UNIVERSIDAD POLITÉCNICA DE VALENCIA
DEPARTAMENTO DE SITEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster en Computación Paralela y Distribuida

SIMULACIÓN COMPUTACIONAL Y PARALELIZACIÓN DE UN SISTEMA DE COMUNICACIONES INALÁMBRICO MIMO: ESTIMACIÓN DEL CANAL Y DECODIFICACIÓN DE SEÑALES

Vicente Puig Borrás

Dirigida por:

Dr. Antonio M. Vidal Maciá

Valencia, Febrero de 2011

Resumen

MIMO (Multiple Input Multiple Output), cuyo nombre viene del empleo de múltiples antenas transmisoras y receptoras, es una tecnología relativamente nueva de enlace inalámbrico que permite una mayor capacidad , alcance y fiabilidad en las comunicaciones con respecto a los sistemas tradicionales de una sola antena transmisora y una receptora. En la presente tesis se va a realizar un estudio en profundidad de todo el proceso de una comunicación en un sistema MIMO, desde la estimación de las condiciones del canal, transmitiendo para ello señales piloto, hasta la decodificación de la señal utilizando algoritmos de ramificación y poda, centrándonos en el Automatic Sphere Decoder de Karen Su. Una vez conseguido un algoritmo que integre todo el proceso se paralelizará aprovechando los múltiples núcleos que en la actualidad ofrecen las GPUs (unidades de procesamiento gráfico). Se trata de un concepto reciente dentro de la informática, llamado GPGPU (General-Purpose Computing on Graphics Processing Units), que trata de estudiar y aprovechar las capacidades de cómputo de una GPU.

Abstract

MIMO (Multiple Input Multiple Output), whose name comes from the use of multiple transmitting and receiving antennas, is a relatively new technology in wireless communication that allows higher capacity, range, and reliability in contrast of traditional single antenna transmitters and receivers. This thesis studies the whole process of communication in a MIMO system: the channel estimation using test signals and decoding received signal using branch and bound algorithms, focusing on the Automatic Sphere Decoder of Karen Su. Decoding phase has been parallelized taking advantage of the multicores offered by GPUs (Graphic Processing Units). This is a recent concept in computer science, called GPGPU (General-Purpose Computing on Graphics Processing Units), which aims to explore and exploit the computational power of a GPU.

Índice de contenido

Capítulo 1

Introducción y objetivos

En este primer capítulo se muestra una visión general del trabajo desarrollado, comentando las motivaciones, realizando un estado del arte del problema y planteando los objetivos que se pretenden alcanzar. Además, se presenta el hardware y software utilizado durante las pruebas experimentales, así como la estructuración del resto de capítulos.

1.1 Motivación

Debido a la alta demanda de una conectividad cada vez más sofisticada que permita la comunicación en cualquier momento y lugar, los sistemas de comunicación MIMO (Multiple Input – Multiple Output) son uno de los sectores más pujantes de la industria de las telecomunicaciones.

Un sistema MIMO esta formado por múltiples antenas transmisoras y receptoras. Estos sistemas ofrecen mayor fiabilidad y calidad de transmisión que los sistemas tradicionales de sólo una antena transmisora y una receptora [1,2].

Por una parte, el trabajo se centra en el estudio exhaustivo y simulación computacional de un sistema de comunicaciones inalámbrico discreto MIMO, desde la estimación de la Matriz de Canal hasta la detección y decodificación de las señales recibidas. Para ambos casos se probarán diversos métodos, aunque se hará mayor hincapié en los algoritmos de ramificación y poda, para finalmente conseguir un algoritmo adaptativo con el objetivo de mejorar la eficiencia y prestaciones del sistema, que en la práctica se reduce a expandir el menor número de nodos posibles, como se verá en el capítulo 4.

Por otra, se va a proceder a la paralelización de dicho algoritmo adaptativo, aprovechando para ello el potencial que ofrece actualmente la arquitectura de cálculo paralelo CUDA (Compute Unified Device Architecture). Se trata tanto de un compilador como de un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten usar una variación del lenguaje de programación C para codificar

algoritmos en GPUs (unidad de procesamiento gráfico), las cuales han evolucionado hasta un punto en que muchas de las aplicaciones industriales actuales se ejecutan en ellas con niveles de rendimiento muy superiores a los que ofrecerían si se ejecutasen en sistemas multinúcleo.

1.2 Formulación del problema

Consideramos un sistema de comunicaciones con *t* antenas transmisoras y *r* antenas receptoras, como se muestra en la Fig.1. La señal se propaga desde cada antena transmisora a cada una de las antenas receptoras, lo que obliga a representar la propagación a través de una matriz, donde el elemento h_{ij} representa la transferencia entre la antena emisora *i* y la antena receptora *j*. Esta matriz se conoce como matriz de canal y es usualmente denotada por **H**. Durante la comunicación, los datos son demultiplexados en *t* partes de igual tamaño para después mapearlos en valores discretos y transmitirlos sobre las *t* antenas transmisoras simultáneamente.

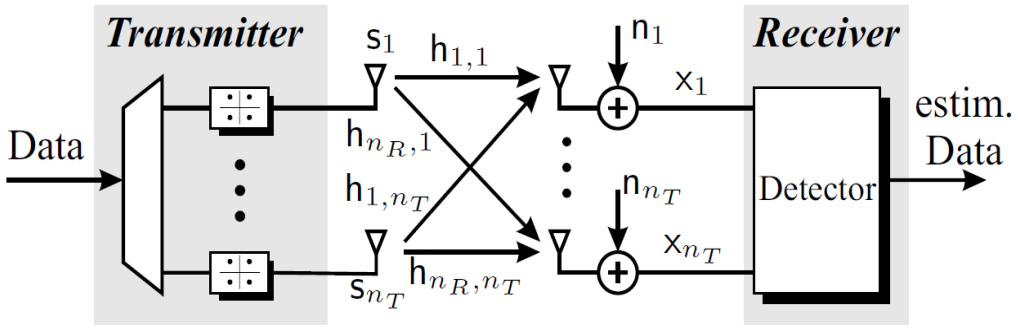


Fig. 1.1: Modelo de un sistema MIMO inalámbrico con *nt* antenas transmisoras y *nr* antenas receptoras.

Si **s** es el vector de señal compleja transmitida de tamaño *t*, y **x** el vector de señal compleja recibida de tamaño *r*, el sistema MIMO se podría representar de la siguiente forma [3]:

$$\mathbf{x=Hs+v} \qquad (1.1)$$

donde **H** es la matriz de canal, *rxt*, y **v** representa el ruido blanco gaussiano de varianza σ^2 que se aprecia en las *r* antenas receptoras.

Para evitar el tener que trabajar con valores complejos se tratarán las partes reales e imaginarias de forma separada, transformando el problema complejo en un problema real **x=Hs+v** donde:

$$H=\begin{bmatrix} \Re(H_c) & \Im(H_c) \\ -\Im(H_c) & \Re(H_c) \end{bmatrix} \quad x=\begin{bmatrix} \Re(x_c) \\ \Im(x_c) \end{bmatrix} \quad s=\begin{bmatrix} \Re(s_c) \\ \Im(s_c) \end{bmatrix} \quad v=\begin{bmatrix} \Re(v_c) \\ \Im(v_c) \end{bmatrix}$$

con **H** de tamaño *nxm*, **s** de tamaño *mx1,* y **x** y **v** de tamaño *nx1*, siendo m=2t y n=2r.

1.2.1 Estimación de la Matriz de Canal

Una de los enfoques más ampliamente usados para la estimación del canal en un sistema MIMO es emplear señales piloto (también llamadas secuencias de entrenamiento). Se envían dichas señales, y con las señales recibidas y el conocimiento de los valores discretos de la constelación, es posible estimar el canal.

En el siguiente capítulo se estudia el rendimiento de dos métodos basados en secuencias de entrenamiento, el método tradicional de mínimos cuadrados (LS) [4], el cual no requiere ningún conocimiento de los parámetros del canal, y una versión refinada de éste, llamado método de mínimos cuadrados escalado (SLS)[5], el cual mejora las prestaciones del primero, pero necesita que la matriz de correlaciones de canal y la potencia de ruido recibida sean conocidas de antemano.

1.2.2 Decodificación de la señal recibida

El segundo problema en la presente tesis es la decodificación de la señal recibida, donde el dominio de solución es un espacio multi-dimensional discreto. Consiste en la resolución de un problema de optimización por mínimos cuadrados en espacios discretos. Así, podemos estimar el valor de *s* en (1.1) resolviendo el problema:

$$\min_{s \in \mathcal{A}^m} ||x - Hs||_2 \qquad (1.2)$$

donde $\mathcal{A}^m$ representa el campo de posibles valores discretos que puede tomar *s* (también llamado constelación de símbolos como veremos más adelante).

No es viable hacer una búsqueda exhaustiva por el espacio discreto para encontrar la solución debido a que presentara una complejidad exponencial [6], por ello, han surgido en los últimos años muchas investigaciones para encontrar una solución a (1.2) de menor complejidad. Están los métodos heurísticos o sub-óptimos [7], que encuentran la solución invirtiendo la matriz de canal, y existen los métodos de máxima verosimilitud (ML, Maximum Likelihood), que sí obtienen la solución exacta aunque a un coste mayor, generalmente basando su funcionamiento en la utilización de esquemas de ramificación y poda (Branch-and-Bound)[8], reformulando el problema a encontrar un camino entre un nodo inicial del grafo (que representa un punto inicial del espacio discreto) y un nodo final (que representaría una solución al problema). Una clase de estos métodos son los métodos de búsqueda directa [9], que se caracterizan por el hecho de que el proceso de la toma de decisiones se basa sólo en los valores de la función objetivo

En la presente tesis se realiza un estudio de diversos métodos de solución, tanto por búsqueda heurística, como es el método *Zero Forcing*, como por búsqueda directa, como el método decodificación esférica (*Sphere Decoding,* SD), haciendo especial

énfasis en una variación de éste como es el *Automatic Sphere Decoding,* ASD [10]. De este método se estudian las modificaciones vistas en [11] para acortar el número de nodos expandidos, como es la utilización de un radio inicial obtenido mediante un método híbrido, *ASD + Zero Forcing,* y la descomposición en valores singulares SVD.

Como trabajo de investigación, se ha seguido otra línea para mejorar la eficiencia del ASD con la utilización de la factorización QR con pivotamiento (QRP)[12], que, combinado con las variaciones utilizadas en [11], reduce notablemente el número de nodos expandidos para encontrar la solución al problema de optimización (1.2).

1.3 Estado del arte

La comunicación mediante sistemas MIMO es uno de los desarrollos más significativos de la última década en el campo de las comunicaciones *wireless*. Se ha demostrado analítica y experimentalmente que en entornos de fácil dispersión estos sistemas ofrecen aumentos notables de capacidad [1,2,13], los cuales proporcionan mayor fiabilidad y calidad de la transmisión.

Sin embargo, para que esto ocurra, es necesario un buen conocimiento del estado del canal (channel state information, CSI). En estos últimos años, ha habido un creciente interés en los métodos de estimación del canal en sistemas MIMO basados en el empleo de señales piloto (también llamadas de entrenamiento) [4,14-18].

En este trabajo se ha hecho un estudio de las técnica de estimación del canal empleando este tipo de señales, utilizando el método de mínimos cuadrados (LS) [4], y una versión refinada de éste conocido por método de mínimos cuadrados escalado (SLS) [5], pero existen muchos estudios con diferentes métodos de estimación, como puede ser el uso del mínimo error cuadrático medio (MMSE [16], RMMSE [5]) . Se puede encontrar más información sobre esta temática en [19-21].

En la parte de la decodificación de la señal, como ya se ha comentado, encontrar la solución ML mediante búsqueda exhaustiva no es posible en términos prácticos. Por ello, a lo largo de los últimos años ha existido un gran interés por parte de los investigadores en encontrar soluciones de menor complejidad [22].

En los últimos años los métodos de Búsqueda Directa han vuelto a tomar cierto protagonismo debido por una parte a la formalización de sus aspectos matemáticos [23] [24] y por otra al advenimiento de la computación paralela y distribuida [25][26]. La gran mayoría de los trabajos publicados los ubican en el campo de la optimización continua, aunque bien pueden ser usados en la optimización discreta o en la optimización de funciones que no son de naturaleza numérica.

En esta tesis se va a realizar un estudio de los métodos basados en SD (*Sphere Decoding*) [27]. Son métodos que siguen un esquema de Ramificación y Poda, y alcanzan la solución ML al problema (1.2). La idea consiste en intentar buscar sólo en

aquellos puntos del espacio discreto que están dentro de la esfera con centro en el vector dado x y un radio r inicialmente escogido. El objetivo es encontrar todos los vectores $s \in A^m$ que cumplan:

$$||x-Hs||^2 \leq r^2 \quad (1.3)$$

En los últimos años han aparecido un número considerable de investigaciones que tratan de mejorar el rendimiento de los métodos SD optimizando algunas cuestiones que influyen en su rendimiento, como es la selección del radio para acotar la búsqueda [28] [29], y el orden de decodificación de los símbolos [30][31].

El ASD [10] es otro método de decodificación esférica, con la singularidad de que no necesita un radio inicial, en cambio lo que hace es asignarle a cada nodo un peso específico. Este método garantiza que el primer nodo visitado del último nivel, será el nodo que contenga la solución óptima. Se puede encontrar más información sobre este método en [32].

En esta tesis se han empleado algunos de los métodos de selección de radio vistos en [11] (referidos al SD) para aplicarlos al ASD. También se ha hecho uso de la QR con pivotamiento [12] a la matriz de canal, como vamos a ver en el capítulo 4.

En la parte de la paralelización, la programación con GPUs esta teniendo una gran aceptación en el mundo de la investigación, desde su debut en 2007, una gran variedad de industrias y aplicaciones han utilizado CUDA con éxito.

En el mercado de consumo, prácticamente todas las aplicaciones de vídeo se han acelerado a través de CUDA, como demuestran diferentes productos de Elemental Technologies, MotionDSP y LoiLo, etc. Algunos de los casos más reconocidos son:

- Amber, un simulador de dinámica molecular empleado por más de 60.000 investigadores del ámbito académico y farmacéutico de todo el mundo para acelerar el descubrimiento de nuevos medicamentos.
- OptiTex Ltd. Ha cambiado por completo el paradigma del diseño en la industria textil gracias a su tecnología CAD/CAM en 3D. Hasta ahora, los modistos tenían que crear los modelos de sus colecciones con tejidos reales para diseñar los prototipos, un proceso caro y laborioso. OptiTex 3D ha modernizado este proceso simulando la textura y el movimiento de las prendas en modelos virtuales, lo que les permite revisar, ajustar y medir las muestras antes de cortar el primer retal. El tiempo de desarrollo de una colección de temporada suele ser de 190 días, pero con las soluciones optiTex 2D basadas en la GPU, el plazo de salida al mercado se ha reducido a 35 días.
- En el mercado financiero, Numerix y CompatibL introdujeron soporte de CUDA para una nueva aplicación de cálculo de riesgo de contraparte y, como resultado, se ha multiplicado por 18 la velocidad de la aplicación. Cerca de 400 instituciones financieras utilizan Numerix en la actualidad.

Puede verse más información sobre el mundo de las GPUs en la dirección web de la empresa NVIDIA [36].

1.4 Objetivos

Este trabajo tiene como objetivo general diseñar, implementar y evaluar algoritmos secuenciales y paralelos para resolver eficientemente el problema de mínimos cuadrados en espacios discretos, particularmente el problema aplicado a la estimación de la matriz de canal y a la decodificación de señales en sistemas inalámbricos MIMO.

Los objetivos específicos de la tesis son:

- Conseguir un algoritmo eficiente para la estimación de la matriz de canal para un sistema de comunicaciones MIMO en el campo discreto
- Estudiar el problema matemático de mínimos cuadrados desde el punto de vista geométrico: punto más cercano en retículas (CVP).
- Obtener un algoritmo adaptativo basado en el Automatic Sphere Decoder (ASD), que consiga decodificar la señal recibida con el menor número de nodos expandidos posibles.
- Integrar los dos algoritmos anteriores.
- Paralelizar el algoritmo de decodificación utilizando para ello hardware gráfico con arquitectura CUDA.
- Analizar, evaluar y optimizar los algoritmos desarrollados y compararlos con las versiones estándar.

1.5 Herramientas hardware

En este trabajo se han utilizado un multiprocesador de 4 cores, Golub, perteneciente al grupo de computación de altas prestaciones INCO2 del Departamento de Sistemas Informáticos y Computación y de la Universidad Politécnica de Valencia. A continuación se detallan sus principales características sacadas de la página web del grupo [40]:

Golub golub.dsic.upv.es					
Model			**Processor**	**Main Memory**	**Cache**
Fujitsu Siemens CELSIUS R650			Intel Cuad-Core Xeon E5430 2.66 GHz	32 GB	6 MB
GPU					
Main Memory	**Multiprocessors**	**Cores**	**Clock Rate**	**Version**	**Concurrent Copy and Execution**
4 GB	30	240	1.3 GHz	1.3 Version Compliant	Yes
Software					
Intel Compiler 11.1					
MKL 10.1.1.019					
Cuda Version 2.3					
Cula Version 1.0					

Fig. 1.2: Características de la máquina Golub del grupo INCO2

De todas las características (Figura 1.2) las que más nos interesan son las referidas a la tarjeta gráfica. Se trata de una GPU NVIDIA Quadro FX5800 con 30 Multiprocesadores de 8 cores cada uno y una memoria global de 4GB.

Otra máquina utilizada ha sido Micromachin, un Quad-Core Xeon de 2 procesadores con 4 cores cada uno del instituto de Telecomunicaciones y Aplicaciones Multimedia también de la UPV

Micromachin micromachin.iteam.upv.es					
Model	**Processor**	**CPUs**	**Cores**	**Main Memory**	**Cache L3**
	Intel(R) Quad-Core Xeon(R) CPU E5530 2.40GHz Hyper-Threading	2	4	48 GB	8 MB

GPU 4 TESLA C1060					
Main Memory	**Multiprocessors**	**Cores**	**Clock Rate**	**Host page-locked memory mapping**	**Concurrent Copy and Execution**
4 GB	30	240	1.3 GHz	Yes	Yes

GPU GeForce 8400 GS					
Main Memory	**Multiprocessors**	**Cores**	**Clock Rate**	**Host page-locked memory mapping**	**Concurrent Copy and Execution**
512 MB	1	8	1.40 GHz	NO	NO

Software
Intel Compiler 11.1
MKL 10.1.1.019
Cuda Version 2.3
Cuda Toolkit and SDK
Cula Premium 1.1 64 bits

1.3: Características de la máquina Micromachin del grupo INCO2

Al igual que sucedía con Golub, lo que más nos interesa es la tarjeta gráfica, en este caso se trata de una GPU 4 Tesla C1060 también con 240 cores.

Como tercer y último equipo se ha utilizado un Macbook Pro, Intel Core 2 Duo a 2.53 Ghz, con 2 GB de RAM, para las pruebas secuenciales del capítulo 6.

1.6 Herramientas software

Las herramientas software utilizadas son la versión 7.9 (R2009b) de Matlab, y las librerías álgebra lineal numérica, tanto Lapack [41-42] como su versión para GPUs Cula [43]. Proporcionan rutinas para resolver sistemas de ecuaciones, problemas de mínimos cuadrados, problemas de valores propios, vectores propios y valores singulares. También proporciona rutinas para factorización de matrices, como la descomposición QR utilizada en este trabajo.

1.7 Organización de la Tesis

El resto de la Tesis está organizada de la siguiente forma:

En el segundo capítulo se realiza un estudio teórico y experimental de dos algoritmos de estimación de canal basados en señales de entrenamiento como son el método de mínimos cuadrados (LS) y el método de mínimos cuadrados escalado (SLS). Se evalúan los costes computacionales de cada algoritmo.

En el tercer capítulo se presenta el problema de la decodificación de señales, y se realiza un estudio del método heurístico *Zero Forcing* y de los métodos de máxima probabilidad *Sphere Decoding* (SD) y *Automatic Sphere Decoding* (ASD).

En el cuarto capítulo se describe con mayor profundidad el ASD y se apuntan diversas mejoras para reducir el número de nodos expandidos como la factorización QR con pivotamiento, o la utilización de un radio para acotar la búsqueda. Este radio se intenta que sea lo más óptimo posible, y para ello se aplican técnicas el uso combinado del ASD con *Zero Forcing* o la descomposición en valores singulares. Se realizan pruebas experimentales variando el tamaño de la constelación y aumentando el número de antenas.

En el quinto capítulo se realiza la integración de la estimación de la matriz de canal junto con la decodificación de las señales recibidas. Se procede también a demostrar el alto grado de paralelización del algoritmo.

En el sexto capítulo se presenta una descripción de la arquitectura CUDA y un estudio teórico y experimental de la paralelización del algoritmo, comparando los tiempos con los obtenidos en la CPU.

En el séptimo se exponen las conclusiones finales y líneas abiertas a la investigación.

Capítulo 2

Estimación del Canal

En este capítulo se va a realizar un estudio para la obtención de una Matriz de canal estimada, utilizando algunos de los métodos basados en secuencias de entrenamiento descritos en [5], como son el método de mínimos cuadrados (LS) y el de mínimos cuadrados escalado (SLS). La idea temática es disponer de herramientas que permitan integrar en un dispositivo virtual todas las etapas del proceso de detección en un sistema MIMO.

2.1 Presentación del problema

Consideramos un sistema MIMO con t antenas transmisoras y r antenas receptoras. El vector de señales recibidas, complejo, de tamaño *rx1,* puede ser expresado como:

$$x_i = Hp_i + v_i \tag{2.1}$$

donde H, *rxt*, es la matriz de canal, p_i, *tx1*, la señal de entrenamiento transmitida, que se supone conocida, y v_i un vector de ruido blanco (de media cero) de tamaño *rx1*.

Para simular un sistema real, se envían $N \geq t$ señales de entrenamiento $p_1, p_2, \ldots, p_N$ a través del canal. La correspondiente matriz de *rxN* dimensiones, $X ([x_1, x_2, \ldots, x_N])$, de las señales recibidas se puede expresar como:

$$X = HP + V \quad con \quad X \in \mathbb{C}^{rxN} \;,\; H \in \mathbb{C}^{rxt} \;,\; P \in \mathbb{C}^{txN} \;\; y \;\; V \in \mathbb{C}^{rxN} \tag{2.2}$$

donde P, *txN,* es la matriz de entrenamiento $[p_1, p_2, \ldots, p_N]$ y V, $[v_1, v_2, \ldots, v_N]$, es la matriz de ruido, *rxN*.

El objetivo de un algoritmo de estimación del canal es reconstruir la matriz de canal H conociendo el valor de las señales recibidas X y las señales de entrenamiento P.

2.2 Método de mínimos cuadrados (LS)

Podemos estimar la matriz de canal utilizando la aproximación por mínimos cuadrados [33]:

$$\hat{H}_{LS} = X P^{\dagger} \tag{2.3}$$

donde $P^{\dagger} = P^H(PP^H)^{-1}$ es la pseudoinversa de P y $(\cdot)^H$ representa la hermitiana traspuesta.

Se va a utilizar la siguiente restricción de potencia para las señales de entrenamiento transmitidas:

$$\| P \|^2_F = \Psi \tag{2.4}$$

donde Ψ es una constante y $\| \cdot \|_F$ es la norma de Frobenius de la matriz. El objetivo es encontrar una matriz de entrenamiento P óptima que minimice el error de estimación de la matriz de canal con respecto a la restricción de potencia transmitida (2.4). Para ello hay que resolver el siguiente problema de optimización:

$$\min_P E\{\|\hat{H}_{LS} - H\|_F^2\} \text{ sujeto a la restricción } \|P\|_F^2 = \Psi \tag{2.5}$$

Tal como se muestra en [5], la función objetivo de (2.5) podría ser reescrita como:

$$J_{LS} = E\{\|\hat{H}_{LS} - H\|_F^2\} = \sigma_n^2 r\, tr\{(PP^H)^{-1}\} \tag{2.6}$$

donde se ha utilizado el hecho de que $E\{V^H V\} = \sigma_n^2 rI$, siendo σ_n^2 la potencia de ruido recibida, I la matriz identidad y $tr\{\cdot\}$ representa la traza de una matriz.

Finalmente se puede demostrar también [5] que cualquier matriz de entrenamiento P es óptima si cumple la siguiente condición:

$$PP^H = \frac{\Psi}{t} I \tag{2.7}$$

En esta tesis se ha utilizado la matriz de la transformada discreta de Fourier (DFT) normalizada tal como se sugiere en [17] que aparte de cumplir con (2.7) también utiliza una restricción adicional para la potencia de pico transmitida por antena:

$$P=\sqrt{\frac{\Psi}{Nt}}\begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & W_N & \cdots & W_N^{N-1}\\ : & : & & :\\ 1 & W_N^{t-1} & \cdots & W_N^{(t-1)(N-1)}\end{bmatrix} \tag{2.8}$$

con $W_N=e^{j2\pi/N}$

Utilizando esta matriz de entrenamiento óptima, y combinando (2.3) con (2.7), se puede obtener una estimación de la matriz de canal calculando:

$$\hat{H}_{LS}=X\,P^{\dagger}=\frac{t}{\Psi}X\,P^{H} \tag{2.9}$$

con un error de estimación del canal:

$$J_{LS}=\frac{\sigma_n^2 t^2 r}{\Psi} \tag{2.10}$$

Se puede ver que el error es proporcional al cuadrado del número de *t*, por tanto, es interesante limitar el número de antenas transmisoras con respecto al número de antenas receptoras utilizadas.

2.2.1 Algoritmo LS

Una vez hecho el estudio teórico presentamos el pseudocódigo y su implementación en lenguaje MATLAB para la estimación de la matriz de canal a partir de la matriz de entrenamiento óptima y de las señales recibidas. Teniendo en cuenta que se va a trabajar en el campo real, se duplica el tamaño de las matrices, siendo *m=2t, n=2r* y *N≥t.*

La versión de MATLAB utilizada ha sido la 7.9.0.529 (R2009b).

Pseudocódigo:

Entradas: P y X $\quad P\in\mathbb{R}^{mxN} \quad y \quad X\in\mathbb{R}^{nxN}$
Salidas: H_{LS} y J_{LS}

1. Calcular P^T

2. Calcular H_{LS} $\quad \hat{H}_{LS}=\frac{t}{\Psi}X\,P^{T}$

3. Calcular J_{LS} $\quad J_{LS}=\frac{\sigma_n^2 t^2 r}{\Psi}$

2.2.2 Evaluación de costes

El coste computacional del algoritmo vendría condicionada fundamentalmente por la multiplicación de matrices entre la matriz de señales transmitidas y la traspuesta de la matriz de entrenamiento. Si tenemos en cuenta que X es una matriz nxN, y P^T una matriz Nxm, el coste del algoritmo sería:

Coste = 2nmN Flops

Si m=n=N, el coste total es $O(N^3)$.

2.3 Método de mínimos cuadrados escalado (SLS)

A pesar de que el método de mínimos cuadrados (2.3) es una forma aceptable de encontrar una estimación de la matriz de canal, existe la posibilidad de escalar de forma óptima la matriz solución para reducir el error cuadrático medio

$$\hat{H}_{SLS} = \gamma \hat{H}_{LS} \tag{2.11}$$

siendo γ el factor de escalado que hay que calcular.

El error en la estimación del canal utilizando el método de mínimos cuadrados escalado se puede expresar con la siguiente fórmula:

$$E\{\|H - \gamma \hat{H}_{LS}\|_F^2\} \tag{2.12}$$

Siguiendo [5] se puede obtener finalmente el factor de escalado que minimiza el error como:

$$\gamma_o = \frac{tr\{R_H\}}{J_{LS} + tr\{R_H\}} \tag{2.13}$$

donde $R_H = E\{H^H H\}$ es la matriz de correlaciones del canal y J_{LS} el error en la estimación de canal obtenido en (2.6). El error con este factor de escalado óptimo siempre será menor que con el anterior método:

$$J_{SLS} = E\{\|H - \gamma_o \hat{H}_{LS}\|_F^2\} = \frac{J_{LS}\, tr\{R_H\}}{J_{LS} + tr\{R_H\}} < J_{LS} \tag{2.14}$$

Si tenemos en cuenta (2.6) y (2.14), vemos que esta diferencia será especialmente grande cuando el canal sea débil o la potencia transmitida pequeña, es decir, si $tr\{R_H\} << \sigma_n^2 t^2 r / \Psi$.

Suponiendo que el valor de tr{R_H} es conocido de antemano, y utilizando (2.3) (2.6) y (2.13) obtenemos la estimación de la matriz de canal con el método de mínimos cuadrados escalado SLS:

$$\hat{H}_{SLS} = \gamma_o \hat{H}_{LS} = \frac{tr\{R_H\}}{J_{LS} + tr\{R_H\}} XP^{\dagger} = \frac{tr\{R_H\}}{\sigma_n^2 r\, tr\{(PP^H)^{-1}\} + tr\{R_H\}} XP^{\dagger} \qquad (2.15)$$

Al ser J_{LS} el único término que depende de P, vemos que el problema de optimización es equivalente al de la estimación de canal LS. Por tanto, la elección de la matriz de entrenamiento P para el estimador de canal SLS es la misma que para el LS (2.8).

Así, con la matriz de entrenamiento óptima y utilizando (2.10) y (2.14) obtenemos que el error cuadrático medio del estimador SLS es:

$$J_{SLS} = \frac{J_{LS}\, tr\{R_H\}}{J_{LS} + tr\{R_H\}} = \frac{\sigma_n^2 t^2 r\, tr\{R_H\}}{\sigma_n^2 t^2 r + \Psi\, tr\{R_H\}} \qquad (2.16)$$

En (2.16) se aprecia que incrementando el número de antenas transmisoras y/o receptoras el error tiende a tr{R_H}, lo que demuestra que asintóticamente y bajo matriz de entrenamiento óptima, el error en la estimación del canal con SLS sólo depende de la potencia del mismo canal.

2.3.1 Algoritmo SLS

El algoritmo con el método SLS obtiene los valores de la estimación de canal del método LS, y luego aplica el factor de escalado correspondiente. Para el cálculo de tr{R_H} se utilizará el mismo sistema seguido en [5], siendo tr{R_H} =tr{$\hat{H}_{LS}^H \hat{H}_{LS}$}. Hay que tener en cuenta, que al estar en el campo de los reales, la traza resultante habría que dividirla por raíz de 2 (al ser las matrices el doble de grandes).

Pseudocódigo:

Entradas:H_{LS} y J_{LS} $\qquad H_{LS} \in \mathbb{R}^{nxm}$
Salidas: H_{SLS} y J_{SLS}

1. Calcular la tr{R_H}=tr{$\hat{H}_{LS}^T \hat{H}_{LS}$}/sqrt(2)

2. Calcular γ_o:

$$\gamma_o = \frac{tr\{\hat{R}_H\}}{J_{LS} + tr\{\hat{R}_H\}}$$

3. Calcular H_{SLS}:

$$\hat{H}_{SLS} = \gamma_o \hat{H}_{LS}$$

4. Calcular J_{SLS}:

$$J_{SLS} = \frac{J_{LS}\, tr\{\hat{R}_H\}}{J_{LS} + tr\{\hat{R}_H\}}$$

2.3.2 Evaluación de costes

El coste computacional del algoritmo vendría caracterizado de nuevo por la multiplicación matriz-matriz entre la matriz de canal por su traspuesta más la obtención de la traza de dicha matriz resultado. Si $\hat{H}_{LS}^T$ es una matriz nxm, y $\hat{H}_{LS}$ una matriz mxn, el coste del algoritmo sería:

Coste multiplicación matriz-matriz = $2\,nm^2$ Flops
Coste traza = n Flops

Por tanto, si m=n, el coste total es también $O(n^3)$.

2.4 Evaluación experimental

En este apartado se van a comparar los dos métodos mediante pruebas experimentales. Para ello, se ha hecho uso de la versión 7.9.0.529 (R2009b) de MATLAB.

El objetivo de estas pruebas es estimar el ECM (Error Cuadrático Medio) normalizado *J/(tr)* respecto a la relación señal a ruido $SNR = \Psi/\sigma_n^2$.

$$MSE_{LS} = \frac{t^2 r}{SNR} \tag{2.17}$$

$$MSE_{SLS} = \frac{t^2 r\, tr\{R_H\}}{t^2 r + SNR\, tr\{R_H\}} \tag{2.18}$$

Para ello, se genera la matriz de señales de entrenamiento óptima (2.8), tomando $N=t$. Además, la matriz de señales recibidas debe ser conocida, cumpliendo con X = HP + V, por tanto, se generan aleatoriamente la matriz de ruido y lo que sería la matriz de canal real:

```
V=1/sqrt(2)*randn(n,N);

H = 1/sqrt(2)*randn(n,m);
H = sqrt(snr/t)*H;
```

El objetivo de este algoritmo es obtener una matriz lo más parecida posible a esta H generada de forma aleatoria.

Una vez generadas P y X, se procede a estimar la matriz de canal variando la relación señal a ruido (que como se ha comentado vendría a representar el cociente entre la restricción de potencia en la señal de entrenamiento transmitida Ψ y la potencia de ruido en el receptor σ_n^2).

Para esta primera prueba (Fig. 2.1) se ha tomado un valor fijo de antenas receptoras, variando el número de antenas transmisoras.

Como se comentó en el capítulo 2.2, el rendimiento del estimador LS decrece con el número de antenas transmisoras debido a que el error es proporcional al cuadrado de estas.

En la segunda gráfica (Fig. 2.2) se aprecia que para el mismo caso que con el estimador LS, y con niveles bajos de SNR, el algoritmo SLS obtiene menor ECM.

Esto se puede ver mejor en las siguientes gráficas (Fig. 2.3 y 2.4) donde se comparan en una misma gráfica los dos estimadores. En estas gráficas también se puede comprobar que cuanto mayor sea el número de antenas transmisoras, mayor es la diferencia de rendimiento entre los dos algoritmos.

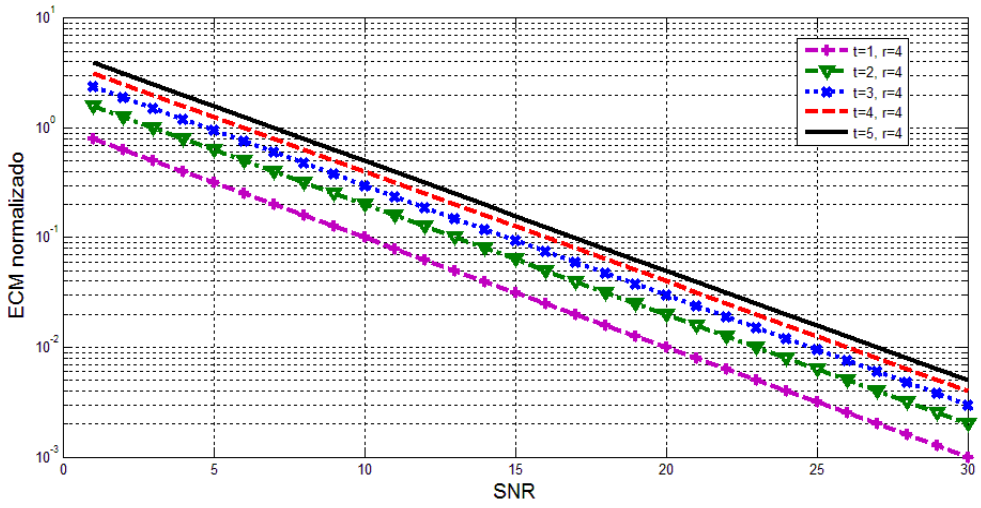


Fig 2.1: ECM de la estimación de la matriz de canal con el algoritmo LS respecto a Ψ/σ_n^2

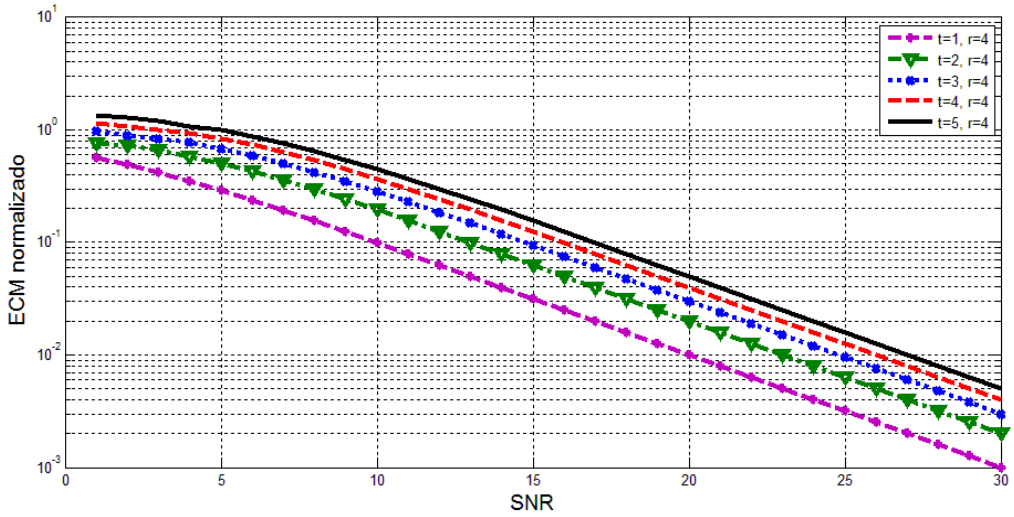


Fig 2.2: ECM de la estimación de la matriz de canal con el algoritmo SLS respecto a Ψ/σ_n^2

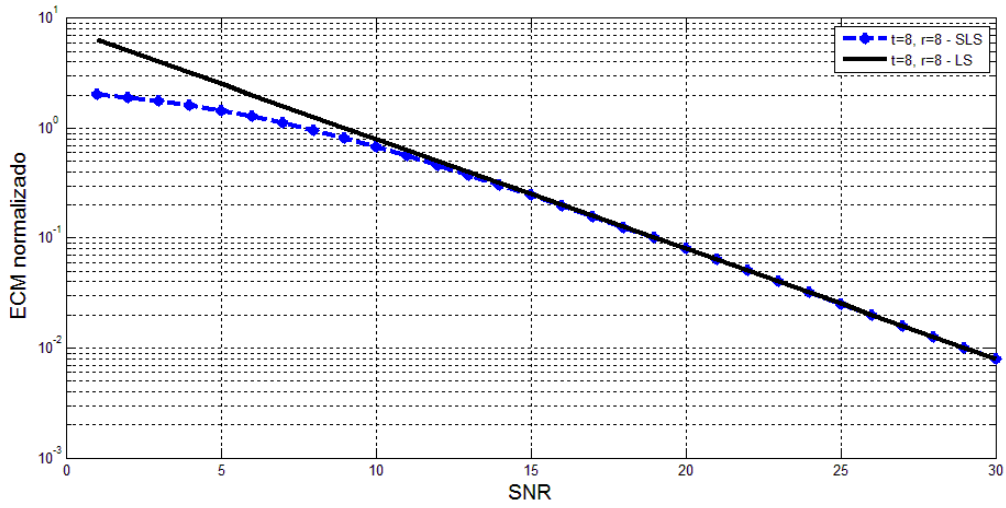


Fig 2.3: ECM para la estimación de la matriz de canal con el algoritmo LS y SLS para 8 antenas transmisoras y 8 antenas receptoras respecto a Ψ/σ_n^2

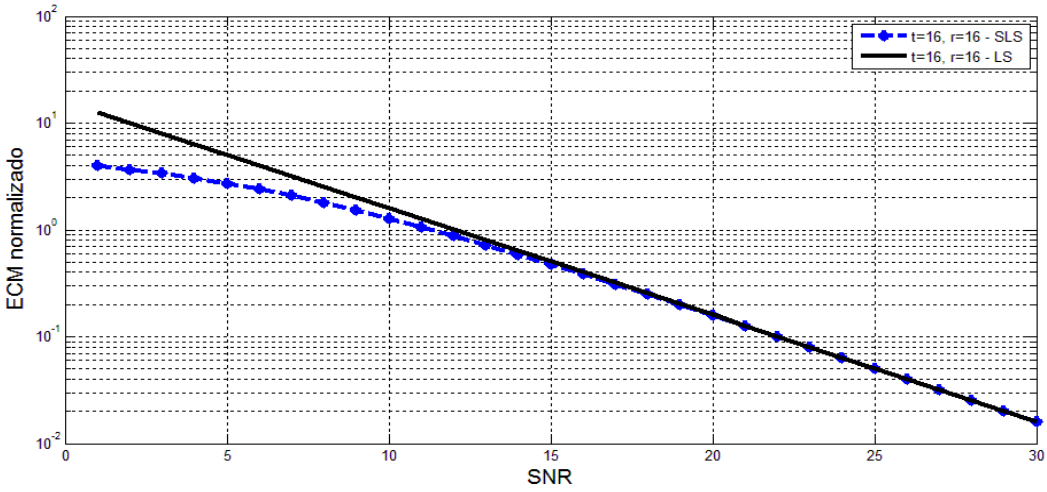


Fig 2.4: ECM para la estimación de la matriz de canal con el algoritmo LS y SLS para 16 antenas transmisoras y 16 antenas receptoras respecto a Ψ/σ_n^2

2.5 Conclusiones

Se puede apreciar que el algoritmo de mínimos cuadrados escalado SLS mejora las prestaciones del algoritmo LS con niveles de relación señal a ruido bajos. El inconveniente que tiene es que requiere que la matriz de correlaciones sea definida de antemano, siendo para ello necesario estimar la matriz de canal con el método LS, para obtener a partir de ésta dicha matriz de correlaciones. De todas formas esto no supone un coste computacional excesivo, por tanto, en la integración de la estimador de la matriz de canal con la decodificación de la señal recibida se va a hacer uso del método SLS. Además, se trata de matrices pequeñas, de 16x16 y 32x32 como máximo, así que el algoritmo es instantáneo.

Capítulo 3

Detección de señales discretas

Una vez realizada la estimación del canal entramos en lo que sería la decodificación de la señal recibida para obtener la señal original. Básicamente se trata de resolver un problema de optimización por mínimos cuadrados en espacios discretos (1.2).

Para ello se va a hacer un profundo estudio de diversos métodos de detección de señales discretas, tanto heurísticos [7] (*Zero Forcing*) como de máxima verosimilitud (*ML, Maximum-Likelihood*), prestando especial atención a los métodos de solución por ramificación y poda (*Branch-and-Bound*)[8] como son el Sphere Decoding (SD) y el Automatic Sphere Decoding(ASD) [10]. Más información puede encontrarse en [11].

3.1 Presentación del problema

Desde el punto de vista matemático, el problema de la decodificación de señales se basa en resolver un problema de optimización por mínimos cuadrados (1.2):

$$\min_{s \in A^m} \quad \|x-Hs\|_2$$

con $x \in \mathbb{R}^{nx1}$, $H \in \mathbb{R}^{nxm}$, $s \in \mathbb{R}^{mx1}$ y $v \in \mathbb{R}^{nx1}$

El problema consiste en encontrar una o varias soluciones, de un conjunto discreto $\mathcal{A}^m$ de posibles soluciones, que minimice la distancia de *Hs* respecto a *x*.

En el contexto de los sistemas discretos, a los posibles valores que puede transmitir la antena se les conoce como constelación de símbolos. En la presente Tesis se ha hecho uso de un tipo de símbolos L-PAM que sigue el siguiente formato:

$$L-PAM = \left\{ \frac{(-L+1)}{2}, \frac{(-L+3)}{2}, \ldots, \frac{(L-1)}{2} \right\} \tag{3.1}$$

Por ejemplo, con L=2:

$$2-PAM = \left\{ \frac{(-2+1)}{2}, \frac{(2-1)}{2} \right\} = \{-0.5, 0.5\}$$

-0.5 y 0.5 serían los posibles valores que se podrían transmitir.

Volviendo a (1.2), la solución mas obvia para obtener la señal transmitida exacta sería resolver dicho problema de optimización realizando una búsqueda exhaustiva por todo el conjunto del espacio discreto $\mathcal{A}^m$ (a este método se le conoce como *Fuerza Bruta*) viendo qué valor es el que minimiza el error. Sin embargo, esto es inviable pues su complejidad es exponencial, L^m. Con L=4 y solo 4 antenas transmisoras (recordemos que m es el doble de antenas transmisoras) ya nos iríamos a 65536 posibles soluciones de s.

En su lugar, se van a utilizar métodos no tan costosos como son el método heurístico "*zero-forcing*", métodos basados en ramificación y poda, o una combinación de los dos como veremos en los siguiente capítulos.

3.2 Métodos heurísticos

Son métodos de solución subóptimos que consiguen una aproximación a la solución del problema de optimización (1.2). En este caso se va a describir el método heurístico "*Zero-Forcing*".

3.2.1 Zero Forcing

El algoritmo *Zero Forcing* lo que hace es resolver el problema de optimización (1.2) como si fuera continuo, para luego aproximar el vector resultado a los valores más próximos de la constelación.

$$\hat{s} = \text{aprox}(H^{-1}x) \tag{3.2}$$

donde aprox(·) es la operación que asocia a cada componente de x el valor de $\mathcal{A}$ más próximo.

Como el coste de calcular la inversa de una matriz es muy grande, primero se aplica la descomposición QR [10] sobre la matriz de canal H, y luego se resuelve como un sistema triangular, aprovechando que las transformaciones ortogonales dejan invariante la 2-norma vectorial:

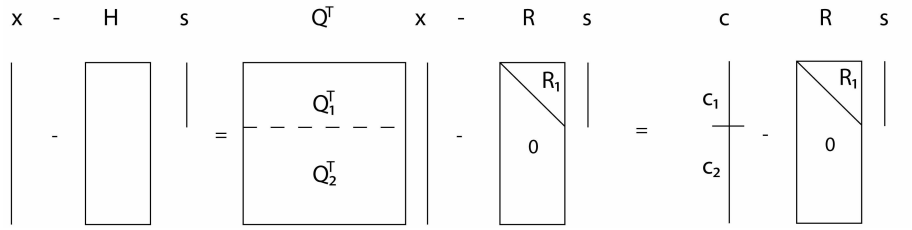
$$||x-Hs||_2 = ||x-QRs||_2 = ||Q^Tx-Q^TQRs||_2 = ||Q^Tx-Rs||_2 \quad = ||c-Rs||_2 \tag{3.3}$$

Por tanto, podemos expresar (1.2) como:

$$\min_{s\in\mathcal{A}^m} ||c-Rs||_2 \tag{3.4}$$

donde $c=Q^Tx$ y $R=\begin{bmatrix} R1 \\ 0 \end{bmatrix}$, $con\ R\in\mathbb{R}^{nxm}$, $c\in\mathbb{R}^{nx1}$ y $s\in\mathbb{R}^{mx1}$

Analizando más detalladamente (3.4) se observa que sólo harían falta los *m* primeros componentes de c:



$$\min_{s \in A^m} \|c - Rs\|_2^2 = \min_{s \in A^m} \left\| \begin{bmatrix} c1 \\ c2 \end{bmatrix} - \begin{bmatrix} R1 \\ 0 \end{bmatrix} s \right\|_2^2 \tag{3.5}$$

y puesto que $\|c\text{-}Rs\|^2 = \|c_1\text{-}R_1s\|^2 + \|c_2\|^2$ (3.6)

finalmente podemos expresar el problema de optimización (3.4) como:

$$\min_{s \in \mathcal{A}^m} \|c_1\text{-}R_1s\|_2 \tag{3.7}$$

$con \quad c_1 \in \mathbb{R}^{mx1}, \quad R_1 \in \mathbb{R}^{mxm} \quad y \quad s \in \mathbb{R}^{mx1}$

Así, podemos replantear (21) como:

$$\hat{s} = \text{aprox}(R_1^{-1} c_1) \tag{3.8}$$

A pesar de ser un método que no siempre encuentra la solución óptima en la mayoría de las ocasiones, resulta muy útil para acortar el rango de búsqueda para las soluciones basadas en ramificación y poda, como se verá más adelante.

3.3 Métodos de máxima verosimilitud (Maximum-Likelihood)

Son métodos que sí encuentran la solución óptima al problema (1.2), aunque su coste es mayor. En este caso nos vamos a centrar en los algoritmos de decodificación esférica (SD – Sphere Decoding)[27], que hacen uso del esquema de ramificación y poda (Branch-and-Bound)[8][34].

Estos métodos son una generalización (o mejora) de la técnica de backtracking (búsqueda atrás - técnica de programación para hacer búsquedas sistemáticas a través de todas las configuraciones posibles dentro de un espacio de búsqueda), y se suelen emplear en problemas de optimización discreta.

Utilizando un esquema de árbol, estos métodos consisten en ir generando ramas de posibles soluciones a partir de otras intermedias. Así, cada nodo del árbol representa una solución parcial al problema, y cada una de las hojas alcanzadas serían soluciones finales.

La característica principal de esta técnica es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas, para "podar" esa rama del árbol y no continuar malgastando recursos y procesos en casos que se alejan de la solución óptima.

Consta de tres etapas:

– Selección: Se encarga de extraer un nodo del árbol.

– Ramificación: Se generan los posibles nodos hijos del nodo seleccionado en la etapa anterior.

– Poda: Se descartan aquellos nodos construidos en la fase anterior que no cumpla cierta condición.

Estas tres etapas se repiten hasta que se encuentre la solución o soluciones deseada(s), o hasta que no queden más nodos que ramificar en la estructura.

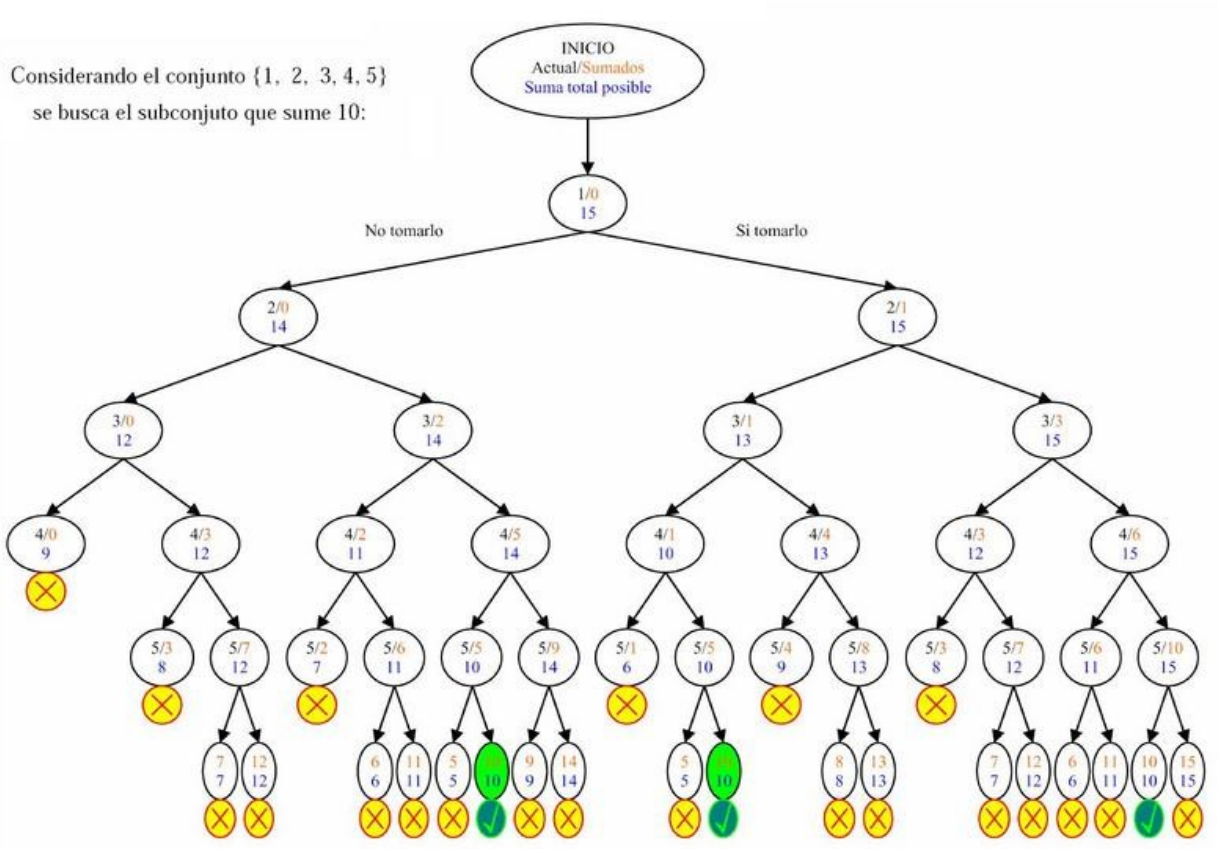


Fig. 3.1: Ejemplo de árbol generado al resolver mediante ramificación y poda un problema de optimización discreta

3.3.1 Sphere Decoding (SD)

Los métodos de decodificación esférica son métodos que siguen un esquema de Ramificación y Poda, y alcanzan la solución de máxima probabilidad al problema. En este caso se ha seguido el mismo procedimiento que en [11], resolviendo el problema desde el punto de vista geométrico, abordando el problema del punto más cercano en retículas, CVP (Closest Vector Problem)[37].

El conjunto de posibles señales discretas (3.1) formarían una retícula, la cual al ser multiplicada por la matriz de canal H quedara deformada, por ejemplo, tal como se muestra en la figura en un caso en que $H \in R^{2x2}$ y L=4:

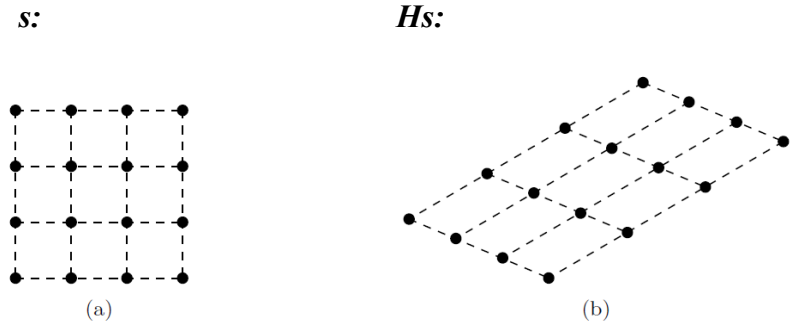


Figura 3.2: (a) Retícula rectangular; (b) Retícula deformada

El objetivo en la detección de señales en sistemas discretos es encontrar el punto de la retícula más cercano a un punto dado. Si tenemos la retícula de posibles soluciones, y consideramos una esfera de centro la señal recibida, x, y radio *r* en la retícula *Hs*, podemos acotar las posibles valores de soluciones, comprobando únicamente soluciones dentro de esa esfera. Obviamente la solución encontrada en el interior de la esfera es también la solución en el espacio completo.

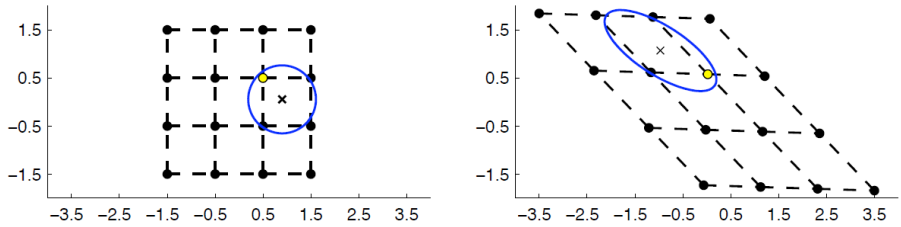


Figura 3.3: Idea del *Sphere Decoding*

Para resolver el sistema discreto (3.7) utilizando el algoritmo de *Sphere Decoding* (SD) lo primero que hay que obtener es una estimación adecuada del radio que hará de límite para saber si hay que "podar" o no dicha ramificación. Este Radio se puede obtener de diversas formas como veremos más adelante, pero en principio nos vamos a centrar en la resolución del problema.

Si particionamos el sistema en bloques e imponemos la condición de búsqueda dentro de la esfera se tiene:

$$\min_{s\in A^m}\|c_1-R_1 s\|_2^2=\min_{s\in A^m}\left\| \begin{bmatrix} c_k^1 \\ c_k^2 \end{bmatrix}-\begin{bmatrix} R_k^{11} & R_k^{12} \\ & R_k^{22} \end{bmatrix}\begin{bmatrix} s_k^1 \\ s_k^2 \end{bmatrix} \right\|_2^2 < r^2 \; con \; 1\leqslant k\leqslant m \qquad (3.9)$$

o bien

$$\|c_1-R_1 s\|_2^2= \left\| \; c_k^1-(R_k^{11} s_k^1+R_k^{12} s_k^2) \; \right\|_2^2 + \left\| \; c_k^2-R_k^{22} s_k^2 \; \right\|_2^2 < r^2 \qquad (3.10)$$

Con ello obtenemos una condición más restrictiva para "podar" la rama en el caso de que las operaciones en dicho bloque superen el radio *r* estipulado por:

$$\left\| \; c_k^2-R_k^{22} s_k^2 \; \right\|_2^2 < r^2 \qquad (3.11)$$

El funcionamiento del algoritmo SD para el caso de la detección de señales discretas es el siguiente:

- En las primeras ramas (tantas como valores posibles en la constelación de símbolos "L"), lo que sería el primer nivel, se asigna un valor perteneciente a la constelación a la última posición del vector de señales transmitidas a detectar, obteniéndose L nodos.
- Se calcula la distancia correspondiente a dicho valor (3.11) entre el vector *x* y el vector más cercano a él en la retícula. En caso de ser mayor que el *r* se "poda".
- Entonces se pasa al siguiente nivel, donde de cada rama no podada salen L ramas más con sus valores correspondientes. Se calcula de nuevo la distancia para cada nodo y se comprueba si la distancia (norma) de cada nodo supera el radio. En caso de ser así se vuelve a "podar". Se pasa al siguiente nivel.
- Este proceso se repite durante tantos niveles cómo tamaño tenga el vector de señales transmitidas, en este caso *m* (3.4), llegando a las hojas del árbol. Se comparan las distancias de los nodos hojas obtenidos y el vector con una distancia menor será la solución al problema.

Puede ocurrir que el Radio elegido fuera demasiado pequeño y acabáramos podando todas las ramas, en este caso la solución es aumentar el Radio y empezar de nuevo el algoritmo. Para que todo quede más claro, veamos un ejemplo. Si tenemos:

$$R_1=\begin{bmatrix}4 & 8 & 6 & 12\\ 0 & 6 & 2 & 4\\ 0 & 0 & 4 & 8\\ 0 & 0 & 0 & 4\end{bmatrix} \qquad c_1=\begin{bmatrix}1\\2\\3\\4\end{bmatrix} \qquad \min_{s\in A^m}\left\|\begin{bmatrix}1\\2\\3\\4\end{bmatrix}-\begin{bmatrix}4 & 8 & 6 & 12\\ 0 & 6 & 2 & 4\\ 0 & 0 & 4 & 8\\ 0 & 0 & 0 & 4\end{bmatrix}\begin{bmatrix}s_1\\s_2\\s_3\\s_4\end{bmatrix}\right\|^2 \tag{3.12}$$

con una constelación de L=2 {-0.5 , 0.5} y tomando Radio=5, se procedería como indica la Figura 3.4, donde dentro del recuadro se sitúa el vector de señales transmitidas, y debajo la distancia con respecto al vector recibido (las operaciones realizadas se muestran más adelante, dividido por niveles).

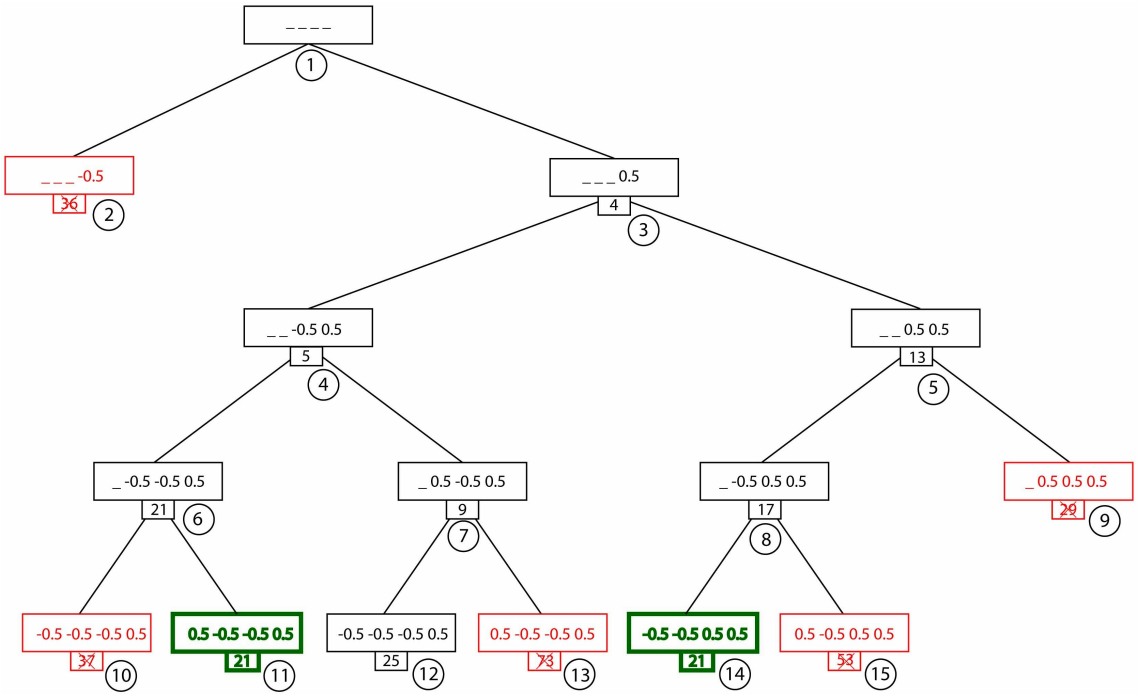


Figura 3.4: Ejemplo del funcionamiento del algoritmo *Sphere Decoding*

Nivel 1:

- $(4 - 4*(-0.5))^2 = 36 > 25$ Poda!!
- $(4 - 4*0.5)^2 = 4 < 25$

Nivel 2:

- $(3 - 4*(-0.5) - 8*0.5)^2 + 4 = 5 < 25$
- $(3 - 4*0.5 - 8*0.5)^2 + 4 = 13 < 25$

Nivel 3:

- $(2 - 6*(-0.5)-2*(-0.5)-4*0.5)^2 + 5 = 21 < 25$
- $(2 - 6*0.5-2*(-0.5)-4*0.5)^2 + 5 = 9 < 25$
- $(2 - 6*(-0.5)-2*0.5-4*0.5)^2 + 13 = 17 < 25$
- $(2 - 6*0.5-2*0.5-4*0.5)^2 + 13 = 29 > 25$ Poda!!

Nivel 4:

- $(1 - 4*(-0.5)-8*(-0.5)-6*(-0.5)-12*0.5)^2 + 21 = 37 > 25$ Poda!!
- $(1 - 4*0.5-8*(-0.5)-6*(-0.5)-12*0.5)^2 + 21 = 21 < 25$
- $(1 - 4*(-0.5)-8*0.5-6*(-0.5)-12*0.5)^2 + 9 = 25 = 25$
- $(1 - 4*0.5-8*0.5-6*(-0.5)-12*0.5)^2 + 9 = 73 > 25$ Poda!!
- $(1 - 4*(-0.5)-8*(-0.5)-6*0.5-12*0.5)^2 + 17 = 21 < 25$
- $(1 - 4*0.5-8*(-0.5)-6*0.5-12*0.5)^2 + 17 = 53 > 25$ Poda!!

De las 3 hojas que quedan se busca la que tenga la menor distancia. En este caso habría 2, por tanto, existen 2 posibles soluciones (los dos están a la misma distancia del punto dado):

$$\hat{s}_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} \qquad \hat{s}_2 = \begin{bmatrix} -0.5 \\ -0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

El siguiente algoritmo resume el proceso de formalmente:

3.3.1.1 Algoritmo Sphere Decoding

Entrada: $R_1 \in \mathbb{R}^{mxm}$, $c_1 \in \mathbb{R}^{mx1}$, L
Salida: $\hat{s} \in \mathbb{R}^{mx1}$

```
A: Matriz; // Aquí se guardarán los nodos expandidos
nodos, padre: Entero;

r = radio_inicial();
A ← agregar(nodo_inicial()) { Adiciona un nodo inicial a la matriz A}
nodo = 0;
mientras (continuar == 1) hacer
        numNodos ← nodos(A);
        para i=1,...,numNodos hacer
                A → extraer_nodo(i){ Se extrae el nodo a expandir }
                para j=1,...,L hacer
                        nodo = nodo + 1;
                        asignar_nivel(nodo);
                        A ← agregar(nodo);
                fin para
         fin para
        numNodos ← nodos(A);
        para i=1,...,numNodos hacer
                dist ← distancia(i);
                si (dist < r)entonces
                        continuar = 1;
                        si (nivel == m) entonces
                                continuar = 0;
                                ŝ ← nodoSolución(A,i)
                                r=dist;
                        fin si
                sino
                        A → elimina_nodo(i);
                        nodo = nodo – 1;
                fin si
        fin para
fin mientras
```

Como se observa en el algoritmo, se va a hacer uso de una matriz para ir almacenando los posibles nodos solución

3.3.2 Automatic Sphere Decoding (ASD)

Como se puede apreciar, la complejidad al utilizar el método *Sphere-Decoding* vendría dada por la elección del radio. Si éste fuera grande el algoritmo sería demasiado costoso y no resultaría eficiente, y si fuera demasiado pequeño y no hubiera puntos dentro de la esfera, el proceso habría de repetirse con un radio mayor, con lo que el coste también sería grande.

El *Automatic Sphere Decoder* (ASD) [10] es una variación del SD que no utiliza un radio inicial. En lugar de realizar un recorrido en profundidad de izquierda a derecha del árbol, lo que se hace es expandir sólo el nodo con la menor distancia con respecto a los valores de la constelación obtenidos en cada nivel más todos aquellos que no se han expandido aún.

El proceso a seguir sería el siguiente:

- En el primer nivel se expanden tantos nodos como valores en la constelación L.
- Se calcula la distancia de cada nodo.
- De todos los nodos expandidos (tanto de este nivel como de los anteriores) se busca cual tiene la menor distancia con los valores de la constelación dados hasta ese momento. Este será el próximo nodo a expandir.
- El proceso se repite hasta que el nodo con la menor distancia sea una de las hojas del árbol.

Si seguimos con el mismo ejemplo que en (3.12):

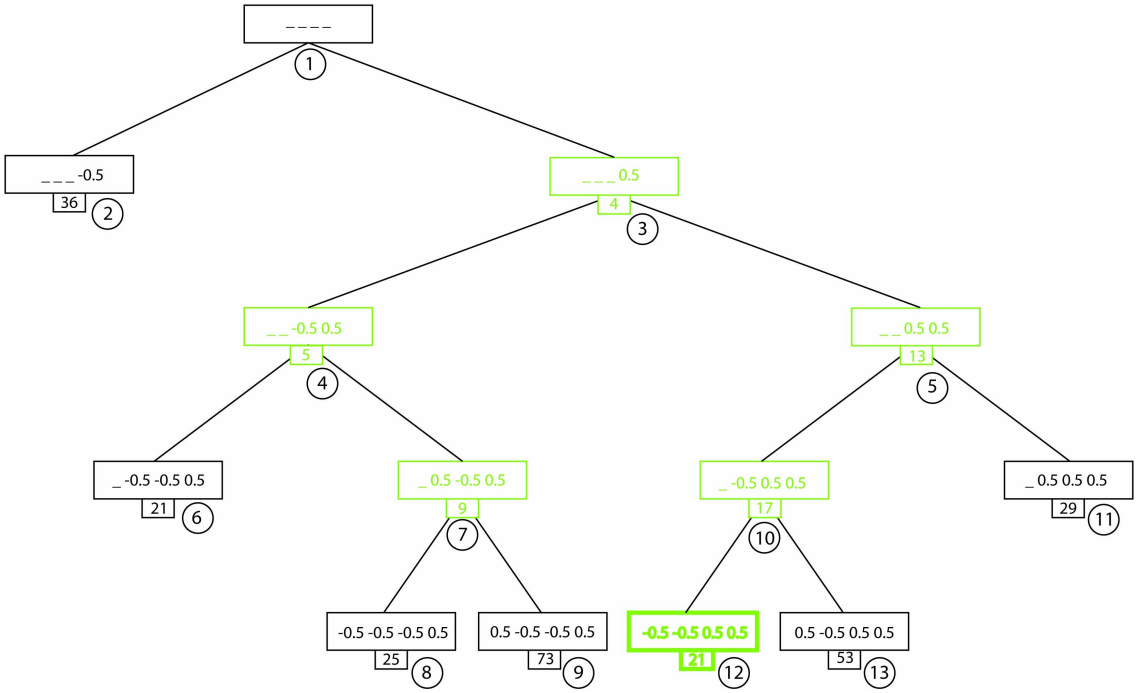


Figura 3.5: Ejemplo del funcionamiento del algoritmo *Automatic Sphere Decoding*

nodos 2 y 3: mín(36,4): 4, expande el 3.
nodos 2, 4 y 5: mín(36,5,13): 5, expande el 4.
nodos 2, 5, 6 y 7: mín(36,13,21,9): 9, expande el 7.
nodos 2, 5, 6, 8 y 9: mín(36,13,21,25,73): 13, expande el 5.
nodos 2, 6, 8, 9, 10 y 11: mín(36,21,25,73,17,29): 17, expande el 10.
nodos 2, 6, 8, 9, 11, 12 y 13: mín(36,21,25,73,29,21,53): 21, los nodos 6 y 12 tienen la menor distancia.

Al ser una hoja quien tiene la mínima distancia entre el vector x y el vector más próximo en la retícula el algoritmo se daría por terminado el algoritmo, siendo la señal detectada:

$$\hat{s} = \begin{bmatrix} -0.5 \\ -0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

La mejora con respecto al algoritmo SD se aprecia viendo el número de nodos visitados por uno y otro método. En este caso y para el ejemplo anterior mientras con el SD se habían visitado 14 nodos, con ASD los nodos visitados se han reducido en 2. Como es lógico, cuanto mayor sea la dificultad del problema a resolver, mayor será la diferencia de rendimiento entre uno y otro algoritmo.

El siguiente algoritmo describe el proceso llevado a cabo:

3.3.2.1 Algoritmo Automatic Sphere Decoding

Entrada: $R_1 \in \mathbb{R}^{mxm}$, $c_1 \in \mathbb{R}^{mx1}$, L
Salida: $\hat{s} \in \mathbb{R}^{mx1}$

```
A: Matriz; // Aquí se guardarán los nodos expandidos
nodos, padre: Entero;

A ← agregar(nodo_inicial()) { Adiciona un nodo inicial a la matriz A}
nodo = 0;
nodoPadre = 1;
mientras (continuar == 1) hacer
        A → extraer(nodoPadre);
        para i=1,...,L hacer
                nodo = nodo + 1;
                asignar_nivel(nodo);
                A ← agregar(nodo);
        fin para
        numNodos ← nodos(A);
        distMax = inf;
        para i=1,...,numNodos hacer
                dist ← distancia(i);
                si (dist < distMax)entonces
                        distMax = dist;
                        nodoPadre = i;
                        continuar = 1;
                        si (nivel == m) entonces
                                continuar = 0;
                                ŝ ← nodoSolución(A,i)
                        fin si
                fin si
        fin para
fin mientras
```

3.4 Conclusiones

En este capítulo hemos visto varios métodos para resolver el problema de mínimos cuadrados discreto (1.2), tanto óptimos como subóptimos. Como una de los principales objetivos de esta tesis son los métodos de decodificación esférica, a continuación se van a presentar los principales ventajas y desventajas de los dos algoritmos vistos en el anterior apartado, el *sphere decoding*, SD, y el *automatic sphere decoding*, ASD.
Para el caso del SD, una de las principales desventajas que encontramos es que depende de la ordenación, si es en profundidad o en anchura. Existen varias soluciones a este problema, como pueden ser las propuestas por Snohrr y Euchner en [35], o por U. Fincke y M. Pohst en [27]. Otro de los principales problemas es que depende del radio inicial, si éste es muy grande el número de nodos a visitar será muy elevado, y si es pequeño y no encuentra ningún nodo, habría que re-elegir de nuevo el radio por tanto el coste también sería grande.

En cuanto al ASD, las mayores ventajas son que no necesita radio [10] y que la ordenación está implícita. Además es sencillo de implementar y garantiza una solución de forma trivial. El gran inconveniente que tiene es que puede necesitar mucha memoria ya que hay que almacenar todos los nodos expandidos en todos los niveles.

El SD está muy estudiado [11,27,32,35,36], sin embargo, el ASD puede mejorarse bastante y hacerse competitivo. En el próximo capítulo se van a ver distintos métodos para mejorar el rendimiento de este algoritmo (ASD).

Capítulo 4

Automatic Sphere Decoder

Como se ha visto en el capítulo anterior, una de las desventajas de utilizar el *Automatic Sphere Decoding* (ASD)[10] es que en cada ramificación se generan tantos nodos como cantidad de símbolos tiene el alfabeto $\mathcal{A}(L)$. Esto puede provocar que el costo de almacenamiento que requiere el algoritmo aumente exponencialmente a medida que aumenten también las dimensiones del problema y el número de símbolos de la constelación. Por tanto, en los próximos apartados se describen diversas técnicas basadas en la elección de un radio inicial [11] que permita hacer podas en la ramificación de los nodos, como sucede en el *Sphere Decoding* (SD), con el objetivo de conseguir que el número de nodos a expandir sea el menor posible. Además, como parte de esta tesis, se propone considerar la descomposición QR con pivotamiento invertido para el pre-procesado de la matriz de canal. Este tipo de técnicas no se han utilizado previamente en el algoritmo ASD.

4.1 Selección de un radio inicial por tomas previas

En el algoritmo mostrado en el capítulo anterior para el ASD, se realizaba una búsqueda de entre todos los nodos expandidos para encontrar el que tuviera la menor distancia. Para ir reduciendo el número de nodos visitados se podría realizar una "*poda*" previa, eligiendo para ello un radio inicial como sucedía con el algoritmo SD. Una de las formas más fáciles de obtener un radio inicial, previo al algoritmo ASD, es calculando la distancia de un número cualquiera de puntos de la retícula elegidos al azar. Se calcula cuál de dichos vectores está más cerca a la señal recibida c_1 en (3.7), y esta distancia será la que utilizaremos como radio.

Se ve claramente que este método para obtener r no es muy eficiente, ya que el éxito del algoritmo depende de si alguno de los puntos de la constelación elegidos de forma aleatoria se encuentra cerca al valor óptimo. En caso de no ser así la mejora sería prácticamente nula.

4.2 Selección del radio inicial basada en el punto de Babai

Otro método más eficiente que el anterior para obtener un radio radio inicial es utilizar la distancia que se obtiene a partir de la señal decodificada utilizando el método *Zero Forcing* (3.8). A este punto de la constelación se le conoce como punto de Babai.

Por tanto, el radio utilizando el punto de Babai sería:

$$r^2 = \|c_1\text{-}R_1 s_{ZF}\|^2 \tag{4.1}$$

Y la condición que deberá cumplir cada nodo expandido del algoritmo es finalmente:

$$\|c_1 - R_1 s\|^2 \leq r^2 \tag{4.2}$$

El radio r se puede cambiar por la distancia correspondiente a las hojas que se vayan encontrando y no sean mínimas en su árbol.

4.3 Descomposición de valores singulares (SVD) en la selección de radios de búsqueda

La descomposición en valores singulares (SVD – *Singular Value Decomposition*) [12] tiene su origen en el intento de los geómetras del siglo XIX por conseguir la reducción de una forma cuadrática a forma diagonal mediante cambios de base ortogonales. Adicionalmente, la SVD provee información acerca de la estructura geométrica de una matriz.

Siguiendo [39] podemos obtener una reducción en la selección del radio de búsqueda utilizando la SVD como se demuestra a continuación.

De (3.9) y (4.2) obtenemos el siguiente sistema particionado en bloques:

$$\|c_1 - R_1 s\|^2 = \left\| \begin{bmatrix} c_k^1 \\ c_k^2 \end{bmatrix} - \begin{bmatrix} R_k^{11} & R_k^{12} \\ & R_k^{22} \end{bmatrix} \begin{bmatrix} s_k^1 \\ s_k^2 \end{bmatrix} \right\|^2 < r^2 \quad con\, 1 \leqslant k \leqslant m \tag{4.3}$$

Se tiene entonces que:

$$\|c_1 - R_1 s\|_2^2 = \left\| c_k^1 - (R_k^{11} s_k^1 + R_k^{12} s_k^2) \right\|_2^2 + \left\| c_k^2 - R_k^{22} s_k^2 \right\|_2^2 < r^2 \tag{4.4}$$

$$\|c_1 - R_1 s\|_2^2 = \gamma_k^2 + \left\| c_k^2 - R_k^{22} s_k^2 \right\|_2^2 < r^2 \tag{4.5}$$

donde

$$\gamma_k^2 = \left\| c_k^1 - R_k^{(11)} s_k^1 - R_k^{(12)} s_k^2 \right\|_2^2 \tag{4.6}$$

Por tanto, de (4.5) obtenemos condición más restrictiva que (4.2):

$$\left\| c_k^2 - R_k^{22} s_k^2 \right\|_2^2 < r^2 - \gamma_k^2 \tag{4.7}$$

con

$$\gamma_k^2 = \left\| c_k^1 - R_k^{(11)} s_k^1 - R_k^{(12)} s_k^2 \right\|_2^2 = \left\| R_k^{(11)} (R_k^{(11)-1} (c_k^1 - R_k^{(12)} s_k^2) - s_k^1) \right\|_2^2 \quad (4.8)$$

Si definimos b_k como

$$b_k = R_k^{(11)-1} (c_k^1 - R_k^{(12)} s_k^2) \quad (4.9)$$

obtenemos:

$$\gamma_k^2 = \left\| R_k^{(11)} (b_k - s_k^1) \right\|_2^2 \quad (4.10)$$

Utilizando la descomposición en valores singulares de $R_K^{(11)}$ podemos obtener una cota inferior de γ_K^2. En efecto, si $\sigma(R_K^{(11)})$ es el menor valor singular de $R_K^{(11)}$ se verifica [12]:

$$\gamma_k^2 = \left\| R_k^{(11)} (b_k - s_k^1) \right\|_2^2 \geq \sigma^2 (R_k^{(11)}) \left\| b_k - s_k^1 \right\|_2^2 \quad (4.11)$$

y puesto que s_K^1 es un subvector de elementos de la constelación en la etapa K, se tiene que

$$\| b_K - Q\{b_K\} \| \leq \| s_K^1 - b_K \| \quad (4.12)$$

donde $Q\{b_k\}$ es un vector con los valores de la constelación de símbolos más próximos respecto al vector b_k, por lo que

$$\gamma_k^2 \geq \sigma^2 (R_k^{(11)}) \left\| b_k - Q\{b_k\} \right\|_2^2 \quad (4.13)$$

Podemos por tanto podar nodos utilizando (4.7) en la forma

$$\left\| c_k^2 - R_k^{22} s_k^2 \right\|_2^2 \leq r^2 - \sigma^2 (R_k^{(11)}) \left\| b_k - Q\{b_k\} \right\|_2^2 \quad (4.14)$$

A pesar de que al utilizar esta variante la mejora producida es importante, también lo es el coste, ya que se tendría que realizar el cálculo de γ para cada nodo del árbol. Sin embargo, como veremos en las pruebas experimentales, este aumento en el coste del algoritmo se contraresta con un número menor de nodos extendidos.

4.4 QR con pivotamiento invertido

Para mejorar aún más el algoritmo se propone realizar la descomposición QR con pivotamiento (QRP)[12] en el pre-procesado de la matriz de canal. En este caso el pivotamiento debe ser invertido, para colocar las columnas de mayor 2-norma de la matriz R al final de esta, ya que son las que se utilizan en los primeros niveles. Así, la distancia en los primeros nodos expandidos es más grande, y las no óptimas son "podadas" antes.

La descomposición QR con pivotamiento genera no solo la matriz unitaria Q y la matriz triangular superior R, sino también una matriz E, llamada de permutación, que cumple que HE = QR. La matriz de Permutación E es elegida de tal forma que los valores de la diagonal de R (en valor absoluto) están ordenados de forma creciente. Así, R_{11} será el menor valor de la diagonal de R en valor absoluto y R_{mm} el mayor. Por tanto, al resolver el sistema en el primer nivel $|| R_{mm} * s_m - c ||$, la distancia obtenida será notablemente mayor que utilizando la descomposición QR sin pivotamiento.

4.5 Pruebas experimentales

A continuación se procede a comparar todos los métodos estudiados para ver cómo responden éstos con diferente razón señal a ruido.

Las pruebas han sido realizadas con constelaciones L-PAM (ver subsección 3.1), tomando valores de L de 2, 4 y 8, y simulando un sistema de comunicaciones MIMO de 4 antenas transmisoras y 4 antenas receptoras.

Las matrices reales H se generaron de forma aleatoria siguiendo una distribución Gaussiana con media cero y varianza 1. En primera instancia se generaron la matrices complejas 4x4 y luego se transformaron al modelo real, tal como se ha descrito en la sub-sección (1.1). A partir de ahora cuando se hable del tamaño del problema se hará referencia al tamaño de las matrices en el campo de los reales, es decir, si fuera 8x8 se trataría de un sistema MIMO de 4 antenas transmisoras y 4 receptoras.

A toda señal transmitida se le ha añadido un ruido gaussiano blanco con media cero y varianza determinada por:

$$\sigma^2 = \frac{m(L_2 - 1)}{SNR} \tag{4.15}$$

donde SNR es la relación señal a ruido en la detección de la señal, y para la cual se han considerado valores de 5, 10, 15, 20, 25 y 30.

En cada caso, se estima la matriz de canal con el algoritmo SLS (2.15) siguiendo el proceso descrito en la evaluación experimental del capítulo 2 (sección 4). Cada prueba se ha repetido para diferentes errores en la estimación del canal:

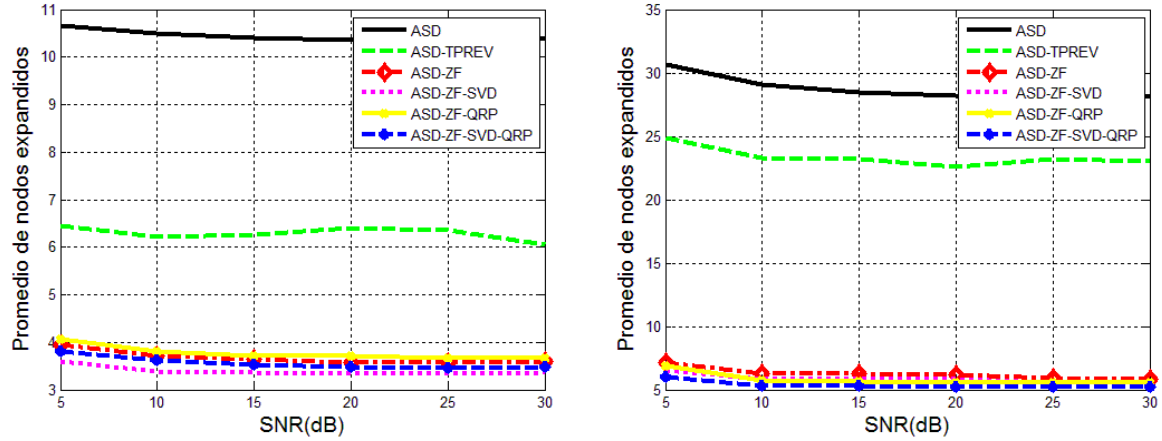
- Mal conocimiento del canal: La Matriz de canal ha sido estimada con una señal a ruido de 5 dB (recordemos que la señal a ruido en la estimación del canal venía dado por Ψ/σ_n^2, fig. 2.3 2.4, no confundir con la SNR en la detección).
- Moderado conocimiento del canal: 15 dB
- Buen conocimiento del canal: 30 dB

Una vez estimada la matriz de canal, se procede a la decodificación de 20 señales utilizando el algoritmo ASD junto con todas las variantes. Esto se repite 10 veces con 10 matrices H distintas, y finalmente se calcula el promedio de nodos visitados durante la búsqueda.

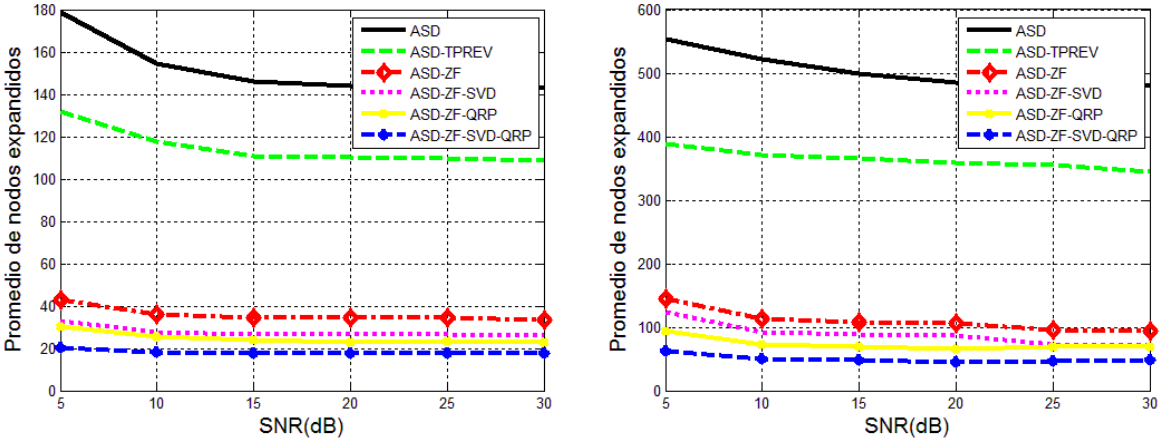
La nomenclatura vista en las gráficas corresponde a:

- ASD: Algoritmo base del *Automatic Sphere Decoding*. No se utiliza Radio inicial.
- ASD-Tprev: Se obtiene un radio inicial a partir de 100 puntos de la constelación elegidos aleatoriamente.
- ASD-ZF: Utiliza el punto de Babai como radio inicial.
- ASD-ZF-SVD: Utiliza el punto de Babai y la descomposición en valores singulares para seleccionar el radio de búsqueda.
- ASD-ZF-QRP: Igual que ASD-ZF incorporando la QR con pivotamiento para el preprocesado de la matriz de canal.
- ASD-ZF-SVD-QRP: Integra todas las mejoras vistas en este capítulo, la elección de un radio inicial a partir del punto de Babai, la reducción de este radio en la búsqueda utilizando la descomposición en valores singulares, y la factorización QR con pivotamiento para el preprocesado H.

En la figura 4.1 se puede ver una comparación tanto del algoritmo base ASD como con cada una de las mejoras estudiadas para un problema 8x8, variando el tamaño de la constelación. Para esta primera prueba se ha supuesto un buen conocimiento del canal.



(a) 2-PAM (b) 4-PAM

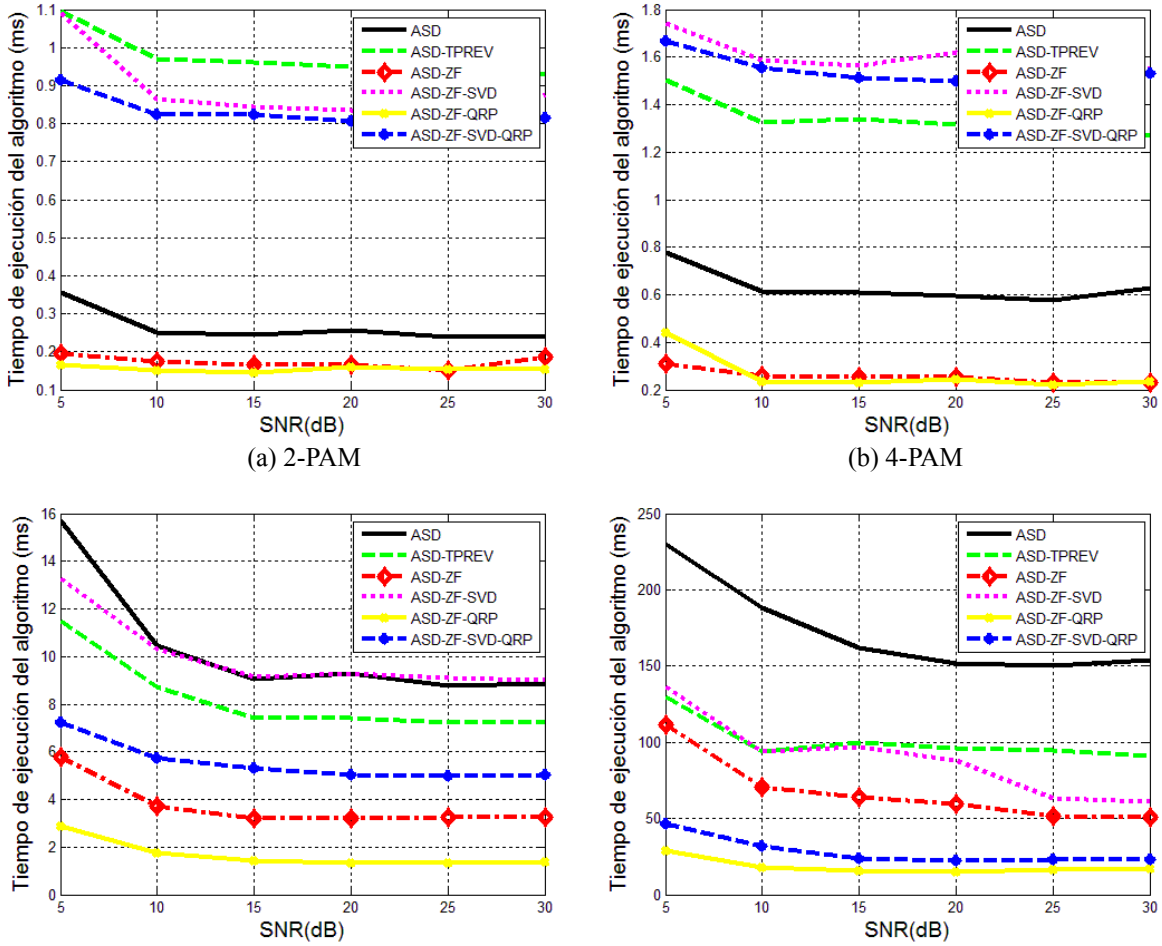


(c) 8-PAM (d) 16-PAM

Figura 4.1: Comparación entre el algoritmo base del ASD junto con todas las mejoras propuestas en problemas 8x8.

Se puede apreciar que la utilización de un radio inicial para acotar la expansión de nodos mejora las prestaciones del ASD, hecho que se ve claramente en (d) donde se pasa de unos 550 nodos con el ASD base hasta menos de 100 utilizando todas las mejoras estudiadas. También nos damos cuenta que con el algoritmo ASD_TPREV la mejora no es muy grande, ya que como se comentó, este método depende que la distancia de alguno de esos puntos de la constelación sea cercano al óptimo.

Además en la figura 4.2 también se puede aprecia que es uno de los métodos que más tiempo utilizan al tener que generar 100 vectores aleatorios de posibles señales transmitidas y calcular su distancia. En cuanto a los demás algoritmos, como también se comentó los que más coste computacional tienen son los dos métodos que realizan la descomposición en valores singulares, aunque queda claramente compensado al ser los que menor número de nodos expande.



(a) 2-PAM (b) 4-PAM

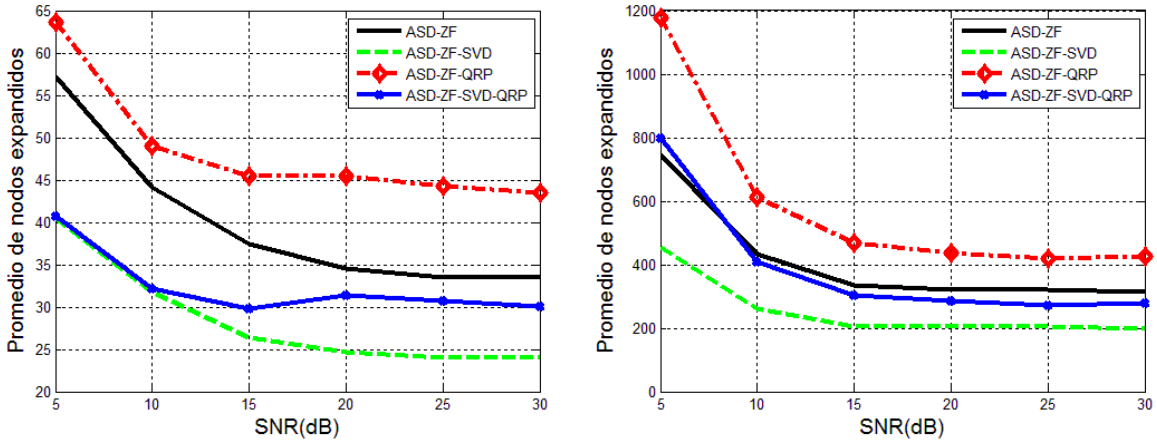
(c) 8-PAM (d) 16-PAM

Figura 4.2: Comparación de tiempos entre los distintos métodos para tamaños de 8x8.

Una vez confirmado que las métodos propuestos mejoran el algoritmo base, nos vamos a centrar ahora en comparar el rendimiento de las distintas mejoras dependiendo del conocimiento de canal que tengamos.

Como podemos ver en la figura 4.3, para un conocimiento pobre del canal se pasa de menos de 100 nodos expandidos en una constelación de 4 símbolos hasta más de 500 en el mejor de los casos con 8-PAM, para valores bajos de SNR.

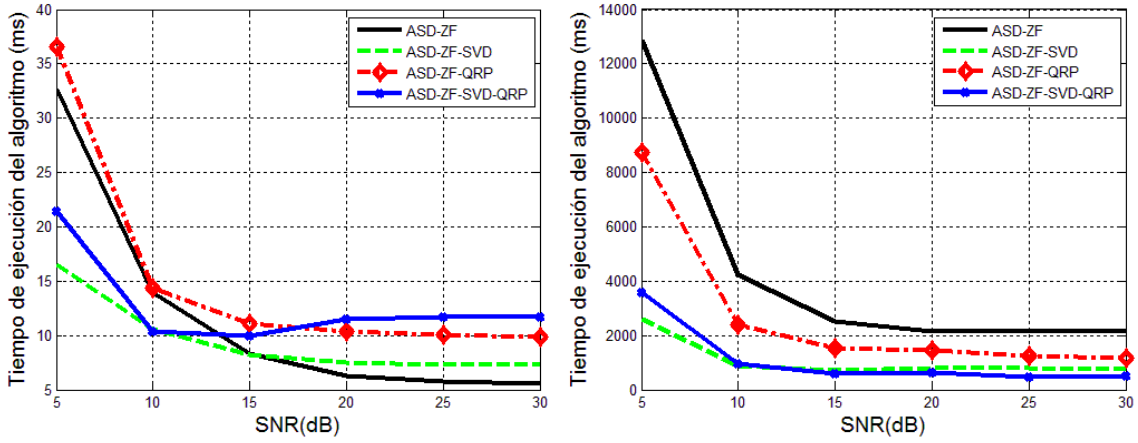
Se puede notar que el algoritmo que mejor responde es la elección del radio haciendo uso de la descomposición SVD, y que el hecho de realizar la QR con pivotamiento no solo no mejora si no que empeora el rendimiento del algoritmo. Se puede decir entonces que la utilización de la descomposición QR con pivotamiento no resulta eficiente con un pobre conocimiento del canal.



(a) 4-PAM (b) 8-PAM

Figura 4.3: Comparación entre las distintas mejoras propuestas en problemas 8x8, variando el tamaño de la constelación.

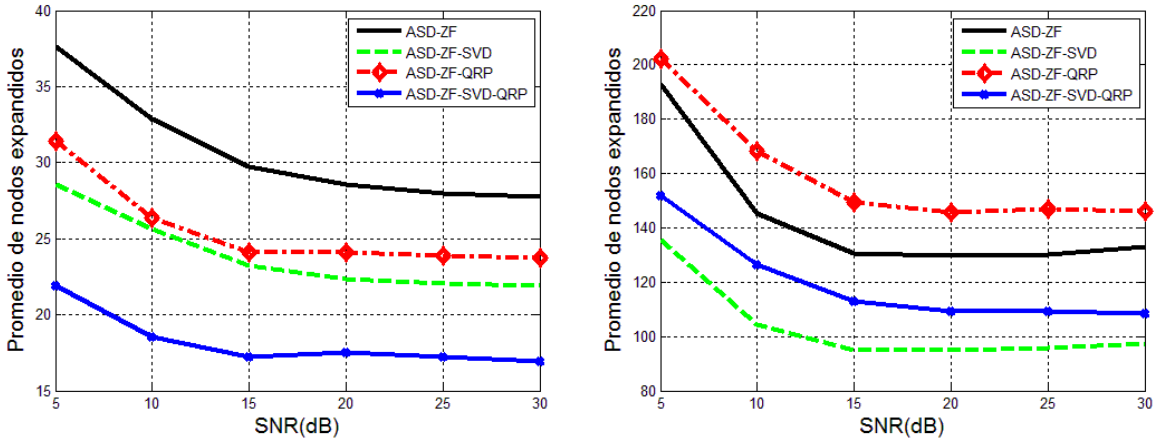
En cuanto a los tiempos (Fig. 4.4) lo más notorio es comprobar que aunque la utilización de la SVD supone un coste adicional al algoritmo por cada nodo, el hecho de expandir bastantes menos que el resto de métodos hace que el tiempo final empleado para encontrar la solución sea el más bajo.



(a) 4-PAM (b) 8-PAM

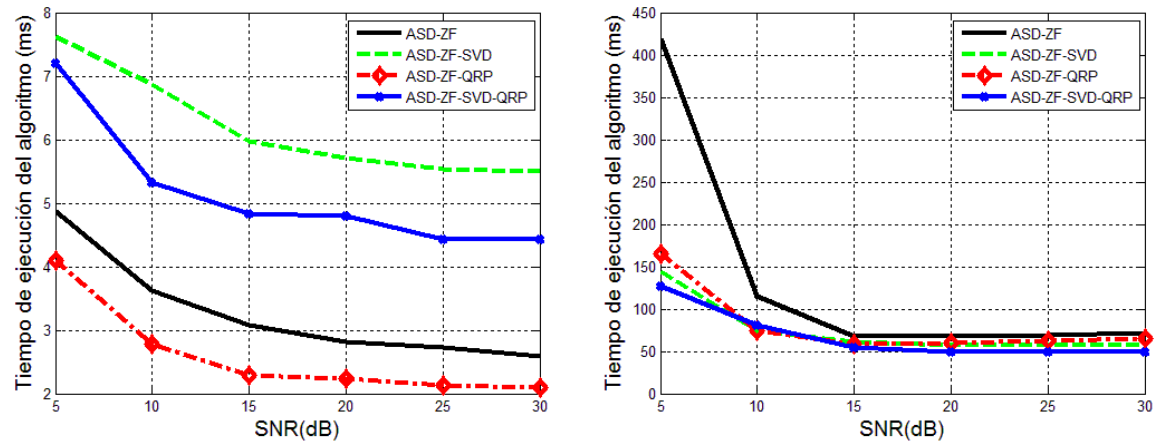
Figura 4.4: Comparación de tiempos en problemas 8x8 variando para valores de constelación 4-PAM y 8-PAM.

Con conocimiento moderado del canal (Figura 4.5 y 4.6) la cantidad de nodos expandidos se reduce notablemente, pero no así el comportamiento de los algoritmos. La mejora SVD sigue siendo el método que mejor responde. Es curioso ver como con una constelación de 8 símbolos, y a partir de una SNR de 10dB, los tiempos de todos los métodos se igualan debido a que, como ya se ha comentado, los métodos más rápidos expanden mayor número de nodos.



(a) 4-PAM (b) 8-PAM

Figura 4.5: Comparación entre las distintas mejoras propuestas en problemas 8x8, variando el tamaño de la constelación.

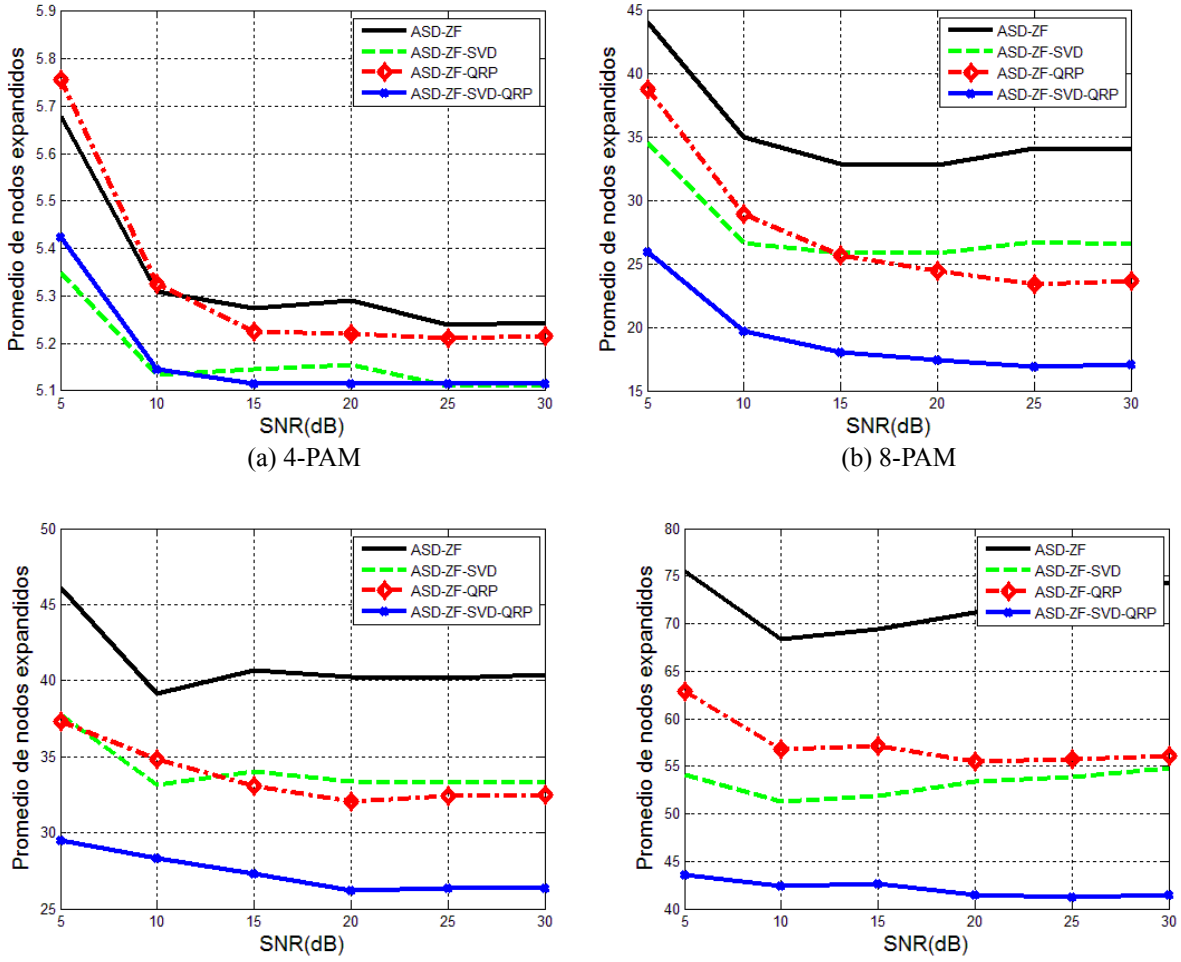


(a) 4-PAM (b) 8-PAM

Figura 4.6: Comparación de tiempos en problemas 8x8 variando para valores de constelación 4-PAM y 8-PAM.

Al realizar las pruebas con un conocimiento alto del canal (Fig. 4.7) se ha creído oportuno aumentar el tamaño de las constelaciones hasta 16-PAM y 32-PAM, y con problemas de 16x16 visto que los nodos expandidos se reducen considerablemente.

Es este contexto es cuando se aprecia la mejora al utilizar la descomposición QR con pivotamiento, que pasa a ser el método que mejor se comporta si lo utilizamos junto con la elección del radio con SVD.

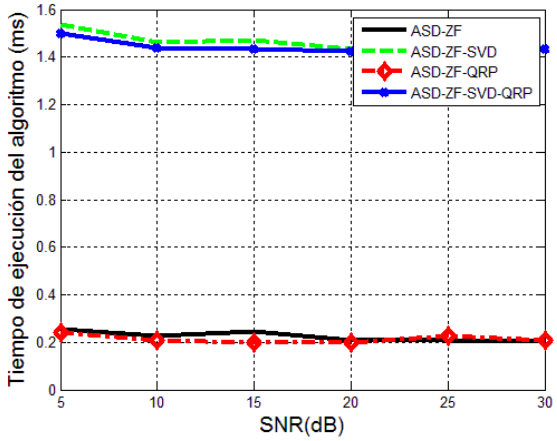


(a) 4-PAM (b) 8-PAM

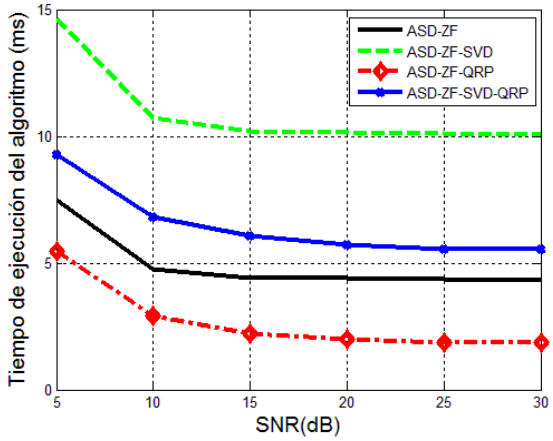
(c) 16-PAM (d) 32-PAM

Figura 4.7: Comparación entre las distintas mejoras propuestas en problemas 8x8, variando el tamaño de la constelación.

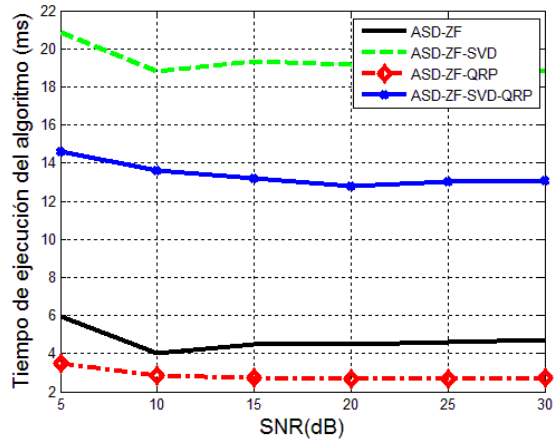
En cuanto a los tiempos de la figura 4.8, se vuelve a ver una correspondencia con respecto al número de nodos expandidos, por eso cuando el tamaño de la constelación aumenta el peso computacional recae más sobre el número de nodos expandidos que sobre el algoritmo en sí. Esto se aprecia claramente en la figura 4.9, para un problema de 16x16.



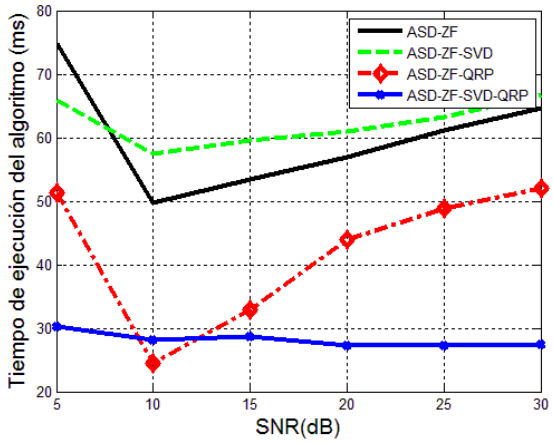
(a) 4-PAM



(b) 8-PAM

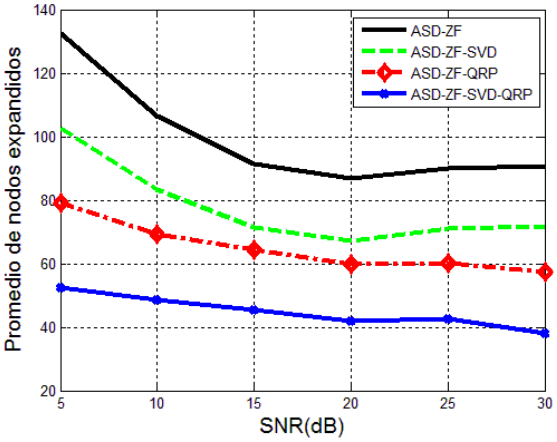


(c) 16-PAM

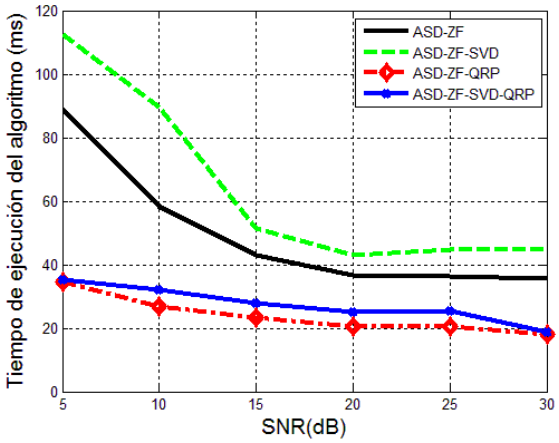


(c) 32-PAM

Figura 4.8: Comparación de tiempos en problemas 8x8 variando para valores de constelación 4-PAM y 8-PAM.



(a)



(b)

Figura 4.9: Nodos expandidos(a) y tiempo utilizado con un tamaño de problema de 16x16 utilizando una constelación 4-PAM

4.6 Conclusiones

Como conclusión, creemos que los experimentos han sido satisfactorios, demostrándose que el hecho de utilizar un radio para acotar los nodos expandidos en el algoritmo ASD provoca una mejora del rendimiento notable. Además se ha comprobado que la utilización de la SVD para la elección del radio no supone un coste adicional, si no todo lo contrario, ya que al expandir una cantidad de nodos inferior, el tiempo de ejecución empleado para la decodificación de la señal se reduce, pudiendo incluso llegar a ser más rápido si el tamaño de problema y/o de constelación aumenta.

En cuanto a la propuesta de utilización de la descomposición QR con pivotamiento para el pre-procesado de la matriz de canal, se demuestra que para que sea eficiente se debe tener un alto grado de conocimiento del canal, ya que si no, no sólo puede que no exista mejora, si no que probablemente la decodificación empeorará. Ahora bien, si se da este caso, buen conocimiento del canal, es con mucho el método más eficiente.

Capítulo 5

Integración

En este capítulo se va a proceder a la integración de las dos partes estudiadas hasta ahora, la estimación del canal y la decodificación de la señal. La idea es obtener un dispositivo virtual que simule el funcionamiento de un sistema de comunicaciones inalámbrico MIMO completo.

5.1 Descripción del problema

Los métodos descritos en el capítulo 2 para la estimación de la matriz de canal hacen uso de unas señales piloto o de entrenamiento, $P\,[p_1, p_2, ..., p_N]$, con las que se obtienen unas señales recibidas, $X\,, [x_1, x_2, ..., x_N\,]$. Recordando (2.2):

$$X = HP + V \quad con \quad X \in \mathbb{C}^{rxN} \;,\; H \in \mathbb{C}^{rxt} \;,\; P \in \mathbb{C}^{txN} \;\; y \;\; V \in \mathbb{C}^{rxN}$$

Básicamente lo que se hace es enviar por las antenas transmisoras N señales de entrenamiento conocidas de antemano a través del canal. Así, con estas señales y las obtenidas por las antenas receptoras, se realiza la estimación del canal con cualquiera de los métodos vistos en el capítulo 2, LS y SLS.
En cuanto a la parte de la decodificación de señales discretas, como se ha visto en el capítulo 3, se trata de resolver un problema de optimización por mínimos cuadrados (3.7):

$$\min \|c_1 - R_1 s\|_2$$
$$s \in \mathcal{A}^m$$

$$con \quad c_1 \in \mathbb{R}^{mx1} \;,\; R_1 \in \mathbb{R}^{mxm} \;\; y \;\; s \in \mathbb{R}^{mx1}$$

En la figura 5.1 se puede ver un diagrama de bloques donde se mostraría todo el proceso integrado, utilizando para la decodificación el método heurístico Zero – Forcing visto en el capítulo 3. La nomenclatura utilizada es la siguiente:

A.S. → Analizador de señales
T. → Temporizador
G.S.P. → Generador de señales de prueba
E.C. → Estimador de canal
P.C.QR → Pre-procesado de la matriz de canal
B.S.E. → Buffer de señales de entrada

D.S.Z.F → Decodificador de señales con Zero Forcing
B.S.S → Buffer de señales de salida

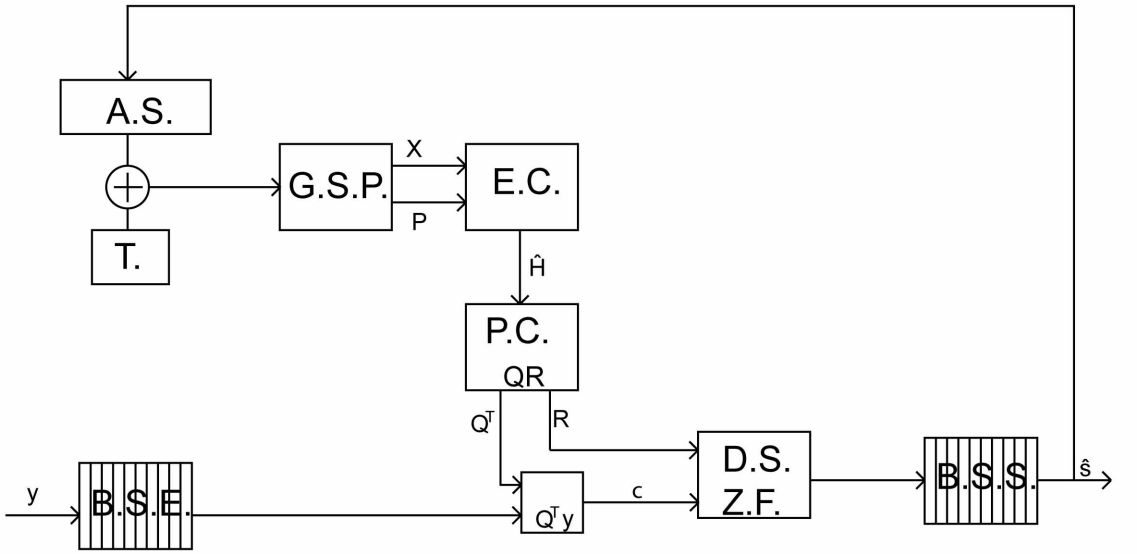


Fig. 5.1: Diagrama de bloques de un sistema decodificador de señales completo utilizando el algoritmo Zero Forcing

El proceso es el siguiente, se estima la matriz de canal generando una señales de prueba como se describe en el capítulo 2. Conocidas las señales de entrenamiento P y las señales recibidas X el estimador del canal E.C. obtiene una matriz H que será la que se utilice para decodificar las señales. A esta matriz hay que aplicarle un procesado previo (3.3), se le realiza la descomposición QR. Con dicha Q y las señales a decodificar (en este caso se ha creído conveniente cambiar la letra asignada a y, para no crear confusión con las señales X utilizadas para la estimación del canal) se obtiene c, que junto con la matriz triangular R son las entradas del decodificador de señales, en este caso el Zero – Forcing.

En principio se puede pensar que la matriz de canal sólo habría que estimarla al principio de la comunicación, pero suelen surgir imprevistos que hacen que el canal cambie, por tanto lo que se hace es re-estimar la matriz de canal cada cierto tiempo. Esto se puede ver también en el diagrama (Figura 5.1) donde existe un analizador de señales que continuamente esta monitoreando las señales decodificadas para ver si perdieran calidad. Aparte, existe un temporizador que cada cierto tiempo obliga a re-estimar el canal, para así estar seguro que la calidad del canal siempre será buena.

El diagrama 5.2 realiza exactamente el mismo proceso, pero esta vez utilizando el algoritmo ASD junto con todas las mejoras propuestas en esta Tesis, tanto la utilización de un radio inicial obtenido mediante el punto de Babai, como la reducción de ese radio con la SVD. Además, al estar seguros que la estimación del canal siempre será buena, se va a hacer uso de la descomposición QR con pivotamiento para mejorar el procesado de la matriz de canal.

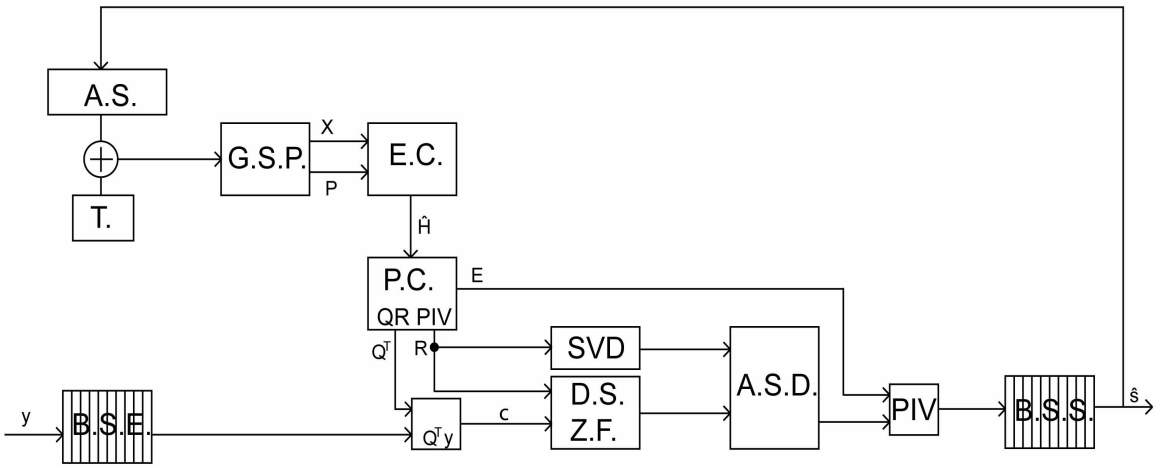


Fig. 5.2: Diagrama de bloques de un sistema decodificador de señales completo utilizando el algoritmo ASD con todas las mejoras implementadas

Por tanto, una vez estimada la matriz H, se calcula la descomposición QRP, y el algoritmo ASD toma los valores singulares obtenidos de la matriz R y la señal decodificada por el algoritmo Zero-Forcing. Internamente calcula el Punto de Babai y decodifica la señal, pero antes de almacenarla en el buffer de salida se ordena de nuevo.

5.2 Análisis

Una vez obtenido un dispositivo virtual que integre las dos partes en las que consiste este trabajo, se procede a su paralelización. El hecho de ir almacenando las señales a decodificar en un buffer, para después procesarlas una a una, sin dependencias entre ellas, nos demuestra el alto grado de paralelización que tiene el sistema.

Como se muestra en la figura 5.3, lo que queremos conseguir es un dispositivo virtual paralelo, que realice el mismo proceso que en la figura 5.2, decodificando las señales, pero en este caso de forma totalmente concurrente.

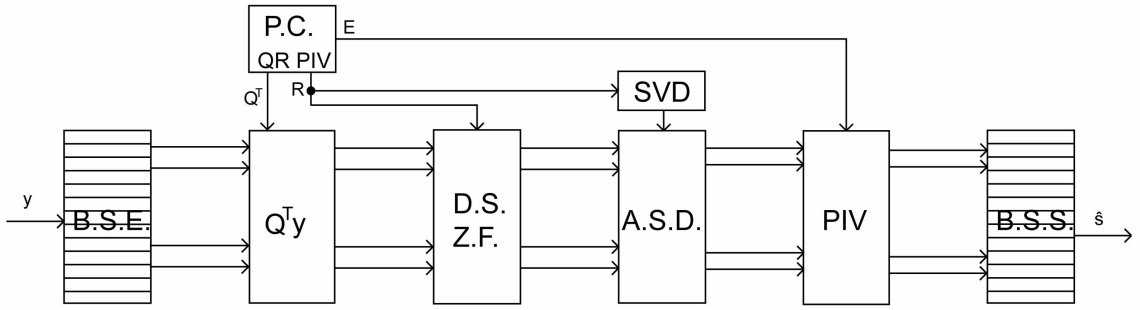


Fig. 5.3: Diagrama de bloques de un sistema decodificador de señales paralelo

El proceso a seguir es el siguiente:

1. Si es la primera vez, se obtiene la matriz de canal H con el estimador.
2. Se le aplica el pre-procesado a la matriz de canal estimada, que no es más que la descomposición QR con pivotamiento.
3. Mientras, el buffer de entrada va almacenando señales a decodificar $y_{1..N}$.
4. Cuando se llena, se multiplica cada cada señal por la matriz ortogonal transpuesta Q^T obtenida de la descomposición QR de H.
5. El siguiente paso es el decodificador utilizando el algoritmo Zero Forcing, todas las señales son decodificadas concurrentemente.
6. Una vez obtenidas las señales decodificadas con el método heurístico, se procede a decodificar con el algoritmo ASD, quien antes de nada calcula internamente el punto de Babai de cada una de las señales. A este decodificador le llegan los valores singulares obtenidos previamente de R, la matriz triangular superior obtenida en el pre-procesado de H, que se utiliza para acotar más el radio de búsqueda y así reducir el número de nodos extendidos.
7. Una vez terminada la decodificación se procede a la ordenación de las señales, orden que marca el vector E, obtenido en el pre-procesado.
8. Y por último llegan se almacenan todas las señales en el buffer de salida

Este proceso se repite continuamente, re-estimando la matriz de canal cuando el temporizador lo marque o cuando el analizador de señales crea que es oportuno.

Este es el esquema que se va a seguir en el próximo capítulo para la paralelización del algoritmo en GPUs de NVIDIA.

Capítulo 6

ASD Paralelo

Como se vio en el capítulo 4, la dificultad del algoritmo ASD se centraba en la extensión de la matriz de posibles soluciones, por tanto se probaron distintos métodos para que ésta fuera mínima. El método que mejor respondió fue el ASD-ZF-SVD, que utiliza el punto de Babai y la descomposición en valores singulares para seleccionar el radio de búsqueda, junto con la QR con pivotamiento, que mejora notablemente los resultados en el caso de tener un buen conocimiento del canal.

En este capítulo lo que se describe es el procedimiento seguido para la implementación paralela de estos dos métodos, realizando un estudio experimental y comparando los tiempos obtenidos en secuencial con los tiempos obtenidos utilizando una tarjeta gráfica con CUDA (figura 6.1), una arquitectura de cálculo paralelo de NVIDIA que aprovecha la potencia de la GPU (unidad de procesamiento gráfico).

Fig. 6.1 – Tarjeta gráfica NVIDIA TESLA con arquitectura CUDA

6.1 Objetivos de la paralelización y motivaciones

En el capítulo 4 se estudiaron y compararon diferentes métodos para mejorar el rendimiento del Automatic Sphere Decoder, siendo el método de valores singulares el que mejor rendimiento obtenía. En este capítulo se describe la versión paralela de este método. Para ello se ha seguido el esquema presentado en la figura 5.3, para decodificar las N señales aprovechando el alto grado de paralelización que ofrece el hardware gráfico con arquitectura CUDA (figura 6.1).

6.2 Introducción a CUDA

Debido a la gran demanda de gráficos 3D que existe hoy en día, las GPUs han evolucionado a unidades de proceso multicores, con soporte multithread, una enorme potencia de cálculo y un elevadísimo ancho de banda (figura 6.2).

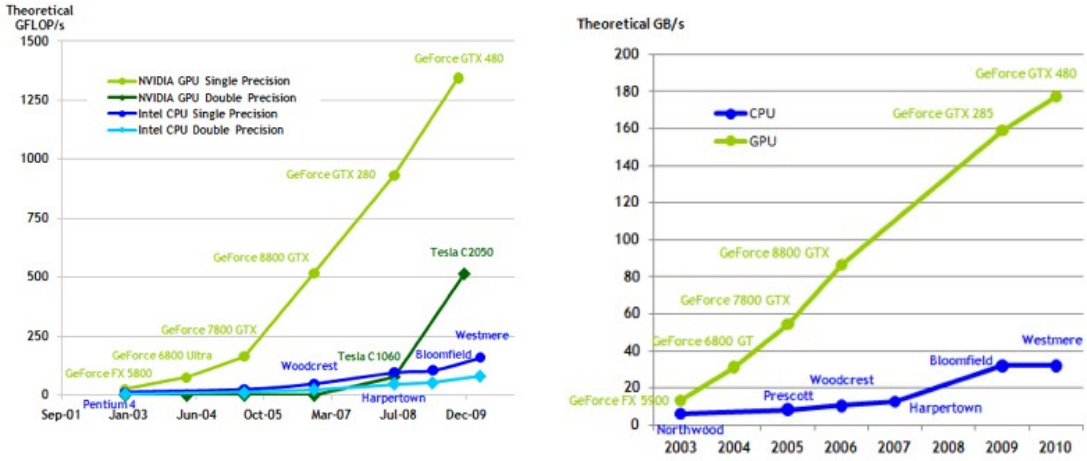


Fig. 6.2 – Diferencia CPU vs GPU: Operaciones en coma flotante por segundo y ancho de banda.

La GPU esta especialmente indicada para problemas SIMD (Single instrucción Multiple Data). Consisten en instrucciones que aplican una misma operación sobre un conjunto más o menos grande de datos. Lo ideal es que cada *thread* realice la misma instrucción sobre un conjunto diferente de datos. Como el mismo programa es ejecutado para cada elemento de datos, no se necesita de un sofisticado control de flujo como ocurre en la CPU.

Fig. 6.3 – La GPU dedica más transistores al procesado de datos

La GPU esta especializada en computación intensiva altamente paralela (exactamente de lo que se trata la representación de gráficos), por tanto, esta diseñada de forma que más transistores son destinados al procesamiento de datos que al almacenamiento de caché en datos y control de flujos, como muestra la Figura 6.3.

Básicamente CUDA es una modificación del lenguaje C para hacer más sencilla la programación en entornos paralelos ejecutados en tarjetas GPU. El funcionamiento es el siguiente: se divide el problema en sub-problemas que pueden ser resueltos independientemente en paralelo por bloques de threads.

Un programa se ejecuta en la CPU (a partir de ahora la llamaremos "host") hasta llegar a uno de los bloques a paralelizar, llamados "kernels" (como se ha comentado antes, se trata de diferentes datos que realizan una misma operación), los cuales se ejecutarán en la GPU, en adelante "device" o dispositivo.

Los bloques se organizan en forma de una malla denominada grid, dentro del cual hay bloques de threads como se ve en la figura 6.4. En este momento puede haber hasta 1024 threads por bloque.

Para poder dividir el trabajo a realizar por cada hilo, existe un identificador único para cada thread de un bloque a través de la variable threadIdx. Lo mismo sucede con los bloques con blockIdx. Otras variables importantes blockDim y gridDim, para obtener el tamaño del bloque y del grid respectivamente. Por tanto, cada thread de un grid puede ser localizable de forma sencilla utilizando la siguiente ecuación:

$$threadIdx + blockIdx*blockDim$$

Se puede encontrar más información sobre CUDA en [36][38].

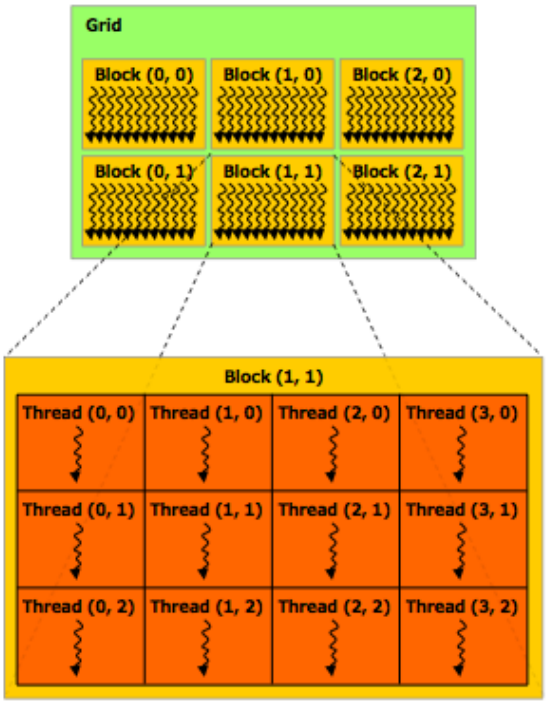


Fig. 6.4 – Grid de bloques de threads.

6.3 Desarrollo de los algoritmos secuenciales

Antes de proceder con los algoritmos paralelos, se han tenido que pasar los algoritmos ya implementados en Matlab a C. El algoritmo en sí consta de las siguientes fases:

- **Estimación del canal**

En la primera fase se estima el canal a partir de unas señales de entrenamiento (2,8) utilizando cualquiera de los método de mínimos cuadrados vistos en el capítulo 2 (ambos algoritmos están implementados, aunque para las pruebas experimentales se ha utilizado el método de mínimos cuadrados escalado). Tanto la matriz H generada de forma aletaoria, como la estimada, se guardan en un fichero para así poder ser utilizadas ambas para la decodificación de las señales.

Este algoritmo es instantáneo debido a que las pruebas se han realizado sobre problemas 8x8 y 16x16, siendo el coste del algoritmo de $O(n^3)$ como se vio en el apartado 2.3.2.

- **Descomposición QR**

Se descompone la matriz de canal estimada H utilizando la función sgeqrf de la librería Lapack.

sgeqrf_(&n,&m,QR,&LDA,TAU,WORK,&LWORK,&INFO);

En la paralelización se utilizó la función culaSgeqrf de la librería cula para calcular la QR, pero no resultaba eficiente ya que se perdía más tiempo en la inicialización de la librería que en el cálculo de la QR con lapack, que es instantánea (la matriz a descomponer como ya hemos comentado es muy pequeña).

Para calcular la QR con pivotamiento se ha utilizado la función sgeqp3, también de lapack.

sgeqp3_(&n,&m,A,&LDA,piv,TAU,WORK1,&LWORK,&INFO);

Donde *piv* es el vector que guarda el orden que deben seguir las columnas de la matriz, por tanto, al finalizar el algoritmo en la GPU hay que ordenar la matriz de nuevo.

- **Obtención de $C = Q^T R$**

También se ha utilizado la librería lapack, en concreto la función sormqr.

sormqr_(&SIDE,&TRANS,&n,&N,&m,QR,&LDA,TAU,C,&LDC,WORK,&LWORK,&INFO);

Al igual que como pasaba con la descomposición QR, también se probó la correspondiente función en cula, culaSormqr, pero como ya se ha comentado se desechó la idea por ser ineficiente.

6.4 Desarrollo del los algoritmos paralelo

Una vez obtenidos los algoritmos en C, el siguiente paso era comprobar qué funciones eran aptas para su paralelización en CUDA. Viendo el esquema 5.3 se puede ver que al almacenar en un buffer una serie de señales a decodificar, cada una independiente de las otras, el problema de la decodificación de dicha matriz se convierte en un problema SIMD (single instruction multiple data), ideal para la paralelización en GPUs. Es decir, cada thread se encargará de decodificar una señal del buffer.

Se han elegido 128 threads por bloque, y tantos bloques como se necesiten dependiendo del número de señales a decodificar:

int blockSize=128; // threads por bloque
int gridSize = (N+blockSize-1)/blockSize;

Debido a que los la matriz triangular R necesita ser accedida por todos los threads prácticamente al mismo instante de tiempo, se almacenó en la memoria constante de la GPU, al ser de sólo lectura y tener un acceso a la memoria más rápido.

Los 3 Kernels ejecutados en la GPU son:

- **Zero Forcing**

```
__global__ void zeroForcing_gpu(float *C, int m, int n, int N, int L, float *Sdec){

        int i,j;
        int tid = threadIdx.x+blockIdx.x*blockDim.x;
        int tb = threadIdx.x;

        __shared__ float sr[512];

        while (tid < N)    {
                for(i=m-1;i>-1;i--){
                        sr[tb]=C[tid*n+i]/Rconst[i*n+i];
                        aproximar_gpu(sr[tb],L);
                        for (j=0; j<i; j++)
                                C[tid*n+j]=C[tid*n+j]-Rconst[i*n+j]*Sdec[i*N+tid];
                }
                tid+=blockDim.x * gridDim.x;
        }
}
```

- **Punto de Babai**

```
__global__ void norma2_gpu (float *S, float *C, int m, int n, int N, float *babai)
{
        int i,j;
        int tid = threadIdx.x+blockIdx.x*blockDim.x;

        while (tid<N){
                babai[tid]=0.0f;
                float mul;
                for (i=0;i<m;i++){
                        mul=0.0f;
                        for (j=i;j<m;j++)
                                mul=mul+Rconst[j*n+i]*S[j*N+tid];
                        mul=mul-C[tid*n+i];
                        babai[tid]=babai[tid]+mul*mul;
                }//for
                tid+=blockDim.x * gridDim.x;
```

```
        }//while
        return;
}//norma2
```

- **ASD con método SVD**

Al tratarse de un algoritmo muy extenso, se encuentra completo en el Anexo I.

Este es el algoritmo más complejo y que más tarda en ejecutarse, debido sobre todo a que la matriz donde se almacenan los nodos extendido puede llegar a ser muy grande. Hay que tener en cuenta que debe almacenar los posibles nodo solución de cada una de las señales a decodificar. El procedimiento es el siguiente, se reserva un array de floats en la memoria global de 4 GB de la máquina (en este caso micromachin) con un tamaño dependiente del problema (ya sea 8x8 o 16x16) y del número de threads, y cada threads almacena sus nodos en su porción de array.

Es decir, si el array es de 100.000 de floats, y se utilizan 100 threads, cada thread dispondrá de un array de 1000 floats para almacenar los nodos expandidos.

Este problema no existe con el algoritmo secuencial, ya que la matriz en la cual se van almacenando los nodos, se inicializa para cada señal a decodificar.

,

Al tratarse de un algoritmo extenso, se encuentra en el Anexo I.

6.5 Análisis teóricos y experimentales

Las pruebas secuenciales han sido realizadas, como se comentó en el primer capítulo, en un Macbook Pro con procesador Intel Core 2 Duo a 2.53 Ghz, mientras que los algoritmos paralelos se han ejecutado en una GPU 4 Tesla C1060 de 30 multiprocesadores con 8 cores cada uno, lo que hace un total de 240 cores.

Para una buena comparación entre los tiempos obtenidos mediante GPU y los obtenidos mediante CPU convendría primero medir la diferencia de velocidad de ambos procesadores. Por tanto, se ha realizado un sencillo programa de suma de vectores y se ha calculado el tiempo en realizar una operación en coma flotante en ambas máquinas, utilizando solo 1 thread en el caso de la GPU:

N	T_CPU (ms)	T_GPU(ms)	1 Flop CPU (us)	1 Flop GPU (us)	CPU vs GPU
100	0,002	0,105	0,0191	1,0467	54,88
1000	0,011	0,561	0,0110	0,5606	51,12
10000	0,153	5,328	0,0153	0,5328	34,86
100000	1,369	52,747	0,0137	0,5275	38,53
1000000	13,752	519,685	0,0138	0,5197	37,79
10000000	133,273	5183,590	0,0133	0,5184	38,89
100000000	1318,120	54039,900	0,0132	0,5404	41,00

Tabla 6.1 – Tiempo de ejecución en suma de vectores, comparando CPU-GPU(1thread)

Como se ve en la tabla 6.1, el tiempo en realizar una operación en coma flotante en la CPU es mucho menor que en la CPU. Para que se vea más claro se va a realizar el mismo procedimiento, pero en lugar de hacer una suma de vectores, ejecutando el algoritmo ASD-SVD con un tamaño de 8x8 (recordemos que equivale a un sistema real de 4 antenas emisoras y 4 receptoras), un tamaño de constelación de 8 y suponiendo buen conocimiento del canal en la estimación. Se han tomado estos valores, pero se podían haber tomado otros sin que el resultado cambiara en exceso.

N	T_CPU (ms)	T_GPU(ms)	CPU vs GPU
128	11,393	525,31	46,11
256	21,438	1106,67	51,62
512	42,627	2282,68	53,55
1024	82,260	4512,91	54,86
2048	162,560	9005,91	55,40
4096	321,760	18331,50	56,97
8192	641,609	36798,20	57,35

Tabla 6.2 – Tiempo de ejecución del algoritmo ASD-SVD, comparando CPU-GPU(1thread)

Como se aprecia en la tabla 6.2, la CPU es unas 55 veces más rápida que la tarjeta gráfica, por tanto, a pesar de disponer de 240 cores para paralelizar el algoritmo, hay que tener en cuenta que estos cores son mucho más lentos, así que como se verá, no en todas las ocasiones se podrá mejorar al algoritmo secuencial.

Para aprovechar al máximo la GPU, convendría que el número de señales a codificar concurrentemente fuera el mayor posible. Esto como veremos supone una limitación importante debido a que cuanto más aumentemos el número de señales, mayor será la cantidad de memoria de la tarjeta gráfica que se necesitaría para resolver el algoritmo.

Para ello la utilización de las técnicas estudiadas en el capítulo 4 han sido fundamentales, en orden a limitar la memoria necesaria para almacenar nodos expandidos en el algoritmo ASD.

En esta primera prueba podemos comprobar que el algoritmo paralelo funciona de tal forma que cuantos más hilos se utilizan, más rápido es el algoritmo:

Antes de ponernos a comparar entre los tiempos de la GPU respecto a la CPU, se ha realizado una pequeña prueba para comprobar si el algoritmo paralelo funciona correctamente. Se trata de un problema 8x8 con una constelación 32-PAM, y una relación señal a ruido en la decodificación de 15. En la siguiente gráfica podemos ver como a medida que aumenta el número de threads utilizados, el tiempo se reduce, pasando de más de 5 segundos con 1 thread, hasta menos de medio segundo cuando se utiliza un thread por señal a decodificar:

Num. Threads	T_GPU (s)
1	5,435
2	3,789
4	2,839
8	2,096
16	1,539
32	1,221
64	0,820
128	0,431

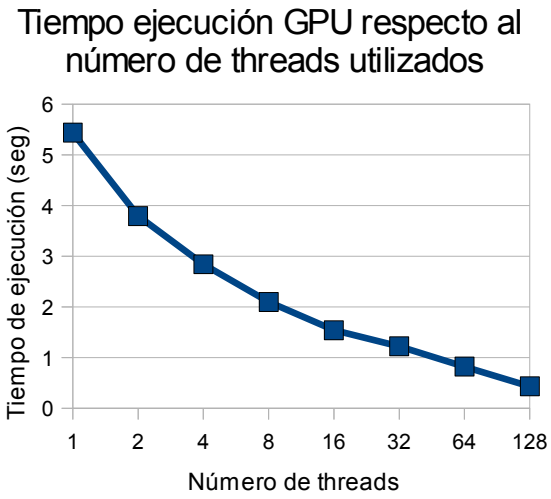


Tabla 6.3 y Fig. 6.5 – Evolución del algoritmo ASD-ZF-SVD, 8x8, 32-PAM y snr=15

A continuación se va proceder a comparar los tiempos obtenidos al ejecutar el algoritmo ASD en la GPU respecto a la CPU, tanto en problemas 8x8 como en 16x16, variando el tamaño de la constelación. Para estas pruebas se ha supuesto un conocimiento bueno del canal.

- **ASD con un problema 8x8 y una constelación 4-PAM**

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia CPU	NodMedia GPU
128	5,78	7,14	0,81	4,01	4,01
256	12,05	7,28	1,65	4,03	4,03
512	21,82	7,23	3,02	4,02	4,01
1024	43,82	8,64	5,07	4,01	4,02
2048	86,47	11,29	7,66	4,02	4,01
4096	171,13	18,33	9,34	4,02	4,02

Tabla 6.4 - ASD-ZF-SVD con tamaño 8X8 y constelación 4-PAM

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia_CPU	NodMedia_GPU
128	5,94	6,51	0,91	4,02	4,01
256	11,27	7,03	1,6	4,03	4,03
512	23,3	7,37	3,16	4,02	4,01
1024	43,98	9,15	4,81	4,02	4,02
2048	87,59	13,44	6,52	4,02	4,01
4096	171,58	22,68	7,57	4,03	4,02

Tabla 6.5 - ASD-ZF-SVD-QRP con tamaño 8X8 y constelación 4-PAM

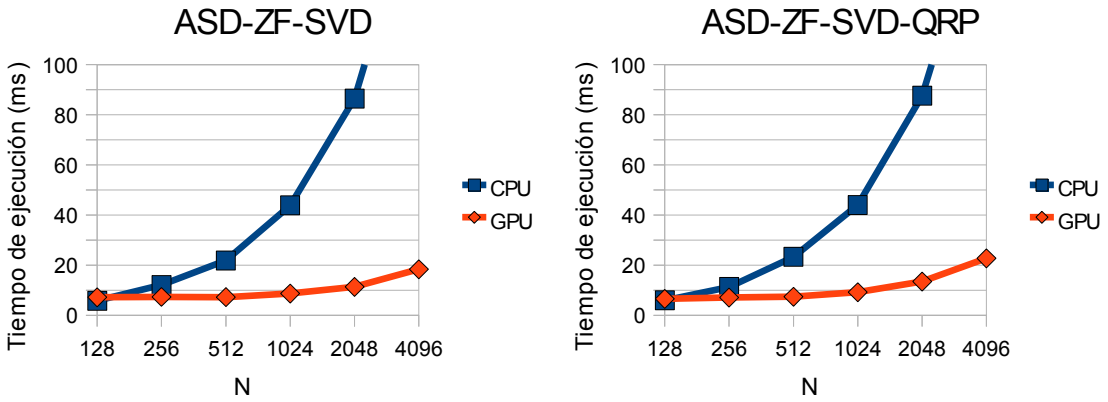


Fig. 6.6 -Diferencia de tiempos CPU vs GPU del algoritmo ASD con y sin QRP, con un tamaño de problema 8X8 y constelación 4-PAM.

En la figura 6.6 podemos ver que con una constelación de 4 valores y un tamaño de 8x8 el algoritmo GPU mejora al de la CPU si el número de señales a decodificar es alto. Otro dato importante que añadimos a la tabla es la media de nodos expandidos tanto en CPU como en la GPU, ya que como vimos en el capítulo 4, afecta en gran medida al tiempo de ejecución del algoritmo.

Los tiempos son prácticamente los mismos si se le aplica el método de la QR con pivotamiento, esto es debido a que el número de nodos expandidos ya era de por sí mínimo, con lo que aplicar la QRP en este caso no tiene ninguna mejora.

- **ASD con un problema 8x8 y una constelación 8-PAM**

N	T_CPU (ms)	T_GPU(ms)	SeepUp	NodMedia CPU	NodMedia GPU
128	11,39	18,86	0,6	10,59	9,82
256	21,44	18,37	1,17	9,55	9,78
512	42,63	22,47	1,9	10,01	9,94
1024	82,26	28,17	2,92	9,89	9,92
2048	162,56	39,19	4,15	10,06	10,01
4096	321,76	46,09	6,98	10,04	10,03

Tabla 6.6 - ASD-ZF-SVD con tamaño 8X8 y constelación 8-PAM

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia CPU	NodMedia GPU
128	10,22	14,17	0,72	9,93	8,59
256	19,56	14,23	1,37	9,58	8,66
512	35,50	17,84	1,99	9,92	8,80
1024	77,08	18,10	4,26	9,85	8,73
2048	153,08	22,51	6,80	10,01	8,72
4096	301,61	37,32	8,08	9,96	8,78

Tabla 6.7 - ASD-ZF-SVD-QRP con tamaño 8X8 y constelación 8-PAM

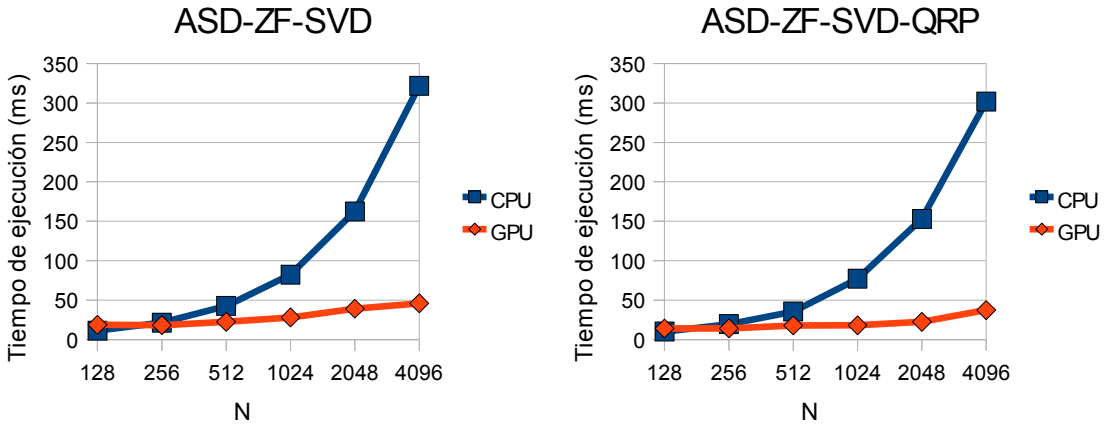


Fig. 6.7 -Diferencia de tiempos CPU vs GPU del algoritmo ASD con y sin QRP, con un tamaño de problema 8X8 y constelación 8-PAM.

Aumentando la constelación a 8-PAM el rendimiento es similar, se aprecia una clara mejora del algoritmo con CUDA con respecto al secuencial, y esta vez sí, se nota una reducción del tiempo de ejecución al aplicar la descomposición QR con pivotamiento.

- **ASD con un problema 8x8 y una constelación 16-PAM**

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia CPU	NodMedia GPU
128	26,75	80,84	0,33	27,47	26,72
256	52,68	87,60	0,60	27,02	26,24
512	104,41	102,32	1,02	26,79	27,33
1024	206,37	110,45	1,87	27,09	27,11
2048	409,24	153,25	2,67	26,87	26,98
4096	828,83	468,10	1,77	27,46	27,28

Tabla 6.8 - ASD-ZF-SVD con tamaño 8X8 y constelación 16-PAM

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia CPU	NodMedia GPU
128	24,66	79,50	0,31	26,00	26,60
256	48,82	82,47	0,59	26,26	25,60
512	95,06	99,88	0,95	26,28	26,91
1024	188,94	106,83	1,77	26,24	25,94
2048	377,14	146,98	2,57	26,29	26,37
4096	758,18	221,04	3,43	26,70	26,56

Tabla 6.9 - ASD-ZF-SVD-QRP con tamaño 8X8 y constelación 16-PAM.

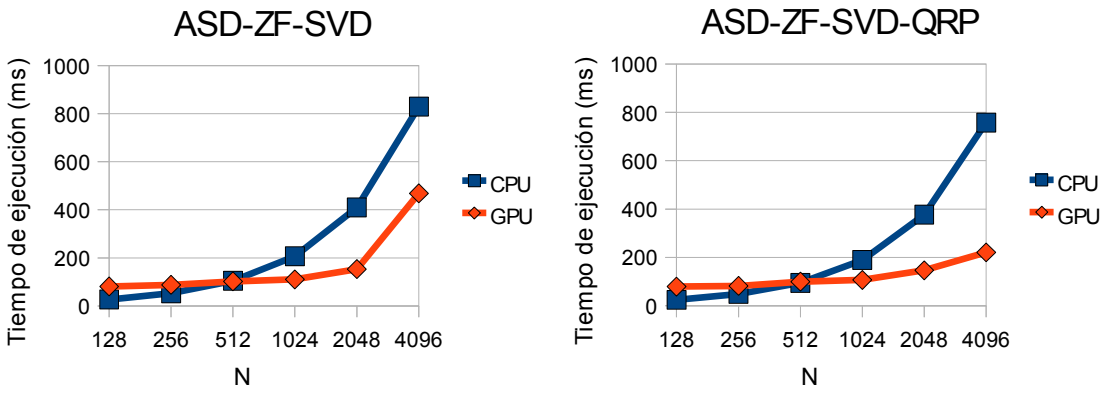


Fig. 6.8 -Diferencia de tiempos CPU vs GPU del algoritmo ASD con y sin QRP, con un tamaño de problema 8X8 y constelación 16-PAM.

Con una constelación de 16-PAM, el número de señales a decodificar para que el algoritmo ejecutado en la GPU sea más rápido con respecto al de la CPU aumenta a más de 512. Esto es debido a que cuanta mayor sea la constelación de símbolos, más alto es el número de operaciones en coma flotante del algoritmo (el número de posibles soluciones aumenta exponencialmente), y como hemos visto al principio de este capítulo, los procesadores de la GPU son mucho más lentos que el de la CPU.

- **ASD con un problema 8x8 y una constelación 32-PAM**

N	T_CPU (ms)	T_GPU(ms)	NodMedia CPU	NodMedia GPU
128	47,86	152,29	43,42	43,48
256	93,39	168,74	44,88	43,96
512	190,68	174,83	44,54	43,90
1024	391,95	473,87	45,22	44,72
2048	771,77	955,54	44,74	45,00
4096	1525,42	1176,26	44,81	44,63

Tabla 6.10 - ASD-ZF-SVD con tamaño 8X8 y constelación 32-PAM

N	T_CPU (ms)	T_GPU(ms)	NodMedia CPU	NodMedia GPU
128	43,51	127,76	42,40	42,58
256	86,53	168,57	43,40	42,98
512	173,90	180,24	43,47	43,11
1024	350,27	300,97	43,79	44,00
2048	687,92	557,58	43,91	43,78
4096	1383,82	654,47	44,01	43,61

Tabla 6.11 - ASD-ZF-SVD-QRP con tamaño 8X8 y constelación 32-PAM

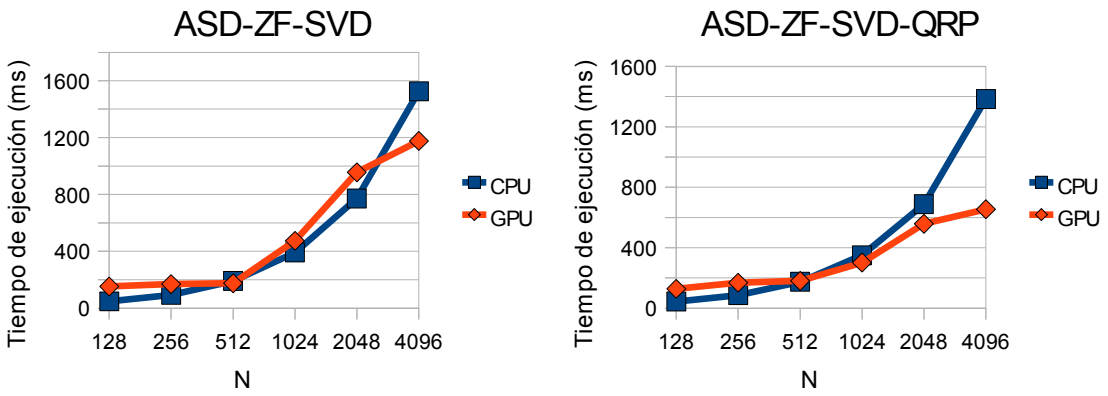


Fig. 6.9 -Diferencia de tiempos CPU vs GPU del algoritmo ASD con y sin QRP, con un tamaño de problema 8X8 y constelación 32-PAM.

- **ASD con un problema 16x16 y una constelación 4-PAM**

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia CPU	NodMedia GPU
128	43,24	419,96	0,10	18,05	20,35
256	106,70	491,45	0,22	19,93	18,96
512	211,81	698,02	0,30	19,55	19,54
1024	408,95	768,94	0,53	19,02	19,89
2048	843,21	972,14	0,87	19,80	19,30
4096	1695,82	1855,43	0,91	20,21	19,72

Tabla 6.12 - ASD-ZF-SVD con tamaño 16X16 y constelación 4-PAM

N	T_CPU (ms)	T_GPU(ms)	SpeedUp	NodMedia CPU	NodMedia GPU
128	42,85	312,43	0,14	14,34	16,91
256	95,45	319,98	0,30	16,43	16,02
512	180,72	373,48	0,48	15,90	16,73
1024	376,92	633,14	0,60	16,74	15,84
2048	746,19	735,49	1,01	16,58	16,68
4096	1487,18	1185,25	1,25	16,59	16,55

Tabla 6.13 - ASD-ZF-SVD-QRP con tamaño 16X16 y constelación 4-PAM

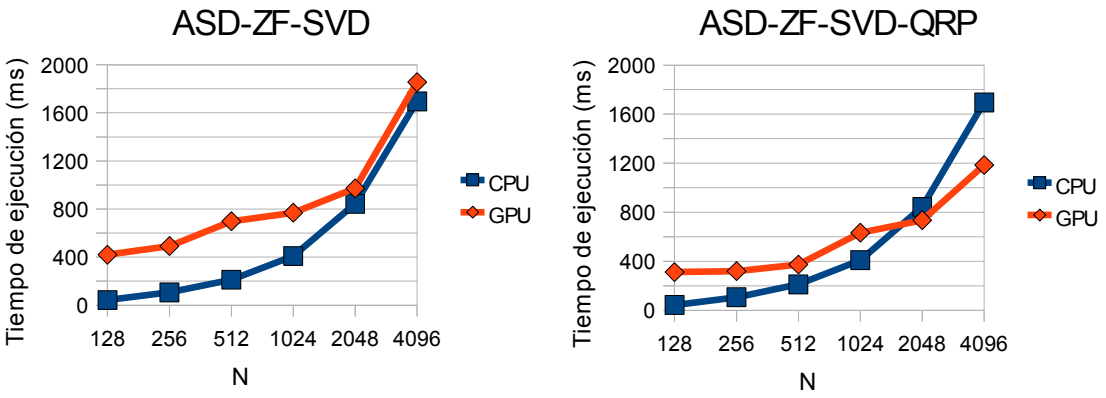


Fig. 6.10 -Diferencia de tiempos CPU vs GPU del algoritmo ASD con y sin QRP, con un tamaño de problema 16X16 y constelación 4-PAM

Aumentando el tamaño del problema a 16x16 se aprecia claramente que el algoritmo ejecutado en la tarjeta gráfica no resulta tan bueno. Aún así se ve una clara mejora si se le aplica la QR con pivotamiento. Una posible solución sería aumentar el buffer de señales a decodificar.

- **ASD con un problema 16x16 y una constelación 8-PAM**

N	T_CPU (seg)	T_GPU(seg)	SpeedUp	NodMedia CPU	NodMedia GPU
128	1,32	10,24	0,13	278,08	262,53
256	2,51	30,84	0,08	272,13	286,86
512	5,99	61,17	0,10	303,89	291,54
1024	12,86	68,20	0,19	312,35	295,51
2048	25,74	79,45	0,32	307,18	293,34
4096	46,60	310,37	0,15	291,67	302,10

Tabla 6.14 - ASD-ZF-SVD con tamaño 16X16 y constelación 8-PAM

N	T_CPU (seg)	T_GPU(seg)	SpeedUp	NodMedia CPU	NodMedia GPU
128	0,86	13,77	0,06	195,53	189,91
256	1,61	11,65	0,14	193,66	186,14
512	3,42	16,15	0,21	212,96	192,27
1024	7,39	27,57	0,27	215,12	212,24
2048	14,56	30,74	0,47	209,78	203,70
4096	27,20	68,01	0,40	206,10	211,54

Tabla 6.15 - ASD-ZF-SVD-QRP con tamaño 16X16 y constelación 8-PAM

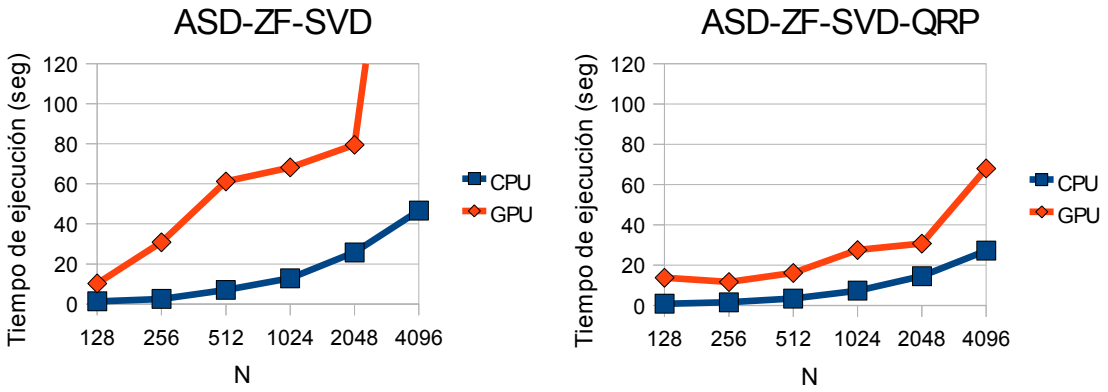


Fig. 6.11 -Diferencia de tiempos CPU vs GPU del algoritmo ASD con y sin QRP, con un tamaño de problema 16X16 y constelación 8-PAM

El número de operaciones en coma flotante es demasiado grande, y por tanto los tiempos al ejecutar el programa en la GPU empeoran. También se observa una gran diferencia entre el rendimiento del algoritmo al aplicar la QRP y sin dicha mejora.

6.6 Conclusiones

Para concluir con el capítulo podemos decir que la paralelización en CUDA resulta eficiente para sistemas MIMO de tamaño 8x8, ya que al aumentar el problema también aumentan el número de operaciones en coma flotante, y se aprecia más la lentitud de los procesadores de la GPU respecto a los de la CPU. Una posible solución seriá aumentar el número de señales a decodificar de forma concurrente, aunque con el inconveniente que cuantas más señales haya que decodificar, más memoria de la GPU es necesarea para poder almacenar los nodos expandidos de todas las señales.

Capítulo 7

Conclusiones generales y líneas abiertas

En este último capítulo de la tesis, resumimos las conclusiones generales obtenidas del trabajo realizado y se proponen futuros trabajos en la misma línea de investigación, al igual que posibles mejoras.

7.1 Conclusiones generales

Se ha diseñado de forma integral un sistema MIMO que incluye un estimador de canal y un decodificador de señales con GPUs de NVIDIA.

Para implementar el estimador del canal, se ha realizado un estudio del artículo de M. Biguesh, A. Gershman, "Training- Based MIMO Channel Estimation: A Study of Estmiator Tradeoffs and Optimal Training Signals", implementando tanto el método de mínimos cuadrados (LS) como el de mínimos cuadrados escalado (SLS), ambos vistos en el capítulo 2.

También para el decodificador de señales, se ha realizado un estudio intensivo del algoritmo de Karen Su "Automatic Sphere Decoder", el cual utiliza un esquema de ramificación y poda para encontrar la solución exacta al problema de mínimos cuadrados discretos, teniendo su principal desventaja en el número de nodos que tiene que ir almacenando en memoria para tamaños de problema o de constelación grandes.

En los capítulos 3 y 4 se ha realizado un estudio y se han implementado métodos para mejorar el rendimiento de este algoritmo, como son la utilización del punto de Babai para determinar el Radio para acotar el número de nodos expandidos, o la gran mejora que supone la descomposición en valores singulares para reducir aún más ese radio.

Otro de los métodos aportados es la descomposición QR con pivotamiento de la matriz de Canal, para así obtener los valores con mayor carga en las primeras ramas del árbol, obligando así a que el Radio sea más grande, y por tanto las no-óptimas son "podadas" antes. Este método es óptimo si tenemos un buen conocimiento del canal, y como gracias al analizador de señales y al temporizador del dispositivo virtual, el conocimiento del canal siempre es bueno, no hay problema en utilizarlo.

También se ha visto que la utilización de las GPUs para paralelizar el algoritmo ASD con todas las mejoras, utilizando para ello la arquitectura Cuda de NVIDIA, aplica una mejora importante respecto al rendimiento en la CPU para tamaños de problema 8x8, aunque ahora mismo para problemas mayores no es eficiente debido a la diferencia de velocidad existente al realizar operaciones en coma flotante entre los cores de las tarjetas gráficas actuales con respecto los de las CPUs.

7.2 Lineas abiertas

- Optimizar el código implementado en CUDA sobre la nueva tarjeta Fermi de NVIDIA, la cual posee 480 procesos y la posibilidad de lanzar kernels en paralelo.
- Mejorar la QR con pivotamiento ya que la utilizada en este tesis es la que se obtiene de lapack, que sólo toma en consideración el valor de mayor peso de la diagonal de la matriz triangular, no la columna entera.
- Una posibilidad para solucionar la deficiencia de rendimiento de las GPU para tamaños de problema grandes sería aumentar el número de señales a decodificar, optimizando y refinando el código utilizado en esta tesis para resolver el problema de memoria.
- Se esta preparando un artículo utilizando las técnicas propuestas con el título "*Implementación del ASD mejorado para decodificar señales discretas en una GPU*" que será presentado en la próxima edición de la conferencia CMMSE 2011.

Bibliografía

[1] G. J. Foschini and M. J. Gans, “On limits of wireless communication in a fading environment when using multi-element antennas”, Wireless Personal Communications, 6 (1998), pp. 311-335.

[2] I. E. Telater, “Capacity of multi-antenna gaussian channels”, Europ. Trans. Telecommun., (1999), pp. 585-595.

[3] E.G. Larsson and P.Stoica, Space-Time Block Coding for Wireless Communications. Cambridge, U.K.: Cambridge Univ. Perss, 2003.

[4] Y. Li, “Optimal training sequences for OFDM systems with multiple transmit antennas”, Proc. GLOBECOM'00, vol.3, pp.1478-1482, 2000.

[5] M. Biguesh, A. Gershman, “Training- Based MIMO Channel Estimation: A Study of Estmiator Tradeoffs and Optimal Training Signals”.

[6] M. R. Garey and D. S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W. H. Freeman, New York, NY, USA, 1979.

[7] J. Pearl, “Heuristics: Intelligent Search Strategies for Computer Problem Solving”, Addison-Wesley Longman Publishing Co., Inc., Boston, MAUSA, 1984.

[8] E. L. Lawler and D. E. Wood, “Branch-And-Bound Methods: A Survey”, Operations Research, 14 (1966), pp. 699-719.

[9] T.G. Kolda, R. M. Lewis, and V. Torczon, “Optimizatio by Direct Search: New perspective on Some Clasical and Modern Methods”, SIAM Review, 3 (2003), pp. 385-442.

[10] Karen su, C. N. Jones and I. J. Wasell. “An automatic sphere decoder”. Submitted to IEEE Transactions on Signal Processing, 2004.

[11] Tesis Doctoral de R. A. Trujillo Rasúa, “Algoritmos paralelos para la solución de problemas de optimización discretos aplicados a la decodificación de señales”.

[12] G. Golub and C. F. Van Loan, “*Matrix Computations”*, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[13] C. C. Martin, J. H. Winters, and N. R. Sollenberger, “Multiple-input multiple-output (MIMO) radio channel measurements”, Proc. VTC'00-Fall, vol. 2, pp. 774-779, Sep. 2000.

[14] T. L. Marzetta, “BLAST training: Estimating channel characteristics for high capacity space-time wireless,” in *Proc. 37th Annu. Allerton Conf. Communications, Control, Computing*, Monticello, IL, Sep. 1999.

[15] Q. Sun, D. C. Cox, H. C. Huang, and A. Lozano, “Estimation of continuous flat fading MIMO channels,” *IEEE Trans. Wireless Commun.*, vol. 1, pp. 549–553, Oct. 2002.

[16] A. Scaglione and A. Vosoughi, “Turbo estimation of channel and symbols in precoded MIMO systems,” in *Proc. ICASSP*, vol. 4, Montreal, PQ, Canada, May 2004, pp. 413–416.

[17] M. Biguesh and A. B. Gershman, “Downlink channel estimation in cellular systems with antenna arrays at base stations using channel probing with feedback,” *EURASIP J. Appl. Signal Process. (Special Issue on Advances in Smart Antennas)*, pp. 1330–1339, Sep. 2004.

[18] X. Ma, G. B. Giannakis, and S. Ohno, “Optimal training for block transmission over doubly selective wireless fading channels,” *IEEE Trans. Signal Process.*, vol. 51, pp. 1351–1366, May 2003.

[19] E. G. Larsson and P. Stoica, *Space-Time Block Coding forWireless Communications*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[20] E. de Carvalho and D. T. M. Slock, “Blind and semi-blind FIR multichannel estimation: (Global) identifiability conditions”, *IEEE Trans.Signal Process.*, vol. 52, pp. 1053–1064, Apr. 2004.

[21] M. Biguesh and A. B. Gershman, “Training-based MIMO channel estimation: Optimal training and estimator tradeoffs,” in *Proc. ICC’04*, vol. 5, Paris, France, Jun. 2004, pp. 2658–2662.

[22] T. Kailath, H. Vikalo, and B. Hassibi, “Space-Time Wireless Systems: From Array Processing to MIMO Communications”, Cambridge University Press, 2006, ch. MIMO receive algortihms, pp. 302-322.”

[23] J. Céa, “Optimisation: Théorie et algorithmes”, (1971).

[24] E. Polak, “Computational Methods in Optimization: A Unified Approach.”, Academic Press, New York, 1971.

[25] J. E. Dennis, Jr. And V. Torczon, “Direct search methods on parallel machines”, SIAM Journal Optimization, 1 (1991), pp. 448-474.

[26] V. Torczon, “Multi-Directional Search: A Direct Search Algorithm for Parallel Machines.”, tech. report, Department of Computational and applied Mathematics, Rice University, Houston, TX, 1990.

[27] U. Fincke and M. Pohst, “Improved methods for calculating vectors of short length in a lattice, including a complexity analysis”, Mathematics of Computation, 44(170) (1985), pp. 463–471.

[28] Q. Liu and L. Yang, “A Novel Method for Initial Radius Selection of Sphere Decoding”, IEEE Vehicular Technology Conference, 2 (2005), pp. 1280–1283.

[29] B. Hassibi and H. Vikalo, “On sphere decoding algorithm. I. Expected Complexity”, IEEE Transactions on Signal Processing, 53 (2005), pp. 2806–2818.

[30] G. J. Foschini, G. D. Golden, F. Reinaldo A. Valenzuela, and P. W. Wolniansky, “Simplified Processing for High Spectral Efficiency Wireless Communication Employing Multi-Element Arrays”, IEEE Journal On Selected Areas in Communications, 17 (1999), pp. 1841–1852.

[31] K. Su and I. J. Wassell, “A New Ordering for Efficient Sphere Decoding”, in International Conference on Communications, May 2005.

[32] K. Su, “Efficient Maximum-Likelihood detection for communication over Multiple Input Multiple Output channels, tech. report, Department of Engineering”, University of Cambridge, 2005.

[33] S. M. Kay, “*Fundamentals of Statistical Signal Processing: Estimation Theory”*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[34] A. Marzal, M. J. Castro, P. Aibar, “Ingeniería Informática. Algorítmica”, cap. 9, 2006/2007, Universidat Jaume I de Castelló.

[36] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger, Closest Point Search in Lattices, IEEE Transactions on Information Theory, 48 (2002), pp. 2201–2214.

[35] C.-P. Schnorr and M. Euchner, “Lattice basis reduction: Improved practical algorithms and solving subset sum problems”, Math. Programming, 66 (1994), pp. 181–191.

[36] http://www.nvidia.com/object/cuda_home_new.html

[37] D. Micciancio, “The Hardness of the Closest Vector Problem with Preprocessing”, IEEE Transactions on Information Theory, 47 (2001).

[38] “*NVIDIA CUDA Programming Guide*” version 3.2.

[39] K. Maurer, J. Jaldén, D. Seethaler, G. Matz, “Achieving a Continuous Diversity-Complexity Tradeoff in Wireless MIMO Systems via Pre-Equalized Sphere-Decoding”, Inst. of Commun. & Radio-Freq. Eng.

[40] http://www.inco2.upv.es

[41] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. Mackenney, S. Ostrouchv, y D. Sorensen, “LAPACK User Guide”, second de. 1995.

[42] http://www.netlib.org/lapack

[43] “Cula Reference Manual”, release 1.1, EM Photonics, www.culatools.com.

ANEXO I

```
__global__ void asd_svd_gpu(float *C, float *babai, int m, int n, int N, int L, float *Sdec, int *maxnodos)
{
	int tid = threadIdx.x+blockIdx.x*blockDim.x;
	int i,j;
	int nivel;
	int continuar;
	int cont;
	int nodo;
	int padre;
	int offset;
	float dist;
	int bl = nodMax*N*(m+3);// NodMax * N * (m+3)
	while (tid < N){
		continuar=1;
		nodo=0;
		padre=0;
		nivel=1;
		maxnodos[tid]=L;
		offset=ceilf((bl/N))*tid;
		A[offset+0*(m+3)+(m+1)]=babai[tid];
		while (continuar == 1){
		// Se crean L ramas del arbol y se calcula su distancia
			for (j=0;j<L;j++){
				nodo=nodo+1;
				ramificar_svd(offset,m,L,nodo,padre,nivel,j);
				// Nivel de la rama (de 1 a m)
				A[offset+nodo*(m+3)+(m+2)]=nivel;
				// Calcula la distancia ||Rs-c||^2
				distancia_svd(tid,offset,babai,C,m,n,N,L,nivel,nodo,Sdec);
				if (nodo > maxnodos[tid])
					maxnodos[tid]=nodo;
			}//for L
			A[offset+padre*(m+3)+m]=INF;//2*babai[tid];
			// Eliminamos todas las ramas con distancia mayor a distmax
			podar_svd(offset,m,babai[tid],nodo,&dist,&cont);
			// Si llega al nivel "m" y su distancia es menor a distmax, esta sera la nueva
distmax
			if (nivel==m)
				if (dist < babai[tid])
					babai[tid]=dist;
			// Buscamos la nueva rama a expandir
			for (i=0;i<cont;i++){
				if (A[offset+i*(m+3)+m] <= dist){
					dist=A[offset+i*(m+3)+m];
					nivel=(int)A[offset+i*(m+3)+(m+2)]+1;
					padre=i;
				} //if
			} //for
			// Si la rama a expandir esta en el ultimo nivel se termina el algoritmo
			if (nivel == m+1){
				for (j=0;j<m;j++)
```

```
                                        Sdec[j*N+tid]=A[offset+padre*(m+3)+j];
                                continuar=0;
                        } //if
                        nodo=cont-1;
                }//while
                tid+=blockDim.x * gridDim.x;
        }//while tid

        return;
}
__device__ void ramificar_svd(int offset, int m, int L, int nodo, int padre, int nivel, int j){
        // Hereda de la rama Padre
        for (int i=0;i<m+2;i++)
                A[offset+nodo*(m+3)+i]=A[offset+padre*(m+3)+i];
        // Anyade un valor mas como Hijo
        A[offset+nodo*(m+3)+(m-nivel)]=-(L-1.0f)/2.0f+j;
        return;
}
__device__ void distancia_svd( int tid, int offset, float *babai, float *C, int m, int n, int N, int L, int nivel,
int nodo, float *y){
        float mul;
        float norm2,ro2,b,qb;
        if (nivel < m){
                // z=R12*s2'-c1
                for (int i=0;i<m-nivel;i++){
                        mul=0.0f;
                        for (int k=m-nivel;k<m;k++)
                                mul=mul+Rconst[k*n+i]*A[offset+nodo*(m+3)+k];
                        y[i*N+tid]=mul-C[tid*n+i];
                }
                //Resolver sistema R11*b=z
                ro2=0.0f;
                for(int i=m-nivel-1;i>-1;i--){
                        b=y[i*N+tid]/Rconst[i*n+i];
                        qb=aproximar_gpu(b,L);
                        for(int k=0;k<i+1;k++){
                                y[k*N+tid]=y[k*N+tid]-Rconst[i*n+k]*b;
                        }
                        ro2=ro2+(b-qb)*(b-qb);
                }
                ro2=ro2*sigmaconst[m-nivel-1]*sigmaconst[m-nivel-1];
                A[offset+nodo*(m+3)+(m+1)]=babai[tid]-ro2;
                /* norm(R22*s2-c2)^2 */
                norm2=0.0f;
                for(int i=m-nivel;i<m;i++){
                        mul=0.0f;
                        for(int k=m-nivel;k<m;k++)
                                mul=mul+Rconst[k*n+i]*A[offset+nodo*(m+3)+k];
                        mul=mul-C[tid*n+i];
                        norm2=norm2+mul*mul;
                }
                A[offset+nodo*(m+3)+m]=norm2;
        }else{
                /* norm(R*s-c)^2 */
                norm2=0.0f;
```

```
                for(int i=0;i<m;i++){
                        mul=0.0f;
                        for(int k=i;k<m;k++){
                                mul=mul+Rconst[k*n+i]*A[offset+nodo*(m+3)+k];
                        }
                        mul=mul-C[tid*n+i];
                        norm2=norm2+mul*mul;
                }
                A[offset+nodo*(m+3)+m]=norm2;
                A[offset+nodo*(m+3)+(m+1)]=babai[tid];
        }
        return;
}
__device__ float aproximar_gpu (float num, int L)
{
        float num_aprox;
        num_aprox=floorf(num)+0.5f;
        if(num>(L-1.0f)/2.0f)
                num_aprox=(L-1.0f)/2.0f;
        if(num<(-L+1.0f)/2.0f)
                num_aprox=(-L+1.0f)/2.0f;
        return num_aprox;
}
```