Implementación Paralela en GPGPU portable del algoritmo WK para enfoque SAR

Autor: **Ing. Javier Nicolás Uranga**

Presentado ante la Universidad Nacional de La Matanza y la Unidad de Formación Superior de la CONAE como parte de los requerimientos para la obtención del grado de:

MAGÍSTER EN DESARROLLOS INFORMÁTICOS DE APLICACIÓN ESPACIAL

UNIVERSIDAD NACIONAL DE LA MATANZA

DIRECTOR:

Dr. Javier Areta
UNRN-CONICET, Bariloche, Argentina

CO-DIRECTOR:
Dra. Mónica Denham
UNRN-CONICET, Bariloche, Argentina

Noviembre, 2018

Jurados

Dra. Laura Frulla
Inv. Principal de la Misión SAOCOM y del SIASGE
CONAE, Buenos Aires, Argentina

Dr. Ing. José Luis Roca
CONAE, Bariloche, Argentina

Mg. Clara Ferrando
UNLaM, Buenos Aires, Argentina

Resumen

Este trabajo propone realizar una implementación del algoritmo WK (Cafforio et al., 1991) para el enfoque de datos de Radar de Apertura Sintética (SAR) utilizando el paradigma de programación paralela en placas gráficas de propósitos generales (GP-GPU). En particular se analizará la implementación en una placa con un procesador embebido NVDIA Tegra K1, de modo que el hardware sea de pequeñas dimensiones, bajo consumo, portátil e independiente de una computadora como el caso de las placas gráficas usuales. El procesamiento de los datos generados por un sistema SAR consiste en una serie de operaciones factibles de ser paralelizadas. Para llevar adelante este trabajo se realiza un estudio del sistema SAR y en particular del algoritmo de enfoque WK. Se ha seleccionado este algoritmo debido a que permite la compensación exacta de factores de distorsión que ocurren en aplicaciones SAR satelitales.

Palabras Clave: Algoritmo WK, Algoritmo Omega-K, Algoritmo Omega Ka, Algoritmo RMA, Range Migration Algorithm, GPU, MATLAB, C, CUDA, MATLAB, Procesamiento de Señales SAR, Computación Paralela.

Abstract

This work proposes an implementation of the WK algorithm (Cafforio et al., 1991) for Synthetic Aperture Radar (SAR) signal processing using the parallel programming paradigm in general purpose graphic cards (GP-GPU). In particular, the implementation on a board with an embedded NVDIA Tegra K1 processor will be analyzed. The main feature of this hardware is the small size, low power consumption, portable and independent of a desktop computer as in the case of the usual graphic cards. The SAR data processing consists in a set of operations that can be parallelized. To accomplish this work, a study of the SAR system and particularly the WK algorithm is carried out. This algorithm has been selected because it allows the exact compensation of distortion factors that occur in satellite SAR applications.

Keywords: WK Algorithm, Omega-K Algorithm, Omega Ka Algorithm, Range Migration Algorithm , GPU, MATLAB, C, CUDA, SAR Signal Processing, Parallel computing.

Agradecimientos

A la Comisión Nacional de Actividades Espaciales (CONAE) y a la Universidad Nacional de la Matanza (UNLaM) por haberme dado la oportunidad de cursar esta maestría.

A los directores de la Unidad de Formación Superior de la CONAE (UFS, Instituto Gulich): Livio Gratton, Leonardo De Ferrariis y a los directores de la Maestría en Desarrollos Informáticos de Aplicación Espacial (MDIAE): Sergio Masuelli y Carlos Barrientos, por su gestión en facilitar los mecanismos institucionales para la exitosa finalización de este trabajo.

A mis directores de tesis Javier Areta y Mónica Denham por todo el valioso tiempo y apoyo que me brindaron para la realización de este trabajo, acompañándome por la Senda del SAR.

A Carlos Barrientos por sus buenos consejos y revisión general de la tesis.

A mis compañeros de maestría, especialmente a Eduardo Sufán por nuestras innumerables charlas técnicas sobre SAR y Ricardo Barbieri, quien colaboró para acceder a hardware necesario para la tesis.

A Marco Alvarez Reyna, por sus didácticas guías en los trabajos de laboratorio.

A Estefanía Nievas Lio, por su ayuda en la revisión de secciones matemáticas.

A mi Familia por su apoyo incondicional.

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Lista de Acrónimos

CONAE	Comisión Nacional de Actividades Espaciales.
CSA	*Chirp Scaling Algorithm*.
CSV	Formato de archivo separado por comas.
CUDA	*Compute Unified Device Architecture*, conjunto de herramientas para desarrollo de software sobre GPU.
DFT	Transformada Discreta de Fourier.
FFT	Transformada Rápida de Fourier directa.
GPC	*Graphics Processing Clusters*.
GP-GPU	Procesador Gráfico de propósito general.
IFFT	Transformada Rápida de Fourier inversa.
IQ	*In-Phase Quadrature*.
IRF	Función de respuesta al impulso.
MAT	Formato de archivo de Matlab.
MPI	Interfaz de pasaje de mensajes.
PLSR	*Peak Side-Lobe Ratio*.
PRF	Frecuencia de muestreo en acimut.
PRT	Tiempo de Repetición de Pulso.
RAW	Datos crudos.

RCM Migración de celdas en rango.

RCMC Corrección de la migración de celdas en rango.

RDA *Range Doppler Algorithm*.

RFM Multiplicación por la función de referencia.

RMA *Range Migration Algorithm*.

SAR Radar de Apertura Sintética

SMX *Streaming Multiprocessor*.

SRC Compresión secundaria en rango.

UML Lenguaje de modelado unificado.

UNLAM Universidad Nacional de La Matanza.

WK Algoritmo de Focalización SAR, Omega-K

Lista de Símbolos

Bs	Ancho de banda del chirp.
C	Velocidad de la luz.
$D(f_\eta)$	Factor de migración en rango.
dB	Decibeles.
η_c	Beam Crossing Time.
η	Tiempo lento en acimut.
f_0	Frecuencia de la portadora.
f_{equi}	Variable de frecuencia en grilla equiespaciada.
f_η	Frecuencia en acimut.
f_τ	Frecuencia en rango.
f'_τ	Variable de mapeo de Stolt en rango.
F_r	Frecuencia de muestreo en rango.
GR_{nc}	Posición del centro de la escena.
H	Altitud.
k_r	Frecuencia del *chirp* en rango.
L_a	Longitud de la antena SAR.
$L0, L1$	Requerimientos de alto nivel conceptual relacionados con los objetivos e hipótesis de la tesis.
$L2A$	Nivel de requerimientos centrados en el usuario.
$L2B$	Nivel de requerimientos centrados en el sistema.

$\boldsymbol{\lambda}$	Longitud de onda del radar.
$\boldsymbol{R_{ca}}$	Rango cercano.
$\boldsymbol{R_{nc}}$	Centro de la escena del rango oblicuo.
$\boldsymbol{R_{ref}}$	Rango de referencia.
$\boldsymbol{R_t}$	Posición del blanco puntual.
$\boldsymbol{S}$	Datos crudos SAR, en dominio del tiempo, 2D.
$\boldsymbol{S1}$	Datos luego de la aplicación de la FFT 2D. Dominio 2D de las frecuencias.
$\boldsymbol{S2}$	Datos luego de la salida de la función RFM. Dominio 2D de las frecuencias.
$\boldsymbol{S3}$	Datos luego de la salida de la función de Stolt. Dominio 2D de las frecuencias.
$\boldsymbol{S4}$	Datos luego de la aplicación de la IFFT. Dominio del tiempo, 2D.
$\boldsymbol{Sw}$	Ancho de barrido (swath) del rango oblicuo.
$\boldsymbol{T_r}$	Duración de la transmisión del pulso.
$\boldsymbol{\tau}$	Tiempo rápido en rango.
$\boldsymbol{\theta_{ref}}$	Función de referencia en rango.
$\boldsymbol{\theta_{rfm}}$	Fase luego de la función RFM.
$\boldsymbol{\theta_{stolt}}$	Fase luego de la interpolación de Stolt
$\boldsymbol{V_r}$	Velocidad del Radar.
$\boldsymbol{w_a}$	Ventana en acimut.
$\boldsymbol{w_r}$	Ventana en rango.

Organización de la Tesis

En este apartado se describe la estructura de capítulos que conforman esta tesis:

CAPITULO 1 ~ INTRODUCCIÓN

Se realiza una revisión del estado del arte respecto del cómputo de los procesadores SAR utilizando GPU y se presentan los objetivos e hipótesis de la tesis. Se delinea también la metodología de desarrollo que guiará los trabajos subsiguientes.

CAPÍTULO 2 ~ MARCO TEÓRICO SAR

Este capítulo comienza revisando los fundamentos teóricos de la teoría de radares de apertura sintética, para abocarse luego a la explicación en detalle de los distintos pasos que conlleva el algoritmo WK. También se brinda una profunda explicación de la interpretación de la interpolación de Stolt mediante las propiedades de la Transformada Discreta de Fourier y una interpretación geométrica.

CAPÍTULO 3 ~ MARCO TEÓRICO: ARQUITECTURA GPU Y CUDA

Aquí se introduce a la arquitectura de GPU Kepler de Nvidia en la que se basa la GPU seleccionada para este trabajo: Tegra K1. También se introduce a las características principales del modelo de programación CUDA.

CAPÍTULO 4 ~ VALIDACIÓN

Se describen los distintos criterios de validación utilizados en el trabajo. Se propone un criterio para comparar la resolución espacial obtenida por el sistema respecto de los valores teóricos esperados. Dado que a partir de las tres implementaciones del WK que se proponen: MATLAB, C y CUDA idealmente se espera obtener iguales resultados frente a las mismas entradas, se especifica aquí un criterio para decidir cuando dos versiones pueden considerarse equivalentes entre sí. Finalmente se presenta el criterio de la PLSR (*Peak Side-lobe Ratio*) sobre la calidad de focalización obtenida.

CAPÍTULO 5 ~ DISEÑO DEL SOFTWARE

En este capítulo se presenta la lista de requerimientos del sistema desde en un alto nivel conceptual basado en objetivos e hipótesis planteadas para luego obtener una derivación de requerimientos cada vez con mayor detalle. También se desarrollan las fases de Análisis, Diseño e implementación del algoritmo WK.

CAPÍTULO 6 ~ RESULTADOS DE VALIDACÓN Y FOCALIZACIÓN

Aquí se muestran los resultados de la aplicación de los criterios de validación introducidos en el Capítulo 3, para el caso de la dimensión en Rango y Acimut de forma separada.

CAPÍTULO 7 ~ RESULTADOS DE PERFORMANCE DEL SOFTWARE

Se exponen los resultados de la performance en tiempo para las versiones de WK implementadas en C y CUDA dos formas: Primero como un todo, y luego discriminando función por función.

CAPÍTULO 8 ~ VERIFICACIÓN

Finalmente se comprueba que todos los requerimientos del último nivel de agregación hayan sido desarrollados.

CAPÍTULO 9 ~ CONCLUSIONES

Se presentan las conclusiones del trabajo de tesis.

CAPITULO 1 ~ INTRODUCCIÓN

Este trabajo propone realizar una implementación del algoritmo WK (Cafforio et al., 1991) para el enfoque de datos de Radar de Apertura Sintética (SAR) utilizando el paradigma de programación paralela en placas gráficas de propósitos generales (GP-GPU). En particular se analizará la implementación en una placa con un procesador embebido NVDIA Tegra K1, de modo que el hardware sea de pequeñas dimensiones, bajo consumo, portátil e independiente de una computadora como el caso de las placas gráficas usuales. El procesamiento de los datos generados por un sistema SAR consiste en una serie de operaciones factibles de ser paralelizadas. Para llevar adelante este trabajo se realiza un estudio del sistema SAR y en particular del algoritmo de enfoque WK. Se ha seleccionado este algoritmo debido a que permite la compensación exacta de factores de distorsión que ocurren en aplicaciones SAR satelitales.

1.1 Aportes

El desarrollo de una versión del algoritmo WK para enfoque de señales de radares de apertura sintética, paralela y acelerada sobre una plataforma GPU de bajo consumo, podrá ser utilizada como punto de partida para el desarrollo de procesadores en misiones futuras de la CONAE. Este trabajo puede dar inicio a líneas de investigación sobre procesamiento masivo de datos en vuelo con GP-GPU, no explorado hasta la actualidad.

1.2 Estado del arte

Durante el relevamiento del estado del arte sobre el algoritmo WK rápidamente quedó en evidencia la escasez de publicaciones que expongan detalles sobre la implementación del algoritmo en lenguajes comerciales para CPU y menos aún sobre implementaciones paralelas para GPU.

Por otra parte acceder a los detalles teóricos del algoritmo fue relativamente directo mediante autores como Cumming & Wong, Bu-Chin Wang, Carrara & Goodman y Soumekh.

Las siguientes, son dos referencias que se aproximan al enfoque dado en este trabajo, pero como se puede notar, al momento de exponer detalles de implementación la información se encuentra recortada y difícilmente se puede usar como base para un desarrollo completo:

- Synthetic Aperture Radar Imaging on a Cuda Enabled Mobile Platform, 2014, Massimiliano Fatica NVIDIA Corporation.

- Near Real Time, Multi GPU WK Algorithm for SAR Processing, 2014, Tiriticco, Et Al. Sapienza University of Rome.

Por tales motivos, este trabajo plantea una implementación desde cero del WK en los lenguajes comerciales de alto nivel y de utilización estándar: MATLAB y ANSI-C, partiendo de las ecuaciones teóricas, basado fuertemente en el enfoque dado por los autores Cumming y Wong, para gradualmente conducir el desarrollo hacia una implementación paralela en CUDA-C para GPU.

1.3 Objetivos

1.3.1 Planteamiento General del Problema

El problema puede dividirse en 2 grandes etapas:

- Implementación secuencial del algoritmo WK.
- Implementación paralela del algoritmo WK.

La etapa secuencial consta de un prototipado en MATLAB el cual sirve para comprender el problema y definir una primera línea base que será la guía para todo el trabajo. La validación de esta implementación se realizó comparando contra los valores teóricos de resolución espacial para un blanco puntual calculado a partir de los parámetros del algoritmo y utilizando un criterio de calidad basado en la PSLR (*Peak Side-lobe Ratio*). Esta etapa también incluye la construcción de una versión en lenguaje ANSI-C basada en la versión MATLAB, la cual correrá sobre uno de los cuatro núcleos de CPU que posee la placa NVIDIA Tegra k1, buscando ahora obtener un mejor rendimiento en tiempo y definir la arquitectura del programa que servirá para encarar la versión paralela.

La etapa paralela, consta de una versión en CUDA-C para GPU utilizando la placa Tegra K1, la cual busca obtener enormes ganancias en tiempo respecto de las versiones secuenciales, y conocer los límites de rendimiento de la placa para distintos tamaños de problemas en WK. (Tamaño de la matriz de datos crudos NxN, creciendo N).

1.3.2 Objetivo

Implementación del algoritmo WK para la plataforma NVIDIA Tegra TK1 utilizando el lenguaje CUDA-C.

1.3.3 Objetivos Específicos

Objetivo específico 1

A partir de la teoría propuesta por los autores (Cumming & Wong 2005), obtener una implementación del algoritmo de enfoque SAR, WK en MATLAB, para un blanco puntual simulado.

Objetivo específico 2

A partir de la implementación en MATLAB obtener una versión secuencial en lenguaje C, adaptada a las CPUs del hardware NVIDIA Tegra K1, para un blanco puntual simulado.

Objetivo específico 3

A partir de la implementación en C, obtener una versión paralela del algoritmo implementándolo en CUDA, adaptado a la GPU del hardware NVIDIA Tegra K1, para un blanco puntual simulado.

1.4 Hipótesis

La presente implementación se basa en las siguientes hipótesis:

Hipótesis principal:

- Es posible obtener una implementación del algoritmo WK basado en un hardware paralelo, GPU y obtener un mejor rendimiento en tiempo que una implementación secuencial basada en CPU.

Otras hipótesis:

- Órbita de avance en acimut rectilínea.
- Se asume que el ángulo de *Squint* es nulo.
- Se asume una superficie plana de la tierra.
- En la escena observada por el SAR, sólo existe un blanco puntual.
- El sistema SAR trabaja en modo *Stripmap*.
- La velocidad de avance la plataforma SAR en acimut se asume que no varía con el rango.

1.5 Metodología de desarrollo

La metodología de desarrollo que mejor se ajusta a la problemática del presente trabajo, es la cascada clásica.

En términos generales podemos dividir el proceso en siete fases:

- Relevamiento de requerimientos (Capítulo 5),
- Análisis (Capítulo 5),
- Diseño (Capítulo 5),
- Implementación (Capítulo 5 y Apéndices del B al E),
- Validación (Capítulo 4),
- Verificación (Capítulo 8)
- y Mediciones (Capítulo 7).

En la fase de implementación tenemos una cascada de alto nivel que secuencializa el desarrollo de las versiones: MATLAB, C y CUDA en ese orden. Luego para cada versión se aplica nuevamente una metodología en cascada para el conjunto de operaciones: FFT2D, RFM, STOLT, IFFT2D (se detallan en el Capítulo 2).

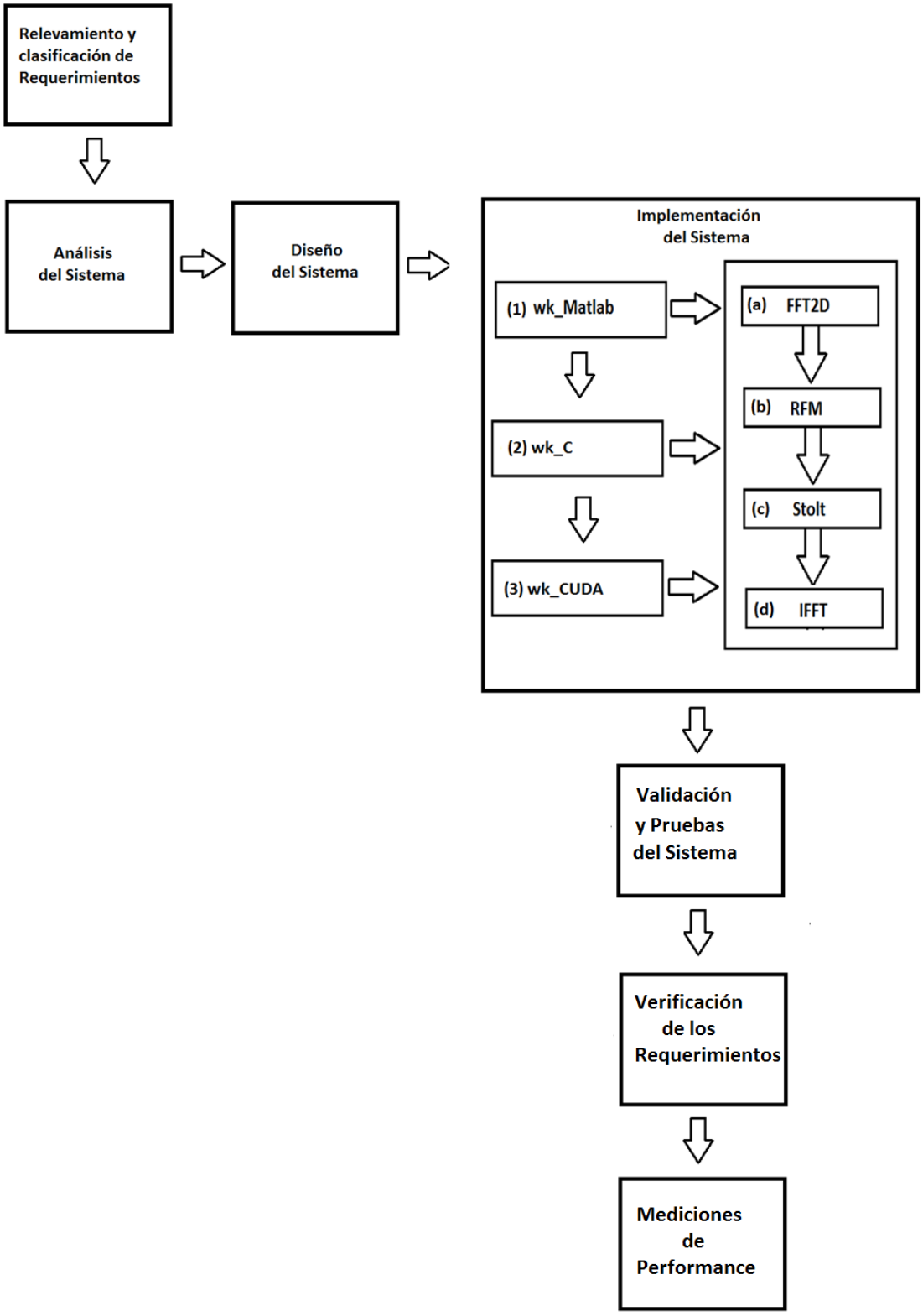


Figura 1.1, Metodología de Desarrollo.

CAPÍTULO 2 ~ MARCO TEÓRICO SAR

Este capítulo presenta los fundamentos teóricos de la teoría de radares de apertura sintética, para abocarse luego a la explicación en detalle de los distintos pasos que conlleva el algoritmo WK.

También se brinda una profunda explicación de la interpretación de la interpolación de Stolt mediante las propiedades de la Transformada Discreta de Fourier y una interpretación geométrica.

2.1 Geometría de adquisición SAR

Un radar de apertura sintética (SAR), es un sensor activo en el rango de microondas que ilumina la superficie terrestre emitiendo pulsos electromagnéticos en dirección perpendicular al movimiento.

Una de las características más comunes de los sensores SAR es la posibilidad de realizar adquisiciones tanto de día como de noche sin importar las condiciones climáticas. Otra característica fundamental para destacar es su orientación "*side-looking"* con respecto a la dirección de vuelo. El sensor SAR debe mirar lateralmente, debido a que de lo contrario le sería imposible determinar si la señal recibida corresponde a un blanco a la derecha o a la izquierda respecto de la dirección de avance.

El modo de operación planteado en este trabajo es llamado *Stripmap* y es quizá uno de los más populares (otros modos pueden ser: *Scan* o *Spotlight)*. En el modo *Stripmap* se pueden considerar dos geometrías, una llamada *Boresight*, en el cual el centro del haz emitido por la antena es perpendicular a la dirección de vuelo. La otra geometría se llama *Squinted*, esto es cuando la antena forma un ángulo, denominado *Squint* (puede ser *foreward* o *backward*), con la dirección de vuelo. En otras palabras el ángulo apuntado por la antena es de $(\pi/2)-\alpha$ grados, respecto de la dirección de vuelo, donde α es el ángulo de *Squint* y es relativo al centro del *Swath*.

El ángulo de *Squint* puede ser una característica deseada del sistema o bien puede deberse a inestabilidades en la actitud de la plataforma. En este trabajo se considera que el ángulo de *Squint* es nulo.

La (Figura 2.1), muestra la geometría de adquisición de datos de un sistema SAR típico. La plataforma, compuesta por un satélite y el sensor SAR, vuela siguiendo la dirección de *acimut* emitiendo pulsos y escuchando los ecos de retorno en dirección oblicua denominada *rango*. El retardo en el arribo del eco determina la distancia entre el sensor y el blanco.

Los pulsos son enviados con un ángulo de mirada θ (Figura 2.1), respecto de la dirección de vuelo a un intervalo de tiempo conocido como "*Tiempo de Repetición de Pulso*" o "*PRT*". Cada uno de estos pulsos ilumina un área de la superficie denominada "*pisada de la antena"* o "*footprint*". El ancho de la superficie que es iluminada por el sensor se conoce como "ancho de barrido" o "*Swath*" y queda determinado por la distancia entre los rangos cercano y lejano.

A medida que la plataforma avanza un blanco puntual en el terreno es iluminado varias veces, durante un lapso tiempo definido t1 y t2 (Tiempo de Integración) recorriendo una distancia L (Figura 2.0), hasta quedar fuera de la vista de la antena.

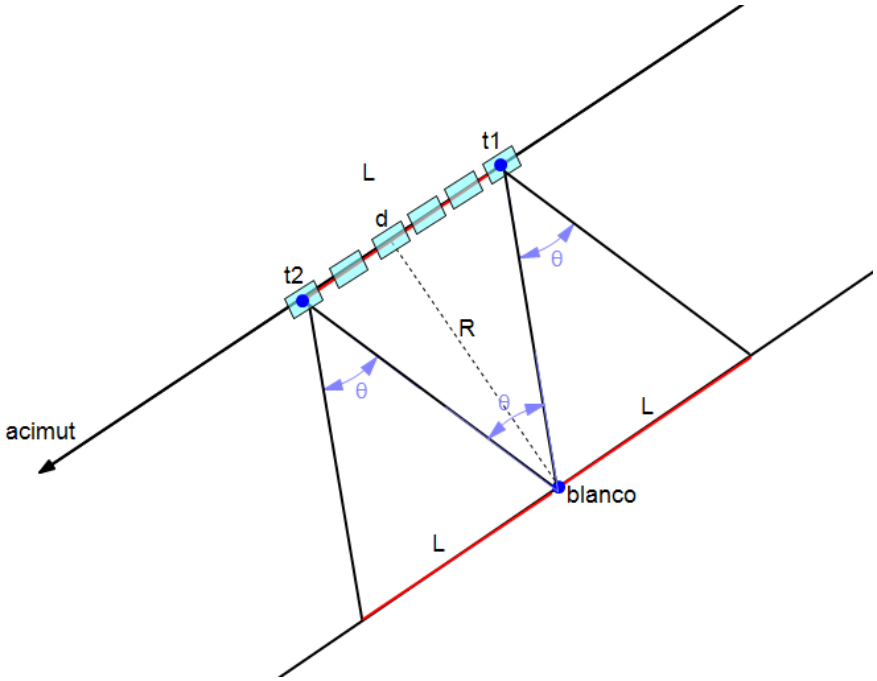


Figura 2.0, Apertura Sintética. Adaptación de (Franceschetti-Lanari 1999)

Si d es la longitud de la antena en acimut, estudiando la fase de los ecos del blanco se puede obtener una resolución tan fina como si hubiera sido adquirido por una antena mucho más grande de ancho L, la cual se denomina Apertura Sintética, y se define por:

$$L = \theta * R;\ \theta = \frac{\lambda}{d} \tag{2.2.0}$$

Donde θ (Figura 2.0), es la apertura del haz en acimut, λ es la longitud de onda del pulso y R es el rango oblicuo (*Slant Range*).

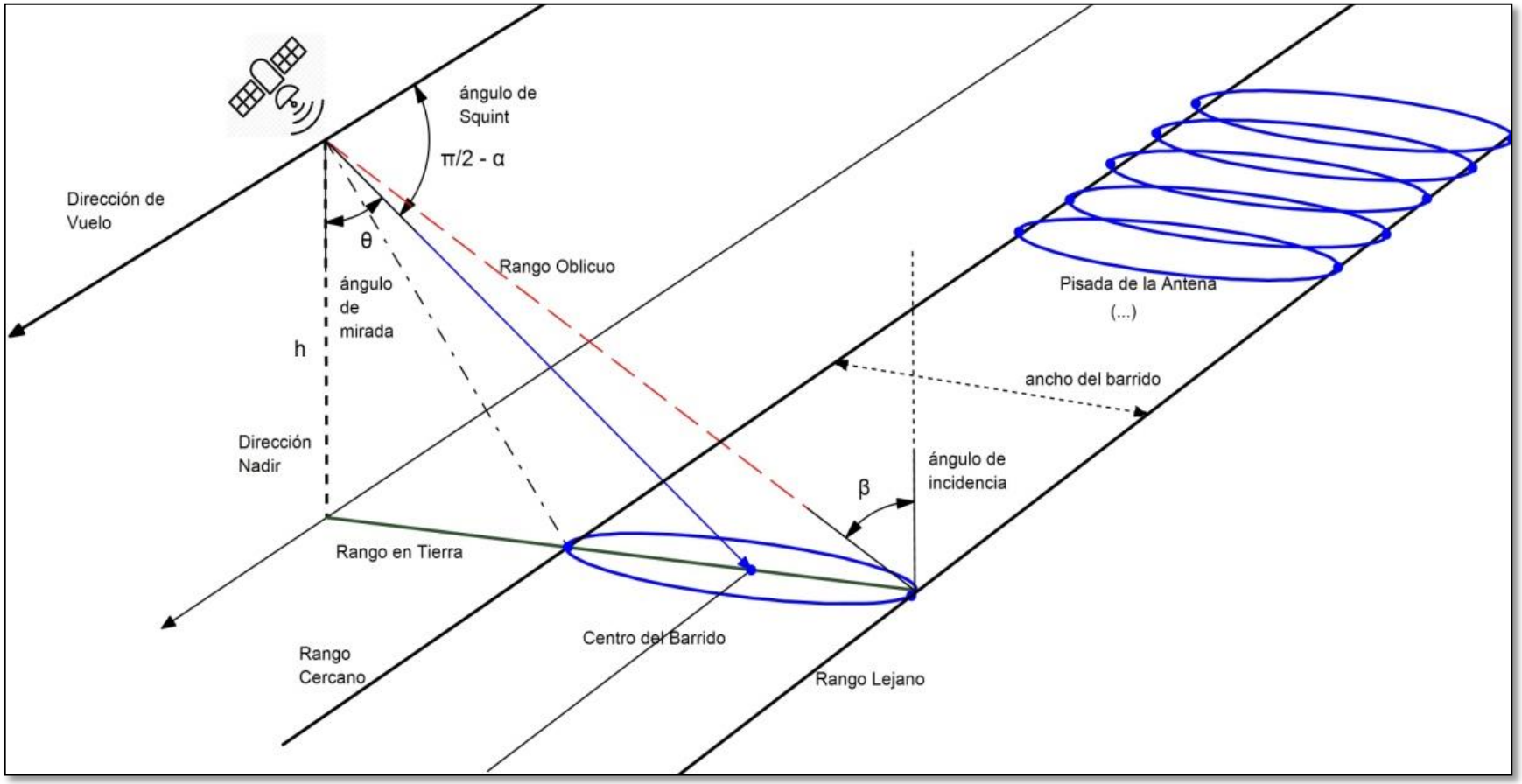


Figura 2.1, Geometría de Adquisición SAR. Adaptación de (Franceschetti-Lanari 1999) y (Curlander y McDonough, 1991)

2.2 Resolución Espacial teórica

La resolución espacial, se define como la distancia mínima (metros) dentro de la cual es posible distinguir la presencia de dos objetos:

Resolución en rango teórica: $\frac{C}{2*B_S}$ = 1.5 metros (2.2.1)

Resolución en acimut teórica: $\frac{D}{2}$ = 0.5 metros (2.2.2)

Dónde:

Variable	Descripción
C	$3x10^8$ [m/s] es la velocidad de la luz
Bs	$1x10^8$ [Hz] es el ancho de banda del chirp
D	1[m] es la longitud de la antena en acimut

Tabla 2.0 Parámetros de resolución espacial en rango y acimut

Estos valores teóricos serán usados posteriormente en el procedimiento de validación comparándolos contra la resolución medida en acimut-rango en número de celdas (convertidas en unidades de distancia).

2.3 Pixel-Spacing

El Pixel Spacing o espaciado entre celdas se expresa en unidades de distancia (metros) y queda determinado por la tasa de muestreo y la velocidad en cada dimensión (Cumming y Wong, 2005):

Pixel Spacing en rango: $\frac{C}{2*f_r}$ = 1.25 metros (2.3.1)

Pixel Spacing en acimut: $\frac{V_r}{PRF}$ = 0.42 metros (2.3.2)

Donde,

Variable	Descripción
C	3x10^8 [m/s] es la velocidad de la luz
f_r	120x10^6 [Hz] es la frecuencia de muestreo en rango
V_r	250[m/s] es la velocidad del SAR
PRF	600[Hz] es la frecuencia de muestreo en acimut

Tabla 2.1 Parámetros del pixel spacing en rango y acimut

2.4 Datos crudos

Sea una matriz de datos crudos de NxM, podemos Interpretar el significado de las dimensiones en filas y columnas de la matriz de la siguiente forma:

El sensor que escucha los ecos se abre N veces cada $\frac{1}{PRF}$ segundos y en cada apertura toma M muestras cada $\frac{1}{f_r}$ segundos.

La separación entre columnas está dado por: $\frac{1}{f_r}$ en unidades de tiempo. Se denomina *tiempo rápido*, al tiempo en la dirección del rango, porque es del orden del microsegundo. La separación entre filas está dado por: $\frac{1}{PRF}$ en unidades de tiempo. Se denomina *tiempo lento*, al tiempo en la dirección de acimut, porque es del orden del milisegundo (Curlander y McDonough, 1991).

Si quisiéramos cambiar la escala y expresar la separación en filas y columnas en metros, cada eje se debe multiplicar por el *pixel spacing* correspondiente.

Asimismo interpretamos el valor de las celdas de la siguiente forma:

Una celda de la matriz de datos crudos es un valor en voltios luego de ser digitalizado en fase y cuadratura (*IQ: In-Phase Quadrature*) resultando en formato complejo por conveniencia para los cálculos. Proceso que se lleva a cabo en el receptor SAR (Bu-Chin Wang, 2008).

Este valor de celda, en el mundo físico representa la integración (que realiza el receptor) de la energía de rebote (eco) de todos los objetos en igual rango respecto de la posición en acimut en que la antena del SAR los iluminó. La posición de igual rango o iso-rango queda caracterizada totalmente por el tiempo en que ese valor fue muestreado. Es decir, el retardo en tiempo en que se muestrea un valor es equivalente a la distancia en el terreno.

Ejemplificando: en la (Figura 2.2) podemos observar dos posiciones en acimut durante la trayectoria de vuelo del SAR, en la primera posición de la derecha se esquematiza la emisión de un chirp en rango y la respuesta de dos ecos desde el terreno. El eco marcado por la banda azul, que realizará un viaje de retorno según el rango oblicuo R2, será recibido por el sensor en tiempo posterior al tiempo en que arriba el eco de la banda roja, el cual viaja una distancia R1.

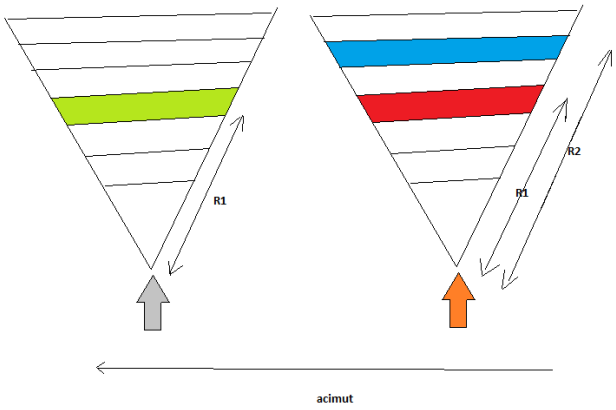


Figura 2.2 Emisión del chirp y recepción del eco

La representación de estos ecos en la matriz de datos adopta la siguiente forma:

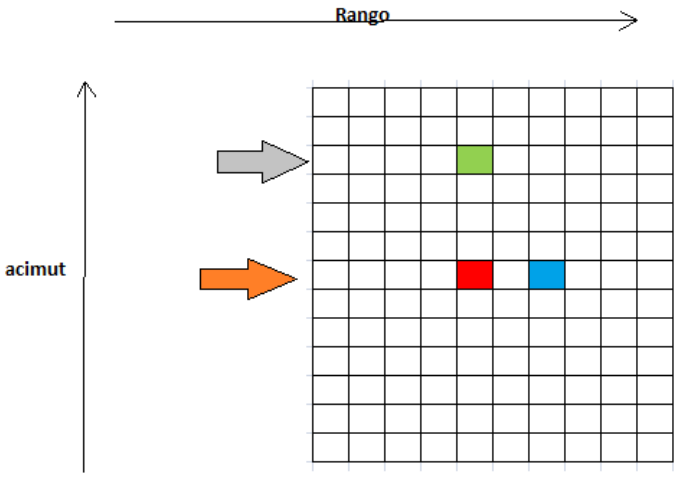


Figura 2.3 Representación en memoria de las señales recibidas

Para una misma posición de acimut: nro. de fila, cada eco que arriba se ubica en una posición de columna distinta.

2.5 Introducción al enfoque SAR

La matriz de datos crudos de entrada al algoritmo está conformada por la suma coherente de una gran cantidad de ecos correspondientes a distintos blancos distribuidos en la escena iluminada por el radar. La potencia proveniente de cualquier blanco puntual se encuentra "dispersa" en la matriz de datos crudos (Figura 2.6), es por ello que resulta necesario un procesamiento que permita obtener una imagen a partir de estos datos. Este proceso es comúnmente denominado focalización o compresión. La imagen obtenida posee valores en formato de número complejo lo cual permite guardar información de la amplitud y fase de la señal recibida. La focalización se realiza en dos dimensiones: rango y acimut.

A medida que el sensor SAR avanza en la dirección de acimut emite pulsos de microondas en dirección del rango y recibe sus ecos. En general el pulso emitido: denominado *chirp en rango* posee fase cuadrática y por lo tanto su frecuencia varía linealmente en el tiempo.

Típicamente los algoritmos de radar recuperan la señal original realizando una operación de correlación cruzada entre el pulso chirp emitido en la dirección del rango y el eco recibido, esta operación también es conocida como *focalización en rango*.

El efecto del avance del sensor con velocidad V en la dirección de acimut, provoca un cambio de fase en la señal que recibe el sensor en cada muestra en acimut. Debido a esto la señal proveniente de un objeto, en tiempo lento, posee una variación de frecuencia lineal y por ello se denomina *chirp en acimut*. Este efecto es dependiente del rango por lo que se necesita aplicar una correlación distinta, ya no contra un solo chirp como en el caso anterior, sino contra una familia de chirps: uno para cada rango, esta operación también es conocida como *focalización en acimut*.

A diferencia de otros algoritmos de focalización como *Range Doppler* o *Chirp Scaling*, *WK*, realiza estos dos tipos de focalizaciónes al mismo tiempo y en dos grados de detalle: primero focaliza en rango y acimut durante la función RFM (Sección 2.7.2), haciendo una focalización gruesa basada en un rango de referencia y posteriormente la refina utilizando la interpolación de Stolt (Sección 2.7.3), también operando en rango y acimut al mismo tiempo.

2.6 Simulación de la Respuesta al impulso de un sistema SAR

El simulador de datos crudos utilizado se basa en la respuesta al impulso de un sistema SAR (Cumming y Wong, 2005):

$$h_{impulso}(\tau,\eta) = w_r\left(\tau - 2\frac{R(\eta)}{C}\right) * w_a(\eta-\eta_c) * e^{-j4\pi f_0\left(\frac{R(\eta)}{C}\right)} * e^{j\pi K_r\left(\tau-2\frac{R(\eta)}{C}\right)^2} \quad (2.6.1)$$

Esta expresión representa la señal recibida de un blanco puntual demodulada y en banda base. Esta señal es la que se almacena en la memoria del sistema SAR como "dato crudo" o también llamada "historia de fase SAR"

Dónde:

$R(\eta)$, es la ecuación del rango, definida por:

$$R(\eta) = \sqrt{(R_0^2 + V_r^2\eta^2)} \quad (2.6.2)$$

Dónde:

Variable	Descripción
τ	Es el tiempo rápido en la dirección del rango.
η	Es el tiempo lento en la dirección de acimut.
η_c	Es el tiempo de intersección del haz de la antena SAR con el plano de cero doppler, el cual es cero en caso de que el ángulo de *squint* sea cero.
$C,$	Es la velocidad de la luz.
K_r	Es la frecuencia del chirp en rango.
w_r	Función ventana en rango.
w_a	Función ventana en acimut.
R_0	Es la velocidad efectiva del radar.
V_r	Es el rango oblicuo en el punto de cero doppler.

Tabla 2.2 Parámetros de la respuesta al impulso SAR

Ventana en Rango:

Para la implementación de w_r (envolvente del *chirp* en rango) se utilizó una ventana de tipo *Kaiser* con el parámetro β=2.5. (Oppenheim, 1999). La utilización de la ventana w_r surge de la necesidad de moderarlos lóbulos laterales del pulso en rango. Ejemplo de una ventana *Kaiser*, para 200 muestras y β=2.5

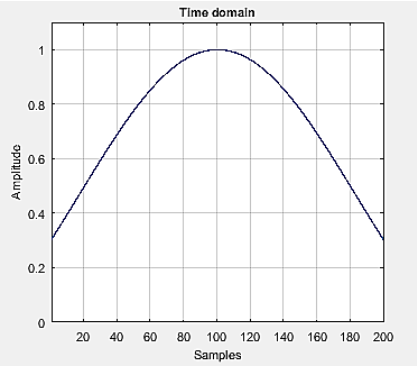


Figura 2.4 Ventana Kaiser

Ventana en Acimut:

Para la implementación de w_a se consideró una función $Sinc(.)^2$. Esta ventana se utiliza para modelar el patrón de antena en la dirección de acimut. Un ejemplo para este tipo de ventana, considerando 100 muestras puede apreciarse en la siguiente ilustración:

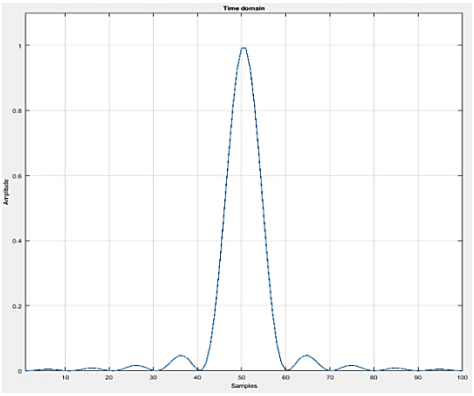


Figura 2.5 Ventana Sinc

La primera exponencial:

$$e^{-j4\pi f_0\left(\frac{R(\boldsymbol{\eta})}{C}\right)} \tag{2.6.3}$$

Es el aporte en fase que realiza el retardo debido al tiempo de viaje del pulso transmitido, es dependiente del rango.

Cabe destacar que $\frac{R(\eta)}{C}$ corresponde al tiempo de viaje del *chirp* en rango y el factor "2" es debido a que se considera un viaje redondo del pulso: la emisión de la señal y retorno del eco.

La segunda exponencial:

$$e^{j\pi K_r\left(\tau - 2\frac{R(\eta)}{C}\right)^2} \tag{2.6.4}$$

Representa la fase cuadrática debido al *chirp* en rango.

La siguiente figura muestra la dispersión de energía de un blanco puntual en la matriz de datos crudos que resulta de la ecuación de la respuesta al impulso $h_{impulso}(\tau, \eta)$. Nótese el efecto de la curvatura de la ecuación de rango $R(\eta)$, denominado "*migración de celdas en rango*" a medida que el sensor SAR avanza en acimut durante el período de tiempo correspondiente a la apertura sintética (tiempo de exposición)

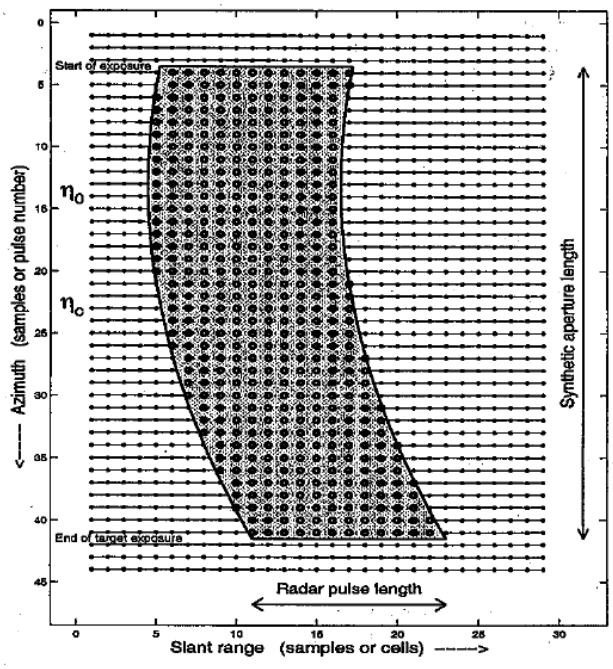


Figura 2.6, Ilustración de la Migración de la energía de un blanco puntual en rango (Cumming & Wong, 2005)

La migración de celdas en rango, también llamado en la literatura como *RCM* por su nombre en inglés "*Range Cell Migration*" es un efecto que todos los procesadores SAR deben corregir y que el algoritmo WK lo resuelve durante mediante la función RFM, también llamada etapa de focalización *Bulk*, o gruesa.

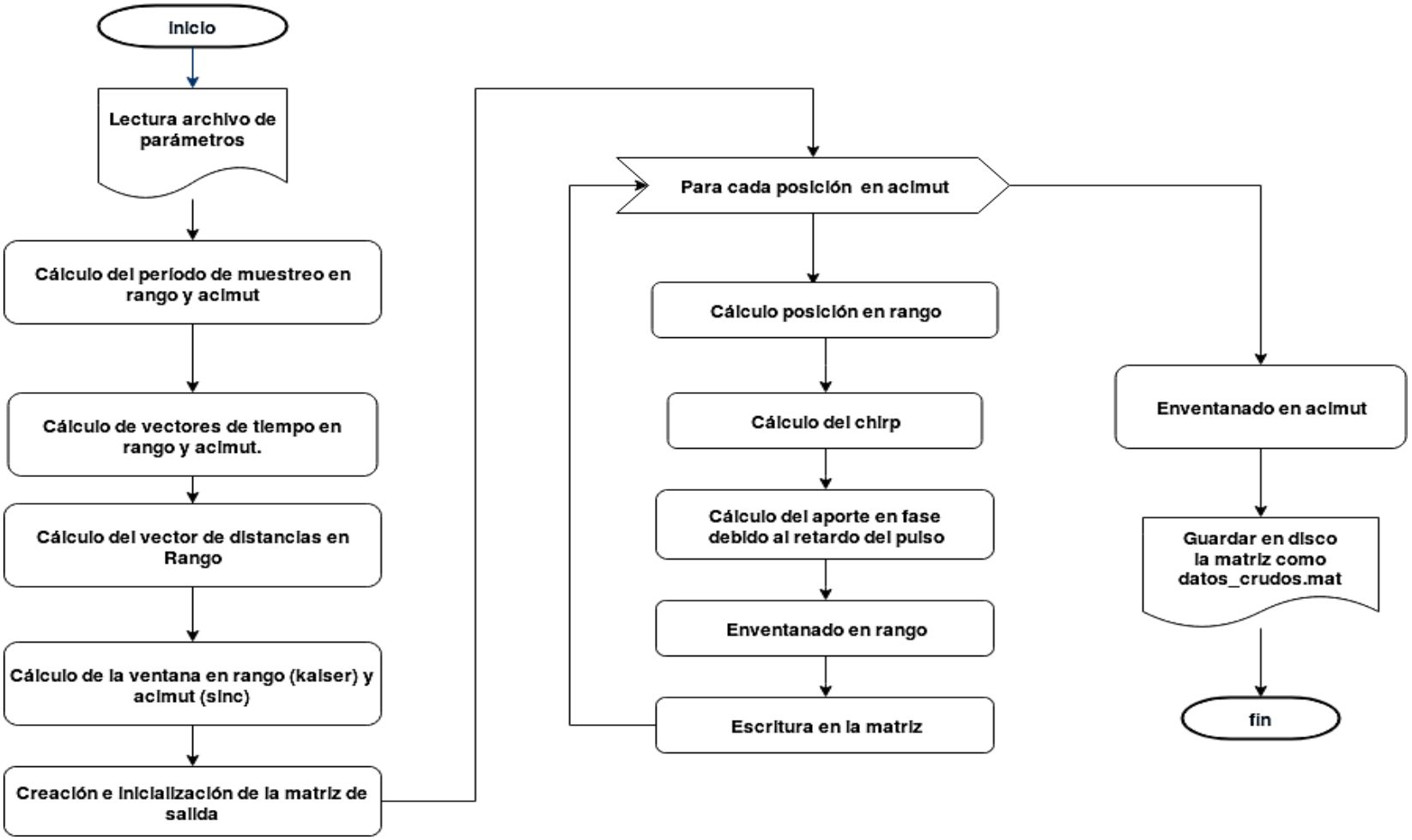


Figura 2.7, Diseño del simulador de respuesta al impulso SAR (Simulador de datos crudos para un blanco puntual)

Parámetros del simulador en Matlab:

Parámetro	Valor	Descripción	Unidad
F_r	120e^{6}	Frecuencia de muestreo en rango	[Hz]
PRF	600	Frecuencia de muestreo en acimut	[Hz]
η_c	0	Beam Crossing Time	[s]
H	10000	Altitud	[m]
Sw	10000	Ancho de barrido (*swath*) del rango oblicuo	[m]
R_{nc}	25000	Centro de la escena del rango oblicuo	[m]
GR_{nc}	$\sqrt{(H^2 + R_{nc}^2)}$	Posición del centro de la escena	[m]
R_{ca}	$\sqrt{\left(\left(GR_{nc} - \frac{Sw}{2}\right)^2 + H^2\right)}$	Rango cercano	[m]
R_t	26000	Posición del blanco puntual	[m]
V_r	250	Velocidad del Radar	[m/s]
T_r	10e^{-6}	Duración de la transmisión del pulso	[S]
λ	0.032	Longitud de onda del radar	[m]
L_a	1	Longitud de la antena SAR	[m]
f_0	9.4e^{9}	Frecuencia de la portadora	[Hz]
k_r	10e^{12}	Frecuencia del *chirp* en rango.	[1/s^2]
N_η	2048	Dimensión en acimut	número de muestras
N_τ	4096	Dimensión en rango	número de muestras

Tabla 2.3, parámetros del simulador de datos crudos para un blanco puntual

2.7 Introducción al algoritmo WK

El algoritmo WK fue Creado en 1991 por Cafforio, Prati y Rocca. Es uno de los algoritmos actuales más precisos comparado con RDA (*Range Doppler Algorithm*) y CSA (*Chirp Scaling Algorithm*), realiza la focalización de la imagen trabajando íntegramente en el dominio bidimensional de las frecuencias y de allí su nombre, W: frecuencia en Rango y K: frecuencia en acimut. Sus principales ventajas es que puede manejar grandes aperturas sintéticas o elevados ángulos de *Squint*. El algoritmo WK también se lo conoce con el nombre de *Range Migration Algorithm, RMA* (Carrara, 1995). Descripción del algoritmo WK (Cumming y Wong, 2005):

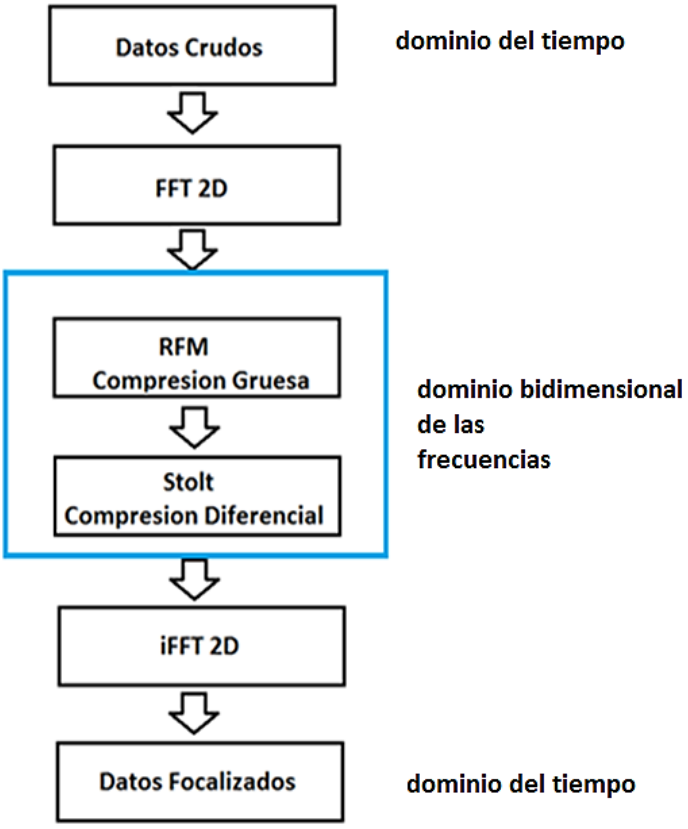


Figura 2.8, Algoritmo WK, (Cumming y Wong, 2005)

2.7.1 Transformada de Fourier Directa-2D

Se aplica la transformada rápida de Fourier bidimensional FFT-2D a los datos crudos para pasar al dominio bidimensional de las frecuencias: rango - acimut.

La Transformada de Fourier 2D, puede ser pensada en dos partes, como una transformada 1D en rango, seguida de una transformada 1D en acimut:

La Transformada en rango resulta:

$$S_{rango}(f_\tau, \eta) = \int_{-\infty}^{\infty} s_0(\tau, \eta) e^{-j2\pi f_\tau \tau}\, d\tau \tag{2.7.1}$$

Donde τ y η es el tiempo rápido y tiempo lento respectivamente, $s_0(\tau, \eta)$ es la señal en banda base, f_τ es la frecuencia en rango y $j = \sqrt{-1}$.

Luego al aplicar la transformada en acimut se obtiene:

$$S_{2df}(f_\tau, f_\eta) = \int_{-\infty}^{\infty} S_{rango}(f_\tau, \eta) e^{-j2\pi f_\eta \eta}\, d\eta \tag{2.7.2}$$

Donde f_η es la frecuencia en acimut.

El espectro resultante tiene la siguiente expresión en banda base:

$$S_{2df}(f_\tau, f_\eta) = \; W_r(f_\tau)\, W_a(f_\eta - f_{\eta c})\, \mathrm{e}^{j\,\theta_{2df}(f_\tau, f_\eta)} \tag{2.7.3}$$

Donde $f_{\eta c}$ es el centroide doppler, $W_r(f_\tau)$ es una ventana en rango y $W_a(f_\eta - f_{\eta c})$ es la envolvente en acimut centrada en $f_{\eta c}$. Las ventanas W_r y W_a, no forman parte de este análisis puesto que ya fueron consideradas durante la generación de los datos crudos y no deben ser compensadas (no se modifican) durante el proceso de focalización.

La fase θ_{2df} tiene la forma:

$$\theta_{2df}(f_\tau, f_\eta) = -\left(\frac{4\pi R_0}{c}\right)\sqrt{(f_0 + f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} - \frac{\pi {f_\tau}^2}{K_r} \tag{2.7.4}$$

Donde R_0 es el rango cercano, C la velocidad de la luz, f_0 es la frecuencia de la portadora, f_τ es la frecuencia en rango, f_η es la frecuencia en acimut, V_r es la velocidad de la plataforma en acimut y K_r es la frecuencia del chirp en rango.

2.7.2 RFM

La RFM, "Multiplicación por la función de referencia" o "focalización gruesa", se computa para un rango seleccionado llamado "Rango de Referencia", generalmente se toma el punto medio del *Swath* (ancho del barrido). Luego de su aplicación, un objetivo en el rango de referencia estará correctamente focalizado, mientras que los demás objetivos en otros rangos quedarán parcialmente focalizados. La "desfocalización" aumenta con la distancia al rango de referencia.

En base a (2.7.4) se construye la fase θ_{ref}:

$$\theta_{ref}(f_\tau, f_\eta) = +\left(\frac{4\pi R_{ref}}{c}\right)\sqrt{(f_0+f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_{Rref}}^2}} + \frac{\pi {f_\tau}^2}{K_r} \tag{2.7.5}$$

Donde R_{ref} es el rango de referencia y V_{Rref} es la velocidad de referencia que corresponde a velocidad de la plataforma en acimut. Posteriormente se suman las dos fases (2.7.4) y (2.7.5) o bien se multiplica a (2.7.3) por $e^{\theta_{ref}}$ como se realizó en nuestra implementación.

La fase a la salida del filtro RFM toma la forma aproximada de:

$$\theta_{rfm}(f_\tau, f_\eta) \approx -\left(\frac{4\pi(R_0 - R_{ref})}{c}\right)\sqrt{(f_0+f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} \tag{2.7.6}$$

La ecuación es aproximada debido a la supocisión teórica de que V_r es independiente del rango, las consecuencias de esta supocisión conducen a errores significativos cuando V_r presenta variaciones del 0.15% en el *Swath* de un rango oblicuo de 50 km. Esta condición constituye una de las principales deventajas del algoritmo. (Cuming y Wong.2005)

2.7.3 Stolt

En esta etapa se realiza el enfoque diferencial correspondiente al resto de los rangos que no se encuentran en el rango de referencia. Se compensa efectos como la fase residual de la *Migración de Celdas en Rango* ó *RCM*, el *Acoplamiento Rango-Acimut* (equivalente a la *SRC*) y la *Fase Residual en Acimut* (modulación en acimut) (Cumming y Wong, 2005).

Las operaciones se realizan en dos etapas: primero plantea un cambio de variables o mapeo sobre el eje de coordenadas de las frecuencias en rango, desde f_τ a f_τ' y luego una interpolación de los valores de intensidad en f_τ en el conjunto de los f_τ', para obtener finalmente los $f_{\tau_{interp}}'$. Este procedimiento completa la compensación de la fase que comenzó con la RFM.

2.7.3.1 Mapeo de Stolt

El mapeo desde f_τ (eje de coordenadas de las frecuencias en rango) en f_τ' (nuevo eje de coordenadas de frecuencias en rango) se realiza de acuerdo a la siguiente expresión:

$$\boldsymbol{f_\tau'} = \sqrt{(f_0 + \boldsymbol{f_\tau})^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} - f_0 \qquad (2.7.7)$$

A continuación se presenta un esquema de visualización del mapeo de Stolt a partir de los valores de frecuencia en rango f_τ:

El cambio de variables de Stolt mapea desde un distribución equiespaciada de los datos (Figura 2.9) hacia una distribución que ya no se encuentra equiespaciada (Figura 2.10)

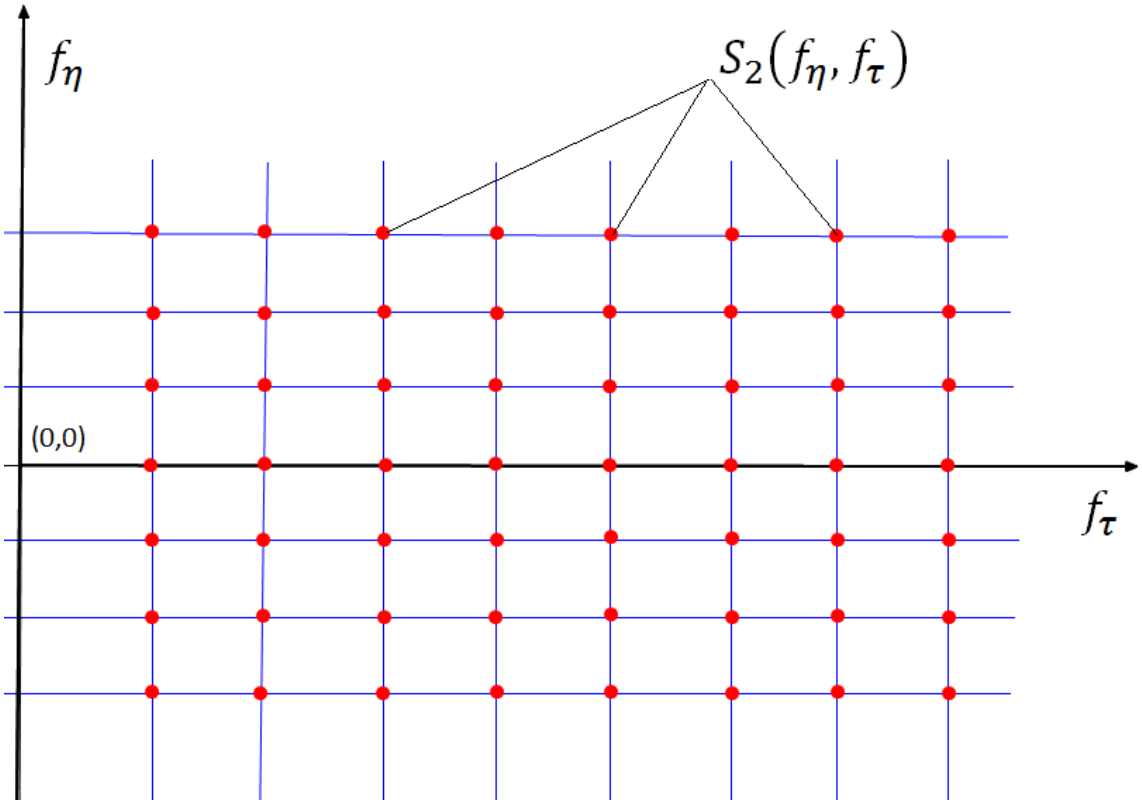


Figura 2.9 Mapeo de Stolt, grilla equiespaciada

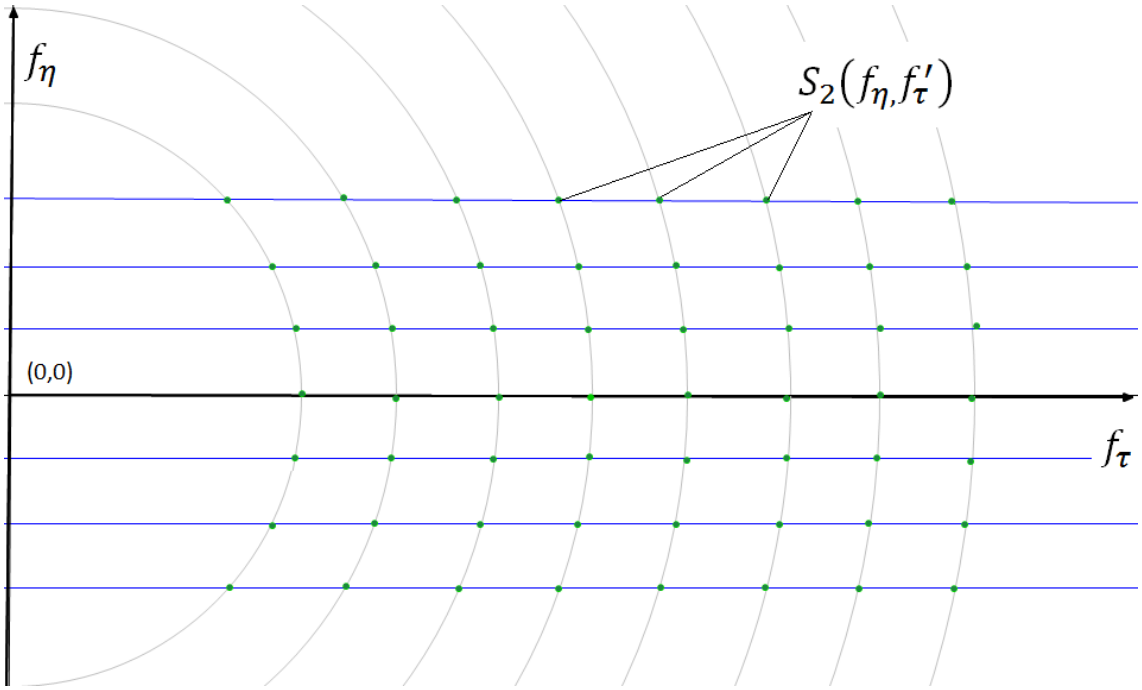


Figura 2.10 Mapeo de Stolt, grilla no-equiespaciada

Para el paso siguiente de interpolación es útil visualizar estos dos conjuntos de datos en un solo gráfico:

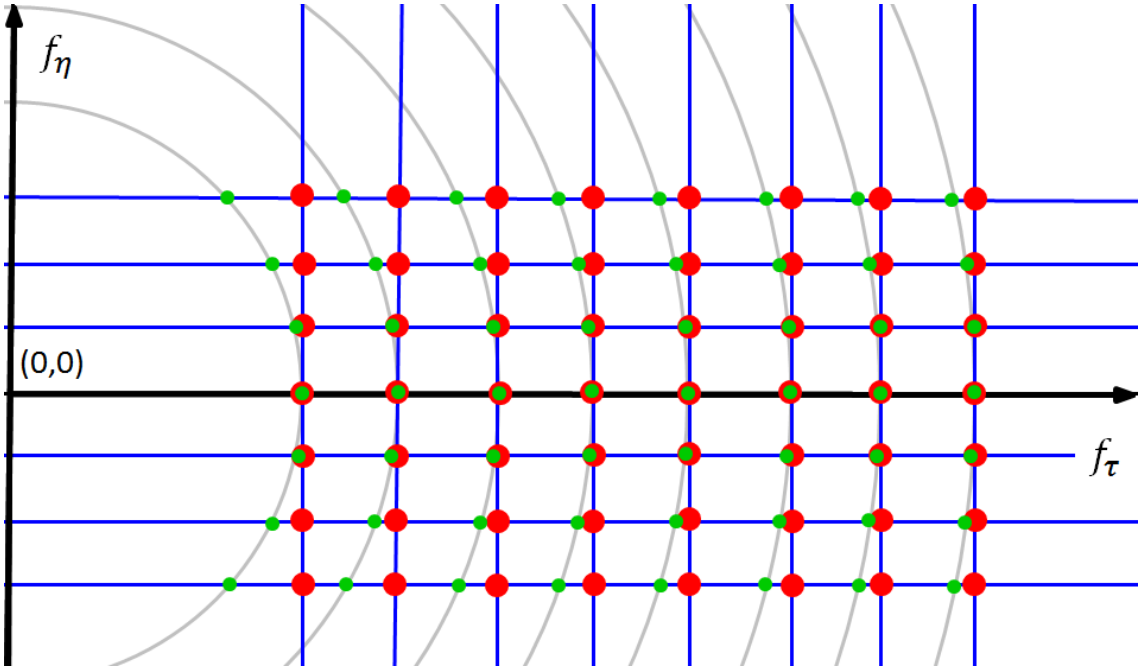


Figura 2.11 Mapeo de Stolt, grilla equiespaciada y no-equiespaciada superpuestas

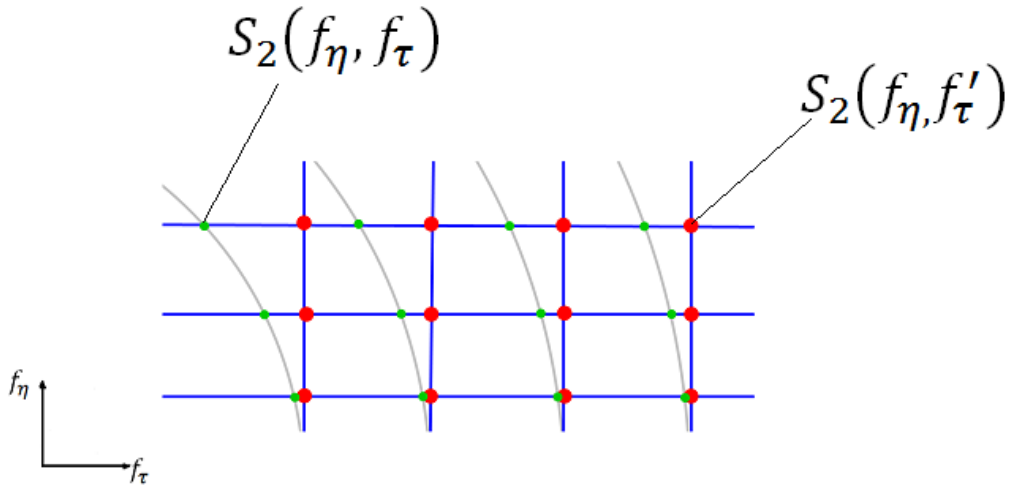


Figura 2.12 Mapeo de Stolt, zoom, grilla equiespaciada y no-equiespaciada superpuestas

2.7.3.2 Interpolación Sinc

Previo a presentar la interpolación de Stolt, revisaremos brevemente el concepto de interpolación $Sinc(.)$ con un ejemplo más simple.

Un procedimiento de interpolación se utiliza para equiespaciar nuevamente los f_τ' que fueron recientemente mapeados, lo cual resulta necesario para poder utilizar luego la transformada inversa de Fourier (IFFT).

Para esta tesis se utilizó un interpolador $Sinc(.)$ con un kernel de 8 muestras, como se muestra en la (Figura 2.13):

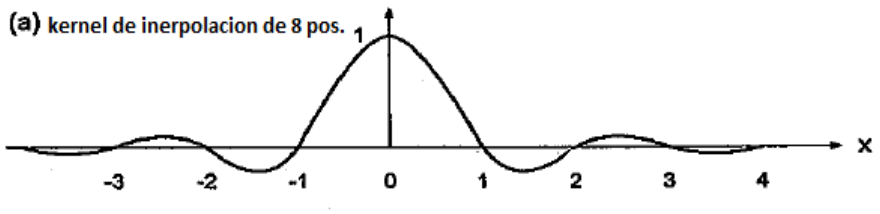


Figura 2.13 Kernel de 8 elementos

Procedimiento de *interpolación Sinc*:

Supongamos que se tiene un conjunto de muestras original $S=\{s_1, s_2, \ldots, s_n\}$

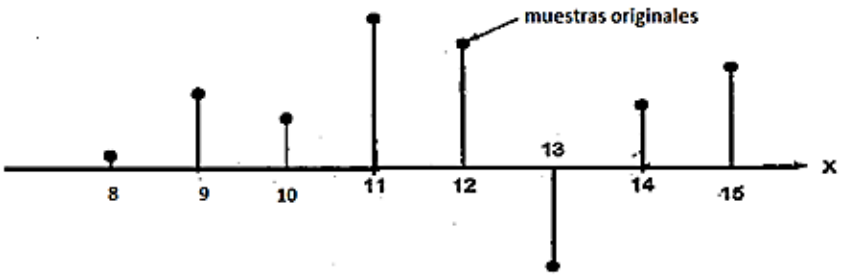


Figura 2.14 Muestras de una señal de genérica S

Y que se necesita interpolar el valor s' correspondiente a $x = 11.7$ a partir de los valores S. Se traslada el kernel de interpolación haciendo coincidir su cero con x. Luego se buscan los cortes entre la muestras originales y la función $Sinc(.)$ en 8 posiciones alrededor del valor a interpolar (indicados con *)

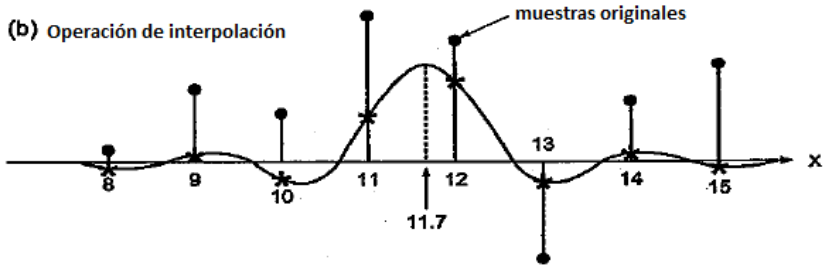


Figura 2.15 Operación de Interpolación

Se calcula entonces una suma de 8 productos entre el valor original de la muestra S_i y $Sinc(x_i)$. El resultado será el valor interpolado en el conjunto de muestras originales.

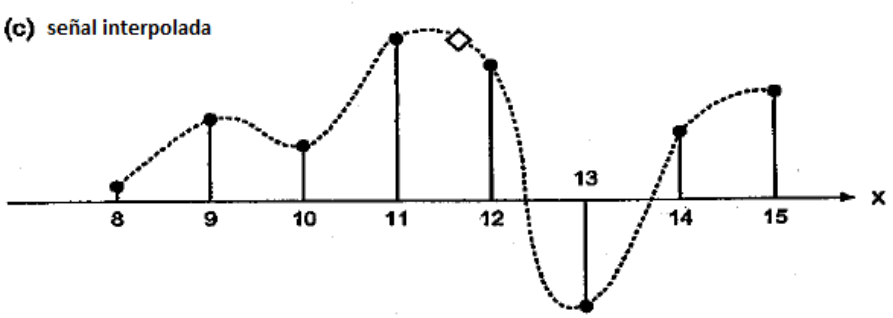


Figura 2.16 Señal interpolada

2.7.3.3 Interpolación de Stolt

Comprendido el procedimiento básico de *interpolación Sinc*, podemos analizar ahora la situación particular del WK. Luego del mapeo y antes de la interpolación los f_τ' están distribuidos en un lugar geométrico circular, no equiespaciados.

Como se mencionó en la sección anterior, se necesita volver a representar las muestras en una grilla de frecuencias equiespaciadas que permita realizar la transformada inversa de Fourier mediante la FFT. Para ello es necesario interpolar los valores de las muestras en f_τ' (que coinciden con los valores de las muestras en f_τ), a valores de frecuencias equiespaciadas f_{equi}.

Se procede trasladando el kernel de interpolación y haciendo coincidir su cero con la frecuencia a la que se desea interpolar:

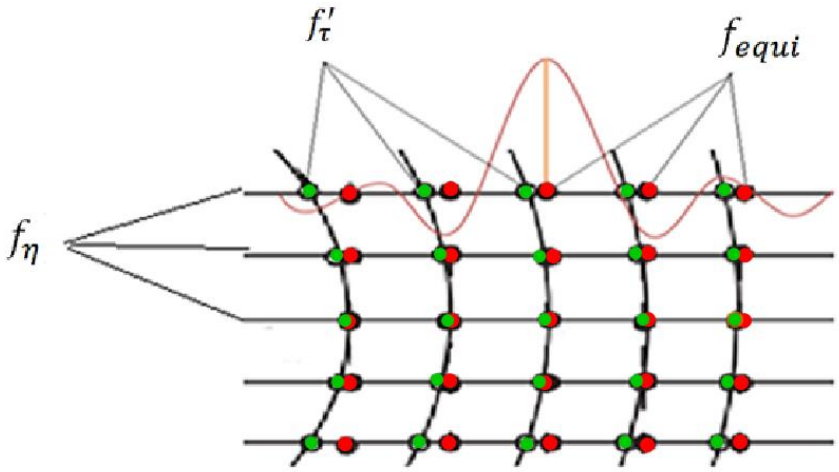


Figura 2.17 Aplicación de un Kernel de 8 posiciones, centrado en un elemento de la grilla equiespaciada, f_{equi}.

Observando el mismo gráfico para un acimut particular:

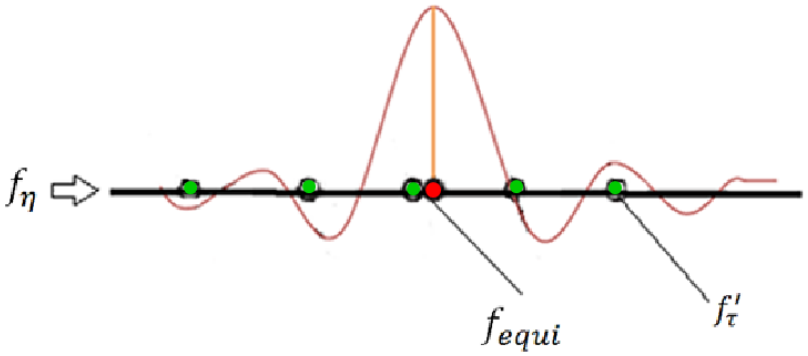


Figura 2.18 Aplicación de un Kernel de 8 posiciones, zoom, centrado en un elemento de la grilla equiespaciada, f_{equi}.

Para interpolar en cada f_{equi} necesitaremos del aporte de 8 muestras de su entorno correspondientes a f'_{τ} pesadas por la función $Sinc(f'_{\tau} - f_{equi})$. La amplitud de f_{equi}, será:

$$\theta_{stolt}(f_{equi}, f_{\eta}) = \sum_{i=1}^{8} \theta_{rfm}(f'_{\tau_i}, f_{\eta}) * Sinc(f'_{\tau_i} - f_{equi})$$

(2.7.8)

La fase luego de la interpolación de Stolt, resulta:

$$\theta_{stolt}(f_{equi}, f_{\eta}) = -\left(\frac{4\pi(R_0 - R_{ref})}{c}\right)(f_0 + f_{equi})$$

(2.7.9)

2.7.4 Transformada Fourier Inversa-2D

Con esta operación se retorna al dominio del tiempo.

Transformada inversa en rango:

$$S_{rango}(\tau, f_{\eta}) = \int_{-\infty}^{\infty} S_{2df}(f_{\tau}, f_{\eta})\, e^{+j2\pi f_{\tau} \tau}\, d\tau$$

Transformada inversa en acimut: (2.7.10)

$$S(\tau, \eta) = \int_{-\infty}^{\infty} S_{rango}(\tau, f_{\eta})\, e^{+j2\pi f_{\eta} \eta}\, d\eta$$

(2.7.11)

En la (Figura 2.19) se puede apreciar como la interpolación de Stolt completa el trabajo de focalización del blanco puntual iniciado por la función RFM, afinando y concentrando aún más la energía:

Luego de la función RFM

Luego de la interpolación de Stolt:

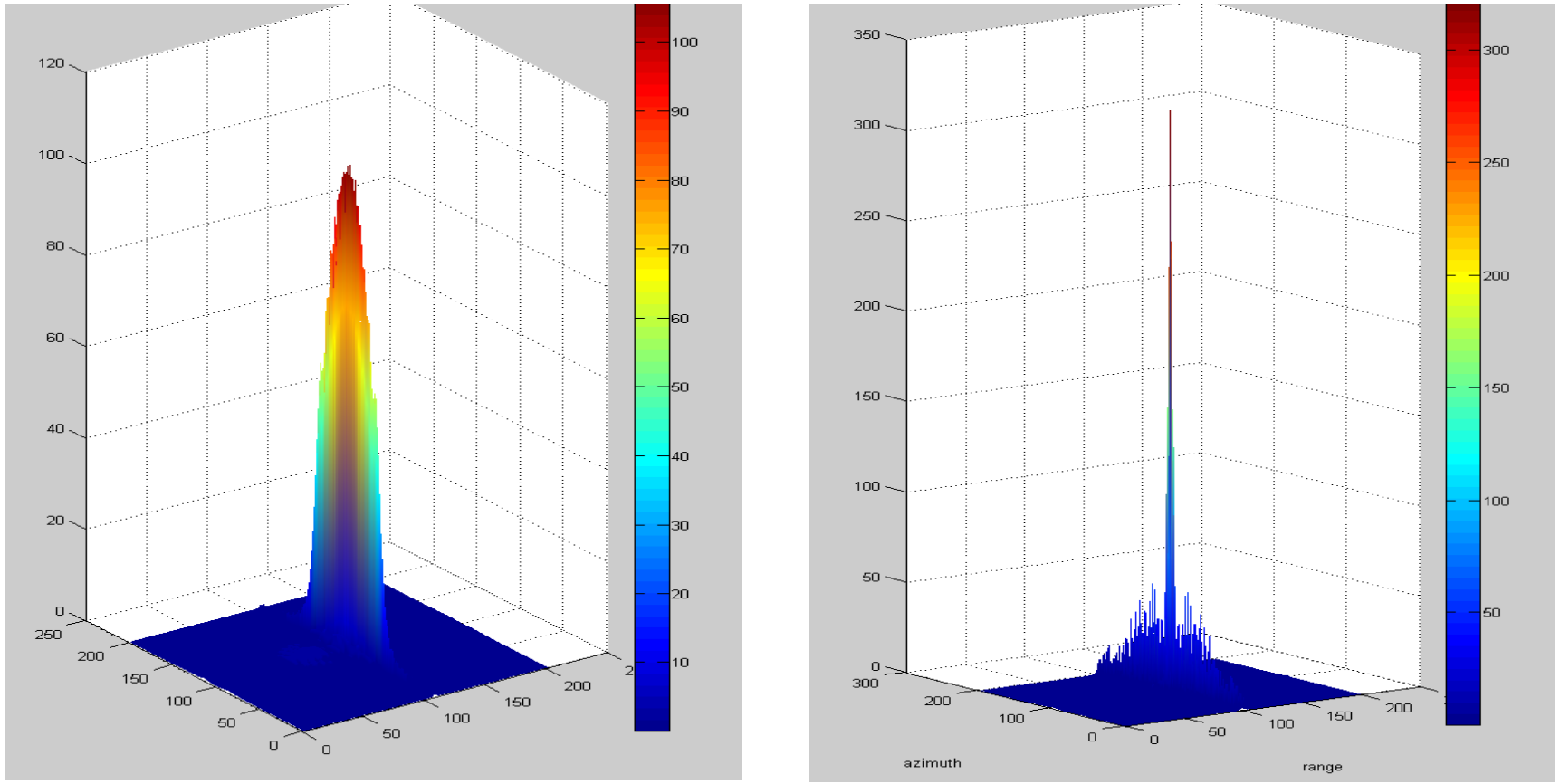


Figura 2.19, Gráficas del módulo de las señales en acimut vs rango en el dominio del tiempo, para las dos principales etapas del proceso de focalización. Recorte en 200x200 muestras alrededor del máximo.

2.8 Parámetros del algoritmo WK

Símbolo	Descripción	Valor	Unidad
V	Velocidad de la plataforma	250	[m/s]
F_c	Frecuencia de la portadora	$9.4x10^9$	[Hz]
F_{slow}	Frecuencia de muestreo en tiempo lento	600	[Hz]
F_{fast}	Frecuencia de muestreo en tiempo rápido	$120x10^6$	[Hz]
K_r	Frecuencia del Chirp en Rango	$10x10^{12}$	[Hz/s]
R_0	Rango de referencia	25500	[m]

Tabla 2.4 Parámetros del algoritmo WK

2.9 Interpretación de la Interpolación de Stolt mediante las propiedades de la Transformada Discreta de Fourier

Luego de la RFM la fase de la señal tiene la siguiente forma:

$$\theta_{rfm}(f_\tau, f_\eta) \approx -\left(\frac{4\pi(R_0 - R_{ref})}{c}\right)\sqrt{(f_0 + f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} \tag{2.9.1}$$

Donde R_0 es el rango cercano R_{ref} es el rango de referencia, c la velocidad de la luz, f_0 es la frecuencia de la portadora, f_τ es la frecuencia en rango, f_η es la frecuencia en acimut, V_r es la velocidad de la plataforma en acimut.

En la (Figura 2.20) podemos ver la situación actual luego de la multiplicación por la función de referencia RFM (cuadro a) para un blanco puntual.

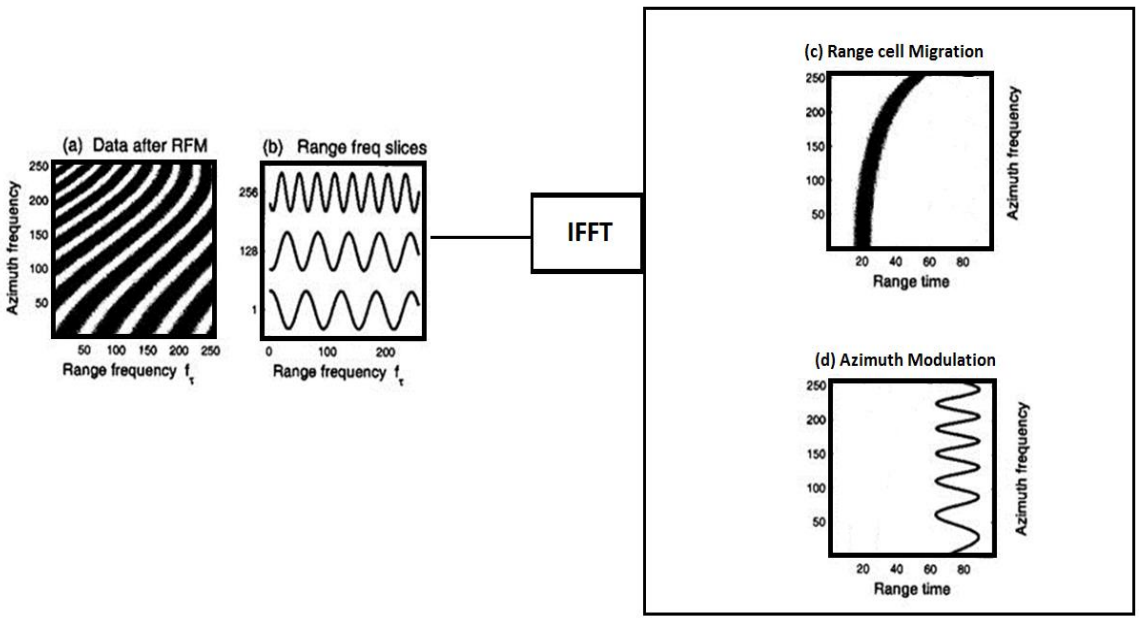


Figura 2.20, Interpretación de Stolt mediante las propiedades de la DFT. Procesamiento de un Blanco Puntual, luego de la RFM

Realizando distintos cortes horizontales en acimut (Figura 2.20) (cuadro b), podemos ver como la frecuencia en rango varía con la posición en acimut y mediante una IFFT la migración de celdas en rango queda en evidencia (cuadro c), como si también la modulación residual en acimut (cuadro d).

Para abordar la idea del mapeo de Stolt y cómo logra corregir la migración de celdas en rango (RCMC) y la fase residual en acimut mediante un cambio de variables se puede explicar según (Cumming & Neo, 2003) con dos expansiones de la raíz cuadrada de la expresión anterior:

$$\sqrt{(f_0 + f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} \tag{2.9.2}$$

Expansión 1:

Sirve para explicar cómo surge la idea del cambio de variables.

Expandiendo (2.9.2) por Taylor hasta el segundo término obtenemos (para ver la comprobación de esta derivación, consultar el Apéndice J, Sección J.1):

$$\sqrt{(f_0 + f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} = \left[(f_0 + f_\tau) - \left(\frac{c^2 {f_\eta}^2}{8{V_r}^2 (f_0 + f_\tau)}\right) + \cdots\right] \tag{2.9.3}$$

Aquí se observa que si en la expansión sólo existiera el término dado por $(f_0 + f_\tau)$ y se aplicara la IFFT a θ_{rfm}en (2.9.1) obtendríamos una compresión y registración perfecta en rango. Surge la idea entonces de hacer el siguiente mapeo denominado mapeo de Stolt, utilizando ahora una nueva variable llamada f_τ'

$$\sqrt{(f_0 + f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} = (f_0 + f_\tau') \tag{2.9.4}$$

Luego θ_{rfm}en (2.9.1) se puede escribir como una fase lineal en f_τ':

$$\theta_{stolt}(f_\tau', f_\eta) \approx -\left(\frac{4\pi(R_0 - R_{ref})}{c}\right)(f_0 + f_\tau')$$

(2.9.5)

Y la IFFT de (2.9.5) resultará en una perfecta compresión y registración en rango. Con el mapeo entonces se remueven todos los términos de fase superiores al término lineal.

Expansión 2:

Sirve para visualizar matemáticamente e interpretar la corrección de la migración de celdas en rango (RCMC) y la fase residual en acimut.

Se define $D(f_\eta)$, como el factor de migración en rango por:

$$D(f_\eta)=\sqrt{1 - \frac{c^2 {f_\eta}^2}{4{V_r}^2 {f_0}^2}}$$

(2.9.6)

Multiplicando y dividiendo convenientemente por ${f_0}^2$ dentro de la raíz en (2.9.2) para que aparezca $D^2_{(f_\eta)}$ y luego expandiendo con Taylor sobre f_τ hasta el tercer término obtenemos (para ver la comprobación de esta derivación consultar el Apéndice J, Sección J.2):

$$\sqrt{(f_0 + f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}}$$

$$\approx \left[f_0 D(f_\eta) + \left(\frac{f_\tau}{D(f_\eta)}\right) - \left(\frac{{f_\tau}^2}{2\, f_0 D^3_{(f_\eta)}} \frac{c^2 {f_\eta}^2}{4{V_r}^2 {f_0}^2}\right)\right]$$

(2.9.7)

El tercer término en (2.9.7) dado por:

$$\left(\frac{{f_\tau}^2}{2\,f_0 D^3_{(f_\eta)}}\frac{c^2{f_\eta}^2}{4{V_r}^2{f_0}^2}\right) \tag{2.9.8}$$

Lo vamos a considerar nulo ya que mantenemos la suposición de que trabajamos con un ángulo de *Squint* también nulo.

Entonces podemos reescribir (2.9.7) como:

$$\sqrt{(f_0+f_\tau)^2-\frac{c^2{f_\eta}^2}{4{V_r}^2}} \approx \left[f_0 D(f_\eta)+\left(\frac{f_\tau}{D(f_\eta)}\right)\right] \tag{2.9.9}$$

Y reescribir (2.9.1) como:

$$\theta_{rfm}(f_\tau, f_\eta) \approx -\left(\frac{4\pi\left(R_0-R_{ref}\right)}{c}\right)\left[f_0 D(f_\eta)+\left(\frac{f_\tau}{D(f_\eta)}\right)\right] \tag{2.9.10}$$

A partir de (2.9.5) y (2.9.10) podemos comenzar a interpretar el significado de la nueva variable f_τ' :

$$(f_0+f_\tau') = \left[f_0 D(f_\eta)+\left(\frac{f_\tau}{D(f_\eta)}\right)\right] \tag{2.9.11}$$

Despejando f_τ' y reagrupando:

$$f_\tau' = \boxed{f_0(D(f_\eta)-1)} + \boxed{\left(\frac{f_\tau}{D(f_\eta)}\right)} \tag{2.9.12}$$

Interpretación de la RCMC (Scaling)

Tomando el segundo término de (2.9.12), reemplazando $D(f_\eta)$ y definiéndolo como $f'_{\tau 1}$ tenemos:

$$\left(\frac{f_\tau}{D(f_\eta)}\right) \approx f_\tau \left(1 + \frac{c^2 {f_\eta}^2}{8 {V_r}^2 f_0}\right) = f'_{\tau 1\, scaling} \tag{2.9.13}$$

Para interpretar la RCMC, ahora podemos reescribir θ_{rfm} con la nueva variable $f'_{\tau 1}$como:

$$\theta_{rfm}\left(f'_{\tau 1\, scaling}, f_\eta\right) \approx -\left(\frac{4\pi\left(R_0 - R_{ref}\right)}{c}\right)\left[f_0 D(f_\eta) + \ f'_{\tau 1\, scaling}\right] \tag{2.9.14}$$

Sea $\Delta\tau$ el tiempo en rango (tiempo rápido) medido desde el rango de referencia R_{ref}:

$$\Delta\tau = \frac{2\left(R_0 - R_{ref}\right)}{c} \tag{2.9.15}$$

Reemplazando en (2.9.14) nos queda:

$$\theta_{rfm}\left(f'_{\tau 1\, scaling}, f_\eta\right) \approx -2\pi\Delta\tau\left[f_0 D(f_\eta) + \ f'_{\tau 1\, scaling}\right] \tag{2.9.16}$$

La frecuencia en rango ya no varía con la posición en acimut (Figura 2.21) (cuadro f), y mediante una IFFT se puede ver como la migración de Celdas en rango ha sido corregida (cuadro g), pero aún persiste la modulación residual en acimut (cuadro h).

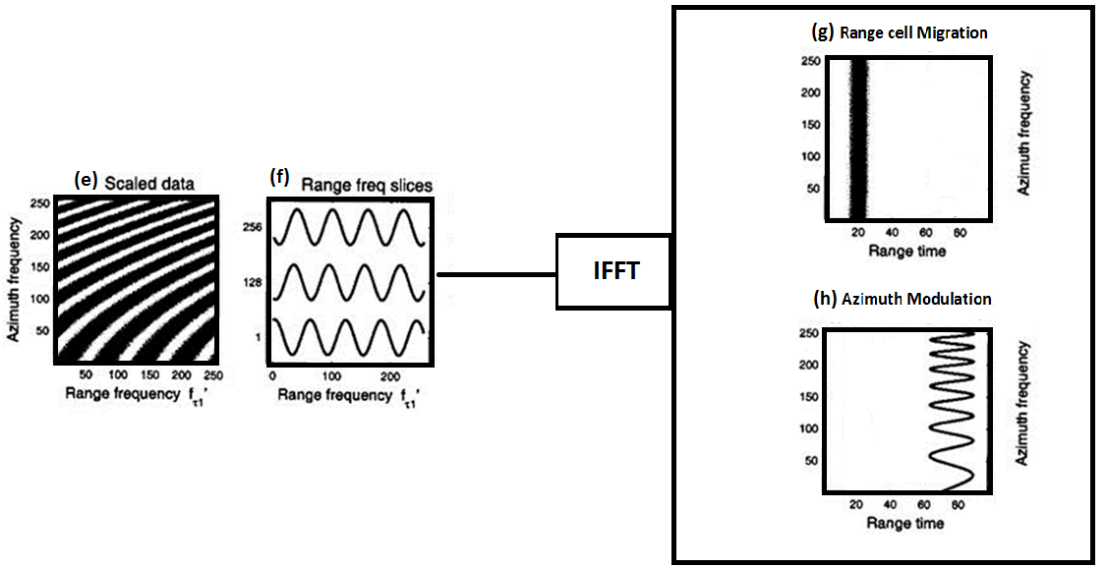


Figura 2.21, Interpretación de Stolt mediante las propiedades de la DFT. Procesamiento de un Blanco Puntual, Interpretación del Scaling

La explicación del porque se da la correcta registración del blanco puntual se fundamenta matemáticamente por la propiedad de traslación de la Transformada de Fourier (en inglés *The Fourier Transform Shift Theorem*):

Multiplicando el espectro de la señal por una exponencial compleja de signo negativo en el exponente en un tiempo $\Delta\tau$, provoca que luego de la transformada inversa en rango se observe un desplazamiento a la derecha de $\Delta\tau$ (Cumming & Wong, 2005), (Oppenheim, 1999):

$$G\left(f'_{\tau 1\ scaling}, f_\eta\right) e^{-(j\ 2\pi\ f'_{\tau 1\ scaling})\,\Delta\tau} \Leftrightarrow g\left(\tau - \Delta\tau\ , f_\eta\right) \tag{2.9.17}$$

Donde G es la transformada de Fourier de g.

Interpretación de la corrección de la modulación residual en acimut (Shifting)

Tomando el primer término de (2.9.12), reemplazando $D(f_\eta)$ tenemos:

$$f_0(D(f_\eta)-1) \approx \left(-\frac{c^2 {f_\eta}^2}{8{V_r}^2 f_0}\right) = f_{shifting} \tag{2.9.18}$$

Lo anterior se puede interpretar como un *shift* constante en rango. El *shift* remueve la componente cuadrática de la frecuencia en acimut, resultado ahora una fase que es lineal tanto en rango como en acimut.

La frecuencia en rango ya no varía con la posición en acimut (Figura 2.22) (cuadro j), mediante una IFFT se puede ver como la migración de celdas en rango ha sido corregida (cuadro k), ahora modulación residual en acimut ha sido corregida (cuadro L) y los datos se encuentran correctamente focalizados.

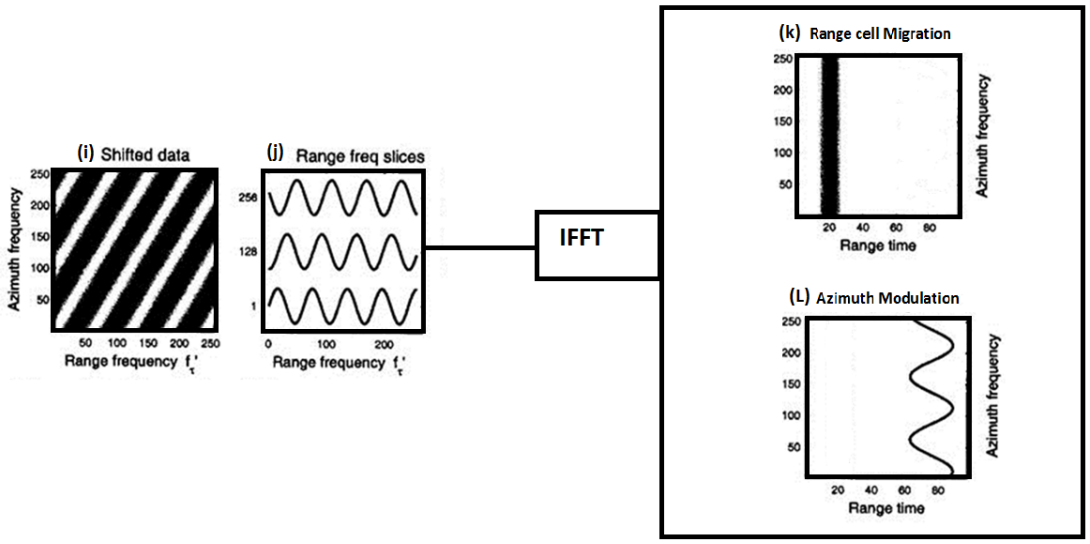


Figura 2.22, Interpretación de Stolt mediante las propiedades de la DFT. Procesamiento de un Blanco Puntual, Interpretación del Shifting

Luego a partir de (2.9.12) la variable f_τ' con que se efectúa el mapeo de Stolt nos queda expresada con un componente de scaling seguido de un shifting:

$$f_\tau' = f'_{\tau 1\, scaling} + f_{shifting} \tag{2.9.19}$$

Nuevamente la explicación del porque se corrige de esta forma la modulación residual tiene su base matemática en la propiedad de traslación de la Transformada de Fourier:

Desplazando el espectro de la señal hacia la derecha por $f'_{\tau 1\, scaling}$, provoca que luego de la transformada inversa en rango ocurra una modulación de la señal por una exponencial compleja de signo positivo en el exponente en un tiempo $\Delta\tau$ (Cumming & Wong, 2005), (Oppenheim, 1999):

$$G\left(f_\tau - f_{shifting},\, f_\eta\right) \Leftrightarrow g\left(\Delta\tau\, , f_\eta\right) e^{+(j\, 2\pi\, f_{shifting})\, \Delta\tau} \tag{2.9.20}$$

Donde G es la transformada de Fourier de g.

2.10 Interpretación geométrica del mapeo de Stolt.

En el (cuadro a) de la (Figura 2.23) se muestra el espectro de un blanco puntual, sin *Squint*, luego de la aplicación de la función RFM y antes del mapeo de Stolt, con el contorno de fase superpuesto. Se observa que los contornos de fase están ligeramente curvados, lo cual indica que el objetivo no está enfocado correctamente.

El componente cuadrático generalmente es muy pequeño en comparación con el ancho de banda del rango, pero se exagera aquí para ilustrar el efecto de compresión diferencial en acimut para un objetivo alejado del rango de referencia.

En el (cuadro b), se muestra la forma del espectro después del mapeo de Stolt (o sea, después de la interpolación de la variable de frecuencia en rango f_τ'). Los contornos de fase ahora están igualmente espaciados y se puede observar que las líneas son rectas y paralelas.

Luego si se extrae un corte vertical a lo largo del eje de las frecuencias en acimut, y un corte horizontal a lo largo del eje de frecuencias en rango, se puede observar que ahora la fase será lineal como se indica en (2.9.5).

En el (cuadro c), se muestra el blanco puntual focalizado (comprimido),

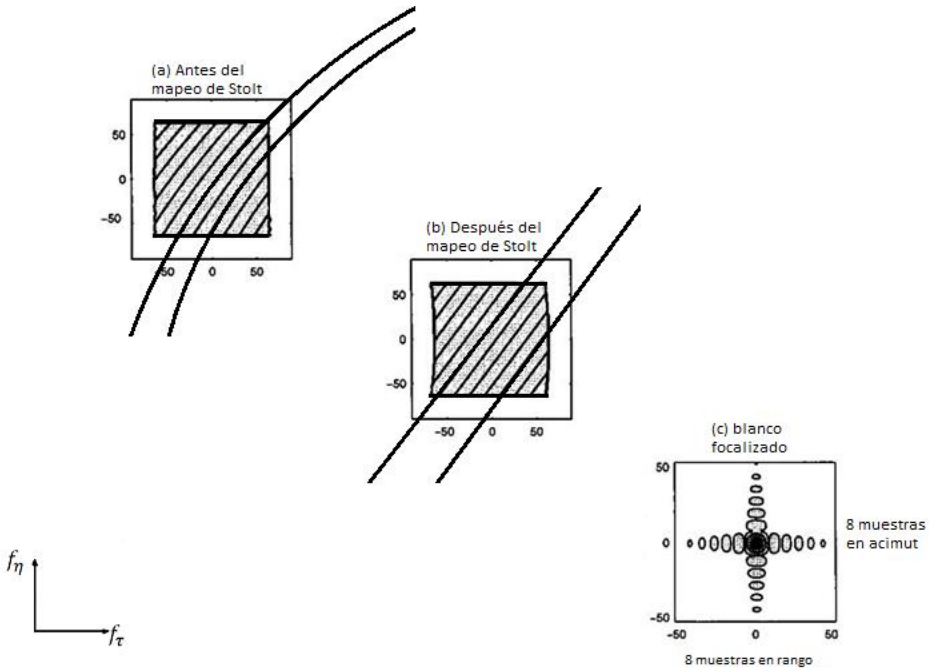


Figura 2.23, Interpretación geométrica del mapeo de Stolt

2.11Ventajas y Desventajas teóricas del algoritmo WK

Según Cumming y Wong 2005, sección 11.6.3, se pueden delinear la siguiente lista de ventajas y desventajas de la versión del WK elegida para este trabajo:

Ventajas:

Ofrece una focalización exacta para todos los ángulos de *Squint* y aperturas sintéticas, mientras se mantenga la suposición de que V(velocidad efectiva del radar) no varía con el rango.
A diferencia de otros algoritmos como *Range Doppler* o *Chirp Scaling*, permite corregir la SRC de manera exacta.
Existe una versión aproximada del algoritmo WK, la cual evita realizar la operación de interpolación, reemplazándola por una multiplicación de fase en el dominio *range-doppler*. Esta aproximación resulta eficaz cuando la RCMC (corrección de la migración de celdas en rango) y la SRC (compresión secundaria en rango) son despreciables.

Desventajas:

Se necesita una operación de interpolación que es costosa de implementar.
No tiene en cuenta la variación de V(velocidad efectiva del radar) con el rango.
Debido a que el algoritmo trabaja en el dominio bidimensional de las frecuencias, no puede hacer frente a cambios rápidos del *centroide doppler*.

CAPÍTULO 3 ~ MARCO TEÓRICO: ARQUITECTURA GPU Y CUDA

Este capítulo presenta la arquitectura de GPU Kepler de Nvidia en la que se basa la GPU seleccionada para este trabajo: Tegra K1.
También se introduce a las características principales del modelo de programación CUDA.

3.1 Procesador Kepler

La arquitectura de GPU Kepler presente en Tegra K1 (2014) es idéntica a la utilizada por las GPU de alta gama de NVIDIA. Tegra K1 es una GPU que apareció en el mercado en el año 2014, convirtiéndose rápidamente en líder de la industria móvil, pensada especialmente para aplicaciones móviles, incluye varias optimizaciones en ahorro de energía para ofrecer un alto rendimiento.
Mientras que las GPU de alta gama para escritorio o supercomputadoras pueden incluir arriba de 2880 CUDA cores de punto flotante en simple precisión y consumen cientos de watts de potencia, La GPU Kepler en Tegra K1 consiste de 192 CUDA cores y consume menos de 2 watts.
La arquitectura Kepler está organizada en *Graphics Processing Clusters* (GPC), *Streaming Multiprocessors* (SMX) y controladores de memoria. Por ejemplo la GPU GeForce GTX 680, también basada en la arquitectura Kepler consiste de 4 GPCs, 8 SMX y 4 controladores de memoria, mientras que la GPU Kepler en Tegra K1 consiste de 1 GPC, 1 SMX y 1 interfaz de memoria.

La arquitectura Kepler en el Tegra K1 también soporta la especificación OpenGL 4.4 y NVIDIA CUDA 6.

El procesador SMX de Kepler consiste de

- 192 CUDA Cores de simple precisión (6 warps)
- 64 cores de procesamiento de doble precisión (2 warps)
- 32 unidades de Load/Store (1 warp)
- 32 unidades de procesamiento de funciones especiales (1 warp)
- 16 unidades de procesamiento de texturas
- 65536 registros de 32 bits
- Warps de 32 hilos
- 4 Planificadores de Warps, cada uno puede despachar
  hasta 2 instrucciones independientes por ciclo.

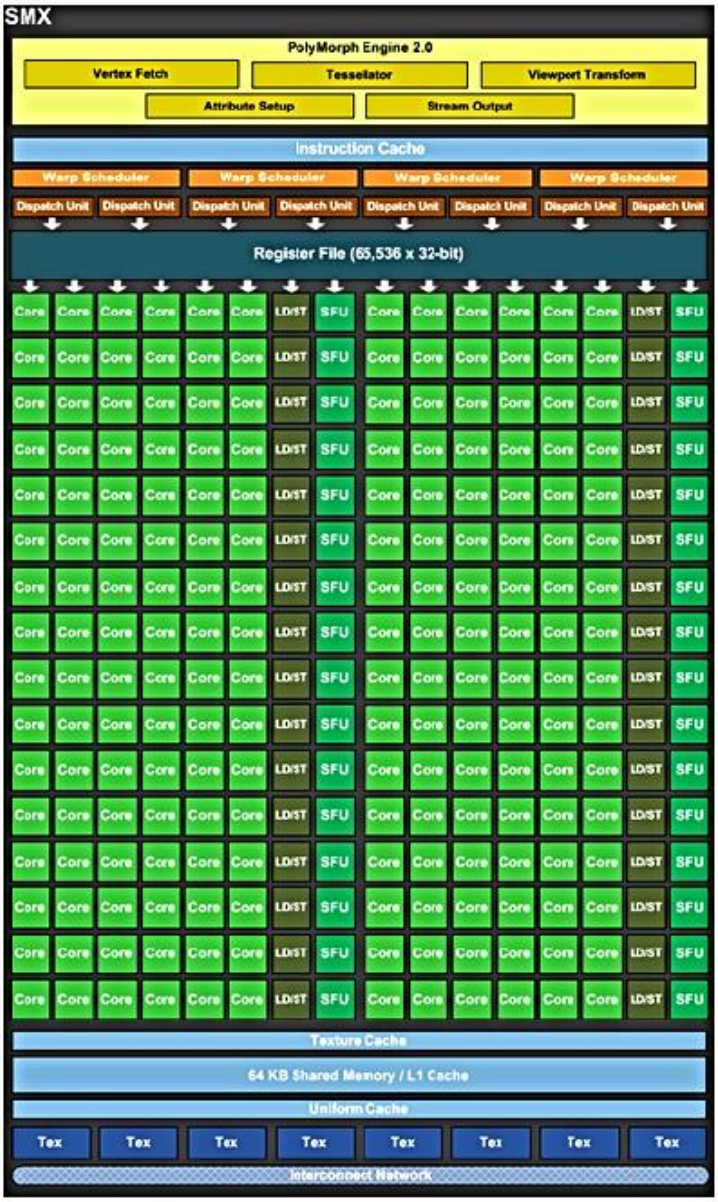


Figura 3.1, Arquitectura del procesador GPU Kepler de Nvidia (Tomado de: Kepler GK110 architecture whitepaper, 2012 NVIDIA Corporation)

Unas de las novedosas características que fueron incluidas en esta arquitectura, que vale la pena resaltar fueron:

- La nueva instrucción *SHFL*, la cual permite el intercambio de datos directamente entre hilos de un *Warp* sin necesitar utilizar la *Shared Memory.*
- La nueva cache es flexible, no requiere que los accesos estén alineados.
- Planificador de hilos *Hiper-Q*: en Kepler, pueden ejecutarse simultáneamente hasta 32 *kernels* procedentes de: varios procesos de MPI, hilos de CPU o streams de CUDA.
- El paralelismo dinámico, mediante el cual un kernel puede lanzar a otro *kernel*.

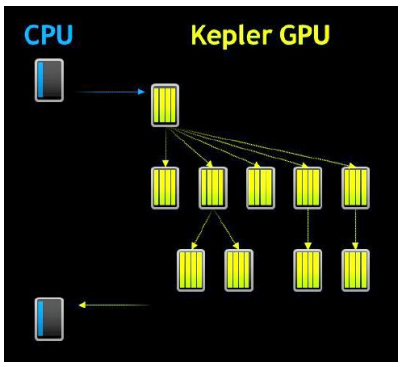


Figura 3.2, Esquema de Paralelismo Dinámico en Kepler (Tomado de: NVIDIA Kepler GK110 architecture whitepaper, 2012 NVIDIA Corporation)

3.2 Jerarquía de memoria en Kepler:

- Cache L1/L2 unificadas
- 64Kb configurable entre Shared Memory la caché L1.
- 48 Kbytes extra para expandir el tamaño de la caché L1.
- Memoria interna (caché L2): 1.5 Mbytes.

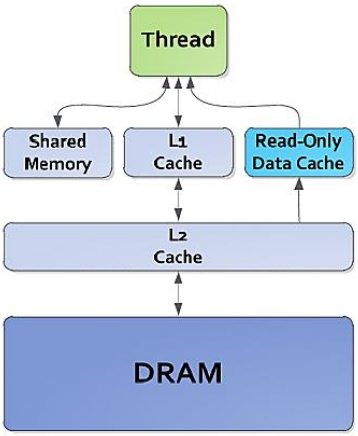


Figura 3.3, Jerarquía de memorias en en Kepler (Tomado de: NVIDIA Kepler GK110 architecture whitepaper, 2012 NVIDIA Corporation)

3.3 Nvidia JetsonTegra K1 Developer Kit

La placa de desarrollo NVIDIA Jetson Tegra K1 (JTK1) es una de las aplicaciones de la arquitectura Kepler para aplicaciones móviles.

Figura 3.4, Nvidia Jetson Tegra K1

Detalle de Nvidia Jetson Tegra K1:

- Dimensiones de la placa: 127mm x 127mm
- Tegra K1 (SoC =CPU+GPU+ISP en un solo chip tiene consumo típico entre 1 y 5 Watts)
- GPU: NVIDIA Kepler "GK20a" GPU con 192 SM3.2 CUDA cores (más de 326 GFLOPS)
- CPU: NVIDIA "4-Plus-1" 2.32GHz ARM quad-core Cortex-A15 CPU (implementa la arquitectura ARMv7-A)
- DRAM: 2GB DDR3L 933MHz EMC x16 de 64-bit
- Almacenamiento: 16GB fast eMMC 4.51 – Memoria flash
- Tarjeta SD/MMC
- Conector SATA
- Puerto USB 3.0
- Puerto Micro USB 2.0
- Puerto HDMI
- RS232: puerto serie DB9
- Ethernet: RTL8111GS Realtek 10/100/1000Base-T Gigabit
- UART
- HSIC
- 3 puertos I2C
- 7 pines GPIO pins (1.8V)
- Poder: a 12V DC
- Ventilador: 12V
- Cuenta además con puertos para cámaras, LCD y TouchScreen
- Principales APIs soportadas: CUDA 6.0 , OpenGL 4.4, OpenGL ES 3.1, OpenCV4Tegra, VisionWorks

Para ver más características de la placa consultar el (Apéndice A).

En la siguiente imagen se puede apreciar una visión global del Tegra K1. Se puede observar la GPU con su único procesador SMX y los 4 cores de CPU ARM-15.

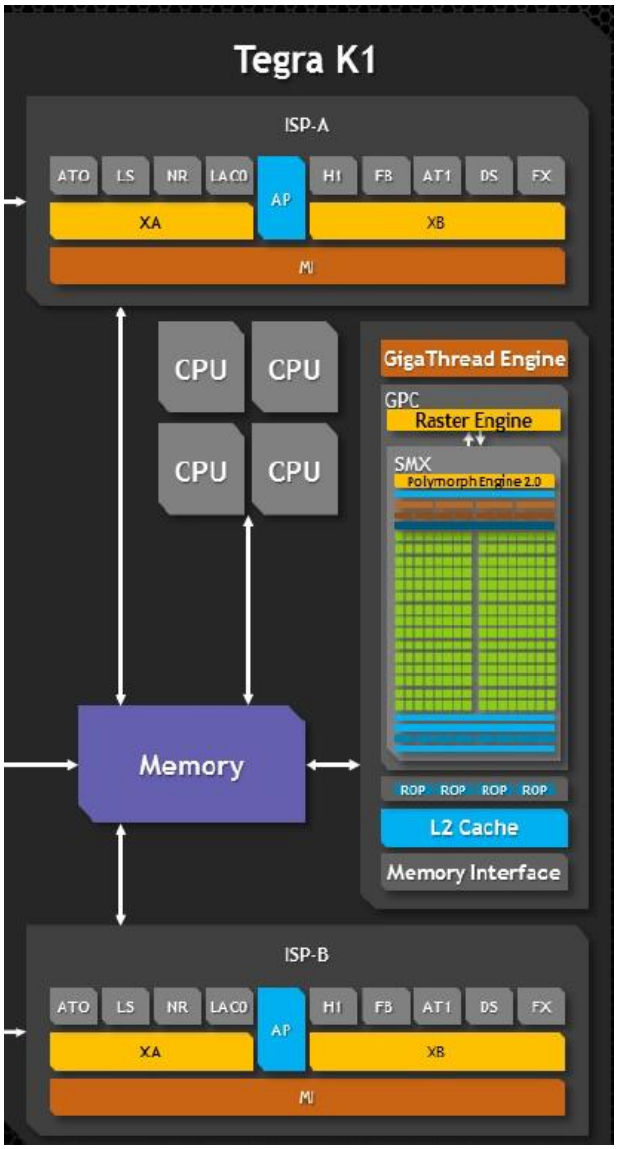


Figura 3.5, Arquitectura del Procesador Tegra K1, (Tomado de: Tegra K1 whitepaper A New Era in Mobile Computing, 2013, NVIDIA Corporation)

3.4 Modelo de programación CUDA

El modelo de programación CUDA consta, de los siguientes elementos fundamentales:

Hilos, que se agrupan en Bloques y estos en una Grilla. Las funciones en CUDA denominadas *Kernels*, antes de su lanzamiento se deben configurar la cantidad de bloques por grilla y la cantidad de hilos por bloque como parámetros adicionales de la función utilizando la sintaxis:

```
__device__ kernel_nombre<<dimGrid,dimBlock>>(param_1,…, param_n)
```

La elección de la cantidad de hilos y bloques queda determinada por un mapeo lógico que realiza el programador de la aplicación para resolver un problema particular.

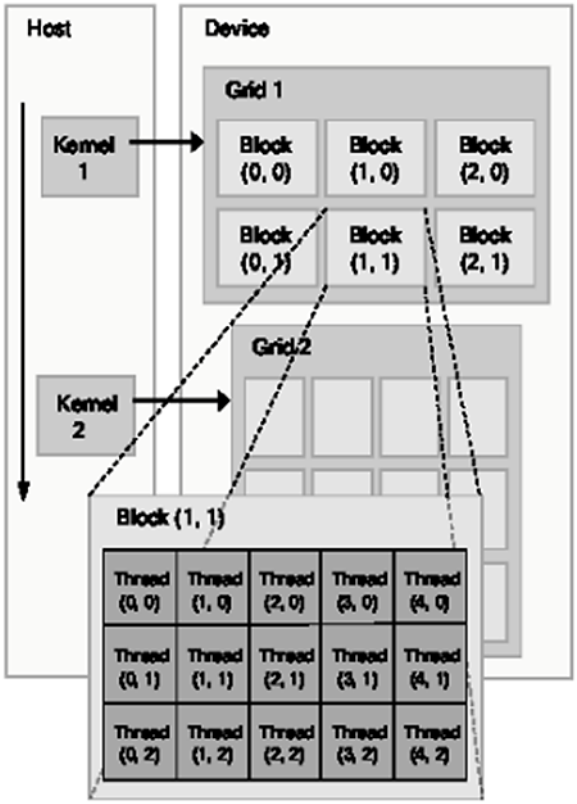


Figura 3.6, adaptado de (Kirk &Hwu, 2010) y (Cheng& Grossman, 2014). Modelo lógico de Programación CUDA. Muestra como dos *kernels* diferentes entran en ejecución en el *Device* (GPU). Cada *Kernel* es ejecutado por unidades lógicas, hilos y Bloques en una Grilla.

Por ejemplo dada una matriz numérica para ser procesada mediando algún algoritmo que trabaje en forma simultánea sobre las filas y columnas, mapeos lógicos comunes pueden ser: a) que cada hilo procese un único elemento (i, j) de la matriz, b) que cada hilo se encargue de una fila “ó” columna completa, c) que cada hilo tome una fila “y” una columna completa, etc.

En el modelo de CUDA hay distintos tipos de memorias que se pueden utilizar:
Principalmente hay que destacar a la memoria global, la cual tiene mayor tamaño pero es también la que posee mayor latencia en el acceso. La memoria global se utiliza para transferir datos entre la CPU y GPU en ambos sentidos. Luego podemos encontrar las memorias de registros y las memorias *shared* de reducido tamaño pero de alta velocidad comparadas con la memoria global.

Se requiere una reserva de memoria global en GPU y luego por lo general una transferencia de datos inicial de CPU a GPU para poder comenzar la ejecución de un *kernel* en la GPU. Una vez finalizada la función se realiza el proceso inverso transfiriendo los datos de GPU a CPU, para continuar trabajando en CPU con los resultados.

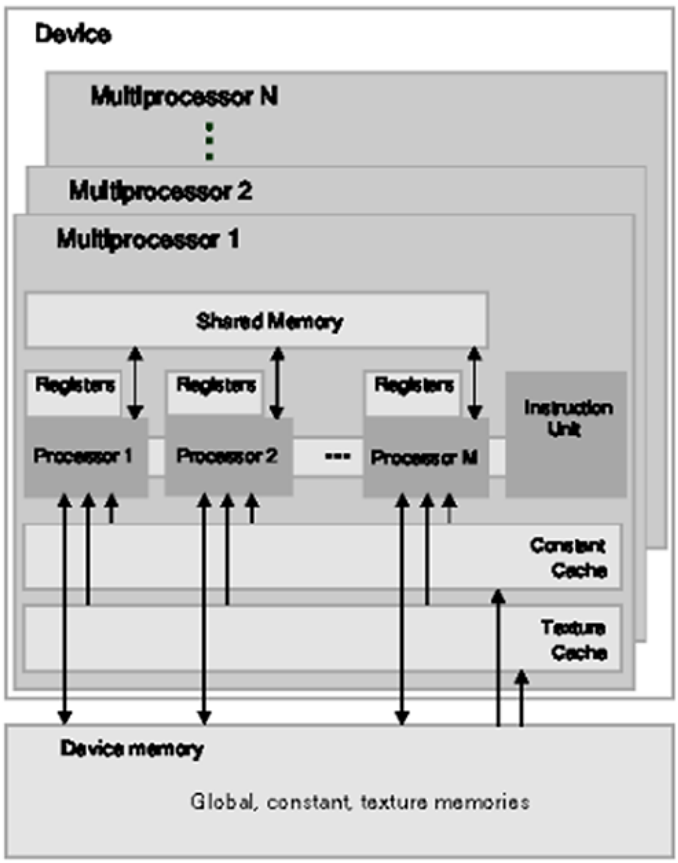


Figura 3.7, adaptado de (Kirk &Hwu, 2010) y (Cheng& Grossman, 2014). Modelo físico del Hardware CUDA. Un Multiprocesador consiste de múltiples *Scalar Processor, Cores*, una unidad de instrucciones *multithread*, una memoria *shared on-chip*. El multiprocesador crea, administra y ejecuta hilos concurrentes con cero sobrecarga (overhead) en la planificación (scheduling). (CUDA Toolkit Documentation, 2018)

La mayoría de las aplicaciones CUDA que buscan optimización en el procesamiento paralelo plantean distintas estrategias de particionamiento de los datos para poder utilizar la capacidad reducida de las memorias *shared* para aprovechar su baja latencia de acceso.

Las memorias de registros son propias de cada hilo mientras que las memorias *shared* son visibles por todos los hilos que pertenecen a un Bloque.

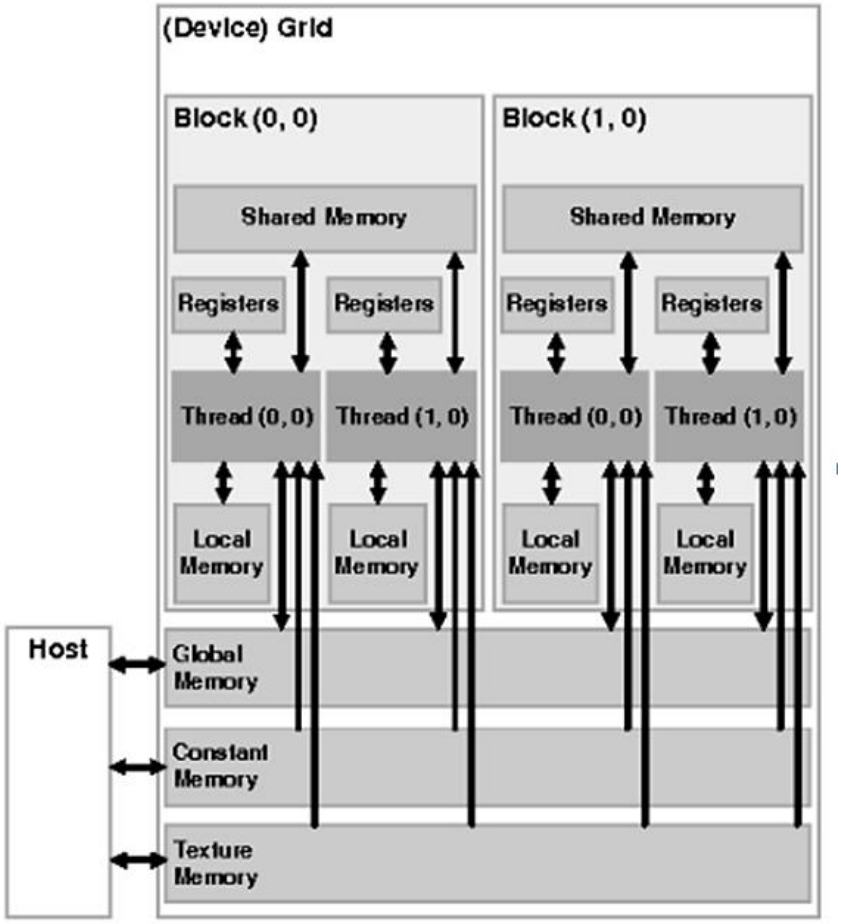


Figura 3.8, adaptado de (Kirk &Hwu, 2010) y (Cheng& Grossman, 2014). Modelo de Memorias en CUDA, en un esquema Lógico (Hilos, Bloque y Grilla) y Físico (acceso de las diferentes tipos de memoria a partir de los componentes lógicos de CUDA).

CAPÍTULO 4 ~ VALIDACIÓN

Este capítulo está dedicado a los distintos criterios de validación utilizados en el trabajo.

Se propone un criterio para comparar la resolución espacial obtenida por el sistema respecto de los valores teóricos esperados.

Dado que a partir de las tres implementaciones del WK: MATLAB, C y CUDA idealmente se espera obtener iguales resultados frente a las mismas entradas, se especifica aquí un criterio para decidir cuando dos versiones pueden considerarse equivalentes entre sí.

Finalmente se presenta el criterio de la PLSR (*Peak Side-lobe Ratio*) sobre la calidad de focalización obtenida.

4.1 Resolución espacial: Criterio de Validación

Para validar la salida del WK se compara la resolución espacial medida contra los valores teóricos presentados anteriormente en la sección 2.2, bajo el siguiente procedimiento:

Dado S4, la señal focalizada por WK:

1. Se convierte la señal a valores de potencia, mediante la expresión:

$$20\log_{10}\big(abs(S4)\big) \tag{4.1.1}$$

2. Se busca el máximo de la señal.

3. Tanto para la dimensión de rango como para acimut, hacer:

    a. Se calcula el corte o caída a -3dB a partir del máximo.
    b. Se cuenta el número de muestras por encima del corte.
    c. Se calcula la resolución espacial teórica de acuerdo a la sección 2.2.
    d. Se calcula el *pixel spacing* de acuerdo a la sección 2.3.
    e. Se obtiene la resolución espacial de WK ("resolución espacial medida") multiplicando el *pixel spacing* por la cantidad de muestras encontradas en (b).
    f. Se obtienen los órdenes de magnitud de (c) y (e).
    g. Se evalúa la condición de éxito del criterio: se requiere que los valores de resolución espacial medidos en (e) pertenezcan al orden de magnitud de los valores de resolución teóricos en (c).

Nota sobre el orden de magnitud:

Sean $\mathcal{E}_1$ y $\mathcal{E}_2$ dos números expresados en notación científica,

$$\mathcal{E}_1 = a_x 10^m$$
$$\mathcal{E}_2 = b_x 10^n \quad (4.1.2)$$

Donde $1 \leq a < 10$ y $1 \leq b < 10$ y n, m valores enteros.
Calculamos:

$$Od\acute{n} = \text{abs (m-n)} \quad (4.1.3)$$

Si $Od\acute{n}$ = 0, decimos que $\mathcal{E}_1$ y $\mathcal{E}_2$ están dentro del mismo orden de magnitud.

Si $Od\acute{n}$ > 0, decimos que $\mathcal{E}_1$ y $\mathcal{E}_2$ no estan dentro del mismo orden de magnitud y que existen $Od\acute{n}$ ordenes de magnitud entre ellos.

4.2 Calidad de focalización: Criterio de validación

Para validar la calidad de focalización del algoritmo se utilizó un criterio basado en la PSLR (*Peak Side-Lobe Ratio*): relación entre el lóbulo principal y los lóbulos secundarios del blanco puntual. Utilizando la señal focalizada S4 a la salida del WK, se realiza el siguiente procedimiento:

Se convierte la señal a valores de potencia utilizando (4.1.1)
Y tanto para acimut como para rango:

Se selecciona el mayor lóbulo secundario y se toma el punto máximo: maxLobSec,
Se selecciona el lóbulo principal y se toma el punto máximo: maxLobPrinc,
Se espera que la diferencia sea mayor o igual a 13dB (Cumming y Wong, 2005):

$$abs\ (LobuloPrincipal_{max} - LobuloSecundario_{\max}) \geq 13\ dB \quad (4.2.1)$$

4.3 Igualdad entre versiones de WK: Criterio de validación

MATLAB vs C/CUDA:

Sea $S4_m$ la matriz de datos focalizados con Matlab y sea $S4_v$ la matriz con datos focalizados a la salida de la versión C ó CUDA. Para cuantificar el grado de similitud de estas matrices, se utiliza el siguiente criterio basado en el error relativo:

$$\mathrm{E} = \left(\frac{abs(S4_m - S4_v)}{abs(S4_m)}\right) \tag{4.3.1}$$

$$e_{max} = \mathrm{MAX(E)} \approx 10^{-5} \tag{4.3.2}$$

Elemento a elemento se realiza la diferencia de estas matrices y se divide respecto de la matriz de referencia $S4_m$. Dado que los datos son complejos se realiza el cociente de los valores absolutos.

A continuación se obtiene el elemento máximo de la matriz resultante E, denominado e_{max}.

Luego para afirmar que dos matrices son equivalentes, consideraremos aceptable que la máxima diferencia e_{max} entre las matrices sea del orden de 10^{-5}.

CAPÍTULO 5~ DISEÑO DEL SOFTWARE

En este capítulo se presenta la lista de requerimientos del sistema desde en un alto nivel conceptual basado en los objetivos e hipótesis planteadas para luego obtener una derivación de requerimientos cada vez con mayor detalle. También se presentan las fases de Análisis, Diseño e implementación del algoritmo WK.

5.1 Listado de los requerimientos del Software

La siguiente tabla, representa el resultado de la etapa de relevamiento de requerimientos,
Y su clasificación en tres niveles: L0, L1, L2 (L2A y L2B), y sus relaciones padre-hijo.

Los requerimientos L0 y L1 son de alto nivel conceptual y están relacionados con los objetivos e hipótesis de la tesis. Los requerimientos L2A están centrados en el usuario y los requerimientos L2B están centrados mayormente en el sistema.

L0	Se deberá desarrollar un producto de software con el objeto de demostrar las ventajas del uso de las técnicas de computación paralela en la resolución de algoritmos de procesamiento de datos provenientes de Radares de Apertura Sintética (SAR)
L1.1	El producto software deberá implementar el algoritmo WK
L2B.1	La implementación del algoritmo WK deberá satisfacer el criterio de calidad denominado PLSR, el cual requiere que la distancia entre el pico principal tanto en rango como acimut a su lóbulo secundario más alto sea >= 13 dB
L2B.2	La diferencia en el resultado de la matriz de datos focalizados por las 3 versiones del algoritmo WK: MATLAB, C y CUDA no deben diferir en más de 10e-5
L2B.3	El algoritmo WK deberá ser capaz de computar números complejos en doble precisión
L2B.4	La implementación final C/CUDA deberá leer como entrada un archivo de parámetros y los datos crudos en formato *.TXT CSV y dar una salida focalizada en formato *.TXT CSV

Figura 5.1 Listado de requerimientos del software, Niveles L0, L1 globales, L2A de usuario, L2B de sistema, parte 1 de 3

L1.2	El producto de software deberá ser configurable		
		L2A.1	El ingreso de datos deberá realizarse mediante un archivo de parámetros modificable por el usuario donde poder configurar las características físicas del sistema SAR
L1.3	El producto de software deberá ser implementado en una arquitectura paralela basada en GPU y una arquitectura secuencial basada en CPU		
		L2A.2	El desarrollo de la versión final deberá realizarse en una arquitectura hardware paralela CPU/GPU especifica: NVIDIA Jetson Tegra K1
		L2A.3	Se deberá implementar una versión final del algoritmo WK Paralela en lenguaje CUDA para GPU
		L2A.4	Para la misma plataforma hardware seleccionanda. Se deberá implementar una versión final del algoritmo WK secuencial en Lenguaje C para CPU, para comparar el rendimiento con la versión paralela
L1.4	El producto de software deberá ser utilizado para evaluar un set de datos crudos de referencia cuya resolución espacial pueda ser contrastada con referencias bibliográficas		
		L2A.5	El algoritmo WK deberá focalizar los datos crudos correspondientes a un blanco puntual, en una escena ideal de tierra plana y Ángulo de squint nulo
		L2B.5	Se deberá implementar en MATLAB un simulador de datos crudos SAR para un Blanco puntual, cuya salida se pueda obtener en 2 formatos *.MAT y *.TXT CSV
		L2B.6	La resolución espacial en rango y acimut entregada por el algoritmo WK deberá satisfacer el orden de magnitud expresado la teoría SAR de acuerdo a las configuraciones en los parámetros del sistema

Figura 5.2 Listado de requerimientos del software, Niveles L1 global, L2A de usuario, L2B de sistema, parte 2 de 3.

L1.5	El producto de software deberá poder medir diferencias en rendimiento en tiempo debidas a las diferentes arquitecturas		
		L2B.7	Función por función, se requiere una aceleración de al menos 2x entre la versión secuencial y paralela de WK
L1.6	La versión inicial deberá ser implementada utilizando lenguajes y formatos tradicionales en el campo del procesamiento de señales y comunicaciones		
		L2A.6	La implementación inicial del algoritmo WK deberá ser realizada en MATLAB para una PC estándar, para facilitar la comunicación con los usuarios expertos en el dominio del problema, obtener una comprensión completa de los detalles del algoritmo y lograr la aprobación para la construcción de las versiones finales.
		L2A.7	El producto de software deberá emitir una salida grafica de la señal focalizada
		L2B.8	La implementación inicial en MATLAB deberá leer como entrada un archivo de parámetros y los datos crudos en formato *.MAT y dar una salida focalizada en formato *.TXT CSV

Figura 5.3 Listado de requerimientos del software, Niveles L1 global, L2A de usuario, L2B de sistema, parte 3.

5.2 Especificación de WK: Análisis, Diseño e Implementación del Software

En la presente sección se especifica el simulador y algoritmo WK, transitando por la etapas del análisis, diseño, y resaltando las partes esenciales de la implementación del software. Se invita al lector a consultar los apéndices correspondientes para acceder a las implementación completa del sistema.

Durante el análisis del software se especifica el "Qué" debe hacer el sistema en un nivel de abstracción elevado independizándose de cómo podría ser realizado.

Durante la etapa de Diseño en cambio, se especifica el "Cómo" el sistema realizará las operación descriptas durante el análisis. Aquí el nivel de abstracción disminuye pero no se hace ninguna referencia a tecnologías de implementación.

La etapa de Implementación es el nivel de cero abstracción, aquí se decide por una tecnología hardware/software concreta para construir la funcionalidad especificada anteriormente.

5.2.1 Especificación de WK: Análisis del Sistema

Id: E1.A	Simular la señal SAR correspondiente a un objetivo puntual	
Análisis:	**Precondición**:	
		Existe el archivo de parámetros
	Curso de acción:	
		1. Acceder a los parámetros del sistema 2. Obtener una salida para la función de respuesta al impulso ($h_{impulso}$): $h_{impulso}(f_\tau, f_\eta) = w_\tau\left(f_\tau - 2\frac{R(f_\eta)}{C}\right) * w_\eta\left(f_\eta - f_{\eta_c}\right) * e^{-j4\pi f_0\left(\frac{R(f_\eta)}{C}\right)} * e^{j\pi K_\tau\left(f_\tau - 2\frac{R(f_\eta)}{C}\right)^2}$ 3. Generación de datos crudos con dimensión 2D en el dominio del Tiempo.
	Postcondición:	
		Se crea el archivo *datos crudos* con dimensión 2D en el dominio del Tiempo.
	Requerimiento:	
		L1.1
	Relaciones de uso:	
		N/A

Id: E2.A	Convertir los datos crudos SAR al dominio bidimensional de las frecuencias.	
Análisis:	**Precondición**:	
		• Existe archivo de datos crudos, con dimensión 2D en el dominio del Tiempo.
	Curso de acción:	
		1. Leer archivo de datos crudos 2D en el dominio del Tiempo y denominarlo S 2. Calcular la *Transformada Discreta de Fourier* sobre S y denominarlo S1
	Postcondición:	
		• Existe S1, los datos crudos han sido transformados al dominio 2D de las frecuencias
	Requerimiento:	
		L1.1
	Relaciones de uso:	
		E1.A

Id: E3.A	Focalización aproximada en Rango para el Rango de Referencia Seleccionado	
Análisis:	**Precondición**:	
		Existe S1, datos crudos en el dominio 2D de las frecuencias
	Curso de acción:	
		1. Leer datos crudos en el dominio de las frecuencias 2D, llamarlos S1. 2. Leer el rango de referencia seleccionado. 3. Crear RFM, matriz de referencia en rango, (θ_{ref}): $\theta_{ref}(f_\tau, f_\eta) = +\left(\frac{4\pi R_{ref}}{c}\right)\sqrt{(f_0+f_\tau)^2 - \frac{c^2 {f_\eta}^2}{4{V_{Rref}}^2} + \frac{\pi {f_\tau}^2}{K_r}}$ 4. Multiplicar la Señal S1 con la función de referencia RFM, para efectuar la focalización gruesa, denominarlo S2
	Postcondición:	
		Los datos de la señal SAR han sido focalizados en el rango de referencia, se crea S2 en el dominio 2D de las frecuencias.
	Requerimiento:	
		L1.1
	Relaciones de uso:	
		E2.A

Id: E4.A	Mapeo de Stolt	
Análisis:	**Precondición**:	
		• Existen vectores de tiempo rápido y tiempo lento
	Curso de acción:	
		1. Leer vectores de tiempo rápido y tiempo lento. 2. Calcular la nueva variable en rango $\boldsymbol{f'_\tau}$ Y denominarla DeltaFast: $$\boldsymbol{f'_\tau} = \sqrt{(f_0 + \boldsymbol{f_\tau})^2 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}} - f_0$$
	Postcondición:	
		• Se crea DeltaFast (nueva variable matricial $\boldsymbol{f'_\tau}$ en rango, esto es en tiempo rápido). • DeltaFast no está equiespaciada.
	Requerimiento:	
		L1.1
	Relaciones de uso:	
		N/A

Id: E5.A	Interpolación de Stolt	
Análisis:	**Precondición**:	
		• Existe DeltaFast nueva variable en rango mapeada, y no esta equiespaciada • Existe S2, señal focalizada al rango de referencia seleccionado.
	Curso de acción:	
		1. Calcular los límites matriciales para la interpolación. 2. Interpolar la nueva variable DeltaFast no equiespaciada, utilizando S2, en una nueva grilla equiespaciada y denominarla S3.
	Postcondición:	
		• Se crea S3, señal focalizada diferencialmente. Y se encuentra en el dominio 2D de las frecuencias.
	Requerimiento:	
		L1.1
	Relaciones de uso:	
		E3.A, E4.A

Id: E6.A	Convertir los datos focalizados diferencialmente al dominio 2D del tiempo	
Análisis:	**Precondición**:	
		• Existe S3, datos focalizados diferencialmente.
	Curso de acción:	
		1. Leer S3, datos focalizados diferencialmente. 2. Calcular la *Transformada Discreta de Fourier Inversa* sobre S3 y denominarla S4 3. Guardar S4 en almacenamiento permanente.
	Postcondición:	
		• Existe S4: Los datos focalizados diferencialmente han sido transformados al dominio 2D del tiempo • S4 se encuentra en almacenamiento permanente.
	Requerimiento:	
		L1.1
	Relaciones de uso:	
		E5.A

5.2.2 Especificación de WK: Diseño del Sistema

Id: E1.D	Simular la señal SAR correspondiente a un objetivo puntual	
Diseño:	**Precondición**:	
		• Existe el archivo de parámetros
	Curso de acción:	
		1. Lectura de parámetros. 2. Cómputo de los vectores Temporales para rango y acimut 3. Cómputo de la función del Rango. 4. Cálculo de las ventanas en rango y acimut. 5. Se aplica la función de impulso SAR a las posiciones en rango y acimut. 6. Se aplican las ventanas en rango y acimut. 7. Se generan los datos crudos con dimensión 2D en el dominio del Tiempo.
	Postcondición:	
		• Se genera archivo *datos crudos* con dimensión 2D en el dominio del Tiempo.
	Requerimiento:	
		L1.1
	Análisis:	
		E1.A
	Relaciones:	
		N/A

Id: E2.D	Convertir los datos crudos SAR al dominio bidimensional de las frecuencias.	
Diseño:	**Precondición**:	
		Existe archivo de datos crudos, con dimensión 2D en el dominio del Tiempo.
	Curso de acción:	
		1. Leer archivo de datos crudos, y se lo denomina S. 2. Operación iFFT-SHIFT sobre S para llevar el origen de coordenadas de los datos crudos a la posición (1,1). 3. Calcular la Transformada Discreta de Fourier sobre S mediante el algoritmo *FFT: Fast Fourier Transform.* FFT supone el origen de coordenadas en la posición (1,1). 4. Operación FFTSHIFT sobre S para llevar el origen nuevamente al centro de la matriz. 5. Denominar S1 a la matriz de datos crudos en el dominio 2D de las Frecuencias. 6. Eliminar S.
	Postcondición:	
		• Existe S1: los datos crudos, transformados al dominio 2D de las frecuencias. • S ha sido eliminada. • El archivo de datos crudos en el dominio del tiempo continúa existiendo.
	Requerimiento:	
		L1.1
	Trazabilidad con Análisis:	
		E2.A
	Relaciones de uso:	
		E1.D

Id: E3.D	Focalización aproximada en Rango para el rango de referencia seleccionado.	
Diseño:	**Precondición**:	
		Existe S1, datos crudos en el dominio 2D de las frecuencias
	Curso de acción:	
		1. Leer datos crudos en el dominio de las frecuencias 2D, llamarlos S1. 2. Leer el rango de referencia seleccionado. 3. Formación del vector de frecuencias en tiempo rápido 4. Formación del vector de frecuencias en tiempo lento 5. Formación de la matriz de frecuencias en tiempo rápido, replicando el vector de frecuencias en tiempo lento 6. Formación de la matriz de frecuencias en tiempo lento, replicando el vector de frecuencias en tiempo rápido. 7. Creación de la RFM como combinación lineal de las matrices de frecuencias en tiempo rápido y tiempo lento bajo una raíz cuadrada 8. Adicionar al resultado del paso 6 una matriz de frecuencias en tiempo rápido. 9. Aplicando la fórmula de Euler, convertir RFM al Exponencial complejo. 10. Multiplicar RFM con S1 elemento a elemento para obtener la señal focalizada al rango de referencia seleccionado, denominarlo S2. 11. Eliminar S1.
	Postcondición:	
		• Los datos de la señal SAR han sido focalizados en el rango de referencia, se crea S2 en el dominio 2D de las frecuencias. • S1 ha sido eliminado.
	Requerimiento:	
		L1.1
	Trazabilidad con Análisis:	
		E3.A
	Relaciones de uso:	
		E2.D

Id: E4.D	Mapeo de Stolt.	
Diseño:	**Precondición**:	
		• Existen vectores de tiempo rápido y tiempo lento
	Curso de acción:	
		1. Leer vectores de tiempo rápido (rango) y tiempo lento (acimut). 2. Formar grillas matriciales de tiempo rápido y tiempo lento replicando los vectores de tiempo rápido, tiempo lento. 3. Formar grilla de frecuencia de la portadora. 4. Evaluar la raíz cuadrada de la combinación lineal de las grillas matriciales de 2 y 3. 5. Obtener grilla matricial de la nueva variable mapeada.
	Postcondición:	
		• Se crea DeltaFast (nueva variable matricial mapeada, $\boldsymbol{f}'_{\boldsymbol{\tau}}$ en rango, esto es en tiempo rápido) • DeltaFast no está equiespaciada.
	Requerimiento:	
		L1.1
	Trazabilidad con Análisis:	
		E4.A
	Relaciones de uso:	
		N/A

Id: E5.D	Interpolación de Stolt.	
Diseño:	**Precondición**:	
		• Existe DeltaFast nueva variable en rango mapeada, y no esta equiespaciada • Existe S2, señal focalizada al rango de referencia seleccionado.
	Curso de acción:	
		1. Calcular un vector kernel de 8 posiciones enteras. Con valores en orden creciente, expresado matemáticamente como $(-4, 4]$, con incremento igual a 1. $\text{Kernel}=[-3, -2, -1, 0, 1, 2, 3, 4]$ 2. Calcular el máximo de la nueva variable matricial mapeada DeltaFast. 3. Calcular el mínimo de la nueva variable matricial mapeada DeltaFast. 4. Calcular el límite superior e inferior en columnas para la nueva grilla equiespaciada. 5. Crear e inicializar la nueva grilla equiespaciada y se denominarla S3. 6. Para cada fila de S2 y por cada elemento: a. Extraer su parte entera. b. Extraer su parte decimal. c. Utilizando el kernel, obtener 8 desplazamientos enteros centrados en la parte entera del elemento actual. d. Con los desplazamientos enteros extraer valores de DeltaFast. Denominarlo Resultado1 e. Utilizando el kernel, obtener 8 desplazamientos decimales centrados en la parte decimal del elemento actual. f. Computar la función $\text{Sinc}(.)$ para cada uno de los desplazamientos decimales. Denominarlo Resultado2 g. Multiplicar Resultado1 con Resultado2 para obtener un nuevo elemento interpolado en S3 7. Eliminar DeltaFast. 8. Eliminar S2.
	Postcondición:	
		• Se crea S3, señal focalizada diferencialmente. Y se encuentra en el dominio 2D de las frecuencias. • DeltaFast ha sido eliminada. • S2 ha sido eliminada.
	Requerimiento:	
		L1.1
	Trazabilidad con Análisis:	
		E5.A
	Relaciones de uso:	
		E3.D, E4.D

Id: E6.D	Convertir los datos focalizados diferencialmente al dominio 2D del tiempo	
Diseño:	**Precondición**:	
		Existe S3, datos focalizados diferencialmente.
	Curso de acción:	
		1. Leer S3, datos focalizados diferencialmente. 2. Operación iFFT-SHIFT sobre S3 para llevar el origen de coordenadas de los datos crudos a la posición (1,1). 3. Calcular la Transformada Discreta de Fourier Inversa sobre S3 mediante el algoritmo *IFFT: Inverse Fast Fourier Transform. IFFT* supone el origen de coordenadas en la posición (1,1). 4. Operación FFTSHIFT sobre S3 para llevar el origen nuevamente al centro de la matriz y lo denomina S4. 5. Escribe S4 en almacenamiento permanente.
	Postcondición:	
		• Existe S4: Los datos focalizados diferencialmente han sido transformados al dominio 2D del tiempo. • S4 se encuentra en almacenamiento permanente. • Se elimina S3.
	Requerimiento:	
		L1.1
	Trazabilidad con Análisis:	
		E6.A
	Relaciones de uso:	
		E5.D

5.2.3 Especificación de WK: implementación en C

<table>
<tr><td>Id:
E1.i</td><td colspan="5">Implementación FFTSHIFT</td></tr>
<tr><td rowspan="15"></td><td>Lenguaje: C</td><td colspan="4">Componente: WK_FUNC</td></tr>
<tr><td>Firma:</td><td colspan="4">void fftshift(double complex *d_matrix,int filas, int cols)</td></tr>
<tr><td>Descripción</td><td colspan="4">Realiza la fftshift para una matriz de datos complejos en 2D.
Para matrices cuadradas también se la utiliza como ifftshift.</td></tr>
<tr><td>Ámbito</td><td colspan="4">CPU</td></tr>
<tr><td>Trazabilidad Diseño:</td><td colspan="4">E6.D, E2.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="3"></td><td>d_matrix</td><td>double complex *</td><td colspan="2">Es una matriz cuadrada. Puede estar en el domino del tiempo o de la frecuencia.</td></tr>
<tr><td>filas</td><td>int</td><td colspan="2">Filas de d_matrix</td></tr>
<tr><td>cols</td><td>int</td><td colspan="2">Columnas de d_matrix</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>d_matrix</td><td>double complex *</td><td>Es una matriz cuadrada de filas x cols que contiene a d_matrix luego del proceso de shifting</td><td>inplace</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td>void</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Id:
E2.i</td><td colspan="5">Implementación FFT</td></tr>
<tr><td rowspan="13"></td><td>Lenguaje: C</td><td colspan="4">Componente: FFTW</td></tr>
<tr><td>Firma:</td><td colspan="4">fftw_plan plan_forward;
fftw_plan fftw_plan_dft_2d(int n0, int n1,
fftw_complex *in, fftw_complex *out, int sign,
unsigned flags);
fftw_execute (plan_forward);</td></tr>
<tr><td>Descripción</td><td colspan="4">Realiza la transformada Rápida de Fourier. Es una implementación muy eficiente de la Transformada Discreta de Fourier.</td></tr>
<tr><td>Ámbito</td><td colspan="4">CPU</td></tr>
<tr><td>Trazabilidad Diseño:</td><td colspan="4">E2.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="5"></td><td>n0</td><td>int</td><td colspan="2">Dimensión en filas de in</td></tr>
<tr><td>n1</td><td>int</td><td colspan="2">Dimensión en columnas de in</td></tr>
<tr><td>in</td><td>(fftw_complex *) equivalente a (double complex *)</td><td colspan="2">Matriz compleja de entrada en el dominio del tiempo</td></tr>
<tr><td>sign</td><td>int</td><td colspan="2">FFTW_FORWARD</td></tr>
<tr><td>flags</td><td>unsigned</td><td colspan="2">FFTW_ESTIMATE</td></tr>
<tr><td colspan="5"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>out</td><td>(fftw_complex *) equivalente a (double complex *)</td><td>Matriz compleja de salida en el dominio de las frecuencias</td><td>Outplace por defecto. Se fuerza el modo Inplace haciendo in=out</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3">Descripción</td></tr>
<tr><td></td><td>fftw_plan</td><td colspan="3">El plan de retorno debe ejecutarse con la función:
fftw_execute (plan_forward)</td></tr>
</table>

<table>
<tr><td>Id:
E3.i</td><td colspan="5">Implementación IFFT</td></tr>
<tr><td></td><td>Lenguaje: C</td><td colspan="4">Componente: FFTW</td></tr>
<tr><td></td><td>Firma:</td><td colspan="4">fftw_plan plan_ backward;
fftw_plan fftw_plan_dft_2d(int n0, int n1,
fftw_complex *in, fftw_complex *out, int sign,
unsigned flags);
fftw_execute (plan_backward);</td></tr>
<tr><td></td><td>Descripción</td><td colspan="4">Realiza la inversa de la transformada Rápida de Fourier. Es una implementación muy eficiente de la inversa de la Transformada Discreta de Fourier.</td></tr>
<tr><td></td><td>Ámbito</td><td colspan="4">CPU</td></tr>
<tr><td></td><td>Trazabilidad Diseño:</td><td colspan="4">E6.D</td></tr>
<tr><td></td><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td></td><td></td><td>n0</td><td>int</td><td colspan="2">Dimensión en filas de in</td></tr>
<tr><td></td><td></td><td>n1</td><td>int</td><td colspan="2">Dimensión en columnas de in</td></tr>
<tr><td></td><td></td><td>in</td><td>(fftw_complex *)
equivalente a
(double complex *)</td><td colspan="2">Matriz compleja de entrada en el dominio de las frecuencias</td></tr>
<tr><td></td><td></td><td>sign</td><td>int</td><td colspan="2">FFTW_BACKWARD</td></tr>
<tr><td></td><td></td><td>flags</td><td>unsigned</td><td colspan="2">FFTW_ESTIMATE</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td></td><td></td><td>out</td><td>(fftw_complex *)
equivalente a
(double complex *)</td><td>Matriz compleja de salida en el dominio del tiempo</td><td>Outplace por defecto. Se fuerza el modo Inplace haciendo in=out</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Tipo Return</td><td>Tipo</td><td colspan="3">Descripción</td></tr>
<tr><td></td><td></td><td>fftw_plan</td><td colspan="3">El plan de retorno debe ejecutarse con la función:
fftw_execute (plan_backward);</td></tr>
</table>

Id: E4.i	Implementación RFM				
	Lenguaje: C	Componente: **WK_FUNC**			
	Firma:	`void rfm (double complex *out_rfm,double*fslow, int nx,double*ffast,int ny);`			
	Descripción	Construye la matriz RFM basado en los vectores de frecuencias en tiempo lento y rápido. Esta matriz servirá luego para realizar la focalización gruesa para el rango de referencia seleccionado.			
	Ámbito	CPU			
	Trazabilidad Diseño:	E3.D			
	Param IN	Nombre	Tipo	Descripción	
		`fslow`	`double*`	Vector de frecuencias en tiempo lento	
		`ffast`	`double*`	Vector de frecuencias en tiempo rápido	
		`nx`	`int`	Dimensión del vector `fslow`	
		`ny`	`int`	Dimensión del vector `ffast`	
	Param OUT	Nombre	Tipo	Descripción	Modo
		`out_rfm`	`(double complex *)`	Es una matriz de frecuencias, cuadrada de `nx x ny`	outplace
	Tipo Return	Tipo			
		`void`			

<table>
<tr><td>Id:
E5.i</td><td colspan="5">Implementación S2</td></tr>
<tr><td></td><td>Lenguaje: C</td><td colspan="4">Componente: WK_FUNC</td></tr>
<tr><td></td><td>Firma:</td><td colspan="4"><code>void S2 (double complex *out_S2,double complex *S1,
double complex *rfm,int nx,int ny);</code></td></tr>
<tr><td></td><td>Descripción</td><td colspan="4">Realiza la focalización gruesa multiplicando la matriz de datos crudos en el dominio de las frecuencias por la matriz RFM.</td></tr>
<tr><td></td><td>Ámbito</td><td colspan="4">CPU</td></tr>
<tr><td></td><td>Trazabilidad Diseño:</td><td colspan="4">E3.D</td></tr>
<tr><td></td><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td></td><td></td><td><code>S1</code></td><td>double complex *</td><td colspan="2">Matriz cuadrada, de datos crudos convertidos al dominio de las frecuencias.</td></tr>
<tr><td></td><td></td><td><code>rfm</code></td><td>double complex *</td><td colspan="2">Matriz cuadrada, para realizar la corrección de focalización gruesa.</td></tr>
<tr><td></td><td></td><td><code>nx</code></td><td>int</td><td colspan="2">Filas de <code>out_S2</code> y de <code>S1</code></td></tr>
<tr><td></td><td></td><td><code>ny</code></td><td>int</td><td colspan="2">Columnas de <code>out_S2</code> y de <code>S1</code></td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td></td><td></td><td><code>out_S2</code></td><td>double complex *</td><td>Matriz cuadrada <code>nx</code> x <code>ny</code>,
Con la señal focalizada al rango de referencia seleccionado.
Focalización gruesa.</td><td>outplace</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td></td><td>void</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Id:
E6.i</td><td colspan="4">Implementación Mapeo de Stolt</td></tr>
<tr><td></td><td>Lenguaje: C</td><td colspan="3">Componente: WK_FUNC</td></tr>
<tr><td></td><td>Firma:</td><td colspan="3"><code>void delta_ffast (double*out_delta_ffast, double*fslow, int nx, double*ffast,int ny);</code></td></tr>
<tr><td></td><td>Descripción</td><td colspan="3">Construye la matriz <code>delta_ffast</code> que representa la nueva variable en rango f'_τ, obtenida por el mapeo de Stolt. Esta matriz será utilizada por el proceso de la interpolación de Stolt.</td></tr>
<tr><td></td><td>Ámbito</td><td colspan="3">CPU</td></tr>
<tr><td></td><td>Trazabilidad Diseño:</td><td colspan="3">E4.D</td></tr>
<tr><td></td><td>Param IN</td><td>Nombre</td><td>Tipo</td><td>Descripción</td></tr>
<tr><td></td><td></td><td><code>fslow</code></td><td>double*</td><td>Vector de frecuencias en tiempo lento</td></tr>
<tr><td></td><td></td><td><code>ffast</code></td><td>double*</td><td>Vector de frecuencias en tiempo rápido</td></tr>
<tr><td></td><td></td><td><code>nx</code></td><td>int</td><td>Dimensión del Vector de frecuencias en tiempo lento y también representa a las filas de <code>out_delta_ffast</code></td></tr>
<tr><td></td><td></td><td><code>ny</code></td><td>int</td><td>Dimensión del Vector de frecuencias en tiempo rápido y también representa a las columnas de <code>out_delta_ffast</code></td></tr>
<tr><td></td><td></td><td colspan="3"></td></tr>
<tr><td></td><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td></td><td></td><td><code>out_delta_ffast</code></td><td>double*</td><td>Matriz cuadra de <code>nx x ny</code>, que representa la nueva variable de frecuencia mapeada.</td><td>outplace</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Tipo Return</td><td>Tipo</td><td colspan="2"></td></tr>
<tr><td></td><td></td><td>void</td><td colspan="2"></td></tr>
</table>

<table>
<tr><td>Id:
E7.i</td><td colspan="5">Implementación interpolación de Stolt</td></tr>
<tr><td></td><td>Lenguaje: C</td><td colspan="4">Componente: WK_FUNC</td></tr>
<tr><td></td><td>Firma:</td><td colspan="4"><code>void stolt (double complex*out_stolt_S3,double*deltafast, double complex *S2,int filas,int cols);</code></td></tr>
<tr><td></td><td>Descripción</td><td colspan="4">Realiza la interpolación de Stolt, focalización diferencial.</td></tr>
<tr><td></td><td>Ámbito</td><td colspan="4">CPU</td></tr>
<tr><td></td><td>Trazabilidad Diseño:</td><td colspan="4">E5.D</td></tr>
<tr><td></td><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td></td><td></td><td><code>deltafast</code></td><td>double*</td><td colspan="2">Matriz cuadrada que representa a la nueva variable en rango obtenida por el mapeo de Stolt y que debe ser interpolada en una grilla equiespaciada.</td></tr>
<tr><td></td><td></td><td><code>S2</code></td><td>(double complex *)</td><td colspan="2">Matriz cuadrada que representa a la señal luego de la focalización gruesa, para el rango de referencia seleccionado.</td></tr>
<tr><td></td><td></td><td><code>filas</code></td><td>int</td><td colspan="2">Filas de <code>deltafast</code> y de <code>S2</code></td></tr>
<tr><td></td><td></td><td><code>cols</code></td><td>int</td><td colspan="2">Columnas <code>deltafast</code> y de <code>S2</code></td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td></td><td></td><td><code>out_stolt_S3</code></td><td>(double complex *)</td><td>Es una matriz cuadrada de <code>filas x cols</code> que contiene la focalización diferencial</td><td>outplace</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td></td><td>void</td><td colspan="3"></td></tr>
</table>

5.2.4 Especificación de WK: implementación en CUDA

<table>
<tr><td>Id:
E8.i</td><td colspan="5">Implementación FFTSHIFT</td></tr>
<tr><td rowspan="15"></td><td>Lenguaje:
CUDA</td><td colspan="4">Componente: WK_Kernels</td></tr>
<tr><td>Firma:</td><td colspan="4"><code>__global__ void fftshift_kernel(cuComplex *d_matrix, int filas, int cols)</code></td></tr>
<tr><td>Descripción</td><td colspan="4">Realiza la fftshift para una matriz de datos complejos en 2D.
Para matrices cuadradas también se la utiliza como ifftshift.</td></tr>
<tr><td>Ámbito</td><td colspan="4">GPU</td></tr>
<tr><td>Trazabilidad Diseño:</td><td colspan="4">E6.D, E2.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="3"></td><td><code>d_matrix</code></td><td><code>cuComplex *</code></td><td colspan="2">Es una matriz cuadrada. Puede estar en el domino del tiempo o de la frecuencia.</td></tr>
<tr><td><code>filas</code></td><td><code>int</code></td><td colspan="2">Filas de <code>d_matrix</code></td></tr>
<tr><td><code>cols</code></td><td><code>int</code></td><td colspan="2">Columnas de <code>d_matrix</code></td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td><code>d_matrix</code></td><td><code>cuComplex *</code></td><td>Es una matriz cuadrada de filas x cols que contiene a d_matrix luego del proceso de shifting</td><td>inplace</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td><code>void</code></td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Id:
E9.i</td><td colspan="5">Implementación FFT/IFFT: Definición del plan 2D: <code>cufftPlan2d</code></td></tr>
<tr><td rowspan="11"></td><td>Lenguaje:
CUDA</td><td colspan="4">Componente: cuFFT</td></tr>
<tr><td>Firma:</td><td colspan="4">cufftResult cufftPlan2d(cufftHandle *plan, int nx, int ny, cufftType type);</td></tr>
<tr><td>Uso</td><td colspan="4">cufftHandle plan;

cufftPlan2d(&plan, Nslow, Nfast, CUFFT_C2C);

cufftExecC2C(plan, d_S, d_S, CUFFT_FORWARD);
ó
cufftExecC2C(plan, d_S, d_S, CUFFT_BACKWARD);</td></tr>
<tr><td>Descripción</td><td colspan="4">Realiza la definición del plan de ejecución 2D para la transformada Rápida de Fourier. Pero no lo ejecuta. Para la ejecución hace falta invocar a la función <code>cufftExecC2C</code>(.), utilizando el parámetro CUFFT_FORWARD para la FFT o CUFFT_BACKWARD para la IFFT</td></tr>
<tr><td>Ámbito</td><td colspan="4">GPU</td></tr>
<tr><td>Trazabilidad
Diseño:</td><td colspan="4">E2.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="4"></td><td>nx</td><td>int</td><td colspan="2">Dimensión en filas de la matriz a la que se aplicara el plan de FFT</td></tr>
<tr><td>ny</td><td>int</td><td colspan="2">Dimensión en columnas de la matriz a la que se aplicada el plan de FFT</td></tr>
<tr><td>plan</td><td>cufftHandle *</td><td colspan="2">Referencia al objeto plan declarado</td></tr>
<tr><td>type</td><td>cufftType</td><td colspan="2">CUFFT_C2C</td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>plan</td><td>cufftHandle *</td><td>Referencia al objeto plan creado</td><td>Se fuerza el modo Inplace haciendo idata = odata en <code>cufftExecC2C</code></td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3">Descripción</td></tr>
<tr><td></td><td>cufftResult</td><td colspan="3">CUFFT_SUCCESS
CUFFT_ALLOC_FAILED
CUFFT_INVALID_VALUE
CUFFT_INTERNAL_ERROR
CUFFT_SETUP_FAILED</td></tr>
</table>

<table>
<tr><td>Id:
E10.i</td><td colspan="5">Implementación FFT/IFFT : Ejecución del Plan 2D: <code>cufftExecC2C</code></td></tr>
<tr><td rowspan="15"></td><td>Lenguaje:
CUDA</td><td colspan="4">Componente: cuFFT</td></tr>
<tr><td>Firma:</td><td colspan="4">cufftResult cufftExecC2C(cufftHandle plan,
cufftComplex *idata, cufftComplex *odata,
int direction);</td></tr>
<tr><td>Uso</td><td colspan="4">cufftHandle plan;
cufftPlan2d(&plan, Nslow, Nfast, CUFFT_C2C);

cufftExecC2C(plan, d_S, d_S, CUFFT_FORWARD);
ó
cufftExecC2C(plan, d_S, d_S, CUFFT_BACKWARD);</td></tr>
<tr><td>Descripción</td><td colspan="4">Ejecuta un plan de FFT o IFFT</td></tr>
<tr><td>Trazabilidad
Diseño:</td><td colspan="4">E6.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="3"></td><td>plan</td><td>int</td><td colspan="2">cufftHandle devuelto por
cufftPlan2d()</td></tr>
<tr><td>idata</td><td>(cufftComplex *)</td><td colspan="2">Puntero a la matriz de números complejos de entrada en la memoria de la GPU para ser transformada</td></tr>
<tr><td>direction</td><td>int</td><td colspan="2">La dirección de la transformación
CUFFT_FORWARD para FFT
CUFFT_INVERSE para IFFT</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>odata</td><td>(cufftComplex *)</td><td>Puntero a la matriz de salida en números complejos en la memoria de la GPU.</td><td>Se fuerza el modo Inplace haciendo idata = odata</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3">Descripción</td></tr>
<tr><td></td><td>cufftResult</td><td colspan="3">CUFFT_SUCCESS
CUFFT_ALLOC_FAILED
CUFFT_INVALID_VALUE
CUFFT_INTERNAL_ERROR
CUFFT_SETUP_FAILED</td></tr>
</table>

<table>
<tr><td>Id:
E11.i</td><td colspan="5">Implementación RFM</td></tr>
<tr><td rowspan="14"></td><td>Lenguaje:
CUDA</td><td colspan="4">Componente: WK_Kernels</td></tr>
<tr><td>Firma:</td><td colspan="4">__global__ void rfm_kernel (cuComplex *out_rfm,
double *fslow, int nx, double *ffast, int ny)</td></tr>
<tr><td>Descripción</td><td colspan="4">Construye la matriz RFM basado en los vectores de frecuencias en tiempo lento y rápido. Esta matriz servirá luego para realizar la focalización gruesa para el rango de referencia seleccionado.</td></tr>
<tr><td>Ámbito</td><td colspan="4">GPU</td></tr>
<tr><td>Trazabilidad
Diseño:</td><td colspan="4">E3.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="4"></td><td>fslow</td><td>double*</td><td colspan="2">Vector de frecuencias en tiempo lento</td></tr>
<tr><td>ffast</td><td>double*</td><td colspan="2">Vector de frecuencias en tiempo rápido</td></tr>
<tr><td>nx</td><td>int</td><td colspan="2">Dimensión del vector fslow</td></tr>
<tr><td>ny</td><td>int</td><td colspan="2">Dimensión del vector ffast</td></tr>
<tr><td></td><td colspan="4"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>out_rfm</td><td>(cuComplex *)</td><td>Es una matriz de frecuencias, cuadrada de nx x ny</td><td>outplace</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td>void</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Id:
E12.i</td><td colspan="5">Implementación S2</td></tr>
<tr><td rowspan="15"></td><td>Lenguaje:
CUDA</td><td colspan="4">Componente: WK_Kernels</td></tr>
<tr><td>Firma:</td><td colspan="4">__global__ void S2_kernel (cuComplex *S1,
cuComplex *rfm, int nx, int ny)</td></tr>
<tr><td>Descripción</td><td colspan="4">Realiza la focalización gruesa multiplicando la matriz de datos crudos en el dominio de las frecuencias por la matriz RFM.</td></tr>
<tr><td>Ámbito</td><td colspan="4">GPU</td></tr>
<tr><td>Trazabilidad
Diseño:</td><td colspan="4">E3.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="4"></td><td>S1</td><td>(cuComplex*)</td><td colspan="2">Matriz cuadrada, de datos crudos convertidos al dominio de las frecuencias.</td></tr>
<tr><td>rfm</td><td>(cuComplex*)</td><td colspan="2">Matriz cuadrada, para realizar la corrección de focalización gruesa.</td></tr>
<tr><td>nx</td><td>int</td><td colspan="2">Filas de rfm y de S1</td></tr>
<tr><td>ny</td><td>int</td><td colspan="2">Columnas de rfm y de S1</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>S1</td><td>(cuComplex*)</td><td>Matriz cuadrada nx x ny,
Con la señal focalizada al rango de referencia seleccionado.
Focalización gruesa.</td><td>inplace</td></tr>
<tr><td colspan="4"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td>void</td><td colspan="3"></td></tr>
</table>

<table>
<tr><td>Id:
E13.i</td><td colspan="4">Implementación Mapeo de Stolt</td></tr>
<tr><td rowspan="13"></td><td>Lenguaje:
CUDA</td><td colspan="3">Componente: WK_Kernels</td></tr>
<tr><td>Firma:</td><td colspan="3">__global__ void delta_ffast_kernel
(double *out_delta_ffast, double *fslow,
int nx, double *ffast, int ny)</td></tr>
<tr><td>Descripción</td><td colspan="3">Construye la matriz delta_ffast que representa la nueva variable en rango f'_τ, obtenida por el mapeo de Stolt. Esta matriz será utilizada por el proceso de la interpolación de Stolt.</td></tr>
<tr><td>Trazabilidad
Diseño:</td><td colspan="3">E4.D</td></tr>
<tr><td>Param IN</td><td>Nombre</td><td>Tipo</td><td>Descripción</td></tr>
<tr><td rowspan="5"></td><td>fslow</td><td>double*</td><td>Vector de frecuencias en tiempo lento</td></tr>
<tr><td>ffast</td><td>double*</td><td>Vector de frecuencias en tiempo rápido</td></tr>
<tr><td>nx</td><td>int</td><td>Dimensión del Vector de frecuencias en tiempo lento y también representa a las filas de out_delta_ffast</td></tr>
<tr><td>ny</td><td>int</td><td>Dimensión del Vector de frecuencias en tiempo rápido y también representa a las columnas de out_delta_ffast</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td rowspan="2"></td><td>out_delta_ffast</td><td>double*</td><td>Matriz cuadra de nx x ny, que representa la nueva variable de frecuencia mapeada.</td><td>outplace</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Tipo Return</td><td>Tipo</td><td colspan="2"></td></tr>
<tr><td></td><td>void</td><td colspan="2"></td></tr>
</table>

<table>
<tr><td>Id:
E14.i</td><td colspan="5">Implementación interpolación de Stolt</td></tr>
<tr><td></td><td>Lenguaje:
CUDA</td><td colspan="4">Componente: WK_Kernels</td></tr>
<tr><td></td><td>Firma:</td><td colspan="4">__global__ void stolt_kernel (cuComplex *out_stolt_S3,
double *deltafast, cuComplex *S2, int filas, int cols)</td></tr>
<tr><td></td><td>Descripción</td><td colspan="4">Realiza la interpolación de Stolt, focalización diferencial.</td></tr>
<tr><td></td><td>Trazabilidad
Diseño:</td><td colspan="4">E5.D</td></tr>
<tr><td></td><td>Param IN</td><td>Nombre</td><td>Tipo</td><td colspan="2">Descripción</td></tr>
<tr><td></td><td></td><td>deltafast</td><td>double*</td><td colspan="2">Matriz cuadrada que representa a la nueva variable en rango obtenida por el mapeo de Stolt y que debe ser interpolada en una grilla equiespaciada.</td></tr>
<tr><td></td><td></td><td>S2</td><td>(cuComplex*)</td><td colspan="2">Matriz cuadrada que representa a la señal luego de la focalización gruesa, para el rango de referencia seleccionado.</td></tr>
<tr><td></td><td></td><td>filas</td><td>int</td><td colspan="2">Filas de deltafast y de S2</td></tr>
<tr><td></td><td></td><td>cols</td><td>int</td><td colspan="2">Columnas deltafast y de S2</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Param OUT</td><td>Nombre</td><td>Tipo</td><td>Descripción</td><td>Modo</td></tr>
<tr><td></td><td></td><td>out_stolt_S3</td><td>(cuComplex*)</td><td>Es una matriz cuadrada de filas x cols que contiene la focalización diferencial</td><td>outplace</td></tr>
<tr><td></td><td></td><td colspan="4"></td></tr>
<tr><td></td><td>Tipo Return</td><td>Tipo</td><td colspan="3"></td></tr>
<tr><td></td><td></td><td>void</td><td colspan="3"></td></tr>
</table>

5.3 Arquitectura del software: diagramas de componentes y secuencia

5.3.1 Arquitectura WK Matlab

Diagrama de componentes WK-Matlab:

WK_Main:

Contiene la función Main (.) que coordina la lógica principal del algoritmo.

WK_Func:

Agrupa todas las funciones de implementación propia, tanto funciones auxiliares como las principales del WK. A Este componente pertenecen las funciones RFM(), Stolt_Mapeo(), Stolt_Interpolacion().

Matlab_Func:

Representa la librería de funciones de MATLAB. A este componente pertenecen las funciones fft2(), fftshift() y sus inversas.

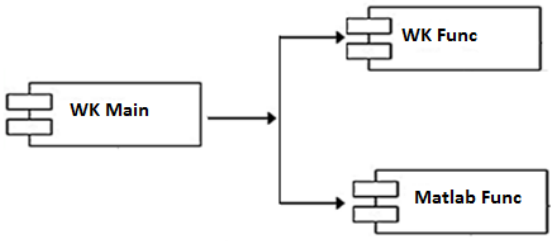


Figura 5.1: diagrama de componentes, basado en UML de WK-Matlab

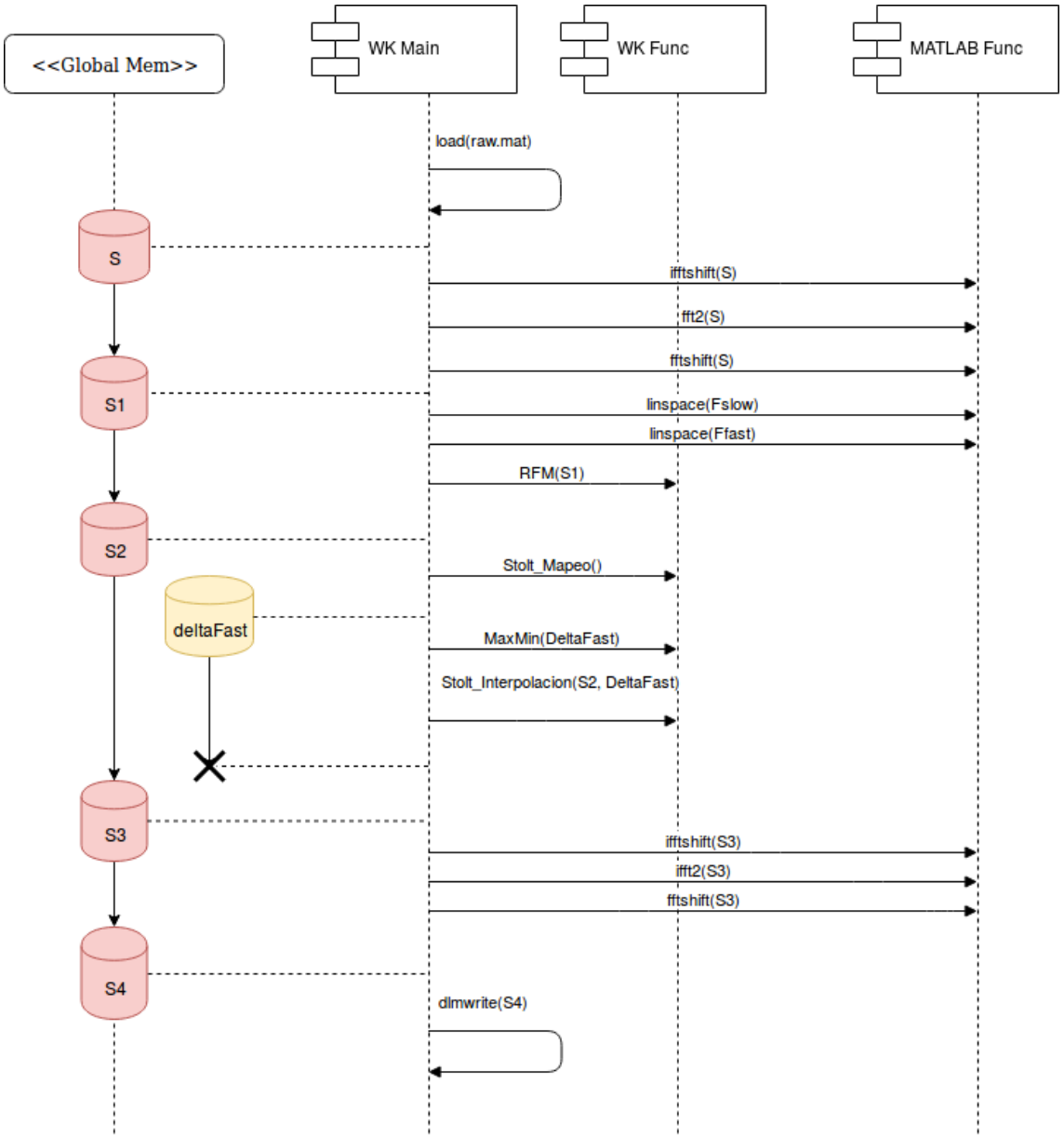


Figura 5.2 diagrama de secuencia basado en UML de WK-Matlab

5.3.2 Arquitectura de WK-C

Diagrama de componentes WK-C:

WK_Main:

Contiene la función Main(.) que coordina la lógica principal del algoritmo.

WK_Func:

Contiene todas las funciones de implementación propia, tanto funciones auxiliares como las principales del WK. A Este componente pertenecen las funciones: fftshift(), rfmS2(), delta_ffast() y Stolt().

FFTW:

FFTW es una biblioteca para C para calcular la transformada discreta de Fourier (DFT) en una o más dimensiones, de tamaño de entrada arbitraria y de datos reales o complejos. El paquete FFTW fue desarrollado en el MIT (*Massachusetts Institute of Technology*) por Matteo Frigo y Steven G. Johnson. Esta librería implementa la DFT mediante el algoritmo FFT (*Fast Fourier Transform*). Para más información ver: (www.fftw.org/fftw3.pdf). A este componente pertenecen las funciones: fftw_plan_dft_2d(), fftw_execute() y todas que llevan el prefijo “fftw_”.

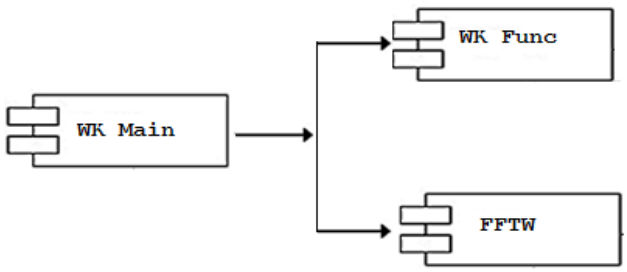


Figura 5.3: diagrama de componentes basado en UML del WK-C

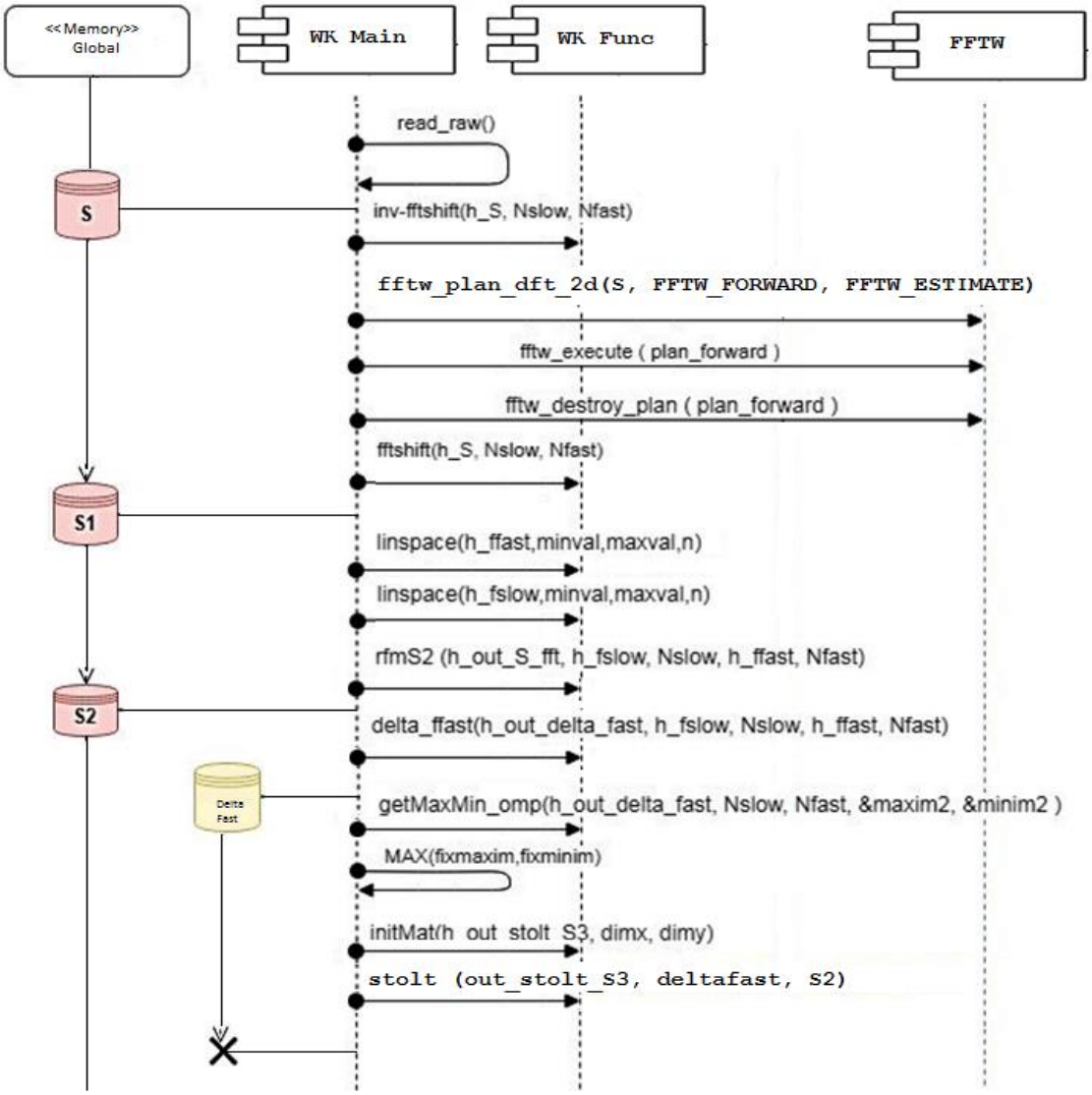


Figura 5.4: diagrama de secuencia basado en UML de WK-C, parte 1 de 2

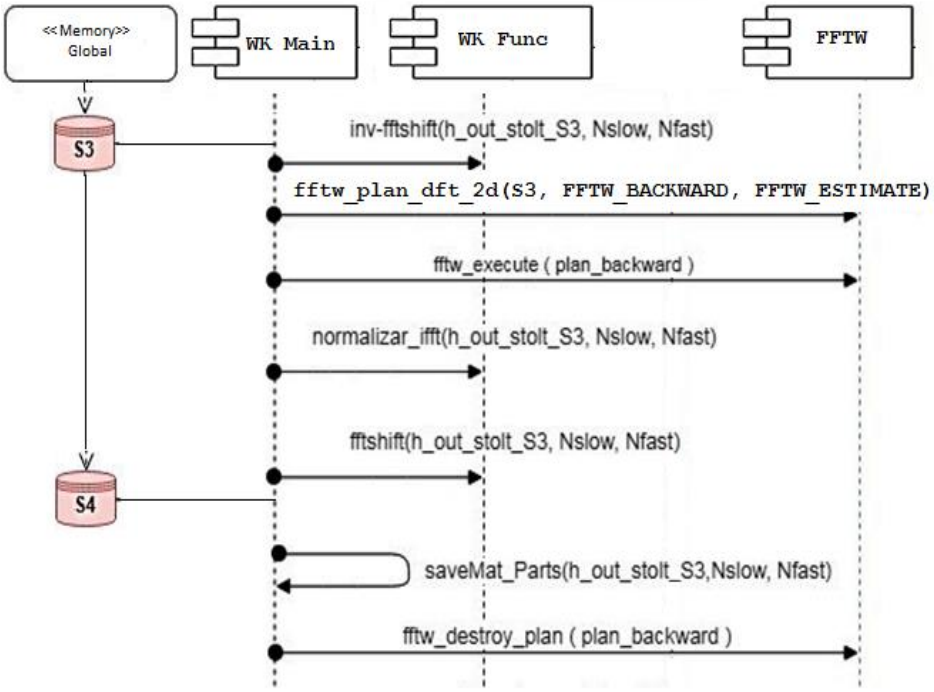


Figura 5.5: diagrama de secuencia basado en UML de WK-C, parte 2

5.3.3 Arquitectura de WK-CUDA

A continuación se ofrece un diagrama de componentes que forman la versión del WK en CUDA

WK_Main:

Contiene la función Main(.) que coordina la lógica principal del algoritmo y funciones auxiliares que se ejecutan en la CPU. WK_Main se comunica con la GPU a través de la interfaz Wk_Wrappers. WK_Main corre en la CPU.

WK_Wrappers:

Representa una capa media de desacople entre la CPU y GPU. La idea de introducir una capa media de desacople es considerada una buena práctica de diseño en ingeniería del software. Esta capa media evita que las invocaciones a las funciones de la GPU (*kernels*), se hagan directamente desde el cuerpo del Main, brindando una interfaz constante para las invocaciones desde la CPU. De esta forma se mantiene a resguardo la lógica principal del algoritmo frente a cambios en las invocaciones a los *Kernels* de CUDA, ya sea en el número de hilos o cantidad de bloques, o bien que se decida en el futuro invocar a otra implementación para una determinada función. WK_Wrappers corre en la CPU. Por Ejemplo: actualmente para

realizar la operación de FFTSHIFT se está utilizando una implementación propia que el futuro se podría reemplazar quizá por otra versión propia optimizada o por una versión de terceros. Como la interfaz de la capa media es inmutable, el algoritmo principal no se verá afectado por cambios de este tipo.

WK_Kernels:
Contiene todas las interfaces para invocar a los kernels tanto de desarrollo propio como de terceros, que implementan las principales funciones del algoritmo en CUDA.

CuFFT:
Es una librería en CUDA para GPU hecha por NVIDIA que implementa la transformada discreta de Fourier mediante el algoritmo FFT (*Fast Fourier Transform*) altamente optimizada para cómputo en punto flotante. Para más información ver: (CUDA Toolkit Documentation v9, 2018).

Thrust:
Es una librería para CUDA basada en la biblioteca de plantillas estándar (STL), creada originalmente por Jared Hoberock y Nathan Bell, proporciona una interfaz de alto nivel para una amplia colección de primitivas paralelas, tales como *Scan*, *Sort* , *Reduce* y otras que se pueden utilizar juntas para implementar algoritmos complejos. Para más información ver: (CUDA Toolkit Documentation v9, 2018).

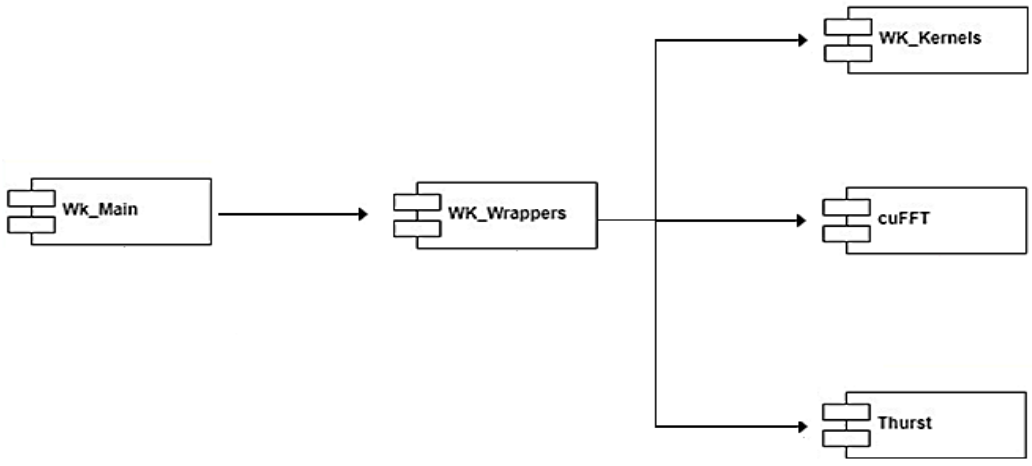


Figura 5.6: diagrama de componentes basado en UML de WK-CUDA

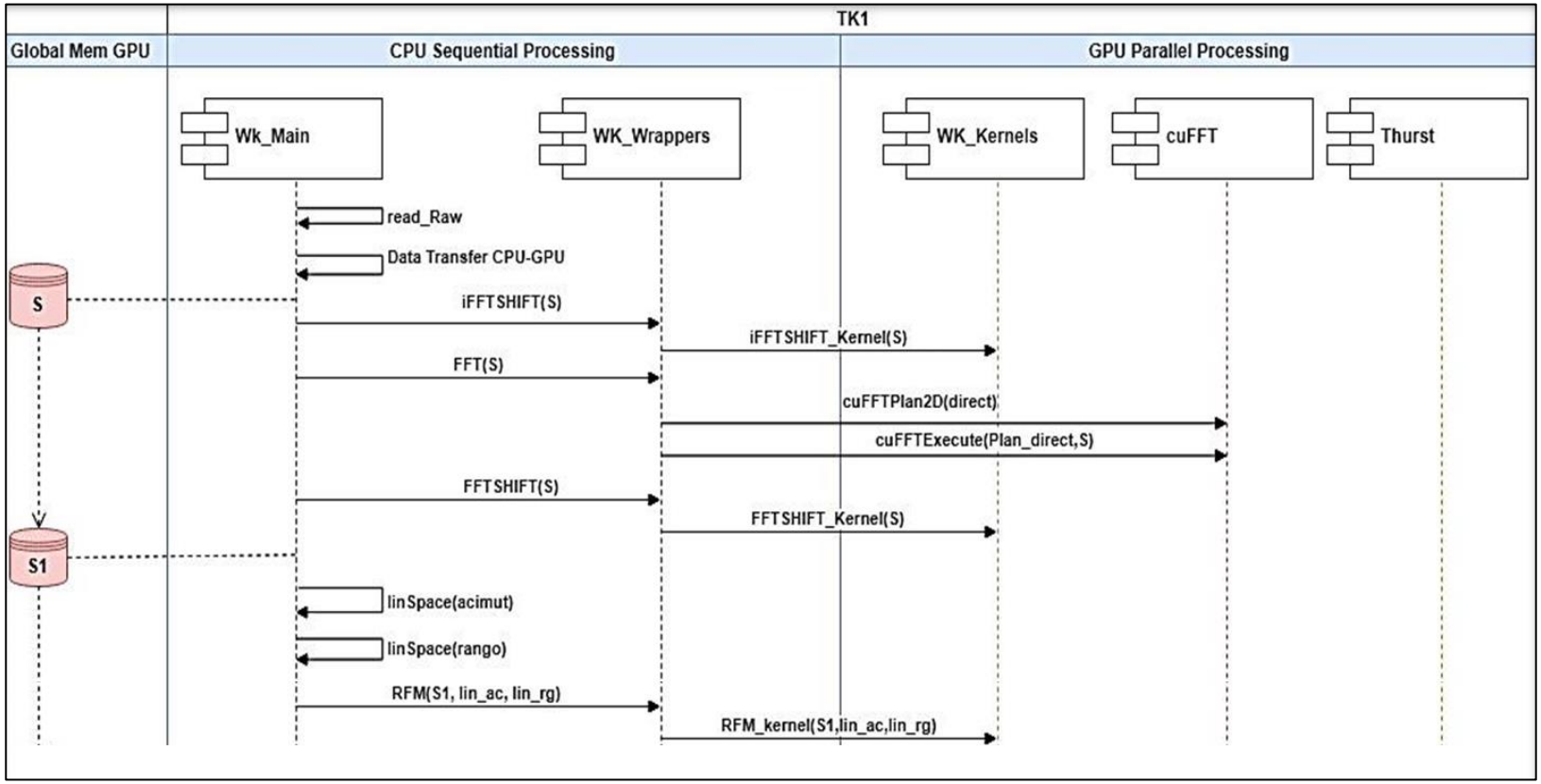


Figura 5.7 diagrama de secuencia basado en UML de WK-CUDA parte 1 de 2.

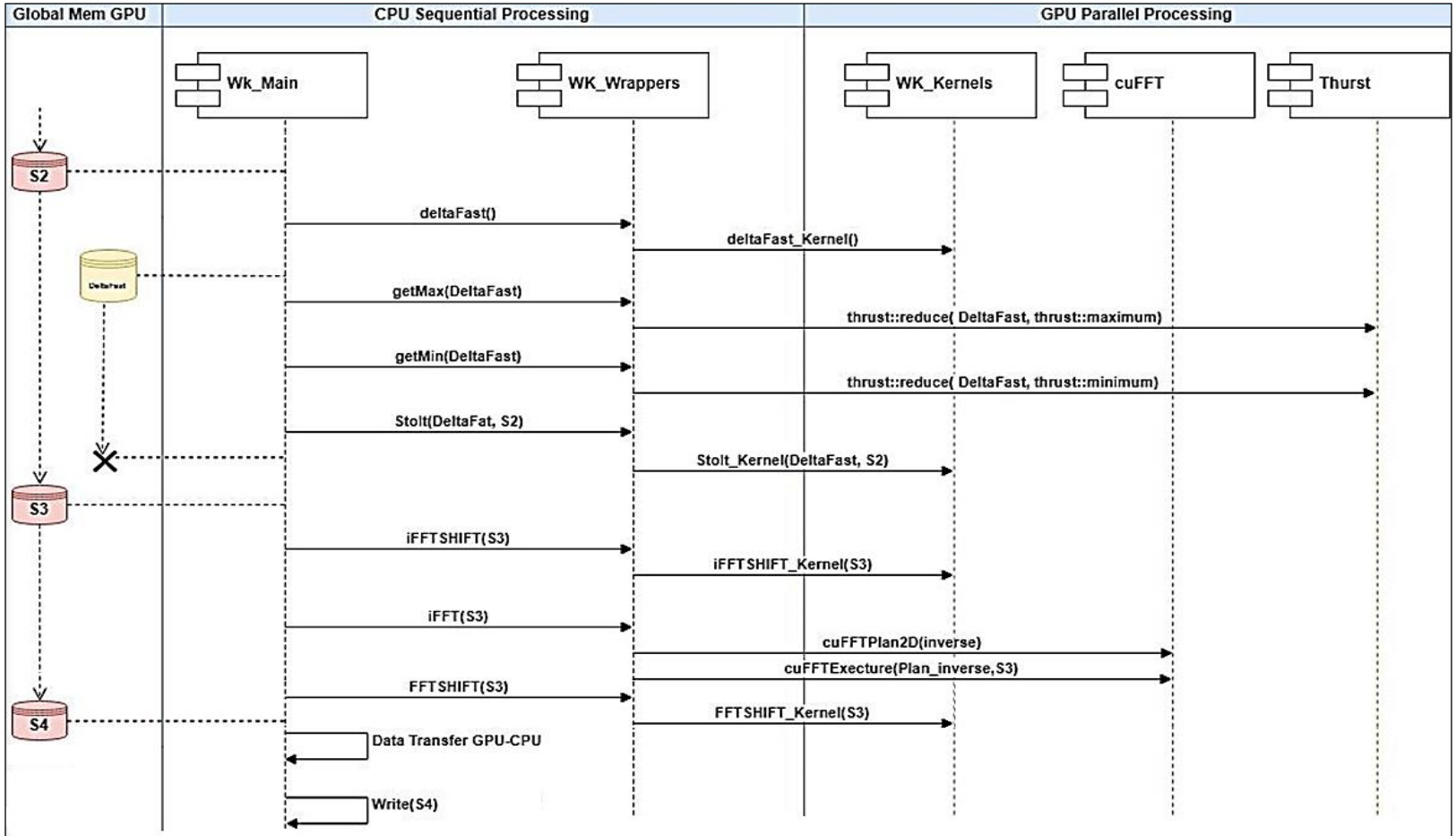


Figura 5.8 : diagrama de secuencia basado en UML de WK-CUDA

5.4 Formatos de Entrada-Salida

En la siguiente sección se presentan los formatos de archivos utilizados en el trabajo.

*Formato *.MAT:*

Es utilizado por el simulador de datos crudos de un blanco puntual como salida. De acuerdo a nuestra implementación este archivo siempre contendrá una única variable: una matriz cuadrada de números complejos en doble precisión.
Los datos crudos en formato *.MAT son ingestados por la versión de WK-MATLAB, generando una respuesta focalizada en dos tipos de salidas configurables: *.MAT y *.CSV. La figura 5.9 ofrece el detalle del formato binario MAT.

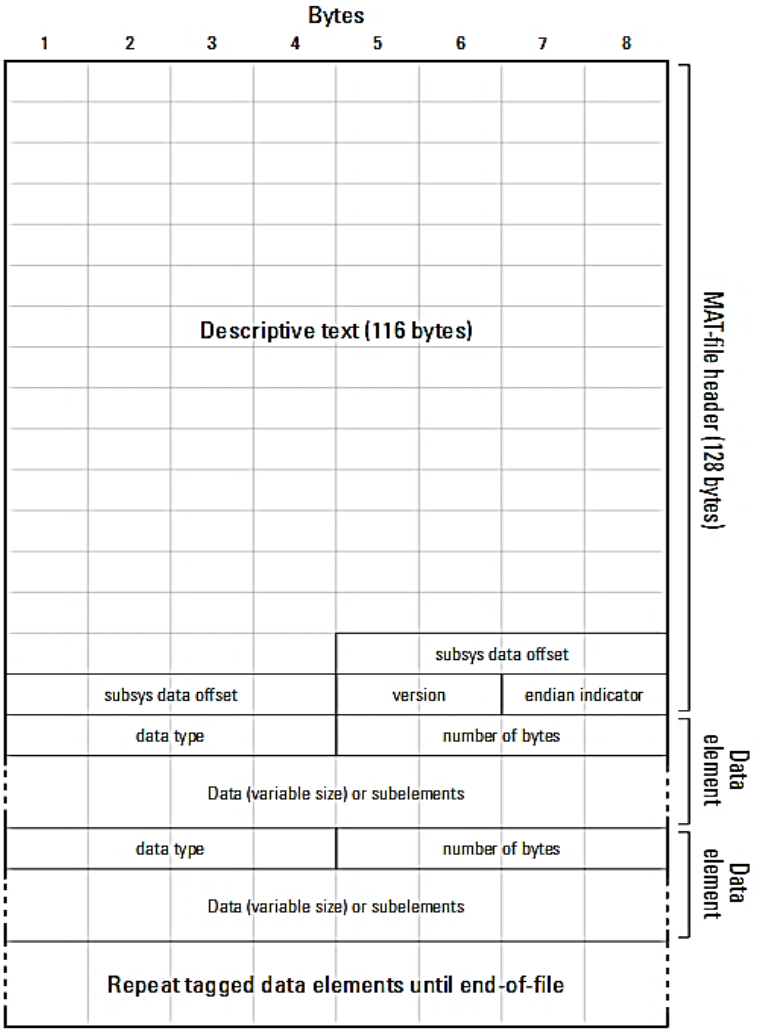


Figura 5.9 Formato de Archivo MAT, nivel 5

*Formato *.CSV*:

Es un formato simple en texto plano. En esta trabajo se utiliza una dupla archivos CSV para almacenar una matriz cuadrada de numeros complejos en doble precisión con signo. En un archivo se almacena la parte real de la matriz y en el otro la parte imaginaria. El caracter de separación es la coma "," y se utiliza el punto "." para indicar la parte decimal. Este formato es utilizado por las versiones de Wk en C y Cuda como mecanismo de ingestión de los datos crudos y para la entrega de la salida focalizada.

Figura 5.10 Formato de CSV. Parte real e imaginaria en doble precisión.

5.4.1 Flujo de Productos

En esta sección se presenta el modelo conceptual del flujo de productos del sistema:

El simulador de datos crudos (Raw) lee el archivo de parámetros y genera un conjunto de datos crudos en formato *.MAT. Luego mediante un script conversor (Apéndice I), son divididos en dos archivos:

uno que representa la parte real de los datos y otro para la parte imaginaria, ambos en formato *.TXT CSV, nombrados con el prefijo “S” en el diagrama. Los archivos “S” serán la entrada para las implementaciones de WK en C y CUDA. Estas versiones de WK entregan una salida focalizada llamada “S4” también como una dupla de archivos *.TXT CSV para la parte real e imaginaria de la señal.

La versión inicial del WK escrita en MATLAB acepta como entrada directamente el formato *.MAT proveniente del simulador. WK MATLAB entrega su salida también en formato *.MAT, denominada “S4.MAT”. Mediante el mismo script conversor (Apéndice I), se obtiene la transformación de S4.MAT en la dupla real e imaginaria en formato *.TXT CSV.

La versión de WK MATLAB se puede configurar para invocar al script graficador (Apéndice F) y generar imágenes *.PNG de la señal focalizada.

El punto de validación del flujo de productos esta destacado en el diagrama. Los archivos con datos reales e imaginarios que provienen de las versiones WK MATLAB y WK C/CUDA se validan mediante un script MATLAB (Apéndice H) para asegurar la igualdad de las versiones y que las comparaciones de performance en tiempo tengan sentido.

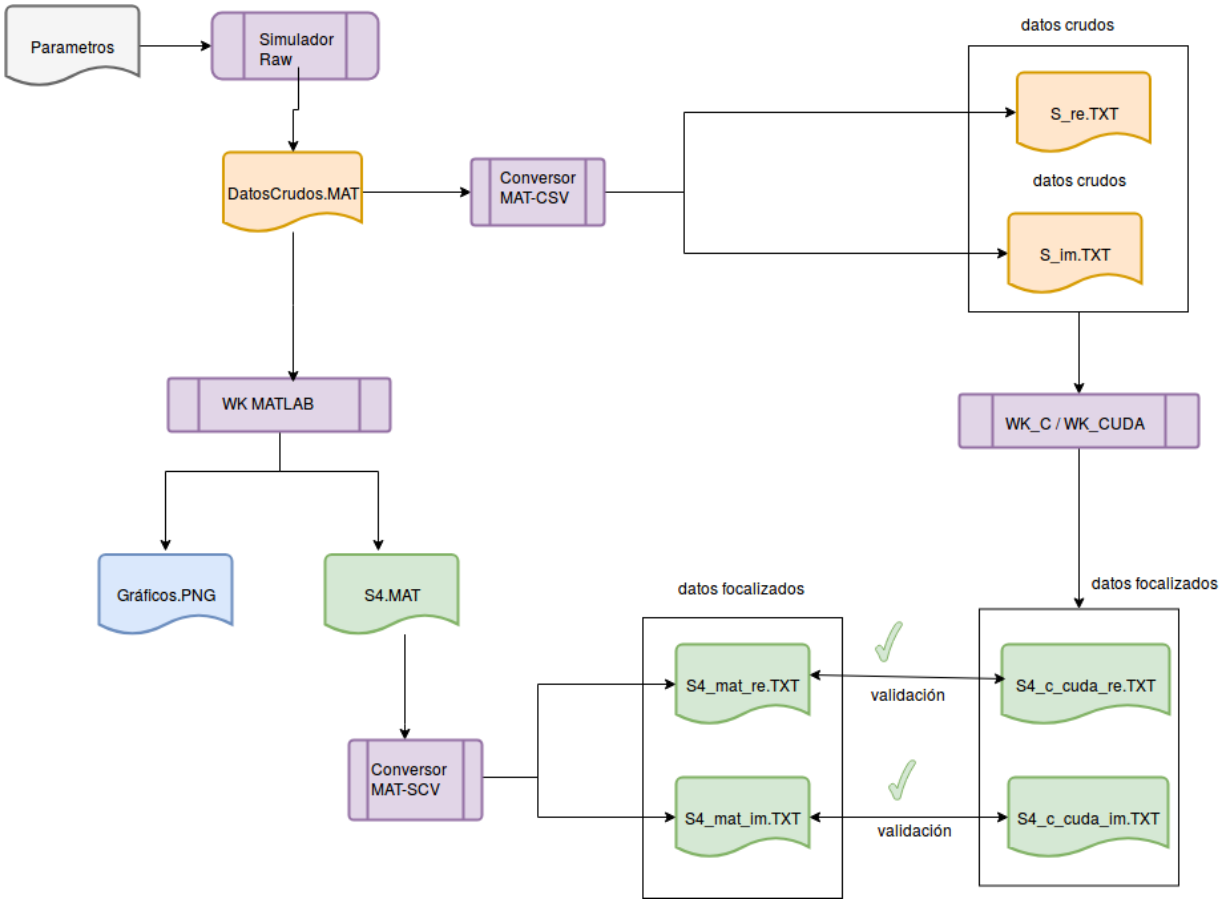


Figura 5.10.1 Flujos de Productos del Sistema

5.5 Compilación y Ejecución de las implementaciones

Simulador_MATLAB	
Compilación	**N/A**
Ejecución	➢ **simulador_IRF**

Wk_MATLAB	
Compilación	**N/A**
Ejecución	➢ **wk_matlab**

Wk_C	
Compilación	**gcc wk.c -lfftw3 -lm -o wk_c**
Ejecución	**./wk_c**

WK_CUDA	
Compilación	**nvcc -arch=sm_32 wk.cu -o wk_cuda –lcufft**
Ejecución	**./wk_cuda**

Tabla 5.1 Compilación de códigos

5.6 Trazabilidad de las implementaciones

En la siguiente sección se presenta la trazabilidad entre las ecuaciones y las implementaciones en MATLAB, C y CUDA, para las principales funciones del algoritmo WK.

5.6.1 Trazabilidad de la implementación del Simulador de datos crudo para un blanco puntual en MATLAB

(a) Ecuación

$$h_{\text{imp}}(\tau,\eta) = w_r\left(\tau - 2R(\eta)/c\right) w_a(\eta-\eta_c) \times \exp\{-j\,4\pi\, f_0\, R(\eta)/c\} \exp\left\{j\pi\, K_r \left(\tau - 2R(\eta)/c\right)^2\right\}$$

(b) MATLAB

```
Ts_tau = 1/p.Fr;
Ts_eta = 1/p.PRF;

Neta = 2048;
Ntau = 4096;

% Time vectors
Eta_meas = linspace(-Neta/2,Neta/2,Neta)*Ts_eta;
Eta      =  Eta_meas + p.eta_c;
Tau      = [0:Ntau-1]*Ts_tau + 2*p.Rca/p.C;


Rv = sqrt(p.Rt.^2 + (p.Vr * Eta).^2);
nReq = floor((Rv-p.Rca)/(Ts_tau*p.C));

beta = 2.5;
Nwr  = floor(p.Tr/Ts_tau); % number of samples in window
wr   = kaiser(Nwr,beta);

BeamWidth = p.lambda/p.La;

theta= atan(p.Vr/p.Rca*(Eta-p.eta_c));

wa= sinc(theta/BeamWidth).^2;

for net=1:Neta
    rangeIx = nReq(net)+[0:Nwr-1];
    rtaSAR(net,rangeIx) = wr'.* exp(-1i * 4 * pi * p.f0 * Rv(net)/p.C) .* exp(1i*pi*p.Kr*(Tau(rangeIx)-2*Rv(net)/p.C).^2);
end

rtaSAR = repmat(wa',1,Ntau) .* rtaSAR;
```

Figura 5.11 Trazabilidad en MATLAB de la Respuesta al impulso de un sistema SAR

5.6.2 Trazabilidad de la implementación de la función RFM en MATLAB, C y CUDA

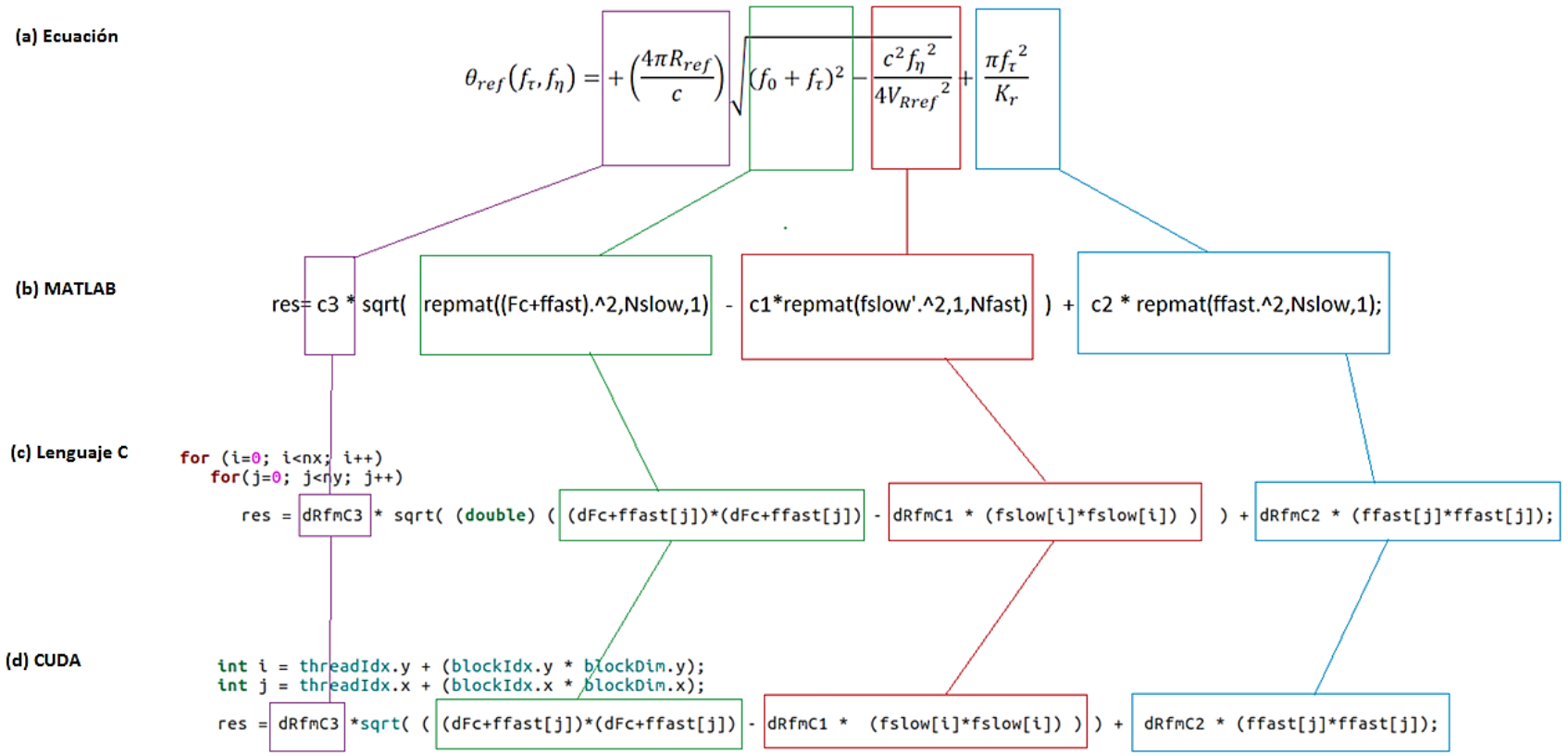


Figura 5.12 Trazabilidad en MATLAB, C y CUDA de la función RFM que realiza la focalización gruesa para un rango de referencia seleccionado

5.6.3 Trazabilidad de la implementación del Mapeo de Stolt en MATLAB, C y CUDA

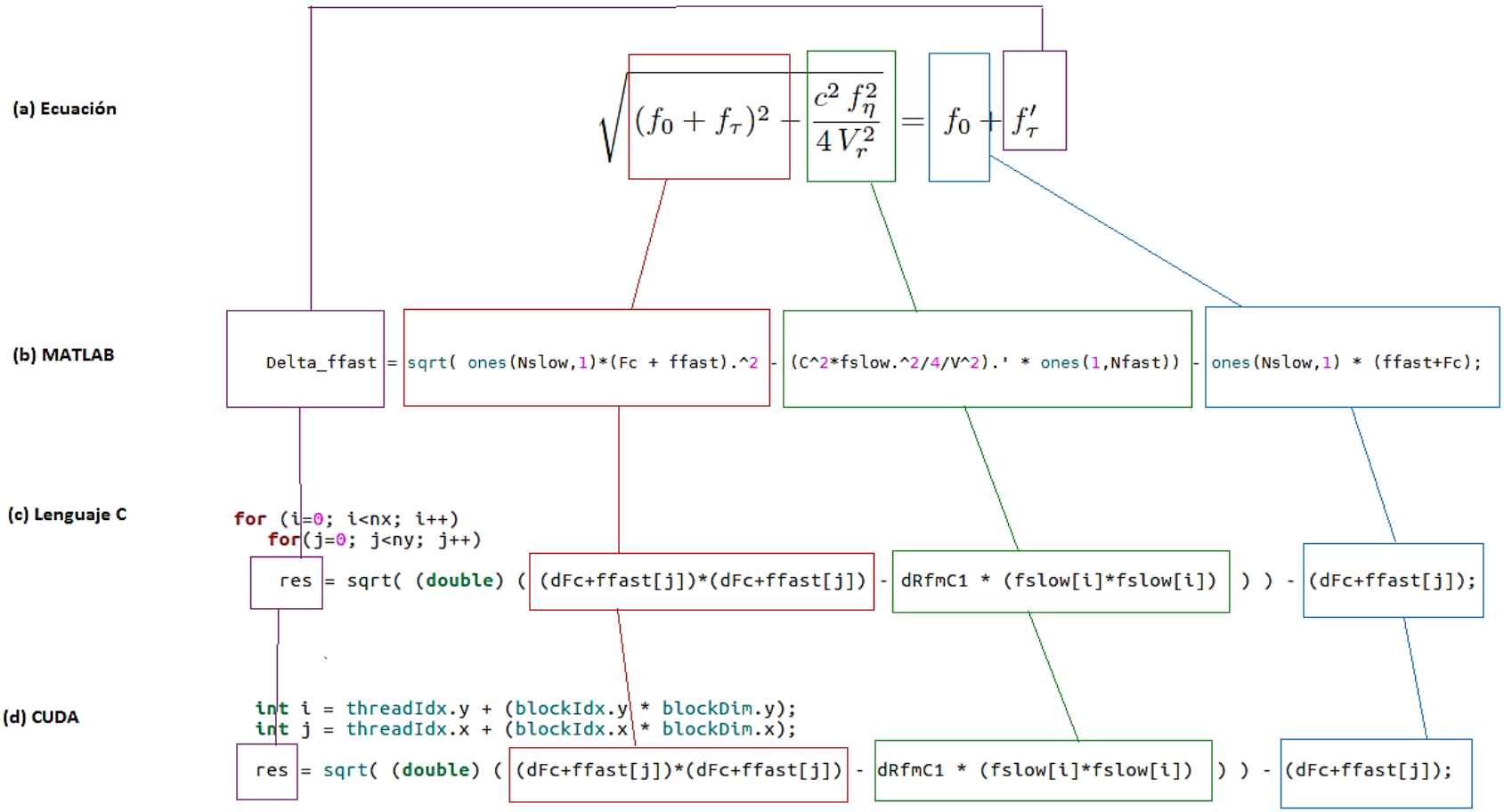


Figura 5.13 Trazabilidad en MATLAB, C y CUDA del Mapeo de Stolt

5.6.4 Trazabilidad de la implementación de la Interpolación de Stolt entre MATLAB y C

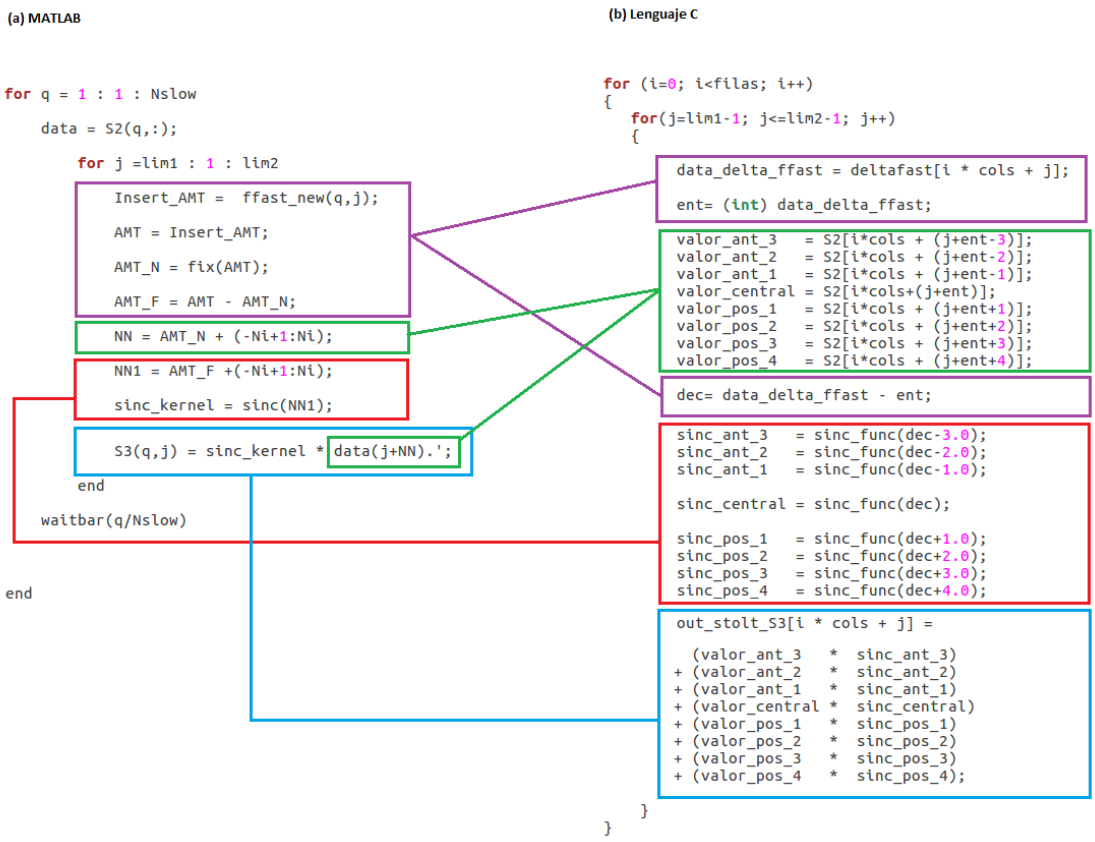


Figura 5.14 Trazabilidad entre MATLAB y C de la interpolación de Stolt

5.6.5 Trazabilidad de la implementación de la Interpolación de Stolt entre C y CUDA

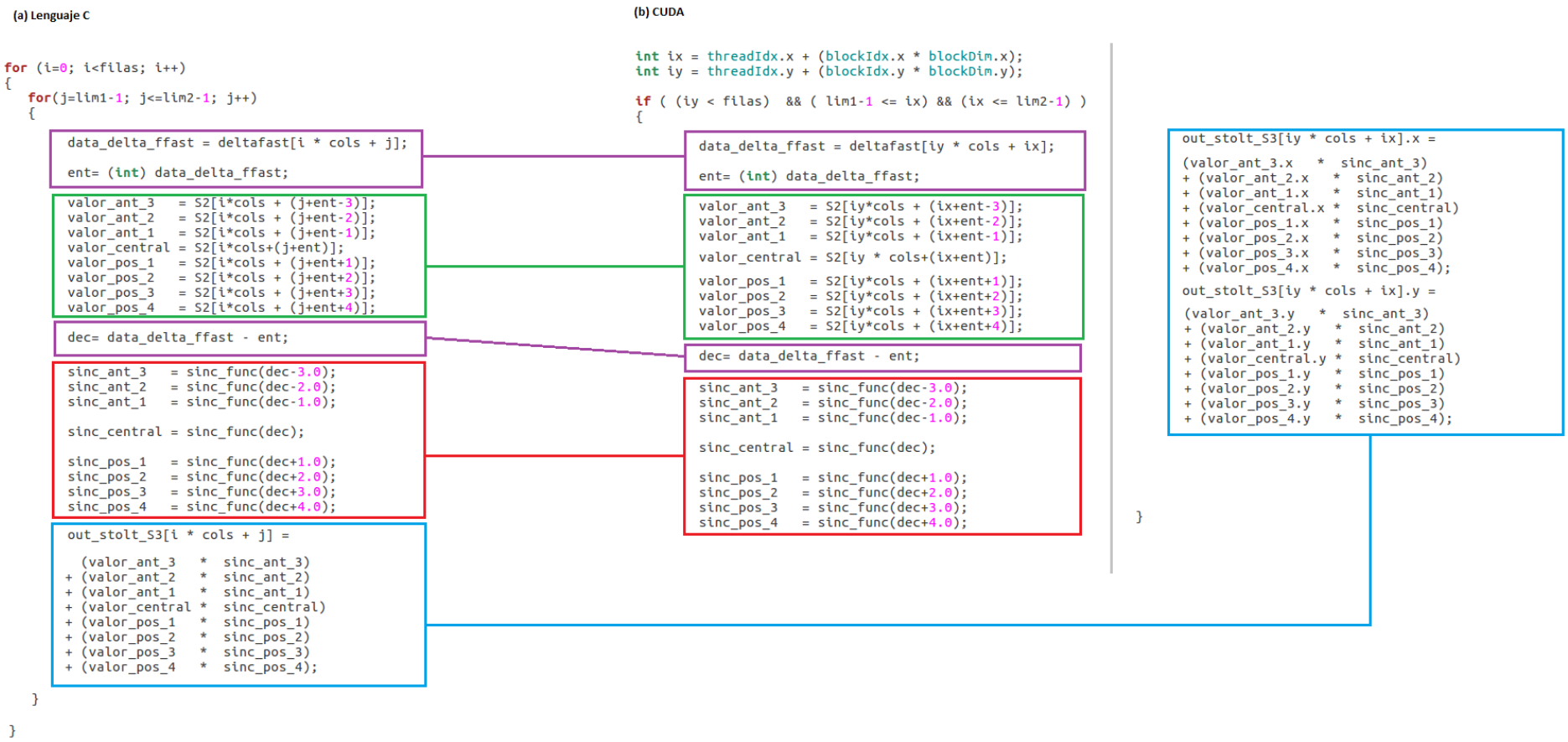


Figura 5.15 Trazabilidad entre C y CUDA de la interpolación de Stolt

CAPÍTULO 6 ~ RESULTADOS DE VALIDACÓN Y FOCALIZACIÓN

Este capítulo presenta los resultados de la aplicación de los criterios de validación introducidos en el Capítulo 3, para el caso de la dimensión en Rango y Acimut de forma separada.

6.1 Resultados de Validación entre versiones

Utilizando aquí la expresión presentada en la sección 4.4:

$$E = \left(\frac{abs(S4_m - S4_v)}{abs(S4_m)}\right) \tag{6.1.1}$$

6.1.1 MATLAB vs C

$$e_{max} = 2.93 \times 10^{-5} \tag{6.1.1.1}$$

Dado que el error relativo está en el orden de la cienmilésima, consideramos que las matrices focalizadas obtenidas mediante el algoritmo WK implementado en MATLAB y C son equivalentes.

6.1.2 MATLAB vs CUDA

$$e_{max} = 8.84 \times 10^{-5} \tag{6.1.1.2}$$

Dado que el error relativo está en el orden de la cienmilésima, consideramos que las matrices focalizadas obtenidas mediante el algoritmo WK implementado en MATLAB y CUDA son equivalentes.

6.2 Resultados de la focalización del WK

Las escalas de los gráficos a continuación están expresadas en número de muestras tanto en rango (eje vertical) como en acimut (eje horizontal). Pueden convertirse a unidades de distancia (m) multiplicando los ejes de rango y acimut por sus valores de Pixel Spacing, según se vio en la sección (2.3)

$$\text{Pixel Spacing en rango} = \frac{C}{2*f_r} \tag{6.2.1}$$

$$\text{Pixel Spacing en acimut} = \frac{V_r}{PRF} \tag{6.2.2}$$

En la (Figura 6.1) se aprecia el resultado de focalizar un blanco puntual en rango y acimut utilizando una matriz de datos crudos de 2048x1400 y para un rango de referencia R_{ref}=27000

Gráfica de la señal focalizada en Rango y Acimut

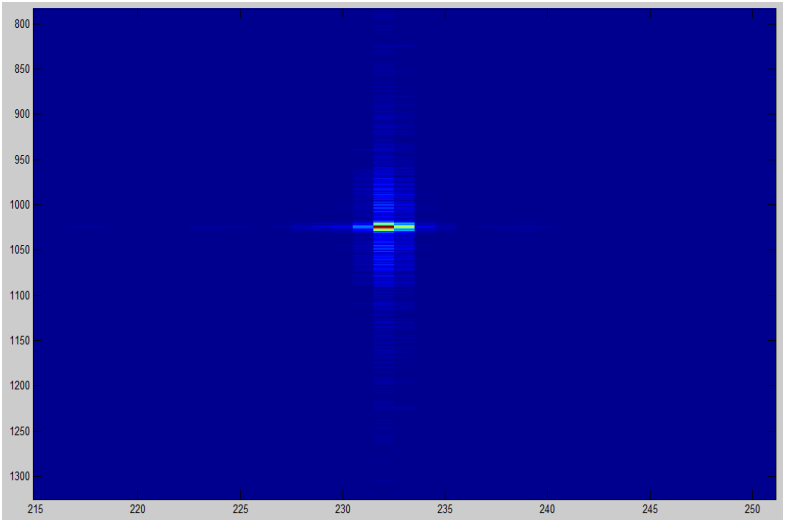


Figura 6.1 Vista del módulo de la señal focalizada $abs(S4)$ en Acimut,eje vertical y en Rango, eje horizontal.

A partir de aquí se pude comprobar (sección 6.3) que la energía del lóbulo principal ha quedado contenida dentro de 1 celda de resolución en rango, equivalente en metros a:

$$\begin{aligned}&\text{Resolución espacial en rango medida}\\&=>1*\text{pixel\_spacing\_rango}\\&=>\mathbf{1.25\ [m]}\end{aligned} \tag{6.2.3}$$

y en 2 celdas de resolución en acimut (sección 6.4), equivalente a:

$$\begin{aligned}&\text{Resolución espacial en Acimut medida}\\&=>2*\text{pixel\_spacing\_Acimut}\\&=>2*0.42\\&=>\mathbf{0.84\ [m]}\end{aligned} \tag{6.2.4}$$

Realizando un corte de la gráfica anterior en la fila de acimut del máximo, podemos visualizar el resultado de la focalización de un blanco puntual en rango.

Gráfica de la señal focalizada en Rango

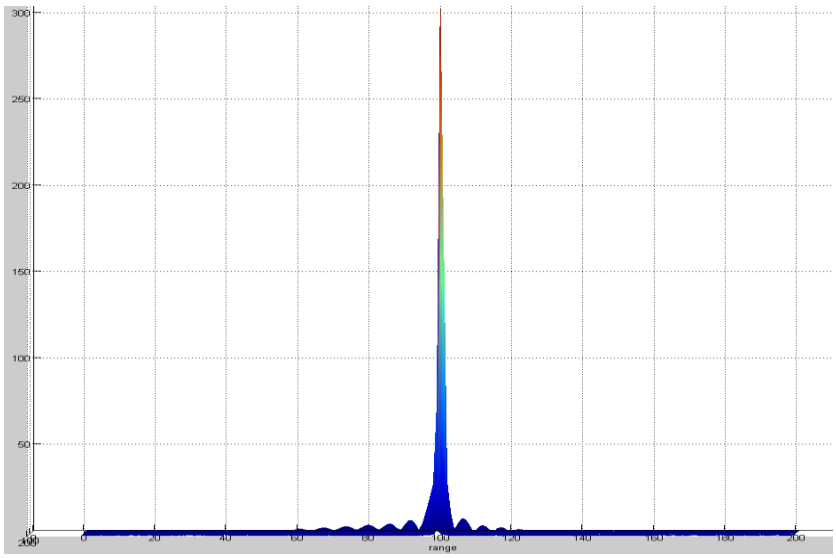


Figura 6.2 Eje vertical módulo de la señal focalizada $abs(S4)$
Eje horizontal: rango en nro. de muestras.

De igual forma, realizando un corte en la columna del rango correspondiente al máximo, podemos visualizar el resultado de la focalización de un blanco puntual en acimut.

Gráfica de la señal focalizada en Acimut

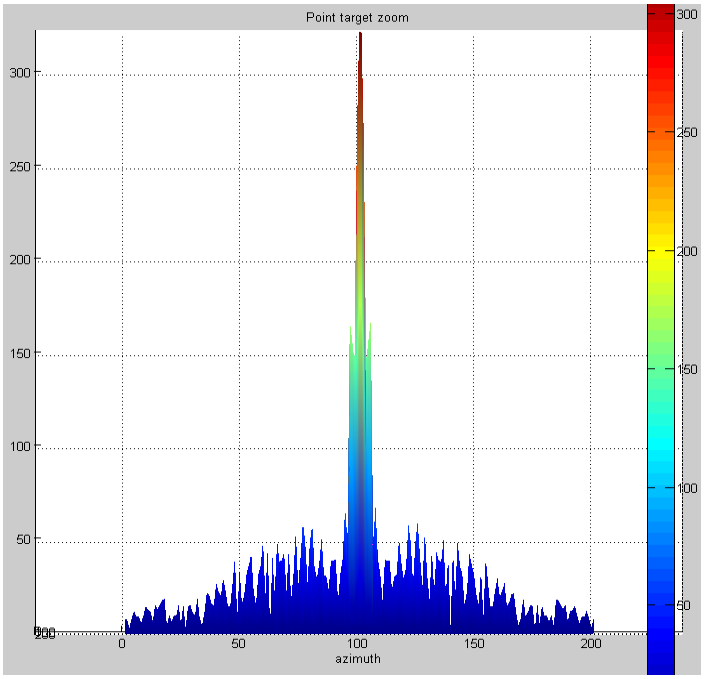


Figura 6.3 Eje vertical módulo de la señal focalizada $abs(S4)$ Eje horizontal: rango en nro. de muestras.

6.3 Resultados de Validación en Rango

Luego de transformar la señal a valores de potencia, se calcula el corte (caída) a -3dB a partir del máximo y se obtiene la cantidad de muestras:

Gráfica de Potencia

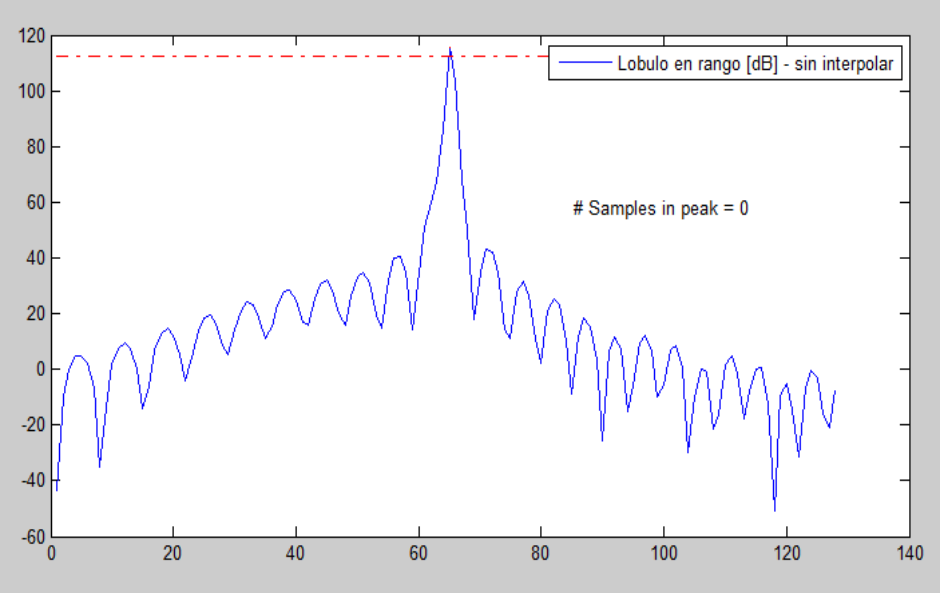


Figura 6.4 Eje vertical en dB calculado como: $20 * \log_{10}(abs(S4))$, donde $S4$ es la señal focalizada, Eje horizontal: rango en nro. de muestras. La línea roja indica el corte a -3dB.

Magnificando el corte alrededor del máximo podemos observar que 1 muestra supera el umbral de -3dB.

Fijado el acimut para la coordenada del máximo en acimut,

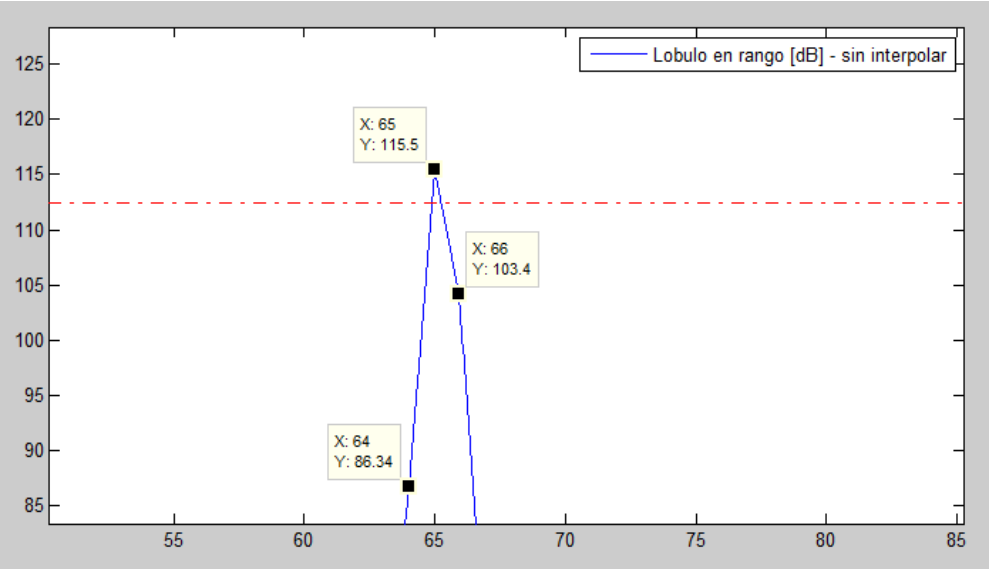


Figura 6.5 Eje vertical en dB calculado como: $20 * \log_{10}(abs(S4))$, donde $S4$ es la señal focalizada, Eje horizontal: rango en nro. de muestras. La línea roja indica el corte a -3dB.

Expresando en unidades de distancia, los valores comparados son:

Resolución teórica (sección 2.2): 1.50 [m] = 1.5 x 10^0 [m] (6.2.5)

Resolución medida (sección 6.2):
(1 muestra *PixelSpacing) = 1.25 [m]= 1.25 x 10^0 [m] (6.2.6)

Aplicación del Criterio de validación de la resolución espacial (sección 4.1):

La resolución espacial en rango obtenida se considera valida ya que posee igual orden de magnitud 10^0 que la resolución teórica.

6.3.1 Calidad de focalización en Rango: PSLR

Siguiendo el criterio de la PSLR se comparan los lóbulos secundarios con el pico principal y se observa un diferencia de aproximadamente (115.5 – 40.71) = 75 dB, que es mucho mayor a los 13dB que se tomó como criterio. Se concluye que satisface ampliamente el criterio.

Gráfica de Potencia

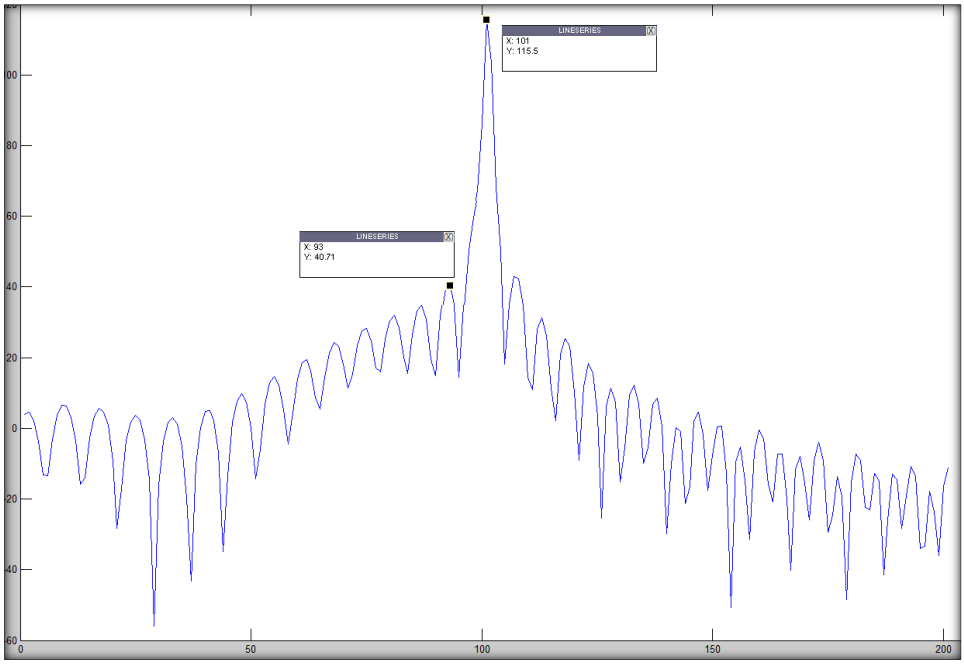


Figura 6.6 Eje vertical en dB calculado como: $20 * \log_{10}(abs(S4))$, donde $S4$ es la señal focalizada, Eje horizontal: rango en nro. de muestras.

6.4 Resultados de Validación en Acimut

Luego de transformar la señal a valores de potencia, se calcula el corte (caída) a -3dB en acimut a partir del máximo y se obtiene la cantidad de muestras:

Fijado el rango para la coordenada del máximo en rango,

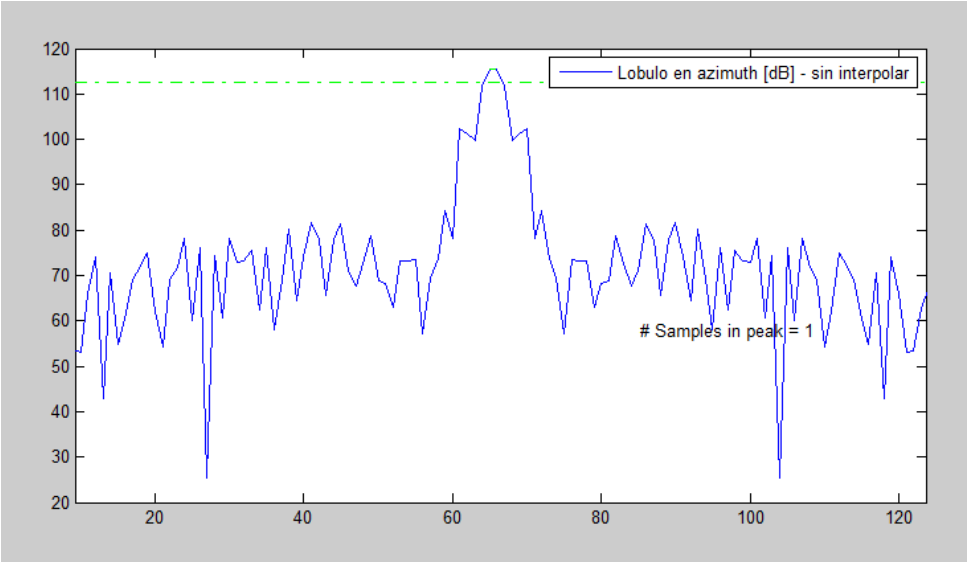


Figura 6.7 Eje vertical en dB calculado como: $20 * \log_{10}(abs(S4))$, donde $S4$ es la señal focalizada, Eje horizontal: rango en nro. de muestras. La línea verde indica el corte a -3dB.

Zoom del corte a -3dB podemos observar que 2 muestras superan el umbral de -3db:

Fijado el rango para la coordenada del primer máximo en rango,

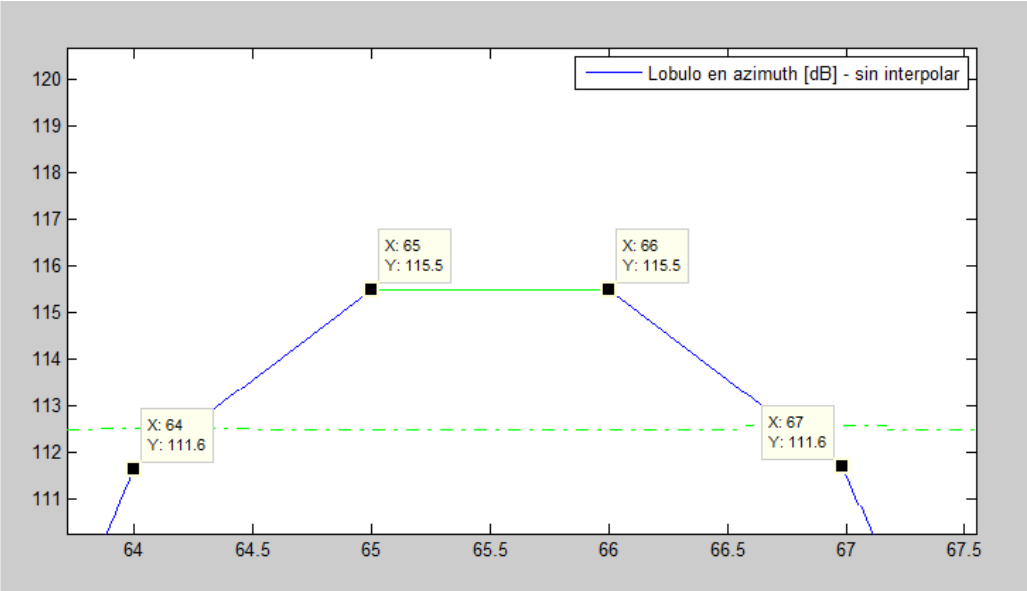


Figura 6.8 Eje vertical en dB calculado como: $20 * \log_{10}(abs(S4))$, donde $S4$ es la señal focalizada, Eje horizontal: rango en nro. de muestras. La línea verde indica el corte a -3dB.

Expresando en unidades de distancia, los valores comparados son:

Resolución teórica (sección 2.2): 0.50 [m] = 5.0 x 10^{-1}[m]

(6.2.7)

Resolución medida (sección 6.2):
(2 muestras*PixelSpacing) = 2*0.42 [m] = 0.84 [m] = 8.4 x 10^{-1}[m]

(6.2.8)

Aplicación del criterio de validación (sección 4.1):

La resolución espacial en acimut obtenida se considera válida ya que posee igual orden de magnitud 10^{-1} que la resolución teórica.

6.4.1 Calidad de focalización en Acimut: PLSR

Se puede observar que la calidad de focalización según el criterio de la PSLR en acimut es satisfactoria. Se registra una caída de -13dB (Cumming y Wong 2005): (115.5-102.5) = 13db

Fijado el rango para la coordenada del primer máximo en rango,

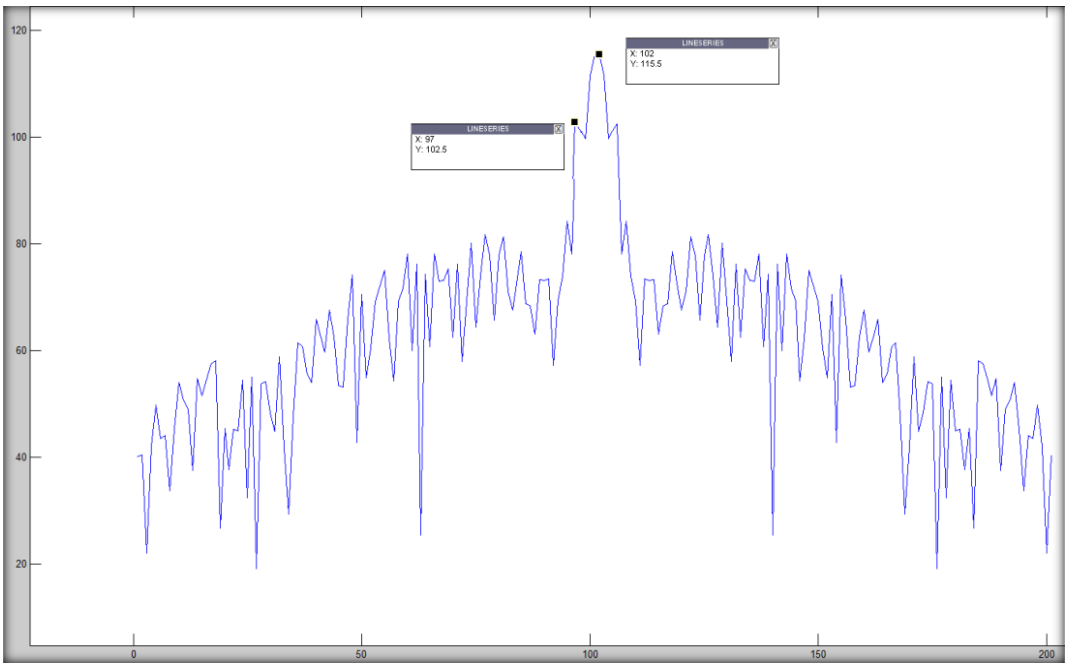


Figura 6.9 Eje vertical en dB calculado como: $20 * \log_{10}(abs(S4))$, donde $S4$ es la señal focalizada, Eje horizontal: rango en nro. de muestras.

CAPÍTULO 7 ~ RESULTADOS DE PERFORMANCE DEL SOFTWARE

Este capítulo presenta el resultado de la evaluación de la performance en tiempo para las versiones de WK implementadas en C y CUDA.

Los resultados se presentan de dos formas comparando los tiempos de ejecución:
Primero como un todo, y luego discriminando función por función.

7.1 Performance del Software

La siguiente imagen muestra las principales funciones de WK en C. Se observa como la interpolación de Stolt y la multiplicación por la función de referencia RFM representan más de la mitad del tiempo del algoritmo.

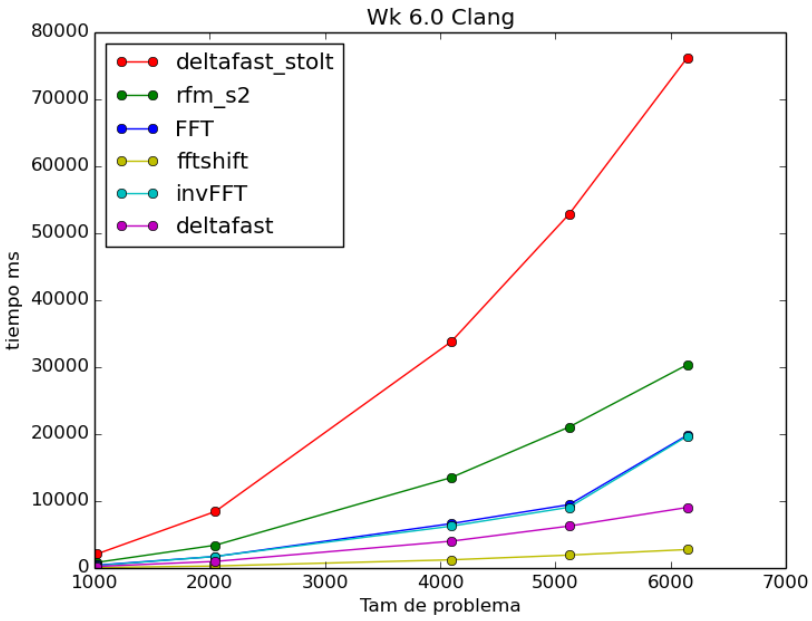


Figura 7.1 Performance en tiempo de las funciones de WK en C.

Al igual que el caso anterior también se puede observar que Stolt y RFM representan más de la mitad del tiempo utilizado por la versión en CUDA.

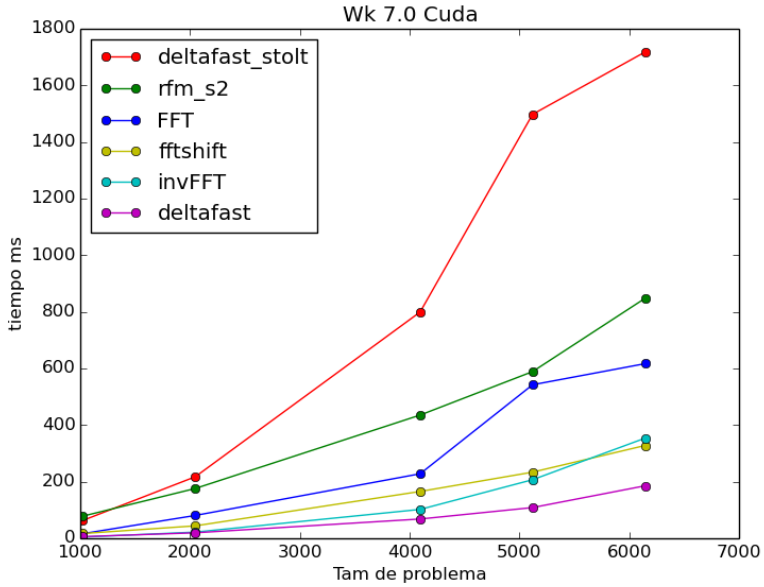


Figura 7.2 Performance en tiempo de las funciones de WK en CUDA.

7.2 Comparación entre C y Cuda

Las siguientes gráficas comparan los rendimientos en tiempo de las versiones completas de WK-C y WK-Cuda y muestran las aceleraciones obtenidas para los tamaños de problema entre 1024 y 6144. Quedan fuera de la medición las funciones de I/O que son compartidas entre ambos códigos, para la lectura de los datos crudos y la escritura de los datos focalizados a persistencia. Incluye los tiempos de las funciones de transferencias entre memorias del tipo *HostToDevice* y *DeviceToHost* entre CPU y GPU para el caso de la versión WK-Cuda.

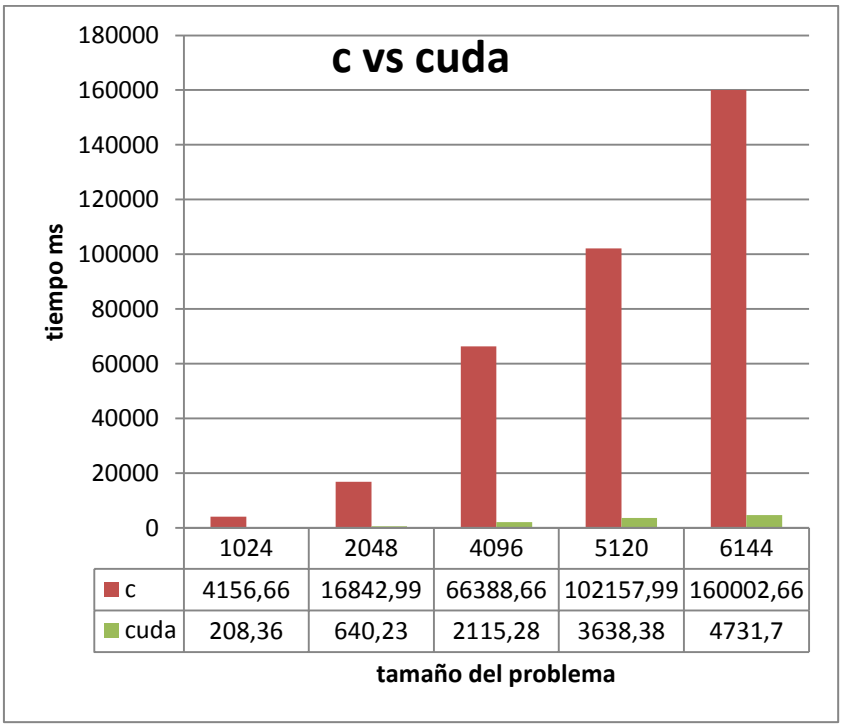


Figura 7.3 Comparación de tiempos de ejecución para cada tamaño de problema entre la versión de WK en CUDA y C.

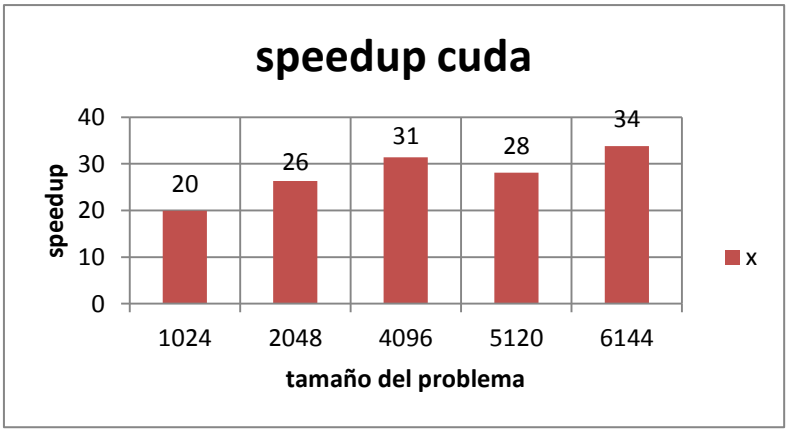
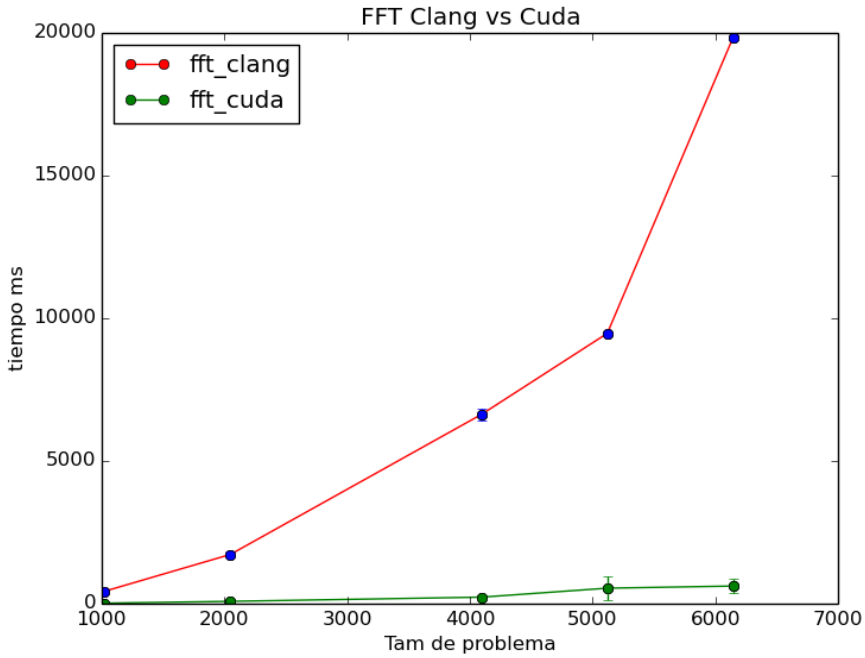


Figura 7.4 Aceleración lograda por la versión de WK en CUDA para cada tamaño de problema. 1X indica cuantas veces más rápida es la versión en CUDA para GPU respecto de la versión en C para CPU.

7.3 Comparación desagregando por las principales funciones de WK

7.3.1 Comparación FFT

La siguiente gráfica muestra la comparación en tiempo de la transformada rápida de Fourier en su implementación en C y CUDA. Se observa el máximo speedup de 32x para n=6144

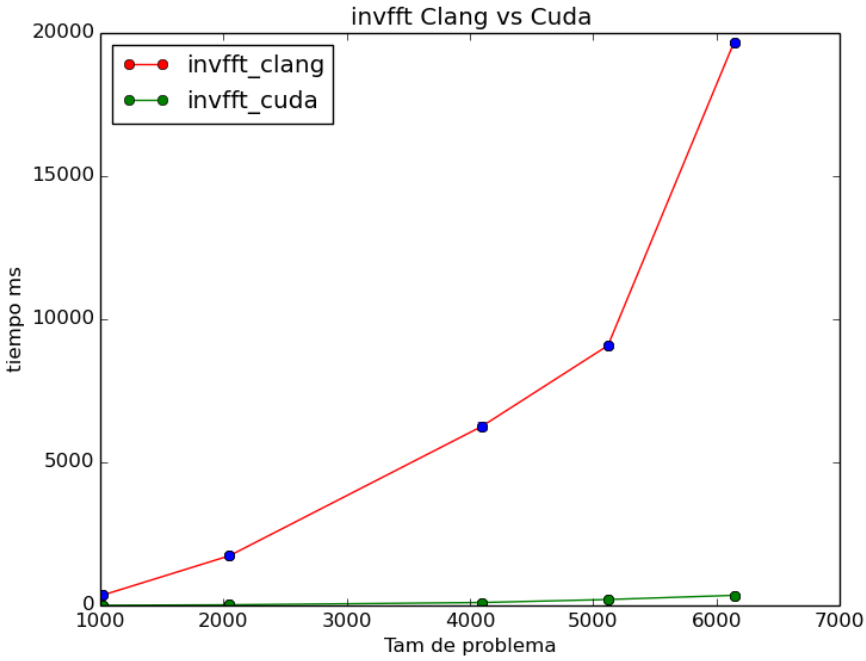


tamaño	1024	2048	4096	5120	6144
c	435,33	1724,33	6635	9473,33	19829,33
cuda	16,49	80,77	228,02	542,42	616,92
speedup	26	21	29	17	32

Figura 7.5 Comparación de la performance en tiempo para la Transformada rápida de Fourier entre la versión de WK en CUDA y C.

7.3.2 Comparación IFFT

La siguiente gráfica muestra la comparación en tiempo de la transformada inversa de Fourier en su implementación en C y CUDA. Se observa el máximo speedup de 56x para n=6144

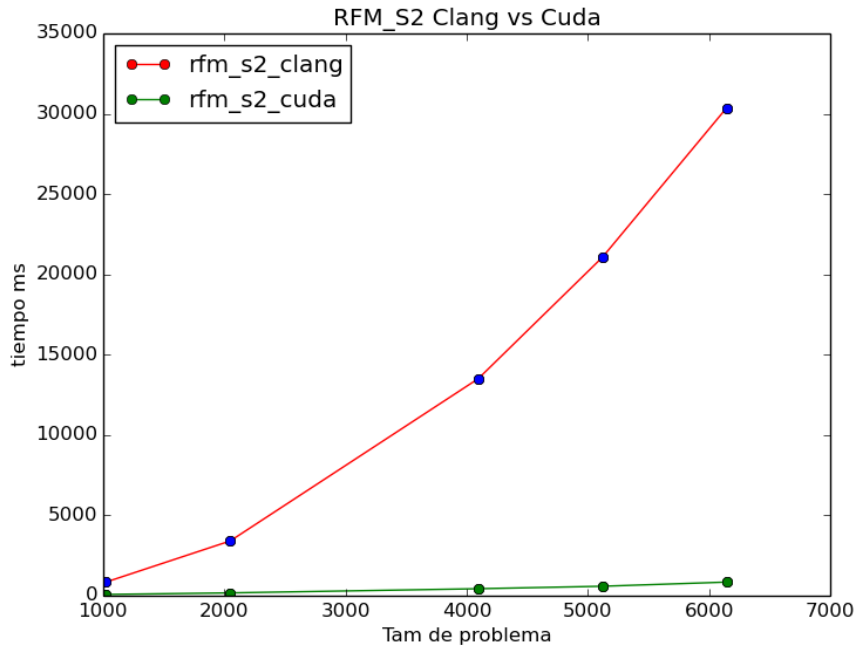


tamaño	1024	2048	4096	5120	6144
c	366	1735,33	6255	9080,67	19683
cuda	4,43	21,88	102,07	207	353,74
speedup	83	79	61	44	56

Figura 7.6 Comparación de la performance en tiempo para la Transformada rápida de Fourier Inversa entre la versión de WK en CUDA y C.

7.3.3 Comparación RFM

La siguiente gráfica muestra la comparación en tiempo de la Multiplicación por la Función de Referencia RFM en su implementación en C y CUDA. Se observa el máximo speedup de 36x para n=6144

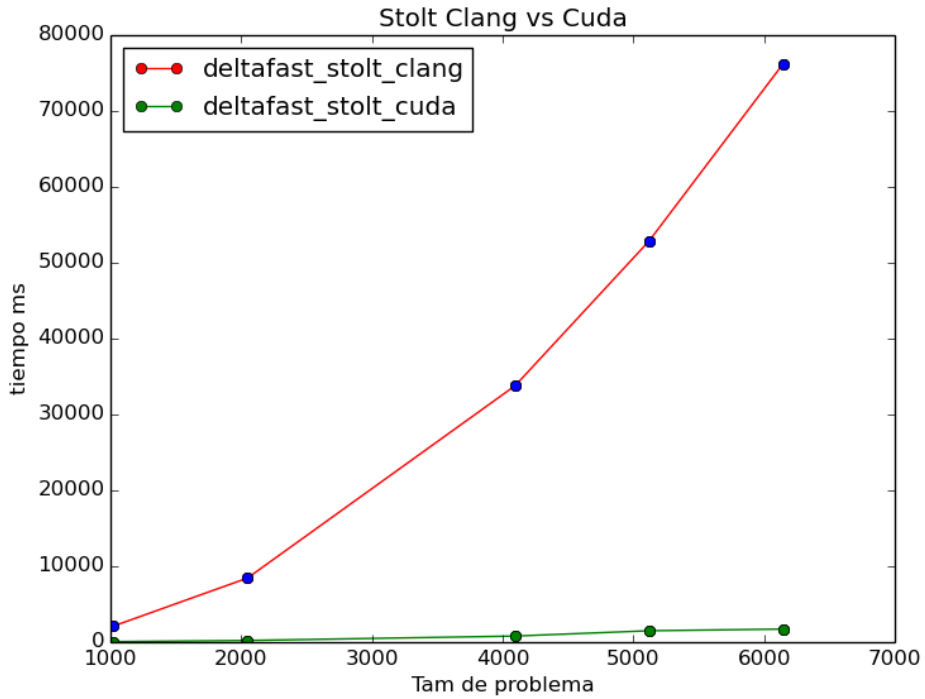


tamaño	1024	2048	4096	5120	6144
c	852,67	3410	13514,33	21067,67	30358,33
cuda	78,36	176,07	435,4	588,95	847,73
speedup	11	19	31	36	36

Figura 7.7 Comparación de la performance en tiempo para la Función RFM entre la versión de WK en CUDA y C.

7.3.4 Comparación Stolt

La siguiente gráfica muestra la comparación en tiempo de interpolación de Stolt (Mapeo e interpolación) utlizando un kernel (interpretación matemática) de 8 posiciones, en su implementación en C y CUDA. Se observa el máximo speedup de 44x para n=6144.

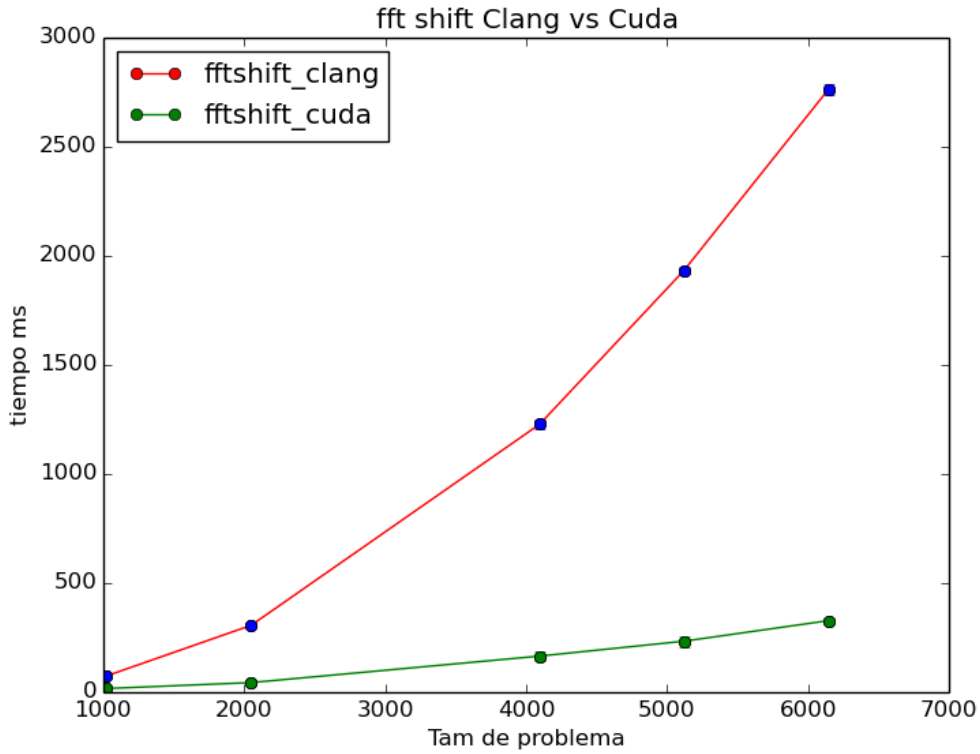


tamaño	1024	2048	4096	5120	6144
c	2121,33	8443	33801	52858	76185,67
cuda	63,18	217,48	799,55	1497,83	1717,85
speedup	34	39	42	35	44

Figura 7.8 Comparación de la performance en tiempo para los procesos combinados de Mapeo e Interpolación de Stolt entre la versión de WK en CUDA y C.

7.3.5 Comparación FFT-SHIFT

La siguiente gráfica muestra la comparación en tiempo de FFT-SHIFT, en su implementación en C y CUDA. Se observa el máximo speedup de 8x a partir de n=5120

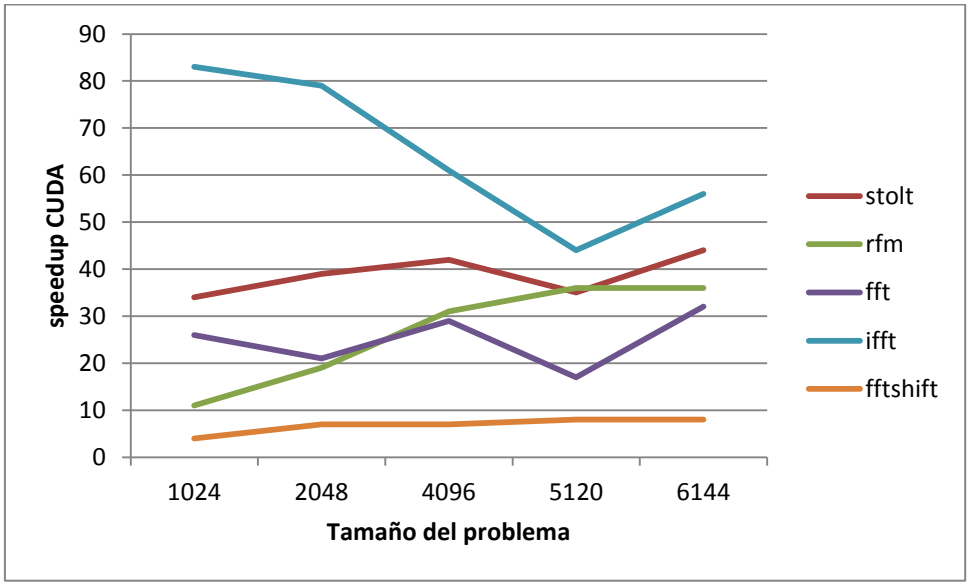


tamaño	1024	2048	4096	5120	6144
c	75	306,33	1228,33	1933	2762,33
cuda	16,88	44	165,51	233,77	327,84
speedup	4	7	7	8	8

Figura 7.9 Comparación de la performance en tiempo para la Función FFTSHIFT entre la versión de WK en CUDA y C.

7.4 Comparación de Speedup-CUDA entre funciones

Se puede observar las mayores aceleraciones en las funciones de Interpolación de Stolt entre 34x y 44x, y en la Transformada Inversa de Fourier alcanzando aceleraciones del orden de las 80x para n: 1024, 2048.

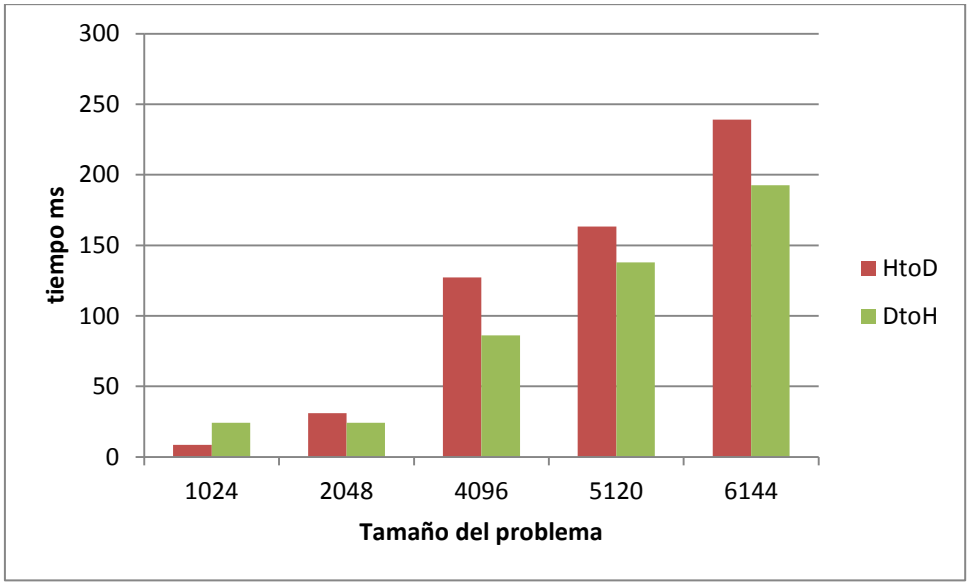


tamaño	1024	2048	4096	5120	6144
stolt	**34**	**39**	**42**	**35**	**44**
rfm	11	19	31	36	36
fft	26	21	29	17	32
ifft	**83**	**79**	**61**	**44**	**56**
fftshift	4	7	7	8	8

Figura 7.10 Comparación de la performance en tiempo entre las funciones de la versión de WK en CUDA.

7.5 Tiempos de transferencia entre memorias CPU-GPU

Por completitud se presenta a continuación un gráfico que muestra los tiempos de transferencias de memoria en dirección CPU->GPU y GPU->CPU para la versión de WK-CUDA.



tamaño	1024	2048	4096	5120	6144
HtoD	8,44	30,89	127,16	163,3	239,03
DtoH	24,22	24,22	86,07	137,89	192,46

Figura 7.11 Comparación de tiempos de transferencia entre las memorias de la CPU y GPU.

CAPÍTULO 8 ~ VERIFICACIÓN

En este capítulo se comprueba que todos los requerimientos del último nivel de agregación hayan sido desarrollados.

8.1 Matriz de Cumplimiento de Requerimientos

ID	Descripción	Padre	Trazabilidad / cumplimiento	Estado

L2A.1	El ingreso de datos deberá realizarse mediante un archivo de parámetros modificable por el usuario donde poder configurar las características físicas del sistema SAR	L1.2	• Capítulo 2 :
			  ○ sección 2.6, tabla 2.3
			  ○ sección 2.8, tabla
				cerrado

L2A.2	El desarrollo de la versión final deberá realizarse en una arquitectura hardware paralela CPU/GPU especifica: NVIDIA Jetson Tegra K1	L1.3	• Sección 5
			  ○ 5.2.3
			  ○ 5.2.4
			  ○ 5.7
			• Apéndice E
			• Apéndice D
				cerrado

L2A.3	Se deberá implementar una versión final del algoritmo WK Paralela en lenguaje CUDA para GPU	L1.3	• Sección 5
			  ○ 5.2.4
			  ○ 5.7
			• Apéndice E
				cerrado

L2A.4	Para la misma plataforma hardware seleccionada. Se deberá implementar una versión final del algoritmo WK secuencial en Lenguaje C para CPU, para comparar el rendimiento con la versión paralela	L1.3	• Sección 5
			  ○ 5.2.3
			  ○ 5.7
			• Apéndice D
				cerrado

L2A.5	El algoritmo WK deberá focalizar los datos crudos correspondientes a un blanco puntual, en una escena ideal de tierra plana y ángulo de squint nulo	L1.4	• Sección 6 ○ 6.2	cerrado
L2A.6	La implementación inicial del algoritmo WK deberá ser realizada en MATLAB para una PC estándar, para facilitar la comunicación con los usuarios expertos en el dominio del problema, obtener una comprensión completa de los detalles del algoritmo y lograr la aprobación para la construcción de las versiones finales.	L1.6	• Sección 5 ○ 5.3.1 • Apéndice C	cerrado
L2A.7	El producto de software deberá emitir una salida grafica de la señal focalizada	L1.6	• Capítulo 6 • Apéndice F	cerrado

L2B.1	La implementación del algoritmo WK deberá satisfacer el criterio de calidad denominado PLSR, el cual requiere que la distancia entre el pico principal tanto en rango como acimut a su lóbulo secundario más alto sea >= 13 dB	L1.1	• Sección 6
			  ○ 6.3.1
			  ○ 6.4.1
				cerrado

L2B.2	La diferencia en el resultado de la matriz de datos focalizados por las 3 versiones del algoritmo WK: MATLAB, C y CUDA no deben diferir en más de 10e-5	L1.1	• Apéndice H
			• Sección 6
			  ○ 6.1.1
			  ○ 6.1.2
				cerrado

L2B.3	El algoritmo WK deberá ser capaz de computar números complejos en doble precisión	L1.1	• Apéndice C • Apéndice D • Apéndice E	cerrado
L2B.4	La implementación final C/CUDA deberá leer como entrada un archivo de parámetros y los datos crudos en formato *.TXT CSV y dar una salida focalizada en formato *.TXT CSV	L1.1	• Apéndice C • Apéndice D • Apéndice E • Sección 5 ○ 5.5	cerrado

L2B.5	Se deberá implementar en MATLAB un simulador de datos crudos SAR para un Blanco puntual, cuya salida se pueda obtener en 2 formatos *.MAT y *.TXT CSV	L1.4	• Sección 5
			  ○ 5.2.1
			  ○ 5.2.2
			• Apéndice B
			• Sección 2
			  ○ 2.6,
			  ▪ Figura 2.7 Diseño
				cerrado

L2B.6	La resolución espacial en rango y acimut entregada por el algoritmo WK deberá satisfacer el orden de magnitud expresado la teoría SAR de acuerdo a las configuraciones en los parámetros del sistema	L1.4	• Apéndice G
			• Sección 6
			  ○ 6.3
			  ▪ Ec. 6.2.5
			  ▪ Ec. 6.2.6
			  ○ 6.4
			  ▪ Ec. 6.2.7
				cerrado

			▪ Ec. 6.2.8	
L2B.7	Función por función, se requiere una aceleración de al menos 2x entre la versión secuencial y paralela de WK	L1.5	• Capítulo 7	cerrado
L2B.8	La implementación inicial en MATLAB deberá leer como entrada un archivo de parámetros y los datos crudos en formato *.MAT y dar una salida focalizada en formato *.TXT CSV	L1.6	• Apéndice C • Apéndice i	cerrado

CAPÍTULO 9 ~ CONCLUSIONES

Se construyeron tres versiones del algoritmo WK, en MATLAB y C para CPU y en CUDA para GPU, todas consistentes entre sí. Se evaluaron sus capacidades de focalización para un blanco puntual respecto de los valores teóricos de resolución espacial y se encontró que estas implementaciones de WK focalizan eficientemente en la dimensión del rango y acimut, dentro del orden de magnitud predicho por la teoría. La aceleración de la versión paralela respecto de la versión secuencial en C es mayor a 20x a partir del tamaño de problema n=1024 llegando hasta 34x para n=6144, lo cual justifica el esfuerzo de paralelización.

En la comparación desagregada de función por función se alcanzaron aceleraciones del orden de 80x para la IFFT para n=1024 y 2048.

Se pudo comprobar de forma práctica que el algoritmo WK presenta características de dependencia al rango de referencia utilizado y que el valor correspondiente al rango del centro de la escena es óptimo.

La memoria global disponible en la GPU del Tegra K1 es limitada a 2GB, por lo tanto en esta primera aproximación se pudieron probar los tamaños de problema desde 1024 a 6144. Para tamaños mayores se requiere mayor esfuerzo y complejidad para particionar las principales operaciones del algoritmo de modo que no saturen la memoria, si bien cada partición podrían ser ejecutadas en paralelo, forzosamente entre una partición y la siguiente habrá secuencialidad mientras nos mantengamos en el seno de una única placa JetsonTegra k1 Dev Kit. Claro está que la ejecución particular de cada partición es paralelizable. Para evitar la secuencialiad de la ejecución de las particiones se debería pasar a un esquema de clúster de Jetson Tegra k1 utilizando OpenMPI+CUDA, lo cual queda relegado para futuros trabajos.

TRABAJO FUTURO

Como trabajo futuro se plantea seguir investigando sobre la focalización en acimut y su relación de dependencia con el parámetro: "*Rango de referencia*" desde el punto de vista teórico y validando con la implementación en MATLAB.

Hacer pruebas para un conjunto de objetivos puntuales simulados y para una escena completa simulada.

Efectuar pruebas con datos crudos reales.

Obtener versiones que implementen particionamiento de FFTSHIFT, FFT2D, RFM, y STOLT para la ejecución en 1 sola placa Tegra K1 para tamaños de problema a partir de n=6144. Analizar para cada caso y por función valores de potencia, tiempo y memoria.

Basado en técnicas de particionamiento y computación distribuida se desea lograr a futuro una versión híbrida OpenMP + OpenMPI del algoritmo para ser ejecutado en un clúster de placas Jetson Tegra-K1 Dev Kit utilizando únicamente sus CPUs ARM cortex A-15.

Construir una versión OpenMPI + CUDA para ser ejecutado en un clúster de placas JetsonTegra-K1 Dev Kit utilizando únicamente sus GPUs, y comparar el rendimiento en tiempo y potencia de lo obtenido en el clúster de CPU.

BIBLIOGRAFÍA

1. C. Cafforio, C. Prati and C. Rocca, F. (1991).SAR data focusing using seismic migration techniques. IEEE Transactions on Aerospace and Electronic Systems, 27(2), 194-207. doi:10.1109/7.78293.
2. Cumming and Wong. (2005). Digital processing of synthetic aperture radar data: Algorithms and implementation. Boston: Artech House.
3. Cheng, Grossman and McKercher. (2014). Professional CUDA C programming. Indianapolis, IN: Wrox.
4. Curlander and McDonough. (1991). Synthetic aperture radar: Systems and signal processing. New York: Wiley.
5. Bu-Chin Wang, Digital Signal Processing Techniques and Applications in Radar Image Processing, 2008
6. MehrdadSoumekh, Synthetic Aperture Radar Signal Processing with MATLAB Algorithms, Wiley, April 1999
7. Kirk &Hwu. (2010). Programming massively parallel processors: A hands-on approach. Burlington, MA: Morgan Kaufmann.
8. Denham, M., Areta, J., &Tinetti, F. G. (2015). Synthetic aperture radar signal processing in parallel using GPGPU. The Journal of Supercomputing, 72(2), 451-467. doi:10.1007/s11227-015-1572-z
9. Mónica Denham; Javier Areta; Isidoro Vaquila; Fernando G. Tinetti. (2013). Procesamiento de Señales SAR: Algoritmo RDA para GPGPU. XIX Congreso Argentino de Ciencias de la Computación. Universidad CAECE (Mar del Plata, Argentina) Red UNCI.
10. Denham Mónica; Javier Areta; Isidoro Vaquila; Fernando G. Tinetti. (2014) SyntheticAperture Radar SignalProcessingusing GPGPU. Congreso Latinoamericano de HPC CARLA 2014. CCTVal y NLHPC. Valparaíso, Chile.
11. Santiago Abbate, Joaquín González, Silvina Gutierrez, Mónica Denham. (2015). Extended Chirp Scaling Algorithm en GPGPU. XVI Workshop on Information Processing and Control.
12. Hubert M.J. Cantalloube. (2012). Real-time Airborne SAR Imaging. Motion compensation and Autofocus issues, EUSAR. 9th European Conference, Nuremberg, Alemania.

13. Fatica, M., & Phillips, E. (2014). Synthetic aperture radar imaging on a CUDA-enabled mobile platform. IEEE High Performance Extreme Computing Conference (HPEC). doi:10.1109/hpec.2014.7040960
14. Tiriticco, Et Al, (2014). Near Real Time, Multi GPU
15. Carrara, W.G., Goodman, R.S., Majewski, R. M., (1995). Spotlight synthetic aperture radar: Signal processing algorithms. *London, Artech House*.
16. Franceschetti, G., & Lanari, R. (1999). *Synthetic aperture radar processing*. CRC press.
17. Henderson, F. M., & Lewis, A. J. (1998). Principles and applications of imaging radar. Manual of remote sensing: Volume 2.
18. Cumming, I. G., Neo, Y. L., & Wong, F. H. (2003, July). Interpretations of the Omega-K algorithm and comparisons with other algorithms. In Geoscience and Remote Sensing Symposium, 2003. IGARSS'03. Proceedings. 2003 IEEE International (Vol. 3, pp. 1455-1458). IEEE.
19. A. V. Oppenheim y R. W. Schafer. (1999). Discrete-Time Signal Processing. Prentice-Hall, Inc., segunda edición.

Apéndice A ~ Características del Tegra K1

A continuación se muestra la salida de la utilidad *deviceQuery* de NVIDA corriendo en el Tegra K1 para obtener las características de la GPU:

```
1   ./deviceQuery
2   Starting...
3
4   CUDA Device Query (Runtime API) version (CUDART static linking)
5   Detected 1 CUDA Capable device(s)
6
7   Device 0: "GK20A"
8     CUDA Driver Version / Runtime Version:6.5 / 6.5
9     CUDA Capability Major/Minor version number: 3.2
10    Total amount of global memory: 1892 MBytes (1984397312 bytes)
11    (1) Multiprocessors, (192) CUDA Cores/MP: 192 CUDA Cores
12    GPU Clock rate:  852 MHz (0.85 GHz)
13    Memory Clock rate: 924 Mhz
14    Memory Bus Width:  64-bit
15    L2 Cache Size: 131072 bytes
16    Maximum Texture Dimension Size (x,y,z)  1D=(65536), 2D=(65536,
17    65536), 3D=(4096, 4096, 4096)
18    Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048
19    layers
20    Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384),
21    2048layers
22    Total amount of constant memory:  65536 bytes
23    Total amount of shared memory per block: 49152 bytes
24    Total number of registers available per block: 32768
25    Warp size:  32
26    Maximum number of threads per multiprocessor:  2048
27    Maximum number of threads per block: 1024
28    Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
29    Max dimension size of a grid size    (x,y,z): (2147483647, 65535,
30    65535)
31    Maximum memory pitch: 2147483647 bytes
32    Texture alignment:  512 bytes
33    Concurrent copy and kernel execution: Yes with 1 copy engine(s)
34    Run time limit on kernels:  No
35    Integrated GPU sharing Host Memory:  Yes
36    Support host page-locked memory mapping: Yes
37    Alignment requirement for Surfaces:  Yes
38    Device has ECC support:  Disabled
```

```
39      Device supports Unified Addressing (UVA): Yes
40      Device PCI Bus ID / PCI location ID: 0 / 0
41      Compute Mode:
42         < Default (multiple host threads can use ::cudaSetDevice() with
43    device simultaneously) >
44    deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.5, CUDA
45    Runtime Version = 6.5, NumDevs = 1, Device0 = GK20A
46    Result = PA
```

Apéndice B ~ Simulador SAR: Blanco Puntual -MATLAB

```
parametrosMKS

Ts_tau = 1/p.Fr;% período de maestro en rango[s]
Ts_eta = 1/p.PRF;% período de maestro en acimut [s]

Neta = 2048;%numero de muestras en acimut
Ntau = 4096;%numero de muestras en rango

% Vectores de Tiempo

% Azimuth measurement time
Eta_meas =linspace(-Neta/2,Neta/2,Neta)*Ts_eta;
Eta      =  Eta_meas + p.eta_c;

% Rango
Tau      =[0:Ntau-1]*Ts_tau + 2*p.Rca/p.C;

% vector Rango
Rv          =sqrt(p.Rt.^2 +(p.Vr * Eta).^2);
nReq        =floor((Rv-p.Rca)/(Ts_tau*p.C));

% ventana en tiempo: Kaiser w/ beta=2.5
beta=2.5;
Nwr  =floor(p.Tr/Ts_tau);
wr   = kaiser(Nwr,beta);

% ventana en acimut
BeamWidth = p.lambda/p.La;
theta     =atan(p.Vr/p.Rca*(Eta-p.eta_c));
wa        = sinc(theta/BeamWidth).^2;
```

```
rtaSAR =zeros(Neta,Ntau);

clear theta
clear Eta
clear Eta_meas


for net=1:Neta

        rangeIx = nReq(net)+[0:Nwr-1];

        rtaSAR(net,rangeIx)= wr'.*exp(-1i * 4 *pi* p.f0 * Rv(net)/p.C)
                    .*exp(1i*pi*p.Kr*(Tau(rangeIx)-2*Rv(net)/p.C).^2);

end

clear Tau
clear nReq
clear Rv
clear wr
% se aplica la ventana en acimut (patron de antena)
rtaSAR =repmat(wa',1,Ntau).* rtaSAR;

save('datosCrudosSinConv_2048_4096.mat','rtaSAR');
```

Apéndice C ~ WK-MATLAB

```
format long g

j=1i;

Ks =10e12;% Chirp Frequency Hz/s =Kr
C =3e8;% Spped of light m/s
V = 250;% Set the platform movement speed (m / s)
Fc =9.4e9;% Set carrier frequency (Hz)

Fslow = 600;% slow time sampling rate,
Ffast =120e6;% fast time sampling rate,

load datosCrudosSinConv_2048_4096.mat;

S=rtaSAR;
clear rtaSAR;

[Nslow,Nfast]=size(S);

% 1. 2D FFT
S11  = ifftshift(S);
S12 =fft2(S11);
clear S11
S1 = fftshift(S12);
clear S12
R0 = 25500;
```

```
% 2. Multiplicación por la Función de Referencia

fslow=linspace(-Fslow/2, Fslow/2 - Fslow/Nslow, Nslow);
ffast=linspace(-Ffast/2, Ffast/2 - Ffast/Nfast, Nfast);

c1=(C^2/(4*V^2))
c2=pi/Ks
c3 =4*pi*R0/C

res= c3*sqrt(repmat((Fc+ffast).^2,Nslow,1)- c1*repmat(fslow'.^2,1,Nfast))+ c2*repmat(ffast.^2,Nslow,1);

res =mod(res,2*pi);
F_ref =cos(res)+sin(res)*j;

S2 = S1 .* F_ref;

clear S1
clear F_ref


% 3.Función Mapeo de Stolt

ffast_new =sqrt(ones(Nslow,1)*(Fc + ffast).^2 -(C^2*fslow.^2/4/V^2).'*ones(1,Nfast));

ffast_new = ffast_new * Nfast/Ffast;
```

```
% 4. Función de Interpolación de Stolt

        Nd =fix(max(max(ffast_new)));
        Nm =fix(min(min(ffast_new)));

        S3 =zeros(Nslow,Nfast);
        Ni = 8/2;
        h = waitbar(0,'Stolt');

        maxNdm=max(abs(Nd),abs(Nm))
        lim1=Ni+maxNdm
        lim2=Nfast-lim1

        for q = 1 : 1 : Nslow

                  data = S2(q,:);
                  for j =lim1 : 1 : lim2

                        Insert_AMT =  ffast_new(q,j);
                        AMT = Insert_AMT;
                        AMT_N =fix(AMT);
                        AMT_F = AMT - AMT_N;
                        NN = AMT_N +(-Ni+1:Ni);
                        NN1 = AMT_F +(-Ni+1:Ni);
                        sinc_kernel = sinc(NN1);
                        S3(q,j)= sinc_kernel * data(j+NN).';
                  End

                  waitbar(q/Nslow)
      end
```

```
83          clear S2
84          clear Delta_ffast
85          S4_11 = ifftshift(S3);
86          clear S3
87          S4_12 =ifft2(S4_11);
88          clear S4_11
89          S4 = fftshift(S4_12);
90          clear S4_12
```

Apéndice D ~ WK-C

```
1   #include <stdlib.h>
2   #include <stdio.h>
3
4   #include <complex.h>
5   #include <fftw3.h>
6   #include <math.h>
7
8
9
10  #define Fc 9.4e9
11  #define C 3.0e8
12  #define V 250.0
13
14  #define R0 25500.0
15  #define Ks 10.0e12
16
17  //#define Fslow 600
18  #define Ffast 120.0e6
19  #define Fslow  6.0e2
20
21  #define Nslow 4096
22  #define Nfast 4096
23
24  #define Ni 4
25
26  #define PRINT 0
27  #define PI 3.1416
28  #define MAX(a,b)((a>b)?a:b)
29
30  //-------kerneles ------------
```

```
31  double dFc;
32  double dRfmC3;
33  double dRfmC2;
34  double dRfmC1;
35
36  double dFfast;
37
38  int dLim;
39
40  double sinc_func(double x)
41  {
42          return sin(PI*x+1e-32)/(PI*x+1e-32);
43  }
```

```
44  //unicamente matrices cuadradas y pares
45  void fftshift(double complex *d_matrix,int filas,int cols)
46  {
47          int fila, col;
48          double complex aux;
49
50          for(fila=0; fila<filas; fila++)
51          {
52                  for(col=0; col<cols; col++)
53                  {
54                                  if(fila<filas/2)
55                                  {
56                                          if(col<cols/2)
57                                          {
58
59
60                                                  aux = d_matrix[fila*cols + col];
61                                                  d_matrix[fila*cols + col]= d_matrix[(fila+(filas/2))*cols+(col+(cols/2))];
62                                                  d_matrix[(fila+(filas/2))*cols +(col+(cols/2))]= aux;
63                                          }
64                                          else
65                                          {
66                                                  aux = d_matrix[fila*cols + col];
67                                                  d_matrix[fila*cols + col]= d_matrix[(fila+(filas/2))*cols +(col-(cols/2))];
68                                                  d_matrix[(fila+(filas/2))*cols +(col-(cols/2))]= aux ;
69                                          }
70                                  }
71
72
73                  }//del 2do for
74          }//del 1er for
75
76  }
```

```
77  void normalizar_ifft (double complex *odata,int nx,int ny)
78  {
79
80          int i,j;
81
82          for(i=0; i<nx; i++)
83          {
84                  for(j=0; j<ny; j++)
85                  {
86                          odata[i*nx+j]=  odata[i*nx+j]/(double)( nx * ny );//nro complejo
87
88                  }
89
90          }
91
92
93  }
```

```
94  //creación de la función de referencia
95  void rfm (double complex *out_rfm,double*fslow,int nx,double*ffast,int ny){
96
97          double d2PI =6.2832;
98          double res =0.0;
99          double root =0.0;
100
101         int i,j;
102
103         for(i=0; i<nx; i++)
104         {
105                 for(j=0; j<ny; j++)
106                 {
107
108                         root=(double)((dFc+ffast[j])*(dFc+ffast[j])- dRfmC1 *(fslow[i]*fslow[i]));
109
110                         res = dRfmC3 *sqrt( root  )+ dRfmC2 *(ffast[j]*ffast[j]);
111
112                         res =fmod(res, d2PI);
113
114                         out_rfm[i*ny+j]=cos(res)+sin(res)*I;
115                 }
116         }
117
118 }
```

```
119  //aplica la RFM a los datos
120  void S2 (double complex *out_S2,double complex *S1,double complex *rfm,int nx,int ny){
121
122          int i,j;
123          //double s1x=0.0;
124          //double s1y=0.0;
125
126          for(i=0; i<nx; i++)
127          {
128                  for(j=0; j<ny; j++)
129                  {
130                          //operador "* complejo"
131                          out_S2[i*ny+j]= S1[i*ny+j]* rfm[i*ny+j];
132                  }
133          }
134
135  }
```

```
136  //creación de la nueva variable para Stolt
137  void delta_ffast (double*out_delta_ffast,double*fslow,int nx,double*ffast,int ny){
138
139          double root =0.0;
140          double res =0.0;
141          int i,j;
142
143          for(i=0; i<nx; i++)
144          {
145                  for(j=0; j<ny; j++)
146                  {
147
148                          root =(double)((dFc+ffast[j])*(dFc+ffast[j])- dRfmC1 *(fslow[i]*fslow[i]));
149
150                          res =sqrt( root )-(dFc+ffast[j]);
151
152                          res = res*(ny/dFfast);
153
154                          out_delta_ffast[i*ny+j]=res;
155                  }
156
157          }
158  }
159
160  double complex make_Complex(double re,double im)
161  {
162
163          double complex n = re + im*I;
164          return n;
165
166  }
```

```
167  void stolt (double complex *out_stolt_S3,double*deltafast,double complex *S2,int filas,int cols){
168
169        double data_delta_ffast =0.0f;
170
171
172        double complex valor_ant_3   = make_Complex(0.0f,0.0f);
173        double complex valor_ant_2   = make_Complex(0.0f,0.0f);
174        double complex valor_ant_1   = make_Complex(0.0f,0.0f);
175        double complex valor_central = make_Complex(0.0f,0.0f);
176        double complex valor_pos_1   = make_Complex(0.0f,0.0f);
177        double complex valor_pos_2   = make_Complex(0.0f,0.0f);
178        double complex valor_pos_3   = make_Complex(0.0f,0.0f);
179        double complex valor_pos_4   = make_Complex(0.0f,0.0f);
180
181
182        int ent=0;
183        double dec=0.0f;
184
185        double sinc_ant_3 =0.0f;
186        double sinc_ant_2 =0.0f;
187        double sinc_ant_1 =0.0f;
188        double sinc_central=0.0f;
189        double sinc_pos_1 =0.0f;
190        double sinc_pos_2 =0.0f;
191        double sinc_pos_3 =0.0f;
192        double sinc_pos_4 =0.0f;
193
194        int lim1 = Ni+dLim;
195        int lim2 = cols-lim1;
196
197        int i,j;
```

```
198         for(i=0; i<filas; i++)
199         {
200                 for(j=lim1-1; j<=lim2-1; j++)
201                 {
202
203                         //valor double
204                         data_delta_ffast = deltafast[i * cols + j];
205
206                         //toma parte enterea del double
207                         ent=(int) data_delta_ffast;
208
209                         //asignacion de double complex
210                         valor_ant_3   = S2[i*cols +(j+ent-3)];
211                         valor_ant_2   = S2[i*cols +(j+ent-2)];
212                         valor_ant_1   = S2[i*cols +(j+ent-1)];
213
214                         valor_central = S2[i*cols+(j+ent)];
215
216                         valor pos 1   = S2[i*cols +(j+ent+1)];
217                         valor_pos_2   = S2[i*cols +(j+ent+2)];
218                         valor_pos_3   = S2[i*cols +(j+ent+3)];
219                         valor_pos_4   = S2[i*cols +(j+ent+4)];
220
221
222                         dec= data_delta_ffast - ent;
223
224
225                         sinc_ant_3   = sinc_func(dec-3.0);
226                         sinc_ant_2   = sinc_func(dec-2.0);
227                         sinc_ant_1   = sinc_func(dec-1.0);
228
229                         sinc_central = sinc_func(dec);
230
231                         sinc_pos_1   = sinc_func(dec+1.0);
232                         sinc_pos_2   = sinc_func(dec+2.0);
233                         sinc_pos_3   = sinc_func(dec+3.0);
234                         sinc_pos_4   = sinc_func(dec+4.0);
235
```

```
236                         out_stolt_S3[i * cols + j]=(valor_ant_3   *  sinc_ant_3)
237                                                   +(valor_ant_2   *  sinc_ant_2)
238                                                   +(valor_ant_1   *  sinc_ant_1)
239                                                   +(valor_central *  sinc_central)
240                                                   +(valor_pos_1   *  sinc_pos_1)
241                                                   +(valor_pos_2   *  sinc_pos_2)
242                                                   +(valor_pos_3   *  sinc_pos_3)
243                                                   +(valor_pos_4   *  sinc_pos_4);
244
245                 }//del 2do for
246         }//del 1er for
247
248
249 }
```

```
250   //------------------ Utils --------------------------
251
252   void initMat(double complex* h_data,int nx,int ny)
253   {
254         int i,j;
255         for(i =0; i < nx; i++)
256           for(j =0; j < ny; j++)
257                   h_data[(i * ny)+ j]=0;
258         return;
259   }
```

```
260  void getMaxMin(double*in,int nx,int ny,double*max,double*min){
261
262        int i,j;
263        int k=0, l=0;
264
265        for(i =0; i < nx; i++)
266        {
267              for( j =0; j <= ny/2; j++)
268              {
269              l=(i*ny)+ j;
270              k=(i*ny)+(ny-j-1);
271
272                    if(in[l]> in[k])
273                    {
274                          if(*min> in[k])
275                                *min= in[k];
276
277                          if(*max< in[l])
278                                *max= in[l];
279                     }
280                     else
281                     {
282                          if(*min> in[l])
283                                *min= in[l];
284                          if(*max< in[k])
285                                *max= in[k];
286                     }
287               }
288        }
289
290        return;
291
292  }
```

```
293  int linspace (double* vector,double minval,double maxval,int n)
294  {
295
296          if(n <2){
297                  return0;
298          }
299
300          int i =0;
301          double step =(maxval-minval)/(floor((double)n)-1.0);
302
303          for(i =0; i < n; i++)
304          {
305              vector[i]= minval + i*step;
306
307          }
308          return 1;
309
310  }
```

```
311  int readMat2(char arch_re[],char arch_im[], complex* h_mat,int nx,int ny)
312  {
313
314        int i,j;
315        double re;
316        double imag;
317
318
319        FILE*pfile_re,*pfile_im;
320        pfile_re =fopen(arch_re,"r");
321        pfile_im =fopen(arch_im,"r");
322
323        if(pfile_re ==NULL)
324        {
325          printf("error al abrir archivo para lectura %s\n", arch_re);
326          return(-1);
327        }
328        if(pfile_im ==NULL)
329        {
330          printf("error al abrir archivo para lectura %s\n", arch_im);
331          return(-1);
332        }
333
334        printf("leyendo archivo: %s\n", arch_re);
335        printf("leyendo archivo: %s\n", arch_im);
```

```
336         for(i =0; i < nx; i++)
337         {
338               for(j =0; j < ny; j++)
339               {
340                 fscanf(pfile_re,"%lf,",&re);
341                 fscanf(pfile_im,"%lf,",&imag);
342
343                 h_mat[i*ny+j]= re+imag*I;
344               }
345
346         }
347
348
349         fclose(pfile re);
350         fclose(pfile_im);
351
352
353         return 1;
354   }
```

```
355  int saveMat_Parts(complex *h_data,int nx,int ny,char re_arch[],char im_arch[])
356  {
357
358        int i,j;
359
360        FILE*pfile;
361        pfile =fopen(re_arch,"w");
362
363
364        if(pfile ==NULL)
365        {
366          printf("error al abrir archivo para escritura %s\n", re_arch);
367          return(-1);
368        }
369
370        printf("Escribiendo archivo: %s\n", re_arch);
371
372
373        for(i =0; i < nx; i++)
374        {
375              for(j =0; j < ny; j++)
376              {
377                        if(j!=ny-1){
378
379                                 fprintf(pfile,"%lf,", creal(h_data[(i * ny)+ j]));
380
381                        }else{//el ultimo caso no termina con ","
382
383                                 fprintf(pfile,"%lf", creal(h_data[(i * ny)+ j]));
384
385                        }
386              }
387              fprintf(pfile,"\n");
388        }
389
390        fclose(pfile);
391
```

```
392         pfile =fopen(im_arch,"w");
393 
394         if(pfile ==NULL)
395         {
396           printf("error al abrir archivo para escritura %s\n", im_arch);
397           return(-1);
398         }
399 
400         printf("Escribiendo archivo: %s\n", im_arch);
401 
402 
403         for(i =0; i < nx; i++)
404         {
405               for(j =0; j < ny; j++)
406               {
407                           if(j!=ny-1){
408 
409                                   fprintf(pfile,"%lf,", cimag(h_data[(i * ny)+ j]));
410 
411                           }else{//el ultimo caso no termina con ","
412 
413                                   fprintf(pfile,"%lf", cimag(h_data[(i * ny)+ j]));
414 
415                           }
416               }
417               fprintf(pfile,"\n");
418         }
419 
420         fclose(pfile);
421 
422         return 0;
423 
424 
425 }
```

```
426  //--------------main WK------------------------
427
428  int main(int argc,char*argv[])
429  {
430
431          //leer S desde el disco
432          double complex *h_S =(double complex *)malloc(Nslow*Nfast *sizeof(double complex));
433
434
435          char re_file[]="re_S_05-Oct-2017_cut4096_pre_15.txt";
436          char im_file[]="im_S_05-Oct-2017_cut4096_pre_15.txt";
437
438
439          readMat2(re_file, im_file, h_S, Nslow, Nfast);
440
441
442          //------------------- iFFTSHIFT -------------
443
444          fftshift(h_S, Nslow, Nfast);
445
446          //------------------- FFT2 --------------------
447
448           double complex *h_out_S_fft =(double complex *)malloc(  Nslow*Nfast *sizeof(double complex));
449
450          fftw_plan plan_forward;
451
452          plan_forward =fftw_plan_dft_2d ( Nslow, Nfast, h_S, h_out_S_fft, FFTW_FORWARD, FFTW_ESTIMATE );
453
454          fftw_execute ( plan_forward );
```

```
455         //limpieza
456
457         fftw_destroy_plan(plan_forward);
458         free(h_S);
459
460
461         //FFTSHIFT
462         fftshift(h_out_S_fft, Nslow, Nfast);
463
464
465
466         double*h_ffast =(double*)malloc( Nfast *sizeof(double));
467         double*h_fslow  =(double*)malloc( Nslow *sizeof(double));
468
469         double minval =0;
470         double maxval =0;
471         int n =0;
472
473         minval =-(Ffast/2);
474         maxval =(Ffast/2)-(Ffast/(double)Nfast);
475         n =Nfast;
476
477         linspace(h_ffast,minval,maxval,n);
478
479         //vector columna
480         minval =-(Fslow/2);
481         maxval =(Fslow/2)-(Fslow/Nslow);
482         n = Nslow ;
483
484         linspace(h_fslow,minval,maxval,n);
```

```
485         //ctes para RFM
486
487         dRfmC1 =(C*C)/(4*V*V);
488         dRfmC2 = PI/Ks;
489         dRfmC3 =(4*PI*R0)/C;
490         dFc=Fc;
491
492
493         //---------llamada kernel rfm: construcción de la función de referencia -----------
494
495         double complex *h_out_rfm =(double complex *)malloc( Nslow*Nfast *sizeof(double complex));
496
497         rfm(h_out_rfm, h_fslow, Nslow, h_ffast, Nfast);
498
499
500         //------------------limpieza ----------------------------------------
501
502        free(h_ffast);
503        free(h_fslow);
504
505
506         //--------------aplico la función de referencia a los datos de S1, se obtiene S2-------------
507
508         double complex *h_out_S2 =(double complex *)malloc( Nslow*Nfast *sizeof(double complex));
509
510         //out=  h_out_S2
511         S2(h_out_S2, h_out_S_fft, h_out_rfm, Nslow, Nfast);
```

```
512         //-------------------limpeza -----------------------------
513
514         free(h_out_S_fft);
515         free(h_out_rfm);
516
517         //-------------------constantes para Delta_fast-------------
518
519         dFfast=Ffast;
520
521         //-------------------llamada al kernel de delta_fast --------
522
523
524         double*h_out_delta_fast =(double*)malloc( Nslow*Nfast *sizeof(double));
525
526         //OUT= d_out_delta_fast
527         delta_ffast(h_out_delta_fast, h_fslow, Nslow, h_ffast, Nfast);
528
529         //*************************max y min**********************
530
531         double maxim = h_out_delta_fast[0];
532         double minim = h_out_delta_fast[0];
533
534
535         getMaxMin(h_out_delta_fast, Nslow, Nfast,&maxim,&minim );
536
537         int fixminim =(int)abs(minim);
538         int fixmaxim =(int)abs(maxim);
539
540         int fixmax_mm = MAX(fixmaxim,fixminim);
541
542         dLim=fixmax_mm;
```

```
543         //***************** STOLT ******************************************
544
545         //creación de la matriz de salida de stolt
546
547         //cantidad en columnas: desde  Ni+fixmax_mm   hasta  Nfast-Ni-fixmax_mm
548
549         //num_cols = (Nfast-Ni-fixmax_mm) - (Ni+fixmax_mm) =  Nfast-2*(Ni+fixmax_mm)
550         //num_rows=: Nslow
551
552
553         int dimx = Nslow;
554         int dimy = Nfast;
555
556
557         double complex *h_out_stolt_S3;
558
559         h_out_stolt_S3=(double complex *)malloc( dimx*dimy *sizeof(double complex));
560
561         initMat(h_out_stolt_S3, dimx, dimy);
562
563         stolt(h_out_stolt_S3, h_out_delta_fast, h_out_S2, Nslow, Nfast);
564
565
566
567         //-------------limpieza --------------------
568
569        free(h_out_S2);
570        free(h_out_delta_fast);
```

```
571             //------------i-FFTSHIFT-----------------------
572
573             fftshift(h_out_stolt_S3, Nslow, Nfast);
574
575             //------------iFFT-----------------------------
576
577
578             fftw_plan plan_backward;
579             plan_backward = fftw_plan_dft_2d ( Nslow, Nfast, h_out_stolt_S3, h_out_stolt_S3, FFTW_BACKWARD, FFTW_ESTIMATE );
580             fftw_execute ( plan_backward );
581
582             normalizar_ifft(h_out_stolt_S3, Nslow, Nfast);
583
584             //----------FFTSHIFT--------------------------
585
586             fftshift(h_out_stolt_S3, Nslow, Nfast);
587
588
589             //----------se guarda en disco S4-------------
590
591             char re_S4_file_debug[]="re_S4_clang.txt";
592             char im_S4_file_debug[]="im_S4_clang.txt";
593             saveMat_Parts(h_out_stolt_S3,Nslow, Nfast, re_S4_file_debug, im_S4_file_debug );
594
595
596             //----------Limpieza -------------------------
597
598             free(h_out_stolt_S3);
599             fftw_destroy_plan ( plan_backward );
600
601             return 1;
602
603     }
```

Apéndice E ~ WK-CUDA

```
1   #include <stdio.h>
2   #include <cufft.h>
3   #include <math.h>
4   #include <thrust/reduce.h>
5   #include <thrust/copy.h>
6   #include <thrust/device_ptr.h>
7
8   #include "helper_cuda.h"
9
10  //----------------------
11
12  #define _Fc     9.4e9
13  #define _Ks     10.0e12
14  #define _Ffast  120.0e6
15  #define _Fslow  6.0e2
16
17  #define _V  250.0
18  #define _R0 25500.0
19
20  //-----------------------
21  #define _C  3.0e8
22  #define _PI 3.1416
23  //---------------------------------
24
25  #define Nslow 4096
26  #define Nfast 4096
27  //Ni = 8/2 -> se toma la mitad del kernel
```

```
28  #define Ni 4
29
30  //----------------------------------------
31
32  #define MAX(a,b)((a>b)?a:b)
33
34  //--------- kerneles -------------------------
35
36
37  __device__ cuComplex dSum_reduce_debug;
38  __device__double dFc;
39  __device__double dRfmC3;
40  __device__double dRfmC2;
41  __device__double dRfmC1;
42
43  __device__double dFfast;
44
45  __device__int dLim;
46  __device__double dPI =  3.1416;
47  __device__double d2PI = 6.2832;
48
49
50  __device__doublesinc_func(double x)
51  {
52          return (sin(dPI*x + 1e-32))/(dPI*x + 1e-32);
53  }
```

```
54  __global__ void init_kernel(cuComplex *iodata, int nx, int ny)
55  {
56
57          int i = threadIdx.y + (blockIdx.y * blockDim.y);
58          int j = threadIdx.x + (blockIdx.x * blockDim.x);
59
60          if(i < nx && j < ny)
61          {
62                  iodata[i*nx+j].x =  0.0;
63                  iodata[i*nx+j].y =  0.0;
64          }
65
66  }
```

```
67  //unicamente matrices cuadradas y pares
68  __global__void fftshift_kernel(cuComplex *d_matrix, int filas, int cols)
69  {
70          int fila = threadIdx.y + (blockIdx.y * blockDim.y);
71          int col = threadIdx.x + (blockIdx.x * blockDim.x);
72          cuComplex  aux;
73          if(fila < filas && col < cols){
74                  if(fila<filas/2)
75                  {
76                          if(col<cols/2)
77                          {
78                                  aux = d_matrix[fila*cols + col];
79                                  d_matrix[fila*cols + col] = d_matrix[(fila+(filas/2))*cols + (col+(cols/2))];
80                                  d_matrix[(fila+(filas/2))*cols + (col+(cols/2))] = aux;
81                          }
82                          else
83                          {
84                                  aux = d_matrix[fila*cols + col];
85                                  d_matrix[fila*cols + col] = d_matrix[(fila+(filas/2))*cols + (col-(cols/2))];
86                                  d_matrix[(fila+(filas/2))*cols + (col-(cols/2))] = aux ;
87                          }
88                  }
89
90          }
91
92  }
```

```
93  __global__void normalizar_ifft_kernel (cuComplex *odata, int nx, int ny)
94  {
95
96          int i = threadIdx.y + (blockIdx.y * blockDim.y);
97          int j = threadIdx.x + (blockIdx.x * blockDim.x);
98
99          if(i < nx && j < ny)
100         {
101
102                 odata[i*nx+j].x =  odata[i*nx+j].x / ( double ) ( nx * nx );
103                 odata[i*nx+j].y =  odata[i*nx+j].y / ( double ) ( nx * ny );
104         }
105
106 }
```

```
107  //creación de la función de referencia
108  __global__void rfm_kernel (cuComplex *out_rfm, double *fslow, int nx, double *ffast, int ny){
109
110          int i = threadIdx.y + (blockIdx.y * blockDim.y);
111          int j = threadIdx.x + (blockIdx.x * blockDim.x);
112
113          double res = 0.0;
114
115          if(i < nx && j < ny)
116          {
117
118
119                  res = dRfmC3 *sqrt( ( (dFc+ffast[j])*(dFc+ffast[j]) - dRfmC1 *  (fslow[i]*fslow[i]) ) )
120                                                                      +  dRfmC2 * (ffast[j]*ffast[j]);
121                  res = fmod(res, d2PI);
122
123                  out_rfm[i*ny+j].x= cos(res);
124                  out_rfm[i*ny+j].y= sin(res);
125
126
127          }
128  }
```

```
129  __global__void S2_kernel (cuComplex *S1, cuComplex *rfm, int nx, int ny){
130
131          int i = threadIdx.y + (blockIdx.y * blockDim.y);
132          int j = threadIdx.x + (blockIdx.x * blockDim.x);
133
134          double s1x=0.0;
135          double s1y=0.0;
136
137          if(i < nx && j < ny)
138          {
139
140                  s1x = S1[i*ny+j].x;
141                  s1y = S1[i*ny+j].y;
142
143                  //multiplicacion de 2 nros complejos
144                  S1[i*ny+j].x = (s1x * rfm[i*ny+j].x) - (s1y * rfm[i*ny+j].y);
145                  S1[i*ny+j].y = (s1x * rfm[i*ny+j].y) + (s1y * rfm[i*ny+j].x);
146
147          }
148  }
149
```

```
150  //creación de la nueva variable deltaFfast que usará Stolt para inteporlar con S2
151  __global__void delta_ffast_kernel (double *out_delta_ffast, double *fslow, int nx, double *ffast, int ny){
152
153          int i = threadIdx.y + (blockIdx.y * blockDim.y);
154          int j = threadIdx.x + (blockIdx.x * blockDim.x);
155
156          double res = 0.0;
157          double root = 0.0;
158
159
160          if(i < nx && j < ny)
161          {
162                  root = (double) ( (dFc+ffast[j])*(dFc+ffast[j]) - dRfmC1 * (fslow[i]*fslow[i])  );
163                  res = sqrt( root ) - (dFc+ffast[j]);
164
165                  //normaliza deltafasta a valores de índice mutiplicando por Nfast/Ffast
166                  //Nfast es ny
167                  res = res*(ny/dFfast);
168
169                  out_delta_ffast[i*ny+j]=res;
170          }
171  }
```

```
172 __global__ void stolt_kernel (cuComplex *out_stolt_S3, double *deltafast, cuComplex *S2, int filas, int cols){
173
174         int ix = threadIdx.x + (blockIdx.x * blockDim.x);//cols index
175         int iy = threadIdx.y + (blockIdx.y * blockDim.y);//rows index
176
177         //declaración e inicialización de variables para el calculo
178
179         double data_delta_ffast = 0.0f;
180
181         cuComplex valor_ant_3   = make_cuComplex(0.0f, 0.0f);
182         cuComplex valor_ant_2   = make_cuComplex(0.0f, 0.0f);
183         cuComplex valor_ant_1   = make_cuComplex(0.0f, 0.0f);
184          cuComplex valor_central = make_cuComplex(0.0f, 0.0f);
185          cuComplex valor_pos_1   = make_cuComplex(0.0f, 0.0f);
186          cuComplex valor_pos_2   = make_cuComplex(0.0f, 0.0f);
187          cuComplex valor_pos_3   = make_cuComplex(0.0f, 0.0f);
188          cuComplex valor_pos_4   = make_cuComplex(0.0f, 0.0f);
189
190         int ent=0;
191         double dec= 0.0f;
192
193         double sinc_ant_3 = 0.0f;
194         double sinc_ant_2 = 0.0f;
195         double sinc_ant_1 = 0.0f;
196         double sinc_central= 0.0f;
197         double sinc_pos_1 = 0.0f;
198         double sinc_pos_2 = 0.0f;
199         double sinc_pos_3 = 0.0f;
200         double sinc_pos_4 = 0.0f;
```

```
201         int lim1 = Ni+dLim;
202         int lim2 = cols -lim1;
203
204         if ( (iy < filas)  && ( lim1-1<= ix) && (ix <= lim2-1) )
205         {
206
207                 data_delta_ffast = deltafast[iy * cols + ix];
208                 //toma parte entera
209                 ent= (int) data_delta_ffast;
210
211                 //valores cuComplex, la asignación es directa real-imag en un solo paso, vale Z1 = Z2
212                 valor_ant_3   = S2[iy*cols + (ix+ent-3)];
213                 valor_ant_2   = S2[iy*cols + (ix+ent-2)];
214                 valor_ant_1   = S2[iy*cols + (ix+ent-1)];
215                 valor_central = S2[iy * cols+(ix+ent)];
216                 valor_pos_1   = S2[iy*cols + (ix+ent+1)];
217                 valor_pos_2   = S2[iy*cols + (ix+ent+2)];
218                 valor_pos_3   = S2[iy*cols + (ix+ent+3)];
219                 valor_pos_4   = S2[iy*cols + (ix+ent+4)];
220
221                 dec= data_delta_ffast - ent;
222
223                 sinc_ant_3   = sinc_func(dec-3.0);
224                 sinc_ant_2   = sinc_func(dec-2.0);
225                 sinc_ant_1   = sinc_func(dec-1.0);
226                 sinc_central = sinc_func(dec);
227                 sinc_pos_1   = sinc_func(dec+1.0);
228                 sinc_pos_2   = sinc_func(dec+2.0);
229                 sinc_pos_3   = sinc_func(dec+3.0);
230                 sinc_pos_4   = sinc_func(dec+4.0);
231
```

```
232                 // La multiplicación por constantes, del tipo: Z1*cte no es valido en cuComplex
233                 // Vale RE{z}= RE{z}*cte y Imag{z}= Imag{z}*cte
234                 //Debido a esto se separa el cálculo en dos líneas, real e imaginario
235
236
237                 out_stolt_S3[iy * cols + ix].x = (valor_ant_3.x   *  sinc_ant_3)
238                                                + (valor_ant_2.x   *  sinc_ant_2)
239                                                + (valor_ant_1.x   *  sinc_ant_1)
240                                                + (valor_central.x *  sinc_central)
241                                                + (valor_pos_1.x   *  sinc_pos_1)
242                                                + (valor_pos_2.x   *  sinc_pos_2)
243                                                + (valor_pos_3.x   *  sinc_pos_3)
244                                                + (valor_pos_4.x   *  sinc_pos_4);
245
246
247                 out_stolt_S3[iy * cols + ix].y = (valor_ant_3.y   *  sinc_ant_3)
248                                                + (valor_ant_2.y   *  sinc_ant_2)
249                                                + (valor_ant_1.y   *  sinc_ant_1)
250                                                + (valor_central.y *  sinc_central)
251                                                + (valor_pos_1.y   *  sinc_pos_1)
252                                                + (valor_pos_2.y   *  sinc_pos_2)
253                                                + (valor_pos_3.y   *  sinc_pos_3)
254                                                + (valor_pos_4.y   *  sinc_pos_4);
255
256
257             }//del if
258
259
260     }
```

```
261 //---------------------------------------wrappers ----------------------------------------
262
263 __host__int fftshift(cuComplex *d_matrix, int filas, int cols)
264 {
265         //dim3 Nthreads(2,2,1);
266         dim3 Nthreads(32,32,1);
267
268         dim3 Nblocks((cols/Nthreads.x) + (cols % Nthreads.x?1:0), (filas/Nthreads.y) + (filas % Nthreads.y?1:0));
269
270
271         fftshift_kernel<<<Nblocks, Nthreads>>>(d_matrix, filas, cols);
272         getLastCudaError("fftshift_kernel() kernel failed");
273
274         checkCudaErrors( cudaDeviceSynchronize() );
275
276         return 1;
277
278 }
279
280 __host__int normalizarIfft(cuComplex *d_matrix_out, int nx, int ny)
281 {
282       //dim3 Nthreads(2,2,1);
283       dim3 Nthreads(32,32,1);
284
285       dim3 Nblocks((nx/Nthreads.x) + (nx % Nthreads.x?1:0), (ny/Nthreads.y) + (ny % Nthreads.y?1:0));
286
287       normalizar_ifft_kernel<<<Nblocks, Nthreads>>>(d_matrix_out, ny, ny);
288       getLastCudaError("normalizar_ifft_kernel() kernel failed");
289
290       checkCudaErrors( cudaDeviceSynchronize() );
291
292       return 1;
293
294 }
```

```
295  __host__int rfm(cuComplex *d_out_rfm, double *d_in_fslow, int nx, double *d_in_ffast, int ny){
296
297        dim3 Nthreads(32,32,1);
298        dim3 Nblocks((ny/Nthreads.y) + (ny % Nthreads.y?1:0), (nx/Nthreads.x) + (nx % Nthreads.x?1:0));
299
300        rfm_kernel<<<Nblocks, Nthreads>>>(d_out_rfm, d_in_fslow, nx, d_in_ffast, ny);
301        getLastCudaError("rfm_kernel() kernel failed");
302
303        checkCudaErrors( cudaDeviceSynchronize() );
304
305        return 1;
306  }
307
308  __host__int S2(cuComplex *d_in_S1, cuComplex *d_in_rfm, int nx, int ny){
309
310     dim3 Nthreads(32,32,1);
311     dim3 Nblocks((ny/Nthreads.y) + (ny % Nthreads.y?1:0), (nx/Nthreads.x) + (nx % Nthreads.x?1:0));
312
313     S2_kernel<<<Nblocks, Nthreads>>>(d_in_S1, d_in_rfm, nx, ny);
314     getLastCudaError("S2_kernel() kernel failed");
315
316     checkCudaErrors( cudaDeviceSynchronize() );
317
318     return 1;
319  }
```

```
__host__int delta_ffast(double *d_out_delta_ffast, double *d_in_fslow, int nx, double *d_in_ffast, int ny){

    dim3 Nthreads(32,32,1);
    dim3 Nblocks((nx/Nthreads.x) + (nx % Nthreads.x?1:0), (ny/Nthreads.y) + (ny % Nthreads.y?1:0));


    delta_ffast_kernel<<<Nblocks, Nthreads>>>(d_out_delta_ffast, d_in_fslow, nx, d_in_ffast, ny);
    getLastCudaError("delta_ffast_kernel() kernel failed");

    checkCudaErrors( cudaDeviceSynchronize() );

    return1;
}


__host__int stolt(cuComplex *d_out_stolt_S3, double *d_in_deltafast, cuComplex *d_in_S2, int nx, int ny){


      dim3 Nthreads(16,16,1);

      dim3 Nblocks((nx/Nthreads.x) + (nx % Nthreads.x?1:0), (ny/Nthreads.y) + (ny % Nthreads.y?1:0));

      stolt_kernel<<<Nblocks, Nthreads>>>(d_out_stolt_S3, d_in_deltafast, d_in_S2, nx, ny);
      getLastCudaError("stolt_kernel() kernel failed");

      checkCudaErrors( cudaDeviceSynchronize() );

      return 1;
}
```

```
349  //------------------ Utils -------------------------
350
351  __host__int init(cuComplex *d_matrix_out, int nx, int ny)
352  {
353
354        dim3 Nthreads(32,32,1);
355
356        dim3 Nblocks((nx/Nthreads.x) + (nx % Nthreads.x?1:0), (ny/Nthreads.y) + (ny % Nthreads.y?1:0));
357
358        init_kernel<<<Nblocks, Nthreads>>>(d_matrix_out, ny, ny);
359        getLastCudaError("init_kernel() kernel failed");
360
361        checkCudaErrors( cudaDeviceSynchronize() );
362
363        return 1;
364
365  }
366
367  __host__void init_mat_host(cuComplex *h_out, int nx, int ny)
368  {
369          int i,j;
370
371          for (i=0; i<nx; i++)
372          {
373                  for(j=0; j<ny; j++)
374                  {
375                          h_out[i*ny+j].x=0.0;
376                          h_out[i*ny+j].y=0.0;
377                  }
378          }
379          return;
380  }
381
```

```
382  //se dá un valor mínimo y máximo del vector y se pide n divisiones
383  __host__int linspace(double* vector, double minval, double maxval, int n)
384  {
385
386          if(n <2){
387                  return0;
388          }
389
390          int i = 0;
391          double step = (maxval-minval)/(floor((double)n) - 1.0) ;
392
393          for (i = 0; i < n-1; i++)
394          {
395              vector[i]=minval + i*step;
396
397          }
398          vector[n-1]=maxval;
399
400
401          return 1;
402  }
```

```
403  __host__int readMat(cuComplex* h_mat, int nx, int ny)
404  {
405        char arch[] = "Sraw_2.txt";
406
407        int i,j;
408        double re;
409        double imag;
410
411        FILE *pfile;
412        pfile = fopen(arch, "r");
413
414        if (pfile == NULL)
415        {
416              printf("error al abrir archivo para lectura %s \n", arch);
417              return (-1);
418        }
419
420        printf("leyendo archivo: %s \n", arch);
421
422
423        for (i = 0; i < nx; i++)
424        {
425          for(j = 0; j < ny; j++)
426          {
427                   fscanf(pfile,"%lf%lfi,",&re, &imag);
428                    h_mat[i*ny+j]= make_cuComplex(re, imag);
429          }
430        }
431        fclose(pfile);
432        return 1;
433  }
```

```
434 __host__int readMat2(char arch_re[], char arch_im[], cuComplex* h_mat, int nx, int ny)
435 {
436
437       int i,j;
438       double re;
439       double imag;
440
441
442       FILE *pfile_re, *pfile_im;
443           pfile_re = fopen(arch_re, "r");
444           pfile_im = fopen(arch_im, "r");
445
446       if (pfile_re == NULL)
447       {
448              printf("error al abrir archivo para lectura %s \n", arch_re);
449              return (-1);
450       }
451       if (pfile_im == NULL)
452       {
453              printf("error al abrir archivo para lectura %s \n", arch_im);
454              return (-1);
455       }
456
457       printf("leyendo archivo: %s \n", arch_re);
458       printf("leyendo archivo: %s \n", arch_im);
```

```
459         for (i = 0; i < nx; i++)
460         {
461               for(j = 0; j < ny; j++)
462               {
463                           fscanf(pfile_re, "%lf,",&re);
464                           fscanf(pfile_im,"%lf,",&imag);
465                           h_mat[i*ny+j]= make_cuComplex(re, imag);
466                }
467
468         }
469
470         fclose(pfile_re);
471         fclose(pfile_im);
472
473
474         return 1;
475   }
```

```
476  __host__int saveMat(cuComplex *h_data, int nx, int ny, char arch[])
477  {
478
479        int i,j;
480
481        FILE *pfile;
482        pfile = fopen(arch, "w");
483
484
485        if (pfile == NULL)
486        {
487          printf("error al abrir archivo para escritura %s \n", arch);
488          return (-1);
489        }
490
491        printf("Escribiendo archivo: %s \n", arch);
```

```
492         for (i = 0; i < nx; i++)
493         {
494           for(j = 0; j < ny; j++)
495                 {
496                  if(j!=ny-1){
497
498                         //terminan en ","
499                         if( h_data[(i * ny) + j].y >0){
500                                 fprintf(pfile,"%.4lf+%.4lfi,", h_data[(i * ny) + j].x, h_data[(i * ny) + j].y);
501                          }
502                         else{
503                                 fprintf(pfile,"%.4lf%.4lfi,", h_data[(i * ny) + j].x, h_data[(i * ny) + j].y);
504                         }
505
506                  }else{
507                         //el ultimo caso no termina con ","
508                         if( h_data[(i * ny) + j].y >0){
509                                 fprintf(pfile,"%.4lf+%.4lfi", h_data[(i * ny) + j].x, h_data[(i * ny) + j].y);
510                         }
511                         else{
512                                 fprintf(pfile,"%.4lf%.4lfi", h_data[(i * ny) + j].x, h_data[(i * ny) + j].y);
513                         }
514
515                  }
516           }
517           printf(pfile,"\n");
518          }
519
520         fclose(pfile);
521
522         return 0;
523
524 }
```

```
525  __host__int saveMat_Parts(cuComplex *h_data, int nx, int ny, char re_arch[], char im_arch[])
526  {
527
528        int i,j;
529
530        FILE *pfile;
531        pfile = fopen(re_arch, "w");
532
533
534        if (pfile == NULL)
535        {
536               printf("error al abrir archivo para escritura %s \n", re_arch);
537               return (-1);
538        }
539
540        printf("Escribiendo archivo: %s \n", re_arch);
541
542
543        for (i = 0; i < nx; i++)
544        {
545               for(j = 0; j < ny; j++)
546               {
547                 if(j!=ny-1){
548
549                            fprintf(pfile,"%lf,", h_data[(i * ny) + j].x);
550
551                 }else{       //el ultimo caso no termina con ","
552
553                         fprintf(pfile,"%lf", h_data[(i * ny) + j].x);
554
555                 }
556               }
557               fprintf(pfile,"\n");
558        }
```

```
559             fclose(pfile);
560             pfile = fopen(im_arch, "w");
561 
562             if (pfile == NULL)
563             {
564                printf("error al abrir archivo para escritura %s \n", im_arch);
565                return (-1);
566             }
567 
568             printf("Escribiendo archivo: %s \n", im_arch);
569 
570             for (i = 0; i < nx; i++)
571             {
572                for(j = 0; j < ny; j++)
573                {
574                  if(j!=ny-1){
575 
576                              fprintf(pfile,"%lf,", h_data[(i * ny) + j].y);
577 
578                  }else{      //el ultimo caso no termina con ","
579 
580                             fprintf(pfile,"%lf", h_data[(i * ny) + j].y);
581 
582                  }
583                }
584                fprintf(pfile,"\n");
585             }
586 
587             fclose(pfile);
588 
589         return 0;
590 
591 }
```

```
592  //------------------------------main WK-------------------------------------------------------------
593
594  int main(int argc, char *argv[])
595  {
596
597
598          double Fc     = (double) _Fc;
599          double Ks     = (double) _Ks;
600          double Ffast  = (double) _Ffast;
601          double Fslow  = (double) _Fslow;
602
603
604          double V  = (double) _V;
605          double R0 = (double) _R0;
606          double C  = (double) _C;
607          double PI = (double) _PI;
608
609           cuComplex *h_S =  (cuComplex *) malloc(  Nslow*Nfast * sizeof(cuComplex));
610
611          //Lectura de datos crudos
612          char re_file[] = "re_S.txt";
613          char im_file[] = "im_S.txt";
614
615          readMat2(re_file, im_file, h_S, Nslow, Nfast);
616
617
618          cuComplex *d_S;
619
620          checkCudaErrors( cudaMalloc((void**)&d_S, sizeof(cuComplex) * Nslow*Nfast) );
621
622          checkCudaErrors( cudaMemcpy(d_S, h_S, Nslow*Nfast * sizeof(cuComplex), cudaMemcpyHostToDevice) );
623
```

```
624         //------------------- iFFTSHIFT -------------solo matrices cuadradas-----------------
625         free(h_S);
626
627         //OUT inplace = d_S
628         fftshift(d_S, Nslow, Nfast);
629
630         //------------------- FFT2 ---------inplace-----------------------------------------------------
631
632         cufftHandle plan;
633         cufftPlan2d(&plan, Nslow, Nfast, CUFFT_C2C);
634
635         //out : d_S inplace
636         cufftExecC2C(plan, d_S, d_S, CUFFT_FORWARD);
637
638         cufftDestroy(plan);
639
640         //------------------- FFTSHIFT ------------solo matrices cuadradas-----------
641
642         //in-place d_S
643         fftshift(d_S, Nslow, Nfast);
```

```
644         //----------------- vectores ffast y fslow generadores de RFM ---------------------------------
645
646         double *h_ffast  =  (double *) malloc( Nfast * sizeof(double));
647         double *h_fslow  =  (double *) malloc( Nslow * sizeof(double));
648
649         //vector fila
650         double minval =0.0;
651         double maxval =0.0;
652         int n =0;
653
654         minval = -(Ffast/2);
655         maxval =  (Ffast/2) - (Ffast/Nfast);
656         n =Nfast;
657
658         linspace(h_ffast,minval,maxval,n);
659
660         //vector columna
661         minval =-(Fslow/2);
662         maxval = (Fslow/2) - (Fslow/Nslow);
663         n =Nslow;
664
665         linspace(h_fslow,minval,maxval,n);
666
667         double *d_ffast;
668
669         checkCudaErrors( cudaMalloc((void**)&d_ffast, sizeof(double) * Nfast) );
670         checkCudaErrors( cudaMemcpy(d_ffast, h_ffast, Nfast * sizeof(double), cudaMemcpyHostToDevice) );
671
672
673         double *d_fslow;
674         checkCudaErrors( cudaMalloc((void**)&d_fslow, sizeof(double) * Nslow) );
675         checkCudaErrors( cudaMemcpy(d_fslow, h_fslow, Nslow * sizeof(double), cudaMemcpyHostToDevice) );
```

```
676         //-------------------limpieza -------------------------------
677
678         free(h_ffast);
679         free(h_fslow);
680
681         //------------------- ctes para RFM-----------------------
682
683         double h_RfmC1 = (C*C)/(4*V*V);
684         double h_RfmC2 = PI/Ks;
685         double h_RfmC3 = (4*PI*R0)/C;
686
687
688         checkCudaErrors( cudaMemcpyToSymbol(dRfmC1,&h_RfmC1,sizeof(double)) );
689         checkCudaErrors( cudaMemcpyToSymbol(dRfmC2,&h_RfmC2,sizeof(double)) );
690         checkCudaErrors( cudaMemcpyToSymbol(dRfmC3,&h_RfmC3,sizeof(double)) );
691
692
693         double h_Fc = Fc;
694         checkCudaErrors( cudaMemcpyToSymbol(dFc,&h_Fc,sizeof(double)) );
```

```
695         //---------llamada kernel rfm: construccion de la funcion de referencia --------
696
697         cuComplex *d_out_rfm;
698         checkCudaErrors( cudaMalloc((cuComplex**)&d_out_rfm, sizeof(cuComplex) * Nslow*Nfast) );
699
700         //OUT inplace= d_out_rfm
701         rfm(d_out_rfm, d_fslow, Nslow, d_ffast, Nfast);
702
703         //--------------aplico la función de referencia a los datos de S1, se obtiene S2------
704         //---------------------- hasta aquí d_S es el teórico S1-------------------------------
705
706         //out= d_S inplace
707         S2(d_S, d_out_rfm, Nslow, Nfast);
708
709         checkCudaErrors( cudaFree(d_out_rfm) );
710
711         //-------------------------a partir de aquí d_S es el teórico S2 -------------------
712         //-------------------------constantes para Delta_fast-----------------------------
713
714         double h_Ffast = Ffast; //esta ok
715         checkCudaErrors( cudaMemcpyToSymbol(dFfast,&h_Ffast,sizeof(double)) );
716
717         //-------------------------llamada al kernel de delta_fast mapeo de stolt ----------
718
719         double *h_out_delta_fast =  (double *) malloc( Nslow*Nfast * sizeof(double));
720
721         double *d_out_delta_fast;
722          checkCudaErrors( cudaMalloc((void**)&d_out_delta_fast, sizeof(double) * Nslow*Nfast) );
723
724         //OUT= d_out_delta_fast
725         delta_ffast(d_out_delta_fast, d_fslow, Nslow, d_ffast, Nfast);
726
727
728
```

```
729         //---------------------------Limpieza -----------------------------------------------------
730
731         checkCudaErrors( cudaFree(d_ffast) );
732         checkCudaErrors( cudaFree(d_fslow) );
733
734         //************************THRUST---reduce max y min****memcpytosymbol**********************
735
736         double cteThrust = h_out_delta_fast[0];
737
738         //---------------------------------limpieza -----------------------------------------------
739
740         free(h_out_delta_fast);
741
742         //-------------------------------------------------------------------------------------------
743
744         thrust::device_ptr<double>d_t = thrust::device_pointer_cast(d_out_delta_fast);
745
746
747         double minim = thrust::reduce(d_t, d_t+(Nslow*Nfast), cteThrust, thrust::minimum<double>());
748         double maxim = thrust::reduce(d_t, d_t+(Nslow*Nfast), cteThrust, thrust::maximum<double>());
749
750
751         int fixminim = (int) abs(minim);
752         int fixmaxim = (int) abs(maxim);
753
754         int fixmax_mm = MAX(fixmaxim,fixminim);
755
756         checkCudaErrors( cudaMemcpyToSymbol(dLim,&fixmax_mm,sizeof(int)) );
757
```

```
758         //********************************** STOLT ***** out-place ******************************
759         //----------------------creacion de la matriz de salida de stolt-----------------------------
760
761         //cantidad en columnas: desde  Ni+fixmax_mm   hasta  Nfast-Ni-fixmax_mm
762         //num_cols = (Nfast-Ni-fixmax_mm) - (Ni+fixmax_mm) =  Nfast-2*(Ni+fixmax_mm)
763         //num_rows=: Nslow
764
765         int dimx = Nslow;
766         int dimy = Nfast;
767
768         cuComplex *d_out_stolt_S3;
769         cudaMalloc((void**)&d_out_stolt_S3, sizeof(cuComplex) * dimx*dimy );
770
771         init(d_out_stolt_S3, dimx, dimy);
772
773
774         //observ: aquí se debe considerar a d_S = S2 de Matlab
775         //IN: double: d_out_delta_fast, cuComlpex: d_S, int: Nslow, Nfast
776         //OUT: cuComplex: d_out_stolt_S3
777         //nx, ny = cols, filas
778         stolt(d_out_stolt_S3, d_out_delta_fast, d_S, Nslow, Nfast);
```

```
779         //-------------------------------limpieza ----------------------
780
781         checkCudaErrors( cudaFree(d_out_delta_fast) );
782         checkCudaErrors( cudaFree(d_S) );//S2
783
784
785         //----------------------------i-FFTSHIFT ----------------------
786
787         //OUT inplace = d_out_stolt_S3
788         fftshift(d_out_stolt_S3, Nslow, Nfast);
789
790         //----------------------------iFFT-------------------------------
791
792         cufftHandle plan2;
793         cufftPlan2d(&plan2, Nslow, Nfast, CUFFT_C2C);
794
795         //out in-place: d_out_stolt_S3
796         cufftExecC2C(plan2, d_out_stolt_S3, d_out_stolt_S3, CUFFT_INVERSE);
797
798         //out in-place: d_out_stolt_S3
799         normalizarIfft(d_out_stolt_S3, Nslow, Nfast);
```

```
800         //--------------------------------limpieza ----------------------------------------
801         cufftDestroy(plan2);
802         //--------------------------------FFTSHIFT-----------------------------------------
803
804         //obtener resultado final S4
805         //OUT inplace = d_out_stolt_S3
806         fftshift(d_out_stolt_S3, Nslow, Nfast);
807
808         //---------------------------------se guarda en disco ------------------------------
809
810         cuComplex *h_out_stolt_S3 =  (cuComplex *) malloc( dimx*dimy * sizeof(cuComplex));
811         init_mat_host(h_out_stolt_S3, Nslow, Nfast);
812
813         checkCudaErrors( cudaMemcpy(h_out_stolt_S3, d_out_stolt_S3, Nslow*Nfast * sizeof(cuComplex),
814                       cudaMemcpyDeviceToHost) );
815
816         char re_S4_file_debug[] = "re_S4_cuda.txt";
817         char im_S4_file_debug[] = "im_S4_cuda.txt";
818
819         saveMat_Parts(h_out_stolt_S3,Nslow, Nfast, re_S4_file_debug, im_S4_file_debug );
820
821         //-----------------------------------Limpieza -----------------------------------------
822
823         free(h_out_stolt_S3);
824         checkCudaErrors( cudaFree(d_out_stolt_S3) );
825
826         return 1;
```

Apéndice F ~ Graficador en MATLAB

```
function n = graficador(S, S1, S2, S3, S4);


%-------------------------------------------------------------------------
    figure
    subplot(2,2,1)
    imagesc (abs (S)); title('1.Datos crudos (modulo)')
    subplot(2,2,2)
    imagesc (abs (S1)); title('2.Espectro 2D de los Datos crudos')
    subplot(2,2,3)
    imagesc (abs (S2));title('3.Luego de la funcion de referencia')
    subplot(2,2,4)
    imagesc (abs (S3)); title('4.Luego de Stolt')

    figure
    subplot(2,1,1)
    imagesc (abs (S2-S3)); title('5.Diferencia interpolada vs no interpolada (modulo)')
    subplot(2,1,2)
    imagesc (real (S2-S3)); title('6.Diferencia interpolada vs no interpolada (parte real)')

    figure
    subplot(2,2,1)
    imagesc (real (S)); title('7.Datos crudos (parte real)')
    subplot(2,2,2)
    imagesc (real ((S1))); title('8.Espectro 2D de los Datos crudos')
    subplot(2,2,3)
    imagesc (real ((S2)));title('9.Luego de la funcion de referencia')
    subplot(2,2,4)
```

```
29
30      imagesc (real ((S3))); title('10.Luego de Stolt')
31
32      %--------------Principal----------------------------------------
33      figure
34      imagesc(abs(S4));
35      title('11.Image after 2-d ifft');
36      xlabel('range');ylabel('azimuth');
37
38      % Hallamos el máximo y graficamos 3D en el entorno
39      [xMax,yMax]=find(abs(S4)==max(max(abs(S4))));
40
41      Delta = -100:100;
42      figure; surf(abs(S4(xMax(1)+Delta,yMax(1)+Delta)))
43      shading interp
44      colorbar
45      title('11.Hallamos el máximo y gradicamos 3D en el entorno');
46      xlabel('range');ylabel('azimuth');
47
48     %-----------------------------------------------------------------
49
50
51      figure
52      subplot(2,2,1); plot(abs(S4(:,yMax(1))));legend('12.Corte de la respuesta final en rango [abs]')
53      subplot(2,2,2); plot(abs(S4(xMax(1),:)));legend('13.Corte de la respuesta final en acimut [abs]')
54      subplot(2,2,3); plot(real(S4(:,yMax(1))));legend('14.Corte de la respuesta final en rango [real]')
55      subplot(2,2,4); plot(real(S4(xMax(1),:)));legend('15.Corte de la respuesta final en acimut [real]')
56      figure
57      plot(real(fftshift(fft(ifftshift(S4(:,yMax))))) ;
58      legend('16.Corte de la respuesta final en rango [abs] - dominio de frec')
59
```

```
%-----------------------------------------------------------------------

[xMax,yMax] = find(abs(S4) == max(max(abs(S4))));

xMax = xMax(1);
yMax = yMax(1);
Delta = -100:100;

figure

imagesc(abs(S4));
title(['Stolt focused image - Point target in (' num2str(xMax) ',' num2str(yMax) ')']);
xlabel('range');
ylabel('azimuth');

figure;

subplot(1,2,1), surf(abs(S4(xMax(1)+Delta,yMax(1)+Delta))), shading
interp,colorbar,xlabel('range');
ylabel('azimuth');
title('Point target zoom')

S2R= fftshift(ifft2(ifftshift(S2)));

subplot(1,2,2), surf(abs(S2R(xMax(1)+Delta,yMax(1)+Delta))), shading interp,colorbar,
title('Same region before Stolt interpolation (only bulk RFM)')

figure
plot(20*log10(abs(S4(xMax,yMax(1)+Delta))));
figure
plot(20*log10(abs(S4(xMax(1)+Delta,yMax(1)))));

```

Apéndice G ~ Validación Resolución Espacial en MATLAB

```
load ('S4.mat');


Ks = 10e12;              % Chirp Frequency 3MHz / us
Ts = 10e-6;              % Chirp pulse width 10us
Bs = Ks*Ts;              % Chirp signal bandwidth (Hz)
C = 3e8;                 % Speed of light m/s

D = 1;                   % Set the actual aperture of the antenna (m)

%-------------------------- de parametros del parametrosMKS -----------
H    = 10000; % Altitude                                -- [m]
Sw   = 10000; % Slant range swath width                 -- [m]
Rnc  = 25000; % Slant range of scene center R(nc)       -- [m]
GRnc = sqrt(H^2+Rnc^2); % Ground range of scene center -- [m]
Rca =  sqrt((GRnc-Sw/2)^2+H^2); % Slant range of closest approach [m]
Rfa = sqrt((GRnc+Sw/2)^2+H^2); % Slant range of farthest approach [m]
```

```
18
19  %------------ fórmulas de cumming en w_ch: resolución espacial teórica-----
20  %--------------Estos valores teoricos son el objetivo a validar -----------
21
22  R_resol_teorica = C/2/Bs;    % Calculate the distance to imaging accuracy
23  A_resol_teorica = D/2;       % Calculate azimuthal imaging accuracy
24
25
26  %Obtengo la cantidad de muestras en rango(Irwx) y acimut(Irwy) a -3dB
27  [Irwx,Irwy] = IRW_fn(S4);
28
29  %------------calculo los pixelSpacing: Cumming -----------
30
31  Fr      =   120e6; % Range sampling rate [Hz]
32  Vr      =   250;     % Effective radar velocity [m/s]
33  PRF     =   600;     % Azimuth sampling rate [Hz]
34
35  Psr = C/(2*Fr); %en metros
36  Psa = Vr/PRF;   %en metros
```

```
% -------------criterio del orden de magintud-----------

resol_ra_med=Irwx*psr; %en metros, resolución espacial en Rango, medido

resol_ac_med=Irwy*psa; %en metros, resolución espacial en Acimut, medido

val_ac= order(A_resol_teorica) - order(resol_ac_med);
val_rg= order(R_resol_teorica) - order(resol_ra_med);

if val_ac==0
    disp 'validacion ok en Acimut igual orden'
else
    disp 'no pasa la validación en Acimut'
end

if val_rg==0
    disp 'validacion ok en rango igual orden'
else
    disp 'no pasa la validación en rango'
end
```

```
function [Irwx,Irwy] = IRW_fn(S4);
% Obtain the 3 dB width in range and azimuth of a focused point target.
% Usage: [Irwx,Irwy]=IRW(Data);
% Input: 2D azimuth-range matrix
% Output: Irwx is the number of samples of the 3dB peak in range dimension
%         Irwy is the number of samples of the 3dB peak in azimuth dimension

% Get the max
[xMax,yMax]=find(abs(S4)==max(max(abs(S4))));

% Define relevant interval
Delta = -100:100;

% Get data around max
rangeData   = S4(xMax(1), yMax(1)+Delta);
azimuthData = S4(xMax(1)+Delta , yMax(1));

%
rangeDatadB   = 20*log10(abs(rangeData));
azimuthDatadB = 20*log10(abs(azimuthData));
range3dBIx    = find((rangeDatadB  -max(rangeDatadB  ))>(-3));
azimuth3dBIx = find((azimuthDatadB-max(azimuthDatadB))>(-3));

Irw_x = max(range3dBIx)  -min(range3dBIx  ); % Equivalent number of samples of 3dB peak
Irw_y = max(azimuth3dBIx)-min(azimuth3dBIx); % Equivalent number of samples of 3dB peak

figure
subplot(2,2,1); plot(rangeDatadB);   hold on ; plot(range3dBIx,rangeDatadB(range3dBIx),'r');
plot(ones(1,length(Delta))*(max(rangeDatadB)-3),'r-.');
legend('Lobulo en rango [dB] - sin interpolar');
text(length(rangeDatadB)*2/3,rangeDatadB(max(range3dBIx))/2,['# Samples in peak = ' num2str(Irw_x)]) subplot(2,2,2);
plot(azimuthDatadB);
hold on ;
plot(azimuth3dBIx,azimuthDatadB(azimuth3dBIx),'g');
```

```
plot(ones(1,length(Delta))*(max(azimuthDatadB)-3),'g-.');
legend('Lobulo en azimuth [dB] - sin interpolar');
text(length(rangeDatadB)*2/3,azimuthDatadB(max(azimuth3dBIx))/2,['# Samples in peak = ' num2str(Irw_y)])

Irwx = ceil(Irw_x+eps);
Irwy = ceil(Irw_y+eps);




function n = order( val, base )

%Order of magnitude of number for specified base. Default base is 10.
%order(0.002) will return -3., order(1.3e6) will return 6.

if nargin < 2
    base = 10;
end
n = floor(log(abs(val))./log(base));
```

Apéndice H ~ Validación Igualdad de Versiones en MATLAB

```
%importar desde Matlab

fname_re_matlab = 're_S4_.txt';
fname_im_matlab = 'im_S4_.txt';

S4_re_matlab=dlmread(fname_re_matlab);
S4_im_matlab=dlmread(fname_im_matlab);

S4_matlab = S4_re_matlab + S4_im_matlab*i;

%importar desde C/CUDA

fname_re_S4_c_cuda = 're_S4_clang.txt';
fname_im_S4_c_cuda = 'im_S4_clang.txt';

S4_re_c_cuda=dlmread(fname_re_S4_c_cuda);
S4_im_c_cuda=dlmread(fname_im_S4_c_cuda);

S4_c_cuda = S4_re_c_cuda + S4_im_c_cuda*i;

%filtra, para valores altos, y toma índices

ixm=abs(S4_matlab) > 0.3;

%Utilizamos la matriz de índices para calcular el error relativo de
%S4_cuda respecto a S4_matlab, con la siguiente expresión:
max(max(abs(S4_matlab(ixm)-S4_c_cuda(ixm))./abs(S4_matlab(ixm))))
```

Apéndice I ~ Conversión *.MAT-*.CSV en MATLAB

```
param_num_pre ='15';
pres = strcat('%.' ,param_num_pre,'f');

fecha=strcat(date ,'_cut4096_', 'pre_', param_num_pre );

fname_re_raw =strcat('re_S' ,'_',fecha, '.txt');
fname_im_raw =strcat('im_S' ,'_',fecha, '.txt');

fname_re_s4 =strcat('re_S4' ,'_',fecha, '.txt');
fname_im_s4 =strcat('im_S4' ,'_',fecha, '.txt');

S=load('datosCrudos.mat')

re_S=real(S);
im_S=imag(S);

%export RAW data como texto
dlmwrite(fname_re_raw,re_S, 'precision', pres);
dlmwrite(fname_im_raw,im_S, 'precision', pres);

S4=load('S4_focalizado.mat')
re_S4=real(S4);
im_S4=imag(S4);

% export S4 focalizado como texto
dlmwrite(fname_re_s4,re_S4, 'precision', pres);
dlmwrite(fname_im_s4,im_S4, 'precision', pres);
```

Apéndice J ~ Desarrollo en Series de Taylor, para la interpretación de la Interpolación de Stolt mediante las propiedades de la Transformada Discreta de Fourier

J.1 Definiciones:

Se define la Serie de Taylor para $g(x)$ alrededor del punto a de la siguiente forma:

$$g(x) = \sum_{k=0}^{\infty} \frac{g^{(K)}(a)}{k!} (x-a)^k$$

$$g(x) \approx g(a) + g'(a)(x-a) + g''(a)\frac{(x-a)^2}{2!} + \cdots \quad \text{(J.1.1)}$$

J.2 Derivaciones para la expansión 1, capítulo 2:

Se desea comprobar la expresión (2.9.3) aplicando la serie de Taylor hasta el segundo término a la función raíz cuadrada.

$$\sqrt{(f_0+f_\tau)^2-\frac{c^2{f_\eta}^2}{4{V_r}^2}}$$

$$\approx\left[(f_0+f_\tau)-\left(\frac{c^2{f_\eta}^2}{8{V_r}^2(f_0+f_\tau)}\right)+\cdots\right]$$

(J.2.1)

A partir de

$$h(f_\tau)=\sqrt{(f_0+f_\tau)^2-\frac{c^2{f_\eta}^2}{4{V_r}^2}}$$

(J.2.2)

Definamos $g(x)$ como:

$$g(x)=\sqrt{(f_0+f_\tau)^2-x}$$

(J.2.3)

Y la variable x como:

$$x=\frac{c^2{f_\eta}^2}{4{V_r}^2}$$

(J.2.4)

Necesitamos calcular la expresión para la derivada primera de $g(x)$:

$$g'(x)=-\frac{1}{2}\frac{1}{\sqrt{[(f_0+f_\tau)^2-x]}}$$

(J.2.5)

Calculo de los 2 primeros términos de Taylor (J.1.1), para $a=0$

Primer término:

$$g(0)=(f_0+f_\tau)$$

(J.2.6)

Segundo término:

$$g'(0)\,(x-0) = \left(-\frac{1}{2}\,\frac{1}{(f_0+f_\tau)}\right)(x)$$

(J.2.7)

Entonces por Taylor (J.1.1):

$$\begin{aligned} g(x) &= \sqrt{(f_0+f_\tau)^2 - x} \\ &\approx\ g(0) + g'(0)\,(x-0)\ + \cdots \\ &\approx\ (f_0+f_\tau) + \left(-\frac{1}{2}\,\frac{1}{(f_0+f_\tau)}\right) x\ + \cdots \end{aligned}$$

(J.2.8)

Usando (J.2.4) para reemplazar a x:

$$\begin{aligned} &\approx\ (f_0+f_\tau) + \left(-\frac{1}{2}\,\frac{1}{(f_0+f_\tau)}\right)\left(\frac{c^2 {f_\eta}^2}{4{V_r}^2}\right) + \cdots \\ &\approx\ (f_0+f_\tau) - \left(\frac{c^2 {f_\eta}^2}{8(f_0+f_\tau)\,{V_r}^2}\right) + \cdots \end{aligned}$$

(J.2.9)

Con lo cual (J.2.1) queda verificado.

J.3 Derivaciones para la expansión 2, capítulo 2:

Se desea comprobar la expresión (2.9.7) aplicando la serie de Taylor hasta el tercer término a la función raíz cuadrada.

$$\sqrt{(f_0+f_\tau)^2-\frac{c^2{f_\eta}^2}{4{V_r}^2}}$$

$$\approx\left[f_0D(f_\eta)+\left(\frac{f_\tau}{D(f_\eta)}\right)-\left(\frac{{f_\tau}^2}{2\,f_0D_{(f_\eta)}^3}\frac{c^2{f_\eta}^2}{4{V_r}^2{f_0}^2}\right)\right]$$

(J.3.1)

Dónde:

$$D(f_\eta)=\sqrt{1-\frac{c^2{f_\eta}^2}{4{V_r}^2{f_0}^2}}$$

(J.3.2)

A partir de

$$h(f_\tau)=\sqrt{(f_0+f_\tau)^2-\frac{c^2{f_\eta}^2}{4{V_r}^2}}$$

(J.3.3)

Se desarrolla el binomio cuadrado y se multiplica y divide por ${f_0}^2$ dentro de la raíz en (J.3.3) para que aparezca $D_{(f_\eta)}^2$ y obtenemos:

$$h(f_\tau)=\sqrt{\frac{{f_0}^2}{{f_0}^2}\left[\left({f_0}^2+2f_0f_\tau+{f_\tau}^2\right)-\frac{c^2{f_\eta}^2}{4{V_r}^2}\right]}$$

$$h(f_\tau) = f_0 \sqrt{\left(1 + \frac{2f_\tau}{f_0} + \frac{{f_\tau}^2}{{f_0}^2}\right) - \frac{c^2 {f_\eta}^2}{4{V_r}^2}}$$

$$= f_0 \sqrt{\left(1 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}\right) + \frac{2f_\tau}{f_0} + \frac{{f_\tau}^2}{{f_0}^2}}$$

(J.3.4)

Según (J.3.2), tenemos:

$$D^2_{(f_\eta)} = \left(1 - \frac{c^2 {f_\eta}^2}{4{V_r}^2}\right)$$

(J.3.5)

Entonces:

$$h(f_\tau) = f_0 \sqrt{\left(D^2_{(f_\eta)} + \frac{2f_\tau}{f_0} + \frac{{f_\tau}^2}{{f_0}^2}\right)}$$

(J.3.6)

Necesitamos calcular la expresión para la derivada primera y segunda de $h(f_\tau)$

Derivada primera de $h(f_\tau)$:

$$h'(f_\tau) = f_0 \frac{1}{2} \frac{1}{\sqrt{\left(D^2_{(f_\eta)} + \frac{2f_\tau}{f_0} + \frac{{f_\tau}^2}{{f_0}^2}\right)}} \left(\frac{2}{f_0} + \frac{2\,f_\tau}{{f_0}^2}\right)$$

(J.3.7)

Derivada segunda de $h(f_\tau)$:

$$h''(f_\tau) = f_0 \left(-\frac{1}{4} \frac{1}{\sqrt{\left(D^2_{(f_\eta)} + \frac{2f_\tau}{f_0} + \frac{{f_\tau}^2}{{f_0}^2}\right)^3}} \left(\frac{2}{f_0} + \frac{2\,f_\tau}{{f_0}^2}\right)^2 + \frac{1}{2} \frac{1}{\sqrt{\left(D^2_{(f_\eta)} + \frac{2f_\tau}{f_0} + \frac{{f_\tau}^2}{{f_0}^2}\right)}} \left(\frac{2}{{f_0}^2}\right) \right)$$

(J.3.8)

Para calcular los 3 primeros términos de Taylor (J.1.1) en el punto $a = 0$, se necesita conocer los valores para: $h(0)$, $h'(0)$ y $h''(0)$:

Cálculo de $h(0)$, a partir de (J.3.6):

$$h(0) = f_0 \quad D_{(f_\eta)}$$

(J.3.9)

Cálculo de $h'(0)$, a partir de (J.3.7):

$$h'(0) = f_0 \frac{1}{2} \frac{1}{D_{(f_\eta)}} \left(\frac{2}{f_0}\right)$$

$$= \frac{1}{D_{(f_\eta)}}$$

(J.3.10)

Cálculo de $h''(0)$, a partir de (J.3.8):

$$h''(0) = f_0\left(-\frac{1}{4}\frac{1}{\sqrt{\left(D^2_{(f_\eta)}\right)^3}}\left(\frac{2}{f_0}\right)^2 + \frac{1}{2}\frac{1}{\sqrt{\left(D^2_{(f_\eta)}\right)}}\left(\frac{2}{{f_0}^2}\right)\right)$$

$$= -\frac{1}{4}\frac{1}{\sqrt{\left(D^2_{(f_\eta)}\right)^3}}\frac{4}{f_0} + \frac{1}{2}\frac{1}{\sqrt{\left(D^2_{(f_\eta)}\right)}}\left(\frac{2}{f_0}\right)$$

$$= -\frac{1}{f_0\sqrt{\left(D^2_{(f_\eta)}\right)^3}} + \frac{1}{f_0\sqrt{\left(D^2_{(f_\eta)}\right)}}$$

$$= -\frac{1}{f_0 D^3_{(f_\eta)}} + \frac{1}{f_0 D_{(f_\eta)}}$$

(J.3.11)

Calculando Taylor (J.1.1):

$$h(f_\tau) \approx h(0) + h'(0)\,f_\tau + h''(0)\,\frac{{f_\tau}^2}{2} + \cdots$$

$$\approx f_0\ D_{(f_\eta)} + \frac{1}{D_{(f_\eta)}}\,f_\tau + \left(-\frac{1}{f_0 D^3_{(f_\eta)}} + \frac{1}{f_0 D_{(f_\eta)}}\right)\frac{{f_\tau}^2}{2} + \cdots$$

$$\approx f_0\ D_{(f_\eta)} + \frac{f_\tau}{D_{(f_\eta)}} - \frac{{f_\tau}^2}{2f_0 D^3_{(f_\eta)}} + \frac{{f_\tau}^2}{2f_0 D_{(f_\eta)}} + \cdots$$

$$\approx f_0\ D_{(f_\eta)} + \frac{f_\tau}{D_{(f_\eta)}} - \frac{{f_\tau}^2}{2f_0 D^3_{(f_\eta)}} + \frac{{f_\tau}^2 D^2_{(f_\eta)}}{2f_0 D^3_{(f_\eta)}} + \cdots$$

$$\approx f_0\ D_{(f_\eta)} + \frac{f_\tau}{D_{(f_\eta)}} + \frac{{f_\tau}^2}{2f_0 D^3_{(f_\eta)}}\left(D^2_{(f_\eta)} - 1\right) + \cdots$$

Por (J.3.2):

$$D_{(f_\eta)}^2 - 1 = -\frac{c^2 {f_\eta}^2}{4{V_r}^2 {f_0}^2}$$

Reemplazando:

$$\approx f_0 \quad D_{(f_\eta)} + \frac{f_\tau}{D_{(f_\eta)}} - \frac{{f_\tau}^2}{2f_0 D_{(f_\eta)}^3} \frac{c^2 {f_\eta}^2}{4{V_r}^2 {f_0}^2} + \cdots$$

Con lo cual se finaliza y queda verificado (J.3.1).