PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UNA ARQUITECTURA RÁPIDA PARA LA APLICACIÓN DEL TEOREMA DE LAS REBANADAS EN EL CÁLCULO DE LA DFT-2D UTILIZANDO LA TRANSFORMADA DE RADÓN PERIÓDICA DISCRETA

Tesis para optar por el título profesional de Ingeniero Electrónico

AUTOR

RODRIGO RAÚL MENDOZA TRELLES

ASESOR

CESAR ALBERTO CARRANZA DE LA CRUZ

Lima - octubre del 2019

Resumen

En el ámbito del procesamiento digital de imágenes, la DFT-2D se utiliza para diversos propósitos como: detección de ruido, aplicación de filtros, tomografía computarizada, etc [1]. Algoritmos rápidos para su cálculo, como la FFT ("Fast Fourier Transform") permiten reducir su complejidad computacional. Esto es posible gracias al uso de recursividad y separabilidad al procesar una cantidad NxN de datos que igualen una potencia de dos (es decir $N = 2^p$, p entero). Sin embargo, es deseable que el aumento de velocidad se pueda dar también en tamaños que no son potencia de dos, con el propósito de tener mayores posibilidades en cuanto a tamaños de imagen.

Por lo anterior mencionado, esta tesis se enfocó en imágenes de tamaño $N \times N$ donde N es un número primo. Por ejemplo, hay 168 números primos menores a 1000, mientras que solo hay 9 números enteros positivos potencia de 2 en ese rango. El método de cálculo de la DFT-2D que se utilizó fue el de la aplicación de la Transformada de Radón Periódica Discreta o DPRT (la cual trabaja con números primos) y seguidamente el Teorema de las Rebanadas de Fourier Discreto o DFST [2]. Se tomó como modelo de solución la FDPRT [3], arquitectura que utiliza hardware en paralelo como técnica de HPC (High Performance Computing). Esta es capaz de calcular la DPRT en tiempo lineal.

En el trabajo realizado diseñó una arquitectura que permita calcular el DFST en tiempo lineal. Para esto, se priorizó la reducción del tiempo de ejecución mediante el uso de hardware paralelo, aunque esto significó que los recursos crezcan de forma cuadrática. De esta manera, se diseñó una arquitectura y un algoritmo que permiten calcular y mapear los puntos de la matriz de Fourier desde la matriz en el espacio de Radón en $2 + \lceil \log_2 N \rceil + N$ ciclos de reloj (asintóticamente O(N)). Asimismo, se cuantificó la cantidad de recursos utilizados en la arquitectura (conversores de punto fijo a punto flotante, sumadores, multiplicadores, etc.) en función del tamaño de la imagen, lo cual permitió corroborar que estos crecen cuadráticamente. Finalmente, se validó el funcionamiento del algoritmo usando el software Matlab R2013a y se realizó una comparación teórica con librerías actuales (FFTW y MKL) para cálculo rápido de la DFT-2D utilizando C en el entorno de desarrollo Visual Studio 2019.

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título	: Diseño de una arquitectura rápida para la aplicación del Teorema de las Rebanadas en el cálculo de la DFT-2D utilizando la Transformada de Radón Periódica Discreta
Área	: Procesamiento Digital de Imágenes
Asesor	: Dr. Ing. Cesar Alberto Carranza De La Cruz
Alumno	: Rodrigo Raúl Mendoza Trelles
Código	: 20140964
Fecha	: 24/09/2019

Descripción y Objetivos

En el ámbito del procesamiento digital de imágenes, la DFT-2D permite hallar el espectro de frecuencias de una imagen con diversos propósitos como: detección de ruido, aplicación de filtros, tomografía computarizada, etc. Algoritmos rápidos para su cálculo, como la FFT ("Fast Fourier Transform") permiten reducir su complejidad computacional. Esto es posible gracias al uso de recursividad y separabilidad al procesar una cantidad NxN de datos que igualen una potencia de dos (es decir N=2^p, p entero). Sin embargo, es deseable que el aumento de velocidad se pueda dar también en tamaños que no son potencia de dos, con el propósito de tener mayores posibilidades en cuanto a tamaños de imagen. Por ello es necesario enfocar en imágenes de tamaño NxN donde N es un número primo. Se sabe que es posible calcular la DFT-2D a través de la Transformada de Radón (la cual trabaja con números primos) y el Teorema de las Rebanadas de Fourier Discreto o DFST ("Discrete Fourier Slice Theorem"). Actualmente existen arquitecturas rápidas capaces de calcular la DPRT ("Transformada de Radón Periódica Discreta"). No obstante, aún no se ha diseñado una arquitectura que permita calcular velozmente el DFST.

El objetivo general del presente trabajo de tesis es contribuir al estado del arte con una nueva forma de calcular rápidamente la DFT-2D en imágenes de dimensión prima.

Los objetivos específicos son:

i. El diseño de una arquitectura rápida para el cálculo del DFST.

ii. El diseño del algoritmo para la mencionada arquitectura.

iii. Cuantificar el tiempo de ejecución del algoritmo propuesto en términos de ciclos de reloj.

iv. Comprobar si se calcula la DFT-2D en tiempo lineal usando la DPRT y el DFST.

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título : Diseño de una arquitectura rápida para la aplicación del Teorema de las Rebanadas en el cálculo de la DFT-2D utilizando la Transformada de Radón Periódica Discreta

Índice

A Dios y a la Virgen María por darme salud y fortaleza.
A Ana María, mi mamá, por su amor, sacrificio y confianza.
A Raúl, mi papá, por su amor, esfuerzo y apoyo.
A mi hermana Fátima, por ser mi mayor fuente de alegría.
A mis primos Alfredo, Karla, Andrea, Paola y Coco, por su ejemplo.
A mis tías, Pochy, Susy, Amparo y María Luisa, por sus consejos, cariño y preocupación a lo largo de mi vida.
A mis tíos, Héctor, Carlos, Pablo y Chino, por su ejemplo de vida y apoyo.
A mis abuelitos Rolo, Donani, Lino y Angélica.
A mis tíos Lino y Elisa, por su bondad y apoyo.
A Ximena, por su amistad, apoyo y cariño durante estos años.
A mis hermanos de la JuFra.
A mis amigos del colegio y la universidad.
A mi asesor, Dr. César Carranza, por su tiempo y conocimiento.
Muchísimas gracias a todos.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

En el ámbito del procesamiento digital de imágenes, la DFT-2D se utiliza para diversos propósitos como: detección de ruido, aplicación de filtros, tomografía computarizada, etc [1]. Algoritmos rápidos para su cálculo, como la FFT ("Fast Fourier Transform") permiten reducir su complejidad computacional. Esto es posible gracias al uso de recursividad y separabilidad al procesar una cantidad NxN de datos que igualen una potencia de dos (es decir $N = 2^p$, p entero). Sin embargo, es deseable que el aumento de velocidad se pueda dar también en tamaños que no son potencia de dos, con el propósito de tener mayores posibilidades en cuanto a tamaños de imagen.

Por lo anterior mencionado, esta tesis se enfoca en imágenes de tamaño NxN donde N es un número primo. Se sabe que es posible calcular la DFT2-2D a través de la Transformada de Radón (la cual trabaja con números primos) y el Teorema de las Rebanadas de Fourier Discreto o DFST ("Discrete Fourier Slice Theorem") [2]. En la actualidad, existen arquitecturas rápidas que permiten el cálculo rápido de la DPRT, como la FDPRT [3] y la DPRT escalable [1]. No obstante, aún no se ha diseñado una arquitectura que permita calcular velozmente el DFST.

El presente trabajo está dividido en cuatro capítulos y una sección de conclusiones y recomendaciones. El primer capítulo hace referencia al estado del arte de tecnologías que permiten el cálculo rápido de la DFT-2D y arquitecturas rápidas de la DPRT, así como la presentación de la motivación de la tesis y los objetivos planteados. El segundo capítulo describe la teoría necesaria para la resolución de problema: el DFST y una arquitectura rápida que usa hardware en paralelo para el cálculo de la DPRT, en la cual se basa el modelo de solución. El tercer capítulo describe la metodología empleada para el diseño de la arquitectura y el algoritmo que permiten la obtención de la matriz en el espacio de Fourier a través del DFST. El cuarto y último capítulo presenta los resultados en cuanto a los recursos utilizados y el tiempo de ejecución total del proceso en función del tamaño de la imagen. De igual manera, se muestran los resultados de la simulación de la arquitectura propuesta en el software Matlab R2013a y de la comparación teórica con librerías actuales para el cálculo de la DFT-2D (FFTW y MKL).

CAPÍTULO 1: Motivación, estado del arte y objetivos

1.1 Motivación

La DFT-2D tiene un amplio rango de aplicaciones en el ámbito del procesamiento digital de imágenes. El uso de esta transformada en una imagen o matriz permite obtener sus espectro de frecuencias con diversos propósitos como detección y eliminación de ruido, aplicación de filtros, detección de líneas, tomografía computarizada, etc. [1]

Con el tiempo han surgido diferentes algoritmos para el cálculo de la DFT-2D como la FFT ("Fast Fourier Transform") que permite reducir la complejidad computacional. Esto se traduce en un aumento de la velocidad de cálculo al cual se llega por medio del uso de recursividad al procesar una cantidad N de datos que igualen una potencia de dos (es decir $N = 2^p$, p entero).

Sin embargo, es deseable que el aumento de velocidad se de en el procesamiento de cantidades de datos que no sean únicamente potencia de dos. Por esta razón, esta tesis se enfoca en el cálculo de la DFT-2D para imágenes de dimensión NxN donde N es un número primo. De esta manera se puede obtener una reducción en el tiempo de ejecución en una mayor posibilidad de tamaño de imágenes.

La teoría de la DFT2-2D permite demostrar que es posible calcular la misma, a través de la transformada de Radón y el Teorema de las Rebanadas de Fourier Discreto o DFST ("Discrete Fourier Slice Theorem"), [2]. Actualmente existen arquitecturas rápidas capaces de calcular la Transformada de Radón Periódica Discreta, [3]. Sin embargo, no existe aún una arquitectura que permita calcular rápidamente el Teorema de las Rebanadas de Fourier discreto.

La motivación de esta tesis es contribuir al estado del arte con la propuesta de una nueva arquitectura rápida que permita el cálculo en tiempo lineal de la Transformada de Fourier Discreta 2D utilizando la DPRT ("Transformada de Radón Periódica Discreta").

1.2 Estado del arte

1.2.1 Transformada Discreta de Fourier 2D

En el área del procesamiento de señales, la Transformada Discreta de Fourier (DFT) permite transformar una señal discreta en el dominio del tiempo a su representación discreta en el dominio de la frecuencia. Esta se desarrolla en una dimensión y para una variable. Para una función discreta $f(x)$, $x = 0, 1, 2, \dots, N-1$, la DFT se obtiene por

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)e^{-j2\pi ux/N} \quad (1.1)$$

donde $j = \sqrt{-1}$ y $u = 0, 1, 2, \dots, N-1$. [4].

De la misma manera, la Transformada Discreta de Fourier 2D (DFT-2D) tiene gran importancia en el procesamiento de imágenes. Para una imagen o matriz $f(x, y)$ con $x = 0, 1, 2, \dots, N-1$ e $y = 0, 1, 2, \dots, M-1$, se halla la DFT-2D mediante

$$F(u, v) = \sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x, y)e^{-j2\pi(ux+vy)/N} \quad (1.2)$$

donde $u = 0, 1, 2, \dots, N-1$ y $v = 0, 1, 2, \dots, M-1$. [4].

La obtención de un valor de $F(u, v)$ requiere la suma de todos los pixeles de la imagen, por esta razón si se tratase de una imagen (matriz) cuadrada $N \times N$, el cálculo del espectro de Fourier para estos N^2 valores sería O(N^4). Sin embargo, debido a la propiedad de separabilidad de la DFT, es posible realizar la DFT-2D aplicando una DFT-1D a lo largo de cada fila de imagen y así formar una matriz $N \times N$ intermedia. Seguidamente, se realiza una DFT-1D en cada columna de esta matriz para llegar a matriz $N \times N$ deseada de los valores de $F(u, v)$. De tal manera, la complejidad de una DFT 2D se reduce de O(N^4) a O(N^3), [5].

Asimismo, el tiempo de ejecución de la DFT se logró reducir aún más gracias al algoritmo de la Transformada Rápida de Fourier (FFT), el cual permite calcular la DFT para N puntos en O ($N \log N$) ciclos de reloj. Sin embargo, para lograr esta reducción de complejidad es necesario que en la matriz $N \times N$ en la que se quiere aplicar la FFT, N sea una potencia de 2. Es decir, $N = 2^k$

donde k es un número entero positivo. Esto se debe a que la FFT 1D representa a la DFT de tamaño N como la suma de dos DFT de tamaño $N/2$; y de esta manera usa recursividad hasta llegar a transformadas de tamaño 2. De esta manera, se logra obtener un tiempo de ejecución proporcional a $N \log_2 N$ para la DFT en una dimensión, [5].

1.2.2 FFT para cantidades de datos distintas a potencias de 2

En la sección 1.2.1 se mencionó que en el algoritmo de la FFT para una matriz $N \times N$, N debe ser una potencia de 2 para que así se pueda usar recursividad para acelerar el cálculo. Sin embargo, en la actualidad existen algoritmos que permiten calcular la FFT sin esta restricción. En esta ocasión se mencionará dos algoritmos que son los de Rader [6] y Bluestein [7]. Ambos explican cómo realizar la FFT en una matriz de $N \times N$, donde N es un numero primo. La ventaja de esto es que existe una mayor cantidad de posibilidades de tamaño de imágenes en las que se puede aprovechar la reducción de complejidad computacional que significa el uso de una transformada rápida de Fourier. Para tener una idea, existen 9 números enteros positivos potencia de 2 menores a 1000, cantidad mucho menor a los 168 primos que existen en ese rango.

El algoritmo Rader se utiliza para calcular la DFT cuando el número de datos de entrada N es primo y considerando la DFT como una convolución cíclica. Esta cantidad prima de puntos se puede calcular como parte de una convolución circular con un número mayor de puntos. Para esto se debe elegir un número altamente compuesto mayor que $2N - 4$ e insertar ceros a la secuencia, [6]. Esta técnica se conoce como zero-padding. El algoritmo de Rader para la FFT se puede usar para calcular la DFT de un grupo de datos de longitud N, y tiene una complejidad computacional O ($N \log N$) cuando N es un número primo.

El algoritmo Bluestein, al igual que el de Rader se puede usar para calcular la FFT para cuando el número de datos de entrada no es potencia de 2. Sin embargo, la diferencia radica en que el algoritmo de Bluestein se puede usar para cualquier número, no solo para los primos.

El mencionado algoritmo expresa la CZT (“Chirp Z Transform”) como una convolución y la implementa usando FFT / IFFT. Ya que la DFT es un caso especial del CZT, es posible calcular la transformada de Fourier discreta (DFT) de cualquier tamaño (incluidos los tamaños primos). De esta manera, en [7] se indica que al usar un equivalente discreto de un Chirp Filter, se puede calcular la DFT como un proceso lineal de filtrado. Asimismo, se afirma que aplicarle un Chirp Filter a una onda es equivalente a tomar su transformada de Fourier. Al usar la FFT de Bluestein para realizar este filtro, es posible calcular la DFT en un tiempo proporcional a $N \log N$ (O ($N \log N$)).

1.2.3 Transformada de Radón Periódica Discreta (DPRT)

La transformada de Radón Periódica Discreta (DPRT) es considerada la forma discretizada de la Transformada continua de Radón [8]. En la actualidad tiene un amplio rango de aplicaciones en el ámbito del procesamiento de imágenes. Algunos ejemplos de sus aplicaciones pueden ser tomografía computarizada, resonancia magnética, teledetección y geofísica [2]. Más aún, como se indica en [1], la DPRT se puede usar en restauración de imágenes, encriptación, análisis de textura y detección de líneas.

Es importante resaltar que al igual que en la transformada continua de Radón, en la DPRT se puede aplicar el teorema de las rebanadas de Fourier (“Fourier Slice Theorem”) y la propiedad de la convolución [1]. Sin embargo, los alcances que han logrado relacionar la DFT y DPRT son muchos más. A continuación, se describen algunas de las últimas contribuciones al estado del arte. En [9] se creó el primer algoritmo de la DPRT directa para calcular la Transformada de Fourier Discreta 2D (DFT2D). Después se presentó un modelo para la DPRT y un algoritmo para calcular la DPRT inversa para imágenes $N \times N$ donde N es primo [10]. Finalmente, en [2] se contribuyó con un algoritmo para calcular la DPRT para imágenes de $N \times N$ donde N es una potencia de dos.

Por mucho tiempo se supo que la complejidad computacional que conllevaba el uso de la transformada de Radón era la razón primordial por la cual no se usaba en el procesamiento de imágenes. Al principio, a ejecución por el método serial [11] para una imagen de $N \times N$ tardaba $N^3 + N^2 + N$ ciclos de

reloj, es decir una complejidad computacional de O (N^3) Con el pasar del tiempo se disminuyó el número de ciclos necesarios. Se desarrolló la ejecución de la DPRT por el método sistólico [12], el cual necesita $N^2 + N + 1$ ciclo de reloj, es decir O (N^2).

1.2.4 Arquitecturas escalables para el cálculo de la DPRT

En [1] se detalla la implementación de arquitecturas en VHDL (VHSIC Hardware Description Language). Estas fueron validadas usando un FPGA (Field-Programmable Gate Array) y lograron reducir el tiempo de ejecución aún más. Este último alcance se desarrolló para para imágenes de tamaño N x N donde N es primo. Tiene un enfoque escalable y rápido para calcular la DPRT y la DPRT inversa. Su metodología está basada en el desplazamiento paralelo y operaciones de adición.

En [3] se indica acerca del algoritmo e implementación de la arquitectura para el cálculo de la llamada FDPRT (Fast Discrete Periodic Radon Transform). Esta arquitectura es denominada rápida por su mayor velocidad en comparación con los métodos anteriores. Es capaz de calcular la DPRT y su inversa en $2N + \lceil \log_2 N \rceil + 1$ y $2N + \lceil \log_2 N \rceil + B + 2$ respectivamente, donde B es el número de bits usado para representar cada pixel de entrada.

En [1] aparece la SFDPRT (Scalable Fast Discrete Periodic Radon Transform), la cual utiliza la escalabilidad para que el usuario pueda decidir entre la rapidez (en ciclos de reloj) y el uso de los recursos disponibles, lo que también permite manejar imágenes más grandes con recursos computacionales limitados. Estos estos parámetros cumplen una relación inversa a través de un factor de escalamiento H. Este es capaz de calcular la DPRT y su inversa en $\lceil N/H \rceil (N + 3H + 3) + N + \lceil \log_2 H \rceil + 1$ y $\lceil N/H \rceil (N + H) + 2\lceil \log_2 N \rceil + \lceil \log_2 H \rceil + B + 3$ respectivamente, donde B es el número de bits usado para representar cada pixel de entrada.

En la Figura 1.1 se puede observar el concepto de la DPRT escalable, en el cual el factor de escalamiento H indica el tamaño N x H que tendrá cada una de las tiras en las que se dividirá la imagen. Con la imagen dividida en un total de K tiras, a cada una se les aplica una DPRT parcial, generándose así una

descomposición en K planos que serán sumados punto a punto para obtener de esta manera la DPRT.

Es importante mencionar que el mínimo que se puede escoger para H es 2 y el máximo es N. Para el caso en el que se requiera un menor tiempo de ejecución se debe ejecutar la DPRT en tiempo lineal ($2N + \lceil \log_2 H \rceil + 1$ ciclos de reloj) para lo cual se requiere recursos cuadráticos ($O(N^2)$). Por otra parte, para el caso en el que se use la mínima cantidad de recursos, el tiempo será cuadrático ($\lceil N/2 \rceil (N+9) + N + 2$ ciclos de reloj) para lo cual se requerirá recursos lineales ($O(N)$).

El mencionado trabajo ha sido la última contribución al estado del arte en tema de velocidad del cálculo de la Transformada Discreta Periódica de Radón y confirma que el tiempo de ejecución ya no es un inconveniente. En la Tabla 1 se pueden observar los más recientes métodos y cuantos ciclos de reloj eran necesarios para el cálculo de la DPRT.

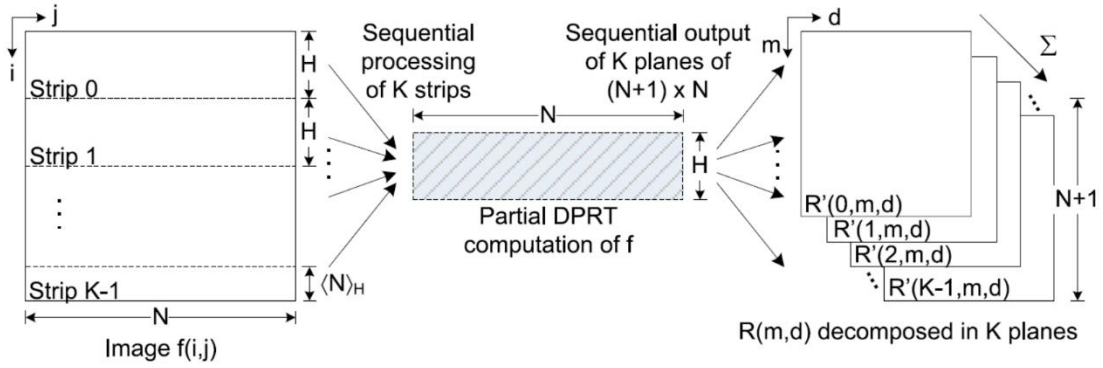


Figura 1.1. Concepto de la DPRT escalable [1]

Tabla 1: Número de ciclos de reloj para calcular la DPRT de una imagen $\boldsymbol{N} \times \boldsymbol{N}$, y H es el factor de escalamiento para la SFDPRT

Método	Ciclos de reloj
Serial [10], [11]	$N^3 + 2N + 1$
Sistólico [10], [12]	$N^2 + N + 1$
SFDPRT [1]	$\lceil \mathrm{N/H} \rceil (N + 3H + 3) + N + \lceil \log_2 H \rceil + 1$
FDPRT [1]	$2N + \lceil \log_2 N \rceil + 1$

1.2.5 The Fastest Fourier Transform in the west (FFTW)

La primera versión de la FFTW se creó en 1997 y en [13] se le describe como un paquete portable en lenguaje C para calcular la DFT en una o más dimensiones. En ese entonces y en el presente se le considera como una de las maneras más rápidas de calcular la DFT. Asimismo, es usualmente empleada en comparaciones con nuevos algoritmos rápidos para el cálculo de la Transformada de Fourier Discreta.

La última versión es la FFTW3. No se trata de un algoritmo fijo para calcular la transformada, sino que, con el objetivo de maximizar el rendimiento, el algoritmo DFT se adapta al hardware. La interacción del usuario con FFTW se produce en dos etapas: planificación, en la que FFTW se adapta al hardware, y ejecución, en la que FFTW realiza el cálculo. El principio de funcionamiento detallado en [14] se resume a continuación.

Para calcular un DFT, el usuario primero invoca el “planificador” FFTW, especificando el “problema” a resolver. El problema es una estructura de datos que describe los datos de entrada (tamaños de matriz y diseños de memoria) pero no contiene los datos en sí. Entonces, el planificador genera un “plan”, que es una estructura de datos ejecutable que acepta los datos de entrada y calcula el DFT deseado.

El planificador FFTW funciona midiendo el tiempo de ejecución real de muchos planes diferentes y seleccionando el más rápido. Asimismo, genera planes de acuerdo con reglas que descomponen recursivamente un problema en “subproblemas” más simples. Cuando el problema se vuelve

"suficientemente simple", la FFTW produce un plan que llama a un fragmento de código de línea recta optimizado que resuelve el problema directamente. Estos fragmentos se llaman codelets.

La versión actual de FFTW (3.3.8) incorpora ideas de los últimos treinta años de literatura de FFT. En [15] se encuentran algunas características, las cuales se listan a continuación:

- Es capaz de calcular la DFT de datos complejos, datos reales, datos reales simétricos pares o impares y la transformada discreta de Hartley (DHT) de datos reales.
- Los datos de entrada pueden tener una longitud arbitraria. FFTW emplea algoritmos O ($\mathrm{N} \log N$) para todas las longitudes, incluidos los números primos.
- Admite datos multidimensionales arbitrarios.
- Admite extensiones vectoriales como: SSE, SSE2, AVX, AVX2, AVX512, KCVI, Altivec, VSX y NEON.
- Incluye transformaciones paralelas (multiproceso) para sistemas de memoria compartida.

Para la mayoría de N (longitud de datos de entrada), la FFTW utiliza el algoritmo Cooley-Tukey [16]. La idea básica de esta FFT es descomponer una DFT de tamaño compuesto $\mathrm{N} = \mathrm{N1} \times \mathrm{N2}$ y expresarla en términos de dos DFT de tamaño $\mathrm{N1}$ y $\mathrm{N2}$, esencialmente, como una DFT bidimensional de tamaño $\mathrm{N1} \times \mathrm{N2}$ donde la salida es transpuesta [14].

Sin embargo, cuando los datos son reales y la cantidad es un número primo, la FFTW utiliza una adaptación del algoritmo de Rader [6] que reduce los requisitos de almacenamiento y tiempo aproximadamente en un factor de dos con respecto al caso complejo. El algoritmo primero reduce el DFT real a la transformada discreta de Hartley (DHT) [17] y luego utiliza el algoritmo de Rader adaptado al DHT. Esta manera de calcular la DFT conserva el rendimiento O ($\mathrm{N} \log N$).

Al realizar benchmarks de rendimiento de la FFTW con otras implementaciones de FFT, se encontró que la FFTW en general es superior

[14]. En cuanto a los resultados, se midió la velocidad de ejecución en MFLOPS realizando el ajuste de N a $(5N \log_2 N)/t$, donde t es el tiempo en μs (sin incluir los costos de inicialización). Este recuento de operaciones de punto flotante se basa en el número asintótico de operaciones para el algoritmo Radix-2 Cooley-Tukey.

1.2.6 Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) es una librería de rutinas matemáticas optimizadas para aplicaciones científicas, de ingeniería y financieras. Las funciones matemáticas incluyen: Álgebra lineal, FFT, estadísticas vectoriales y ajuste de datos, Matemáticas vectoriales y solucionadores diversos [18]. Las rutinas están optimizadas para procesadores Intel.

Sus algoritmos FFT funcionan en una, dos o tres dimensiones. Asimismo, es capaz de calcular rápidamente la DFT para cantidades de datos distintas a potencias de 2. Otras características de la FFT de MKL (según [18]) son:

- Versiones unidimensionales en precisión simple y doble
- Transformaciones multidimensionales, complejo a complejo, real a complejo y real a real de longitud arbitraria
- Interfaces FFTW para compatibilidad con API estándar de la industria

En la Figura 1.2 se muestra un gráfico comparativo de rendimiento (GFLOPS) entre la MKL y la FFTW para el cálculo de la DFT-2D. En el caso de la MKL se utilizó la versión 2018 Gold, mientras que para la FFTW se utilizó la versión 3.3.7. Asimismo, la configuración que se usó fue la siguiente: Intel® Core i5-7600, 1×4 cores, 3.50 GHz, 6 MB CPU Cache, 64 GB RAM, OS RHEL 7.2 [18]. Como se observa, para tamaños iguales a potencias de 2, en general la MKL es más rápida que la FFTW, y es en el rango 512×512 a 512×1024 en el cual su ventaja se encuentra muy por encima del promedio.

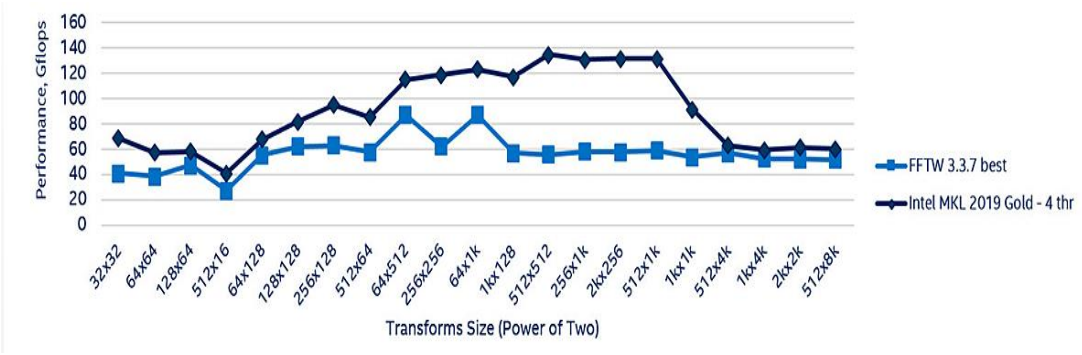


Figura 1.2. Comparación de rendimiento (GFLOPS) entre MKL y FFTW para el cálculo de la DFT-2D para tamaños iguales a potencias de 2. [18]

1.3 Objetivos

1.3.1 Objetivo principal

Diseñar una nueva arquitectura rápida para el cálculo de la Transformada Discreta de Fourier 2D (DFT-2D) utilizando arquitecturas rápidas de la Transformada de Radón Periódica Discreta (DPRT) y el Teorema de las Rebanadas de Fourier Discreto

1.3.2 Objetivos específicos

Diseñar una arquitectura rápida para aplicar el DFST (“Discrete Fourier Slice Theorem”) al cálculo de la DFT-2D utilizando la DPRT

Diseñar el algoritmo para la mencionada arquitectura

Cuantificar el tiempo de ejecución del algoritmo propuesto en términos de ciclos de reloj y el tamaño de la imagen

Comprobar si se calcula la DFT-2D en tiempo lineal usando la DPRT y la DFST.

CAPÍTULO 2: Marco teórico y modelo de solución

2.1 Marco Teórico

A continuación, se abordan los conceptos que se utilizarán para la solución de esta tesis.

2.1.1 Arquitectura rápida para el cálculo de la DPRT

La implementación de arquitecturas en dispositivos FPGA se realiza con el fin de reducir el tiempo de ejecución. Un ejemplo se encuentra en [3], donde se explica la FDPRT ("Fast Discrete Periodic Radon Transform"). Este es un método cuya arquitectura fue implementada en FPGA y se destaca por su corto tiempo de ejecución en comparación con métodos anteriores. La mencionada arquitectura calcula la DPRT de una imagen de dimensiones $N \times N$ (N = primo) en N + ⌈ $\log_2$N ⌉ + 1 ciclos de reloj, y considerando que la imagen tarda en cargar N ciclos, el total llega a ser un total de 2N + ⌈ $\log_2$N ⌉ + 1 ciclos de reloj.

Para poder ejecutar esta arquitectura, se debe considerar que se usarán todos los recursos necesarios para procesar la imagen entera. Como se explicó en el capítulo anterior, para tener una mayor velocidad es necesario un mayor uso de recursos. A diferencia de la SFDPRT [1], la FDPRT ([1], [3]) no tiene una arquitectura escalable que permite procesar la imagen por partes de tamaño variable para después sumarlas. Esta última suspende el uso de la RAM y en cambio guarda la imagen de entrada en una matriz de registro [1].

Para una imagen f de dimensiones $N \times N$, la DPRT R está dada por

$$R(m,d) = \begin{cases} \sum_{i=0}^{N-1} f(i, \langle d + mi \rangle_N), & 0 \le m \le N, \\ \sum_{j=0}^{N-1} f(d,j), & m = N, \end{cases} \tag{2.1}$$

donde $d = 0, 1, 2, \dots, N-1$ y $m = 0, 1, 2, \dots, N-1$ corresponden a las columnas y filas (respectivamente) de la imagen en el dominio de Radón [1]. En la Figura 2.1 se muestran los parámetros correspondientes a (2.1). Y, también, la imagen f que a través de la DPRT llega a ser una imagen en el dominio de Radón R.

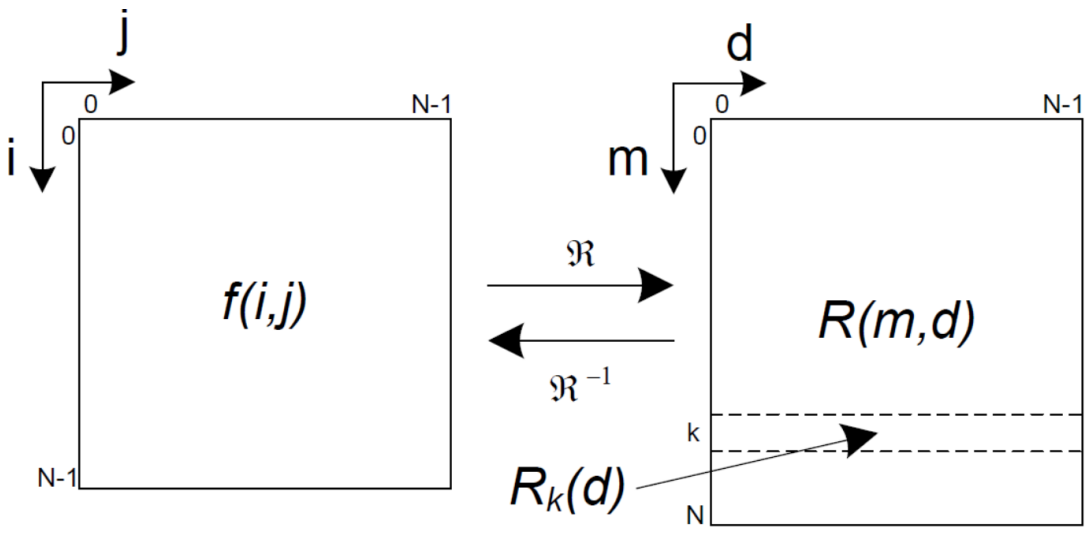


Figura 2.1. Ilustración de la DPRT y su inversa (iDPRT) para una imagen f de dimensiones N x N [1]

Asimismo, el cálculo completo de la matriz R se puede realizar con O(N) utilizando la arquitectura de la FDPRT. Esta es una complejidad computacional mucho menor en comparación con una ejecución serial ([10], [11]) o sistólica ([10], [12]) con $O(N^3)$ y $O(N^2)$ respectivamente.

En la Figura 2.2 se muestra el diagrama de tiempo en función de ciclos de reloj que tarda en ejecutar el algoritmo de la FDPRT. Como se puede observar, primero se carga la imagen. Después se calculan todos puntos de cada fila de la matriz R en paralelo y usando pipeline. Se usan registros de desplazamiento que en [3] se denominan “Circular Shift Registers”.

Para ejemplificar gráficamente el funcionamiento de la FDPRT, se puede observar la Figura 2.3(a). En esta se encuentra la implementación del hardware para una imagen d 7 x 7 (N = 7, primo). Seguidamente, en la Figura 2.3(b), se observa que la arquitectura consiste en un núcleo FDPRT y una máquina de estados o FSM (Finite State Machine, en inglés). La imagen es ingresada fila por fila, la carga total requiere N ciclos.

Después, la FSM se encarga del desplazamiento, el cual demora un ciclo. Ya que la arquitectura utiliza Pipeline, puede ejecutar las N – 1 primeras proyecciones en N – 1 ciclos, (6 para este ejemplo). Las últimas dos proyecciones requieren dos ciclos como indica la Figura 2.2. Finalmente, ocurre una latencia en la última suma que dura $\lceil \log_2 N \rceil$, 3 ciclos para este

ejemplo (ver Figura 2.3(c)). En conclusión, la ejecución de la FDPRT toma 2N + [$\log_2$N] + 1 ciclos de reloj. Para el ejemplo de la Figura 2.3(a) de una imagen de 7 x 7, toma 17 ciclos.

Se debe mencionar que para este ejemplo también se toma en cuenta que la imagen tiene B bits por pixel. Las entradas y salidas que se pueden ver en la Figura 2.3(a) corresponden a una imagen de dimensiones 7 x 7. En la Figura 2.3(c) se puede observar la utilización de sumadores con pipeline, registros y multiplexores. En [3] se indica que se requiere $N \times N$ registros de B bits, $N \times N$ multiplexores de 3 entradas de B bits y N árboles sumadores. Cada árbol sumador necesita $2^{\lceil \log_2 N \rceil} - 2$ registros y $2^{\lceil \log_2 N \rceil - 1} - 1$ sumadores.

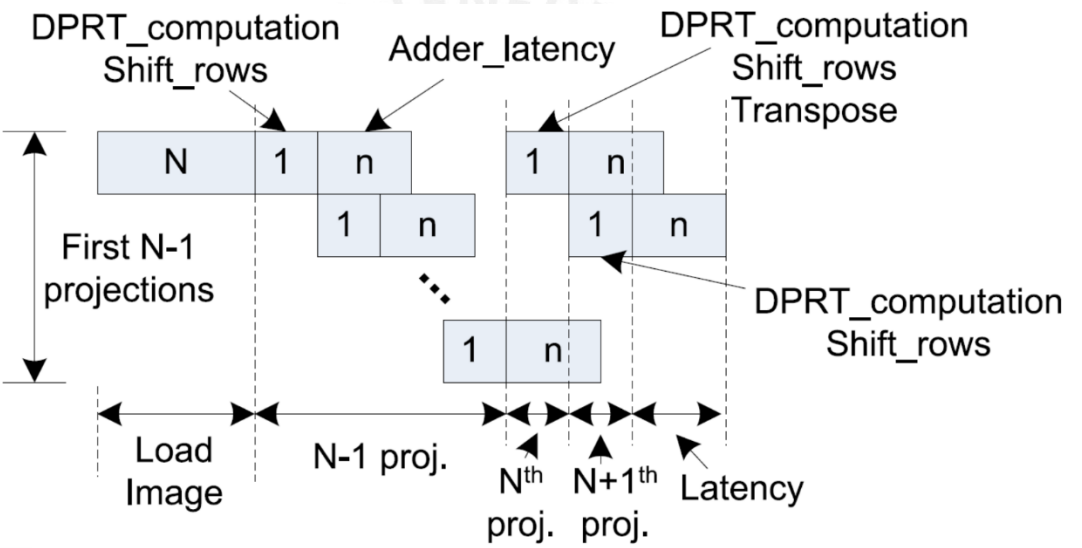


Figura 2.2. Tiempo de ejecución de la FDPRT con $\mathrm{n} = \lceil \log_2 N \rceil$ [1]

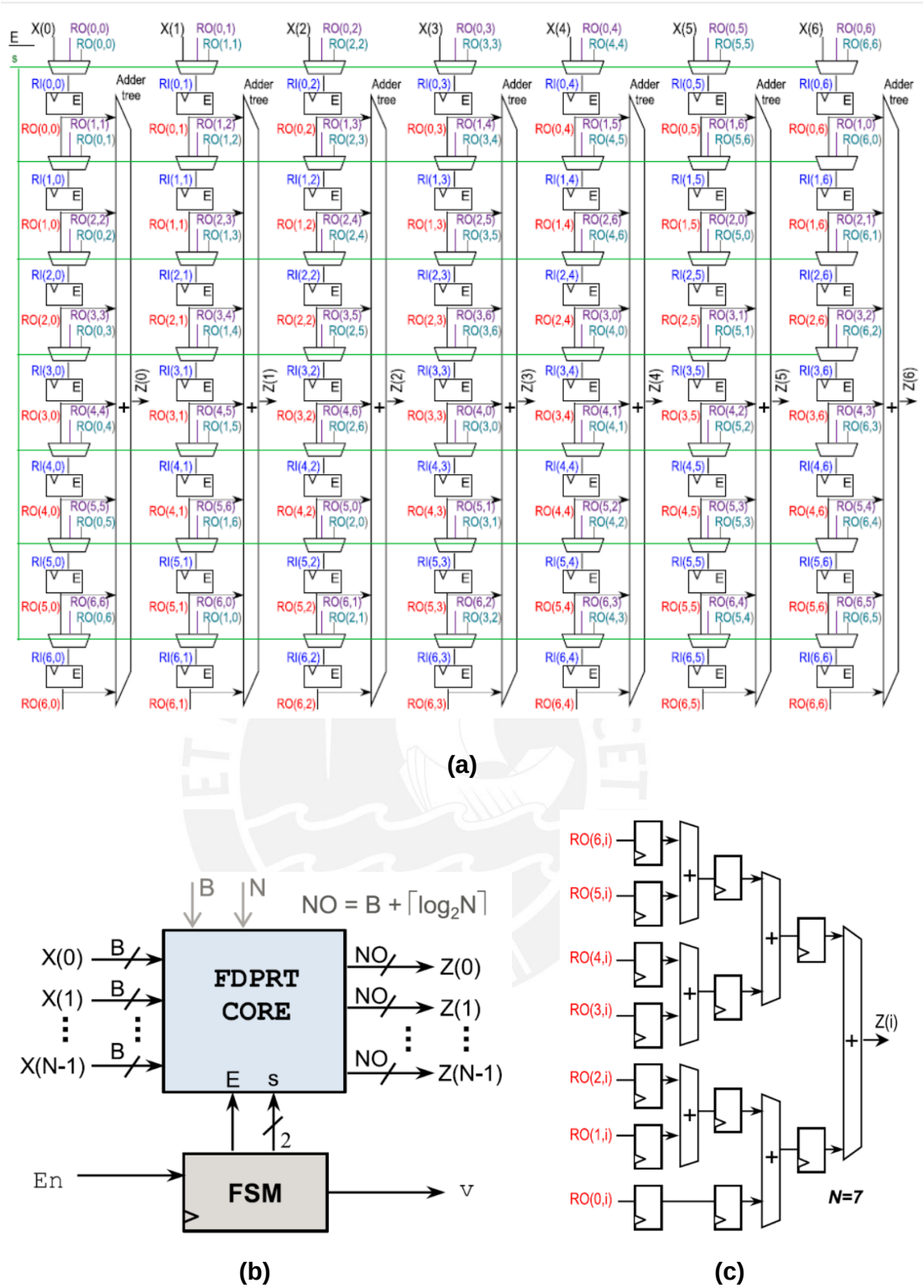


Figura 2.3. (a) Estructura del núcleo FDPRT [1]. **(b)** Núcleo FDPRT y máquina de estados (FSM) [1]. **(c)** Arquitectura con arboles sumadores y Pipeline [1].

2.1.2 Teorema de las Rebanadas de Fourier Discreto

El Teorema de las rebanadas de Fourier describe la relación entre el espectro de Fourier proyectado y el espectro original de una determinada imagen. En [2] se le considera como una transformación de coordenadas en el dominio de Fourier. Asimismo, en [2] se presenta una relación similar para el dominio de Radón: el Teorema de las Rebanadas de Fourier Discreto o DFST ("Discrete Fourier Slice Theorem"). Este permite llegar al dominio de Fourier partiendo desde una imagen en el dominio de Radón y a través de un mapeo de cada punto. En la Figura 2.4 se puede observar lo anterior mencionado.

Al asumir que se trabaja con una imagen f de dimensiones $N \times N$, en [2] se mencionan dos casos: cuando N es primo y cuando N es potencia de dos. Como se mencionó al inicio de esta tesis, se trabajará con N = primo.

Se debe calcular la DPRT de f, la cual se denomina R. Las dimensiones de R son $(N+1) \times N$. Posteriormente, para proceder con el cálculo del DFST y llegar al dominio de Fourier, se aplica

$$F(\langle -mv \rangle_N, v) = \sum_{d=0}^{N-1} R(m,d) e^{-j\frac{2\pi(vd)}{N}} \quad (2.2)$$

$$F(u,0) = \sum_{d=0}^{N-1} R(N,d) e^{-j\frac{2\pi(ud)}{N}} \quad (2.3)$$

donde $u = 0, 1, 2, \ldots, N-1$ y $v = 0, 1, 2, \ldots, M-1$ corresponden a las columnas y filas (respectivamente) de la imagen en el dominio de Fourier [4]. Esto se ilustra en la Figura 2.5.

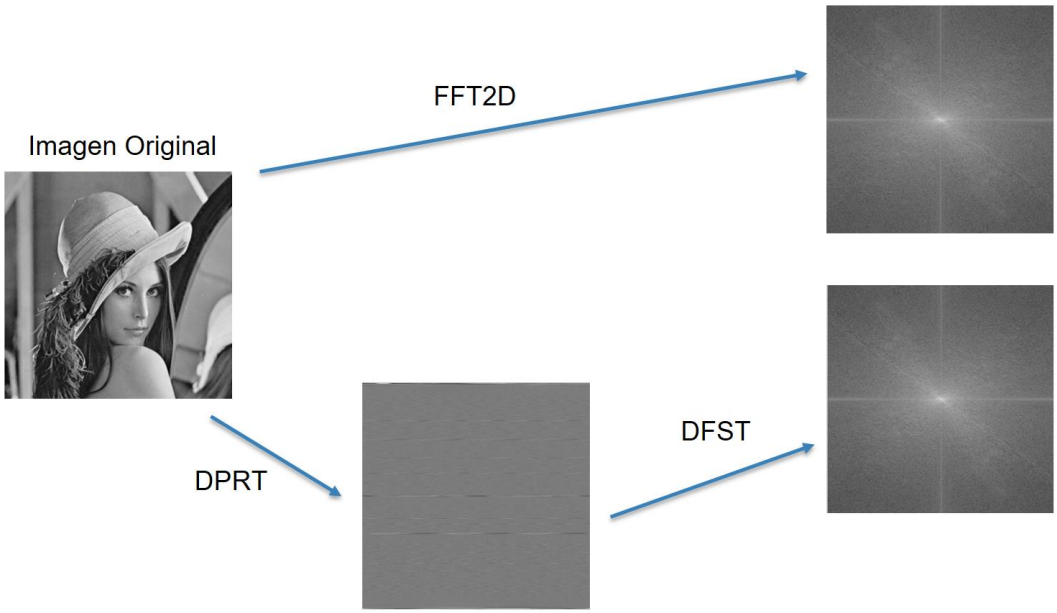


Figura 2.4. Uso del DFST para llegar al dominio de Fourier.

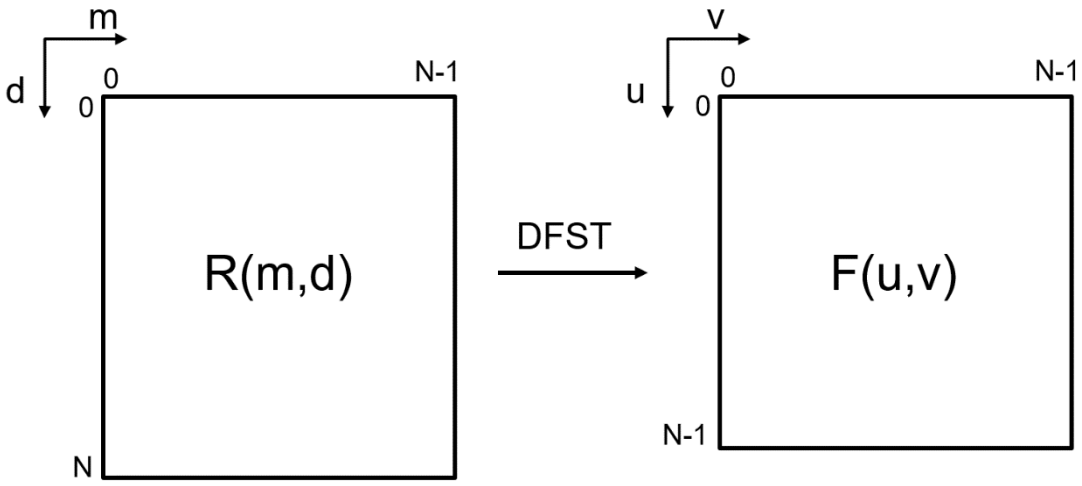


Figura 2.5. Ilustración del DFST para obtener el espectro de Fourier desde el dominio de Radón.

La demostración de (2.1) y (2.2) se encuentra en [2]. En otras palabras, con el cálculo de la DFST se logra realizar el mapeo correspondiente del espectro de Fourier de f con el previo cálculo de su DPRT.

Debido a las dimensiones de la matriz R, el proceso de cálculo del DFST requiere $N^2 + N$ operaciones. Si bien F solo tiene N^2 pixeles, esas N operaciones de más se dan debido a que el pixel $F(0,0)$ se calcula N +1 veces.

2.2 Modelo de solución

Se quiere realizar el diseño de una arquitectura rápida que permita calcular el DFST (“Discrete Fourier Slice Theorem), [2]. Este indica que después de ser aplicada la transformada de Radón a una matriz o imagen f y obteniéndose así la matriz R, se puede calcular la DFT-2D a través de (2.2) y (2.3).

El primer paso es calcular la DPRT. Para esto ya se ha diseñado e implementado una arquitectura rápida que con el uso de sumadores y pipeline puede calcular la DPRT para números primos en tiempo lineal, [3]. El algoritmo para el cálculo se encuentra en la Figura 2.6.

El segundo paso para calcular la DFT-2D usando la DPRT, es calcular y mapear los puntos de la matriz de Fourier a través del DFST. Sin embargo, el cálculo para una imagen $N \times N$ donde N es primo (imágenes objetivo de esta tesis) requiere una complejidad computacional que hacen que este camino no sea atractivo en comparación con el uso directo de una FFT-2D.

Entonces, se busca el diseño de una arquitectura rápida que reciba una matriz en el espacio de Radón y a usando el DFST, permita obtener el espacio de Fourier. Para lograr diseñar esta arquitectura se utiliza como referencia a [1]. No obstante, es necesario agregar determinados componentes y otras etapas que se explican en el Capítulo 3.

Compute in parallel $C_o(d)$
for k = 1 to p-1 **do**
    Shift in parallel the last p-1 rows
    HCSR(i), i=1, …, p-1
    Compute in parallel C_k(d)
end for
Shift in parallel the last p-1 rows
    HCSR(i), i=1, …, p-1
    Transpose of the image
Compute in parallel $Bo(d)$

Figura 2.6. Algoritmo para el cálculo de la FDPRT [3]

CAPÍTULO 3: Diseño de la solución

3.1 Aspectos generales del diseño

Es común que en el diseño de arquitecturas se busque optimizar tanto el uso de recursos como el tiempo de ejecución. En el capítulo anterior se explicó el tema de escalabilidad como el compromiso que había entre estos dos parámetros. Sin embargo, el objetivo de la presente tesis es lograr agilizar al máximo el proceso de cálculo del DFST. Por lo mencionado, se prioriza el emplear el mínimo tiempo posible en el cálculo sin importar la cantidad de hardware utilizado.

El método en el que se basa el diseño es la paralelización del proceso de cálculo. La optimización en cuanto al tiempo se traduce en el uso de hardware paralelo (pipeline), lo cual permitirá una rápida obtención de los coeficientes de Fourier. Asimismo, el proceso de mapeado de los mismos, finalizará en la de Fourier (obtención de la DFT-2D).

A manera de esquematización de la solución propuesta, se presenta el diagrama de bloques en la Figura 3.1. El parámetro de entrada es la matriz R en el dominio de Radón. Su dimensión es de $N\ x\ (N+1)$ y fue calculada mediante la DPRT [3]. Cada componente de la matriz es un número de punto fijo. En la aplicación del DFST se multiplica determinados puntos de la matriz R con números en punto flotante correspondientes a la parte exponencial de las mencionadas ecuaciones. Los productos se suman de una manera determinada para obtener los coeficientes de Fourier.

Es muy importante la presunción de que las constantes exponenciales (en su mayoría números complejos) se encuentran previamente calculadas y almacenadas en memoria. Dichas constantes son números en punto flotante, a diferencia de los coeficientes de R. Por ello, se debe añadir una etapa de conversión de punto fijo a flotante para los coeficientes de la matriz R.

La última parte del proceso consiste en la ubicación de los coeficientes calculados en la matriz F de dimensión $N\ x\ N$. A esta operación se le llama mapeado en memoria según el diagrama de bloques de la Figura 3.1. La designación de las direcciones en que deben ser almacenados obedecen al

patrón establecido por (2.2) y (2.3). Cada parte del proceso de cálculo del DFST se detalla durante el presente capítulo.

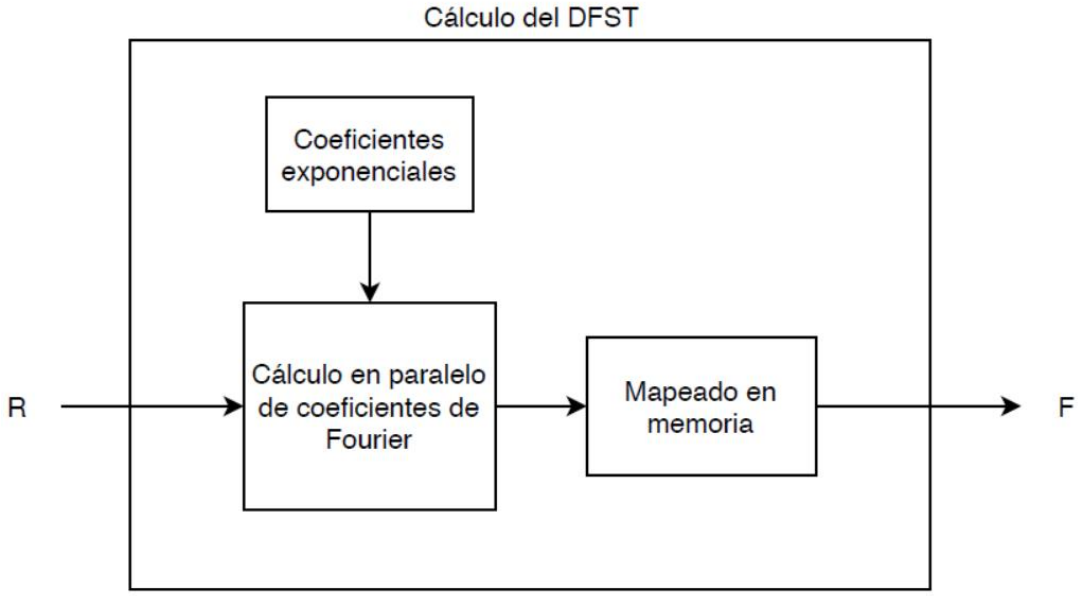


Figura 3.1. Diagrama de bloques del cálculo del DFST

3.2 Coeficientes exponenciales

Se les llama coeficientes exponenciales a las constantes (en su mayoría complejas) que se utilizan en el cálculo del DFST. Como se mencionó previamente, se asumirá que estas se encuentran previamente calculadas y almacenadas.

En las ecuaciones 2.2 y 2.3, se puede observar que la parte exponencial es una función que depende de tres parámetros: el tamaño de la imagen N, la posición relativa (columna de la matriz) del coeficiente de Radón: d (donde $d = 0, 1, 2, \dots, N-1$), y la posición (columna o fila) donde el resultado será ubicado en la matriz de Fourier (u, v).

De acuerdo a lo mencionado se puede definir la función exponencial compleja K, para una matriz de dimensión $N\,x\,N$ donde N es primo, de la siguiente manera:

$$K(x,y) = e^{-j\frac{2\pi(xy)}{N}} \tag{3.1}$$

donde $j = \sqrt{-1}$, $x = 0, 1, 2, \dots, N-1$, $y = 0, 1, 2, \dots, N-1$

3.3 Diseño del bloque de cálculo en paralelo de coeficientes de Fourier

Este primer bloque permite calcular los puntos que luego serán mapeados para formar la DFT-2D. Como se mencionó previamente, para calcular la DFT-2D con el método propuesto en esta tesis es necesario multiplicar determinados puntos de la matriz de Radón con las constantes exponenciales.

Se considera que constantes exponenciales son de tipo punto flotante y de 64 bits. La razón de la elección de esta cantidad de bits corresponde a que la mitad de ellos son utilizados para la parte real de la constante, mientras que los 32 bits restantes están designados a la parte imaginaria.

3.3.1 Cálculo mediante el método tradicional

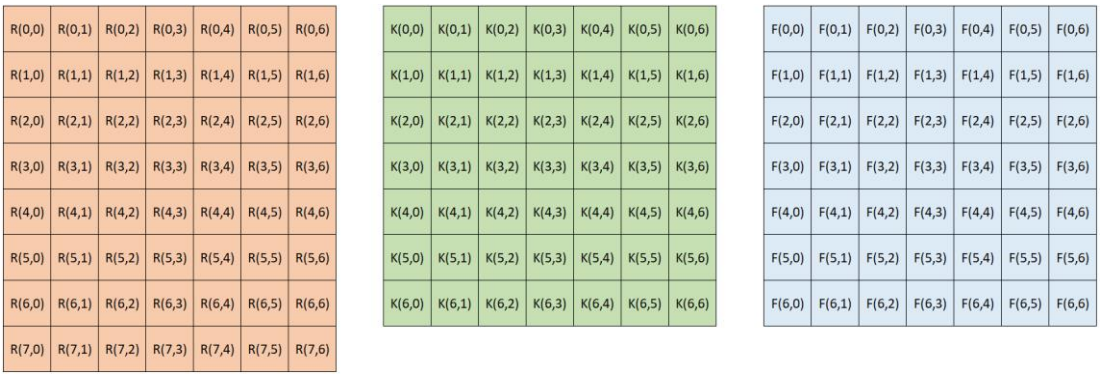
Tradicionalmente, el cálculo de los coeficientes de Fourier se realiza de una manera determinada por las ecuaciones 2.2 y .2.3. En esta sección se considera una matriz $N \times N$ con N =7. Con este tamaño de imagen se puede ejemplificar de manera gráfica el cálculo de puntos de la matriz de Fourier.

En la Figura 3.2 (a) se muestra la imagen R en el dominio de Radón. Ciertos puntos de esta matriz deben ser multiplicados con determinados puntos de la matriz de constantes exponenciales (Figura 3.2 (b)) para obtener los puntos que formarán la matriz de Fourier (Figura 3.2 (c)). La primera fila de R (m=0) permite obtener la primera dirección de F. Esta fila debe ser multiplicada punto a punto con las filas de la matriz K. Al realizar la multiplicación con la primera fila de K (x=0) y sumar estos productos se obtiene F (0,0). En la Figura 3.3 se aprecia que los puntos a ser multiplicados tienen el mismo color en la escala de grises. Es decir que para N=7:

$$F(0,0) = R(0,0) * K(0,0) + R(0,1) * K(0,1) + R(0,2) * K(0,2) + R(0,3) * K(0,3) + R(0,4) * K(0,4) + R(0,5) * K(0,5) + R(0,6) * K(0,6)$$

Análogamente, la multiplicación punto a punto con la siguiente fila de K (x=1) y la posteriormente suma de estos resultados permite obtener $F(0,1)$. Al realizar el mismo proceso con todas las filas de K (x=0, 1, …,6). Se obtendrá la primera dirección de F. La ubicación de estos puntos calculados en la matriz F se observa en la Figura 3.4. En esta figura se observa que cada punto de la

dirección 0 en F corresponde a un color en la escala de grises relacionado con la fila de la matriz K que se usó para calcular el mencionado punto.



(a) (b) (c)

Figura 3.2. (a) Imagen en el espacio de Radón de tamaño 8x7. **(b)** Matriz de constantes exponenciales para N =7. **(c)** Imagen en el espacio de Fourier de tamaño 7x7.

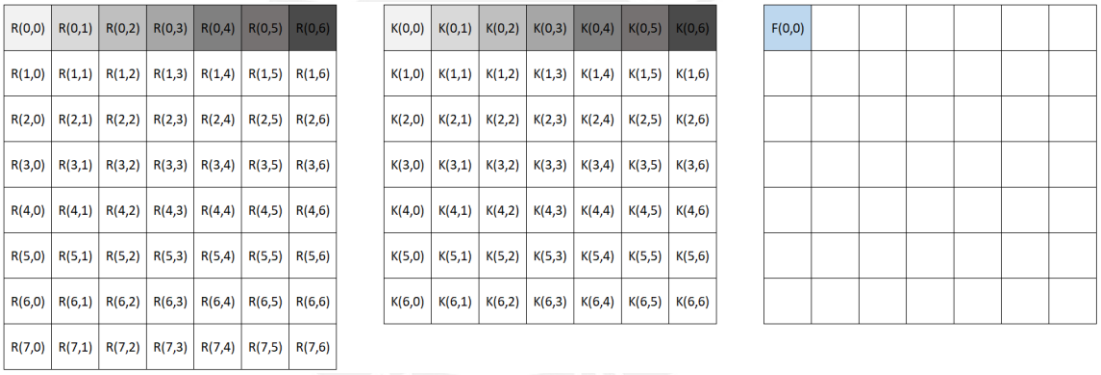


Figura 3.3. Ilustración de la obtención del punto F(0,0) utilizando la primera fila de la matriz R (m=0).

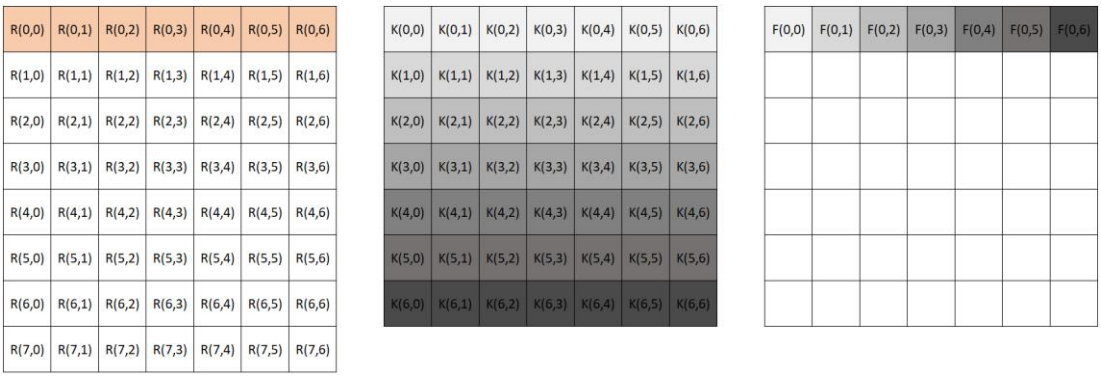
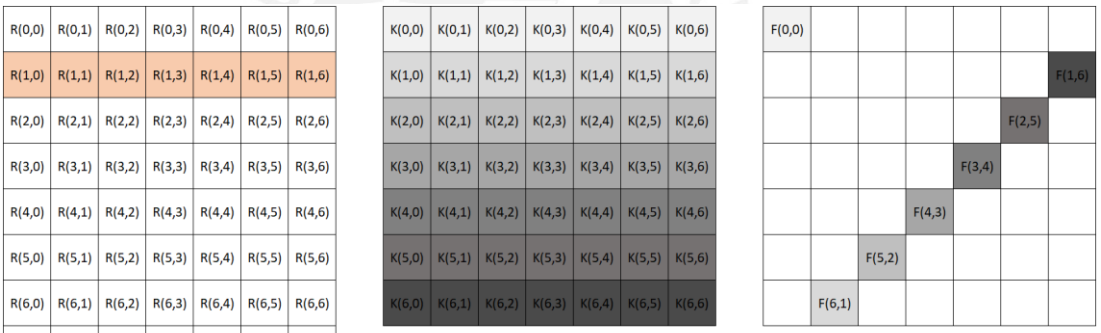


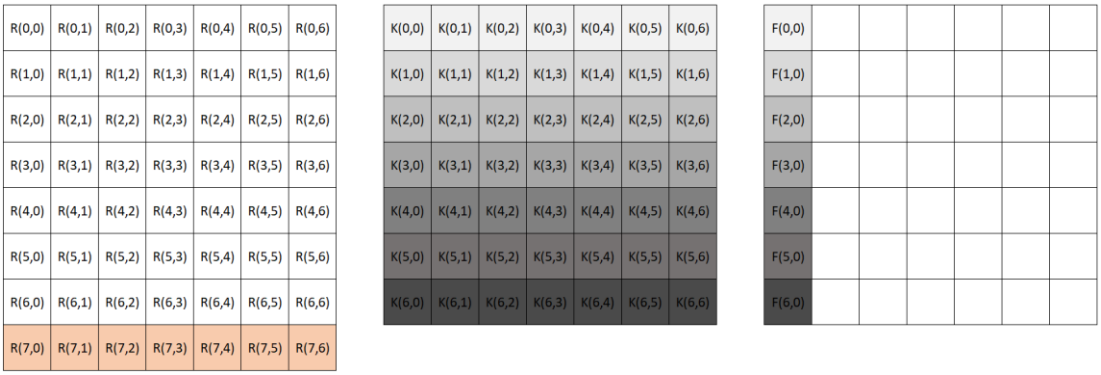
Figura 3.4. Ilustración de la obtención de la dirección 0 de F utilizando la primera fila de la matriz R (m=0) y todas las filas de la matriz K (x=0, 1, 2, …, 6).

En la Figura 3.5 (a) y Figura 3.5 (b) se muestra la ubicación de los puntos de la dirección 1 y dirección 7 respectivamente (N=7). Los valores de las filas R_1 y R_7 son multiplicados punto a punto con las constantes de todas las filas de la matriz K. Cada punto calculado de F está asociado a un color en la escala de grises que corresponde a cada fila de K.

En resumen, cada fila R_m permite calcular la dirección m de F, (m=0, 2, …, 7). Para N=7, el cálculo de las 7 primeras direcciones (m=0, 1, …, 6) corresponden a la ecuación 2.2, mientras que la última dirección de F (m=7) que se calcula con R_7 corresponde a la ecuación 2.3. En la Figura 3.6 se observa cómo se llena la matriz de Fourier mediante las 8 direcciones al variar m. En la mencionada figura se muestra que los puntos que se calculan con el valor de m están sombreados, mientras que los que ya fueron calculados tienen fondo blanco. De esta forma, se ilustra como progresivamente se llena la matriz F mediante N+1 direcciones correspondientes a las $N+1$ filas de R.



(a)



(b)

Figura 3.5. (a) Ubicación de puntos de la dirección 1 para N=7. **(b)** Ubicación de puntos de la dirección 7 para N=7.

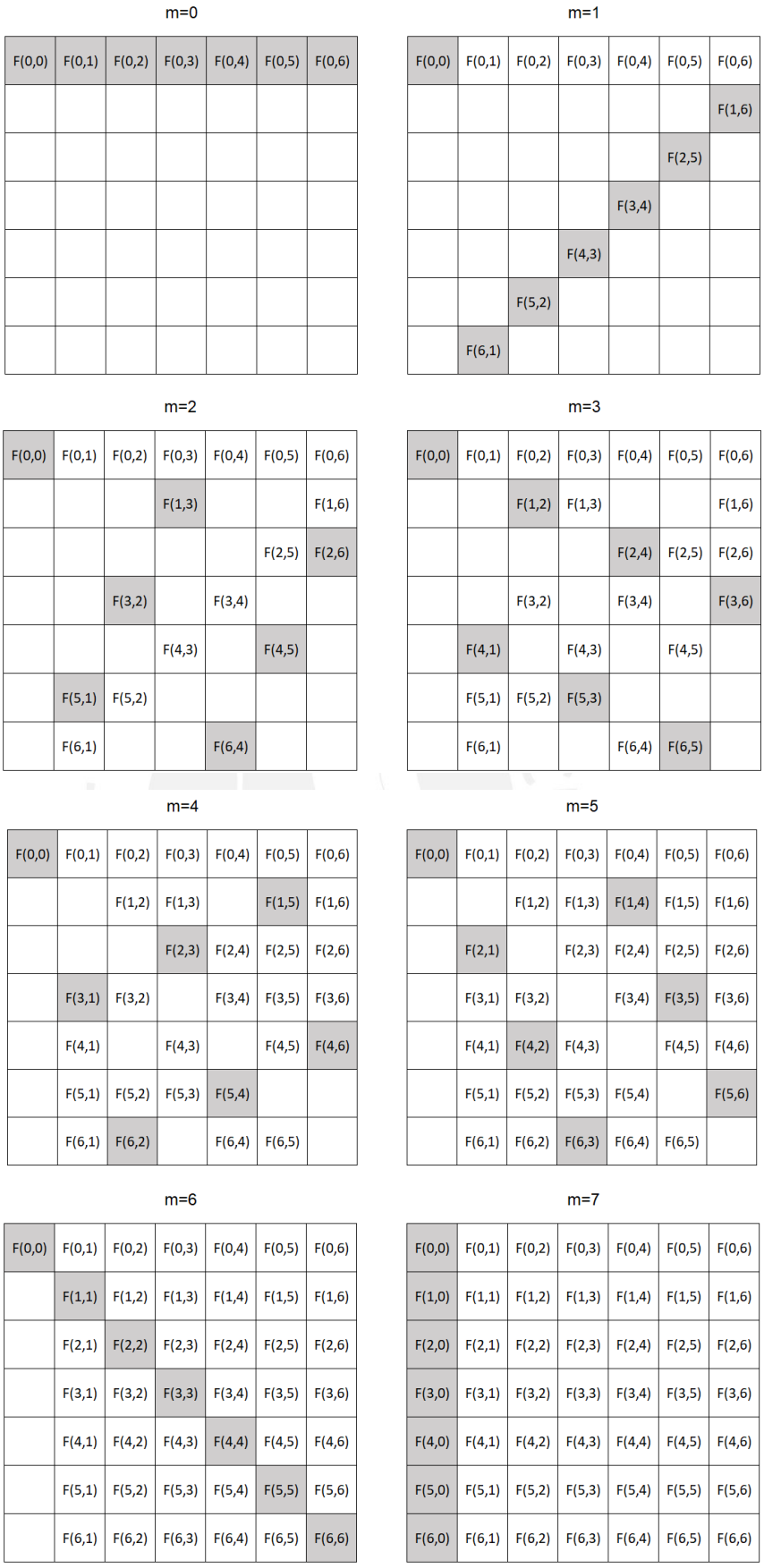


Figura 3.6. Ilustración de la obtención de la matriz F para N=7.

3.3.2 Cálculo mediante planos

En la Figura 3.6 se puede observar que en cada dirección m los puntos trazados se encuentran en columnas distintas excepto en la última dirección (m=7). Al considerar la memoria como un arreglo de N SRAM verticales, es imposible ubicar en memoria todos los puntos de la última dirección debido a que todos pertenecen a la misma columna.

La forma de cálculo que se propone en esta tesis es el uso de planos en paralelo para la construcción de la matriz de Fourier. Un plano consta de conversores de punto fijo a punto flotante, multiplicadores y un árbol sumador cuyo resultado es un punto del espacio de Fourier. Eventualmente este plano calcula toda una dirección m. El uso de registros dentro del plano permite diseñar una arquitectura basada en hardware paralelo (pipeline) para calcular los puntos correspondientes a una SRAM.

Un plano puede calcular una dirección en N ciclos más una determinada latencia. Si se utiliza $N+1$ de estos planos en paralelo, en el mismo tiempo se puede calcular la matriz F en su totalidad. Sin embargo, este cálculo se puede optimizar con respecto al tiempo de ejecución ya que en cada plano calcula el punto F (0,0).

La entrada de cada plano es un vector R_m correspondiente a una fila de la matriz de Radón. Después de esperar una latencia, a la salida del plano se obtiene un punto de la matriz de Fourier que luego será mapeado en una SRAM. Debido a que F tiene en total N columnas, se utilizarán N planos denotados como Plano (v), donde v = 0, 1, 2, …, N-1. Consecuentemente, se utilizarán N SRAM denotadas como SRAM(v), donde v = 0, 1, 2, …, N-1.

Existe una diferencia entre el Plano(0) y los demás. Esto se debe a que los puntos que calcula este plano (primera columna de F) coinciden con los puntos calculados en la dirección N por el método teórico. Por este motivo, el Plano(0) recibirá como parámetro de entrada el vector R_N, es decir la última fila de la matriz de Radón, y varían las filas de la matriz K.

Por otra parte, los planos v = 1, 2, …, N-1, tienen asociado un vector de constantes exponenciales correspondientes a una fila K_v. Las entradas R_m son las que varían en cada ciclo de reloj (m=0, 1, 2, …, N-1).

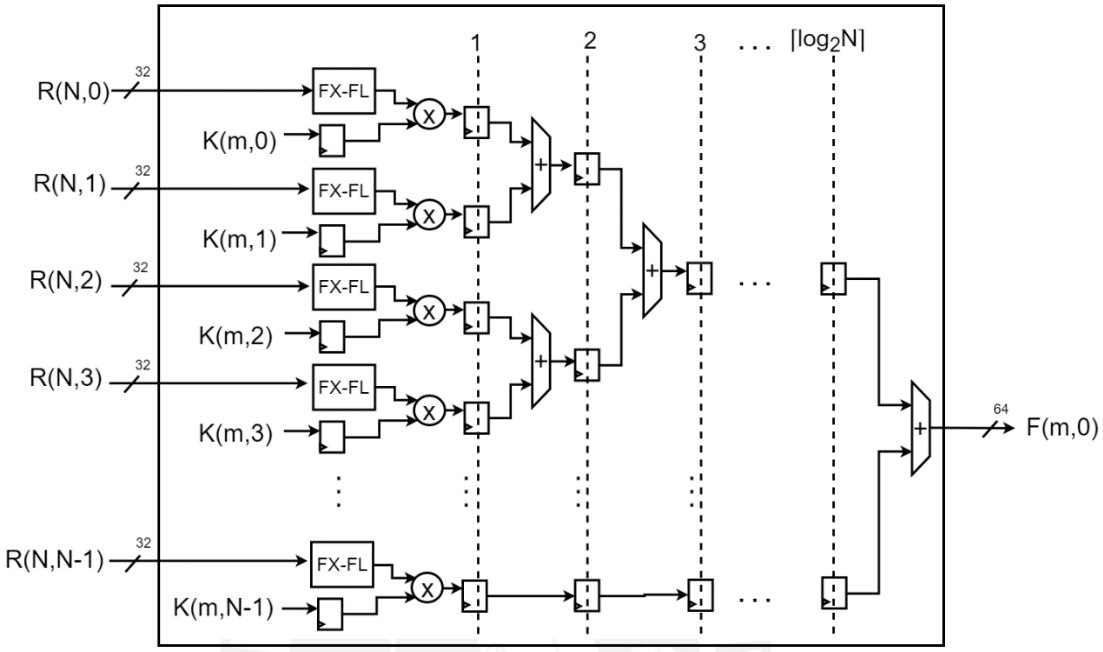


Figura 3.7. Arquitectura del Plano(0) para calcular la primera fila de la matriz de Fourier. m = 0,1, 2, …N-1. N: primo.

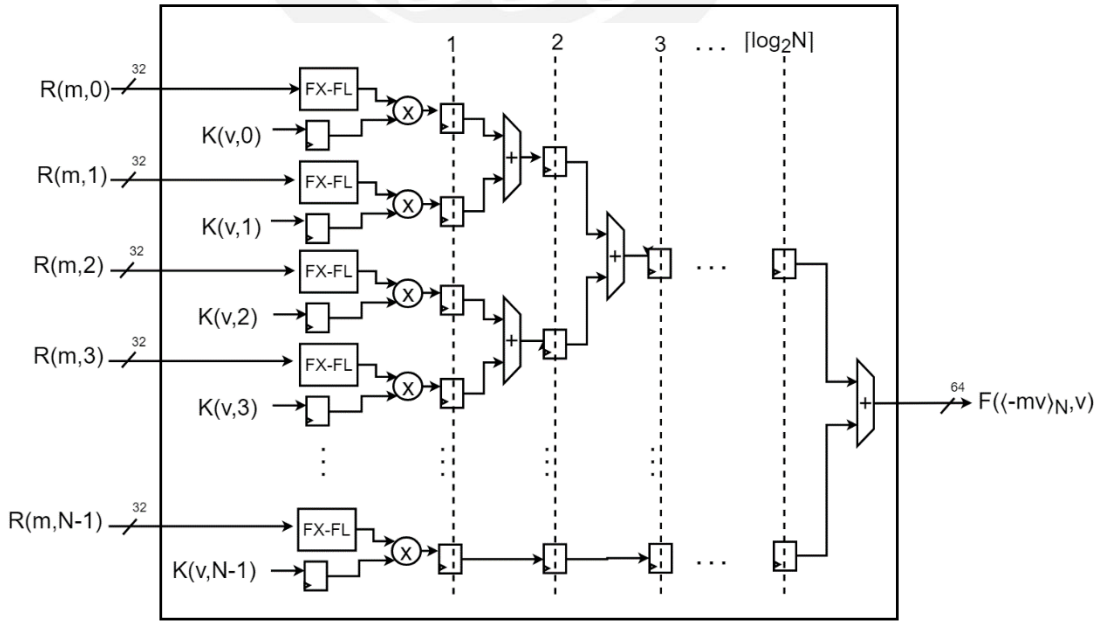


Figura 3.8. Arquitectura del Plano(v) con v = 1, 2, …, N-1 para calcular las últimas N-1 filas de la matriz de Fourier. m=0,1, 2, …N-1. N: primo.

Se observa que las arquitecturas presentadas en las Figuras 3.7 y 3.8 son similares. Ambas reciben un vector como entrada y a la salida se obtiene un punto que se ubica posteriormente en memoria. El vector Rm son N números en punto fijo de 32 bits. Antes de multiplicar estos puntos con las constantes exponenciales se añade un conversor de punto fijo a punto flotante (representado con la unidad FX-FL). El conversor debe tener como entrada un número con signo en punto fijo y generar un número en punto flotante de 64 bits con 32 bits designados para la parte real y 32 bits para la parte imaginaria. Como se sabe, la matriz de Radón solo tiene números naturales, por lo tanto, solo habrá datos en la parte real. Además, debido a que se asume que la conversión requiere un ciclo de reloj, se coloca un registro (flip flop) para la constante exponencial. De esta manera ambos factores llegan sincronizados al multiplicador.

Para poder utilizar pipeline en la arquitectura es necesario el uso de registros. Por esta razón, estos se colocan a la salida de los multiplicadores y de los sumadores. De esta manera, en cada ciclo de reloj se cambia las constantes exponenciales en el caso del Plano(0) o el vector R_m para el caso de los demás planos. Después de la etapa de multiplicación, se tienen todos los valores que se deben sumar. La manera en que se suman está explícita en las Figuras 15 y 16. Se trata de agrupar en parejas a manera de llaves y así sucesivamente hasta llegar a la respuesta final. Ya que el resultado de cada suma es almacenado en un registro, se entiende que el resultado la suma total se obtiene luego de $\lceil \log_2 N \rceil$ ciclos de reloj. Este valor es conocido como la latencia.

A manera de ejemplo, se muestra la arquitectura para N=7. Para este caso la latencia es $\lceil \log_2 7 \rceil$, es decir 3 ciclos de reloj. En la Figura 3.9 se muestra la arquitectura del Plano(0), que permite calcular la primera columna de la matriz F, SRAM(0). Asimismo, en la Figura 3.10 se muestra la arquitectura de los planos v=1, 2, …, 6, que permiten calcular los puntos del resto de las columnas de F.

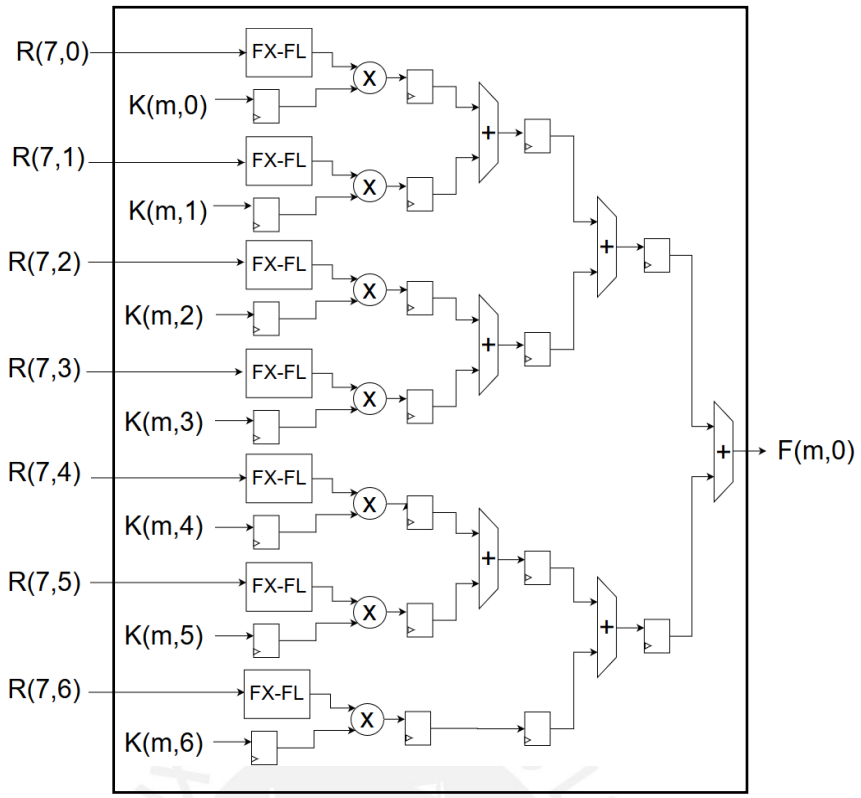


Figura 3.9. Arquitectura del Plano(0) para calcular la primera columna de la matriz de Fourier. m=0,1, 2, …, 6. N=7.

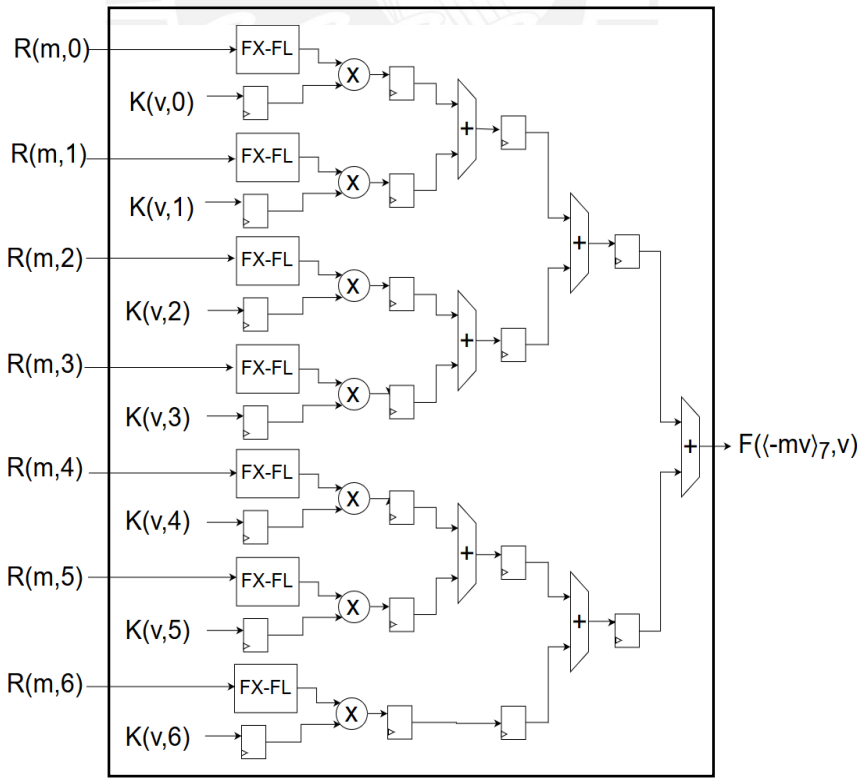


Figura 3.10. Arquitectura del Plano(v) con v=1, 2, …, 6 para calcular las últimas 6 columnas de la matriz de Fourier. m=0,1, 2, …6. N=7.

En general, un plano permite calcular todos los puntos de una columna de la matriz de Fourier en un total de $2 + \lceil \log_2 N \rceil + N$ ciclos de reloj. Esta cantidad de ciclos proviene de la Figura 3.11. En ella, se puede apreciar el uso de pipeline para el cálculo toda una columna de F. Los rectángulos celestes simbolizan el tiempo que tarda el cálculo de un punto de la matriz de Fourier desde un vector de la matriz de Radón. Para la conversión de punto fijo a punto flotante y a multiplicación se considera un ciclo de reloj respectivamente. Como se sabe, la latencia ocupa $\lceil \log_2 N \rceil$ ciclos y se representa en el gráfico mediante n. Al final se designa un ciclo para el almacenamiento en memoria. Ya que se colocan N de estos rectángulos en pipeline, se suma N-1 ciclos de reloj correspondientes al ingreso de los demás vectores.

Como se mencionó anteriormente, al utilizar N planos en paralelo es posible calcular todos los componentes de la matriz F en el mismo periodo de tiempo ($2 + \lceil \log_2 N \rceil + N$). Por lo tanto, el esquema de tiempos de la Figura 3.11 es válido también para el cálculo de toda la matriz F.

En la Figura 3.12 se puede apreciar el esquema total de la arquitectura (N planos) que permite el cálculo de los puntos que posteriormente serán mapeados. En dicha figura se observa que el Plano(0) recibe el vector R_N mientras que los otros planos reciben como entrada el mismo vector R_m, m varía sucesivamente entre 0 y N-1 en cada ciclo de reloj.

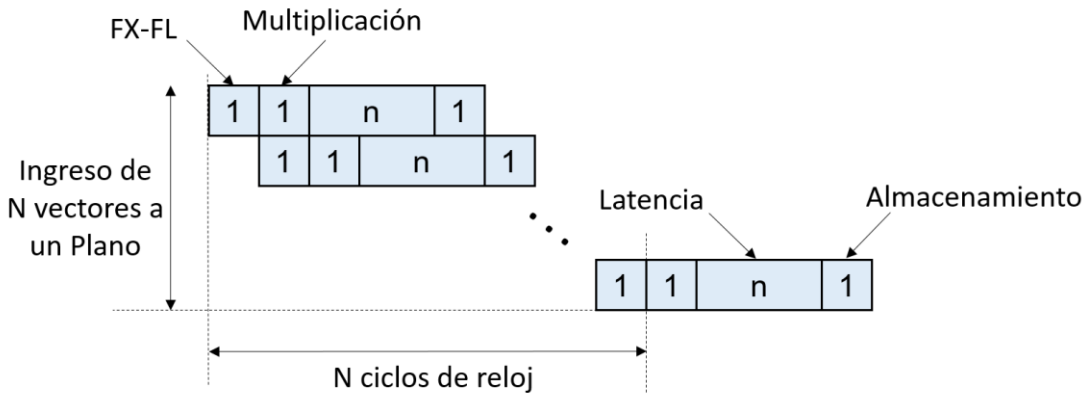


Figura 3.11. Tiempo de ejecución del DFST con la arquitectura propuesta.

$\mathbf{n} = \lceil \log_2 N \rceil$.

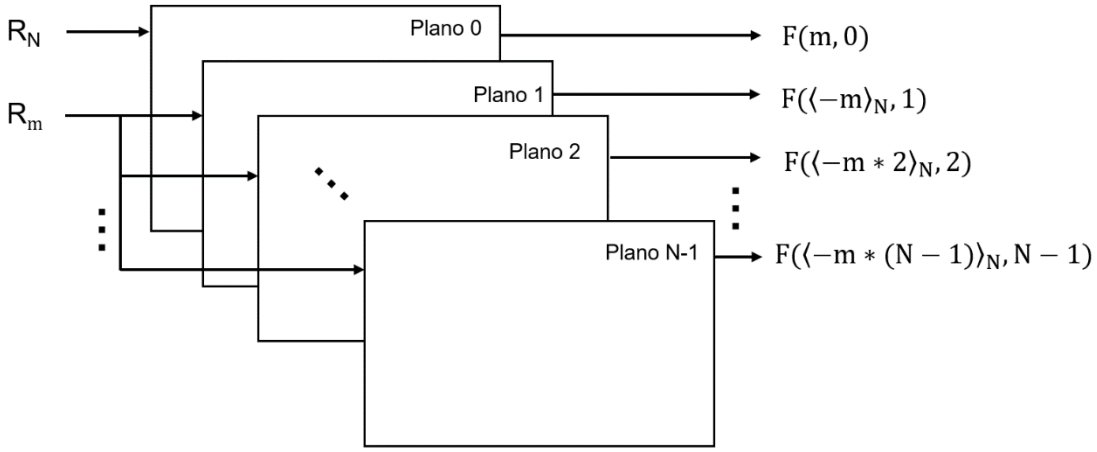


Figura 3.12. Uso de N planos para calcular la matriz de Fourier (F) para una imagen de tamaño N x N donde N es primo. m = 0,1,2, …, N-1.

Como conclusión de esta sección, se presenta un ejemplo de mapeado mediante el uso de planos para una imagen de dimensión $N \times N$ con N =7. En la Figura 3.13 se observa que cada en cada ciclo se mapean N elementos. El mapeado total dura 7 ciclos de reloj (N ciclos de reloj para cualquier N primo). En cuanto al hardware, sí es posible el mapeado propuesto ya que en todos los ciclos de reloj se accede únicamente a una dirección por memoria. Es decir, que cada punto pertenece a una columna distinta en cada ciclo.

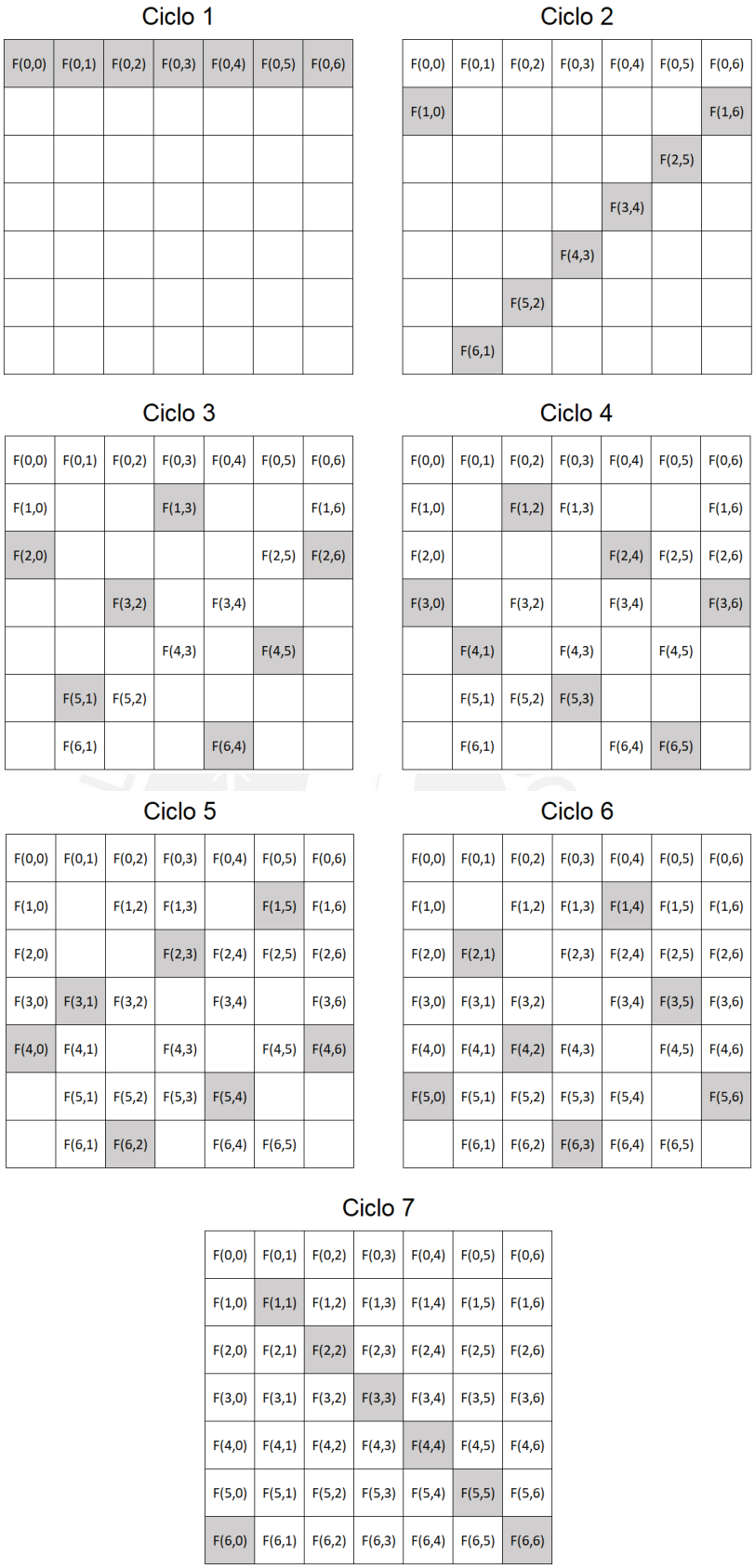


Figura 3.13. Ilustración del mapeado usando planos para N=7

3.4 Diseño del bloque de mapeado en memoria

Como se mencionó en la sección anterior, se considera la memoria como un arreglo de N SRAM. Cada una de estas representa una columna de la matriz de Fourier. En esta sección se presenta el hardware necesario para ubicar o mapear los puntos que fueron calculados por los planos.

Para el caso de SRAM(0), la data proviene del Plano(0). Por cada ciclo llega un número de 64 bits en punto flotante que debe ser ubicado en la primera columna. El primer dato debe ser ubicado en la primera dirección de la memoria, es decir $F(0,0)$. En el siguiente ciclo de reloj, el dato que llega debe ser ubicado en la segunda dirección de memoria que es $F(1,0)$. Así sucesivamente por N ciclos de reloj hasta llenar los N datos de la SRAM(0) (toda la primera columna). Se utiliza un contador cíclico de orden N que debe empezar en 0 ($\lceil \log_2 7 \rceil$ bits) y con incremento=1. El modelo de la arquitectura propuesta para este caso se muestra en la Figura 3.14(a).

En los otros casos, el direccionamiento de la data en las SRAM es diferente. La ubicación de los puntos sigue la fórmula de mapeo:

$$\mathrm{F}(\langle -\mathrm{m}\times\mathrm{v}\rangle_{\mathrm{N}}, \mathrm{v}),$$

donde m hace referencia al ciclo de reloj, por lo que varía entre 0 y N-1. v como ya se sabe, se refiere al índice de la SRAM. Al igual que para la SRAM(0), en cada ciclo llega a la SRAM(v) (v=1, 2, …, N-1) un número de 64 bits en punto flotante que debe ser ubicado en la columna v de F. En el caso de la SRAM(1), el primer dato debe ser ubicado en la primera dirección de la memoria, es decir $F(0,1)$. En el siguiente ciclo de reloj (m=1), siguiendo la fórmula de mapeo, el dato que llega debe ser ubicado en F(N-1,1). Después, para el siguiente ciclo de reloj (m=2), el número debe ser ubicado en F(N-2,1) y así sucesivamente por N ciclos de reloj hasta llegar a F(1,1) y llenar los N datos de la SRAM(1) (toda la segunda columna). Para el caso de N=7, el orden de mapeado de la SRAM(1) es:

$$\mathrm{F}(0,1), \quad \mathrm{F}(6,1), \quad \mathrm{F}(5,1), \quad \mathrm{F}(4,1), \quad \mathrm{F}(3,1), \quad \mathrm{F}(2,1), \quad \mathrm{F}(1,1)$$

Para lograr el mapeo deseado en cada SRAM(v), se utiliza un contador cíclico de orden N que debe empezar en 0 ($\lceil \log_2 7 \rceil$ bits) y con incremento=-v. El modelo de la arquitectura propuesta para este caso (v=1, 2, …, N-1) se muestra en la Figura 3.14(b).

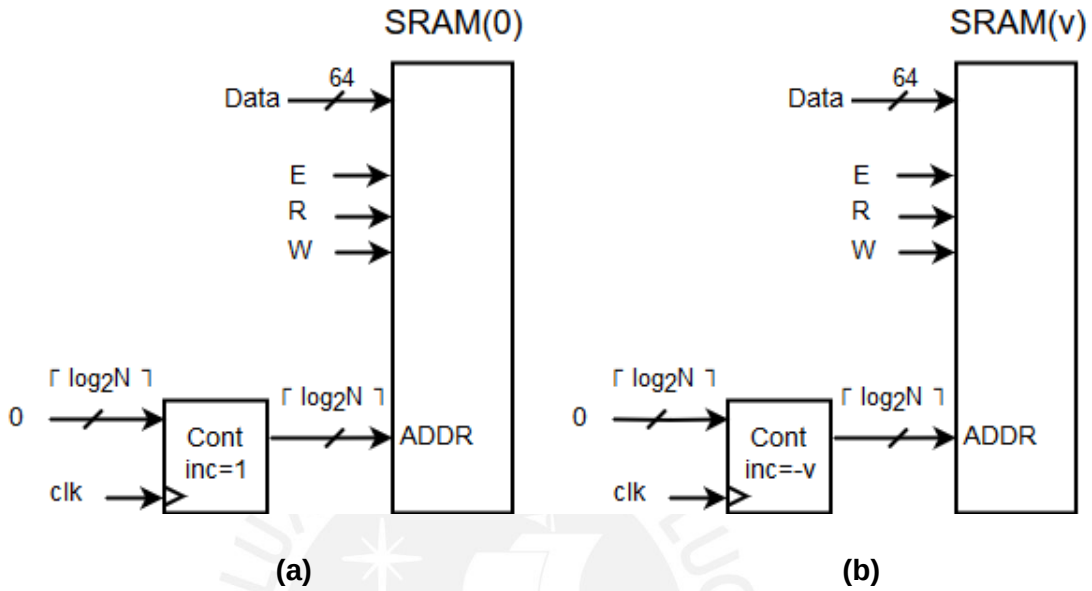


Figura 3.14. (a) Arquitectura para el mapeo de SRAM(0). N: primo. **(b)** Arquitectura para el mapeo de SRAM(v), v=1, 2, …, N-1. N: primo.

3.5 Algoritmo

Para concluir este capítulo, en la Figura 3.15 se presenta el algoritmo propuesto para el cálculo del Teorema de las Rebanadas de Fourier Discreto. Se trata de un algoritmo en paralelo que utiliza la arquitectura descrita en el presente capítulo. Recibe como parámetro de entrada la imagen en el espacio de Radón (matriz R). La matriz de constantes exponenciales K se asume pre calculada. El algoritmo presenta la ejecución paralela de N planos (N primo) que permite la obtención de la matriz F.

Por cada plano, la primera instrucción es la conversión de N puntos de la matriz de Radón de punto fijo a punto flotante correspondientes a una fila (vector R_m). En el caso del Plano(0), en cada ciclo se utiliza la última fila de R, es decir $m = N$, debido a que este plano calcula los puntos de la primera columna de F. Para el resto de planos (Plano(v), v=1, 2, …, N-1), en cada ciclo ingresa un vector diferente R_m, m=0, 1, ..., N-1.

La segunda instrucción es la multiplicación de cada punto del vector de R ya convertido en punto flotante con las contantes exponenciales de una fila de K. En el caso del Plano(0), en cada ciclo los valores del vector R_N se multiplican con las constantes de una de las filas K_m (m=0, 1, ..., $N-1$). El resto de planos (Plano(v), v=1, 2, …, $N-1$) tiene asociado vector K_v que se multiplica con un vector R_m (m=0, 1, ..., $N-1$) diferente en cada ciclo.

La tercera instrucción es la ejecución del árbol sumador cuyo funcionamiento es el mismo para cada plano. La última instrucción es el mapeo o ubicación de puntos en la matriz F. En el caso del Plano(0), los puntos son ubicados en la primera columna. Los puntos calculados por los otros planos (Plano(v), v=1, 2, …, $N-1$) se mapean en las últimas $N-1$ columnas de F.

En términos de ciclos de reloj, se utiliza un ciclo para la conversión y multiplicación respectivamente. El árbol sumador ocupa $\lceil \log_2 N \rceil$ ciclos en obtener el primer resultado y se requiere un ciclo para la ubicación del punto en la matriz F. Se añade $N-1$ ciclos correspondientes a la entrada de los vectores usando pipeline y se obtiene el total de $2 + \lceil \log_2 N \rceil + N$ ciclos de reloj.

```
procedure Fast_DFST(R)
    In_Parallel.Compute Planes v = 0, …, N-1 where
        Plane v = 0
            Convert R_N to floating_point
            Multiply R_N x K_m, m=0, …, N-1
            Compute Adder_Tree
            Map in F(m, 0)
        Plane v = 1, …, N-1
            Convert R_m to floating_point, m=0, …, N-1
            Multiply R_m x K_v, m=0, …, N-1
            Compute Adder_Tree
            Map in F(-m×v Mod N, v)
end procedure
```

Figura 3.15. Algoritmo propuesto para el cálculo rápido del DFST

CAPÍTULO 4: Resultados

En este último capítulo se presenta las gráficas de recursos y de tiempo de ejecución para validar el comportamiento de la arquitectura diseñada. Como se mencionó previamente, el diseño de una arquitectura rápida traducida en un tiempo lineal de ejecución con respecto al tamaño de la imagen, implica el uso de recursos en proporción cuadrática.

4.1 Recursos

El cálculo de recursos se hace en base a las Figuras 3.7, 3.8 y 3.14. En el caso de las Figuras 3.7 y 3.8 se observan componentes como sumadores, multiplicadores, registros y unidades de conversión de punto fijo a punto flotante (FX-FL). De la misma manera, en la Figura 3.14 se observan componentes como SRAM y contadores para el mapeado en memoria. La cantidad de recursos a utilizar en una arquitectura está en función al tamaño de la imagen (N, primo). En la Tabla 2 se muestra cómo se calcula la cantidad de cada tipo de componente a utilizar en el cálculo del DFST para una imagen de $N \times N$ mediante la arquitectura propuesta.

Para un árbol sumador, se define A_{SUM} como la cantidad de sumadores compuestos y A_{REG} como la cantidad de registros. A_{SUM} y A_{REG} crecen linealmente con respecto a N y se pueden calcular usando el algoritmo mostrado en la Figura 4.1.

Según las Figuras 3.7 y 3.8, los sumadores compuestos reciben dos números en punto flotante de 64 bits. Uno de ellos es un punto de la matriz de Radón y el otro una constante exponencial. Sin embargo, los sumadores compuestos son la representación de dos sumadores en punto flotante de 32 bits que trabajan en paralelo. Por lo tanto, la cantidad de sumadores aumenta al doble, es decir $2 \times A_{SUM}$. Asimismo, ya que esta cantidad corresponde a la cantidad de sumadores por un plano, se debe multiplicar por el total de planos que se utilizan en la arquitectura. Es decir, la cantidad total de sumadores resulta $2 \times N \times A_{SUM}$, para N primo.

En el caso de los registros utilizados, en la Figura 3.7 se muestra que a A_{REG} se le debe añadir N registros correspondientes al almacenamiento de las

constantes exponenciales. Por lo tanto, la cantidad de registros por plano es $A_{REG} + N$. Al igual que para los sumadores, se debe multiplicar por N (primo) a esta cantidad para así obtener el total de registros utilizados en la arquitectura, el resultado es $N \times (A_{REG} + N)$.

Para el cálculo de la cantidad de multiplicadores, en primer lugar, se debe definir que son de 32 bits. Además, el bloque multiplicador en las Figuras 3.7 y 3.8 en realidad representa dos multiplicadores que operan en paralelo. Se sabe que se tiene que multiplicar un punto de la matriz de Radón (convertido a punto flotante) y una constante exponencial. Ambos factores cuentan con una parte real y una imaginaria. Teóricamente se necesitan cuatro multiplicadores para ejecutar la multiplicación compleja (M):

$$\begin{aligned} M &= \left(R(m,d)_{real} + jR(m,d)_{imag}\right) \times \left(K(x,y)_{real} + jK(x,y)_{imag}\right) \\ &= R(m,d)_{real} \times K(x,y)_{real} + jR(m,d)_{real} \times K(x,y)_{imag} \\ &\quad + jR(m,d)_{imag} \times K(x,y)_{real} - R(m,d)_{imag} \times K(x,y)_{imag} \end{aligned}$$

Sin embargo, la parte imaginaria del punto de la matriz de Radón es igual a 0, es decir: $R(m,d)_{imag} = 0$. Entonces, la multiplicación resultaría de esta manera:

$$M = R(m,d)_{real} \times K(x,y)_{real} + jR(m,d)_{real} \times K(x,y)_{imag}$$

De esta forma, al utilizar dos multiplicadores que realicen ambas operaciones en paralelo, se obtiene M en un ciclo de reloj. El total de multiplicadores que se utiliza en un plano es $2 \times N$ y para toda la arquitectura es $2 \times N \times N$.

Las unidades de conversión de punto fijo a punto flotante utilizadas en un plano ascienden a N (directamente proporcional al tamaño de la imagen). Como se usan N planos, el total es $N \times N$.

En cuanto a los recursos utilizados para el mapeo de los puntos de Fourier, se utilizan SRAM y contadores. La cantidad de SRAM es el total de columnas de la matriz de Fourier, es decir N. De la misma manera, se utiliza un contador por cada SRAM, por lo tanto, son necesarios N de estos componentes.

Tabla 2: Uso de recursos para diferentes implementaciones del DFST utilizando la arquitectura propuesta

Recursos	Cantidad	Nota
Sumadores	$2 \times N \times A_{SUM}$	Punto flotante, 32 bits
Registros	$N \times (A_{REG} + N)$	Punto flotante, 64 bits
Multiplicadores	$2 \times N \times N$	Punto flotante, 32 bits
Unidades FX-FL	NxN	Ejecuta la conversión en 1 ciclo de reloj
SRAM	N	Capacidad N, 64 bits de bus de datos
Contadores	N	Cíclicos (orden N)

En la Tabla 3, se presenta de manera informativa la cantidad de recursos a utilizar para algunos tamaños relevantes de imágenes. Asimismo, se presentan las gráficas del total de componentes requeridos para las implementaciones en función del tamaño N de la imagen. El rango de tamaño incluye todos los números primos menores o iguales que 1021 (Figura 4.2 a la 4.7).

```
procedure Recursos_Arbol(N)
    Areg=Asum= 0
    h =⌈ log2 N⌉
    a = N
    for z = 1 to h do
        x = a Mod 2
        a = ⌊a/2⌋
        Asum = Asum + a
        Areg = Areg + a × 2 + x
        a = a + x
    end for
    return Areg, Asum
end procedure
```

Figura 4.1. Algoritmo para el cálculo de la cantidad de sumadores compuestos (A_{REG}) y registros (A_{SUM}) en un árbol sumador

Tabla 3: Uso de recursos para implementaciones de la arquitectura propuesta para imágenes de tamaños relevantes

N	Total sumadores	Total registros	Total multiplicadores	Total unidades FX-FL	Total SRAM	Contadores
7	84	140	98	49	7	7
61	7,320	11,163	7,442	3,721	61	61
127	32,004	48,260	32,258	16,129	127	127
257	131,584	199,689	132,098	66,049	257	257
509	517,144	777,243	518,162	259,081	509	509
1021	2,082,840	3,127,323	2,084,882	1,042,441	1,021	1021

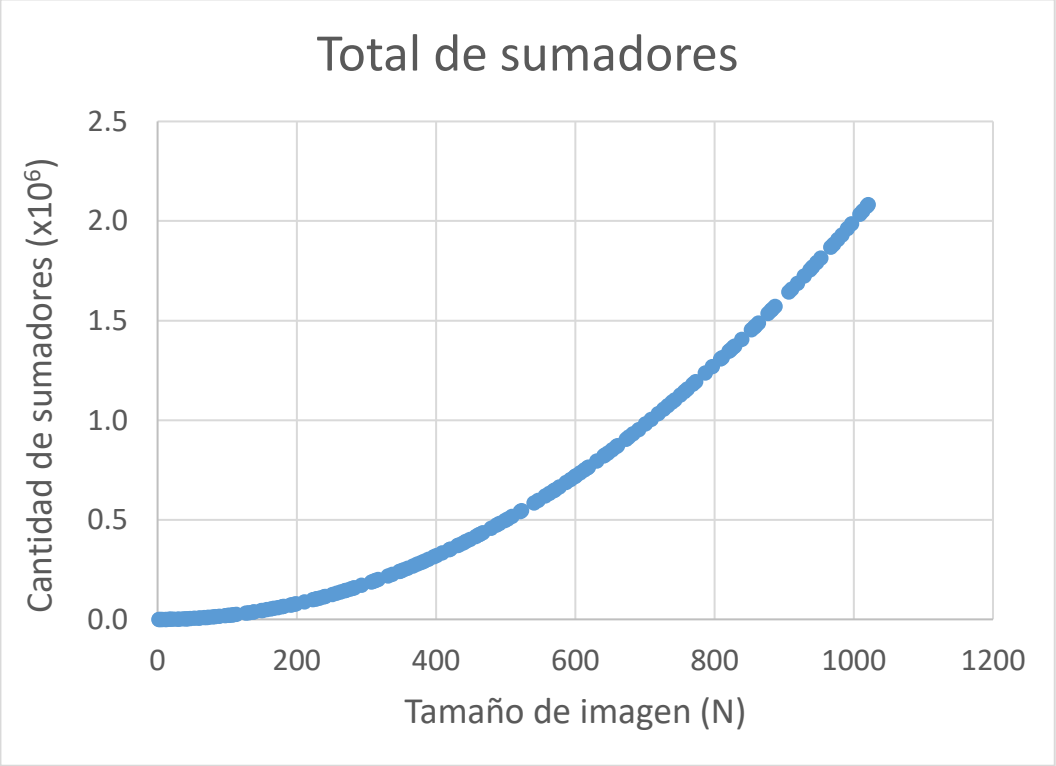


Figura 4.2. Gráfica del total de sumadores requeridos para implementación de la arquitectura en función del tamaño de la imagen (N, primo)

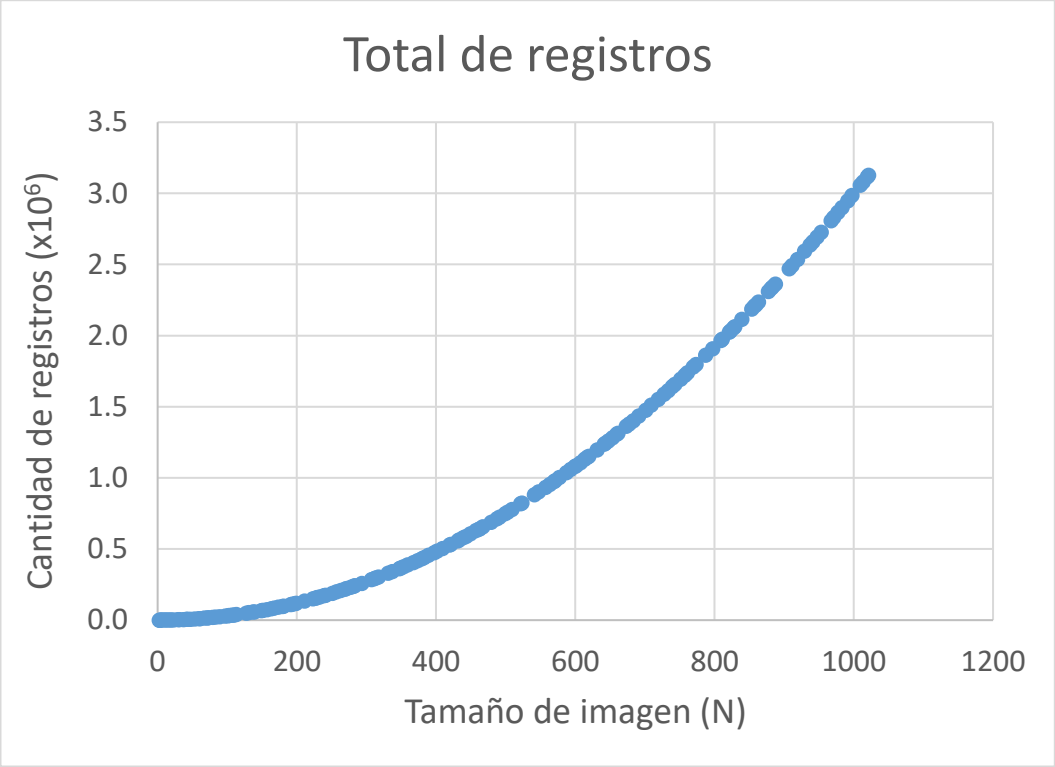


Figura 4.3. Gráfica del total de registros requeridos para implementación de la arquitectura en función del tamaño de la imagen (N, primo)

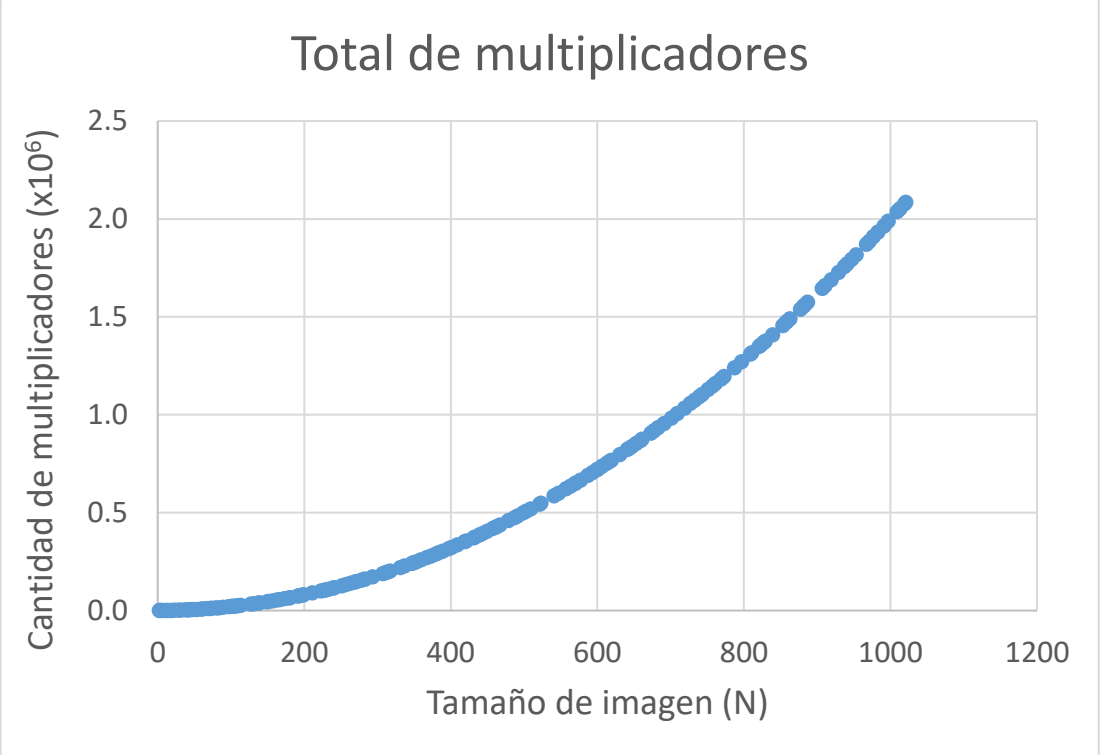


Figura 4.4. Gráfica del total de multiplicadores requeridos para implementación de la arquitectura en función del tamaño de la imagen (N, primo)

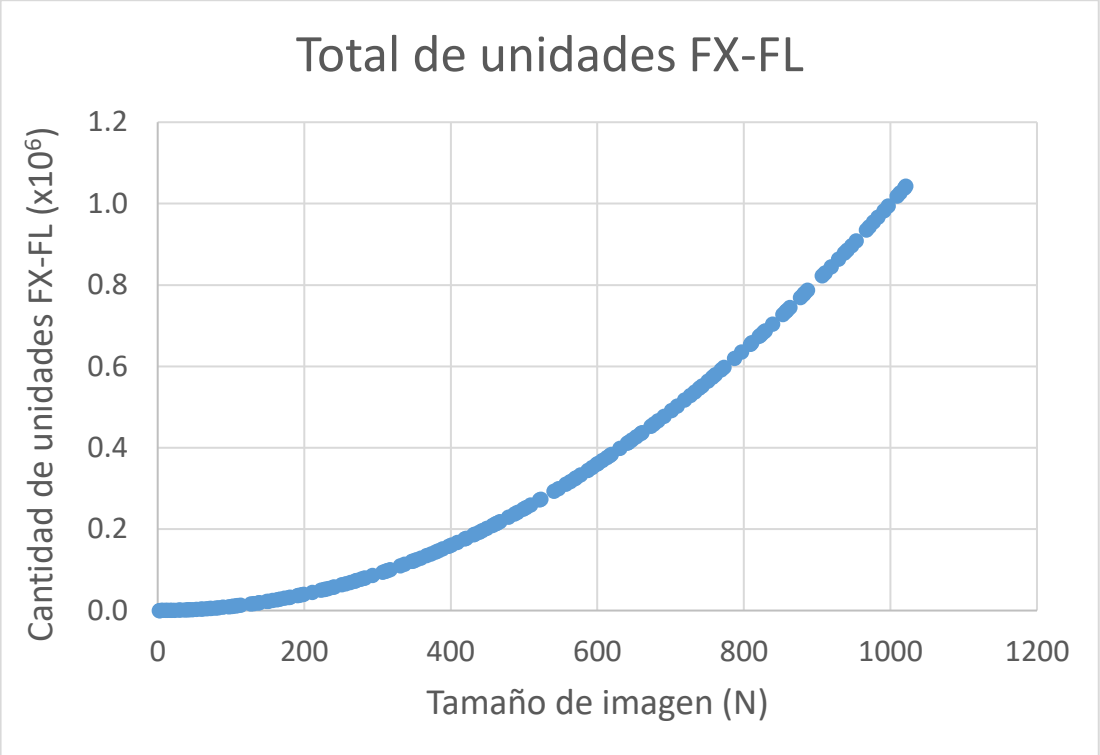


Figura 4.5. Gráfica del total de unidades FX-FL requeridas para implementación de la arquitectura en función del tamaño de la imagen (N, primo)

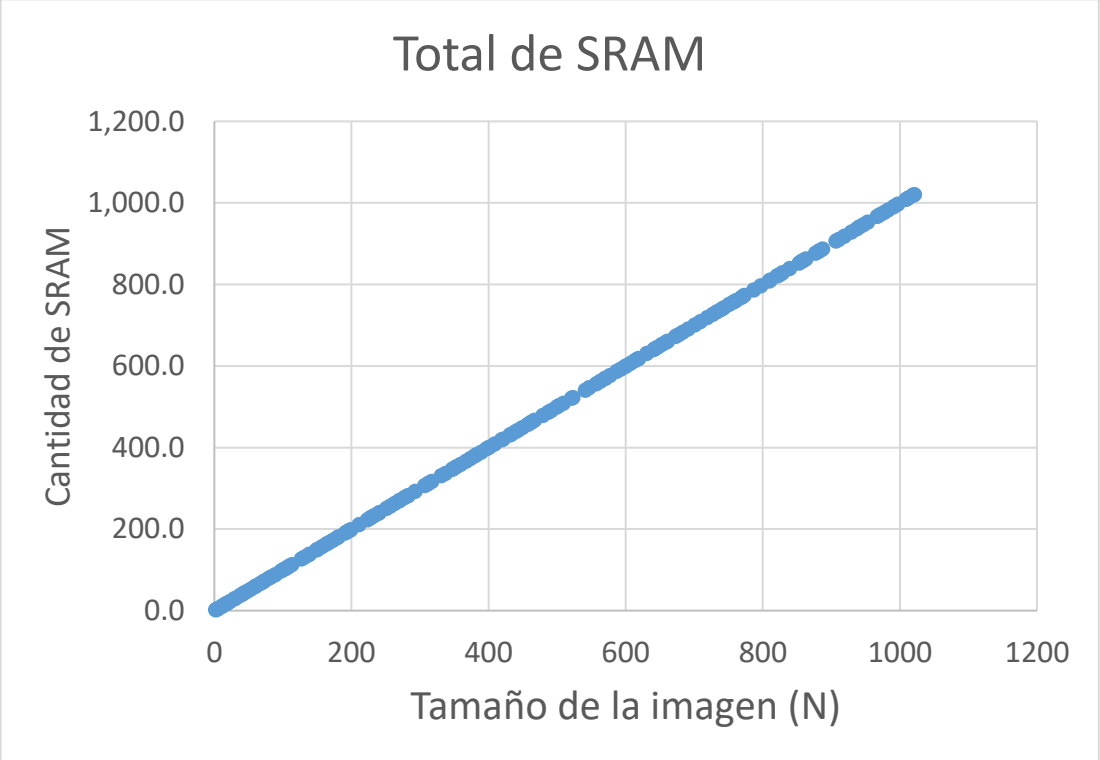


Figura 4.6. Gráfica del total de SRAM requeridas para implementación de la arquitectura en función del tamaño de la imagen (N, primo)

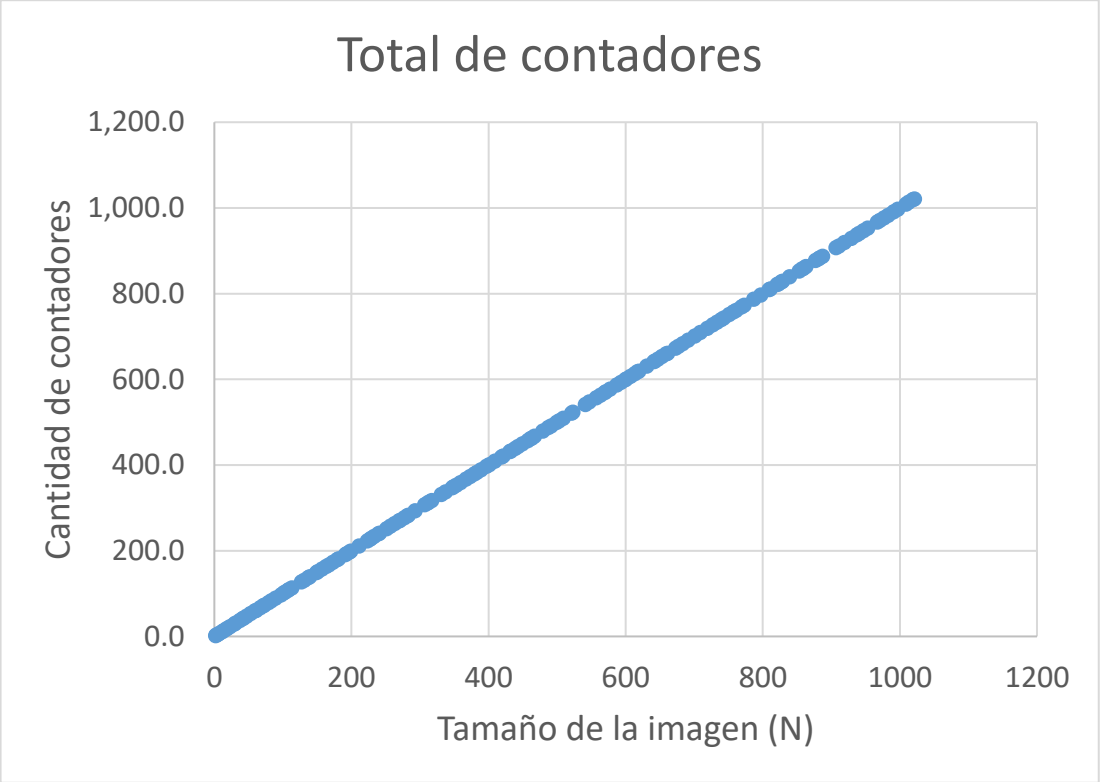


Figura 4.7. Gráfica del total de contadores requeridos para implementación de la arquitectura en función del tamaño de la imagen (N, primo)

4.2 Tiempo de ejecución

Como se mencionó en el subcapítulo 3.3, el tiempo de ejecución del DFST utilizando la arquitectura propuesta es de $2 + \lceil \log_2 N \rceil + N$ ciclos de reloj. De esta manera, si se añade el tiempo que tarda la ejecución de la FDPRT [1] ($2N + \lceil \log_2 N \rceil + 1$ ciclos de reloj), el total de ciclos de reloj necesarios para el cálculo de la DFT-2D es de $3N + 2\lceil \log_2 N \rceil + 3$ con el método propuesto.

En la Figura 4.8 se presenta la gráfica de la cantidad de ciclos de reloj necesarios para la ejecución rápida del DFST para distintas implementaciones. Se utiliza el mismo rango de valores que en las gráficas de recursos de la sección anterior.

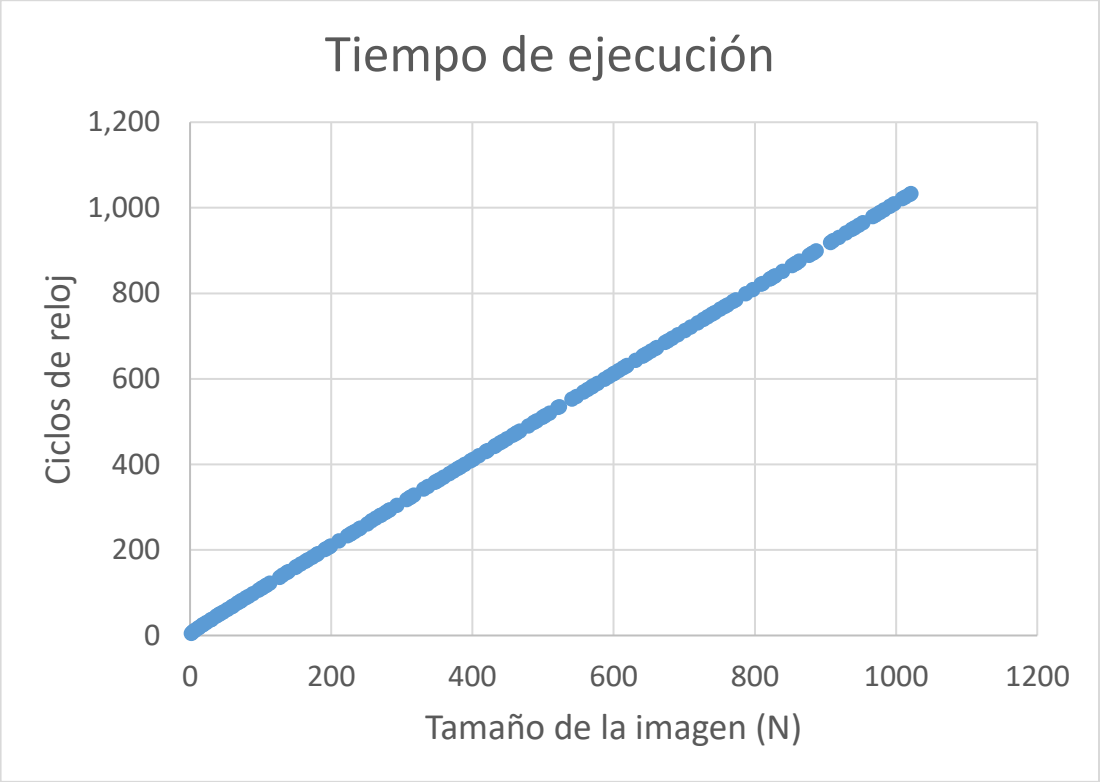


Figura 4.8. Gráfica del tiempo de ejecución, en términos de ciclos de reloj, del DFST utilizando la arquitectura propuesta en función del tamaño de la imagen (N, primo)

4.3 Validación del algoritmo en Matlab

Para comprobar el correcto funcionamiento del algoritmo se implementa la función F_DFST en el software Matlab R2013a. Esta función recibe como parámetro de entrada la matriz en el espacio de Radón y sigue las instrucciones del algoritmo propuesto en el capítulo anterior (Figura 3.15). Sin embargo, al tratarse de una simulación en serie, la ejecución de los N planos se realiza secuencialmente.

En primer lugar, se calcula la matriz cuadrada K de constantes exponenciales. Según el algoritmo, en cada plano se multiplica un vector de la matriz de Radón con un vector de la matriz K. Se realizan las multiplicaciones correspondientes y los resultados son almacenados en un arreglo llamado 'registro' de dimensión Nx1. En este, los datos se agrupan de 2 en 2 y se suman simulando un árbol sumador. Los resultados de estas sumas son ubicados al inicio del arreglo. De esta manera, luego de $\lceil \log_2 N \rceil$ etapas, el resultado se encuentra en la primera dirección del arreglo.

Seguidamente del cálculo de cada punto, se procede a mapearlo. El Plano(0) se encarga de llenar la primera columna de F. De la misma manera, cada Plano(v) se encarga de mapear los puntos de la columna v de la matriz F, donde v=0, 2, …, N-1.

Se busca validar el funcionamiento en base a la exactitud del cálculo de la DFT-2D mediante el método propuesto. Para lograrlo, se recupera la imagen desde el dominio de Fourier con la transformada inversa de Fourier de Matlab y se compara mediante la métrica del error cuadrático medio (MSE, por sus siglas en inglés) con la imagen original. De igual modo, se evalúa la exactitud de la DFT-2D de Matlab (FFT).

Para la validación final, se calcula la DFT-2D de una matriz I mediante dos métodos:

- F: calculada mediante la transformada de radón y seguidamente el teorema de las rebanadas de Fourier
- F_fft: calculada mediante la FFT de Matlab

Ambas matrices en el espacio de Fourier se recuperan mediante la función de transformada inversa de Fourier de Matlab (ifft2). Así se obtiene I_rec_FDST e I_rec_FFT respectivamente.

Para la prueba, se utilizan los tamaños de imagen de la Tabla 3. Se crean matrices con números enteros aleatorios entre 0 y 255 que representan una imagen con profundidad de 8 bits. Los resultados se muestran en la Tabla 4.

Tabla 4: Resultados de la prueba en Matlab

N	MSE (FDFST)	MSE (fft2)
7	3.360490e-10	1.385819e-10
61	4.638190e-09	7.628985e-10
127	9.923691e-09	2.496280e-09
257	2.542723e-08	1.390962e-09
509	4.703201e-08	4.398861e-09
1021	9.908282e-08	3.310274e-09

Con los resultados expuestos en la Tabla 4 se afirma que la DFT-2D fue calculada exitosamente mediante la simulación del algoritmo propuesto. La obtención de valores de MSE diferentes de 0 indican que la DFT-2D no es totalmente ideal y que las diferencias entre las imágenes recuperadas y la imagen original son producto de las aproximaciones propias del software y el uso de datos de presición simple (single). Asimismo, se observa que el cálculo mediante el método propuesto tiene un mse menor a 10e-7 en punto flotante, 32 bits.

4.4 Limitaciones con el hardware actual (2019)

En el presente trabajo se ha diseñado una arquitectura que es independiente de la tecnología existente. Sin embargo, en caso de querer proceder con la implementación de la misma (por ejemplo, en un FPGA), se debe tener en cuenta que la alta cantidad de recursos necesarios limita el tamaño de imagen que se puede procesar. En los FPGAs modernos los métodos basados en FFT están en gran medida limitados por el número de multiplicadores (DSP) disponibles [19]. Por lo tanto, se considera como el factor limitante la cantidad de multiplicadores.

Tabla 5. Hardware disponible según modelo FPGA (serie Intel Agilex F) [20].

Intel Agilex F-Series Device Names	Logic Elements (LE)	M20K Blocks	M20K Mbits	MLAB Counts	MLAB Mbits	Variable Precision DSP Blocks	18x19 Multipliers
AGF 004	392,000	1,900	38	6,644	4.3	1,150	2.3K
AGF 006	573,480	2,844	56	9,720	6.2	1,640	3.3K
AGF 008	764,640	3,792	74	12,960	8.3	2,296	4.6K
AGF 012	1,200,000	5,568	110	20,338	13	4,000	8K
AGF 014	1,437,240	7,110	139	24,360	15.6	4,510	9K
AGF 022	2,200,000	11,616	210	37,288	21	6,250	12.5K
AGF 027	2,692,760	13,272	259	45,640	29.2	8,736	17K

Se utiliza como referencia, la hoja de resumen de los dispositivos de la serie Agilex F de Intel [20]. En el mencionado documento se muestra la tabla de la Figura 4.9 en la que se muestra la cantidad de multiplicadores según el modelo. Estos multiplicadores (expresados como bloques DSP) son compatibles con la arquitectura propuesta en esta tesis ya cuentan con capacidad de punto flotante que incluye precisión simple (32 bits). Con los datos de la Figura 4.9 y el cálculo de recursos de la Tabla 2 se estima que el tamaño de la imagen que se puede procesar con el hardware disponible (máximo 8736 multiplicadores) en la actualidad (2019) es de 61×61.

4.5 Comparación

En esta sección se realiza una comparación teórica en cuanto a la velocidad de ejecución de la DFT-2D usando el método propuesto y dos librerías de FFT: FFTW y MKL de Intel [21]. Se utiliza los tamaños de imagen de la Tabla 3 en los 3 casos. Para ambas librerías se compila y ejecuta la DFT-2D con Visual Studio 2019 en una computadora con las siguientes características: Intel Core i7-4510U CPU@2.6Ghz, memoria 8GB RAM, SO Windows 10 Pro 64-bit.

Para cada tamaño de imagen se ejecuta un total de 21 iteraciones de la DFT-2D. El tiempo de la primera no se considera en la estadística puesto que corresponde al periodo de “warm up” del código. De siguientes 20 iteraciones se obtiene un mínimo y un promedio de tiempo de ejecución. Los resultados se observan en la Tabla 5. Los datos de la mencionada tabla solo contemplan los tiempos de ejecución, mas no los tiempos de inicialización, planeamiento o set up.

Tabla 6: Tiempo de ejecución mínimo y promedio de la DFT-2D (20 iteraciones) usando librerías FFTW y MKL (Intel) compilado en Visual Studio 2019

N	FFTW		MKL	
	mínimo (ms)	promedio (ms)	mínimo (ms)	promedio (ms)
7	0.00335	0.003412	0.00053	0.000608
61	0.31135	0.433205	0.05084	0.070023
127	0.79640	1.030129	0.19423	0.204580
257	4.23796	4.622430	2.48024	1.447075
509	19.14445	20.329350	3.52681	3.878300
1021	83.13333	90.693162	17.30012	18.128838

Por otra parte, si bien el método para calcular la DFT-2D desarrollado en esta tesis (DPRT y DFST) no se implementó, se deben tener determinadas consideraciones para realizar una comparación teórica justa. Establecidos los tamaños de imagen, en primer lugar, se asume que el dispositivo en el que se ejecuta la DFT cuenta con un máximo de 8736 multiplicadores (como se indica en la sección 4.4). En segundo lugar, se usan los tiempos de ejecución de la SFDPRT [1] con el parámetro H=2 y el FDFST calculado en el presente trabajo. Como última consideración se asume una frecuencia de 100 MHz.

Sin embargo, para algunos tamaños de imagen se excede el número de multiplicadores disponibles. Por ejemplo, para $N = 509$ son necesarios 518162 multiplicadores, cada uno de los 509 planos requiere 1018 multiplicadores. Con la cantidad de multiplicadores disponibles solo es posible implementar 8 planos. Esto indica que proceso no se puede realizar completamente en paralelo. Se debe realizar en serie utilizando solo 8 planos a la vez, lo cual aumenta 64 veces el tiempo de ejecución ($\lceil 509/8 \rceil$). A esta constante se le llama Factor FDSF en la Tabla 6, en la cual se muestra el tiempo de ejecución teórico de la DFT-2D para los 6 tamaños de imagen.

Tabla 7: Tiempo de ejecución de la DFT-2D usando el método propuesto

N	Ciclos de reloj SFDPRT H=2 [1]	Ciclos de reloj FDFST	Factor FDSFT	Total	Tiempo ejecución (ms) f=100 MHz
7	73	73	1	146	0.00146
61	2233	69	1	2302	0.02302
127	8833	136	4	9377	0.09377
257	34573	268	17	39129	0.39129
509	132601	520	64	165881	1.65881
1021	527353	1033	256	791801	7.91801

El resultado de la comparación de tiempo de ejecución se muestra en la Figura 4.10. En los casos de la FFTW y la MKL se utiliza el tiempo mínimo de ejecución de la DFT-2D para cada tamaño de imagen (Tabla 5). Para el primer tamaño ($N = 7$), se observa que en todos los casos el tiempo de ejecución es sumamente corto (por debajo de 0.01 ms). Sin embargo, para los demás tamaños de imagen, se aprecia que la FFTW tiene un mayor tiempo de ejecución que la MKL. Asimismo, el uso de arquitecturas rápidas de DPRT y DFST es el método que menos tiempo emplea. La escala logarítmica empleada en la Figura 4.10 permite observar que la diferencia entre el método propuesto en esta tesis y la FFTW es de aproximadamente un orden de magnitud.

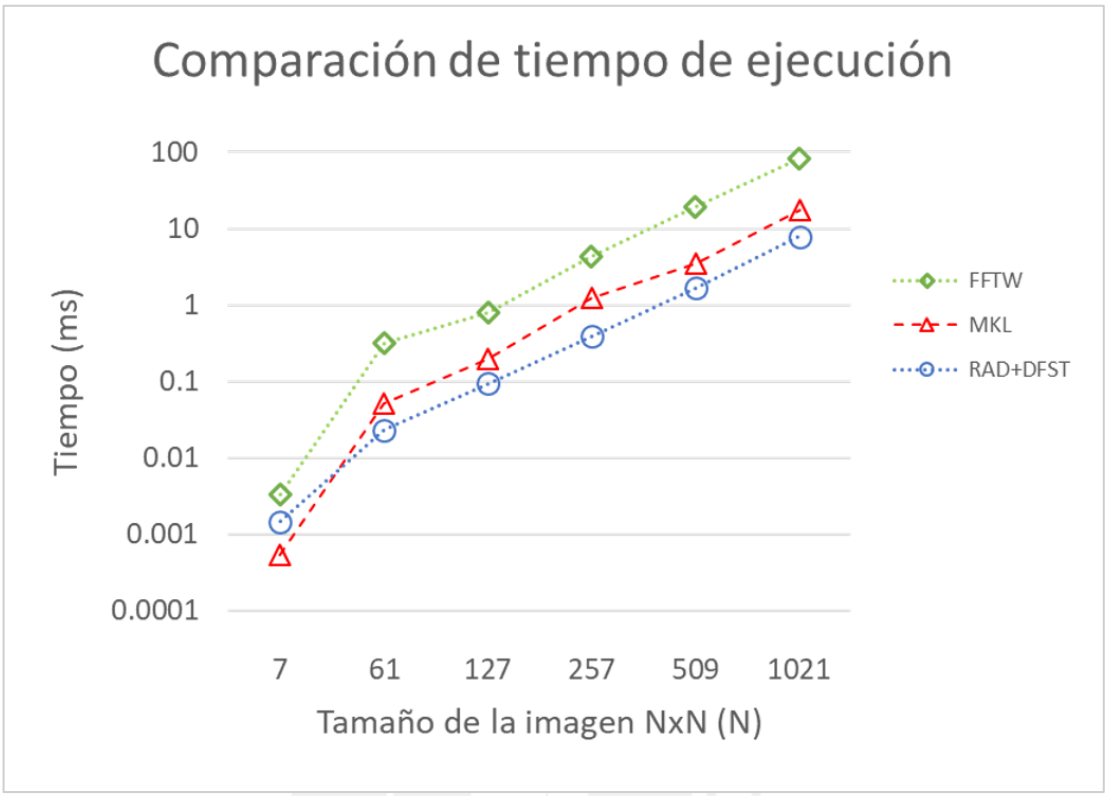


Figura 4.9. Comparación de tiempo de ejecución de la DFT-2D

Por otra parte, se realiza la comparación de velocidad para los 3 casos. La unidad medida es MFLOPS y se define como la transformación del tamaño $N \times N$ a $(5N^2 \log_2 N)/t$, donde t es el tiempo en µs. Esta transformación está basada en el número asintótico de operaciones $(5N \log_2 N)$ para el algoritmo Radix-2 Cooley-Tukey [14], la cual es válida únicamente para la DFT en 1 dimensión. Por lo tanto, se ajusta a $10N^2 \log_2 N$ para la DFT-2D. No obstante, al tratarse de datos de entrada reales, se reduce esta cantidad a la mitad, es decir, $5N^2 \log_2 N$.

Los resultados se muestran en la Figura 4.11. Para el primer tamaño ($N = 7$), la MKL tiene una mayor velocidad que los otros dos métodos. Sin embargo, para los demás tamaños, el método propuesto (DPRT y DFST) supera a la MKL y la FFTW. Por ejemplo, para $N = 1021$, es 2.18 veces más rápido que la MKL y 10.5 veces más rápido que la FFTW. Asimismo, se aprecia que la MKL es más rápida que la FFTW en el cálculo de la DFT-2D para imágenes de dimensión prima.

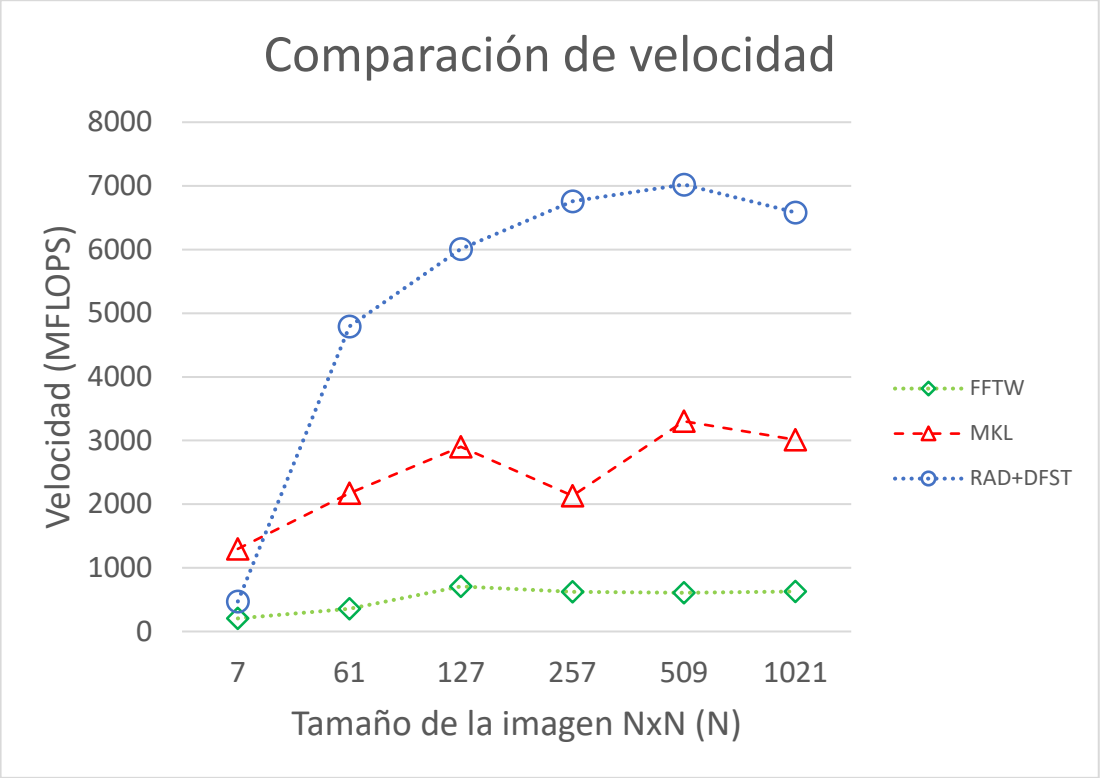


Figura 4.10. Comparación de velocidad de ejecución de la DFT-2D

Las pruebas realizadas en esta sección, tanto teóricas como prácticas, permiten realizar una comparación teórica del método diseñado en la presente tesis. Si bien no se cuenta con una implementación, las consideraciones tomadas permiten tener una noción del rendimiento de la arquitectura diseñada en cuanto a velocidad y tiempo de ejecución en comparación con tecnologías actuales.

Conclusiones

- El presente trabajo contribuye al estado del arte con una forma de calcular rápidamente la DFT-2D en imágenes de dimensión prima. Se ha presentado la base teórica que, independientemente de la tecnología existente, complementa el FDPRT realizado en [3] con el cálculo rápido del Teorema de las Rebanadas de Fourier Discreto.
- Se ha conseguido diseñar una arquitectura rápida que permite calcular el Teorema de las Rebanadas de Fourier Discreto a partir del espacio de Radón. El uso de hardware en paralelo (pipeline) ubican la presente tesis en el ámbito del HPC (High Performance Computing).
- Se logró diseñar el algoritmo que describe el uso en paralelo de N planos que permiten el cálculo de los puntos de la matriz del espacio de Fourier a partir de la matriz en el espacio de Radón y su posterior mapeado.
- Se cuantificó el tiempo de ejecución para el cálculo del DFST en términos de ciclos de reloj y el resultado fue de $2 + \lceil \log_2 N \rceil + N$, asintóticamente O($N$) para implementaciones destinadas a imágenes de tamaño $N \times N$ (N: primo).
- Asimismo, se cuantificó el total de recursos necesarios para estas imágenes. Las gráficas permitieron comprobar que mientras que el tiempo de ejecución tiene un comportamiento lineal, los recursos crecen de forma cuadrática.

Recomendaciones

- Debido a que el presente trabajo es una continuación de la FDPRT [3] para el cálculo de la DFT-2D, el proceso se ejecuta cuando la matriz de Radón ya está calculada. Esto se debe a que la FDPRT primero calcula la primera fila de la matriz R y así sucesivamente hasta llegar a la última. La posibilidad de modificar la máquina de estados que controla la arquitectura de la FDPRT para que calcule primero la última fila de la matriz R, permitiría que estos datos ya puedan ser utilizados en el cálculo del DFST y de esta manera reducir aún más tiempo de ejecución total.
- En el presente trabajo se ha considerado que los componentes de los planos encargados de la conversión de punto fijo a flotante, la multiplicación y la suma requieren un ciclo de reloj. Se sugiere modelar las latencias de estos componentes de la siguiente manera: $CONV_{LAT} + MUL_{LAT} + SUM_{LAT} + \lceil \log_2 N \rceil + N - 1$, ($N$, primo), tomando en cuenta la tecnología actual.
- Al considerar que se busca aplicar la DFT-2D a una imagen en la cual todos los datos son reales, se podría aplicar la propiedad de simetría conjugada-par [21]. Por ejemplo, en la librería Math Kernel de Intel para el cálculo de la DFT-2D, se utiliza esta propiedad para almacenar únicamente mitad de todo el resultado del cálculo. Se recomienda ahondar de mayor manera en este tema con el objetivo de aplicar el concepto en la arquitectura. De esta manera se podría reducir el número de operaciones y recursos.
- La aplicación de escalabilidad a la arquitectura propuesta sería una alternativa muy factible ante el considerable uso de recursos. Se recomienda, para la aplicación de un trabajo futuro, el diseño e implementación de arquitectura escalable como la mencionada en [1] para el cálculo del DFST.

Bibliografía

[1] C. Carranza, D. Llamocca, and M. Pattichis, “Fast and scalable computation of the forward and inverse discrete periodic radon transform,” *IEEE Trans. Image Process.*, vol. 25, no. 1, pp. 119–133, 2016.

[2] T. Hsung, D. P. K. Lun, and W. C. Siu, “The discrete periodic radon transform,” *IEEE Trans. Signal Process.*, vol. 44, no. 10, pp. 2651–2657, 1996.

[3] C. Carranza, D. Llamocca, and M. Pattichis, “The Fast Discrete Periodic Radon Transform for Prime Sized Images: Algorithm, Architecture, and VLSII/FPGA Implementation,” vol. 2, pp. 169–172, 2014.

[4] R. C. Gonzalez and R. E. Woods, “Digital image processing,” *Nueva Jersey*. pp. 150–151, 2008.

[5] U. of Auckland, “Spatial Frequency Domain,” 2014. [Online]. Available: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic1.htm#spatial.

[6] C. M. Rader, “Discrete Fourier Transforms When the Number of Data Samples Is Prime,” *Proc. IEEE*, vol. 56, no. 6, pp. 1107–1108, 1968.

[7] L. Bluestein, “A linear filtering approach to the computation of discrete Fourier transform,” *IEEE Trans. Audio Electroacoust.*, vol. 18, no. 4, pp. 451–455, 1970.

[8] S. R. Deans, “The Radon Transforn and some of its Applications.” p. 294, 1983.

[9] A. M. Grigoryan, “Comments on ‘the discrete periodic radon transform,’” *IEEE Trans. Signal Process.*, vol. 58, no. 11, pp. 5962–5963, 2010.

[10] F. Matúš and J. Flusser, “Image Representations via a Finite Radon Transform,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 10, pp. 996–1006, 1993.

[11] A. A. S. Chandrasekaran, “High speed/low power architectures for the finite radon transform,” in *Proc. Int. Conf. Field Program. Logic Appl.*, 2005, pp. 450–455.

[12] S. M. and A. B. S. Chandrasekaran, A. Amira, “An efficient VLSI architecture and FPGA implementation of the finite ridgelet transform,” *Real-Time Image Process*, vol. 3, no. 3, pp. 183–193, 2008.

[13] M. Frigo and S. G. Johnson, “The Fastest Fourier Transform in the West,” 1997.

[14] M. Frigo and S. G. Johnson, “The Design and Implementation of FFTW3,” *Proc. IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[15] M. Frigo and S. G. Johnson, “Manual for FFTW (version 3.3.8, 24 May 2018),” 2018.

[16] J. W. Cooley and J. W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series,” vol. 19, no. 90, pp. 297–301, 1965.

[17] R. N. Bracewell, *The Hartley Transform*. New York: Oxford Univ. Press, 1986.

[18] Intel, “Intel ® Math Kernel Library,” 2019. [Online]. Available: https://software.intel.com/en-us/mkl.

[19] C. Carranza, D. Llamocca, and M. Pattichis, “Fast 2D Convolutions and Cross-Correlations Using Scalable Architectures,” *IEEE Trans. Image Process.*, vol. 26, no. 5, pp. 2230–2245, 2017.

[20] Intel, “Intel ® Agilex ™ FPGA Advanced Information Brief,” 2019.

[21] Intel, “Intel ® Math Kernel Library: Developer Reference,” rev. 023, 2019.

ANEXOS

Códigos en MATLAB

Código para simular el diseño propuesto, cálculo del DFST. Entrada: matriz R en espacio de Radón, Salida: matriz F en espacio de Fourier.

```
function F = F_DFST( R )
    N = size(R,1)-1; %Calcula el tamaño de la imagen
    h=ceil(log2(N));
    F=zeros(N,N);   %Matriz de Fourier que será mapeada
    %El arreglo 'registro' representa los registros del
    %árbol sumador
    registro=zeros(N,1);

    % Matriz de constantes exponenciales

    K=zeros(N,N);
    for u=0:(N-1)
        for v=0:(N-1)
            K(u+1,v+1)=exp(-1i*2*pi*u*v/N);
        end
    end

    % Plano v=0
    %Las multiplicaciones son almacenadas en 'registro'
    %Se simula el árbol sumador correspondiente al Plano(0)
    %para N: primo
    %Se suman los valores de 2 en 2 y el resultado se ubica
    %en la parte superior de 'registro'
    for u=1:N
        for d=1:N
            %Multiplicación de vectores R_N x K_m
            registro(d,1)=R(N+1,d)*K(u,d);
        end
        a=N;
        for y=1:h %Simulación del árbol sumador
            i=1;
            j=1;
            x=ceil(a/2);
            while(i<=x)
                if (mod(a,2)==1)&&(i==x)
                    registro(i,1)=registro(j,1);
                    break;
                end
                registro(i,1)=registro(j,1)+registro(j+1,1);

                i=i+1;
                j=j+2;
            end
            a=ceil(a/2);
        end
       %El mapeo se da en la primera columna de F
        F(u,1)=registro(1,1);
     end
```

```
    %Plano v=1, 2, …, N-1
    %Las multiplicaciones son almacenadas en 'registro'
    %Se simula el árbol sumador correspondiente al Plano(v)
    %para N: primo
    %Se suman los valores de 2 en 2 y el resultado se ubica
    %en la parte superior de 'registro'

    for v=1:N   %v indica el índice en plano
        for m=1:N
            for d=1:N
                %Multiplicación de vectores R_m x K_v
                registro(d,1)=R(m,d)*K(v,d);
            end
            a=N;
            for y=1:h %Simulación del árbol sumador
                i=1;
                j=1;
                x=ceil(a/2);
                while(i<=x)
                    if (mod(a,2)==1)&&(i==x)
                        registro(i,1)=registro(j,1);
                        break;
                    end

registro(i,1)=registro(j,1)+registro(j+1,1);

                    i=i+1;
                    j=j+2;
                end
                a=ceil(a/2);
            end
            %El mapeo se da en la columna v
            F(mod(-(m-1)*(v-1),N)+1,v)=registro(1,1);

        end
    end

    F=F;
end
```

Código para obtener una matriz cuadrada de números aleatorios entre 0 y 255.

```
function r=random(N)
Rand_N=zeros(N,N);
    for i=1:N
        for j=1:N
            Rand_N(i,j)= round(255*rand);
        end
    r=Rand_N;
    end
end
```

Código calcular la transformada de Radón.

```
function r = rad(I)
    N = size(I,1);
   %Cálculo de las N primeras direcciones primas
    for m=1:N
        for d=1:N
            sum=double(0);
            for i=1:N
                sum = sum + I(i,1+mod(d-1+(m-1)*(i-1),N));
            end
            RAD(m,d)=sum;
        end
    end

    %Cálculo de la última dirección prima
    for d=1:N
        sum=double(0);
        for j=1:N
            sum = sum + I(d,j);
         end
         RAD(N+1,d)=sum;
    end
    r=RAD;
end
```

Código calcular el error cuadrático medio (MSE).

```
function ecm = MSE(I1,I2)
    N = size(I1,1);
    M = size(I2,1);
    sum=double(0);
    for i=1:N
        for j=1:M
            sum=sum+(I1(i,j)-I2(i,j))^2;
        end
    end
    ecm=sum/(N*M);
end
```

Código para calcular la exactitud de imagen recuperada con la imagen original usando el error cuadrático medio. Se recupera la imagen desde el dominio de Fourier mediante la iFFT de Matlab. Se comparan los 2 casos: DFT-2D mediante FFT de Matlab (F_fft) y DFT-2D mediante la transformada de Radón y el DFST (F). Se utilizan 6 tamaños de imagen y datos de 32 bits (single).

```
clear all;
clc;
sizes=[7 61 127 257 509 1021]; %Tamaños de imagen elegidos

for i=1:6
    I=random(sizes(i));    %Matriz aleatoria según el tamaño
    I=single(I);
    R=rad(I);

    F=single(F_DFST(R)); % DFT-2D mediante DPRT y DFST
    F_fft=fft2(I); % DFT-2D mediante FFT de Matlab

    %Se recupera la imagen con iFFT de Matlab
    I_rec=ifft2(F);
    I_rec_fft=ifft2(F_fft);

    %Mean Squared Error entre I (original) y las I recuperadas
    mse=MSE(I,I_rec);
    mse_fft=MSE(I,I_rec_fft);

    %Se muestran los resultados
    fprintf('Para N=%d\nMSE(FDFST)=%d
\nMSE(fft2)=%d\n\n',sizes(i),mse,mse_fft);
end
```

Códigos en C++ (Visual Studio 2019)

Código para calcular DFT-2D usando FFTW

```
#include <iostream>
#include <fftw3.h>
#include "mkl.h"
#include "mkl_dfti.h"
#define ITER 20


// FFTW DFT 2D
int main()
{
	double ti, te;
	double min=99999;
	double sum=0;
	double prom;

	int N = 7;
	for (int i = 1;i <= ITER+1;i++) {
		//WARM UP
		double* in = (double*)fftw_malloc(sizeof(double) *
		N * N);
		fftw_complex* out =
(fftw_complex*)fftw_malloc(sizeof(fftw_complex) * N * N);

		ti = dsecnd();
		fftw_plan p = fftw_plan_dft_r2c_2d(N, N, in, out,
FFTW_ESTIMATE);
		te = (dsecnd() - ti);

		if (i == 1) {
			printf("DFT usando libreria FFTW para N=%d\n",
N);
			printf("WARM UP: %.5f ms \n\n", te * 1000);
		}
		else {
			//printf("Prueba %d, Tiempo plan : %.5f
miliseconds \n", i-1,te * 1000);

			ti = dsecnd();
			fftw_execute(p);
			te = (dsecnd() - ti);

			fftw_free(in); fftw_free(out);
			//printf("Tiempo ejec Pueba %d: %.5f
miliseconds \n\n", i,te * 1000);
			printf("%.5f ", te * 1000);
			if (te < min) min = te;
			sum += te;
		}

		fftw_destroy_plan(p);
	}
```

```
        prom = sum / ITER;
        printf("\nT (min)=%.5f miliseconds \n", min * 1000);
        printf("T (prom)=%.5f miliseconds \n", prom * 1000);

        return 0;
}
```

Código para calcular DFT-2D usando MKL

```
// Pontificia Universidad Catolica del Peru - Cesar Carranza,
May 2019
// Only FFT for Rodrigo thesis Sep 17th 2019
#define ITER 20
#include <iostream>
#include <string>
#include <conio.h> // For _getch();
#include <complex.h>
#include "mkl.h"
#include "mkl_dfti.h"


int main()
{
	double timeIni, timeElapsed;
	int N;
	double min = 99999;
	double sum = 0;
	double prom;

	N = 1021;
	for (int i = 1;i <= (ITER + 1);i++) {
		float *rImage;
		rImage  = (float*)mkl_calloc( (long)N *((long)N / 2
+ 1) * 2, sizeof(float), 64);

		// Compute the DFT for Image and kernel
		DFTI_DESCRIPTOR_HANDLE Phandle;
		MKL_LONG stat, size[2];

		//Setup DFT2D, Real input, 32bits
		size[0] = N; // height of the zero padded image
		size[1] = N; // width of the zero padded image
		//printf("size0 %i, size1 %i\n", size[0], size[1]);
		MKL_LONG iStride[] = { 0,(N/2+1)*2,1 }; // Starting
offset=0, colsMemory, 1
		MKL_LONG oStride[] = { 0,N/2+1,1 };
		//printf("Input strides : %i,%i,%i\n", iStride[0],
iStride[1], iStride[2]);
		//printf("Output strides: %i,%i,%i\n", oStride[0],
oStride[1], oStride[2]);

		timeIni = dsecnd();
		stat = DftiCreateDescriptor(&Phandle, DFTI_SINGLE,
DFTI_REAL, 2, size);
		//printf("stat=%i, Create descriptor  \n",stat);
		stat =
DftiSetValue(Phandle,DFTI_CONJUGATE_EVEN_STORAGE,DFTI_COMPLEX_
COMPLEX); // COMPLEX_COMPLEX storage is recommended for Intel
		//printf("stat=%i, DFTI_CONJUGATE_EVEN_STORAGE
\n",stat);
		stat = DftiSetValue(Phandle, DFTI_INPUT_STRIDES,
iStride);
		//printf("stat=%i, DFTI_INPUT_STRIDES  \n", stat);
```

```
            stat = DftiSetValue(Phandle, DFTI_OUTPUT_STRIDES,
oStride);
            //printf("stat=%i, DFTI_OUTPUT_STRIDES  \n", stat);
            stat = DftiCommitDescriptor(Phandle);
            //printf("stat=%i, DftiCommitDescriptor  \n",
stat);
            timeElapsed = (dsecnd() - timeIni);
            //printf(" DFT %ix%i, Setup time %.5f miliseconds
\n", size[0], size[1], (timeElapsed * 1000));


            if (i == 1) {

                  printf("FFT usando libreria MKL para N=%d\n",
N);
                  printf("WARM UP: %.5f ms \n", timeElapsed *
1000);
            }
            else {
                  //printf(" Prueba %d, Setup time %.5f
miliseconds \n", i-1, (timeElapsed * 1000));

                  timeIni = dsecnd();
                  stat = DftiComputeForward(Phandle, rImage);
                  //printf("stat=%i, DftiComputeForward Image
\n", stat);
                  timeElapsed = (dsecnd() - timeIni);

                  stat = DftiFreeDescriptor(&Phandle);
                  //printf(" Prueba %d, FFT time %.5f
miliseconds \n\n", i - 1, (timeElapsed * 1000));
                  printf("%.5f ", (timeElapsed * 1000));

                  if (timeElapsed < min) min = timeElapsed;
                  sum += timeElapsed;
            }

            mkl_free_buffers();
            mkl_free(rImage);

      }
      prom = sum / ITER;
      printf("\nT (min)=%.5f miliseconds \n", min * 1000);
      printf("T (prom)=%.5f miliseconds \n", prom * 1000);

      return 0;
}
```