# Busqueda de RRATS con de-dispersión basada en GPU.

(traducción realizada por Angel Cancio del documento arXiv:1107.2516v1)

## 1. INTRODUCCIÓN

Este trabajo principalmente se ocupa de la de-dispersión. Esto se refiere a una familia de técnicas empleadas para invertir el efecto de refracción dependiente de la frecuencia del medio interestelar (ISM) en las señales de radio que pasan por él. También utilizamos la dispersión interestelar para restringir el espacio de parámetros de una determinada búsqueda.

De los estudios de pulsares , está bien determinado que la propagación a través del ISM cumple la ley dispersión de plasma frío, donde el tiempo de retardo entre dos frecuencias f1 y f2 está dado por la relación cuadrática:

$$\Delta t \simeq 4.15 x\, 10^{6}\, ms \times f1^{-2} - f2^{2} \times DM \quad (1)$$

donde **DM** es la **medida de dispersión** en $pc \cdot cm^{-3}$ , $f1$ y $f2$ están en MHz. Los objetos astrofísicos tienen un valor determinado de DM asociado a ellos, que depende de la cantidad total de electrones libres a lo largo de la línea de visión y, por lo tanto, a la distancia al objeto desde la Tierra.

Eliminar el efecto de dispersión de los datos astrofísicos se hace típicamente de dos formas, dependiendo del tipo de datos y los requisitos del experimento. Para el total de los datos de potencia del banco de filtros, donde el ancho de banda espectral de la observación está típicamente divido en un número de canales de frecuencia estrechas, la de-dispersión consiste en un desplazamiento relativo en el tiempo de todos los canales de frecuencia de acuerdo con la ecuación [1]. Esto se conoce como de-dispersión incoherente ya que se realiza sobre los datos incoherentes. Para los datos en banda base, la de-dispersión se puede hacer por la convolución de los datos observados de tensión con la inversa de la función de transferencia del ISM. Esto se conoce como de-dispersión coherente, que es más precisa en cuanto a la recuperación de la forma intrínseca de la señal de astrofísica, pero mucho más exigente en carga computacional. En este trabajo nos ocupamos de la de-dispersión incoherente, aplicada a los datos de potencia total.

La de-dispersión es una operación costosa, escalada como $O(N^2)$ para los algoritmos de fuerza bruta, con técnicas más optimizadas puede disminuir $O(N \cdot \log(N))$ . Para cualquier búsqueda ciega de transitorios rápidos, la de-dispersión necesita ser realizada sobre un intervalo de valores de DM, prácticamente multiplicando el número de operaciones requeridas por el número de valores de DM en la búsqueda. Típicamente, miles de valores de DM finamente espaciados necesitan ser considerados, por lo que toda la operación requiere un cálculo extremadamente intensivo. En el pasado, las búsqueda de transitorios de radio, como búsquedas de un impulso único de un Pulsar, se han realizado offline en el archivo de datos, utilizando herramientas estándar de computación para procesamiento en clusters de gran tamaño o supercomputadoras. La potencia de estas búsquedas se destaca por el descubrimiento de los radio transitorios rotantes (RRATs).

Aquí se discute cómo estas búsquedas se pueden realizar en línea mediante el uso de estándar de unidades de procesamiento gráfico de propósitos generales (GPU). Con la introducción, en los últimos años, de nuevas interfaces de alto nivel de programación de aplicaciones (API), tales como CUDA (NVIDIA Corporation 2010), estos dispositivos se pueden utilizar fácilmente para la descarga de datos en paralelo, algoritmos de procesamiento intensivo de CPU. La idea de realizar el procesamiento de señales transitorias con en las GPU no es nueva. Por ejemplo, Allal et al. (2009)

han creado una configuración totalmente funcional y en tiempo real, basada en GPU para procesamiento coherente de de-dispersión en el Nan AY Radio Observatorio. La libraría de procesamiento digital de señales para astronomía de púlsares, escrita por Van Straten y Bailes (2010), también se puede utilizar para llevar a cabo la de-dispersión coherente con GPU.

Barsdell, Barnes & Fluke (2010) analizan varios algoritmos fundamentales que se utilizan en astronomía y tratan de determinar si se pueden implementar en un entorno de computación paralela masiva. Uno de los algoritmos que analizan es el algoritmo de de-dispersión por fuerza bruta, y afirman que esto probablemente se realice con una alta eficiencia en dicho entorno, mientras que indican que la intensidad aritmética óptima es poco probable de determinar sin un análisis detallado de los patrones de acceso a memoria del algoritmo. El algoritmo de de-dispersión incoherente de hecho ha sido implementado usando CUDA como un caso de prueba para el proyecto CRAFT ASKAP (ver Macquart et al. y Dodson et al. 2010).

# 2. PARÁMETROS PARA BÚSQUEDA CIEGA DE TRANSITORIOS RÁPIDOS

Los estudios de transitorios rápidos se basan en un número de parámetros de búsqueda, que dependen de las características de los instrumentos que se utilizan y las características de las señales del objetivo astrofísico (Cordes 2008). Por ejemplo, un estudio de dispersión de transitorios rápidos pueden ser diseñados usando los siguientes parámetros de entrada: (i) *la frecuencia central* a la cual se realiza la observación (ii) el *ancho de banda de frecuencia* (iii) el *número de canales de frecuencia y el ancho de banda del canal* (iv), la t*asa de muestreo* a la cual los datos digitales están disponibles (v), el *ancho de señal esperado*, lo que depende del caso (vi) y el nivel de *relación señal a ruido* (S / N) aceptado de una señal detectada.

La dispersión y el scattering son dependientes de la frecuencia y el ancho de banda al cual se realiza el estudio, y ayuda a definir los límites de un estudio de la siguiente manera:

El **valor máximo DM** (DMmax) puede ser elegido de acuerdo con el valor máximo DM en el que hay expectativa justificada de descubrir fuentes. Esto puede estar relacionado con las coordenadas galácticas del estudio, las capacidades del hardware y también las consideraciones relativas al scattering interestelar. El scattering tiene el efecto de reducir el pico de S/N de una señal, y se relaciona con DM a través de una relación empírica. (ver Bhat et al. 2004).

El número de **canales de frecuencia** depende del ancho de banda máximo aceptable. Esto puede ser escogido teniendo en cuenta el corrimiento de dispersión dentro de cada canal para las DM más grandes en el intervalos de DMs buscado. Dependiendo del formato de los datos transmitidos por el telescopio, puede ser deseable y necesario generar canales de frecuencia más estrechos.

La **etapa de de-dispersión**, que es la unidad de discretización del intervalo de DM, se elige en función del ancho de las señales astrofísicas del objetivo y la perdida de S/N admisible que se produce cuando el de-dispersado está en una DM ligeramente errónea.

La de-dispersión en si misma se puede realizar mediante un algoritmo de fuerza bruta, introduciendo desplazamientos de tiempo para cada canal, o utilizando algoritmos de de-dispersión de aproximaciones rápidas. La técnica de subbanda de de-dispersión usada por PRESTO[1] (Ransom S. 2001), y se basa en el hecho de que los valores adyacentes de DM a menudo reutilizan las mismas muestras de tiempo para crear la serie de tiempo de-dispersa. Toda la banda se divide en un

1 PRESTO es un software de búsqueda y análisis de púlsares desarrollado por Scott Ransom

número de canales, o subbandas. Cada subbanda se de-dispersa de acuerdo con un conjunto de valores groseramente espaciados de DM y se colapsa en un solo canal de frecuencia, representante de la banda. El reducido número de canales pre-procesador son entonces de-dispersos usando un paso mucho más fino de DM, lo que resulta en la de-dispersión de las series de tiempo para todos los DMs dentro del intervalo de búsqueda. Esta técnica resulta en una pérdida ligera de sensibilidad, pero disminuye en gran medida el tiempo de procesamiento. Las subbandas de de-dispersión requieren parámetros adicionales, que en PRESTO se ofrecen como un plan de estudio. Esto determina la forma en la cual el intervalo de DM se divide en pasos, donde cada paso categorizará los datos utilizando un factor de categorización diferente (cuanto más grande es el DM, mayor será la deformación de la señal en el tiempo, con lo que podremos reducir el coste computacional por un promedio de muestras) . Para cada paso el intervalo de DM será dividido, cada intervalo ΔSubDM será separado. Este valor determina el paso de subbanda DM entre valores consecutivos de DM nominales. El paso DM se utiliza entonces para dividir cada una de estas particiones.

La Tabla 1 proporciona un ejemplo de un plan de estudio para una intervalo amplio de DM (0 - 150,66 $pc \cdot cm^{-3}$ ) producido por DDplan, una secuencia incluida en PRESTO, que genera los parámetros del estudio tratando de minimizar el corrimiento inducido por la división del ancho de banda en subbandas. Este corrimiento es el efecto aditivo del corrimiento sobre cada canal, cada sub-banda, el ancho de banda total y la velocidad de muestreo (suponiendo el peor error de DM de todos los casos).

| Pass | Low DM ($pc\ cm^{-3}$) | High DM ($pc\ cm^{-3}$) | ΔDM ($pc\ cm^{-3}$) | Bin | Δ $Sub_{DM}$ ($pc\ cm^{-3}$) |
|---|---|---|---|---|---|
| 1 | 0.00 | 53.46 | 0.03 | 1 | 0.66 |
| 2 | 53.46 | 88.26 | 0.05 | 2 | 1.2 |
| 3 | 88.26 | 150.66 | 0.10 | 4 | 2.4 |

*Tabla 1: Ejemplo de plan de estudio de de-dispersión subbanda***.**

# 3. DE-DISPERSION BASADA EN GPU

Usando la API de CUDA, hemos implementado de-dispersión incoherente, que funciona en cualquier GPU con CUDA conectado a un ordenador host. Nuestro código modular paraleliza el host y la ejecución en el GPU mediante el uso de múltiples hilos durante la entrada, procesamiento y salida. Mientras que el hilo de entrada está leyendo datos, el GPU está ocupado realizando la de-dispersión de un buffer previamente leído y el hilo de salida esta post-procesando una series de tiempo de-dispersa. Esta configuración en hilos se representa en la figura 1.

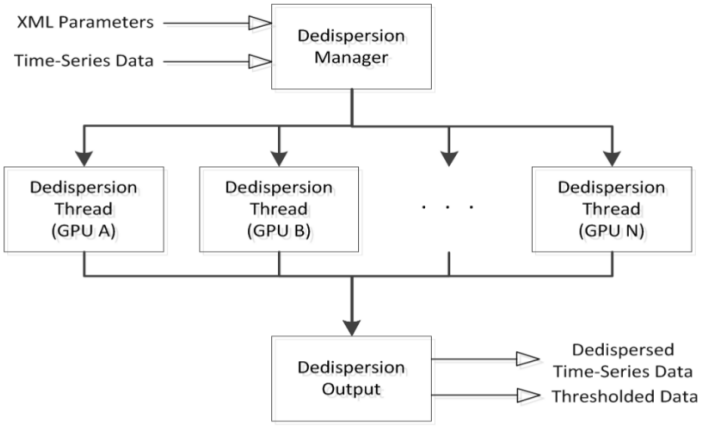



*Figura 1: El jerarquía de hilos de alto nivel. El gestor de de-dispersión inicializa todo el sistema y se encarga entonces de la lectura de los datos de entrada. Se crea un hilo de de-dispersión por GPU que se ocupa de todas las copias de memoria y la ejecución del kernel en ese GPU. El hilo de salida de de-dispersión recibe entonces la serie de tiempo de-dispersa de todo los GPU y realiza una ráfaga de búsqueda simple.*

El gestor de-de-dispersión es el hilo principal, que se encarga de la inicialización y la sincronización del resto de la aplicación. El hilo de entrada se encarga de todas las llamadas relacionadas con CUDA, y una instancia para cada GPU adjunto se crea. El rango de DM se divide entre estos hilos, de tal manera que cada uno procesará el mismo buffer de entrada para diferentes valores de DM. La salida de de-dispersión de la memoria del GPU se copia a la memoria principal, a una dirección que se desplaza adecuadamente para dar cabida a todos los hilos de entrada.

El hilo de la salida construye la series de tiempo de-dispersa y envía los resultados a un archivo de datos. También es responsable de la detección de eventos. El enfoque más simple es a continuación calcular la media (μ) y la desviación estándar (σ) de todo el buffer procesado, y utilizar estos valores para aplicar un umbral en particular, un múltiplo la desviación estándar (nσ). Todos los valores que están por encima del umbral son enviados a un archivo como una lista de tripletes de la forma. (tiempo, DM, intensidad).

Actualmente, se supone un sistema homogéneo, y no se realiza balanceo de carga entre los dispositivos. Cada hilo se divide en tres "etapas de procesamiento" conceptuales, que son custodiadas por varios mecanismos de sincronización de subprocesos. Esta configuración se muestra en la figura 2. Las tres etapas son: (i) la sección de entrada, entrada de datos a ser procesados por el hilo (ii) la sección de procesado, que contiene el núcleo de de-de-dispersión, es la sección principal del hilo y la parte que más se tarda en completar (iii) la sección de salida, donde el buffer de salida es procesado y puesto a disposición del siguiente hilo y cualquier actualización de los parámetros se realiza.

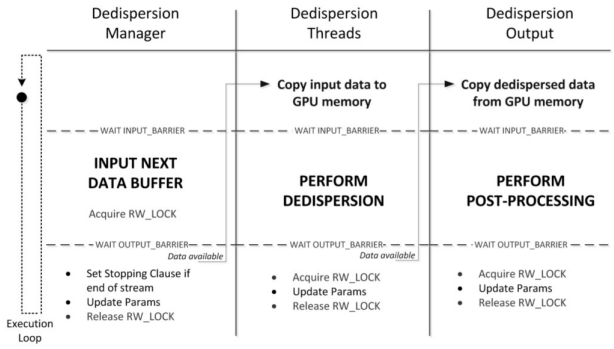



*Figura 2. Cada hilo tiene tres etapas principales: las etapas de entrada, procesamiento y salida. Los datos tienen que fluir de un tipo de hilo al siguiente, por lo que objetos de sincronización tienen que ser utilizados para asegurarse de que no se sobrescriben los datos o se vulvan a procesar. Barrera y bloqueos de RW se utilizan para controlar el acceso a las secciones críticas. Véase el texto principal para una explicación más detallada.*

Las dos tecnicas de de-dispersión antes mencionadas se han aplicado, a saber, la fuerza bruta y la sub-banda de de-dispersión, que entre ellas tienen algunos elementos en común:

**Desplazamiento máximo**: En un buffer que contiene nsamp muestras a procesar con un valor de DM distinto de cero, cada canal requerirá un cierto desplazamiento, donde los canales de las frecuencias más bajas requieren el mayor desplazamiento. Suponiendo que este cambio es de s muestras, y nsamp muestras necesitan ser de-dispersa, se requieren nsamp + s muestras que estén disponibles, donde s es dependiente del valor del DM. Dado que el GPU estará de-dispersando ndms valores de DM en cualquier momento, se necesita la cantidad de muestras adicionales para atender el valor máximo DM. Como referencia, se utilizará el término maxshift (mshift) para este cambio, que puede calcularse mediante la manipulación de la ecuación 1:

$$m_{shift} = \frac{8.3 \times 10^6 ms \times \Delta f \times f^{-3} \times DM_{max}}{t_{samp}} \quad (2)$$

donde tsamp está en ms, DMmax es el valor máximo de DM procesad en el GPU y $f$ y $\Delta f$ están en MHz.

Muestras procesables: la transferencia de datos entre el GPU y la CPU es ineficientes. Por esta razón, el buffer de entrada deben de llenar la mayor cantidad de memoria del GPU como sea posible, dejando suficiente espacio para almacenar el valor de maxshift y el búffer de salida. La manera más simple de calcular el número máximo de muestras que se ajustan en la memoria es:

$$n_{samp} = \frac{memoria - (m_{shift} \times n_{chans})}{n_{dms} + n_{chans}} \quad (3)$$

El objetivo es mantener todos los datos dentro de la memoria del GPU y realizar todo el procesamiento, de esta manera si hay un número de operaciones a realizar (como el binning), nsamp deben acomodarse.

Desplazamientos: Cada canal requiere una cantidad diferente de desplazamiento, para cada DM. Para un buffer de entrada con $nchans$ canales, cuando se de-dispersa para $ndms$ valores de DM, la estructura de datos requerida tiene un tamaño de $ndms \times nchans$ valores. Para la cantidad de canales y valores de DM requeridos para la mayoría de las situaciones, esta estructura de datos no cabe en la memoria constante, y podría reducir en gran medida la velocidad de ejecución si se

calcularan para cada bloque, para cada DM. Almacenar estos valores en la memoria global desaceleraría el núcleo. Para contrarrestar esto, el cálculo se divide en dos partes, con la primera parte realizada en la CPU:

$$t_{chan} = 4.15 \times 10^3 \times \left(f_1^{-2} - f_2^{-2}\right) \quad (4)$$

donde $f_1$ y $f_2$ están en MHz. Esto da un desplazamiento independiente de DM para cada canal de frecuencia (es decir, el retardo de dispersión por unidad DM), resultando en una estructura de datos de tamaño *nchans* , que en circunstancias normales se ajustará a la memoria constante. La segunda parte del cálculo se realiza entonces en el GPU:

$$t_{DM} = \frac{t_{chan} \times DM}{t_{samp}} \quad (5)$$

La división por $t_{samp}$ podría realizarse también en el CPU, pero esto daría lugar a errores de redondeo cuando se moldea el resultado a un solo punto de precisión (el valor en el que el GPU operaría) como $t_{chan}$ es generalmente muy pequeño es más eficiente realizar el cálculo en el GPU en lugar de utilizar doble precisión.

## 3.1. Dedispersión incoherente por fuerza bruta

El algoritmo de fuerza bruta es el más simple y más preciso de implementar par la de-dispersión de datos incoherentes, pero es también el menos eficiente en procesamiento inteligente. Suponiendo $N_s$ muestras, cada una con $N_c$ canales y de-dispersado para $N_{DM}$ valores DM, la complejidad de y tiempo algorítmica del algoritmo de fuerza bruta es $O\left(N_s \times N_c \times N_{DM}\right)$ . $N_s$ puede ser visto como un flujo infinito de muestras, mientras que $N_c$ y $N_{DM}$ por lo general tienen un valor similar, dando como resultado aproximado $N^2$ operaciones para cada muestra de entrada.

Existen tres formas principales en las que se puede paralelizar este algoritmo (Barsdell, Barnes & Fluke 2010): (a) $N_s$ hilos paralelos, cada uno calcula la suma de una única entrada de una muestra de tiempo para todos los canales de forma secuencial (b) $N_c$ hilos paralelos cooperan para sumar cada muestra a su vez (c) $N_s \times N_c$ hilos paralelos cooperan para completar el cálculo completo en paralelo. La implementación actual utiliza una variante del esquema de (a), donde cada hilo resume la entrada para una única muestra de tiempo. Debido al gran número de muestras que pueden caber en la memoria del GPU, cada hilo termina el procesando más de una muestra. Una manera de entender esto es imaginar la cuadrícula CUDA como una ventana deslizante que se mueve a lo largo de las muestras de entrada, a intervalos discretos iguales al número total de hilos en una fila. En cada posición de la cuadricula, los hilos son asignados a sus respectivas muestras.

El núcleo puede procesar cualquier número de valores DM simultáneamente, y esto se hace mediante la creación de una red de dos dimensiones, donde se asigna a cada fila un valor diferente de DM para de-dispersión. La salida de $N_{DM}$ series de tiempo, cada una con $N_s$ muestras, es la salida de la salida del hilo para el post-procesamiento.

Este núcleo no es de cálculo muy intensivo, realizando menos de diez operaciones de punto flotante por ciclo de lectura de la memoria global . Esto hace que el algoritmo de de-dispersión se limitado por memoria. Por esta razón, dependiendo de la forma en que los datos se leen desde el dispositivo de entrada, un cambio de sentido (matriz transpuesta) podría ser necesaria con el fin de almacenar

los datos en el orden de canal. Con esta configuración de memoria, y que cada hilo procesando una muestra por valor de DM, los hilos dentro de medio warp[2] (16 hilos), accederán al buffer de entrada de una manera casi completamente fusionada. Esto también se aplica para almacenar el resultado en el buffer de salida desde todos, ya que todos los hilos de una fila se desplazarán por la misma cantidad, lo que resulta en almacenamientos que se realizan de una manera fusionada también.

La memoria compartida se utiliza también para reducir la lectura de la memoria global. Cada valor de salida requiere $N_c$ adiciones, y la realización de estas adiciones en la memoria global reduciría drásticamente el rendimiento. Para contrarrestar esto, a cada hilo se asigna una celda de memoria compartida donde se realizan las adiciones. El resultado final se copia a la memoria global.

## 3.2. De-dispersión sub-banda

La de-dispersión sub-banda utiliza aspectos de la de-dispersión por fuerza bruta, sin embargo, también está influenciada por el algoritmo de árbol, que reutiliza sumas de los grupos de canales de frecuencia para diferentes valores de DM. Se basa en el hecho de que los valores de DM adyacentes (dado un paso de DM apropiado) utilizará muestras superpuestas durante la suma, por lo que se divide el rango de DM en varios sub-intervalos, cada uno centrado alrededor de un valor de DM nominal. El ancho de banda también se divide en varias sub-bandas, lo que resulta en una partición del conjunto de canales. Los retardos correspondientes a la DM nominal para cada canal en una sub-banda, menos el retardo de la frecuencia más alta en esa subbanda, se restan de cada sub-banda de canal. Esto resulta en un conjunto de sub-bandas parcialmente de-disperso. Este esquema se representa en la figura 3. La de-dispersión normal es usada para generar la serie de tiempo de-dispersos para el resto de los valores de DM dentro de la mismo sub-intervalo de DM.

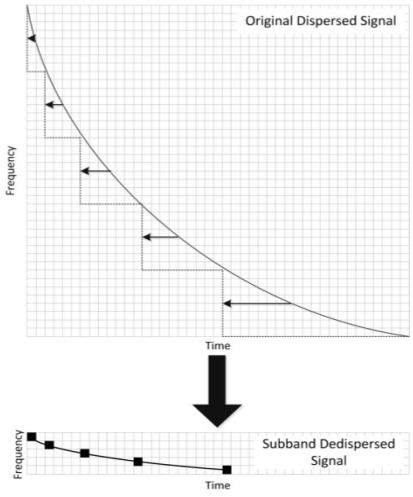



*Figura 3. Ejemplo simple para ilustrar visualmente cómo trabaja la de-dispersión subbanda. Los canales se dividen en un conjunto de sub-bandas, donde los retardos correspondientes a la DM nominal para cada canal en la sub-banda, menos el retardo a la frecuencia más alta en la sub-banda, se restan de cada canal. La señal de subbanda de-dispersa es entonces de-dispersada usando la de-dispersión por fuerza bruta para un rango de valores DM alrededor del valor nominal de DM.*

Dependiendo del número de subbandas utilizadas, el tamaño de los rangos de DM, así como otros

2 Warp: Un conjunto o grupo de 32 hilos forma un warp. Los hilos de un mismo warp corren en el mismo multiprocesador.

factores, se puede limitar el error inducido en el resultado de estas aproximaciones. Mejoras se pueden hacer mediante el binning de las muestras de entrada cuando la dispersión es tan alta que un pulso con una anchura requerida se esparce a través de múltiples muestras de entrada. Se agruparán $N_b$ medias de muestras consecutivas. Para hacer el código GPU compatible con la forma en que PRESTO genera sus parámetros para estudios, el agrupamiento también se implementa. Hay, pues, tres grandes etapas en la de-dispersión por sub-banda:

(i) Realizar agrupación de datos, si es necesario.
(ii) Realizar de-dispersión subbanda y generar la serie de tiempo intermedio.
(iii) De-dispersar la serie de tiempo resultante para generar la serie de tiempo de-dispersa final.

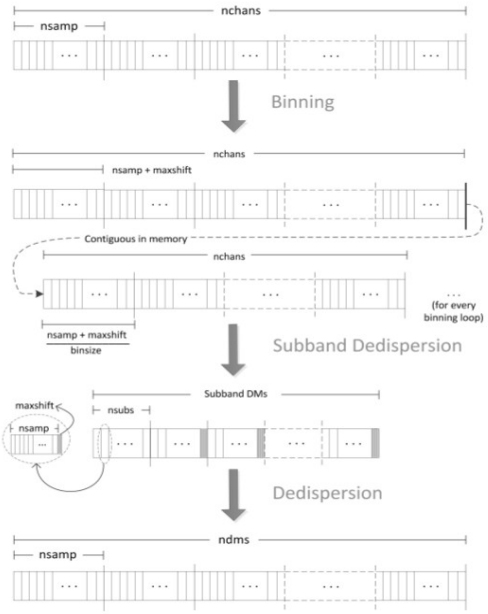



*Figura 4. La de-dispersión subbanda requiere de tres pasadas de los datos: binning (agrupamiento), de-dispersión sub-banda y de-dispersión por fuerza bruta. Durante todo el proceso los datos se guardan en la memoria del GPU, y esta ilustración muestra cómo se organizan los datos antes y después de cada pasada.*

Las transferencias de datos entre el host y el GPU son costosas, por lo que las etapas anteriores se llevan a cabo en el GPU sin que ningún dato vuelva al host, con una apropiada reorganización del buffer de datos después de cada ejecución del kernel. La organización de la memoria después de cada etapa se representa en la figura 4, que puede ser descrita de arriba hacia abajo como sigue:

(i) el buffer de entrada, que contiene $(n_{samp}+m_{shift})\times n_{chans}$ valores.
(ii) el agrupamiento (binning) de $N_b$ medias de muestras adyacentes, por lo que la salida de un bucle de agrupamientoserá $(n_{samp}+m_{shift})\times n_{chans}/N_b$ , donde cada bucle tiene un valor diferente para $b$ . La salida de cada bucle de agrupamiento se coloca en la cola de la salida anterior, como se muestra en el diagrama. Para $l$ bucles, el buffer terminará teniendo $l$ bloques lógicos, cada uno con un tamaño de bin iferente $b_i$ . La memoria, en muestras, que se requiere para este procedimiento es:

$$mem=\sum_{i=0}^{l}\left(\frac{(n_{samp}+m_{shift})\times n_{chans}}{b_i}\right) \quad (6)$$

(iii) La de-dispersión en subbandas genera $N_{sub}$ series de tiempo intermedias para cada DM nominal, cada uno compuesto de $(n_{samp}+m_{shift})/b_i$ muestras que contienen canales $n_{sub}$ . El maxshift de las muestras tienen que ser preservado para que la próxima etapa pueda procesar todas

las $n_{samp}$ . La memoria, en muestras, requerida para esta etapa es:

$$mem = \sum_{i=0}^{l} \left( \frac{(n_{samp} + m_{shift}) \times n_{sub} \times N_{sub}}{b_i} \right) \quad (7)$$

(iv) El resultado final consiste en la serie de tiempo de-dispersos para todos los valores de DM, cada una conteniendo $n_{samp}/b_i$ valores. Así, el requisito de memoria para esta etapa, en muestras, es el siguiente:

$$mem = \sum_{i=0}^{l} \left( \frac{n_{dms} + n_{samp}}{b_i} \right) \quad (8)$$

Habiendo definido los requisitos de memoria de entrada y salida para todas las etapas de GPU, el número de muestras que se pueden procesar, entonces se pueden calcular. El tamaño de los buffers de entrada y salida se puede calcular tomando el tamaño del buffer más grande respecto de las etapas de procesamiento, el cual puede entonces ser utilizado para calcular el número de muestras que caben en la memoria.

La núcleo de de-dispersión de sub-banda es muy similar al de fuerza bruta, el único cambio importante es que no todos los canales se suman para generar la serie, y más de un valor es generado por cada muestra de entrada. Esto hace que el algoritmo sea menos intensivo en el cálculo y más limitado en memoria (mismo número de solicitudes de entrada, más peticiones de salida). Sin embargo el número de los valores nominales de DM es sólo una fracción del número total de valores de DM, y esto reduce en gran medida el número de cálculos que deben ser realizados por el algoritmo de fuerza bruta.

# 4. RESULTADOS Y COMPARACIÓN

Para probar el código, un archivo que contiene una señal de pulso se ha generado utilizando el generador de pulsos falso que viene co SIGPROC[3] (Lorimer, http://sigproc.sourceforge.net). Los parámetros que se usaron para generar el archivo falso se enumeran en la tabla 2. Los datos de filterbank falsos son generados como 1024 series de tiempo, uno para cada canal de frecuencia. Cada uno se compone de un pulso cuadrado de la altura $8\sqrt{1024} = 0.25$ y ruido Gaussiano con media 0 y desviación estándar 1. La SNR del pulso promedio simulado, integrada en frecuencia tiene un valor medio de 8.

| Parameter | Value |
|---|---|
| Pulsar Period | 1000 ms |
| Duty Cycle | 1% |
| Pulsar DM | 75.00 pc $cm^{-3}$ |
| Top Frequency | 156.0 MHz |
| Channel Bandwidth | 5.941 kHz |
| Number of Channels | 1024 |
| Sampling Time | 165 $\mu s$ |

*Tabla 2. Los parámetros utilizados para generar el archivo falso para la evaluación. Un pulso con un período de 1 segundo y 1% del ciclo de trabajo se ha creado en una frecuencia central de 153 MHz con ancho de banda de 6 MHz. El ancho de banda se divide en 1.024 canales y se utilizó un tiempo de muestreo de 165 ms. La DM es de 75 $pc\ cm^{-3}$ .*

3 SIGPROC es un paquete de software diseñado para estandarizar el análisis de varios tipos de datos de pulsares muestreados rápido

Se realizó de-dispersión por fuerza bruta con valores de DM 1000 con un paso entre DM de $0.1\ pc\ cm^{-3}$ . La figura 5 muestra la salida del código de dispersión, que captura todos los pulsos con SNR mayor que 5.

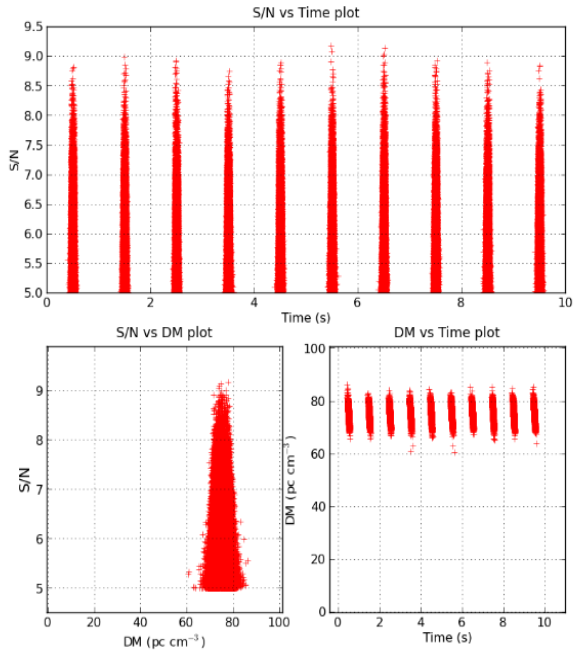
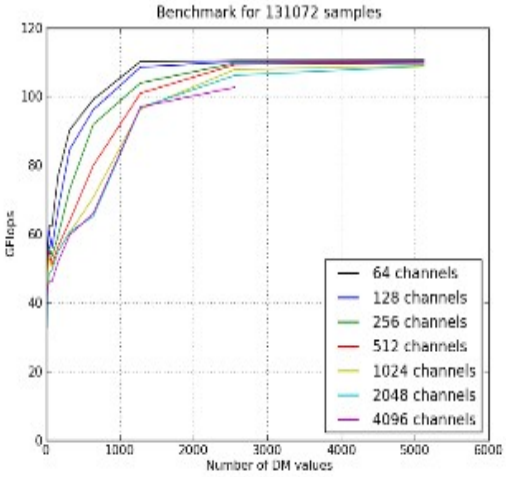


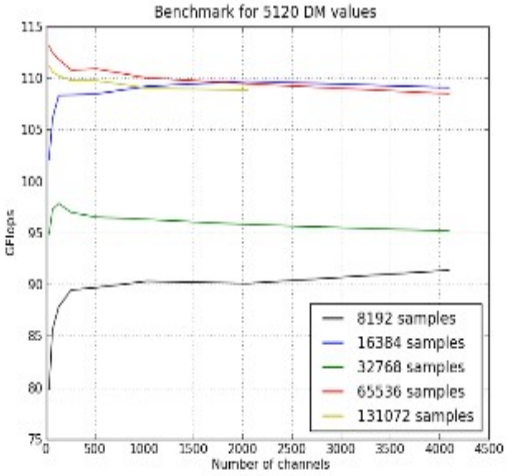

*Figura 5. Salida de de-dispersión por fuerza bruta para un archivo de entrada que contiene un púlso simulado (véase el texto para más detalles). Los gráficos corrsponden a SNR en función del tiempo (arriba), SNR en función del DM (abajo a la izquierda) y DM en función del tiempo (parte inferior derecha). Un umbral de 5σ se aplica a la salida.*

El rendimiento de las implementaciones CUDA se ha medido. Los datos falsos se generan durante la ejecución, con todos los elementos inicializados co el mismo valor. El tiempo necesario para generar y copiar los datos hacia y desde la memoria GPU no está incluido en las temporizaciones.

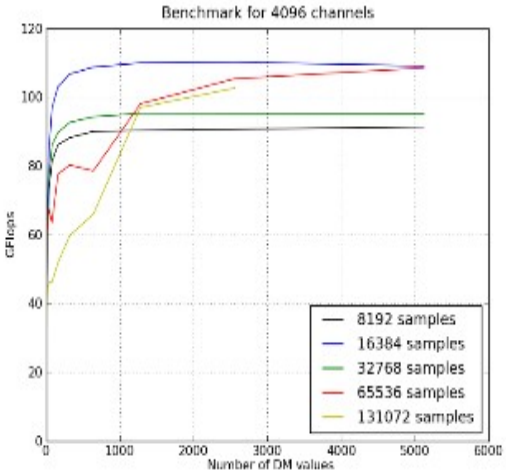
La figura 6 muestra el rendimiento alcanzado durante el dedispersado con diferente número de canales, muestras y valores de DM. Las diferentes configuraciones de parámetros dará lugar a combinaciones óptimas diferentes, por ejemplo, en los casos en que el número de particiones de datos a procesar es exactamente divisible por el número de procesadores y bloques de hilo que está utilizando. La tendencia general es que el rendimiento aumenta linealmente con el número de canales, muestras y valores de DM hasta que el nivel máximo de ocupación del GPU se alcanza, después de lo cual el comportamiento se vuelve asintótico. El tamaño de bloque óptimo es de 128, ya que un menor número de hilos se traducirá en menor latencia oculta y más hilos aumentará la latencia sin beneficios de rendimiento. El tamaño de la cuadrícula no afecta demasiado el rendimiento, excepto para el caso donde hay demasiados hilos en cada bloque.




(a) Maximum number of samples

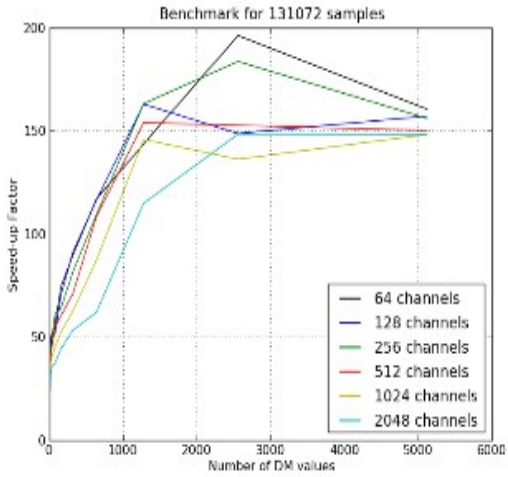



(b) Maximum number DM values

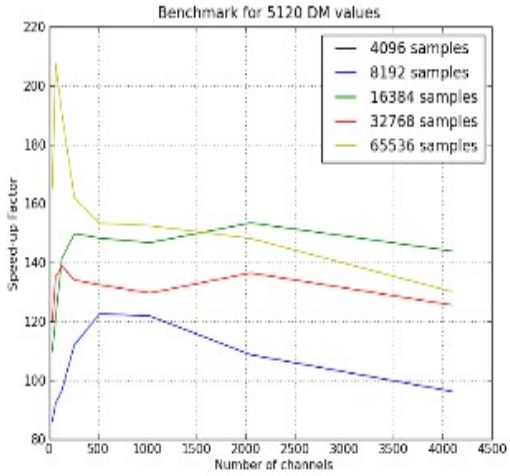



(c) Maximum number of channels

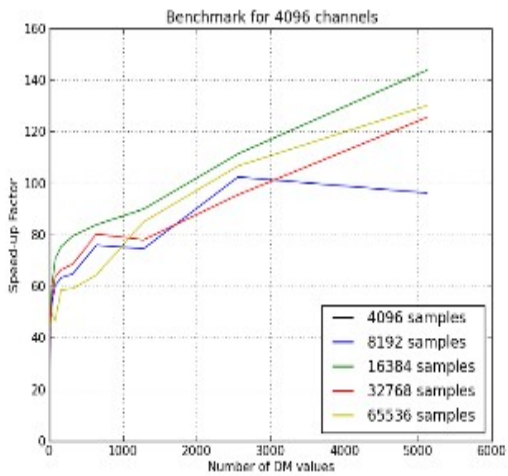
Figura 6. Rendimiento de-dispersión por fuerza bruta para una implementación en CUDA.




(a) Maximum number of samples




(b) Maximum number DM values




(c) Maximum number of channels

Figura 7. Velocidad de de-dispersión por fuerza bruta.

Como ya se dijo, el algoritmo de dispersión está enlazado a la memoria, y tanto el GPU y la CPU pasarán la mayor parte de su tiempo a la espera de los datos. Por esta razón, la tasa de Flop alcanzado en el GPU es un porcentaje pequeño de la máxima teórica para la C-1060, entre 80 y 120 Gflops, que asciende a alrededor de 15-20%. El ancho de banda de memoria logrado en el GPU es de aproximadamente 55 GB/s, que es aproximadamente 50% de la máxima teórica.

Las mismas pruebas se realizaron en una CPU, específicamente en un núcleo de un QuadCore Intel Xeon 2,7 GHz. El rendimiento de la CPU disminuye casi linealmente a medida que el número de muestras o canales aumenta debido a fallos de caché. Este rendimiento es comparado con el rendimiento del el GPU para producir las gráfico de comparación que se ven en la Figura 7. Esto demuestra la aceleración de desempeño obtenida en la dedispersion por fuerza bruta utilizando GPUs, para diferentes valores de parámetros. A partir de estos diagramas se desprende que en promedio se obtiene una velocidad de aproximadamente dos órdenes de magnitud, entre 50x - 200x dependiendo de los parámetros utilizados, con el aumento de velocidad como el aumento de número de muestras de entrada por canales.

La implementación CUDA fue comparada con los dos scripts comúnmente utilizados, la de-dispersión con PRESTO y con SIGPROC. Un archivo falso fue generado contienendo una observación de 600 segundos centrada en 153 MHz con un ancho de banda de 6,24 MHz dividida en 1024 canales y que tiene una velocidad de muestreo de 165 ms (que contiene un total de aproximadamente $3{,}6 \times 10^6$ muestras). Este ensayo se realizó a través de los tres paquetes de software por un valor DM único. Para el código del GPU y PRESTO sólo la actual de-dispersión fue cronometrada, ya que SIGPROC también carga el archivo en el bucle así que contiene algún tiempo de E / S de archivo también. Los tiempos se muestran en la tabla 3.

| Suite | Timing |
|---|---|
| GPU Code | 0.257s |
| PRESTO | 28.321s |
| SIGPROC | 58.099s |

*Tabla 3. Conparación de tiempo entre el Código GPU, Presto y SIGPROC para un DM.*

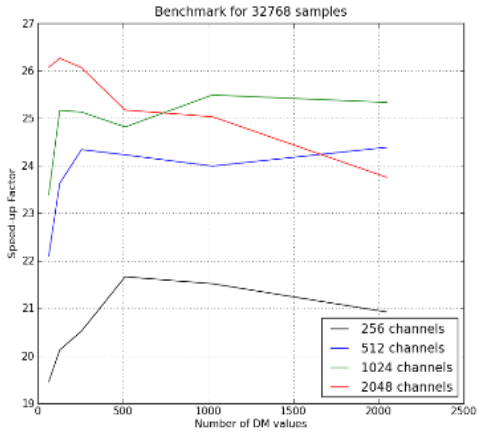
| Pass | Low DM (pc cm$^{-3}$) | High DM (pc cm$^{-3}$) | $\Delta DM$ (pc cm$^{-3}$) | Bin | $\Delta Sub_{DM}$ (pc cm$^{-3}$) |
|---|---|---|---|---|---|
| 1 | 0.00 | 53.46 | 0.03 | 1 | 0.66 |
| 2 | 53.46 | 88.26 | 0.05 | 2 | 1.2 |
| 3 | 88.26 | 150.66 | 0.10 | 4 | 2.4 |

*Tabla 4. Plan de estudio de-dispersión subbanda para comparar PRESTO y el código GPU, para una observación de 60-segundo a 300 MHz con un ancho de banda de 16 MHz dividido en 1024 canales.*

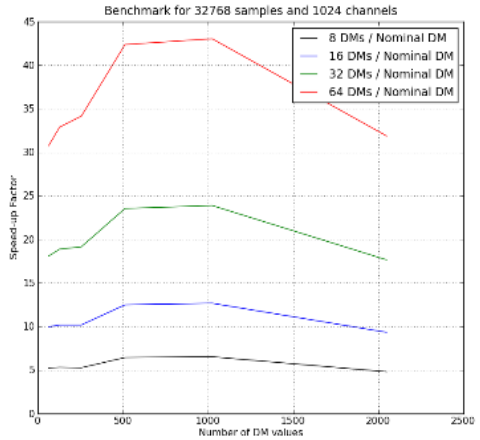
El algoritmo utilizado para realizar la de-dispersión subbanda (ambas etapas) es casi idéntico a la utilizada en el de de-dispersión por fuerza bruta, por lo que las pruebas de escala no se repitieron.

La Figura 8 representa un diagrama de comparación entre los dos algoritmos para varios parametros de de-dispersión. El factor de aceleración depende de las combinaciones óptimas de parámetros, así como el número de sub-bandas, el número de los valores nominales de DM en el intervalo de DM empleado para la de-dispersión subbanda. El factor de aceleración disminuye linealmente con el incremento del número de los valores nominales de DM para un rango particular, ya que se necesita más trabajo por hacer en la primera etapa del algoritmo (aunque el mismo número de valores DM se procesan, el primer paso será generalmente más intensivo ya que los datos no se han reducido todavía). El número de los valores nominales de DM y el número de sub-bandas dependerá de la

cantidad de de-dispersión permisible, donde un paso grande subbanda-DM y pocas subbandas resulta en una mayor cantidad de manchas. Estos valores deben ser ajustados para obtener el mejor equilibrio entre SNR y la velocidad de procesamiento.




(a) The speed up of subband versus brute force de-dispersion




(b) Different DM values per nominal DM in subband de-dispersion

*Figura 8. La velocidad relativa de la de-dispersión sub-banda frente a la de-dispersión por fuerza bruta en el GPU.*

La aplicación de-dispersión subbanda en el GPU también se comparó con el script prepsubband de PRESTO, sobre el que se basa el algoritmo. Un archivo falso con un solo haz de 60 segundos de observación a 300 MHz con un ancho de banda de 16 MHz y 1024 canales. El plan utilizado para la prueba se enumera en la tabla 4. El tiempo necesario para procesar todo el archivo con el código del GPU y con PRESTO fue de 90s y 7540s, respectivamente. Una vez más, esto indica que el código de GPU es aproximadamente dos órdenes de magnitud más rápido que la aplicación de CPU, en este caso 84x más rápido. Para esta prueba, PRESTO se ejecutó en modo de un solo hilo.

## 5. DE-DISPERSION EN TIEMPO REAL

El aumento de rendimiento obtenido de las GPU las convierten en un candidato ideal para el uso en sistemas de tiempo real. Un servidor con una tarjeta gráfica CUDA de alta gama tiene el poder suficiente para de-dispersar miles de valores de DM en tiempo real (dependiendo de los parámetros del telescopio). Características adicionales se requieren para tal sistema, tal como una forma de leer

e interpretar los datos de entrada del telescopio, y el buffer de canalización adicional entre el flujo de entrada y los buffers de de-dispersión.

Como prueba de concepto, el código GPU se amplió para incluir un canalizador (una simple FFT utilizando la librería NVIDIA CUFFT) y un núcleo para calcular la potencia de tensiones complejas entrantes, para simular la situación de conectar una máquina a un registrador de banda de base. Esto se utilizó dentro de una aplicación más amplia que (i) lee en paquetes UDP, llenando buffers dentro de un framework de doble buffer (ii) envía los buffers llenos hacia el código GPU (iii) canaliza y calcula la potencia total (iv) transpone los datos de manera que estén en orden de canal (v) realiza de-dispersión. Un dato UDP emulado se utiliza para crear un flujo de voltaje simulado a partir de archivos de datos falsos de SIGPROC y los envían al procesamiento canalizado. Esta configuración se muestra en la figura 9.

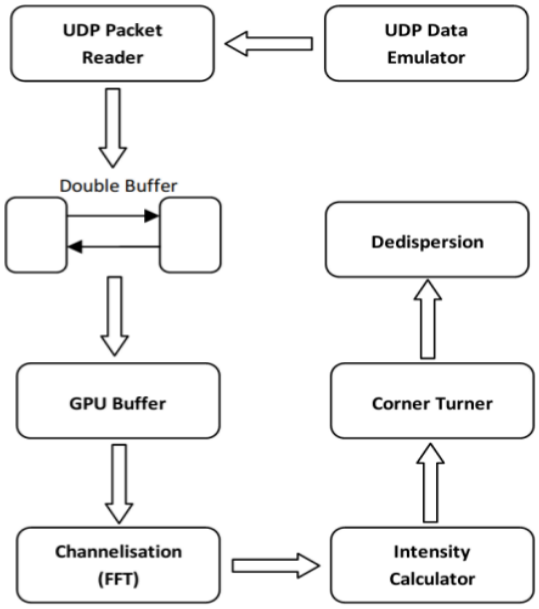



*Figura 9. Una configuración de de-dispersión en tiempo real. El emulador de datos UDP empaqueta archivos de SIGPROC y envía los datos a través de UDP a la cadena de procesamiento, donde los paquetes se leen, interpretan y almacenan en un buffer doble.*

Un archivo de observación falso fue generado, cuyos parámetros se definen en la tabla 5, lo suficientemente grande como para que múltiples iteraciones de la tubería se requieran, junto con los parámetros de procesamiento. Se utilizó el algoritmo de de-dispersión por fuerza bruta para la prueba. Tenga en cuenta que la velocidad de salida del emulador de datos no coincide con el la velocidad de salida de datos del telescopio simulado, por lo que la forma de determinar si la tubería está procesando en tiempo real es el tiempo el tiempo que el GPU necesita para procesar un buffer completo, y luego comparar esa con el número de muestras originalmente almacenados en el buffer.

| Parameter | Value |
|---|---|
| Center Frequency | 610 MHz |
| Bandwidth | 20 MHz |
| No. of Subbands | 256 |
| Sampling Time | $12.8\mu s$ |
| Channels per Subband | 8 |
| Number of DM values | 500 |
| Maximum DM value | 60 pc $cm^{-3}$ |

*Tabla 5. Una observación falsa para probar la tubería en tiempo real. Un archivo falso se generó con una observación en una frecuencia central de 610 MHz y ancho de banda de 20 Mhz con 256 canales, produciendo 78125 espectros por segundo. El canalizador produce 8 canales por subbanda, y 500 valores de DM se utilizan para la búsqueda de dispersión, con una DM máximo de 60 $pc\,cm^{-3}$ .*

| Stage | Time |
|---|---|
| CPU to GPU copy | 475ms |
| Channelisation | 458 ms |
| Intensity Calculation | 20 ms |
| Corner Turn | 112 ms |
| De-dispersion | 4500 ms |
| GPU to CPU copy | 220 ms |
| Total | 5785 ms |

*Tabla 6. Tiempo de varias etapas en la cadena de procesamiento.*

Los tamaños de buffer de GPU se estableció a $2^{19}$ espectros, lo que equivale a alrededor de 6.7s de datos del telescopio, lo que significa que todo el procesamiento del GPU para cada buffer debe completarse dentro de este marco de tiempo. Los tiempos promedio para cada etapa de la tubería se enumeran en la tabla 6. El tiempo total de procesamiento en una tarjeta NVIDIA C1060 es de aproximadamente 5,8s. Esto deja suficiente tiempo adicional para la sincronización CPU-GPU de las operaciones adicionales de memoria, también proporcionan un margen suficiente para el procesamiento del CPU a otros procesos en ejecución o el propio sistema operativo. La prueba se ejecuta en un servidor con dos procesadores Intel Xeon QuadCore 2.7 GHz y con 12 GB de memoria RAM DDR3, que es un modesto sistema de procesamiento en línea.

# 6. CONCLUSIONES

Se han implementado dos algoritmos de de-dispersión, fuerza bruta y sub-banda, usando CUDA, que permite el procesamiento paralelo de datos para ser ejecutado en cualquier número de GPUs con CUDA. Esto ha conducido a una aceleración del rendimiento de aproximadamente dos órdenes de magnitud, entre 50 y 200 para las configuraciones de los parámetros determinados, en comparación con una implementación de la CPU de un solo subproceso. Comparación detallada con dos aplicaciones tradicionales de análisis de pulsares, Presto y SIGPROC, confirman los resultados. Por último, un prototipo de tubería de de-dispersión para búsqueda en tiempo real fue diseñado, que lee un flujo de datos UDP del telescopio y realiza la canalización por FFT y la de-dispersión.

Se está trabajando en este proyecto, con planes de añadir varios módulos de procesamiento adicional en la tubería. La de-dispersión coherente es útil para estudiar púlsares cuyos valor de DM ya se conoce. Otros esquemas de de-dispersión también se están considerando, tales como la realización de análisis de chirps en el dominio de la frecuencia para detectar chirps que representan las señales dispersas. El GPU nos proporciona suficiente poder de procesamiento (por unidad de coste) capaz de aplicar algoritmos de procesamiento intensivo que de otro modo no sería factible en un sistema CPU convencional. Como resultado, estamos en condiciones de llevar a cabo búsquedas

en tiempo real para radio-transtorios rotantes con telescopios adecuados de bajo costo.