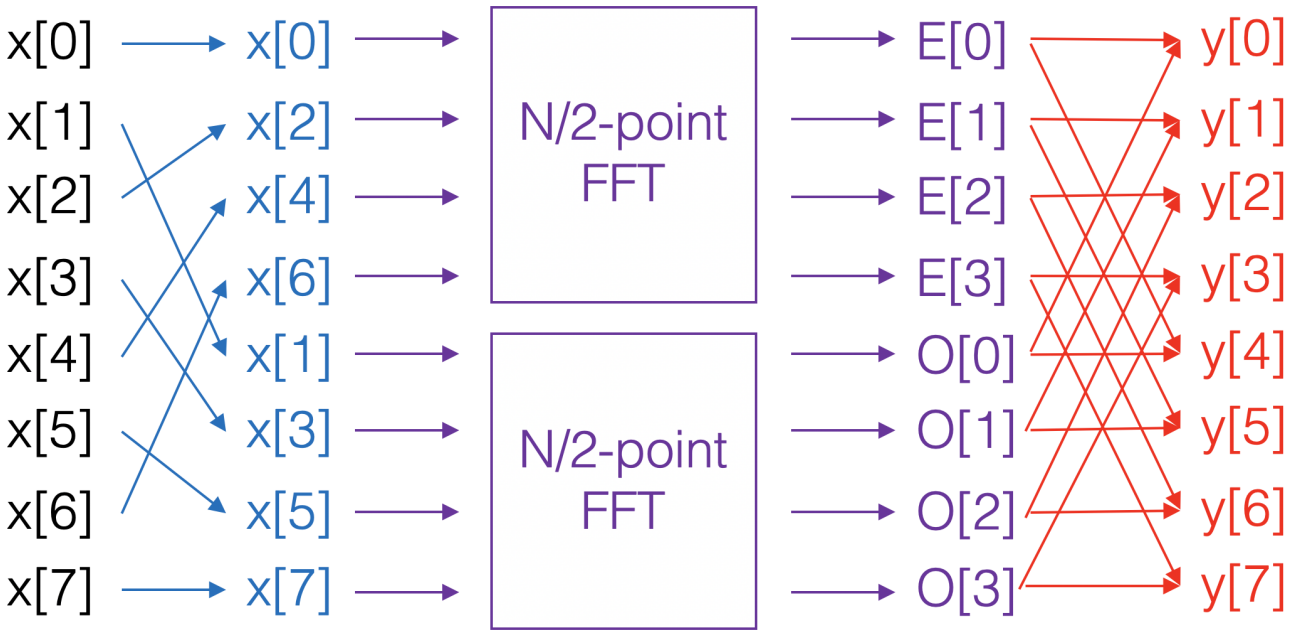


Figura 4: Diagrama de FFT

Paralelización

La Transformada Discreta de Fourier (DFT) es un problema para el cual hay una larga historia de algoritmos paralelos. El algoritmos para la Transformada Rápida de Fourier (FFT) para resolver la DFT es sencillo de paralelizar. Proporciona dos oportunidades principales de paralelización, siendo uno las sub divisiones de los vectores de datos y el otro la multiplicación de matrices. Además de éstas dos áreas, gran parte de la investigación se ha ido en reducir los costos de comunicación siendo una parte vital y complicada de la programación paralela. La topología que se ha encontrado más eficiente es la de red de mariposa. Y el algoritmo FFT es tan importante que la topología de red de mariposa es también llamada red FFT ya que es dónde más común se ve esta topología.

Algoritmo en paralelo

El algoritmo paralelo para la transformada de Fourier rápida es sencillo de entender una vez se comprende el algoritmo secuencial. A continuación se muestra un ejemplo de algoritmo FFT paralelo sobre números complejos:

FFT(A)

```
1 n = |A|
2 if (n=1) then return A
3 else
4     in parallel do
5         EVEN = FFT({A[2i : i ∈ [0..n/2]})
6         ODD = FFT({A[2i + 1] : i \in [0..n/2] })
7     return {EVEN[j]+ODD[j]e^{2πij/n} : j∈[0..n/2]}++{EVEN[j]−ODD[j]e^{2πij/n} : j∈[0..n/2]}
```

Simplificando su funcionamiento, el algoritmo se llama recursivamente en los elementos *odd* y *even* y después junta los resultados. El algoritmo tiene una complejidad temporal de O*(n* log *n)*, así como su versión secuencial, y tiene una profundidad recursiva de O*(*log *n)*. Lo importante de notar aquí (quedará más claro al revisar la topología) es que la paralelización se lleva a cabo conjunto con la recursividad.

Topología de red de mariposa

La topología de red de mariposa es una técnica para enlazar múltiples computadoras a una red de alta velocidad. Esta forma de topología de red de interconexión multi-etapa puede ser usada para conectar diferentes nodos en un sistema multiprocesador. En el algoritmo FFT se usa esta topología donde cada nodo es una unidad de procesamiento que podrá trabajar con una fracción de los datos iniciales. Pero como vimos en el algoritmo, la comunicación es muy importante, ya que para calcular los coeficientes de fourier se usa un dato par y uno impar del vector inicial. Debido a esto se necesita esta topología que tiene exactamente los canales de comunicación entre procesadores necesarios.

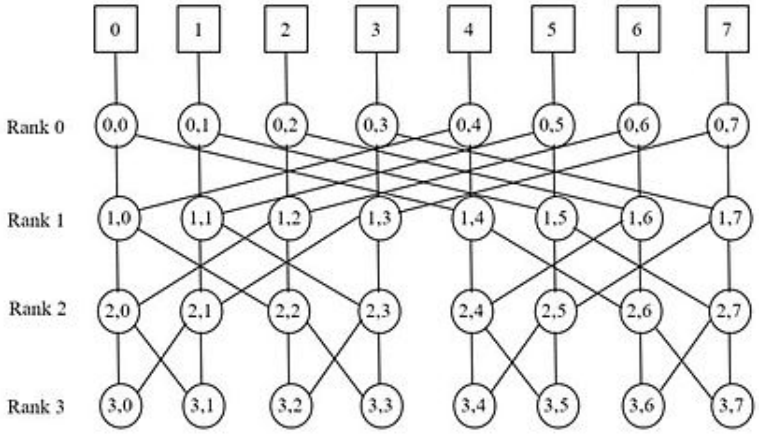


Figura 5: Topología de red de mariposa.

Tipo de paralelismo

El tipo de paralelismo que se utiliza en este algoritmo es el de **paralelismo de datos**.Éste es un paradigma de la programación concurrente que consiste en subdividir el conjunto de datos de entrada a un programa, de manera que a cada procesador le corresponda un subconjunto de esos datos. Cada procesador efectuará la misma secuencia de operaciones que los otros procesadores sobre su subconjunto de datos asignado. En resumen: se distribuyen los datos y se replican las tareas. Idealmente, esta ejecución simultánea de operaciones, resulta en una ***aceleración neta global del cómputo***.

Para este ejemplo en particular del algoritmo rápido de la transformada de Fourier utilizaremos este tipo de paralelismo, pues este es es el mas adecuado gracias a que la mayor parte del programa depende de ciclos, los cuales al paralelizarlos cada procesador efectúa la misma operación pero con diferentes datos asignados.

Métricas de desempeño

Para una N igual 1, vemos que el algoritmo secuencial se ejecuta en menos tiempo que su versión paralela.

Para una N igual a 10, el algoritmo secuencial es ahora más lento que su versión paralela, y mientras se siga incrementando el valor de N, la versión paralela sigue siendo más rápida que la secuencial.

Todos los datos usados en las siguientes gráficas son el promedio de 5 ejecuciones para cada situación particular.

Comparativa de Tiempos

Las siguiente gráfica es una comparativa del tiempo que se tarda el programa en calcular la transformada de Fourier para una medida experimental de 150 puntos. (Idealmente en la vida real estos serían alguna señal digital captada que se está tratando de filtrar).

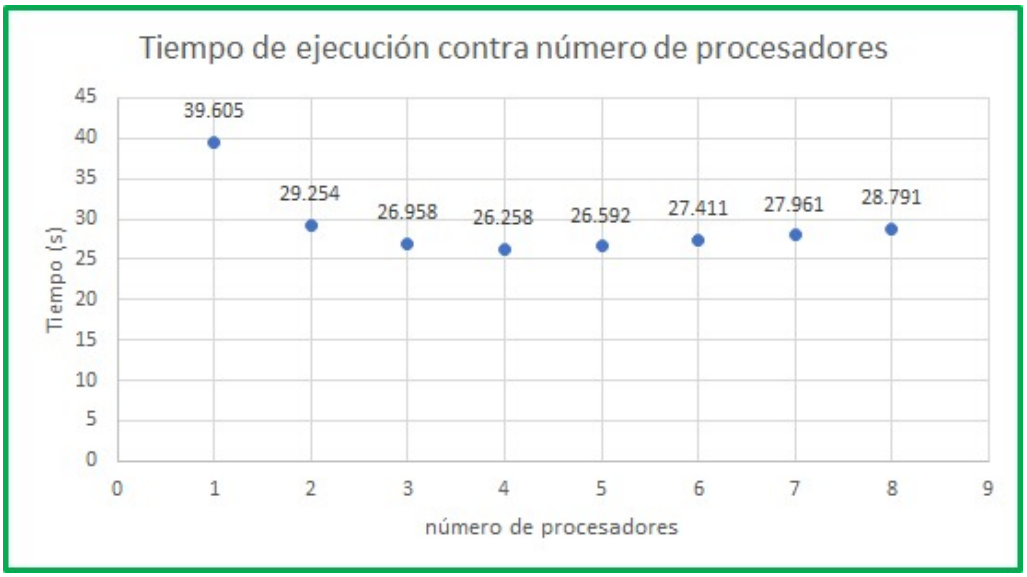


Figura 6: Gráfica 1. Comparativa de Tiempos

En la gráfica podemos observar como el tiempo con un sólo procesador, es decir al realizar una ejecución totalmente secuencial se tarda 39.605 segundos en calcular los 150 coeficientes de Fourier necesarios. De ahí conforma vamos agregando más unidades de procesamiento el tiempo va bajando, hasta llegar a un óptimo con respecto del tiempo en 4 procesadores. Después para 5, 6, 7, 8 procesadores se empieza a tardar más tiempo. Aunque nunca llega cercano a los 39.605 del procesamiento único.

Speedup

El Speedup es la relación de tiempo de ejecución de un programa ejecutándose en un solo procesador sobre el tiempo de ejecución del mismo programa ejecutándose en n procesadores. Esta medida es muy parecida con respecto a la gráfica anterior (del tiempo neto). Sin embargo ahora se mide qué tan más rápido es el uso de múltiples procesadores con respecto a uno sólo. Esta métrica solo considera aspectos temporales, no toma en cuenta otros como la topología elegida, el balance de carga, la granularidad, etc.

$$S(n) = \frac{T(1)}{T(n)}$$

Num. Procesadores	1	2	3	4	5	6	7	8
Speedup	1.0	1.35	1.47	1.51	1.49	1.45	1.42	1.38

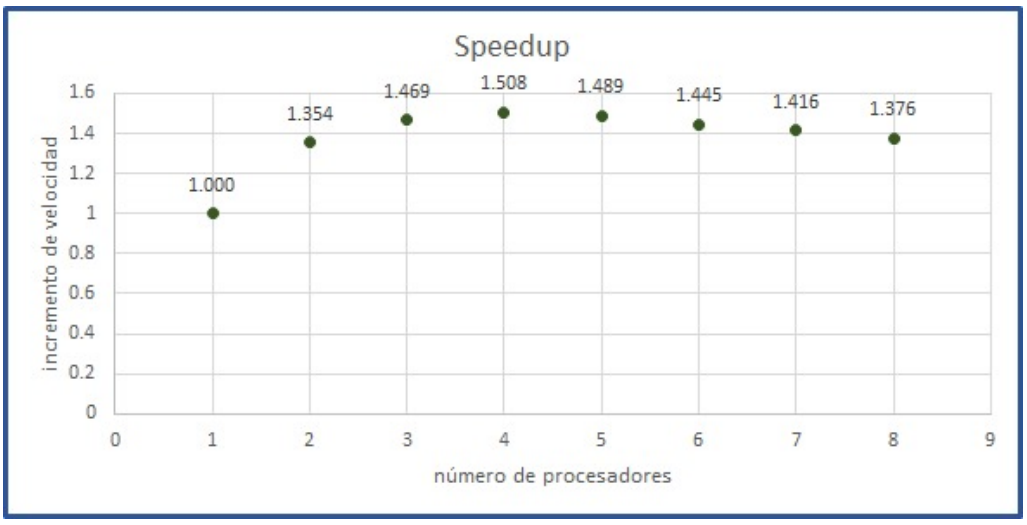


Figura 7: Gráfica 2. Speedup

En la gráfica de arriba podemos ver como la medida de Speedup parte de que el tiempo que tarda en un sólo procesador será la medida de una unidad de velocidad. Los demás datos son cuántas veces se incrementa la velocidad con n número de procesadores. Al igual que con la gráfica temporal (son los mismo datos expresados diferentes), vemos que el punto más óptimo es con cuatro procesadores. Al usar los cuatro procesadores en paralelo se tiene 1.508 veces la velocidad de ejecución inicial. Después para 5, 6, 7, y 8 empieza a bajar esta cantidad. Las razones más probables de esto, son que para la cantidad de datos usadas (150 medidas) la subdivisión de cálculos es eficiente entre cuatro, pero después los tiempos de comunicación entre los procesadores es mayor al beneficio de seguir dividiendo las tareas. Esto podría cambiar si se usaran más o menos datos.

Eficiencia

Se asocia a la idea de que n procesadores deben hacer el trabajo en una fracción $\frac{1}{n}$ del tiempo que le lleva a un solo procesador. Esta medida toma en cuenta el porcentaje de aprovechamiento de cada procesador individual que se use.

$$E(n) = \frac{T(1)}{n \cdot T(n)} = \frac{S(n)}{n}$$

Num. Procesadores	1	2	3	4	5	6	7	8
Eficiencia	1.0	0.67	0.49	0.38	0.30	0.24	0.20	0.17

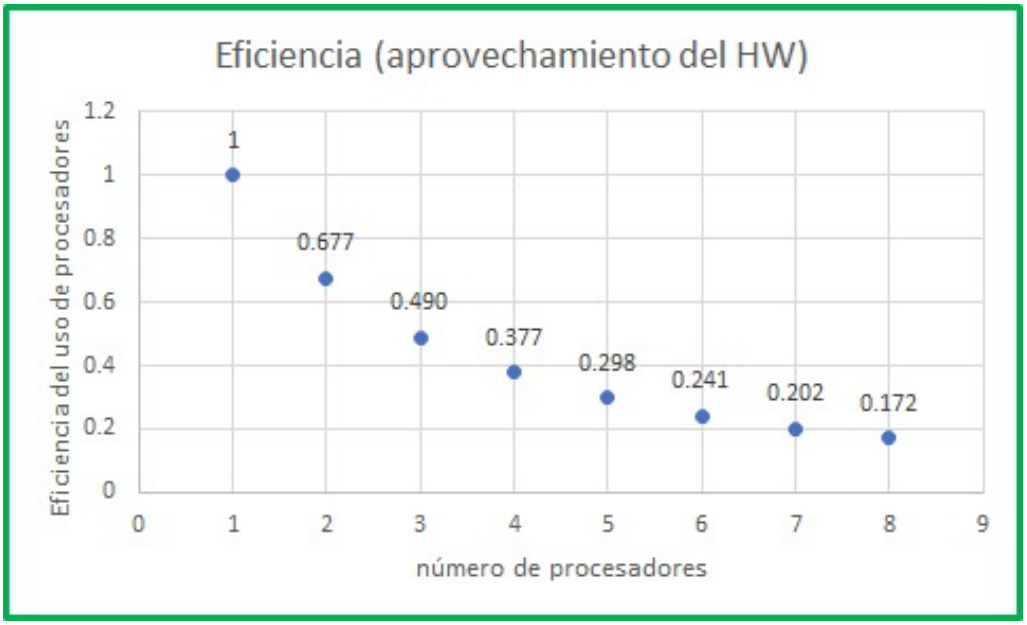


Figura 8: Gráfica 3. Eficiencia

El valor de la eficiencia refleja el aprovechamiento de los recursos de Hardware del sistema, por lo tanto, siempre será menor a 1. Se parte nuevamente de la idea que el aprovechamiento máximo del procesador es realizar la ejecución en 39.605 segundos que es lo que tardó un sólo procesador en hacerlo. Entonces para seguir teniendo un aprovechamiento del 100 % ó 1 se tendría que realizar en dos procesadores en 19.802 segundos (la mitad). Esto desde la teoría no sería posible debido a que algo de tiempo se debe perder en la comunicación entre procesadores. En la gráfica vemos como donde más se aprovechan los recursos del procesador son con 2 procesadores. A partir de ahí todo va bajando. Esto es de esperarse, y da una perspectiva a las gráficas anteriores dónde podríamos creer que lo más óptimo era usar 4 procesadores. Pues aquí vemos que no se aprovechan arriba del 40 % cada uno de ellos.

Fracción Serial

También conocida como métrica Karp-Flatt, relaciona el Speedup y la eficiencia con el propósito de tomar en cuenta otros factores además del tiempo. Hay quienes piensan que la fracción serial proporciona un método más real de medir el rendimiento del proceso paralelo. La fracción serial determinada experimentalmente es una métrica muy útil por dos razones: primero toma en cuenta el costo de comunicación en procesos paralelos y segundo puede ayudar a detectar otras fuentes de retrasos o ineficiencias que son ignorados con los modelos temporales más simples.

$$f = \frac{\frac{1}{s} - \frac{1}{n}}{1 - \frac{1}{n}}$$

Num. Procesadores	2	3	4	5	6	7	8
f	0.48	0.52	0.55	0.59	0.68	0.66	0.69

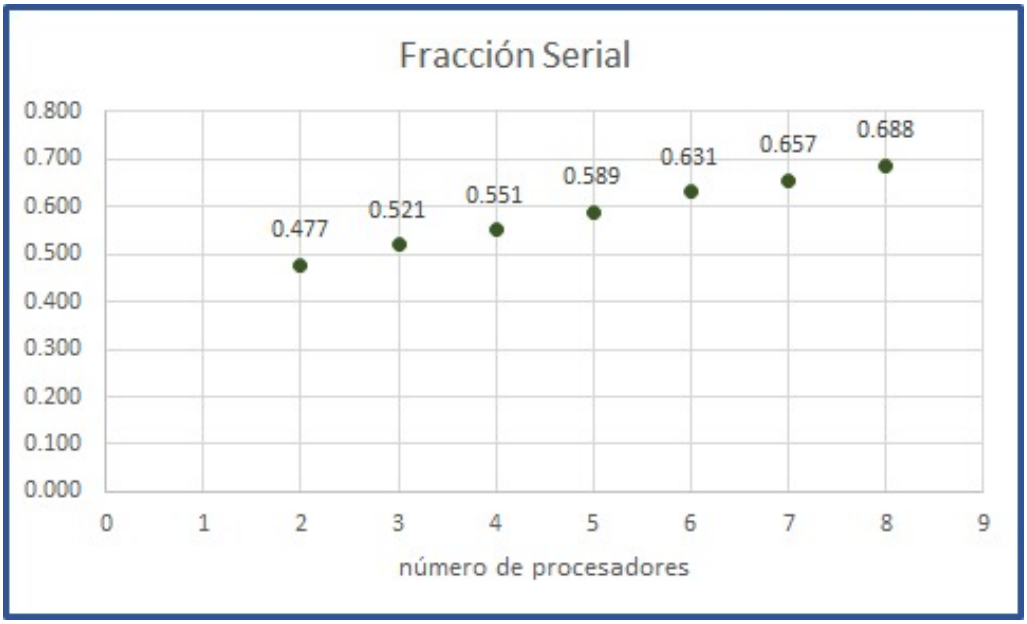


Figura 9: Gráfica 4. Fracción Serial

En esta última métrica que se analizó lo que podemos ver es qué tan paralelizadamente eficiente es el proceso. Nos permite distinguir si los decrementos en la eficiencia al aumentar en procesadores son debido a oportunidades limitadas de paralelizar o incrementos en la carga arquitectónica (mal balance de carga). Entre menor el valor de f mejor la paralelización. Por esto podemos algo interesante que no habíamos visto en las otras métricas, y es que entre más procesadores, la fracción serial aumenta, en ningún lado hay un aparente punto de inflexión. Esto probablemente se debe a la cantidad limitada de datos con los que se trabajó. De haber usado más, probablemente veríamos un cambio en la monotonía de la gráfica. Según esta métrica en términos de paralelización lo mejor es dos procesadores.

Habiendo visto estas métricas del paralelismo, queda la pregunta ¿Cuántos procesadores puedo usar para que mi ejecución sea lo más eficiente posible? La respuesta es que va depender de tus necesidades, tus capacidades (en términos de Hardware) y un poco de criterio personal. No se puede cumplir con todas las métricas de la manera más eficiente posible, se tendrán que aceptar niveles bajos en una para tener altos en otra. Por ejemplo si se cuenta con una capacidad grande de procesadores sin usarse y quieres el tiempo más rápido, podrías usar el punto más alto del Speedup aunque se desaprovechen los procesadores. O en caso de que se tengan muchos programas o la vez podrías buscar el mayor aprovechamiento de procesadores para usarlos en diferentes tareas. Así es que las métricas nos dan una idea del funcionamiento, pero finalmente la configuración debe ser decisión del programador.

Formas de comunicación

El medio de comunicación es por medio de la memoria compartida, que es aquel tipo de memoria que puede ser accedida por múltiples programas, ya sea para comunicarse entre ellos o para evitar copias redundantes. La memoria compartida es un modo eficaz de pasar datos entre aplicaciones. Dependiendo del contexto, los programas pueden ejecutarse en un mismo procesador o en procesadores separados.

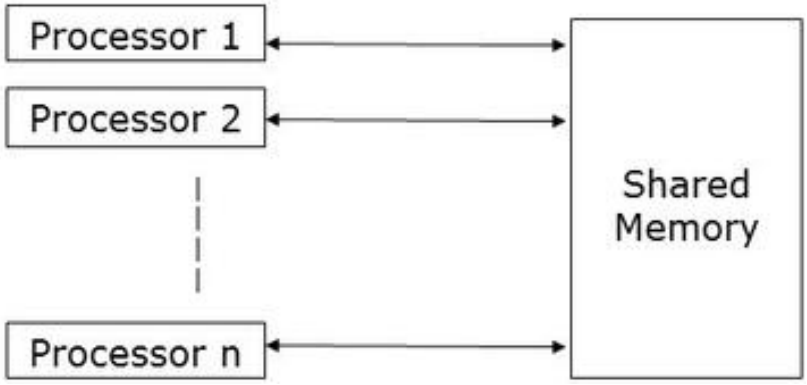


Figura 10: Diagrama Memoria compartida

Granularidad

A partir de una comunicación por memoria compartida, podemos decir que la granularidad es de tipo fina pues la mayor parte del código se basa en las operaciones con matrices las cuales al trabajar con paralelismo de datos por lo cual al ser la mayor carga de trabajo en operaciones paralelas la granularidad se vuelve fina gracias a que el numero de llamadas es mayor , esto lo podemos comprobar en las pruebas que veremos a continuación, en donde al trabajar con pocos datos el numero de llamadas es mayor al de operaciones por lo que su versión secuencial es mas eficiente que la paralela cuando se trabaja con pocos datos.

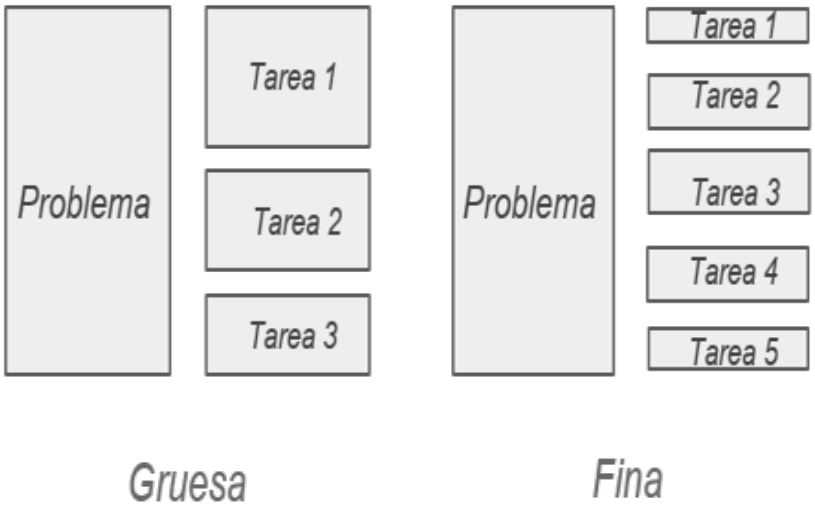


Figura 11: Tipos de Granularidad

Implementación

El código seleccionado para llevar a cabo la Transformada Rápida de Fourier (FFT) se lleva a cabo utilizando el lenguaje C y la API OpenMP, este código no es de nuestra autoría [4], pero se modificó para que se pudiera adaptar a nuestras necesidades y poder reutilizarlo para las pruebas en su forma secuencial. Involucra las siguientes etapas, dentro de un ciclo for que se repite 20 veces:

1. Se generan dos vectores de números complejos con números pseudo-aleatorios, almacenados en 4 arreglos, 2 contienen las partes reales de cada vector, y los otros 2 contienen las partes imaginarias de cada vector.
2. Se inicializan las tablas de senos y cosenos que son necesarias para el cálculo de la FFT de acuerdo al valor de n.
3. Se realiza un paso de la FFT hacia adelante y hacia atrás.
4. Se repite el ciclo.

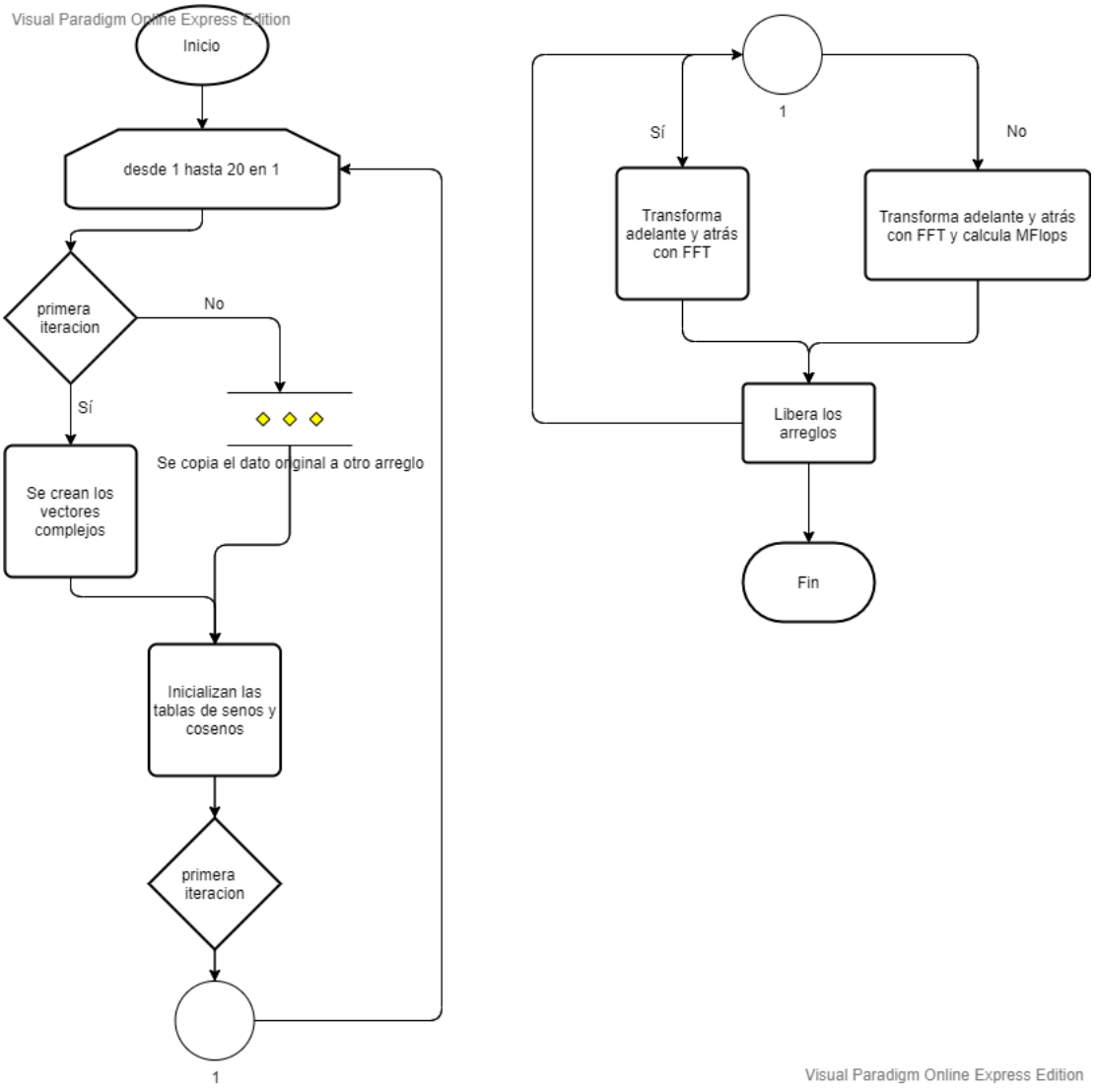


Figura 12: Diagrama de flujo de FFT en paralelo.

Pruebas

Se realizaron pruebas entre la versión secuencial y la versión paralela del algoritmo FFT a partir de la siguiente formula:

FFT (FFT (X(1:N))) == N * X(1:N)

El lenguaje de programación implementado fue Open MP y el numero de núcleos fue de 4

Para N=1 Secuencial

```
18 January 2021 04:59:12 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en secuencia.

  Numero de procesadores disponibles = 4
  Numero de hilos =              4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

           2      1000  7.859082e-017  -2.863817e-008  -1.431908e-011  -698368.666016
           4      1000  1.209837e-016  4.388858e-008  2.194429e-011  1822797.791406
           8      1000  6.605630e-017  1.162989e-007  5.814945e-011  2063647.950030
          16      1000  1.232264e-016  9.999505e-004  4.999752e-007    640.031711
          32       100  1.249348e-016  9.534415e-008  4.767207e-010  1678131.300024
          64       100  1.780479e-016  -7.054769e-008  -3.527384e-010  -5443126.870178
         128       100  1.795875e-016  1.000002e-003  5.000010e-006    895.998289
         256       100  1.984648e-016  3.999908e-003  1.999954e-005    512.011718
         512        10  1.838748e-016  2.000105e-003  1.000052e-004    230.387920
        1024        10  2.172566e-016  1.999918e-003  9.999590e-005    512.020987
        2048        10  2.347084e-016  4.999970e-003  2.499985e-004    450.562748
        4096        10  2.419067e-016  1.999906e-003  9.999532e-005   2457.714902
        8192         1  2.503041e-016  3.000112e-003  1.500056e-003    354.973357
       16384         1  2.597819e-016  3.999987e-003  1.999993e-003    573.441909
       32768         1  2.818931e-016  1.600005e-002  8.000023e-003    307.199120
       65536         1  2.755240e-016  3.299991e-002  1.649996e-002    317.751148
      131072         1  3.043879e-016  3.299998e-002  1.649999e-002    675.219707
      262144         1  3.159392e-016  8.599996e-002  4.299998e-002    548.673768
      524288         1  3.167919e-016  1.459999e-001  7.299995e-002    682.293084
     1048576         1  3.150524e-016  3.190000e-001  1.595000e-001    657.414456

FFT_OPENMP:
  Fin normal de ejecucion.

18 January 2021 04:59:14 PM

Process returned 0 (0x0)   execution time : 2.530 s
Press any key to continue.
```

Para N=1 Paralelo

```
18 January 2021 05:05:16 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en paralelo.

  Numero de procesadores disponibles = 4
  Numero de hilos =              4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

           2      1000  7.859082e-017  3.390000e-001  1.695000e-004      0.058997
           4      1000  1.209837e-016  2.629999e-001  1.314999e-004      0.304183
           8      1000  6.820795e-017  3.770000e-001  1.885000e-004      0.636605
          16      1000  1.312671e-016  5.039999e-001  2.520000e-004      1.269841
          32       100  1.387007e-016  6.500006e-002  3.250003e-004      2.461536
          64       100  1.784741e-016  8.400007e-002  4.200004e-004      4.571425
         128       100  1.911039e-016  1.050001e-001  5.250005e-004      8.533325
         256       100  2.016687e-016  1.100000e-001  5.500000e-004     18.618183
         512        10  1.977335e-016  1.300001e-002  6.500003e-004     35.446138
        1024        10  2.255626e-016  1.299990e-002  6.499950e-004     78.769832
        2048        10  2.435083e-016  1.500003e-002  7.500017e-004    150.186323
        4096        10  2.513547e-016  1.899998e-002  9.499990e-004    258.694997
        8192         1  2.579525e-016  2.000052e-003  1.000026e-003    532.466276
       16384         1  2.683104e-016  5.999968e-003  2.999984e-003    382.295399
       32768         1  2.897807e-016  7.999893e-003  3.999947e-003    614.408205
       65536         1  2.822297e-016  1.499998e-002  7.499991e-003    699.051469
      131072         1  3.125032e-016  2.400009e-002  1.200004e-002    928.423200
      262144         1  3.168578e-016  4.400010e-002  2.200005e-002   1072.404901
      524288         1  3.233599e-016  6.799998e-002  3.399999e-002   1464.922722
     1048576         1  3.234230e-016  1.459999e-001  7.299995e-002   1436.406549

FFT_OPENMP:
  Fin normal de ejecucion.

18 January 2021 05:05:19 PM

Process returned 0 (0x0)   execution time : 3.416 s
Press any key to continue.
```

Para N=10 Secuencial

```
18 January 2021 04:59:42 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en secuencia.

  Numero de procesadores disponibles = 4
  Numero de hilos =             4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

          20      1000  8.537772e-001  2.000000e-003  1.000000e-006    100.000001
          40      1000  8.096629e-001  9.999790e-004  4.999895e-007    800.016819
          80      1000  8.485440e-001  1.200010e-002  6.000052e-006    199.998283
         160      1000  7.340861e-001  2.399993e-002  1.199996e-005    266.667493
         320       100  8.363330e-001  6.999954e-003  3.499977e-005    228.572923
         640       100  8.200579e-001  1.499996e-002  7.499982e-005    256.000618
        1280       100  8.155705e-001  3.100008e-002  1.550004e-004    289.031546
        2560       100  8.169067e-001  7.500004e-002  3.750002e-004    273.066514
        5120        10  7.923575e-001  6.000076e-003  3.000038e-004    767.990233
       10240        10  8.101234e-001  2.400001e-002  1.200001e-003    426.666462
       20480        10  8.220416e-001  9.500005e-002  4.750002e-003    237.136723
       40960        10  8.040792e-001  1.250001e-001  6.250005e-003    393.215694
       81920         1  7.962851e-001  8.000082e-003  4.000041e-003   1331.186357
      163840         1  8.081469e-001  3.299988e-002  1.649994e-002    695.081239
      327680         1  8.159570e-001  8.500000e-002  4.250000e-002    578.258798
      655360         1  8.037431e-001  1.429999e-001  7.149995e-002    733.270429
     1310720         1  7.962577e-001  3.509999e-001  1.754999e-001    634.821846
     2621440         1  8.077929e-001  7.379999e-001  3.690000e-001    639.375686
     5242880         1  8.151664e-001  2.975000e+000  1.487500e+000    334.839401
    10485760         1  8.039938e-001  8.392000e+000  4.196000e+000    249.898953

FFT_OPENMP:
  Fin normal de ejecucion.

18 January 2021 05:00:21 PM

Process returned 0 (0x0)   execution time : 38.198 s
Press any key to continue.
```

Para N=10 Paralelo

```
18 January 2021 05:05:31 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en paralelo.

  Numero de procesadores disponibles = 4
  Numero de hilos =              4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

            N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

           20      1000  8.537772e-001  5.300000e-001  2.650000e-004      0.377358
           40      1000  8.096629e-001  5.980001e-001  2.990001e-004      1.337792
           80      1000  8.485440e-001  7.790000e-001  3.895000e-004      3.080873
          160      1000  7.340861e-001  1.259000e+000  6.295000e-004      5.083400
          320       100  8.363330e-001  1.379999e-001  6.899996e-004     11.594210
          640       100  8.200579e-001  2.390000e-001  1.195000e-003     16.066942
         1280       100  8.155705e-001  2.829999e-001  1.415000e-003     31.660784
         2560       100  8.169067e-001  2.170000e-001  1.085000e-003     94.377866
         5120        10  7.923575e-001  2.700008e-002  1.350004e-003    170.666185
        10240        10  8.101234e-001  3.299992e-002  1.649996e-003    310.303802
        20480        10  8.220416e-001  5.699999e-002  2.849999e-003    395.228163
        40960        10  8.040792e-001  8.800004e-002  4.400002e-003    558.545188
        81920         1  7.962851e-001  1.399996e-002  6.999979e-003    760.687993
       163840         1  8.081469e-001  3.400006e-002  1.700003e-002    674.634045
       327680         1  8.159570e-001  8.499991e-002  4.249995e-002    578.259459
       655360         1  8.037431e-001  1.090001e-001  5.450004e-002    961.995691
      1310720         1  7.962577e-001  1.960000e-001  9.800001e-002   1136.848869
      2621440         1  8.077929e-001  8.979999e-001  4.490000e-001    525.455719
      5242880         1  8.151664e-001  1.940000e+000  9.700000e-001    513.477916
     10485760         1  8.039938e-001  3.702000e+000  1.851000e+000    566.491632

FFT_OPENMP:
  Fin normal de ejecucion.

18 January 2021 05:06:03 PM

Process returned 0 (0x0)   execution time : 32.113 s
Press any key to continue.
```

Para N=50 Secuencial

```
18 January 2021 05:02:36 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en secuencia.

  Numero de procesadores disponibles = 4
  Numero de hilos =              4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         100      1000  1.001026e+000  2.699994e-002  1.349997e-005     37.037116
         200      1000  9.063760e-001  5.500004e-002  2.750002e-005     72.727215
         400      1000  9.427557e-001  8.600001e-002  4.300000e-005    139.534871
         800      1000  9.322318e-001  1.259999e-001  6.299996e-005    253.968400
        1600       100  9.331531e-001  1.400005e-002  7.000026e-005    571.426468
        3200       100  9.447340e-001  3.100005e-002  1.550002e-004    619.353849
        6400       100  9.185633e-001  8.199999e-002  4.100000e-004    546.341502
       12800       100  9.320175e-001  1.980001e-001  9.900003e-004    517.171563
       25600        10  9.315296e-001  6.700010e-002  3.350005e-003    343.880069
       51200        10  9.303725e-001  1.410001e-001  7.050005e-003    363.120302
      102400        10  9.300261e-001  1.849999e-001  9.249994e-003    608.865229
      204800        10  9.302140e-001  4.410001e-001  2.205000e-002    557.278801
      409600         1  9.299454e-001  8.300010e-002  4.150005e-002    641.541358
      819200         1  9.323193e-001  1.680000e-001  8.400001e-002    682.666589
     1638400         1  9.293493e-001  4.150000e-001  2.075000e-001    592.192785
     3276800         1  9.293158e-001  1.114000e+000  5.570000e-001    470.635554
     6553600         1  9.304864e-001  2.949000e+000  1.474500e+000    377.793147
    13107200         1  9.322584e-001  8.927000e+000  4.463500e+000    264.287670
    26214400         1  9.320340e-001  2.155600e+001  1.077800e+001    231.060308

Process returned -1073741819 (0xC0000005)   execution time : 109.742 s
Press any key to continue.
```

Para N=50 Paralelo

```
18 January 2021 05:07:33 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en paralelo.

  Numero de procesadores disponibles = 4
  Numero de hilos =             4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         100      1000  1.001026e+000  1.088000e+000  5.440000e-004      0.919118
         200      1000  9.063760e-001  1.238000e+000  6.190000e-004      3.231018
         400      1000  9.427557e-001  1.217000e+000  6.084999e-004      9.860313
         800      1000  9.322318e-001  1.306000e+000  6.530000e-004     24.502297
        1600       100  9.331531e-001  1.379999e-001  6.899997e-004     57.971039
        3200       100  9.447340e-001  1.629999e-001  8.149996e-004    117.791476
        6400       100  9.185633e-001  2.449999e-001  1.225000e-003    182.857197
       12800       100  9.320175e-001  9.590001e-001  4.795000e-003    106.777885
       25600        10  9.315296e-001  1.440000e-001  7.200000e-003    160.000000
       51200        10  9.303725e-001  1.990000e-001  9.949998e-003    257.286494
      102400        10  9.300261e-001  3.610000e-001  1.805000e-002    312.022144
      204800        10  9.302140e-001  6.510000e-001  3.255000e-002    377.511502
      409600         1  9.299454e-001  1.380000e-001  6.899999e-002    385.855152
      819200         1  9.323193e-001  2.870000e-001  1.435000e-001    399.609812
     1638400         1  9.293493e-001  5.300001e-001  2.650001e-001    463.698021
     3276800         1  9.293158e-001  1.078000e+000  5.390000e-001    486.352513
     6553600         1  9.304864e-001  2.308000e+000  1.154000e+000    482.717498
    13107200         1  9.322584e-001  4.420000e+000  2.210000e+000    533.777385
    26214400         1  9.320340e-001  9.409000e+000  4.704500e+000    529.358693

Process returned -1073741819 (0xC0000005)   execution time : 107.901 s
Press any key to continue.
```

Para N=100 Secuencial

```
18 January 2021 05:56:02 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en secuencia.

  Numero de procesadores disponibles = 4
  Numero de hilos =              4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

            N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

          200      1000  9.395808e-001  7.999999e-002  4.000000e-005     25.000002
          400      1000  9.564980e-001  1.349999e-001  6.749995e-005     59.259301
          800      1000  9.240786e-001  2.199999e-001  1.100000e-004    109.090952
         1600      1000  9.325115e-001  3.119999e-001  1.560000e-004    205.128247
         3200       100  9.448918e-001  6.500005e-002  3.250002e-004    246.153673
         6400       100  9.186924e-001  1.780000e-001  8.900001e-004    215.730314
        12800       100  9.325095e-001  3.760000e-001  1.880000e-003    238.297864
        25600       100  9.316171e-001  9.099999e-001  4.549999e-003    225.054971
        51200        10  9.305450e-001  2.180001e-001  1.090000e-002    211.376093
       102400        10  9.300060e-001  3.850001e-001  1.925000e-002    265.973959
       204800        10  9.302815e-001  8.630001e-001  4.315000e-002    261.042853
       409600        10  9.298957e-001  2.678000e+000  1.339000e-001    183.539959
       819200         1  9.323349e-001  5.710000e-001  2.855000e-001    186.507893
      1638400         1  9.293476e-001  1.129000e+000  5.645000e-001    203.167417
      3276800         1  9.293200e-001  2.320000e+000  1.160000e+000    211.862065
      6553600         1  9.304876e-001  5.406000e+000  2.703000e+000    193.965226
     13107200         1  9.322599e-001  1.070000e+001  5.350000e+000    208.245233
     26214400         1  9.320341e-001  2.299600e+001  1.149800e+001    205.191860

Process returned -1073741819 (0xC0000005)   execution time : 145.505 s
Press any key to continue.
```

Para N=100 Paralelo

```
18 January 2021 05:50:37 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en paralelo.

  Numero de procesadores disponibles = 4
  Numero de hilos =              4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         200      1000  9.395808e-001  9.410001e-001  4.705001e-004      2.125398
         400      1000  9.564980e-001  1.003000e+000  5.015000e-004      7.976072
         800      1000  9.240786e-001  2.007000e+000  1.003500e-003     11.958147
        1600      1000  9.325115e-001  1.857000e+000  9.285000e-004     34.464191
        3200       100  9.448918e-001  2.059999e-001  1.030000e-003     77.669930
        6400       100  9.186924e-001  2.890000e-001  1.445000e-003    132.871956
       12800       100  9.325095e-001  4.499999e-001  2.250000e-003    199.111138
       25600       100  9.316171e-001  8.180001e-001  4.090000e-003    250.366722
       51200        10  9.305450e-001  1.680001e-001  8.400003e-003    274.285605
      102400        10  9.300060e-001  3.980000e-001  1.990000e-002    257.286464
      204800        10  9.302815e-001  4.900000e-001  2.450000e-002    459.755102
      409600        10  9.298957e-001  1.206000e+000  6.030000e-002    407.562156
      819200         1  9.323349e-001  2.820000e-001  1.410000e-001    377.645370
     1638400         1  9.293476e-001  5.530000e-001  2.765000e-001    414.784839
     3276800         1  9.293200e-001  1.091000e+000  5.455000e-001    450.522448
     6553600         1  9.304876e-001  2.269000e+000  1.134500e+000    462.131313
    13107200         1  9.322599e-001  4.332000e+000  2.166000e+000    514.363795
    26214400         1  9.320341e-001  9.477000e+000  4.738500e+000    497.899338

Process returned -1073741819 (0xC0000005)   execution time : 87.250 s
Press any key to continue.
```

Para N=200 Secuencial

```
18 January 2021 05:58:43 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en secuencia.

  Numero de procesadores disponibles = 4
  Numero de hilos =             4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         400      1000  9.431494e-001  9.499998e-002  4.749999e-005     42.105272
         800      1000  9.367156e-001  1.530000e-001  7.650001e-005    104.575146
        1600      1000  9.315125e-001  5.060000e-001  2.530000e-004     94.861664
        3200      1000  9.367655e-001  7.250000e-001  3.625000e-004    176.551726
        6400       100  9.201461e-001  2.160001e-001  1.080000e-003    148.148097
       12800       100  9.308572e-001  4.159999e-001  2.080000e-003    184.615420
       25600       100  9.308080e-001  7.280000e-001  3.640000e-003    246.153861
       51200       100  9.309691e-001  1.312000e+000  6.560000e-003    312.195116
      102400        10  9.296986e-001  2.770000e-001  1.385000e-002    332.707551
      204800        10  9.302891e-001  6.070000e-001  3.035000e-002    337.397029
      409600        10  9.298634e-001  2.496000e+000  1.248000e-001    180.512825
      819200        10  9.323068e-001  5.171000e+000  2.585500e-001    190.106365
     1638400         1  9.293565e-001  1.219000e+000  6.095001e-001    174.726811
     3276800         1  9.293200e-001  1.795000e+000  8.975000e-001    255.572141
     6553600         1  9.304908e-001  4.981000e+000  2.490500e+000    197.357958
    13107200         1  9.322598e-001  1.016000e+001  5.080000e+000    206.412600
    26214400         1  9.320349e-001  2.253900e+001  1.126950e+001    197.721638

Process returned -1073741819 (0xC0000005)   execution time : 144.324 s
Press any key to continue.
```

Para N=200 Paralelo

```
18 January 2021 05:53:40 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en paralelo.

  Numero de procesadores disponibles = 4
  Numero de hilos =             4

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         400      1000  9.431494e-001  1.140000e+000  5.700000e-004      3.508772
         800      1000  9.367156e-001  1.212000e+000  6.060000e-004     13.201321
        1600      1000  9.315125e-001  2.209000e+000  1.104500e-003     21.729290
        3200      1000  9.367655e-001  2.227000e+000  1.113500e-003     57.476423
        6400       100  9.201461e-001  3.079999e-001  1.539999e-003    103.896139
       12800       100  9.308572e-001  4.150000e-001  2.075000e-003    185.060225
       25600       100  9.308080e-001  7.649999e-001  3.825000e-003    234.248382
       51200       100  9.309691e-001  1.294000e+000  6.470000e-003    316.537886
      102400        10  9.296986e-001  3.760000e-001  1.880000e-002    245.106367
      204800        10  9.302891e-001  6.870001e-001  3.435000e-002    298.107690
      409600        10  9.298634e-001  1.483000e+000  7.415000e-002    303.816573
      819200        10  9.323068e-001  2.588000e+000  1.294000e-001    379.845427
     1638400         1  9.293565e-001  5.679999e-001  2.840000e-001    374.985971
     3276800         1  9.293200e-001  1.036000e+000  5.180000e-001    442.810834
     6553600         1  9.304908e-001  2.280000e+000  1.140000e+000    431.157896
    13107200         1  9.322598e-001  4.339000e+000  2.169500e+000    483.326113
    26214400         1  9.320349e-001  9.176000e+000  4.588000e+000    485.663472

Process returned -1073741819 (0xC0000005)   execution time : 93.613 s
Press any key to continue.
```

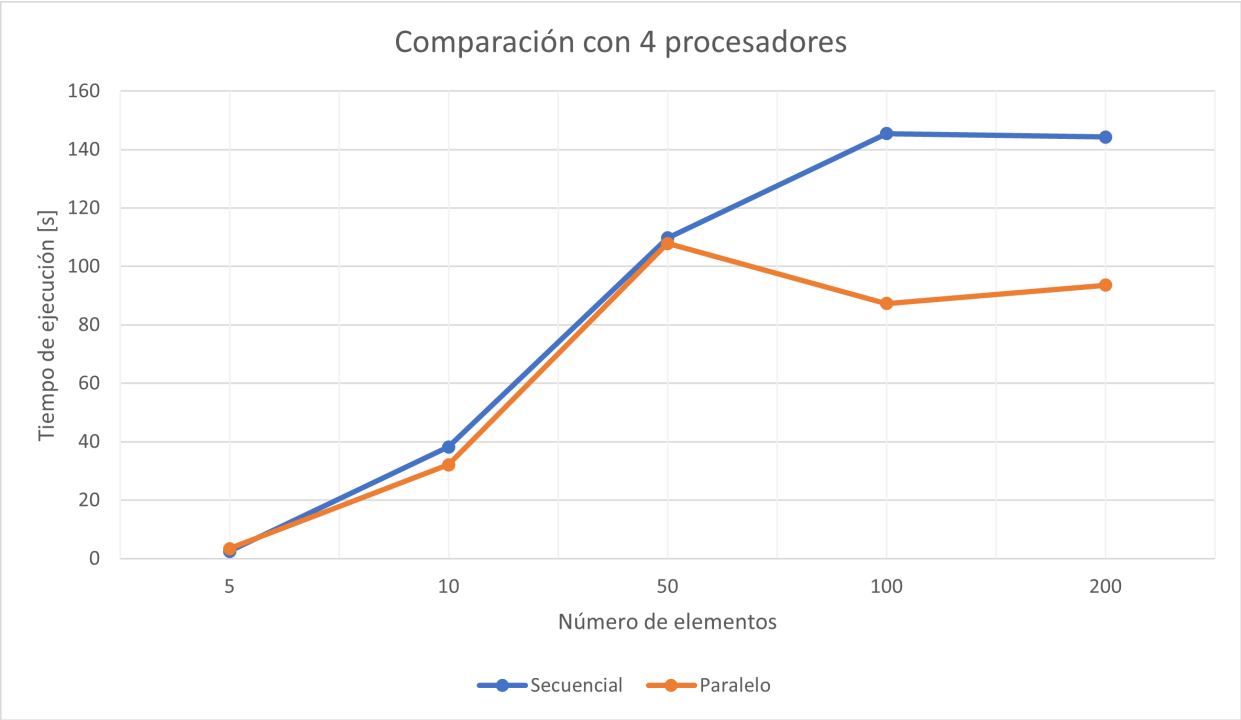


Figura 13: Gráfico comparativo entre el algoritmo secuencial y el algoritmo paralelo.

Para N=200 Secuencial 8 núcleos

```
18 January 2021 06:38:25 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en secuencia.

  Numero de procesadores disponibles = 8
  Numero de hilos =             8

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         400      1000  9.431494e-001  5.199999e-002  2.600000e-005     76.923088
         800      1000  9.367156e-001  9.500002e-002  4.750001e-005    168.421019
        1600      1000  9.315125e-001  2.430001e-001  1.215000e-004    197.530813
        3200      1000  9.367655e-001  6.240000e-001  3.120000e-004    205.128201
        6400       100  9.201461e-001  1.220001e-001  6.100003e-004    262.294947
       12800       100  9.308572e-001  2.279999e-001  1.140000e-003    336.842181
       25600       100  9.308080e-001  7.070000e-001  3.535000e-003    253.465350
       51200       100  9.309691e-001  1.273000e+000  6.365000e-003    321.759638
      102400        10  9.296986e-001  3.210001e-001  1.605000e-002    287.102746
      204800        10  9.302891e-001  6.319999e-001  3.160000e-002    324.050679
      409600        10  9.298634e-001  1.313000e+000  6.565001e-002    343.153056
      819200        10  9.323068e-001  2.248000e+000  1.124000e-001    437.295370
     1638400         1  9.293565e-001  6.360001e-001  3.180000e-001    334.893033
     3276800         1  9.293200e-001  9.830000e-001  4.915000e-001    466.685658
     6553600         1  9.304908e-001  2.422000e+000  1.211000e+000    405.879449
    13107200         1  9.322598e-001  6.076000e+000  3.038000e+000    345.153384
    26214400         1  9.320349e-001  1.121400e+001  5.607000e+000    397.400393

Process returned -1073741819 (0xC0000005)   execution time : 76.886 s
Press any key to continue.
```

Para N=200 Paralelo 8 núcleos

```
18 January 2021 06:50:35 PM

FFT_OPENMP
  C/OpenMP version

  Muestra la implementacion de la Transformada Rapida de Fourier
  de un vector complejo, utilizando OpenMP para su ejecucion en paralelo.

  Numero de procesadores disponibles = 8
  Numero de hilos =             8

  Revision de precision:

    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)

           N      NITS    Error         Tiempo        Tiempo/Llamada     MFLOPS

         400      1000  9.431494e-001  1.031000e+000  5.155000e-004      3.879728
         800      1000  9.367156e-001  1.223000e+000  6.114999e-004     13.082585
        1600      1000  9.315125e-001  1.283000e+000  6.415000e-004     37.412313
        3200      1000  9.367655e-001  1.529000e+000  7.645000e-004     83.714844
        6400       100  9.201461e-001  1.800000e-001  9.000002e-004    177.777731
       12800       100  9.308572e-001  2.860001e-001  1.430000e-003    268.531390
       25600       100  9.308080e-001  3.940001e-001  1.970000e-003    454.822267
       51200       100  9.309691e-001  5.770001e-001  2.885000e-003    709.878563
      102400        10  9.296986e-001  1.390001e-001  6.950003e-003    663.021264
      204800        10  9.302891e-001  3.189999e-001  1.595000e-002    642.006442
      409600        10  9.298634e-001  7.969999e-001  3.985000e-002    565.319997
      819200        10  9.323068e-001  1.363000e+000  6.815000e-002    721.232526
     1638400         1  9.293565e-001  2.600000e-001  1.300000e-001    819.199874
     3276800         1  9.293200e-001  6.119999e-001  3.060000e-001    749.594862
     6553600         1  9.304908e-001  1.240000e+000  6.200000e-001    792.774144
    13107200         1  9.322598e-001  2.324000e+000  1.162000e+000    902.388976
    26214400         1  9.320349e-001  4.703000e+000  2.351500e+000    947.575605

Process returned -1073741819 (0xC0000005)   execution time : 50.866 s
Press any key to continue.
```

Conclusiones

El algoritmo paralelo para la computación de la Transformada de Fourier, mejor conocido como Fast Fourier Transform (**FFT**) es uno de los algoritmos más utilizados y más importantes de los últimos tiempos. Quizá puede ser uno de los algoritmos más importantes en la historia debido a su gran rango de aplicaciones y a la relevancia que éstas tienen. De forma sencilla lo que el algoritmo hace es transformar las medidas experimentales de una onda electromagnética en una función analítica de frecuencia. Esto permite poder filtrar las señales, pero también conocer más información sobre ella. Algunos de sus usos con mayor repercusión son el internet, el GPS y la radio (nada más).

Es por esto que el poder realizar un algoritmo lo más eficiente posible en términos de computación fue esencial. Y así se llegó al FFT paralelo, con muchas implementaciones históricas y muchas investigaciones sobre como mejorarlo. Pero antes de poder analizar todo el aspecto del paradigma paralelo, primero se debe entender el algoritmo y esto resultó ser más complicado de lo que inicialmente pensamos.

En términos de los objetivos del proyecto, estos fueron cumplidos satisfactoriamente tanto en los generales como en los particulares. Durante el desarrollo del proyecto se afianzaron la gran mayoría de los conceptos vistos en la clase de teoría. Se reviso el funcionamiento de los procesadores en paralelo par poder hacer una buena ejecución que nos diera datos para el análisis. Se determinó el algoritmo tanto secuencial como paralelo de la transformada de Fourier, su tipo de paralelismo (de datos), topología de red (topología de red de mariposa), forma de comunicación (memoria compartida), granularidad (fina, mucha comunicación) y las diferentes métricas de desempeño con su análisis correspondiente. Con respecto a los objetivos particulares consideramos que sí se logró un gran entendimiento del algoritmo (tanto la parte lógica como matemática). Esto nos permitió dar una explicación clara del algoritmo y su paralelización como se puede apreciar en el video.

Finalmente el algoritmo FFT fue muy interesante de investigar y ver todas las aplicaciones que tiene en el mundo real y en nuestras vidas. Y el entender que de una forma paralela podemos mejorar su eficiencia nos dio un punto de vista más apreciativo sobre la programación paralela. En nuestro caso, la paralelización si representa una gran ventaja al secuencial. Por lo que sentimos aprendimos mucho de esta actividad del semestre.

David Calderón Jiménez

Para mi este nuevo paradigma de programación es aparte de interesante muy útil pues nos abre las puertas a aplicaciones que difícilmente podríamos hacer en una programación secuencial.Desde mi punto de vista al nivel en el que estamos hilos me parece mas útil que propiamente la programación paralela pues los hilos nos permiten trabajar con programas sencillos los cuales necesitan realizar 2 o mas acciones al mismo tiempo y la programación paralela se aplica de mejor manera cuando se trabaja con enormes cantidades de información las cuales a nuestro nivel muy pocas veces hemos visto.

Por el momento me quedare con esta primera aproximación a la programación paralela pero a su vez se que es un tema que tiene muchas aplicaciones y un campo de investigación muy amplio por lo que en un futuro yo desearía profundizarlo mas.

Humberto Ignacio Hernández Olvera

Este nuevo paradigma de programación, definitivamente es algo que se me complica entender, quizás la lógica en las líneas de los códigos es algo que puedo seguir, ya que al final de cuentas, sigue siendo código muy similar a lo que estuvimos trabajando previamente, el hecho de poder identificar en donde se tiene que hacer la paralelización es lo que me causa ruido, al igual que encontrar la aplicación del paradigma, es algo que tendría que reforzar para poder llegar a un entendimiento adecuado del paradigma.

En cuanto al algoritmo que desarrollamos, veo que la Transformada de Fourier tiene una buena aplicación en el campo de la física y la ingeniería. Con este primer acercamiento, y teniendo una idea de cómo es que se puede programar, creo que ayudará bastante para los siguientes retos a los que nos enfrentemos en la carrera más adelante, ya que no llegaremos en blanco.

Iñaky Ordiales Caballero

El proyecto en general me gustó y me pareció algo desafiante. Si bien la implementación en código no es difícil de seguir, el entender a un buen nivel el algoritmo secuencial para luego ir al paralelo requiere de una revisión de las matemáticas detrás de él. Esto hizo que nos tardáramos más tiempo en la parte escrita, ya que sentimos que no hacía sentido presentar el algoritmo sin el contexto anterior adecuado. Por lo que las primeras páginas sirven de introducción al algoritmo, tanto en el ámbito matemático como en el de la computación paralela. Finalmente el algoritmo FFT fue muy interesante de investigar y ver todas las aplicaciones que tiene en el mundo real y en nuestras vidas. Y el entender que de una forma paralela podemos mejorar su eficiencia nos dio un punto de vista más apreciativo sobre la programación paralela. Por lo que sentí que aprendimos mucho de esta actividad del semestre.

Autoevaluación

El momento de mirar atrás y auto evaluar un proyecto que realizamos siempre es un proceso interesante. Si bien mientras lo elaboramos nos damos cuenta de nuestras dificultades y fortalezas de lo que se va haciendo y escribiendo en el reporte. No es hasta este punto donde ya todo quedó finalizado, donde se hicieron varias lecturas de prueba y finalmente se decidió no modificar nada más, que realmente podemos evaluar la calidad y claridad de la información y del trabajo en general.

Al momento de iniciar con la elaboración del proyecto de investigación buscamos algunas opciones de algoritmos y enviamos nuestra propuesta. El que se decidió que hiciéramos fue el algoritmo para computar la Transformada de Fourier. Y a pesar de haber leído un poco al respecto antes de enviarlo como posible opción, realmente no investigamos a profundidad hasta estar seguros que fue el elegido. Para el momento en que nos adentramos más en el algoritmo creo que nos dimos cuenta que para entenderlo, tendríamos que entender las matemáticas detrás de él. Y a pesar de no ser las más complicadas, sí representaron un desafío. Sin embargo pudimos leer muchos recursos y ver varios videos en internet para comprenderla por completo. A partir de que comprendimos el algoritmo, pudimos empezar a realizar todo el análisis de la paralelización junto con las pruebas que conllevan.

Para no seguirnos alargando mucho, consideramos que todo el proceso desde el cómo y de dónde sale el algoritmo secuencial, hasta la parte de paralelizarlo, ver la topología, comunicación y granularidad que tiene está claramente explicado en este documento. Además el video deja claro lo más importante sobre el algoritmo paralelo, que era lo principal del proyecto. Por lo que consideramos que el trabajo entregado en todas sus diferentes partes es de buena calidad y con el que nos sentimos satisfechos. De poder cambiar algo, probablemente nos hubiera gustado hacer muchas más pruebas para poder tener más información analizable sobre el paralelismo del algoritmo. Pero debido a los núcleos de nuestras compus no lo pudimos hacer. Por lo demás sentimos el trabajo es muy bueno.

Poner un número para representar tu propio trabajo siempre es difícil. Muchas veces podemos terminar no siendo objetivos o por el contrario sobrecompensar y ser muy estrictos. Haciendo nuestro mejor esfuerzo para evitar esto, pensamos que el trabajo merece un **9.5** ya que si bien no es perfecto, es muy bueno y cumple con los requisitos.

Fuentes de información

[1] S G Aki. «The design and analysis of parallel algorithms». En: (ene. de 1989). URL: https://www.osti.gov/biblio/5692994.

[2] Guy E. Blelloch y Bruce M. Maggs. «Parallel Algorithms». En: ().

[3] esacademic. *Paralelismo de datos*. URL: https://esacademic.com/dic.nsf/eswiki/900010 (visitado 16-01-2021).

[4] Wesley Petersen y John Burkardt. *Fast Fourier Transform Using OpenMP*. URL: https://people.sc.fsu.edu/~jburkardt/c_src/fft_openmp/fft_openmp.html (visitado 15-01-2021).

[5] K.R. Rao, D Kim y Jae-Jeong Hwang. *Fast Fourier Transform - Algorithms and Applications*. Ene. de 2010, pág. 442. ISBN: 978-1-4020-6628-3. DOI: 10.1007/978-1-4020-6629-0.