Extended Chirp Scaling In GPGPU

S. Abbate, J. Gonzalez, S. Gutierrez, J. Areta and M. Denham

***Abstract*— We present in this article the implementation of Chirp Scaling Algorithm for SAR systems signal processing. This algorithm takes advantage of Chirp signals properties to process the information acquired from aerial radar systems and output high resolution images of earth ground. We propose a parallel implementation of the algorithm in GPGPU (General Purpose Graphic Processing Unit) using CUDA C. Several bottlenecks and penalizations are detected and corrected within various versions of the implementation, reducing processing time.**

***Keywords*— Signal Processing, Chirp Scaling Algorithm, Graphic Cards, GPGPU, SAR.**

I. INTRODUCCIÓN

LA INDUSTRIA del videojuego ha generado un gran avance tecnológico en el campo de la computación paralela gracias al desarrollo de herramientas de programación orientadas a la inclusión de las unidades de procesamiento gráfico (GPU), como co-procesador genérico de una CPU con el objetivo de liberar a la CPU del proceso de renderizado de las aplicaciones gráficas. Esto ha permitido utilizar la gran capacidad de cálculo de las placas gráficas para aplicaciones de propósito general, denominadas GPGPU (General Purpose Graphics Processing Units), y constituyen actualmente una pieza fundamental en la computación de alto rendimiento.

Se propone en este artículo, la utilización de una arquitectura CPU-GPGPU para el procesamiento de datos de un sistema SAR, basado en el algoritmo CSA (Chirp Scaling Algorithm). La elección de esta arquitectura se fundamenta en el tipo de operaciones a realizar, en las que puede aprovecharse el concepto de múltiples hilos de ejecución concurrentes operando sobre un gran volumen de datos.

La siguiente sección detalla las características fundamentales de un sistema SAR. En la sección 3 se presenta el algoritmo ECS (Extended Chirp Scaling) que es una extensión del Chirp Scaling Algorithm, y fue implementado en este trabajo [4]. En la sección 4 se describe brevemente la arquitectura y modo de ejecución de la GPU y su relación con las operaciones del algoritmo. En la sección 5 se presenta un detalle de las versiones realizadas de la implementación, para describir finalmente los resultados obtenidos, conclusiones y propuestas de trabajo a futuro.

II. FUNDAMENTOS DE SAR

Un sistema de Radar de Apertura Sintética (SAR) combina fundamentos de radar y herramientas de procesamiento de señales para crear el efecto de una antena equivalente de grandes dimensiones. Este sistema es frecuentemente utilizado para la captura de imágenes de la superficie terrestre desde una plataforma satelital o aerotransportada. Es un sistema de captura de imágenes activo, en el cual se emiten pulsos electromagnéticos en dirección aproximadamente perpendicular al movimiento de la plataforma (dirección de acimut), y se reciben los ecos de esos pulsos de manera coherente a medida que se reflejan en la superficie. En un esquema tradicional de radar, la resolución que se obtiene en acimut es inversamente proporcional a la apertura de la antena utilizada. Sin embargo, en un sistema SAR puede aproximarse como la mitad de la longitud de la antena [1], pudiendo emplear así, antenas de dimensiones más reducidas y logrando una gran mejora en la resolución en acimut del sistema, compatible con aplicaciones de alta resolución como los ópticos.

En la Fig. 1 se observa un sistema simplificado en el cual se considera un objetivo hipotético de la superficie terrestre. Se denomina acimut a la dirección de vuelo de la plataforma y rango a la dirección en la que se envían los pulsos.

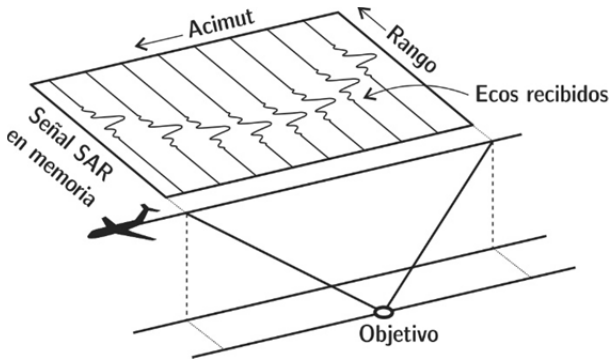


Figura 1. Geometría del sistema SAR y almacenamiento de los datos crudos en una matriz (ecos recibidos) [1].

Compresión de pulso y resolución

Intuitivamente, para poder discriminar dos objetivos cercanos entre sí, la duración del pulso enviado debe ser lo suficientemente corta como para caber en la distancia que los separa. Como un pulso de corta duración es una señal de gran ancho de banda, presenta una correlación angosta (y de allí su buena resolución espacial), puede comprimirse un pulso de mayor duración modulando en frecuencia y luego procesando mediante un filtro adaptado. Se logra de esta manera, al igual que en una señal de corta duración, un pulso de correlación angosto.

La resolución que se obtiene en un sistema en el que se envían pulsos de energía estará determinada entonces, por el ancho de banda de la señal transmitida [1] [3].

Señal SAR bidimensional

Cada eco recibido se digitaliza y almacena en memoria

S. Abbate, UNRN student, santiabbate@gmail.com
J. Gonzalez, UNRN student, joacoasr@gmail.com
S. Gutierrez, UNRN student, silvinaggutierrez@gmail.com
J. Areta, UNRN, CONICET, jareta@unrn.edu.ar
M. Denham, UNRN, CONICET, mdenham@unrn.edu.ar

como se muestra en la Fig. 1. De esta manera se obtiene una señal de dos dimensiones del sistema de radar, siendo sus coordenadas, Tiempo en rango y Tiempo en acimut. El tiempo en rango es comúnmente denominado "Tiempo rápido" al estar relacionado con la distancia por la velocidad de propagación de los pulsos electromagnéticos (velocidad de la luz), mientras que el tiempo en acimut se denomina "Tiempo lento" ya que las distancias asociadas a él están dadas por la velocidad de desplazamiento de la plataforma y por la PRF (frecuencia de repetición de pulso - Pulse Repetition Frequency).

Los pulsos emitidos por un sistema SAR son generalmente señales de frecuencia moduladas en las que la frecuencia instantánea es una función lineal en el tiempo. Se conoce a estas señales con el nombre de "chirp" y presentan un ancho de banda que responde al alcance del barrido en frecuencia de la modulación [1]. Este ancho de banda es el que determina la resolución que se obtiene en rango.

Frecuencia Doppler

A medida que el radar avanza enviando pulsos, un determinado objetivo puntual es iluminado desde varias posiciones y se reciben ecos tanto al acercarse como al alejarse. Esto genera que la señal recibida se vea afectada por un corrimiento Doppler que va aumentando a medida que se acerca al objetivo, y disminuyendo a medida que se aleja.

Puede modelarse la inclusión de esta frecuencia Doppler como una modulación aproximadamente lineal en frecuencia de la señal en la dirección de acimut [1], obteniendo también así, una señal tipo chirp en la dirección de acimut.

Migración de Celda en Rango

El efecto de migración de celda en rango (Range Cell Migration - RCM) es inherente a un sistema de radar SAR y deriva del movimiento de la plataforma y la configuración geométrica del sistema. Al ir "iluminando" con los pulsos transmitidos una zona del suelo durante el trayecto en acimut, se obtiene información de un mismo objetivo desde distintas posiciones. Sin embargo, la distancia al objetivo no se mantiene constante, y en la señal adquirida, un mismo punto en el suelo ocupa en memoria, distintas posiciones en rango (Fig. 3 (a)).

Para poder independizar el procesamiento en operaciones de una sola dimensión es necesario corregir este efecto de forma de alinear la información que se corresponde a un único punto a una sola posición en rango (RCMC).

III. EXTENDED CHIRP SCALING

Puede considerarse a un sistema SAR como un sistema lineal, caracterizado por una respuesta al impulso dada [1] [2]. El objetivo del procesamiento es el de realizar la operación de "deconvolución" de la respuesta del sistema con el perfil de reflectividad del suelo, que es la información utilizada para la formación de imágenes.

Existen diversos algoritmos de procesamiento caracterizados principalmente por el dominio en el que trabajan, ya sea temporal o frecuencial, cada uno aprovechando propiedades del dominio en búsqueda de eficiencia y precisión.

Se describe a continuación el algoritmo utilizado, conocido como Extended Chirp Scaling [4], cuyo esquema se muestra en la Fig. 2. Se indica el dominio en el cual se realizan las operaciones, siendo el dominio "Rango-Doppler" el resultante de la aplicación de la transformada de Fourier en la dirección de acimut, y el dominio frecuencial el resultante de la aplicación de la transformada de Fourier en ambas direcciones. *τr*, *τa* son las dimensiones de "Tiempo en rango" y "Tiempo en acimut", y *fr*, *fa* "Frecuencia en rango" y "Frecuencia en acimut" (luego de los cambios de dominio).

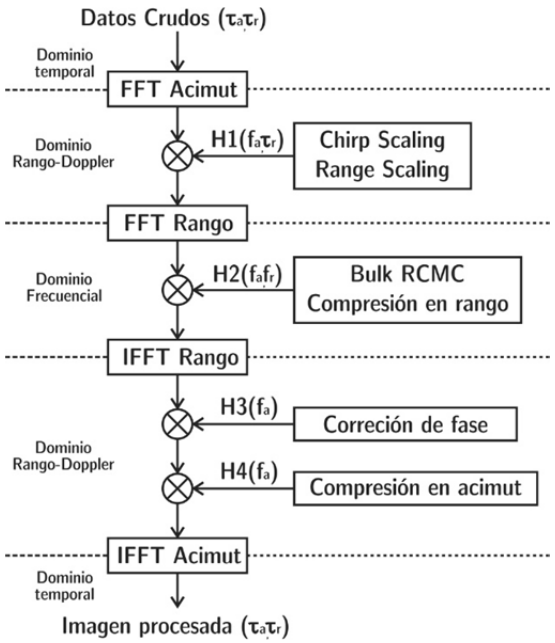


Figura 2. Diagrama en bloques del algoritmo utilizado basado en [4].

El procesamiento completo realiza las siguientes operaciones,

(i) Corrección de migración de celda (Range Cell Migration Correction y Chirp Scaling);
(ii) Cambio de escala en rango (Range Scaling);
(iii) Compresión en rango;
(iv) Compresión en acimut;

resumidas en la aplicación de cuatro filtros o funciones de fase como se ve en la Fig. 2 y cuya estructura es

$$H_x(a,b) = exp[\phi(a,b)] \qquad (1)$$

donde los argumentos particulares correspondientes a cada función de fase están descriptos detalladamente en [4].

RCMC (Bulk RCMC y Chirp Scaling)

El objetivo de la operación RCMC es el de alinear en rango la información correspondiente a la trayectoria en acimut de un objetivo. El algoritmo ECS realiza esta corrección en dos etapas, multiplicando los datos por funciones chirp que agregan un adecuado corrimiento de fase, en un caso un desplazamiento constante (bulk) y en otro diferencial (chirp scaling), resultando en un desplazamiento

apropiado de las señales recibidas de forma que un mismo objeto en tierra posee sus mediciones alineadas en la matriz de datos.

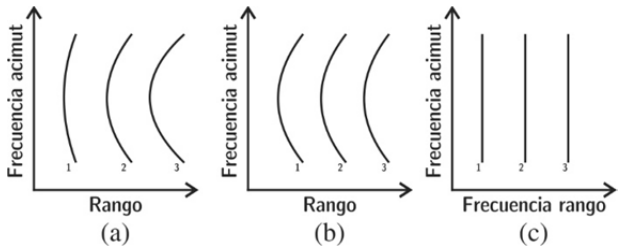


Figura 3. Trayectoria de 3 objetivos puntuales. (a) Efecto de RCM, (b) Corrección mediante Chirp Scaling (Filtro 1), (c) Corrección mediante Bulk RCMC (Filtro 2).

Puede observarse como, a pesar de estar dispersa en acimut, la información de cada objetivo en la Fig. 3 (c) se corresponde con una única posición en rango. Esta dispersión en acimut luego se corrige en la compresión en acimut (aplicando el filtro 4, H4 en la Fig. (2)).

IV. ECS EN GPGPU

Arquitectura GPU

La evolución de la industria ha llevado a desarrollar dos filosofías distintas de diseño de arquitectura de hardware. Una de ellas es la multinúcleo implementada en microprocesadores (CPU), la cual, busca mantener la velocidad de ejecución de los programas secuenciales mientras va cambiando su ejecución en múltiples núcleos. Otra de ellas, es la implementada con muchos núcleos, que se centra más en la ejecución de rendimiento de aplicaciones paralelas. Cada uno de estos núcleos es un procesador multihilo que comparte su control y memoria cache de instrucciones con otros núcleos. Las GPUs, son un ejemplo de este tipo de arquitectura [10] [11] [8].

La utilización de GPUs es conveniente para aplicaciones que tengan gran requerimiento de cómputo, cuyas tareas puedan asignarse a hilos (threads) que se ejecuten en forma paralela. Las GPUs poseen planificadores y manejadores de bloques, warps, instrucciones, etc. que hacen un uso eficiente de las distintas unidades de procesamiento y memorias instaladas en las mismas [8] [12].

CUDA

CUDA (Computer Unified Device Architecture) es una herramienta que permite desarrollar aplicaciones en un entorno híbrido CPU-GPU, aprovechando las características intrínsecas a cada una y de esta forma lograr aplicaciones de alto rendimiento.

Con respecto a la GPU, CUDA brinda organización del trabajo paralelo a través de hilos de ejecución concurrentes y acceso a la jerarquía de memoria de la GPU.

Los hilos en el modelo CUDA son agrupados en bloques, y poseen memoria compartida (shared), la cual pueden usar como medio de comunicación entre ellos. A su vez, varios bloques forman una grilla (grid) siendo los hilos de diferentes bloques, independientes. Además de a la memoria compartida, los hilos tienen acceso a otros tipos de memorias: memoria local (privada de cada uno), memoria global (compartida por hilos de distintos bloques), y dos espacios adicionales de memoria: la memoria de constantes y la memoria de texturas, ambas de solo lectura para los hilos y de escritura para la CPU.

Como se mencionó anteriormente, un programa CUDA trabaja en un entorno híbrido en el que se diferencian dos ámbitos, el host y el device. El host es la computadora (CPU) al cual está conectada la tarjeta gráfica mediante un bus PCI-Express, y será quien gobierne su comportamiento. El device es la tarjeta gráfica (dispositivo). La comunicación de datos entre el CPU y GPU se lleva a cabo a través de la memoria global, de constante y de textura [10] [11] [12].

Entre otras cosas, CUDA es una extensión del lenguaje de programación C. CUDA permite definir funciones llamadas kernels, las cuales son invocadas desde la CPU y ejecutadas en el dispositivo. Cuando se invoca un kernel, se lanzan N threads que ejecutan dicho kernel. Es posible identificar de forma unívoca a cada uno de los hilos, lo que hace posible el control de la ejecución y así aprovechar el concepto de arquitectura de programación SIMD (Simple Instruction Multiple Data), al ejecutar simultáneamente cada hilo de un kernel, las mismas operaciones sobre un dato contenido en conjunto de datos más grande. CUDA extiende este concepto a SIMT (Simple Instruction Multiple Thread) que tiene como diferencia que cada hilo ejecuta de forma independiente [13].

Características del algoritmo ECS

Los algoritmos de procesamiento de señales SAR en general, y el algoritmo ECS en particular, operan sobre una gran cantidad de datos (las imágenes frecuentemente están formadas por millones de píxeles), realiza operaciones con altos requerimientos computacionales, no hay dependencia de datos, entre otras características. Estos aspectos convierten a las GPUs en una arquitectura muy conveniente para este tipo de aplicaciones [10] [11].

Los filtros que componen el algoritmo (Fig. 3), descriptos en la sección 3, están caracterizados por ser funciones de valor complejo y bidimensionales (ya sea en tiempo o en frecuencia) [4], al igual que los datos adquiridos por el sistema SAR (Sección 2). Entonces, tanto los datos como las funciones de fase, pueden representarse numéricamente de forma matricial. La aplicación de los filtros en las distintas etapas del algoritmo, se corresponde con un producto elemento a elemento entre las matrices señal y filtro.

Otra característica del algoritmo es que se realizan “cambios de dominio” mediante la transformada y transformada inversa de Fourier. De esta forma, el algoritmo ECS se basa en: (i) armado en forma matricial de los filtros, (ii) producto de matrices elemento a elemento, (iii) transformadas y antitransformadas de Fourier.

V. PRUEBAS EXPERIMENTALES

En la Tabla 1 se muestra la arquitectura utilizada en las pruebas experimentales. A continuación se exponen las

principales características de las distintas versiones del algoritmo ECS. Se ha propuesto una primer versión paralela y luego optimizaciones para lograr reducir los tiempos de cómputo. Los tiempos de cómputo de cada versión se muestran en la Tabla 2 en la siguiente sección.

TABLA I
ARQUITECTURA DEL HARDWARE UTILIZADO

CPU	Intel Core i5-2500K @ 3.30GHz
GPU	NVIDIA GeFOrce GTX 570. 480 CUDA Cores
Memoria	1280MB
SO	Ubuntu 12.04LTS / LINUX
CUDA	Versión 5.5 (CUDA compiler version V5.5.0)

Versión 0 (secuencial)

El paso inicial es la implementación del algoritmo para su ejecución en arquitectura CPU. Se implementan las funciones de armado de filtros y producto punto de matrices, mientras que para las operaciones de cambio de dominio se recurre a una librería que implementa la Transformada Rápida de Fourier (FFT). La librería utilizada es la FFTW3 [5]. Visto que la librería trabaja aplicando transformadas unidimensionales en una sola dirección se implementaron funciones para trasponer matrices, paso previo a la transformada en algunos casos. Esta primera versión ayuda a verificar correctitud del método como así también detectar cuellos de botella de la aplicación.

Versión 1

Se implementan kernels para ejecución de funciones de armado de los filtros, y las operaciones de multiplicación´ y traspuesta de matrices. Esta primer paralelización resulta ineficiente ya que para el armado de los filtros se ejecutan funciones tanto en la CPU como en la placa gráfica. Se observa como principal inconveniente en esta versión, diversas transferencias de datos entre CPU y GPU disminuyendo la proporción del tiempo de procesamiento frente a transferencia de datos.

Se recurre a la librera CUFFT para la implementación de las transformadas de Fourier (FFT e IFFT) [6].

Versión 2

Con el objetivo de minimizar la cantidad de transferencias de datos se implementan algunas modificaciones en el armado de los filtros. En versiones anteriores se inicializaban matrices preliminares para conformar los filtros, usando los parámetros del sistema SAR. Se observa que hay información redundante en estas matrices al tener todas sus filas o columnas iguales y se implementa entonces, el armado de los filtros directamente mediante kernels en GPU en base a vectores inicializados en la CPU. Se minimiza así la transferencia de datos al mover vectores de CPU a GPU, en lugar de matrices.

Aplicando este primer paso de optimización se observa una ganancia de más de 10 veces en el tiempo de ejecución total cuando se compara con la V1 (primer versión paralela).

Versión 3

La librería CUFFT implementa de forma eficiente la transformada discreta de Fourier aplicando el algoritmo Cooley-Tukey en distintas bases (radix-2, radix-3, radix-5 y radix-7). Es por eso que una transformada de dimensiones que puedan factorizarse en alguna de esas bases, estará optimizada por CUFFT [6]. Se cambian entonces, las dimensiones de los datos a procesar (2000x4000) por valores correspondientes a potencias de 2 (2048x4096). Se cumple así, con una de las recomendaciones de la librería en la que se sugiere utilizar dimensiones expresadas en una sola de las 4 bases posibles [6].

No se observa una ganancia aparente en las operaciones de FFT e IFFT. Esto puede deberse a que CUFFT está realmente optimizada para transformadas cuyas dimensiones puedan ser expresadas en 4 bases distintas. Se propone como hipótesis que las recomendaciones de la librería se hacen más evidentes ante transformadas de mayor dimensión a las que se trabajaron.

Versión 4

Se optimiza la creación de los filtros, armando en la placa grafica los arreglos base utilizados para conformarlos, en función de los parámetros del sistema.

En este caso, los tiempos son similares a la versión anterior. Se esperaba una ganancia mayor, ya que se evitan iteraciones secuenciales en CPU y se reemplazan por ejecuciones de threads en GPU (para inicializar los vectores de funciones de fase).

Otra prueba que se ha realizado se basa en evitar un lanzamiento de kernel por cada vector base utilizado, se agrupa la implementación de la Versión 4, en un solo llamado a kernel mediante el cual se generan en la placa grafica todos los vectores necesarios para cada filtro.

La memoria global de la GPU es suficiente para alojar todos estos vectores. Hay vectores que se utilizan en más de una función de fase, entonces, cada vector se mantiene en memoria mientras la aplicación lo necesite.

Esta optimización propuesta no resulta en una reducción de tiempo importante, lo que muestra que para esta aplicación, el lanzamiento de kernels no significa penalizaciones en los tiempos de cómputo.

Implementación de la FFT

Visto que el algoritmo realiza las operaciones de FFT e IFFT en ambas direcciones (rango y acimut), es necesario trasponer la matriz de datos ya que la librería CUFFT está implementada únicamente en una dirección. Con el objetivo de eliminar el tiempo de cómputo correspondiente a la traspuesta de la matriz y poder realizar la transformada en ambas direcciones sobre la misma matriz de datos, se implementa la FFT según [7]. Sin embargo, al realizar una implementación propia no se logran equiparar los tiempos de ejecución logrados por CUFFT, inclusive para arreglos de dimensiones mucho menores, y se determina, por lo tanto, seguir utilizando la librería cuya implementación está muy optimizada.

VI. RESULTADOS

Tiempos de etapas del algoritmo

Habiendo detallado las distintas optimizaciones realizadas sobre el algoritmo paralelo inicial, los tiempos de ejecución son presentados en la tabla 2, donde se detallan los tiempos para cada una de las operaciones y las distintas versiones propuestas. Para el caso de los filtros se toma el tiempo de armado del filtro y el producto con la señal.

TABLA II
TIEMPOS OBTENIDOS, EN MILISEGUNDOS

Función	V1	V2	V3	V4
FFT en Acimut	159.5	163.0	161.7	160.7
1ra FF	461.8	15.4	15.3	15.4
FFT en Rango	11.3	11.2	11.3	10.6
2da FF	1193.4	9.8	9.7	11.5
IFFT en Rango	15.1	154	15.3	15.1
3ra FF	400.8	9.6	9.6	9.1
4ta FF	750.3	8.9	9.0	7.0
Total	3015.0	258.2	256.8	254.5

(FF: Función de Fase)

Tiempos CUFFT

La API de CUFFT provee un mecanismo de configuración denominado plan mediante el cual se configura la ejecución de la FFT o IFFT de acuerdo a la operación a realizar y la configuración de hardware instalada.

Se observó que la mayor carga de tiempo de ejecución de la aplicación recae sobre las transformadas y anti transformadas y en especial en la creación del primer plan. Se detallan en la Tabla 3 las mediciones realizadas sobre las operaciones de FFT e IFFT, discriminando para cada una el tiempo de creación del plan. Se muestran solo los tiempos de la V1 y V4 ya que son muy similares los tiempos observados en todas las versiones.

TABLA III
TIEMPOS DE OPERACIONES CON LIBRERÍA CUFFT

Operación	V1	V4
Plan FFT en Acimut	140.4	139.9
FFT en Acimut	159.5	160.7
Plan FFT en Rango	0.286	0.182
FFT en Rango	11.3	11.5
Plan IFFT en Rango	0.233	0.234
IFFT en Rango	15.1	15.1
Plan IFFT en Acimut	0.001	0.001
IFFT en Acimut	22.8	25.1

Se observa que la primera llamada a la función para crear un plan significa aproximadamente el 60 % del tiempo total de cómputo. En la bibliografía consultada se ha podido constatar que la primera llamada a una función de esta API inicializa estructuras y datos internos. En este hecho se reconoce la penalización en el rendimiento. De todas formas, si se considera esta penalización, y se divide por todas las FFT e IFFT que se resuelven en la aplicación (2000 2 + 4000 2) esta penalización sigue siendo conveniente a la alternativa de no utilizar CUFFT.

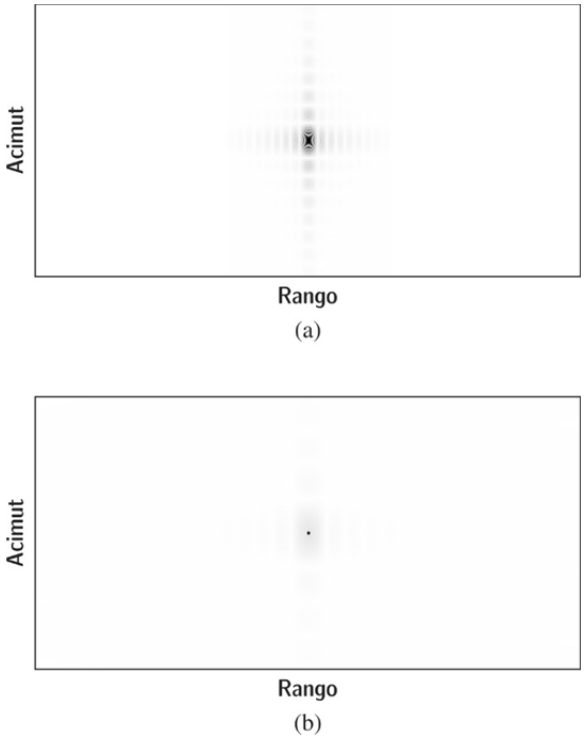


Figura 4. (a) Datos crudos de un objetivo puntual simulado, (b) Imagen procesada (objetivo enfocado).

VII. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se presenta una implementación optimizada para GPGPU del algoritmo ECS. El estudio del algoritmo ECS tanto como el estudio de las placas gráficas mostraron que este tipo de aplicación puede llegar a obtener muy buen rendimiento en dichas placas.

Se parte de una solución secuencial, la cual fue estudiada e instrumentada a fin de detectar cuellos de botella de la aplicación como así también oportunidades de paralelización y división del trabajo.

Luego se implementó una primer versión paralela, la cual fue una solución directa, aprovechando las características SIMD del algoritmo ECS. A partir de esta aplicación´ como así también del estudio de las placas GP-GPU, de librerías (CUFFT por ejemplo), se propusieron e implementaron diversas optimizaciones en función del análisis de los tiempos de procesamiento.

Las sucesivas optimizaciones mostraron una buena ganancia en la Versión 2, donde se utiliza la placa para la inicialización de los vectores auxiliares y de las funciones de fase en la placa gráfica, evitando así procesamiento secuencial en CPU como así también transferencia de datos CPU-GPU. El resto de las optimizaciones no han mostrado grandes ganancias, pero se considera que en aplicaciones con mayor cantidad de datos esto puede tener una influencia más notoria.

REFERENCIAS

[1] I. CUMMING, F, WONG. *Digital Processing of Synthetic Aperture Radar Data.* Artech House, Capítulos *3,4*. 2005.

[2] M. RICHARDS. *Fundamentals of Radar Signal Processing.* McGraw-Hill, Capítulo *8*. 2005.

[3] M. SKOLNIK. *RADAR Systems*. McGraw-Hill, Capítulos *1,2.* 2000.

[4] A. MOREIRA, J. MITTERMAYER, R. SCHEIBER. *"Extended Chirp Scaling Algorithm for Air and Spaceborne SAR Data Processing in Stripmap and ScanSAR Imaging Modes".* Geoscience and Remote Sensing, IEEE Transactions, volumen 34, número 5, pp. 1123–1136. 1996.

[5] M. FRIGO, S. G. JOHNSON. *FFTW Home Page.* 2013. URL: hrrp://www.fftw.org

[6] NVIDIA CORPORATION. *CUFFT Library. User Guide,* Sección *2.9*. 2014.

[7] K. MORELAND, E. ANGEL. *The FFT on a GPU.* Graphics Hardware. 2003.

[8] R. FARBER. *Application Design and Development.* Morgan Kaufmann Publishers Inc. 2011

[9] NVIDIA CORPORATION. *CUFFT Library. User Guide*. 2014.

[10] F. PICCOLI. *Computación de Alto Desempeño en GPU*, Universidad Nacional de San Luis, Sección *3*.3. 2011.

[11] J. SANDERS, E. KANDROT. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional. 2010.

[12] J. COOK. *CUDA Programming. A Developer's Guide to Parallel Computing with GPUs.* Morgan Kaufmann. 2013.

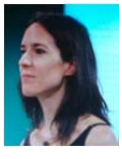
[13] J. CHENG, M. GROSSMAN, T. MCKERCHER. *Professional CUDA C Programming.* Wrox, Capítulo *3*. 2014.

Santiago Abbate, began his Electronics Engineering studies in 2011, at "Universidad Nacional de Río Negro". He was involved in projects regarding parallel processing of SAR systems signals and evaluation and design of SONAR systems within a Scientific stimuli scholarship. He is developing, at the present, a technological evaluation for fresh water bathymetries as final thesis project.

Rodrigo Joaquin Gonzalez, Electronics Engineering student at "Universidad Nacional de Río Negro" finished his high school studies at "Don Bosco" school in 2005. Member of parallel algorithms processing for SAR systems project.

Silvina Gabriela Gutierrez, received a degree in "Automation and Control Systems technician" form "Escuela Cooperativa Técnica Los Andes", Argentina. She started working in INVAP in 2011 within Argentina Radarization Program. She is actually studying 5th year of Electronics Engineering at "Universidad Nacional de Río Negro".

Javier A. Areta, graduated as an Electronics Engineer from the National University of La Plata, Argentina, in 2001, and received the Ph.D. degree in Electrical Engineering from the University of Connecticut in 2008. Currently, he is a research associate of the National Council of Scientific and Technical Research of Argentina (CONICET) and Associate Professor at the Department of the Electrical Engineering of the National University of Río Negro in Bariloche, Argentina. His research interests are in the area of statistical signal processing, detection and estimation theory, and their applications in sensor arrays and remote sensing systems.



Monica Denham, graduated as System Analyst in 2003 and obtained her degree in computer science in 2005 both from National University of La Plata, Argentina. In 2007 she received her Computer Science researcher degree, and Ph.D. in computer science in 2009 form "Universidad Autónoma de Barcelona", Spain. Her research is focused in parallel computing and high performance computing. At the moment her work involves radar signal processing and prediction systems applications.