# GNSS-R Cross Correlation: Diseño, implementación y optimización de la correlación cruzada

Carlos Calvin Palomares

**Resumen—** Las señales de los Sistemas de Navegación Global por Satélite pueden ser utilizadas para determinar la altura del mar. Para llevar a cabo estos experimentos es necesario una implementación eficiente de la correlación cruzada, que permita el procesamiento en tiempo real de las señales. Este artículo presenta una implementación eficiente de la correlación cruzada que se ha desarrollado para CPU y GPU utilizando las transformadas rápidas de fourier, cuyo tiempo de cómputo hace veraz el proceso en tiempo real.

**Palabras clave**—GNSS, Correlacióncruzada, FFT, GPU, Tiempo-real.

**Abstract**—Global Navigation Systems Satellite' s signals can be used to determine the height of the sea. In order to carry out these experiments is required an efficient implementation of the crosscorrelation, which allows realtime processing of signals. This paper presents an efficient implementation of the crosscorrelation which has been developed for CPU and GPU using the fast Fourier transforms, whose computational time makes the process possible in real time.

**Index Terms**—GNSS, Crosscorrelation, FFT, GPU, Real-time.

## 1 Introducción

Las señales de los Sistemas de Navegación Global por Satélite (GNSS Global Navigation Satellite Systems) se usan en aplicaciones de teledetección oceanográfica, como la altimetría marina, que mide la altura del nivel del mar.

Las medidas altimétricas se calculan con la diferencia de tiempo entre la recepción de la señal que llega directamente del satélite y la recepción de la misma señal rebotada en la superficie del mar. La figura 1 muestra como se determina la altura (h) a partir de las señales recibidas.

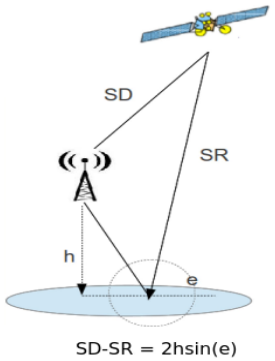


Fig.1: La altura del nivel del mar (h), se calcula usando la señal directa (SD), la señal reflejada (SR) y la elevación del satélite respecto al punto de reflexión (e).

1* *E-mail de contacte: carlos.calvin@e-campus.uab.cat*
2* *Menció realitzada: Enginyeria de Computadors.*
3* *Treball tutoritzat per: Juan Carlos Moure (CAOS)*
4* *Curs 2013/14*

### Experimento GNSS-Ri

En el Instituto de Estudios Espaciales de Cataluña han desarrollado un método para mejorar la precisión en la medición altimétrica, denominado GNSSRi[1], procesando las señales GPS de forma distinta al enfoque habitual. Para evaluar este nuevo método, se realizará el experimento de equipar una avioneta con 8 antenas en la parte superior, que recibirán la señal directamente y 8 antenas más en la parte inferior, que recibirán la señal reflejada en la superficie marina. Posteriormente, se deteterminará el retraso entre la señal directa y reflejada mediante el estudio de la correlación cruzada de ambas señales. En la figura 2 [2], se presenta el esquema del experimento.

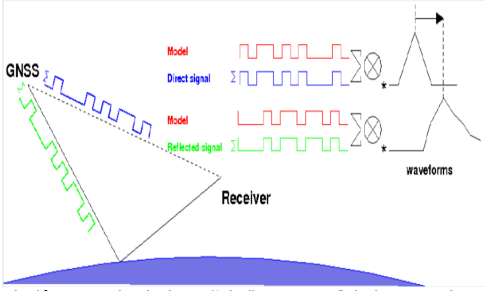


Fig2:Correlación cruzada de la señal directa y reflejada para obtener los datos (waveforms) que nos permiten determinar el retraso entre ambas señales.

Durante dos horas de trayecto se recogerán una gran cantidad de datos: cada segundo se generan 160 millones de muestras. Por tanto es necesario computar la correlación cruzada de forma rápida y eficiente.

**Objetivo del trabajo**

En este trabajo se ha llevado a cabo el diseño e implementación del proceso de correlación entre dos señales, con el objetivo de realizarlo en tiempo real.

En el cómputo de la correlación la velocidad del algoritmo de las transformadas rápidas de fourier [3] (FFT), es determinante para realizar el proceso. Por tanto, la elección de las librerías que realizan este algoritmo será de especial importancia y se menciona en el documento. La metodología se explica en el punto 2. La sección 3 describe la implementación del correlador, incluyendo el diseño del diagrama de flujo, la implementación en CPU single-thread y la implementación en CPU multi-thread. En el apartado 4 se mostrarán las medidas tomadas y los resultados de la ejecución en CPU. En el apartado 5 se explicará la adaptación del diseño para la arquitectura GPU. Los resultados en GPU se mostrarán en el apartado 6 y las conclusiones se presentan en la sección 7.

## 2 METODOLOGÍA

La metodología está enfocada en diseñar e implementar el proceso de correlación de forma iterativa, asegurando la calidad de los resultados mientras se realizan los cambios adecuados en el rendimiento del proceso. Esta metodología ha sido estructurada en 2 versiones:

### 2.1 Versión secuencial

Se han diseñado las funciones involucradas en la correlación, dibujando el diagrama de flujo de datos para la visualización del procesamiento.

La versión secuencial del diseño realizado se ha implementado siguiendo los siguientes pasos:

- Elegir el lenguaje de programación.
- Buscar y analizar las librerías utilizadas para las transformaciones.
- Diseñar los casos de prueba para asegurar la integridad de los resultados.
- Implementar el proceso utilizando el lenguaje y las librerías escogidas.
- Validar la funcionalidad de la implementación, usando los casos de prueba.

Se ha medido y analizado el rendimiento de la implementación, obteniendo la información necesaria para optimizar el proceso.

### 2.2 Versión multithread

Para la implementación de la versión multithread, se ha definido el modelo de programación a utilizar y se ha paralelizado el proceso escogiendo el método de descomposición y definiendo la granularidad. Para la ejecución se ha definido la planificación, se ha analizado el rendimiento y optimizado la aplicación.

## 3 DISEÑO E IMPLEMENTACIÓN EN CPU

En este apartado se explica el diseño e implementación del proceso de correlación desarrollado para ejecutarse en la CPU. El desarrollo sigue la metodología explicada en el apartado anterior.

### 3.1 Diseño del proceso de correlación

La correlación cruzada [4] que se explica en este apartado realiza distintos procesos para obtener el retraso entre la señal directa y reflejada. En el esquema de la figura 3, se muestra el diagrama de flujo con los procesos, que se explican a continuación.

#### 3.1.1 Transformación 1: FFT Forward

Disponemos de un input de dos señales, directa y reflejada, que están contenidas en dos matrices de 80.000 columnas x 1.024 filas (realmente son 1.000 valores pero se añaden 24 ceros para que los valores puedan correlacionarse mediante la transformada ápida de fourier). La primera transformación realiza una transformada rápida de fourier a cada fila de las dos matrices, y guarda su resultado en otras dos matrices de 80.000x1.024.

#### 3.1.2 Transformación 2: Multiplicación

La segunda transformación se aplica sobre las matrices obtenidas en el proceso anterior. Se realiza una operación elemento a elemento, multiplicando cada elemento de la matriz de la señal directa por el conjugado complejo de los elementos de la matriz de la señal reflejada. El resultado se guarda en una matriz de 80.000x1.024, que equivale al espectro de frecuencias de las señales [5].

#### 3.1.3 Reducción: Promedio

La reducción se aplica sobre la matriz obtenida en la transformación 2. Se realiza un promedio de cada 80 filas en una sola fila, de manera que obtenemos una matriz de 1.000x1.024. Esta matriz equivale también al espectro de frecuencias, pero gracias al promedio se reduce el ruido que se pudiera haber generado en el paso anterior.

#### 3.1.4 Transformación 3: FFT BACKWARD

Es el proceso que dará los resultados finales. Se realiza una operación en cada fila de la matriz de 1.000x1.024 obtenida en la reducción, se aplica la transformada inversa de fourier a cada fila y se guarda el resultado en una matriz de 1.000x1.024. Esta matriz equivale a las funciones de correlación de las señales, donde el valor más grande corresponde al pico de correlación. Este punto nos indica el retraso que existe entre ambas señales [6].

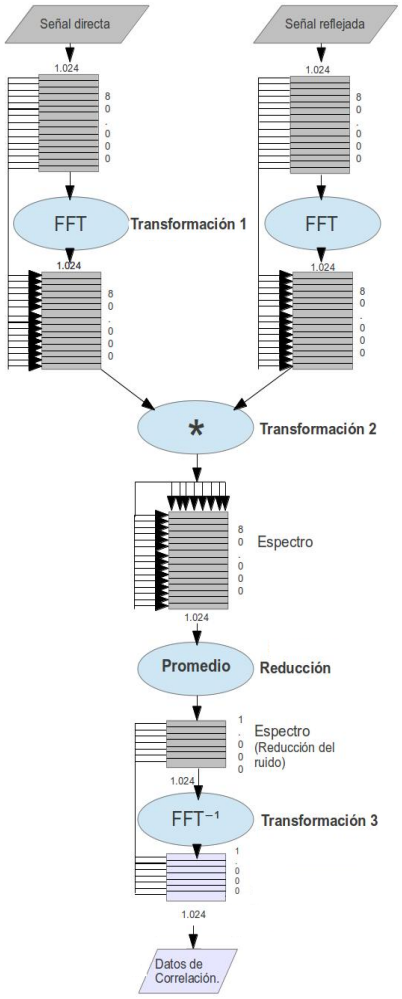


Fig.3: Diagrama de flujo del algoritmo de la correlación cruzada.

## 3.2 Implementación de la versión secuencial

### 3.2.1 Lenguaje de programación y librerías utilizadas

El lenguaje de programación elegido es Fortran. La razón de esta elección es que el proyecto se realiza en colaboración con el IEEC, donde se está acostumbrado al uso del mismo. Además, siendo un lenguaje especialmente adaptado al cálculo numérico y a la computación científica, tiene una comunidad y una gama de librerías suficientemente extensa para la realización de este proyecto. Las librerías escogidas, para la implementación en CPU, son las FFTW [7] y las MKL [8]. Se han utilizado para las transformaciones de Fourier y se ha analizando su rendimiento en el proceso de correlación.

### 3.2.2 Definición de los casos de prueba

Se definen los casos de prueba para realizar los tests de regresión. Mientras se ha ido implementando el proceso se van realizando optimizaciones: los casos de prueba nos permiten verificar el funcionamiento y la integridad de los resultados cuando se realizan cambios.

Los casos de prueba consisten en el resultado esperado por la función de correlación. Se han obtenido generando cuatro señales sintéticas y procesándolas mediante una función de correlación cruzada disponible en el software matemático Matlab. Se utilizan estos resultados para verificar las salidas del proceso implementado.

### 3.2.3 Implementación del proceso

La primera versión funcional se ha implementado realizando los siguientes pasos:

- Uso de las librerías FFTW para realizar las transformadas de fourier. Las librerías adaptan la función de transformación al input especificadomediante la creación de un plan [9], guardando la información de la entrada, la salida y de las opciones de optimización para realizar las operaciones.
- Transformación directa aplicada a cada fila utilizando la función FFT, utilizando los planes creados mediante las librerías FFTW y transformando las matrices.
- Multiplicación elemento a elemento, realizando la multiplicación compleja entre un elemento y el conjugado del otro.
- Promedio de 80 filas a 1 para la reducción del ruido del espectro de las señales.
- Transformación inversa aplicada a cada fila utilizando la función FFT, para transformar el espectro en los datos de correlación.

### 3.2.4 Rendimiento y optimizaciones

Para mejorar la ejecución del proceso, se fusionan las distintas operaciones de la implementación. En vez de guardar los resultados de cada transformación en una matriz temporal, se realizan las transformaciones en bloques de 80 filas y se obtiene en cada iteración una fila de la matriz de resultado, como se muestra en la figura 4.

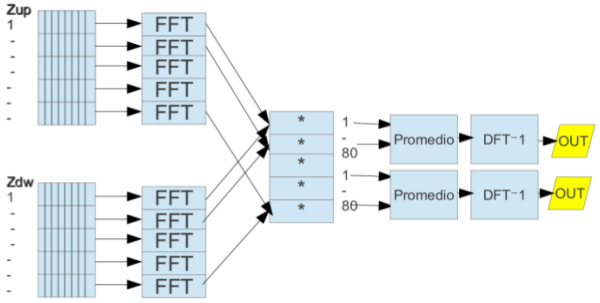


Fig.4: Transformaciones realizadas en bloques de 80 filas.

Con esta optimización se reduce el uso de la memoria RAM y se disminuye el número de instrucciones. Se dan más detalles acerca de las mejoras en el apartado de resultados.

### 3.3 Implementación de la versión multithread

#### 3.3.1 Modelo de programación

Para la implementación de la versión multithread se ha utilizado el modelo de paralelismo de memoria compartida, usando threads para la ejecución concurrente.

#### 3.3.2 Descomposición del proceso

Los diferentes métodos para descomponer un proceso son el paralelismo funcional (descomposición funcional) y el paralelismo de datos (descomposición de dominio). En el paralelismo funcional dividimos el proceso en distintas tareas que son distribuidas a múltiples procesadores. En el paralelismo de datos se distribuyen los datos y se replican las tareas. La descomposición puede generar un gran número de tareas pequeñas, granularidad fina, o bien un número reducido de tareas de mayor tamaño, granularidad gruesa.

La paralelización del proceso de correlación se ha definido de 2 maneras.

Una forma es descomponiendo las transformaciones de fourier utilizando las rutinas multithreading de las librerías FFTW.

• Se realizan distintas tareas de forma concurrente en las transformaciones.

• Hay un gran número de tareas que se realizan a lo largo del proceso de correlación o granularidad fina.

Esta opción no es la más adecuada, ya que el tamaño de las filas que se transforma no es lo suficientemente grande y el tiempo de gestionar los threads es mayor que el ganado en la ejecución concurrente. "Por lo general, el problema tendrá que procesar al menos unos pocos miles de puntos de datos antes de que los threads resulten beneficiosos" [10].

La otra forma es descomponiendotodo el proceso de correlación, desde la transformación FFT hasta la obtención de los datos de correlación.

• Se realizan las mismas tareas con distintos datos, dividiendo las filas de la matriz entre los threads disponibles.

• Hay un número reducido de tareas, pero de mayor tamaño o granularidad gruesa.

Finalmente se ha implementado mediante descomposición de dominio y granularidad gruesa, como se muestra en la figura 5.

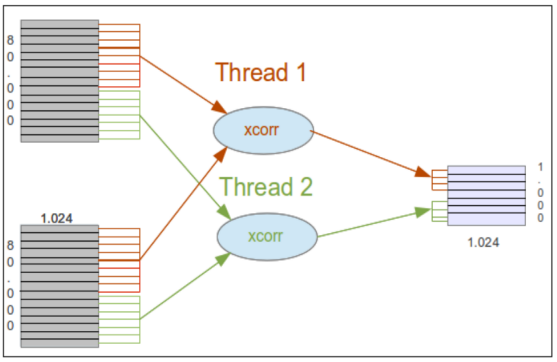


Fig.5: Paralelismo de datos en el proceso de correlación cruzada: se distribuyen los datos y replican las tareas.

#### 3.3.3 Comunicación y planificación

En la planificación de la ejecución paralela los datos se asignan a los procesadores tratando de maximizar su utilización. En el proceso no es necesario el balanceo de carga entre threads ya que el tiempo de correlación es estable con distintos datos. Por tanto la planificación estática, donde los datos se asignan antes de la ejecución del programa, es más eficiente.

#### 3.3.4 Implementación del proceso

Se implementa la versión multithread utilizando las extensiones de la librería OpenMP [11], ya que permite la comunicación con memoria compartida, soporta paralelismo de datos, puede planificar de forma estática y hay variada información de su implementación con el lenguaje Fortran.

#### 3.3.5 Rendimiento y optimizaciones.

Tras la implementación y el análisis de la versión multithread, se ha optimizado utilizando las librerías matemáticas de Intel MKL (Math Kernel Library). Estas librerías son más eficientes y permiten la creación de los planes de forma independiente, consiguiendo una mayor escalabilidad y un menor número de instrucciones respecto a las librerías FFTW.

## 4 Resultados de la ejecución en CPU

Se han realizado las ejecuciones y análisis del proceso de correlación en el PC cuyas características se muestran en la figura 6.

**Sistema operativo:**

Ubuntu versión 12.04 64 bits.
Núcleo Linux 3.5.0-36-generic

**Hardware:**

- Procesador : Intel(R) Core(TM) i5-3210M:
  clock: 2.50GHz.
  CPUs : 4.
- Memoria cache:
  L1 32 KB.
  L2 256 KB.
  L3 3 MB.
- Memoria RAM: 3,6 GB.

Fig.6: Sistema operativo y características de la arquitectura del PC de integración.

Para el análisis de la distintas versiones explicadas, obtenemos los ciclos transcurridos en ejecutar las instrucciones, las instrucciones ejecutadas y el tiempo transcurrido en realizar la correlación.

En la figura 7 mostramos el tiempo y speedup de la versión secuencial, la versión multithread con FFTW y la versión multithread con MKL. En la figura 8 podemos observar los ciclos transcurridos y las instruciones ejecutadas en las tres versiones.

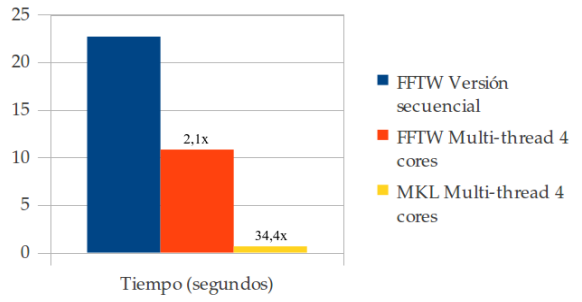


Fig.7: Tiempo transcurrido durante la ejecución del proceso de correlación.

| Versión | Ciclos | Instrucciones | IPC |
|---|---|---|---|
| FFTW Versión secuencial | 75G | 214G | 2,84 |
| FFTW Multi-thread 4 threads | 115G | 176G | 1,53 |
| MKL Multi-thread 4 threads | 14G | 18G | 1,26 |

Fig.8: Ciclos transcurridos, instrucciones e IPC de las tres versiones implementadas.

En la figura 7 podemos observar que la versión FFTW no escala linealmente con el aumento de threads, para 4 threads se obtiene un speedup de 2,1x respecto a la versión secuencial. La figura 8 muestra como el IPC entre la versión secuencial y la versión multithread FFTW disminuye, lo que explicaría la falta de escalabilidad entre las dos versiones.

En la versión MKL se ha conseguido un aumento muy significativo del speedup respecto a la primera versión, sin embargo esto se debe al uso de librerías mejor implementadas que reducen el número de instrucciones de las transformaciones, por tanto se debería de haber realizado un mejor estudio e implementación de la versión secuencial antes de paralelizar la aplicación.

# 5 ADAPTACIÓN DE LA CORRELACIÓN EN GPU

## 5.1 Metodología

En el siguiente apartado, se explica la metodología para adaptar la implementación del proceso de correlación a la arquitectura de las GPU. Se ha adaptado el proceso siguiendo las siguientes 6 fases:

- Elegir el lenguaje de programación.
- Buscar y analizar las librerías utilizadas para las transformaciones.
- Implementar el proceso utilizando el lenguaje y librerías escogidas.
- Ajustar el tamaño del problema para adaptar la ejecución a la placa utilizada.
- Validar la funcionalidad de la implementación usando los casos de prueba.
- Analizar el rendimiento y optimizar la implementación.

## 5.2 Implementación del proceso en GPU

Se ha diseñado y desarrollado el módulo para adaptarlo a la arquitectura de las GPU, tal y como se explica a continuación.

### 5.2.1 Lenguaje de programación y librerías Utilizadas

Se ha implementado la versión en CUDA [12] con el lenguaje C, ya que se tiene una mayor familiaridad con el mismo y hay un mayor soporte y documentación sobre el uso de CUDA en C que en el lenguaje Fortran.

Las librerías utilizadas son las CUFFT [13], que permiten realizar las transformaciones rápidas de Fourier en las GPU.

### 5.2.2 Implementación del proceso

El proceso se ha implementado realizando los siguientes puntos:

- Uso de las librerías CUFFT para desarrollar las transformaciones de Fourier. Las librerías adaptan la función de transformación al tamaño y al número de transformadas especificadas mediante la creación de un plan. Por tanto la llamada calcula el número de bloques y threads que se han de lanzar en la GPU en función del número de transformaciones de los datos de entrada.

- Se realiza la multiplicación compleja entre un elemento y el conjugado del otro y el promedio de 80 filas a 1 para la reducción del ruido del espectro de las señales. Se fusionan ambos procedimientos para aumentar el rendimiento, implementándolos en un kernel a medida (función que se ejecuta desde la GPU).

En la figura 9 podemos observar el esquema de los procesos que se ejecutan en la GPU. A la izquierda de la figura observamos la memoria RAM, disponible para la CPU. Los datos almacenados en la memoria RAM se transfieren a la GPU a través del bus PCIExpress, tal y como muestran los cuadros rojos de la figura. Una vez en la GPU, se realizan las transformaciones con las librerías CUFFT, las multiplicaciones y el promedio con el kernel desarrollado a medida, como se observa a la derecha de la figura.

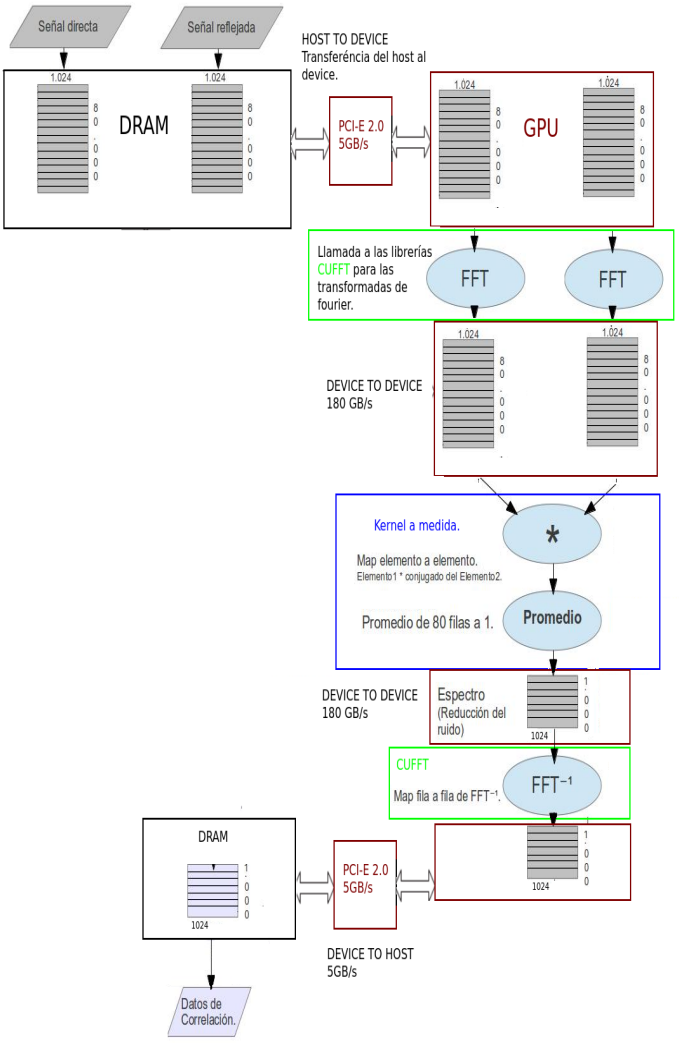


Fig.9: Diagrama de los procesos de correlación cruzada que se ejecutan en la GPU.

### 5.2.3 Ajustes, rendimiento y optimizaciones

**División de los datos que se transfieren al GPU.**

Para la ejecución y las pruebas de la aplicación implementada, se tendrán que realizar ciertos ajustes según las especificaciones de la placa escogida. La primera especificación a tener en cuenta es el tamaño de la memoria disponible en la GPU. Para los ajustes, se realiza la ejecución de la simulación en la placa cuyas características se muestran en la figura 10.

En la simulación se necesitan 2,36 GB para realizar los cálculos, sin embargo la placa utilizada en las ejecuciones solo dispone de 2GB, por tanto se realizarán las operaciones en dos partes.

NVIDIA GeForce GTX 680

| | |
|---|---|
| **Especificaciones de la GPU** | |
| Frecuencia de reloj | 1006 |
| **Especificaciones de la memoria** | |
| Cantidad de memoria | 2048MB |

Fig.10: Especificaciones de la GPU utilizada en las ejecuciones.

**Solapamiento de cómputo y transferencia de datos.**

Un factor a tener en cuenta es que la cantidad de datos a copiar es bastante grande, realizándose además cálculos independientes sobre unas estructuras de datos y otras ( las transformaciones de la señal1 y la señal2 son independientes). Por tanto una strategia para mejorar el rendimiento sería la de solapar la transferencia con el cómputo, tal y como podemos observar en la figura 11.

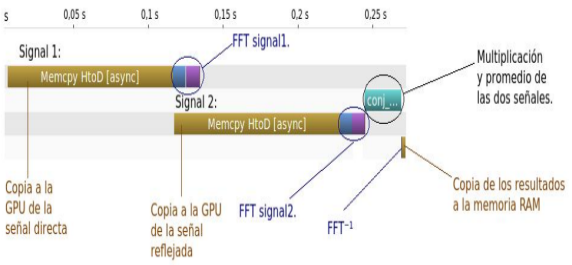


Fig.11: Timeline de los distintos procesos del algoritmo de correlación cruzada en GPU.

## 6 RESULTADOS DE LA EJECUCIÓN EN GPU

Se han realizado las ejecuciones y análisis del proceso de correlación en la GPU cuyas características se muestran en la figura 10.

Para el análisis de la implementación adaptada en GPU, se han ejecutado las distintas versiones obteniendo las métricas necesarias para determinar su rendimiento: la eficiencia de la memoria, la ocupación de la placa, el tiempo transcurrido y el solapamiento entre las transferencias y el cómputo.

En la figura 12 mostramos el tiempo de la versión CUFFT sin solapamiento y la versión con solapamiento. En la figura 13 podemos observar el speed-up obtenido respecto a la versión implementada con las librerías MKL ejecutada en la CPU.

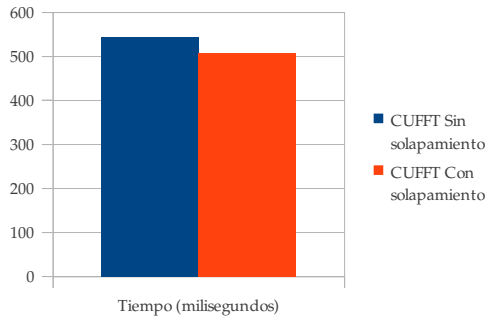


Fig.12: Tiempo transcurrido durante la ejecución del proceso de correlación.

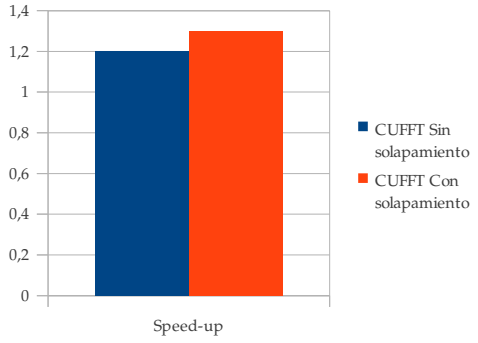


Fig.13: Speed-up obtenido en la GPU respecto a la CPU.

En los resultados obtenidos el tiempo de transferencia de datos ocupa un 77% de la ejecución.

Esto se debe a que estamos limitados por el ancho de banda del bus PCI Express: Tranferimos 2,36 GB a la GPU y el ancho de banda es de 5,5GB/s, por tanto el tiempo mínimo que tardará la transferencia de datos es de 429 ms.

La figura 14 muestra el porcentaje de tiempo que ocupan cada proceso en la ejecución en GPU. Resalta el alto porcentaje ocupado por la transferencia de datos.

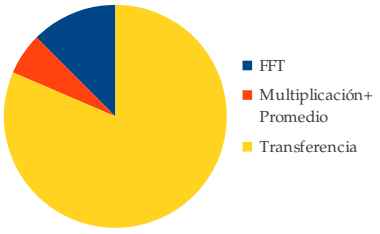


Fig.14: Porcentaje que ocupa cada proceso sobre el total en la ejecución.

## 7 CONCLUSIONES

Por los resultados obtenidos en la ejecución de la última versión adaptada a CUDA, cabe la posibilidad de realizar el procesamiento de datos en tiempo real. El tiempo necesario para la ejecución es de 0,508 segundos, de los cuales solamente 60 milisegundos están dedicados en exclusiva al cómputo. Por tanto se muestran dos posibilidades para ejecutar la aplicación en tiempo real.

Una primera opción es recoger los datos del satélite y transferirlos a la GPU a través de la memoria DRAM, con una latencia mayor de 0,5 segundos para cada segundo de datos, como se muestra en la figura 15.

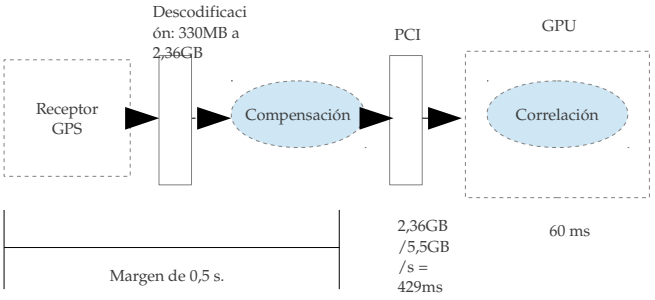


Fig.15: Procesamiento en tiempo real transfiriendo los datos de la DRAM a la GPU.

La segunda opción es recoger los datos del satélite y transferirlos directamente a la GPU, evitando la transferencia de datos. Sin embargo,mediante esta opción habrá que implementar los procesos de compensación de la fase también mediante la arquitectura GPU, como se muestra en la figura 16.

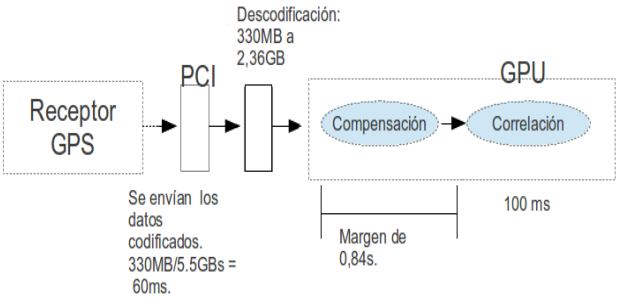


Fig.16: Procesamiento en tiempo real transfiriendo los datos directamente a la GPU.

La implementación del correlador, desarrollada en Fortran y OpenMP, cumple las expectativas del TFG, cuya implementación se añadirá al proyecto realizado en el IEEC.

La implementación desarrollada en C y adaptada a la arquitectura GPU, pese a cumplir la viabilidad para obtener resultados en tiempo real, no cumple las expectativas del trabajo. La latencia producida al transferir los datos por el bus PCIE hace equivalente el tiempo obtenido a la versión desarrollada en Fortran.

Queda pendiente realizar todo el proceso del proyecto en la GPU para evitar la transferencia de datos.

## REFERENCIAS

[1] NoguésCorreig, S. Ribó, J.C. Arco, E. Cardellach, A. Rius. PARIS Interferometric Technique Proof of Concept PITPoC pp 15.
[2] NoguésCorreig, S. Ribó, J.C. Arco, E. Cardellach, A. Rius. PARIS Interferometric Technique Proof of Concept PITPoC pp 4.
[3] Bracewell, R. The Fourier Transforms and its Applications McGrawHill, pp. 47.
[4] Bracewell, R. The Fourier Transforms and its Applications McGrawHill, pp. 47.
[5] Nicolás Herrera Peredo, Análisis Espectral basado en FFT, pp. 63.
[6] Jonathan D. Romney , Theory of Correlation in VLBI , pp. 18.
[7] Matteo Frigo , Steven G. Johnson, manual for FFTW, version 3.3.3, 25 November 2012, pp. 01.
[8] Intel® Math Kernel Library , Reference Manual , Fourier Transform Functions , pp. 2862.
[9] Matteo Frigo , Steven G. Johnson, manual for FFTW, version 3.3.3, 25 November 2012, pp. 22.
[10] Matteo Frigo , Steven G. Johnson, manual for FFTW, version 3.3.3, 25 November 2012, pp. 51.
[11] OpenMP Application Program Interface , Version 4.0 July 2013 , pp. 02.
[12] MK CUDA Programming Morgan Kaufmann, 2013 Elsevier Inc, pp. 13.
[13] CUFFT Library PG05327050_v01 | April 2012, pp. 04

## BIBLIOGRAFÍA

• Manual FFTW, Matteo Frigo , Steven G. Johnson , versión 3.3.3, 25 Noviembre 2012.
• Using MMXTM Instructions to Perform Complex 16Bit FFT , From Intel® Developer Services www.intel.com/IDS , 1996.
• Intel Math Kernel Library Reference Manual, capitulo 12 Fast Fourier Transforms, Julio 2013.
• Foro para desarrolladores de Intel, https://devtalk.nvidia.com/.
• Foro de desarrollo en general, http://stackoverflow.com/.
• CUFFT Library PG05327050_v01 | April 2012 .
• MK CUDA Programming Morgan Kaufmann, 2013 Elsevier Inc.
• MultiGPU Programming Paulius Micikevicius NVIDIA August 30, 2011 .
• CUBLAS LIBRARY DU06702001_v5.5 | July 2013 User Guide .
• NVIDIA Performance Primitives (NPP) Version 5.0 September 7, 2012 .
• Foro para desarrolladores de NVIDIA https://devtalk.nvidia.com/