# Procesamiento de Señales SAR en GPGPU

Mónica M. Denham[1,2], Javier A. Areta[1]

[1] Universidad Nacional de Río Negro, Río Negro, Argentina
[2] Instituto de Investigación en Informática III-LIDI, Universidad Nacional de La Plata, La Plata, Argentina
mdenham@unrn.edu.ar, jareta@unrn.edu.ar

**Abstract.** Este trabajo tiene como objetivo presentar las principales características del diseño e implementación de algoritmos paralelos para el procesamiento de señales de radar de apertura sintétitca (SAR). Se analizan las razones por las que el problema es paralelizable y se presentan implementaciones de algoritmos clásicos para la obtención de imágenes. También se presentan adaptaciones de estos algoritmos para ser utilizados sobre arquitecturas paralelas de tipo GPU de propósito general. Estas modificaciones son propuestas con el objetivo de obtener algoritmos altamente eficientes y escalables en este tipo de arquitecturas.

## 1 Introducción

En este trabajo se aborda el procesamiento de señales de radares SAR (*Synthetic Aperture Radar*) utilizando algoritmos implementados en forma paralela en GPU (*Graphic Processing Unit*) de propósito general. Este procesamiento se basa en la combinación de múltiples "mediciones" (datos recibidos por un sensor, en este caso un radar) para generar información útil al ser humano. El tipo de procesamiento realizado puede aplicarse también en problemas tales como procesamiento de señales sonar [5], sistemas de detección de movimientos sísmicos, arreglos de sensores terrestres, etc. Luego, el obtener soluciones paralelas eficientes para uno de estos dominios puede generalizarse a otros campos.

Si bien existen diversos tipos de procesamiento de señales SAR, la aplicación más difundida es la de generar imágenes de la superficie terrestre de alta resolución. Estas imágenes se utilizan en áreas de diversa índole como cartografía, teledetección, agronomía, estudio de corrientes marinas, estudio de dispersión de fluidos en océanos o mares, detección de cambios terrestres, información de respuesta de emergencia, etc. En la actualidad, estos radares generan imágenes fiables de alta resolución (por ejemplo en [12] se reportan resoluciones de $1m$) independientemente de las condiciones meteorológicas y de iluminación.

Los radares SAR son radares activos que trabajan usualmente en el orden de las microondas, característica que hace posible que opere de forma correcta en presencia de nubes, cobertura de copas de árboles, etc. Son radares de pequeñas dimensiones que se acoplan a aeronaves (aviones o satélites) y que aprovechan la trayectoria de los mismos para "barrer" la superficie terrestre y sintetizar imágenes a partir del envío de múltiples pulsos y la combinación de los ecos de

dichos pulsos recibidos por el radar (esquema en figura 1). La energía de dicho eco recibido es proporcional a la reflectancia del terreno [3] [1]. Al ser este tipo de radar activo tiene su propia fuente de energía, por lo que no depende de factores externos para poder operar (luz solar por ejemplo).

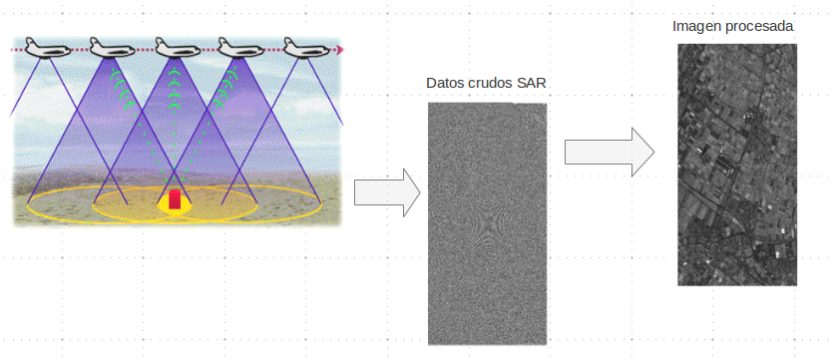


**Fig. 1.** Envío de pulsos y recepción de ecos, datos crudos SAR y datos procesados.

El principio básico de funcionamiento de un radar clásico es el de medición del tiempo de viaje de un pulso enviado desde el radar, siendo este tiempo proporcional a la distancia a la que se encuentra el objeto que lo ha reflejado. Este tipo de radar tiene en general una buena resolución en rango pero su resolución en acimut decae en forma proporcional a la distancia a la que se encuentra el objeto. Este decaimiento de la resolución con la distancia está relacionado con la apertura de la antena, que a su vez es inversamente proporcional a las dimensiones de la misma. Por ello el uso de radares convencionales para la generación de imágenes es inadecuado ya que para obtener una adecuada resolución el tamaño de la antena resulta prohibitivo (esto depende a su vez de la longitud de onda utilizada, pero resoluciones del orden del metro en banda X implicarían antenas del orden de la centena de metros para aplicaciones aerotransportadas).

Para obtener una antena con gran apertura sin necesidad de que esta antena esté compuesta por un solo elemento, se propuso el uso de distintas mediciones de radar y su posterior combinación coherente, de manera de refinar los valores estimados. Para ello se recurre a tomar muestras temporales consecutivas de radar que iluminen un área común, lo que puede verse en forma análoga a tener un arreglo de sensores "sintético" –además de asincrónico (figura 1)– y de gran apertura (proporcional a la distancia recorrida por el sensor móvil en el tiempo que dura la recolección de datos). Estos datos se procesan luego conjuntamente para obtener lo que se denomina una medición de apertura sintética.

La información generada por el radar (datos crudos) se almacena en matrices y corresponde a dos dimensiones de trabajo: las mútiples filas son cada uno de los ecos recibidos (en la dirección de movimiento del sensor, o acimut) y cada fila está compuesta por celdas que corresponden con las muestras en rango. Este

modo de operación genera un gran volumen de datos. A modo de ejemplo, se toma un escenario real descrito en [6]: radar que opera en el satélite ERS donde una escena de 100x100km$^2$, genera una matriz de datos crudos de 26800 líneas y cada línea (fila) está formada por 5616 píxeles. Esto genera 300MB de datos crudos a procesar. Los cálculos son realizados en punto flotante, lo que genera una matriz de 1.2GB de procesamiento. A su vez, en este escenario, la salida es una matriz de 500MB aproximadamente: 25000 líneas de 4912 píxeles, cada uno codificado como número complejo de 2 + 2 bytes.

Luego, el procesamiento de señales SAR implica el procesamiento de un gran volumen de datos. A su vez, las aplicaciones actuales generan la necesidad de aumentar la precisión de las imágenes: esto implica algoritmos más complejos sobre los datos, lo que resulta en tiempos de cómputo mayores. Se observa también que el requerimiento de obtener estas imágenes en tiempo real (o cercano al tiempo real) es cada vez mayor.

Todas estas características llevan a que este tipo de aplicaciones sea un gran desafío desde el punto de vista de la tecnología *High Performance Computing*.

Una forma natural de enfrentar este problema es a través del cómputo paralelo. En este trabajo se propone utilizar la arquitectura GPGPU (*General Purpose Graphic Processing Unit*). La arquitectura GPU nace para realizar cómputo gráfico, pero debido a su alto rendimiento en cómputo complejo los investigadores han propuesto el uso de estos dispositivos para acelerar soluciones a problemas físicos como son análisis de flujos de fluidos, transformadas de Fourier (FFT), crecimiento de cristales, etc [1].

Dada su alta capacidad de procesamiento y su bajo costo, el uso de esta tecnología es cada vez mayor, se estima que su uso será masivo y en un futuro arquitecturas paralelas de tipo *cluster* estarán formados por nodos donde cada nodo sea una o múltiples placas GPU conectadas.

Para obtener aplicaciones eficientes y escalables en este tipo de arquitectura es necesario tener en cuenta sus características de hardware como así también del software relacionado a la programación en las mismas. Es necesario luego realizar un diseño de las aplicaciones específicas para GPU, aplicaciones que maximicen el alto poder de paralelismo y de cómputo de dichas placas.

El estudio de la bibliografía relacionada al procesamiento de señales SAR muestra la tendencia de soluciones desarrolladas para *clusters* de computadoras, y muy poco se observa en relación al desarrollo en GPGPU.

Este trabajo propone el diseño e implementación de dos algoritmos para el procesamiento de señales SAR: RDA (*Range Doppler Algorithm*) y CSA (*Chirp Scaling Algorithm*) [1] [3] [11]. Estos algoritmos son ampliamente utilizados en este dominio y en este trabajo significan un primer paso en el desarrollo de procesamiento de señales SAR.

Dichos algoritmos serán específicamente desarrollados para GPGPU, donde se intentará lograr algoritmos altamente eficientes y escalables.

La próxima sección describe las principales características de la arquitectura GPGPU. La sección 3 presenta los algoritmos RDA y CSA y por último, la sección 4 presenta el estado del trabajo actual, trabajo futuro y conclusiones.

## 2 Arquitectura GPGPU

Los procesadores gráficos GPU nacen para aplicaciones específicas: aplicaciones gráficas, tridimiensionales o videojuegos, pero su alta potencia de cómputo, bajo costo y reducido consumo han convertido estas placas en arquitecturas utilizadas para cómputo de alto rendimiento de uso general [7] [9].

Se utilizan como co-procesador para resolver tareas altamente paralelizables, mientras que el código menos paralelizable puede seguir ejecutándose en la CPU [9].

Una GPU está compuesta por un gran número de núcleos (*cores*) de procesamiento. El procesamiento en una GPU se basa en la definición de funciones llamadas *kernels* las cuales se ejecutan en paralelo (y concurrentemente) en los *cores* disponibles. Dichos *cores* se agrupan en multiprocesadores, existiendo diversos multiprocesadores en cada GPU.

Para la programación de aplicaciones paralelas en GPU existe un modelo de programación ampliamente difundido llamado CUDA (*Compute Unified Device Architecture*) [2]. CUDA propone el modelo de programación SIMD (*Simple Instruction Multiple Data*) y es una extensión del lenguaje C/C++ que agrega funcionalidad para el manejo de *threads* (*kernels*) concurrentes y la jerarquía de memoria de las placas GPU [2] [10] [4].

Una aplicación CUDA está compuesta por código secuencial y código paralelo. El código secuencial se ejecuta en la CPU mientras que el código paralelo (*kernels*) se ejecuta en la placa GPU. La CPU comanda la ejecución de los *kernels* en la GPU.

Por cada *kernel* definido en una aplicación CUDA se crean múltiples *threads* (pueden ser miles) que se ejecutan en paralelo. Los *threads* se organizan en bloques (de 1, 2 o 3 dimensiones) y a su vez, los bloques se organizan en grids (de 1 o 2 dimensiones, y 3 dimensiones para las arquitecturas más actuales) (figura 2). Es el programador el que define la configuración de los *kernels* en bloques y grid. Esta configuración puede ser distinta para cada *kernel*.

En las arquitecturas actuales (Fermi, Tesla, Quadro [7]) cada bloque puede tener hasta 1024 *threads* y cada grid puede tener hasta un máximo de 65535 bloques por dimensión (1, 2 o 3 dimensiones). Esta gran cantidad de *threads* que se pueden crear por cada *kernel* constituyen una forma distinta de diseñar los algoritmos: el número de *threads* paralelos es "infinito", siendo esto una gran ventaja respecto a otras arquitecturas paralelas, donde la cantidad de procesos paralelos está limitado, por ejemplo, por la cantidad de nodos y procesadores presentes en un *cluster* de computadoras.

Todos los *threads* creados en un *kernel* ejecutan la misma instrucción sobre distintos datos. La potencia de las GPU reside en este altísimo poder de paralelismo y la alta capacidad de cómputo de cada núcleo de procesamiento. Las GPUs aprovechan el paralelismo de datos presente en las aplicaciones.

Los *threads* de cada bloque se ejecutan en *warps* de 32 *threads* (en las arquitecturas actuales, probablemente en el futuro este número aumente). Los *threads* de un *warp* ejecutan todos la misma instrucción sobre distintos datos. Cuando

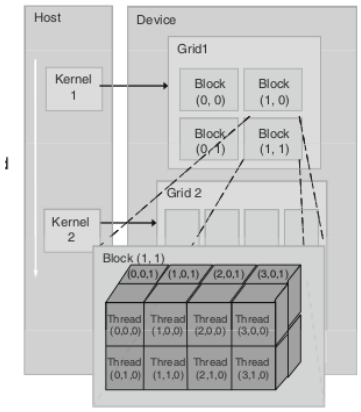


**Fig. 2.** Organización de *threads* en bloques y en grids [4].

hay bifurcaciones y distintos *threads* ejecutan distintas instrucciones, un subgrupo de *threads* espera a la ejecución de otro subgrupo de *threads*, dependiendo de las instrucciones que ejecuten.

Por otro lado, las placas GPU tienen distintos tipos de memoria definiendo una clara jeraquía. Cada tipo de memoria tiene sus propias características: tipo y costo de acceso, alcance y ciclo de vida de los objetos definidos en ella, costo de accesos y mecanismos de optimización. CUDA provee mecanismos para operar con esta jerarquía de memorias presente en los dispositivos GPU [9]. Conociendo las características de la jerarquía de memoria instalada en el dispositivo GPU, el programador puede tomar decisiones respecto al almacenamiento y acceso a los datos a fin de mejorar la eficiencia de la aplicación en la GPU, como así también, evitar posibles penalizaciones por la latencia que significa el acceso a cada memoria.

La jerarquía de memoria y tipos de accesos en un sistema GPU están esquematizados en la figura 3 ([9]).

Existe una memoria global a la cual acceden todos los *threads* del grid, y que es el medio de comunición entre CPU y GPU. El acceso a dicha memoria es muy costoso en términos de tiempo de acceso. Además existen las memorias *constante* y de *textura*, las cuales cuentan con una memoria *cache* por lo que su tiempo de acceso es rápido. Estas memorias son de escritura para la CPU y de lectura para los *threads*. A nivel de bloque existe una memoria compartida (*shared* en la figura) que es de muy rápido acceso pero de dimensiones mucho más pequeñas que las memorias anteriormente mencionadas. Por último, cada *thread* cuenta con registros cuyo acceso es el más veloz de todos, pero como se puede preveer, la capacidad y cantidad de los registros es limitada.

La jerarquía de memorias presente en las placas GPU son un desafío para el desarrollador, ya que el rendimiento de las aplicaciones se puede ver muy

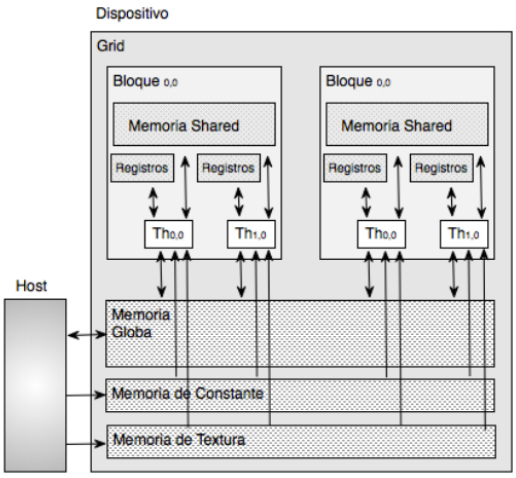


**Fig. 3.** Arquitectura GPU. Multiprocesadores y jerarquía de memorias y accesos. En la figura, host designa a la CPU y Dispositivo a la GPU [9].

afectado por la utilización de dichas memorias. En general, se debe evitar el acceso a memoria global mientras que se debe maximizar la correcta utilización de memoria compartida y registros internos de cada *thread*.

Por otro lado, para muchas de las librerías existentes para C, existe su versión para CUDA optimizadas para su ejecución en los dispositivos gráficos. Estas librerías operan de forma óptima y de forma transparente al usuario, definen la mejor configuración de *threads* en bloques y del grid. Además optimizan la utilización de la jerarquía de memorias instalada en dichos dispositivos. Ejemplos de estas librerías son cuBLAS (*CUDA Basic Linear Algebra Subprograms*), cuFFT (*CUDA Fast Fourier Transform*), MAGMA (similar a LAPACK para GPU), etc. Todas estas librerías son una ventaja para el programador, ya que de forma transparente al usuario, se utiliza de forma óptima los recursos de la placa GPU.

Debido a las características propias de las placas GPU, las aplicaciones deben tener ciertas características que las harían convenientes para su desarrollo en GPU. Estas características son: alto nivel de paralelismo, alto requerimiento de cómputo, gran volumen de datos, que los cálculos realizados sobre los datos no tengan dependencias entre sí, datos que se puedan organizar en vectores o matrices (para satisfacer la configuración física de los *threads* en arreglos de 1, 2 o 3 dimensiones), poca comunicación CPU-GPU, pocas secciones críticas.

La próxima sección presenta las principales características de los algoritmos RDA y CSA. Al final de dicha sección se muestra cómo satisfacen dichos algoritmos las características anteriormente mencionadas, haciendo del procesamiento de señales SAR en arquitecturas GPU un campo muy prometedor desde el punto de vista del HPC.

## 3 Algoritmos RDA y CSA en GPGPU

Actualmente existen distintos algoritmos para el procesamiennto de señales SAR. Los algorimos más difundidos en este área son *Range Doppler Algorithm* (RDA), *Chirp Scaling Algorithm* (CSA), *Omega-K Algorithm*($\omega - K$), *Back Projection Algorithm*, etc.

En este trabajo se toman dos de los algoritmos más utlizados para la obtención de imágenes utilizando radares SAR: RDA y CSA. Dichos algoritmos son el paso inicial para continuar con el desarrollo de otros métodos y etapas en todo el desarrollo de procesamiento de señales SAR. La figura 4 muestra las principales operaciones realizadas en cada uno de estos algoritmos.

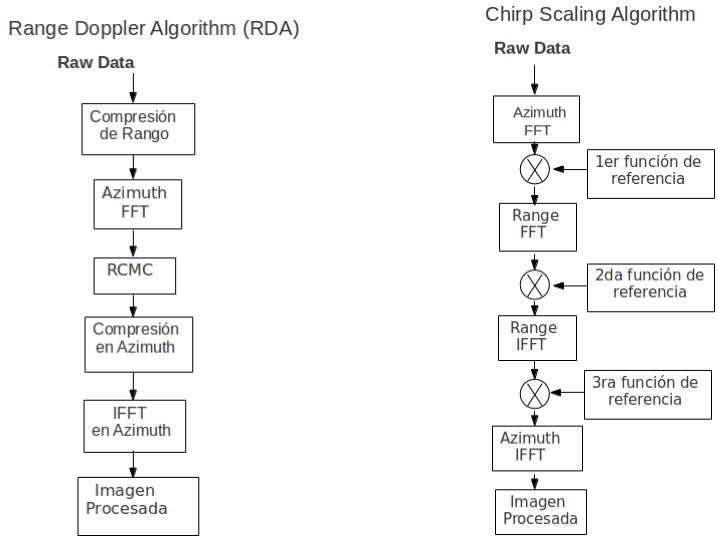


**Fig. 4.** Principales pasos de *Range Doppler Algorithm* y *Chirp Scaling Algorithm*.

El algoritmo RDA fue uno de los primeros algoritmos desarrollados para procesamiento SAR de uso civil. Hoy en día, continúa siendo uno de los algoritmos más utlizados, gracias a su favorable compromiso entre simplicidad, eficiencia y precisión [3].

Los pasos básicos del algoritmo RDA son: compresión de rango (implementada a través de FFT, filtros e IFFT), FFT en acimut (para cambio de dominio), RCMC (*Range Cell Migration Correction*: corrección de migración de celdas en rango), compresión en acimut y una última IFFT para llevar los datos al dominio del tiempo.

A su vez, el algoritmo CSA se basa en la aplicación de 4 FFTs y multiplicaciones en la señal de entrada. Para ambos algoritmos, sus pasos permiten variantes en su forma de implementación.

Como se puede observar, los algoritmo RDA y CSA se basan en múltiples transformadas de Fourier y antitransformadas, multiplicaciones y aplicación de filtros. La complejidad computacional de dichos algoritmos está dominada por las transformadas de Fourier; para una matriz de $n \times m$ la complejidad computacional es de $n * \log(n) + m * \log(m)$.

Considerando la complejidad computacional de los algoritmos y teniendo en cuenta el gran volumen de datos involucrados en el procesamiento de señales SAR, se observa la necesidad de una implementación eficiente, que genere imágenes de alta resolución y exactitud a la vez que el tiempo de obtención de dichas imágenes se encuentre dentro de los límites del tiempo real.

Los algoritmos se desarrollan en forma modular, de modo que permitan el diseño, desarrollo, implementación, prueba y puesta a punto de forma independiente e incremental. Los datos crudos SAR serán almacenados en forma vectorial, y estos datos se copiarán a memoria global del dispositivo (GPU) al comienzo de la aplicación.

Para las operaciones que involucran transformadas de Fourier, se proponen dos alternativas. La primera implica buscar la rutina óptima dentro de la librería cuFFT y la segunda se considerará realizar una implementación propia que permita tener mayor control sobre la descomposición a la que se somete a los datos. Las aplicaciones de filtros y multiplicaciones serán desarrolladas en paralelo, haciendo que cada *thread* realice el cálculo sobre el dato (o datos) que le correspondan. Para dichos cálculos, se combinará la forma de dividir los *threads* en bloques para que a su vez, se pueda utilizar la memoria compartida y local de cada *thread* de forma óptima.

Ambos algoritmos se desarrollarán teniendo en cuenta los distintos formatos de datos: números complejos y representaciones de números en simple y doble precisión. Además, las arquitecturas GPU consiguen su máximo rendimiento cuanto mayor sea la carga de trabajo que tengan. Esto se considera en los actuales diseños de los algoritmos.

Por otro lado, se sabe que los *kernels* que tienen múltiples bifurcaciones en su código no consiguen el rendimiento óptimo en GPU: esto se debe a la ejecución en *warps* que se ha mencionado anteriormente. Este es otro aspecto que se está considerando durante el diseño y desarrollo del código.

Una vez realizado el cómputo sobre los datos, se copia el vector resultante desde la GPU a la memoria de la CPU.

En este trabajo, se proponen algoritmos modulares, donde cada módulo tiene una interfaz definida facilitando el intercambio y prueba de los mismos.

Una vez copiados los datos a la GPU, la CPU comandará la ejecución de distintos *kernels* que se ejecutarán en la GPU y que resolverán las tareas paralelizables: cada FFT e IFFT, cada filtrado, multiplicación, se realizan en paralelo, haciendo que cada *thread* opere sobre un dato que compone la señal.

Una vez presentadas las características de los algoritmos como así también de la arquitectura GPU, se puede observar que el procesamiento de señales SAR verifica:

1. Alta carga computacional para cada *thread* (que compense el alto costo de la transferencia de datos CPU-GPU): los algoritmos se basan, fundamentalmente, en FFTs e IFFTs.
2. Poca dependencia de datos (que los *threads* sólo necesiten datos de su memoria local o compartida, evitando accesos costosos a memoria global). En los algoritmos RDA y SCA no hay dependencia de datos.
3. Mínima transferencia de datos entre CPU y GPU: se realiza sólo al comienzo la copia de datos crudos SAR a la memoria global y al finalizar el procesamiento se copia la imagen ya procesada en sentido inverso.
4. Pocas secciones críticas (escrituras y accesos a las mismas posiciones de memoria): en los algoritmos utilizados cada pixel se calcula sin depender de los demás.
5. Necesidad de que los datos se adapten a las estructuras de datos que manejan las GPUs: vectores o matrices. En los algoritmos utilizados el procesamiento se basa en vectores.

## 4 Trabajo actual, trabajo futuro y conclusiones

En este trabajo se propone la implementación paralela de los algoritmos RDA y CSA para procesamiento de señales SAR específicamente para placas gráficas GPU.

En dichos algoritmos se encuentran presentes las características que hacen que un algoritmo sea conveniente para implementar en GPU y obtener el máximo rendimiento en las placas gráficas. La implementación de dichos algoritmos son el primer paso para la implementación de distintas tareas que involucra el procesamiento de señales SAR.

Actualmente se están implementando dichos algoritmos. Se está trabajando con datos sintéticos y se espera trabajar con datos reales provenientes de grupos de trabajo de áreas similares con las que se está en comunicación.

Se cuenta con 2 GPUs instaladas y en funcionamiento. Una es un modelo GeForece GTX 550 Ti, con 192 nucleos (4 multiprocesadores de 48 núcleos cada uno). Otra GPU disponible es modelo GeForce GTX 570 que cuenta con 480 *cores*. Se dispone de CUDA y CUDA Toolkit funcionando en LINUX y en Windows. Se dispone también de la herramienta *Visual Profiler* que permite obtener datos muy interesantes, como por ejemplo, el rendimiento, requerimientos, etc de cada uno de los *kernels* que componen la aplicación.

Proximamente se estima finalizar con la implementación de dichos algoritmos. Se evaluará su eficiencia y exactitud. Además, las imágenes obtenidas con dichos algoritmos también serán evaluadas, abriendo esto otra tarea en este campo.

## References

1. Wang Bingnan, Zhang Fan, Xiang Maosheng: SAR Raw Signal Simulation based on GPU Parallel Computation. Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009. Volumen 4, páginas IV-617 - VI-620. ISBN: 978-1-4244-3394-0.

2. NVIDIA CUDA Web site: http://www.nvidia.com/object/cuda_home_new.html Accedido en julio de 2012.
3. Ian G. Cumming, Frank H. Wong: Digital Processing of Synthetic Aperture Radar Data. Artech House. ISBN-10: 1-58053-058-3
4. David B. Kirk, Wen-mei W. Hwu: Programming Massively Parallel Processors: a Hands-on Approach. ISBN-10: 0123814723. Febrero de 2010.
5. Jeffrey Krolik: Radar and Sonar Signal Processing: Similarities and Differences. Duke University. Department of Electrical and Computer Engineering. Durham. 2005.
6. Antonio Martinez, Francisco Fraile, Jordi Mallorquí, Leonardo Nogueira, Jordi Gabaldá, Antoni Broquetas, Antonio Gonzalez: PARSAR: Parallelisation of a Chirp Scaling Algorithm SAR Processor. Data Systems in Aerospace - DASIA 97, Proceedings of the meeting held 26-29 May, 1997 in Sevilla, Spain. Edited by T.-D. Guyenne. ESA SP-409. Paris: European Space Agency, 1997, páginas 1346 - 1350.
7. NVIDIA Web site: http://la.nvidia.com/page/home.html. Accedido en julio de 2012.
8. NVIDIA CUDA Toolkit 4.2. CUFFT Library. Programming Guide. Marzo 2012.
9. María Fabiana Piccoli: Computación de Alto Desempeño en GPU. Universidad Nacional de San Luis, San Luis, Argentina. 2011.
10. Jason Sanders, Edward Kandrot: CUDA by Example: An Introduction to General-Purpose GPU Programming. ISBN-10: 0131387685. Julio de 2010.
11. Mehrdad Soumekh: Synthetic Aperture Radar Signal Processing. Wiley-Interscience. John Wiley & Sons, Inc. Nueva York. ISBN 978-0-471-29706-2.
12. TerraSAR-X: Imágenes por satélite de radar TerraSAR-X. Web site: http://www.astrium-geo.com/es/463-imagenes-por-satelite-de-radar-terrasar-x. Accedido en julio de 2012.