Departamento de Sistemas Informáticos
Universidad Politécnica de Valencia

Tesis Doctoral
para optar al grado de
Doctor en Informática

Computación Paralela de la Transformada Wavelet; Aplicaciones de la Transformada Wavelet al Álgebra Lineal Numérica

Presenta:
Liesner Acevedo Martínez

Dirige:
Dr. Víctor Manuel García Mollá

Valencia, Julio de 2009

Le dedico este trabajo a los míos: Liano, Mary, Kiri y Tay. Liano (mi viejo), porque siempre has estado ahí, porque te has quitado todo por nosotros y lo has hecho todo por nosotros. Me diste el carácter y la fuerza de voluntad. Viejo, te regalo esto y todo lo que venga

Mary (mi viejita), nunca te pude explicar lo que era un doctorado, sin embargo siempre te sentí orgullosa de tu hijo. Me has dado el baile y la alegría. Mi vieja, te dedico esta "cosa" desde el corazón

Kiri (mi hermanito), tienes una opinión tan alta de mí que a veces tengo que correr para merecerla, espero haber llegado con esto a una parada intermedia. Me has dado "la calle" y esa sensación de que nunca estoy ni voy a estar solo. Kiro, te voy a dedicar lo que viene

Mi "brujita", no es necesario que te lo dedique, todo lo mío sabes que es tuyo.

Agradecimientos

Agradezco en primer lugar a mi director de tesis: Víctor Manuel García Mollá, por su paciencia y apoyo; por todas sus tiradas de orejas y sus correcciones certeras; pero sobre todo por su guía, confianza y palabras de aliento.

Al Dr. Antonio Vidal, a quién veo como un director más, le agradezco la asignatura "Conceptos y Métodos de la Computación Paralela" donde aprendí cuán bellamente complejo puede ser ese mundo de cálculos y mensajes. Le agradezco además todo el soporte, los consejos y las gestiones que nos ayudaron a seguir adelante.

A todos los profesores que impartieron asignaturas del doctorado: Antonio, Víctor, Vicente Vidal, Román, Pablo Galdámez, Vicente Hernández y Enrique (Kike).

Al comité de revisores: Dr. José Ranilla Pastor, Dr. Domingo Giménez Cánovas y el Dr. Enrique S. Quintana Ortí.

Agradezco la oportunidad a todos los miembros de la oficina CETA-UPV de la CUJAE y al consejo científico de esa universidad, en especial a María del Carmen Armenteros, Exiquio Leyva y Alejandro Rosette.

A mis compañeros del departamento de Programación de la UCI por su apoyo: Yadilka, Carlos, José Albert, Karel Osorio, Yaniela, Renier, Yanesky.

A los que me contagiaron con el bicho de la investigación y me apoyaron en la elección de este doctorado: Daniel Gálvez y Pedro Piñero.

A los buenos amigos de Valencia: Erik, Lisandro, Idel, Maikel y todos los demás. Gracias por hacerme la vida más llevadera.

A mi estudiante Rigoberto Leander por seguirme en esta aventura.

A mi compañero de batalla, a mi amigo Rafe. Fueron 6 años dando guerra, resistiendo ciclones, olas de calor, pasando penas en los aeropuertos, sed en los bares, viendo de lejos el Corte Inglés y "poniéndonos las botas" en los chinos, en fin, ..., siendo cubanos aquí y allá, riéndonos de todo y dándonos ánimos para llegar al final.

A todos los que de una forma u otra contribuyeron al desarrollo de esta tesis.

"Seamos realistas y hagamos lo imposible."

Ernesto Ché Guevara.

Resumen

Esta tesis ha tenido como doble objetivo el estudio de la computación paralela de la Transformada Wavelet Discreta (DWT), y el estudio de las aplicaciones de la DWT en el campo del álgebra lineal numérica

Inicialmente se realiza un estudio de las distintas variantes de paralelización de la DWT y se propone una nueva variante paralela, en memoria distribuida, con distribuciones de datos orientadas a bloques de matrices, como la distribución bidimensional cíclica a bloques (2DBC) típica de la librería ScaLAPACK. La idea es que la DWT en muchos casos es una operación intermedia y debe ajustarse a las distribuciones de datos que se estén usando. Se define y demuestra una forma de calcular exactamente la cantidad de elementos que debe comunicar cada procesador para que se puedan calcular de forma independiente (sin necesidad de comunicaciones posteriores) todos los coeficientes de la DWT en una cantidad de niveles determinada. Finalmente se propone una variante específica, más eficiente, para el cálculo de la DWT-2D cuando se aplica como paso previo a la resolución de un sistema de ecuaciones distribuido 2DBC. Esta propuesta consiste en realizar una permutación de las filas y columnas del sistema que disminuye las comunicaciones.

Otro de los aportes de esta tesis es el de considerar como un caso típico, el cálculo de la DWT-2D no estándar en matrices dispersas; proponemos algoritmos para realizar esta operación sin necesidad de construir explícitamente la matriz wavelet. Además tenemos en cuenta el fenómeno de rellenado (fill-in) que ocurre al aplicar la DWT a una matriz dispersa. Para ello exploramos con los métodos de reordenamiento clásicos de grado mínimo y de reducción a banda. De forma adicional sugerimos como pueden influir esos reordenamientos a la convergencia de los métodos multimalla ya que ocurre una redistribución de la norma de la matriz hacia los niveles inferiores de la representación multi-escala, lo que garantizaría una mejor compresión.

El campo de aplicación de la transformada wavelet que se propone es la resolución de grandes sistemas de ecuaciones lineales. En esta tesis expondremos dos aplicaciones específicas: paralelización de precondicionadores de sistemas lineales basados en la DWT, y el cálculo eficiente de la DWT-2D en matrices dispersas en conjunción con el análisis multi-resolución inherente a las wavelet y los métodos multimalla. Se estudian los Métodos Wavelet Multimalla Algebraicos (Wavelet Algebraic Multigrid Methods, WAMG), algoritmos que combinan los métodos multimalla con las wavelet, y que no necesitan de ningún conocimiento del problema a resolver; solo la matriz de coeficientes y la parte derecha del sistema. En particular se proponen dos nuevas variantes de los algoritmos WAMG. La primera se basa en la descomposición inducida del sistema lineal al aplicar la DWT y la segunda reduce el coste computacional de los ciclos del algoritmo multimalla saltando operaciones en algunos niveles o mallas. Finalmente se estudia la aplicación de los WAMG a la resolución eficiente de sistemas lineales desplazados.

Resum

Aquesta tesi ha tingut com a doble objectiu l'estudi de la computació paral·lela de la Transformada Wavelet Discreta (DWT), i l'estudi de les aplicacions de la DWT en el camp de l'àlgebra lineal numèrica

Inicialment es realitza un estudi de les diferents variants de paral·lelització de la DWT i es proposa una nova variant paral·lela, en memòria distribuïda, amb distribucions de dades orientades a blocs de matrius, com la distribució bidimensional cíclica a blocs (2DBC) típica de la llibreria ScaLAPACK. La idea és que la DWT en molts casos és una operació intermèdia i s'ha d'ajustar a les distribucions de dades que s'estiguen utilitzant. Es defineix i demostra una forma de calcular exactament la quantitat d'elements que ha de comunicar cada processador per tal que es puguen calcular de forma independent (sense necessitat de comunicacions posteriors) tots els coeficients de la DWT en una quantitat de nivells determinada. Finalment es proposa una variant específica, més eficient, per al càlcul de la DWT-2D quan s'aplica com pas previ a la resolució d'un sistema d'equacions distribuït 2DBC. Aquesta proposta consisteix a realitzar una permutación de les files i columnes del sistema, i disminueix les comunicacions.

Una altra aportació d'aquesta tesi és el de considerar com un cas típic, el càlcul de la DWT-2D no estàndard en matrius disperses; proposem algorismes per realitzar aquesta operació sense necessitat de construir explícitament la matriu DWT. A més tenim en compte el fenomen d'emplenat (fill-in) que ocorre en aplicar la DWT a una matriu dispersa. Per a això explorem amb els mètodes de reordenació clàssics de grau mínim i de reducció a banda. De manera addicional suggerim com poden influir aquests reordenaments a la convergència dels mètodes multimalla ja que ocorre una redistribució de la norma de la matriu cap als nivells inferiors de la representació multi-escala, el que garantiria una millor compressió.

El camp d'aplicació de la transformada wavelet que es proposa és la resolució de grans sistemes d'equacions lineals. En aquesta tesi exposarem dues aplicacions específiques: paral·lelització de precondicionadors de sistemes lineals basats en la DWT, i el càlcul eficient de la DWT-2D en matrius disperses en conjunció amb l'anàlisi multi-resolució inherent a les wavelet i els mètodes multimalla. S'estudien els Mètodes Wavelet Multimalla Algebraics (Wavelet Algebraic Multigrid Methods, WAMG), algorismes que combinen els mètodes multimalla amb les wavelet, i que no necessiten cap coneixement del problema a resoldre; només la matriu de coeficients i la part dreta del sistema. En particular es proposen dues noves variants dels algorismes WAMG. La primera es basa en la descomposició induïda del sistema lineal en aplicar la DWT i la segona redueix el cost computacional dels cicles de l'algorisme multimalla saltant operacions en alguns nivells o malles. Finalment s'estudia l'aplicació dels WAMG a la resolució eficient de sistemes lineals desplaçats.

Abstract

This thesis has had as double objective the study of the parallel computation of Discrete Wavelet Transform (DWT), and the study of the applications of the DWT in the field of numerical linear algebra.

Firstly a study is made of different variants of parallelization of the DWT, and a new parallel variant is proposed, in distributed memory, with data distributions oriented to blocks of matrices, like the two-dimensional block cyclic distribution (2DBC) used in the ScaLAPACK library. The idea is that, in many cases, the DWT is an intermediate operation and must be adjusted to the distributions of data that are being used. A new strategy has been proposed for computation of the multilevel parallel DWT, based on determining exactly how many elements must send and receive each processor so that all the wavelet coefficients can be calculated of independent form (no need of later communications). Finally another strategy has been proposed for the efficient calculation of the DWT-2D when it is applied as a previous step to the resolution of a system of equations distributed with the 2DBC distribution. This proposal consists of performing a permutation of rows and columns of the system that reduces the communications.

Another contribution of this thesis is to consider as a typical case, the calculation of the nonstandard DWT-2D of sparse matrices; we propose algorithms to conduct this operation without building the DWT matrix. In addition we consider the "fill-in"phenomenon that happens when the DWT is applied to a sparse matrix. The classic methods of reordering matrices, minimum degree and reduction to band algorithms, have been tested for this problem. It has been suggested that these reorderings can influence the convergence of the multigrid methods (proposed in the last chapter) since they cause a redistribution of the norm of the matrix towards the lower levels of the multi-scale representation, which would guarantee a better compression.

The main field of application of wavelet transform proposed is the solution of large systems of linear equations. In this thesis we will propose two specific applications: parallelization of preconditioners of linear systems based on the DWT, and the efficient calculation of the DWT-2D in sparse matrices, in conjunction with the multi-resolution analysis inherent to wavelets and the multigrid methods. The Algebraic Wavelet Multigrid Methods are studied (Wavelet Algebraic Multigrid Methods, WAMG), these are algorithms that combine the multigrid methods with DWT, and do not need any detailed knowledge of the problem to solve; only the matrix of coefficients and the right part of the system are needed. Two new variants of WAMG algorithms have been proposed. The first is based on the decomposition of the linear system when applying the DWT, and the second reduces to the computational cost of the cycles of the multigrid algorithm by skipping operations in some levels or meshes. Finally the application of the WAMG to the efficient resolution of shifted linear systems is studied.

Índice general

Capítulo 1

Introducción

La transformada wavelet discreta (DWT) es producto de una síntesis de ideas que han surgido a través de los años en varios campos del saber, como las matemáticas y el procesamiento de señales. En términos generales, la transformada wavelet es una herramienta que divide datos, funciones u operadores en diferentes componentes de frecuencia y luego estudia cada componente en resoluciones que se ajustan a su escala [26].

Existen muchas líneas de investigación actuales ligadas a la DWT. Esto es debido a que el número de aplicaciones que se benefician de esta herramienta crece diariamente. En la literatura se pueden encontrar un sinnúmero de aplicaciones de la transformada wavelet en distintas ramas: estadística [5, 10, 63], análisis de series temporales [49, 51, 21], procesamiento de señales [45, 62], procesamiento de imágenes [54, 47, 35], bases de datos [37], minería de datos [41], etc. Hay un boletín de noticias en la web dedicado solo a las aplicaciones de la trasformada wavelet llamado "Wavelet Digest" [2].

Un ejemplo clásico del uso de wavelets en el procesamiento de imágenes se muestra en la figura 1.1. En ella se observa como, en cada aplicación de los wavelets, la imagen original se comprime, obteniéndose una imagen de similar calidad pero reduciendo el almacenamiento necesario a la mitad (La imagen ha sido procesada con el Wavelet Toolbox de MATLAB[48]). Este tipo de procedimientos son el eje central de dos de las principales aplicaciones de la DWT en la compresión de imágenes: el formato `JPEG-2000` [60] y el método WSQ[1][36] que utiliza el FBI para comprimir su base de

[1]Del inglés Wavelet Scalar Quantization, cuantización escalar de wavelets

datos de huellas dactilares.

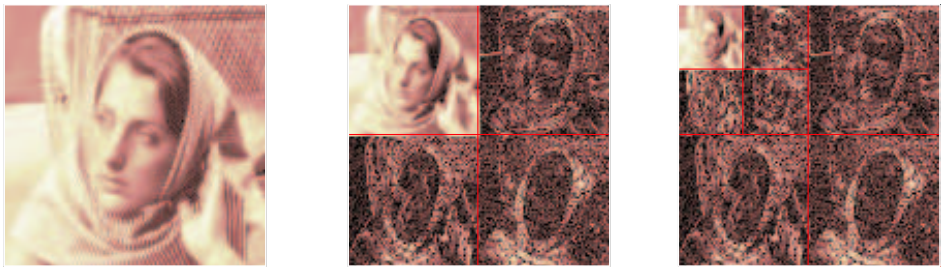


Figura 1.1: Imagen comprimida en 3 escalas de la DWT-2D

La trasformada wavelet se puede utilizar tanto para comprimir señales (o vectores) como imágenes (o matrices); de hecho se puede usar para cualquier dimensión. Aunque en teoría los algoritmos para calcular la DWT son rápidos, cuando se aplican a volúmenes de datos muy grandes es necesario buscar variantes de paralelización. En esta tesis introduciremos una nueva variante de paralelización de la DWT y en particular un algoritmo para calcular la DWT en matrices dispersas. También estudiaremos algunas aplicaciones de la DWT al álgebra lineal numérica, en particular la construcción de precondicionadores para resolver sistemas de ecuaciones lineales de gran dimensión, tanto densos como dispersos, y aplicaciones a los métodos multimalla algebraicos.

1.1. Cálculo paralelo de la transformada wavelets en memoria distribuida

Los trabajos más importantes para el cálculo paralelo de la DWT están mayormente enfocados al tratamiento de imágenes y señales y la estrategia que usan está en consonancia con obtener una transformada lo más rápido posible como operación única. El principal trabajo sobre paralelización del cálculo de wavelets en memoria distribuida es el de Nielsen y Hegland en [50], donde exponen dos modelos para el cálculo paralelo de la DWT-2D: un primer modelo al que le llaman “DWT Replicada” y un segundo modelo llamado “DWT eficiente en comunicaciones”. El procedimiento utilizado por la “DWT Replicada” se basa en una distribución de datos donde cada procesador reciba la misma cantidad de filas consecutivas de la matriz a transformar.

Se calcula la DWT-1D de cada fila, se traspone de forma paralela toda la matriz, y luego cada procesador calcula la DWT-1D de cada nueva fila. Por su parte en procedimiento de la "DWT eficiente en comunicaciones" usa una distribución de datos similar al de "DWT Replicada" pero evitando el paso de trasponer de forma paralela usando un esquema de comunicaciones más eficiente.

En Chaver et al. [18] se proponen dos variantes al trabajo de Nielsen y Hegland para el caso de la DWT no Estándar. La primera modificación tiene que ver con usar el mismo esquema de la "DWT replicada" pero evitando el paso de trasponer la matriz para hacer un mejor uso de la jerarquía de memoria. La segunda modificación propuesta en [18] tiene que ver con el cálculo repetido o multinivel de la DWT, se determina un "nivel crítico" a partir del cual el cálculo paralelo no puede mejorar el secuencial. Así el cálculo de la DWT se replica a partir de este "nivel crítico". Estas mejoras siguen teniendo la dificultad de que están optimizadas para el cálculo de la DWT como operación única.

En el capítulo 3 proponemos una variante paralela de la DWT-2D, que luego es usada en el epígrafe 5.1 en la paralelización de los precondicionadores de Chen en [16], que tiene en cuenta otras distribuciones de datos, en particular aquellas orientadas a bloques de matrices como la 2DBC de ScaLAPACK. La idea es que la DWT en estos casos es una operación intermedia y debe ajustarse a las distribuciones de datos que se estén usando para resolver el sistema. Esta propuesta se basa en replicar la cantidad de elementos necesarios para que cada procesador calcule de forma independiente todos sus coeficientes wavelet en una cantidad de niveles determinada. Además se define y demuestra una forma de calcular exactamente la cantidad de elementos que debe comunicar cada procesador. Finalmente se propone una variante específica, más eficiente, para el cálculo de la DWT-2D en la distribución 2DBC considerando una permutación de las filas y columnas del sistema.

Otro de los aportes de esta tesis es el de considerar como un caso típico, el cálculo de la DWT-2D no estándar en matrices dispersas. La mayor parte de las implementaciones existentes se basan en construir explícitamente la matriz wavelet dispersa y calcular la transformada de una matriz dispersa operando directamente con esta matriz. En esta tesis proponemos algoritmos para realizar esta operación sin necesidad de construir explícitamente la matriz wavelet. Además tenemos en cuenta el fenómeno de rellenado (*fill-in*) que ocurre al aplicar la DWT a una matriz dispersa. Para ello exploramos con los métodos de reordenamiento clásicos, como los de grado mínimo [9, 29] y de reducción a banda [25, 3]. De forma adicional sugerimos cómo esos

reordenamientos pueden afectar la convergencia de los métodos multimalla ya que ocurre una redistribución de la norma de la matriz hacia los niveles inferiores de la representación multi-escala, lo que garantizaría una mejor compresión.

1.2. Aplicaciones de DWT al álgebra lineal

La definición recursiva de los wavelets, su localización controlada tanto en tiempo como en espacio, la propiedad de los momentos nulos, capacidad de compresión, su rapidez y facilidad de cálculo, es lo que la hace una herramienta particularmente interesante y con múltiples usos dentro del álgebra lineal numérica [12].

Nuestro campo de aplicación de la transformada wavelet será, fundamentalmente, la resolución de grandes sistemas de ecuaciones lineales; sistemas que surgen en el proceso de solución de ecuaciones diferenciales. En esta tesis expondremos dos aplicaciones específicas: paralelización de precondicionadores de sistemas lineales basados en la DWT, y el cálculo eficiente de la DWT-2D en matrices dispersas en conjunción con el análisis multi-resolución inherente a las wavelet y los métodos multimalla.

En la resolución de sistemas de ecuaciones lineales se aprovecha una propiedad fundamental y es que cuando se transforma un sistema de ecuaciones lineales de la forma:

$$Ax = b \tag{1.1}$$

al espacio wavelet:

$$WAW^T x = Wb \Leftrightarrow \tilde{A}\tilde{x} = \tilde{b} \tag{1.2}$$

la transformación tiene la propiedad de ser ortogonal, por lo que no afecta los autovalores de la matriz del sistema, propiedad que garantiza que resolver el sistema (1.1) utilizando algún método iterativo sea equivalente a resolver el sistema (1.2) con el mismo método iterativo. La matriz W es la matriz de transformación wavelet, se verá más en detalle en el capítulo 2.

Existen en la bibliografía dos vertientes importantes de la aplicación de la DWT en la construcción de precondicionadores eficientes para la resolución de sistemas lineales densos y de gran dimensión [20]: los métodos que construyen una aproximación de la matriz del sistema (o de la inversa) después de aplicada la DWT estándar [35, 17, 11, 19, 64, 59] y los métodos que aprovechan la estructura multinivel que se obtiene al aplicar la DWT no estándar [16, 12, 32].

En [32] se usa la trasformada wavelet no estándar para definir una nueva álgebra matricial llamada Álgebra No Estándar. En este trabajo se redefinen las operaciones básicas del álgebra basándose en la estructura multi-nivel que se obtiene al aplicar la DWT no estándar y aprovechando las capacidades de compresión y momentos nulos de la DWT se obtienen nuevos operadores eficientes para gran variedad de problemas. En particular se redefinen la descomposición LU y la solución de sistemas triangulares con lo que se obtiene un método de solución directo. En el epígrafe 5.2 se aplica esta álgebra para definir un precondicionador similar a los LU incompletos (ILU) usando la LU no estándar con estructura dispersa. En [16] se sigue esta linea para proponer dos nuevos precondicionadores apoyándose en el desarrollo del sistema multinivel reordenado en complemento de *Schur*, y se demuestra que en algunos casos mejora la convergencia de GMRES precondicionado con ILU. En el epígrafe 5.1 propondremos una variante paralela para estos métodos que hace uso de una nueva propuesta para el cálculo paralelo de la DWT-2D no estándar.

Otra de las aplicaciones importantes de la DWT es en tratar de mejorar la eficiencia de los métodos multimallas. Los esquemas generales de los métodos multimalla son muy similares al proceso multiresolución para el cálculo de la DWT.

Un esquema multimalla consiste en tres elementos fundamentales [14]: Las representaciones de las matrices en varias escalas, las matrices de restricción y las matrices de interpolación. La representación de la matriz en varias escalas permite la manipulación de varios esquemas iterativos en mallas de diferentes escalas. Las matrices de restricción e interpolación se usan para convertir el error residual de una escala a sus escalas adyacentes. Específicamente una matriz de restricción proyecta el error residual desde una escala hasta su adyacente más gruesa, mientras que una matriz de interpolación hace el mapeo del error residual desde una escala a su adyacente más fina.

Existen en la literatura varios artículos dedicados a buscar la conexión entre estos dos métodos, proponiéndose una gran variedad de métodos Wavelet-Multimalla. Muchos de estos trabajos describen algoritmos multimalla que usan la transformada wavelet como base para la discretización en la resolución de ecuaciones en derivadas parciales. Estos métodos pueden ser muy eficientes pero no son generales debido a que, para aplicarlos, se necesita un conocimiento detallado de la discretización y del problema que le da origen. Por otro lado, existen algoritmos tipo multimalla, que se pueden aplicar a cualquier sistema lineal no importa de donde surja. Solo se necesita conocer la matriz y la parte derecha del sistema. Estos métodos multimalla se conocen

como “algebraicos”.

En la presente tesis se tratarán los Métodos Wavelet Multimalla Algebraicos (Wavelet Algebraic Multigrid Methods, WAMG), algoritmos que combinan los métodos multimalla con las wavelet, y que no necesitan de ningún conocimiento del problema a resolver; solo la matriz de coeficientes y la parte derecha del sistema.

Desde nuestro punto de vista el algoritmo principal, wavelet multimalla algebraico, fue propuesto por Wang, Dutton y Hao en [34] y retomado por Pereira y otros autores en [52]; otro algoritmo importante fue el propuesto por D. Leon en [40]. Sin embargo, hay variantes muy interesantes de estos algoritmos que no se discuten en los artículos citados. En esta tesis propondremos dos variantes de los algoritmos WAMG. La primera se basa en la descomposición inducida del sistema lineal al aplicar la DWT. La segunda reduce el costo de los ciclos del algoritmo multimalla saltando operaciones en algunos niveles o mallas.

Una observación importante (la cual discutiremos en detalle en el capítulo 6) de estos métodos multimallas basados en la transformada wavelet, es que son especialmente adecuados para problemas en los que se necesita resolver varios sistemas de ecuaciones con matrices desplazadas $A - hI$. Esto resulta muy relevante para la solución de Ecuaciones en Derivadas Parciales (PDEs) y sistemas de ecuaciones diferenciales ordinarias dependientes del tiempo.

1.3. Objetivos

Objetivo General

Diseñar, implementar y evaluar algoritmos secuenciales y paralelos, eficientes y robustos para el cálculo de la DWT; analizar posibles aplicaciones en la ingeniería en general y en el álgebra lineal numérica en particular.

Objetivos específicos

- Proponer algoritmos paralelos para el cálculo de la DWT en matrices densas, basándonos en los ya existentes que están orientados fundamentalmente al tratamiento de imágenes.
- Proponer algoritmos secuenciales y paralelos para el cálculo de la DWT en matrices dispersas.

- Proponer algoritmos de reordenamiento de las matrices dispersas con el objetivo de disminuir el rellenado (*fill-in*) cuando se calcule la DWT.
- Analizar posibles aplicaciones de la DWT en la construcción de precondicionadores para resolver sistemas de ecuaciones lineales.
- Analizar las posibles aplicaciones de la DWT, para acelerar la convergencia de los métodos Multimalla (*MultiGrid*).

1.4. Estructura del documento

En el **capítulo 2** describimos teóricamente la transformada wavelet y la transformada wavelet packet (PWT).

En el **capítulo 3** nos centraremos en la paralelización de la DWT: Extendiendo los trabajos de Nielsen y Hegland [50]. En particular propondremos una variante para utilizar la DWT paralela en conjunto con la librería ScaLAPACK, y como podemos adaptar eficientemente el cálculo de la transformada cuando se usa la distribución de datos que ha demostrado ser la más eficiente para esta librería.

En el **capítulo 4** propondremos variantes del calculo de la DWT cuando se trabaja con matrices dispersas. El objetivo principal es el de obtener algoritmos eficientes, tanto secuenciales como paralelos, para el cálculo de la DWT sobre matrices dispersas almacenadas en formato CSR. También se proponen variantes de reordenación de las matrices que minimicen el efecto de relleno cuando se efectúa el calculo de la DWT.

En el **capítulo 5** presentamos el algoritmo de precondicionado expuesto en [16] y proponemos una variante de paralelización. Además describimos el álgebra no estándar y proponemos un precondicionador no estándar similar a la LU incompleta.

En el **capítulo 6** se introducen los métodos multimalla algebraicos (WAMG). Y se proponen dos nuevas variantes de los mismos.

En el **capítulo 7** se plantean las principales conclusiones, aportes y trabajos futuros que se derivan del desarrollo de la tesis.

Capítulo 2

Descripción de la transformada wavelet

2.1. Introducción a las transformadas

Tradicionalmente las señales (o funciones) se representan en el dominio del tiempo (o del espacio), ya sea en forma de funciones continuas (señales analógicas) o como muestreos discretos en el tiempo (señales digitales). La transformación desde el dominio del tiempo al dominio de la frecuencia revela otros aspectos de las señales como la magnitud de cambio de los componentes de la señal en el tiempo o el espacio. La transformada de Fourier es una herramienta con la capacidad de representar señales o funciones tanto en el dominio del tiempo como en el dominio de la frecuencia. El análisis de Fourier ha dominado el procesamiento y análisis de señales en el último medio siglo, debido principalmente a que la frecuencia es la llave para el procesamiento y filtrado de señales físicas.

Sin embargo, ciertas señales, cuya amplitud tiene fuerte variación en el tiempo o señales cuyo contenido de frecuencia es variable de un instante de tiempo a otro (señales no estacionarias) no pueden ser analizadas a fondo mediante la transformada de Fourier, debido a ciertas limitaciones de este análisis en el campo tiempo-frecuencia. La Transformada de Fourier detecta la presencia de una determinada frecuencia pero no brinda información acerca de la evolución en el tiempo de las características espectrales de la señal. Muchos aspectos temporales de la señal, tales como el comienzo y el fin de una señal finita y el instante de aparición de una singularidad en una señal

transitoria, no pueden ser analizados adecuadamente por el análisis de Fourier. Es en estos términos de análisis donde entra en juego una nueva herramienta matemática llamada Transformada Wavelet.

El término *wavelet* se define como una "pequeña onda" o función localizable en el tiempo. Vista desde una perspectiva del análisis o procesamiento de señales puede ser considerada como una herramienta matemática para la representación y segmentación de señales, análisis tiempo-frecuencia e implementación de algoritmos sencillos y rápidos desde el punto de vista computacional. Las características propias de la transformada wavelet nos otorgan la posibilidad de representar señales en diferentes niveles de resolución, representar en forma eficiente señales con variaciones de picos abruptos, así como analizar señales no estacionarias. Nos permite conocer el contenido en frecuencia de una señal y cuándo estas componentes de frecuencia se encuentran presentes en la señal.

2.2. La transformada de Fourier

La transformada de Fourier, en esencia, descompone o expande una señal o función en senos y cosenos de diferentes frecuencias cuya suma corresponde a la señal original. En otras palabras es capaz de distinguir las diferentes componentes de frecuencia de la señal, y sus respectivas amplitudes. Puede verse como una técnica matemática para transformar el punto de vista de una señal desde la base de tiempo a la base de la frecuencia, tal como se representa esquemáticamente en la figura 2.1.

La transformada de Fourier de una función del tiempo $f(t)$ se define como:

$$F(w) = \int_{-\infty}^{+\infty} f(t)e^{iwt}dt \tag{2.1}$$

y la transformada inversa de Fourier, como:

$$f(t) = \int_{-\infty}^{+\infty} f(w)e^{-iwt}dw \tag{2.2}$$

La transformada de Fourier puede obtener una representación en el dominio de la frecuencia de una señal que se encuentra originalmente en el dominio del tiempo.

Figura 2.1: Esquema de transformación de la transformada de Fourier.

La relación existente entre la representación de la señal original a través de funciones senoidales y cosenoidales y la exponencial, proviene de la definición de la identidad de Euler:

$$e^{iwt} = \cos(wt) + sen(wt)i \tag{2.3}$$

$$e^{-iwt} = \cos(wt) - sen(wt)i \tag{2.4}$$

Mediante esta función exponencial es posible formar un conjunto de funciones ortogonales de la forma:

$$\{e^{inwt} : n = 0, \pm 1, \pm 2, \pm 3, ...\} \tag{2.5}$$

sobre un intervalo $[t_0, t_0 + T]$ y por lo tanto podemos descomponer o expandir la señal original (en el dominio del tiempo) de la siguiente manera:

$$f(t) = \sum_{-\infty}^{+\infty} F_n e^{-inwt} \tag{2.6}$$

Estas funciones exponenciales pueden ser referidas como las funciones bases de la transformada de Fourier, y debido a su propiedad de ortogonalidad, es posible obtener los valores o coeficientes F_n como términos de semejanza entre la señal original y la función exponencial.

$$F_n = \frac{1}{T} \int_{t_0}^{t_0+T} f(t) e^{inwt} dt \tag{2.7}$$

2.2.1. La transformada discreta de Fourier

Cuando hablamos de procesamiento digital de señales en forma automática nos vemos enfrentados al uso de un computador. Debido a que los computadores trabajan sólo con datos discretos, el cálculo numérico de la transformada de Fourier de $f(t)$ requiere valores discretos o muestreos de $f(t)$, o sea valores de la forma f_k con $[k = 0, 1, 2 \ldots]$. Esto significa que mediante el uso de un computador es posible calcular la transformada $F(w)$ sólo para valores discretos de w, es decir, obtendremos valores de la transformada de la forma F_n con $[n = 0, 1, 2, \ldots]$. Nos podemos referir a f como una señal en el tiempo (ya no como función).

Por lo tanto, supongamos que $f(t)$ es una señal periódica de período T y que sólo conocemos sus valores en N puntos igualmente espaciados en el tiempo. Entonces, si $f(kT_s)$ corresponde a la k-ésima muestra de $f(t)$ y $F(nw_s)$, donde $ws = 2\pi f_s$ (f_s es la frecuencia con la que se realizan las muestras) corresponde al n-ésima muestra de $F(w)$, y además definimos a N como el número de muestras de la señal o longitud de la señal, podemos reescribir la Transformada de Fourier, de una señal de período T, en su forma discreta como:

$$F_n = \sum_{k=0}^{N-1} f_k e^{\frac{i2\pi kn}{N}} \quad n, k = 0, 1, 2, ..., N-1 \tag{2.8}$$

De donde se puede deducir que F_n tiene período N al igual que f_k. Con lo que podemos decir que el conjunto de coeficientes (F_n) con $n = 0, 1, 2, \ldots, N-1$ es denominado la Transformada Discreta de Fourier (DFT) de los valores muestreados (f_k) con $k = 0, 1, 2, \ldots, N-1$.

Teniendo los coeficientes F_n también podemos obtener los valores de f_k, de la siguiente manera:

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{\frac{-i2\pi kn}{N}} \quad n, k = 0, 1, 2, ..., N-1 \tag{2.9}$$

El cálculo de F_n, $n = 0, 1, 2, \ldots, N-1$ y f_k con $k = 0, 1, 2, \ldots, N-1$ y haciendo $W_N^{kn} = e^{\frac{i2\pi\, k\, n}{N}}$, se puede representar en forma matricial como sigue:

$$\begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & ... & 1 \\ 1 & W_N & ... & W_N^{N-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{N-1} & ... & W_N^{(N-1)*(N-1)} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix} \tag{2.10}$$

y

$$\begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & ... & 1 \\ 1 & W_N^{-1} & ... & W_N^{-(N-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & W_N^{-(N-1)} & ... & W_N^{-(N-1)*(N-1)} \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_n \end{bmatrix} \tag{2.11}$$

A la matriz obtenida se le llama matriz de Fourier. De forma equivalente:

$$F_n = \sum_{k=0}^{N-1} f_k W_N^{kn} \tag{2.12}$$

$$f_k = \frac{1}{n} \sum_{n=0}^{N-1} F_n W_N^{-kn} \tag{2.13}$$

La matriz de Fourier es densa, es decir, todos sus elementos son distintos de cero. Por lo tanto, el número de multiplicaciones que se deben realizar para la obtención de la DFT de una señal de longitud N, es de N^2 multiplicaciones. Se deben evaluar N términos de series de Fourier sobre N puntos.

2.2.2. Transformada rápida de Fourier (FFT)

Con el fin de implementar en forma práctica la Transformada Discreta de Fourier mediante el uso de computadores, se hace necesario disminuir su costo computacional. A mediados de la década del sesenta J.W Cooley y J.W Tukey en [22] desarrollaron un algoritmo denominado la Transformada Rápida de Fourier (FFT). La FFT elimina información redundante que existe en la DFT, debido a que explota las propiedades de periodicidad y simetría del factor de fase W_N. Estas propiedades son:

$$W_N^{k+\frac{N}{2}} = -W_N^k \quad \text{Simetría}$$

$$W_N^{k+\frac{N}{2}} = W_N^k \quad \text{Periodicidad}$$

Existen básicamente dos tipos de algoritmos FFT:

FFT diezmado de tiempo El algoritmo de diezmado de tiempo toma la totalidad

de los datos de entrada f_k y los separa en sus muestras pares y sus muestras impares, cada una con una longitud igual a la mitad de la longitud de la señal original.

FFT diezmado en frecuencia El algoritmo de diezmado de frecuencia al igual que el diezmado de tiempo separa la señal original de longitud N en dos secuencias con una longitud igual a $\frac{N}{2}$; la diferencia con el diezmado en tiempo reside en que una secuencia contiene la primera mitad de las muestras ($k = 0, 1, \ldots, \frac{N}{2} - 1$) y la otra secuencia contiene la otra mitad ($k = \frac{N}{2}, \frac{N}{2} + 1, \ldots, N$).

Otro punto importante es que el algoritmo FFT es más eficiente cuando se aplica sobre una señal donde el número de muestras N es una potencia de 2. El principio de la FFT se basa en el método "divide y vencerás", ya que divide la señal de N puntos en dos secuencias de datos de $\frac{N}{2}$ puntos, la señal de entrada o salida respectivamente, según el tipo de algoritmo [43].

2.2.3. Transformada corta de Fourier (STFT)

Como ya es sabido, la transformada de Fourier constituye una herramienta mediante la cual podemos obtener información sobre cómo está distribuida la energía de una señal a través de sus distintas componentes de frecuencia, es decir, podemos conocer todas las componentes de frecuencia existentes en la señal y sus respectivos aportes energéticos. Todo lo anterior se puede resumir diciendo que la transformada de Fourier tiene una perfecta resolución en frecuencia lo que la hace una herramienta muy útil para el análisis de señales estacionarias. Sin embargo, ella no puede ser aplicada con el objeto de obtener información precisa sobre la localización temporal (espacial) las diferentes componentes de frecuencia en la señal, como es el caso de señales quasi-estacionarias o no estacionarias cuyo contenido espectral varía con el tiempo. En otras palabras, la transformada de Fourier posee una muy pobre resolución en tiempo. En un esfuerzo por corregir esta deficiencia, en 1946 Denis Gabor adaptó la Transformada de Fourier para poder analizar una pequeña sección de la señal en un determinado tiempo (mediante una especie de ventana). Esta adaptación es la que se conoce como STFT, la cual lleva una señal del plano del tiempo al plano bidimensional de tiempo y frecuencia, tal como se presenta esquemáticamente en la figura 2.2

La forma de dividir la señal se realiza mediante lo que llamaremos una función-

tiempo-ventana $h(t)$, cuyo ancho o soporte corresponde a la longitud de cada segmentación de la señal. Con la función ventana encuadramos la señal alrededor de un instante de tiempo τ y calculamos su transformada de Fourier, luego trasladamos la función ventana hasta que no se superpone con la anterior, cubriendo una nueva porción de la señal a la que volvemos a calcular su transformada de Fourier. Este proceso se repite hasta que se haya cubierto la totalidad de la señal.

Figura 2.2: Esquema de transformación de la transformada de Fourier por intervalos (STFT).

El resultado de lo expresado anteriormente se define en forma matemática de la siguiente manera:

$$STFT(t,w) = \int_{-\infty}^{+\infty} x(t)h^*(\tau - t)e^{-iwt}dt \tag{2.14}$$

Si consideramos a $h(t)$ como una función ventana de valores sólo reales no complejos de tal manera que $h(-t) = h^*(t)$ entonces nos queda:

$$STFT(t,\xi) = \int_{-\infty}^{+\infty} x(t)h(t - \tau)e^{-i\xi t}dt \tag{2.15}$$

La expresión anterior calcula el producto interno entre la señal y la función tiempo-ventana trasladada y modulada.

Es importante mencionar que la STFT representa una especie de compromiso entre el dominio del tiempo y el de la frecuencia de una señal, ya que provee algo de información acerca de cuándo y a qué frecuencia de una señal ocurre un determinado evento. Sin embargo, solamente se puede obtener dicha información con una precisión limitada, la cual está acotada por el tamaño de la ventana. Mientras que el compromiso entre la información del tiempo y la frecuencia puede resultar útil, el inconveniente surge debido a que una vez que se escoge un determinado tamaño para la ventana

de tiempo, dicha ventana es la misma para todas las frecuencias. Muchas señales requieren un acercamiento más flexible, de modo tal que sea posible variar el tamaño de la ventana para determinar con mayor precisión el tiempo o la frecuencia.

2.3. Transformada wavelet

El análisis mediante wavelet representa el paso lógico siguiente a la STFT: una técnica mediante ventanas con regiones de tamaño variable. El análisis wavelet permite el uso de intervalos grandes de tiempo en aquellos segmentos en los que se requiere mayor precisión en baja frecuencia, y regiones más pequeñas donde se requiere información en alta frecuencia. Esta idea es la que se muestra en forma esquemática en la figura 2.3.

Figura 2.3: Esquema de transformación de la transformada wavelet.

2.3.1. Transformada Wavelet Continua (CWT)

La transformada wavelet continua intenta expresar una señal $x(t)$ continua en el tiempo, mediante una expansión de términos o coeficientes proporcionales al producto interno entre la señal y diferentes versiones escaladas y trasladadas de una función prototipo $\psi(t)$ más conocida como wavelet madre. Asumiendo que tanto la señal como la nueva función $\psi(t)$ son de energía finita, entonces podemos definir:

$$CWT(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} x(t)\psi(\frac{t-b}{a})\, dt \tag{2.16}$$

Como se puede observar han aparecido dos nuevas variables a y b. La variable

a controla el ancho o soporte efectivo de la función $\psi(t)$, y la variable b nos da la ubicación en el dominio del tiempo de ψ.

Ahora bien, para que este análisis sea posible y además para poder lograr una perfecta reconstrucción de la señal a partir de la transformada, la función $\psi(t)$ debe cumplir con la condición de admisibilidad [15]. El cumplimiento de esta condición significa que el valor medio de $\psi(t)$ es igual a 0, lo que a su vez implica obligatoriamente que $\psi(t)$ tenga valores tanto positivos como negativos, es decir, que sea una onda. Además, como $\psi(t)$ es una función que "ventaniza" la señal sobre un intervalo de tiempo dado por a alrededor de un punto $t = b$, se observa intuitivamente que $\psi(t)$ es de soporte compacto, es decir, es una onda definida sobre un intervalo de tiempo finito, de ahí el porque de su nombre, wavelet u "ondita".

Mediante la variable de escala se puede comprimir ($|a| < 1$) o dilatar ($|a| > 1$) la función $\psi(t)$, lo que dará el grado de resolución con el cual se esté analizando la señal. Por definición la Transformada Continua Wavelet es más una representación tiempo-escala que una representación tiempo-frecuencia. En particular, para valores pequeños de a la CWT obtiene información de $x(t)$ que está esencialmente localizada en el dominio del tiempo mientras que para valores grandes de a la CWT obtiene información de $x(t)$ que está localizada en el dominio de la frecuencia. En otras palabras, para escalas pequeñas la CWT nos entrega una buena resolución en el dominio del tiempo, mientras que para escalas grandes la CWT nos entrega una buena resolución en el dominio de la frecuencia. Cuando el valor de la variable a cambia, tanto la duración como el ancho de banda de la wavelet cambian pero su forma se mantiene igual. En lo anteriormente dicho se encuentra la diferencia principal entre la CWT y la STFT, ya que la primera ocupa ventanas de corta duración para altas frecuencias y ventanas de larga duración para bajas frecuencias, mientras que la STFT ocupa una sola ventana con la misma duración tanto para altas frecuencias como para bajas frecuencias.

La Transformada Continua Wavelet es un transformada reversible siempre y cuando la función $x(t)$ cumpla con la condición de admisibilidad. La reconstrucción es posible usando la siguiente formula:

$$x(t) = \frac{1}{c_\psi^2} \int_a \int_b CWT_x^\psi(a,b) \frac{1}{a^2} \psi(\frac{t-b}{a}) dbda \tag{2.17}$$

2.3.2. Cálculo de la transformada wavelet continua

En este punto se presenta en forma cualitativa un método sencillo para obtener la transformada wavelet de una determinada señal.

Antes de describir los pasos a seguir, debe elegirse una función wavelet, la que será la wavelet madre y servirá como prototipo para todas las ventanas que se emplean en el proceso. Existe una importante cantidad de familias de funciones wavelets que han probado ser especialmente útiles; entre ellas destacan la Haar, Daubechies, Biortogonal, Coiflets, Symlets, Morlet, Sombrero mexicano y Meyer, entre otras.

Los pasos a seguir para determinar la transformada wavelet continua de una señal tal y como se describe en [48] son:

1. Comenzando con un determinado valor de a (escala), por ejemplo 1, para la señal wavelet, se ubica ésta al comienzo de la señal a analizar (en $t = 0$). Luego, se multiplican entre sí ambas señales y el resultado se integra sobre todo el espacio de tiempo. El resultado de dicha integral se divide por la raíz cuadrada de a, con el objeto de normalizar la energía y de este modo obtener una función Transformada con la misma energía a cualquier escala. Este resultado es el valor de la transformación wavelet en tiempo cero y $a = 1$. Es importante mencionar que este resultado indica cuán correlacionada está la wavelet con el segmento de la señal original. Lógicamente, el resultado dependerá de la elección de la función wavelet. Este paso queda representado en la figura 2.4.

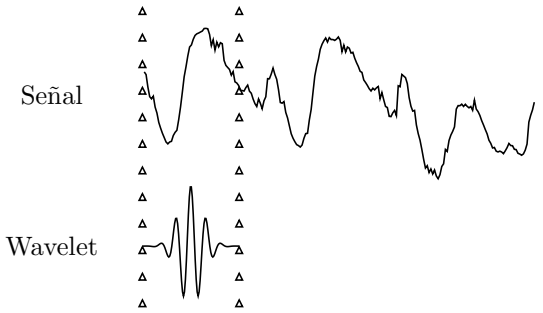


Figura 2.4: Paso 1 para la obtención de la transformada wavelet continua.

2. La función wavelet (en la misma escala, por ejemplo $a = 1$) se traslada en tiempo (hacia la derecha) en b, y se vuelve a realizar el procedimiento descrito en el

paso 1. Se debe repetir esto hasta llegar al final de la señal a analizar. Este paso queda ilustrado en la figura 2.5

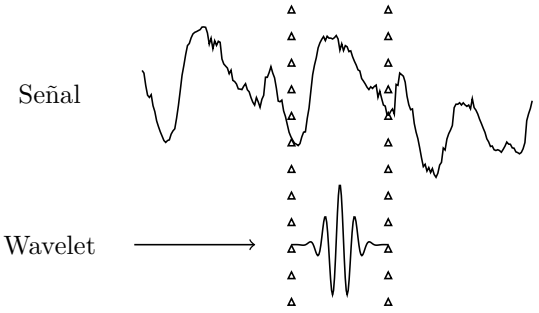


Figura 2.5: Paso 2 para la obtención de la transformada wavelet continua.

3. Se varía el valor de a (escala) y se vuelven a realizar los pasos 1 y 2 hasta haber barrido todo el rango de frecuencias que se desea analizar. Nótese que dado que se trata de una Transformación continua, tanto la traslación en tiempo como la variación de escala deberían realizarse de manera continua. Sin embargo, si es necesario obtener la transformada wavelet por medios computacionales, la condición anterior se reduce a considerar un paso suficientemente pequeño. Cada cálculo para un determinado valor de a llena la correspondiente fila de datos del plano tiempo-escala. Este paso se ilustra en la figura 2.6

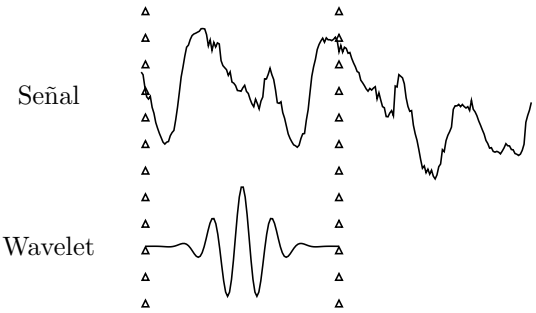


Figura 2.6: Paso 3 para la obtención de la transformada wavelet continua.

Cuando se haya completado el cálculo para todos los valores de a, se habrá obtenido la transformada wavelet continua de la señal.

Como se ilustra en el algoritmo anterior, la variable b controla la ubicación de la

función en el espacio de tiempo permitiendo deslizar $\psi(t)$ sobre el intervalo de tiempo en el que se haya definido $x(t)$. Un punto importante es que la función wavelet se traslada cubriendo toda la señal para cada valor de a, es decir, si la escala escogida es pequeña habrá más traslaciones de $\psi(t)$ que si la escala escogida es grande.

La continuidad de la CWT reside en que tanto la variable de escala como la variable de traslación varían de forma continua. Sin embargo, en términos de cálculo computacional es imprescindible discretizar la transformada, y la suposición más lógica es que tanto los valores de escala como traslación sean discretos. Adelantándonos un poco a lo que es la Transformada Wavelet Discreta, la forma más común de discretizar los valores de a y b es utilizar una red diádica [45], es decir, $a = 2^{-j}$ y $b = k2^{-j}$ con $j, k \in Z$, de tal manera que el conjunto de funciones:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a}), \quad a, b \in \mathbb{R},\, a \neq 0 \tag{2.18}$$

se transforma en el conjunto de funciones:

$$\psi_{j,k} = 2^{\frac{j}{2}}\psi(2^j - k), \quad j, k \in \mathbb{Z} \tag{2.19}$$

que corresponde a la versión diadicamente discretizada de la función wavelet.

2.3.3. Análisis multiresolución

Para realizar un Análisis Multiresolución (MRA) se necesita de una función base $\phi = \phi(t)$ bien localizada tanto en tiempo como en frecuencia. Una de las más cómodas es la función indicador $\phi = 1_{[0,1)}$ definida por la fórmula siguiente:

$$1_{[0,1)} \stackrel{\text{def}}{=} \begin{cases} 1\ si & 0 \leqslant t \leqslant 1 \\ 0\ si & t < 0 \text{ ó } \geqslant 1 \end{cases} \tag{2.20}$$

ϕ está evidentemente localizada en 0, tiene escala unitaria debido al hecho de que es diferente de 0 en un intervalo de longitud 1. Puede ser trasladada a una nueva posición, el intervalo $[k, k+1)$, usando la formula siguiente:

$$\phi(t) = 1_{[0,1)} \stackrel{\text{def}}{=} \begin{cases} 1\ si & k \leqslant t \leqslant k+1 \\ 0\ si & t < k \text{ ó } t \geqslant k+1 \end{cases} \qquad k = \pm 1, \pm 2, ... \tag{2.21}$$

Cualquier f puede describirse con resolución unitaria, usando la formula:

$$f_0(t) = \sum_k a_k \phi_k(t) \tag{2.22}$$

La función "Escalonada" f_0 es constante en el intervalo $[k, k+1)$ para cualquier k, donde toma el valor $a_k = \int f(t)\phi_k(t)\,dt$, el valor promedio de f en dicho intervalo. Si f no cambia mucho en ese intervalo entonces f_0 aproxima a f razonablemente bien. Un ejemplo de ello se muestra en la figura 2.7

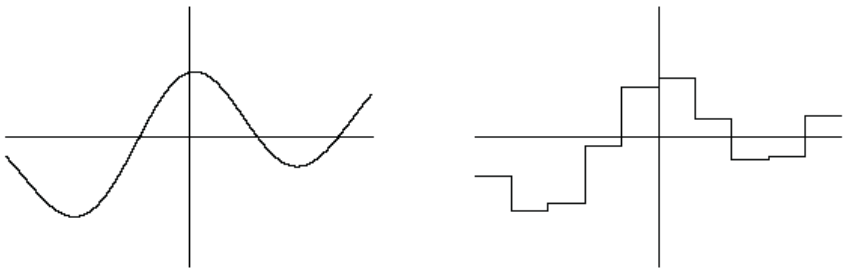


Figura 2.7: La función original f (izquierda) y su representación con resolución unitaria (derecha).

Para obtener una mejor aproximación de f podemos aumentar la resolución en la ecuación (2.22) usando una versión más pequeña de ϕ_k. Este efecto lo podemos lograr dilatando la variable t por un factor de 2:

$$\phi_{j,k}(t) \stackrel{\text{def}}{=} \phi(2^j t - k); \quad j = 1, 2, ...; \quad k = \pm 1, \pm 2, ... \tag{2.23}$$

En esta nueva notación de índices dobles denotamos $\phi_{0,0}$ por la ϕ original y la antigua ϕ_k por $\phi_{0,k}$. Una resolución unitaria corresponde a $j = 0$; la resolución se duplica en la medida que j aumenta. En la figura 2.8 se muestran tres aproximaciones de f correspondientes a $j = 0, 1, 2$.

Un análisis multiresolución correcto de f debe cumplir la propiedad de que $|f - f_j|$ tienda a 0 cuando j tienda a ∞. Obviamente esta propiedad solo depende de la elección de ϕ, debido a que ϕ determina completamente a f_j para todos los $j = 1, 2, ...$ La función indicador $1_{[0,1)}$ cumple esta propiedad, sin embargo hay otras opciones para elegir ϕ que brindan una mayor velocidad de aproximación. Una forma de generar

este tipo de funciones es buscar las soluciones de la siguiente ecuación:

$$\phi(t) = \sum_k h_k \phi(2t-k) \tag{2.24}$$

$$\int \phi(t)\,dt = 1 \tag{2.25}$$

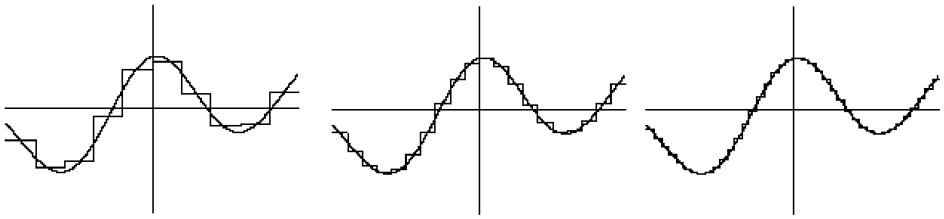


Figura 2.8: Tres aproximaciones de f: con resolución unitaria f_0 (izquierda), con resolución doble f_1 (centro) y con resolución cuádruple f_2 (derecha).

La solución de este par de ecuaciones se llama usualmente *función escala*. Dicha función está determinada por los números $\{h_k\}$, llamados coeficientes del filtro *pasa-bajo*. Se obtiene una solución única para ϕ si se satisfacen las siguientes condiciones:

$\{h_k\}$ tiene un número finito de coeficientes diferentes de cero, y

$$\sum\nolimits_k h_k = 2; \tag{2.26}$$

$$\sum_k h_k h_{k+2n} = 0, \quad \text{para } n \neq 0; \qquad \sum_k h_k^2 = 2. \tag{2.27}$$

La función indicador $1_{[0,1)}$ es solución de la ecuaciones (2.24) y (2.25) con $h_0 = h_1 = 1$ y $h_k = 0$ para todo $k \neq 0, 1$. Esta es la secuencia de filtros más simple posible.

Cualquier secuencia que satisfaga (2.26) y (2.27) produce una función ϕ que es *ortogonal* a sus traslaciones enteras:

$$\int \phi(t)\phi(t-k)\,dt = 0, \ \text{con}\, k \in \mathbb{Z}, k \neq 0; \quad \int \phi^2(t) = 1 \tag{2.28}$$

En estos casos, la fórmula que expande la función f para una resolución j se asemeja

a la fórmula en series de Fourier:

$$f_j(t) = \sum_k a_k \phi_{j,k}(t); \quad a_k = 2^j \int f(t)\phi_{j,k}(t)\, dt. \tag{2.29}$$

El factor 2^j se necesita debido al adelgazamiento de ϕ en la resolución j. No obstante, dicha expansión difiere de la lista de muestras de f en puntos separados en 2^{-j}, lo que hace que provea de poca compresión. Hay una forma de codificar solamente las diferencias entre las aproximaciones sucesivas de f.

$$f = f_0 + (f_1 - f_0) + (f_2 - f_1) + (f_3 - f_2) + ... \tag{2.30}$$

Esta codificación funciona debido a que $|f - f_j|$ tiende a 0 cuando j tiende a ∞, de forma que los coeficientes necesarios para codificar $f_j - f_{j-1}$ deben ser más pequeños en la medida en que aumenta j. Estas diferencias decrecen más rápido para funciones suaves. Además, el número de coeficientes necesarios para codificar $f_j - f_{j-1}$ es proporcional a 2^j, de forma que para hacer aproximaciones cercanas de funciones f suaves solo se necesitarían unos pocos coeficientes de diferenciación no significativos. El truco es expandir las diferencias como superposición de otra función:

$$f(t) = \sum_k c_{0,k}\phi_{0,k}(t) + \sum_k d_{0,k}\psi_{0,k}(t) + \sum_k d_{1,k}\psi_{1,k}(t) + \sum_k d_{2,k}\psi_{2,k}(t) + ... \tag{2.31}$$

Donde $c_{0,k}$ y $d_{j,k}$, $j = 0, 1, 2...$ son amplitudes que se determinan para cada función f. Esta fórmula se cumple debido a que ϕ cumple con (2.24) y (2.25). Como resultado de esto existe una función que recibe el nombre de Wavelet Madre, que genera los componentes en la ecuación (2.31) a base de dilataciones y desplazamientos:

$$\psi(t) = \sum_k g_k \psi(2t - k); \quad g_k = (-1)^k h_{1-k}(t), \quad k = 0, \pm 1, \pm 2, ... \tag{2.32}$$

Nótese que el filtro *pasa-bajo* $\{h_k\}$ determina el filtro *pasa-alto* $\{g_k\}$ que genera $\psi(t)$.

Las dos secuencias $\{h_k\}$ y $\{g_k\}$ se denominan filtros de cuadratura conjugados, con los cuales es posible encontrar eficientemente los coeficientes $c_{0,k}$ y $d_{j,k}$ de la ecuación (2.31).

2.4. Transformada Wavelet Discreta (DWT)

Para aplicar la transformada wavelet a una serie de datos numéricos, se hace necesario implementar una transformada discreta. La idea fue desarrollada por Mallat en 1989 [44], y se basa en el MRA visto en el epígrafe 2.3.3. Mallat diseñó un algoritmo basado en un banco de filtros que permite obtener una transformada wavelet, de forma poco costosa, a partir de los datos de interés. Dichos bancos de filtros están basados en las ecuaciones (2.24) y (2.32).

2.4.1. Filtros de un nivel

En la mayoría de las señales son las componentes de baja frecuencia las que le otorgan a la señal la mayor parte de su información, o bien, le dan una especie de identidad a la señal, mientras que las componentes de alta frecuencia se encargan de incorporar características más particulares. Es por ello que se subdividen las componentes de una señal en dos categorías:

- Aproximaciones (baja frecuencia)
- Detalles (alta frecuencia)

A continuación, surge la idea de separar estas dos componentes a través de filtros. Lo anterior queda ejemplificado en el diagrama de la figura 2.9, donde S es la señal que se desea analizar, A la salida del pasa-bajos y D la salida del filtro pasa-altos. Naturalmente, los filtros son diseñados de tal manera que sean complementarios, es decir, la suma de A y D debe ser S. Si se diseñaran los filtros en forma muy separada se perdería información, o en caso contrario se estaría amplificando la banda de entrecruzamiento. Sin embargo, este procedimiento tiene la desventaja que se duplica el número de datos originales, pues por cada muestra de S se genera un par de muestras (A, D), por lo que el costo matemático y computacional se incrementa. Para remediar esta dificultad se propone un método que guarda la mitad de los puntos (A, D), sin perder en ello información de la señal S. Este procedimiento es conocido como **submuestreo** o **decimación**. La idea se ilustra en la figura 2.10.

Los círculos con flechas representan la eliminación de datos o submuestreo. Luego, cD y cA son los nuevos coeficientes obtenidos de la etapa de filtración. Intuitivamente se puede concluir que al tener cD y cA, en conjunto, se tiene la misma cantidad de

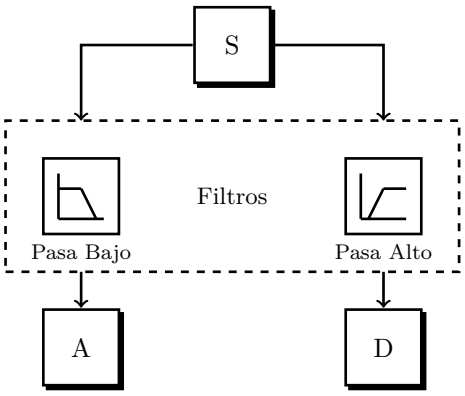


Figura 2.9: Diagrama de descomposición de señales usando bancos de filtros.

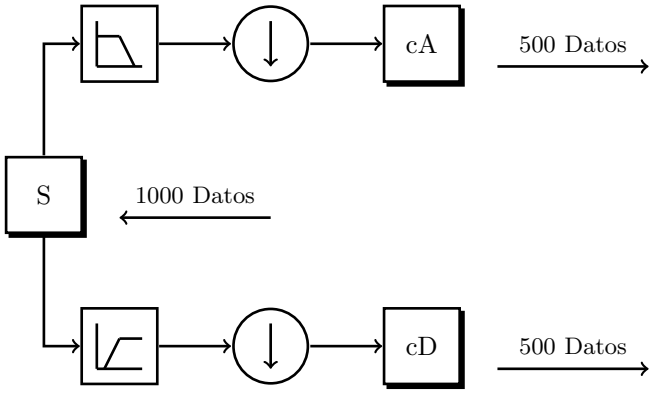


Figura 2.10: Diagrama de descomposición de señales usando bancos de filtros y submuestreo.

datos que las de la señal original S, y se ha mantenido la información necesaria. En la figura 2.10 se ejemplifica la idea para una señal S de 1000 datos, obteniéndose en la salida dos series de aproximadamente 500 datos cada una.

Para muchas señales de mayor complejidad, no basta con dos bandas de frecuencias (alta y baja), sino que más bien debe hacerse una descomposición de más niveles para poder separar las características y poder analizarlas independientemente. Surge la idea entonces de filtros multiniveles.

2.4.2. Filtros multinivel

Para realizar la motivación expuesta en el punto anterior, basta con iterar el proceso de filtrado, es decir, aplicar el mismo procedimiento a las señales de salida de la primera etapa, y así sucesivamente hasta el nivel de precisión que se desee. Lo anterior da origen a una descomposición multinivel conocida como ramificación o árbol de descomposición wavelet, cuya idea es expuesta en la figura 2.11.

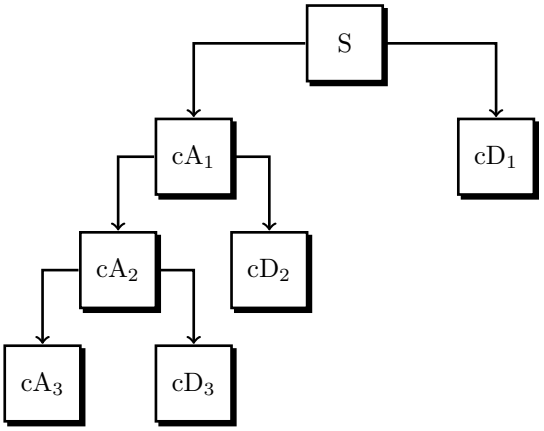


Figura 2.11: Árbol de descomposición wavelet.

Nótese que cD_1 resulta ser la componente de más alta frecuencia de la señal, y cA_3 la de menor frecuencia. Al ser descompuesta la señal en mayor cantidad de bandas de frecuencia se posee una información más detallada acerca de S, esta metodología es conocida como multiresolución. Surge de forma inmediata la inquietud acerca del diseño del algoritmo, relativo al número de niveles a utilizar. Podría suponerse, de manera intuitiva, que se obtienen resultados óptimos con un mayor número de niveles de descomposición, pero, esto no siempre es así. En [23] se recomienda una ramificación que vaya de acuerdo a la naturaleza de la señal a estudiar.

2.4.3. Reconstrucción wavelet.

En los puntos anteriores se explicó la base teórica acerca de la descomposición wavelet. Por tratarse de una transformación es deseable poder establecer su inversión, o en otras palabras, poder volver a la señal original a partir de los datos de salida del árbol. El proceso anterior es conocido como reconstrucción wavelet o Transformada

Inversa de Wavelet (discreta). La metodología sigue el razonamiento en dirección contraria, es decir, a partir de los coeficientes cA_i y cD_i (i depende del número de niveles) debe obtenerse S. Lo anterior queda ilustrado en la figura 2.12.

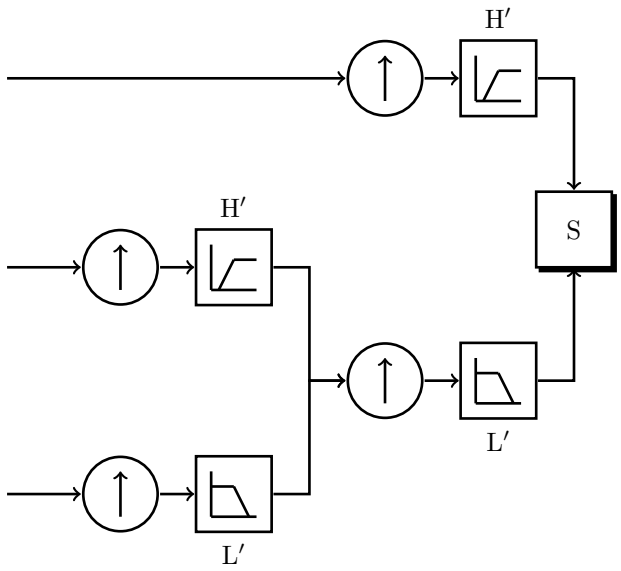


Figura 2.12: Esquema de reconstrucción wavelet.

En este caso se debe realizar una sobre-representación de la muestra para compensar el submuestreo realizado en el proceso de descomposición, luego pasa por un proceso de filtrado, para finalmente reconstruir S. La etapa crítica en este proceso es el filtrado, pues la elección de los filtros es determinante en la calidad de la reconstrucción. En [27] se discute el diseño, introduciendo filtros de descomposición H y L (para pasa-altos y pasa-bajos respectivamente), y sus filtros de reconstrucción correspondientes H' y L', diseñados a partir de una teoría llamada "Quadrature Mirror Filters", la cual no será analizada en mayor detalle en este trabajo.

2.4.4. Representación algebraica de la DWT-1D

El marco teórico de la DWT está fundamentado en el concepto de multiresolución visto en la sección 2.3.3.

Algebraicamente, el objetivo de aplicar la DWT a un vector es obtener un vector transformado que tiene en la mitad, conocida como parte alta, la misma informa-

ción de alta frecuencia que el vector original y en otra mitad, conocida como parte baja, la información de baja frecuencia. El éxito de la DWT en el procesamiento de señales reside en el hecho de que usualmente la mayor parte de la información de alta frecuencia es relativamente pequeña y puede descartarse, permitiendo así una compresión eficiente de los datos. Para poder completar esta operación la DWT debe ser invertible además de ser fácil y rápida de calcular ($O(n)$ operaciones para un vector que pertenezca a $\mathbb{R}^n$, ver sección 2.5).

Para extraer la información a partir de los datos, se le aplican filtros digitales. La información de baja frecuencia se obtiene aplicando un filtro G, denominado pasa-bajo, determinado por los coeficientes $\{g_i\}_{i=1}^{L}$ convolucionándolo con los datos; similarmente para obtener información de alta frecuencia se aplica un filtro de alta frecuencia H, denominado pasa-alto, $\{h_i\}_{i=1}^{L}$.

No cualquier par de filtros pasa-alto y pasa-bajo es válido para componer la DWT; aunque existen muchos de estos pares de filtros. Esto implica que la DWT no es una operación única ya que depende de la elección de los filtros. A la longitud de los filtros L, se le denomina orden de la transformada wavelet.

Este par de filtros se debe aplicar (convolucionar) sobre el vector de los datos, de tal forma que para obtener un nivel de la transformada wavelet sobre un vector v de longitud 2^k se siguen los siguientes pasos:

1. Convolución: Convolucionar ambos filtros con el vector v, obteniendo dos secuencias de longitud n.

2. Decimación o Submuestreo: Descartar los elementos que ocupan las posiciones impares en cada secuencia y unir las subsecuencias resultantes; poniendo en la parte alta del vector la convolucionada con el filtro H y en la parte baja la convolucionada con el filtro G.

El resultado es un vector de la misma dimensión n que el vector original y con la misma información de alta frecuencia en las primeras $\frac{n}{2}$ posiciones y de baja frecuencia en las últimas $\frac{n}{2}$ posiciones.

Si se representa de forma algebraica, estos dos pasos son equivalentes a multiplicar el vector v por la matriz (2.33)

$$W = \begin{pmatrix} H \\ G \end{pmatrix} \tag{2.33}$$

donde H y G, definidos en (2.34) y (2.35), son la representación matricial de los filtros de descomposición de alta y baja frecuencia respectivamente, que permiten representar la convolución y decimación de la señal en una operación única:

$$H = \begin{pmatrix} h_1 & h_2 & \cdots & h_L & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & h_1 & h_2 & \cdots & h_L & 0 & \cdots & \cdots & 0 \\ \vdots & \cdots & \cdots & \ddots & & \ddots & \ddots & & \vdots & \\ h_3 & h_4 & \cdots & h_L & 0 & \cdots & & 0 & h_1 & h_2 \end{pmatrix}_{\frac{n}{2}\times n} \tag{2.34}$$

$$G = \begin{pmatrix} g_1 & g_2 & \cdots & g_L & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & g_1 & g_2 & \cdots & g_L & 0 & \cdots & \cdots & 0 \\ \vdots & \cdots & \cdots & \ddots & & \ddots & \ddots & & \vdots & \\ g_3 & g_4 & \cdots & g_L & 0 & \cdots & & 0 & g_1 & g_2 \end{pmatrix}_{\frac{n}{2}\times n} \tag{2.35}$$

Finalmente la DWT-1D se obtiene al multiplicar la matriz (2.33) por un vector v. El resultado de esta operación se muestra en (2.36), donde los d_i son los componentes de alta frecuencia (componentes de detalle) del vector v y los s_i son los componentes de baja frecuencia (componentes de aproximación) de v.

$$Wv = \begin{pmatrix} Hv \\ Gv \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_{n/2} \\ s_1 \\ \vdots \\ s_{n/2} \end{pmatrix} \tag{2.36}$$

Para obtener L niveles de la transformada wavelet (ver sección 2.4.2) se aplica L veces a la parte de baja frecuencia del vector. Con frecuencia se denomina a esta transformada wavelet multinivel directamente transformada wavelet.

2.4.5. Representación algebraica de la DWT-2D

La DWT-2D, bidimensional, de una matriz se obtiene aplicando la DWT unidimensional (DWT-1D) en cada fila y columna de la matriz. De esta forma un nivel de la DWT-2D aplicada sobre una matriz $T_0 \in \mathbb{R}^{n\times n}$ se puede representar como

$\widetilde{T_0} = W_n T_0 W_n^T$, con W_n definida anteriormente. En este caso se obtiene la siguiente partición:

$$\widetilde{T_0} = W T_0 W^T = \begin{pmatrix} A_1 & B_1 \\ C_1 & T_1 \end{pmatrix} \tag{2.37}$$

donde $T_1 = G T_0 G^T,\ A_1 = H T_0 H^T,\ B_1 = H T_0 G^T,\ C_1 = G T_0 H^T$.

Si queremos aplicar más de un nivel de la DWT-2D, tenemos tres posibilidades de hacerlo. La llamada forma estándar, la no estándar y la trasformada wavelet no estándar extendida.

Transformada wavelet estándar

La transformada wavelet estándar se obtiene aplicado la DWT-1D en L niveles sobre las columnas y luego DWT-1D en L niveles sobre las filas, ver la figura 2.13.

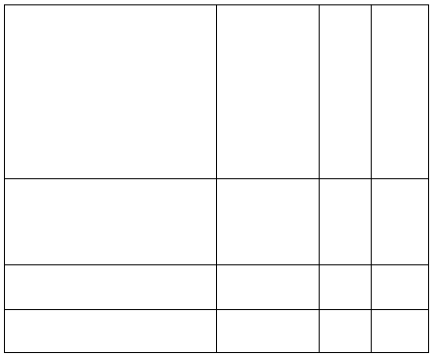


Figura 2.13: Matriz descompuesta en 3 niveles de la transformada wavelet estándar.

Para expresar esta operación de forma matricial definiremos las matrices Q_i, P_i y W_i de la siguiente forma:

$$Q_i = \begin{pmatrix} h_1 & h_2 & \cdots & h_L & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & h_1 & h_2 & \cdots & h_L & 0 & \cdots & \cdots & 0 \\ \vdots & \cdots & \cdots & \ddots & & \ddots & \ddots & & \vdots & \\ h_3 & h_4 & \cdots & h_L & 0 & \cdots & & 0 & h_1 & h_2 \end{pmatrix}_{\frac{n}{2^i} \times \frac{n}{2^{i-1}}} \tag{2.38}$$

$$P_i = \begin{pmatrix} g_1 & g_2 & \cdots & g_L & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & g_1 & g_2 & \cdots & g_L & 0 & \cdots & \cdots & 0 \\ \vdots & \cdots & \cdots & \ddots & & \ddots & \ddots & & \vdots & \\ g_3 & g_4 & \cdots & g_L & 0 & \cdots & & 0 & g_1 & g_2 \end{pmatrix}_{\frac{n}{2^i} \times \frac{n}{2^{i-1}}} \tag{2.39}$$

$$W_i = \begin{pmatrix} P_i & \\ & Q_i \end{pmatrix} \tag{2.40}$$

Sea

$$\overline{W}_j = \begin{pmatrix} I_{n-2^{j-1}} & \\ & W_{j-1} \end{pmatrix} \tag{2.41}$$

Con $\overline{W}_1 = W$, entonces quedaría

$$W = \overline{W}_L \overline{W}_{L-1} \ldots \overline{W}_1 \tag{2.42}$$

Luego la transforma wavelet estándar de una matriz T quedaría como $\tilde{T} = WTW^T$.

Transformada wavelet no estándar

El otro tipo de transformada es la forma no estándar [16] que se obtiene aplicando un nivel de la DWT sobre las columnas y un nivel sobre las filas, con lo que se obtiene una partición de la matriz del mismo modo que en (2.37). Entonces se aplica otra vez la DWT en un nivel por filas ($W_{n/2}$) y por columnas ($W_{n/2}^T$) sobre la matriz T_1; esta operación se repite L veces, siempre sobre la submatriz de baja frecuencia. Esto define un procedimiento recursivo. Dada una matriz A, se define $T_0 = A$, y seguidamente se aplican las transformaciones que permiten una descomposición sucesiva para los distintos niveles:

$$T_i = P_i T_{i-1} P_i^T \tag{2.43}$$

$$A_i = Q_i T_{i-1} Q_i^T \tag{2.44}$$

$$B_i = Q_i T_{i-1} P_i^T \tag{2.45}$$

$$C_i = P_i T_{i-1} Q_i^T \tag{2.46}$$

Al final la transformada wavelet de una matriz T_0 viene definida por el conjunto de matrices $\{(A_1, B_1, C_1); (A_2, B_2, C_2); ...; (T_n, A_n, B_n, C_n)\}$ donde n es la cantidad de niveles wavelet aplicados. En la figura 2.14 se muestra gráficamente el proceso de descomposición de una matriz en 3 niveles de trasformada wavelet no estándar. Es de destacar que para la implementación no se utiliza memoria adicional ya que se puede ocupar el mismo espacio de almacenamiento que la matriz original a transformar.

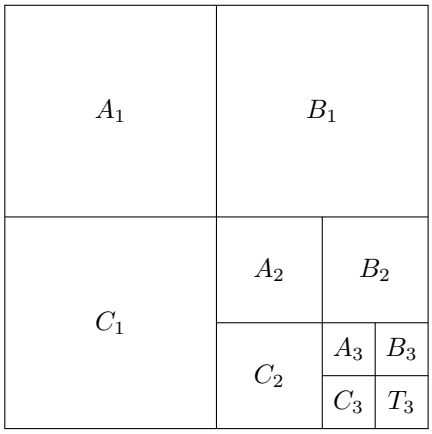


Figura 2.14: Matriz descompuesta en 3 niveles de la transformada wavelet no estándar.

Transformada wavelet no estándar extendida

La "Transformada wavelet no estándar extendida" es muy similar a la transformada no estándar. Para calcularla se utiliza el mismo procedimiento recursivo usando las ecuaciones (2.43), (2.44), (2.45) y (2.46) con la particularidad de que también se almacenan las submatrices $T_0, T_1, ...T_{n-1}$. En este caso se puede intuir que, para la implementación, no basta con utilizar el espacio de almacenamiento de la matriz original como sí sucede con la no estándar; en su lugar se deben almacenar una sucesión decreciente de submatrices:

$$\{(T_1, A_1, B_1, C_1); (T_2, A_2, B_2, C_2); ...; (T_n, A_n, B_n, C_n)\}$$

En la figura 2.15 se puede observar una matriz descompuesta en tres niveles en forma no estándar extendida.

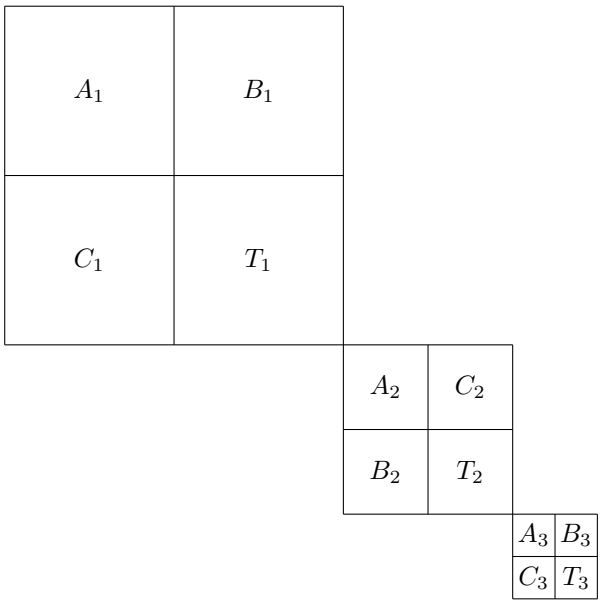


Figura 2.15: Matriz descompuesta en 3 niveles de la transformada wavelet no estándar extendida.

2.5. Implementación de la DWT

Si bien la forma matricial del cálculo de la DWT tiene un interés didáctico, existe una forma más eficiente de realizar esta operación.

El algoritmo 2.1 muestra la implementación de la transformada wavelet discreta sobre un vector. Esta implementación se conoce como transformada rápida de wavelet o DWT.

El operador $\langle 2i + j \rangle_n$ se define como el resto de la división entera entre $2i + j$ y n. Este operador permite que se tomen componentes de la primera parte del vector v cuando la suma $2i + j$ se haga mayor que n. Físicamente este procedimiento es equivalente a considerar la señal discreta v como una señal periódica en n.

El costo de aplicar un nivel de la DWT a un vector de longitud n usando filtros de longitud m es de nm productos y nm sumas, en total $2nm$ flops. En cada nivel se divide a la mitad el tamaño del vector por lo tanto el costo total para los k niveles

estaría dado por:

$$C_{n,k}^{1} = \sum_{i=0}^{k-1} 2m\frac{n}{2^i} = 4mn\left(1 - \frac{1}{2^k}\right) \tag{2.47}$$

Con lo que se concluye que la complejidad es de $O(n)$.

Algoritmo 2.1 DWT de k niveles de un vector de longitud n

1. **para** $l := 1$ **to** k **hacer**
2. *Inicializar a w como un vector nulo de longitud n*
3. **para** $i := 0$ **to** $\frac{n}{2} - 1$ **hacer**
4. **para** $j := 0$ **to** $m - 1$ **hacer**
5. $w_i = w_i + h_j v_{\langle 2i+j \rangle_n}$
6. $w_{i+\frac{n}{2}} = w_i + g_j v_{\langle 2i+j \rangle_n}$
7. **fin para**
8. **fin para**
9. $v_{0:n-1} = w$
10. $n = \frac{n}{2}$
11. **fin para**

La trasformada inversa también se puede implementar usando un algoritmo rápido, ver algoritmo 2.2.

Algoritmo 2.2 DWT inversa de k niveles de un vector de longitud N

1. $n := \frac{N}{2^{k-1}}$
2. **para** $l := 1$ **to** k **hacer**
3. *Inicializar a w como un vector nulo de longitud n*
4. **para** $i := 0$ **to** $\frac{n}{2} - 1$ **hacer**
5. **para** $j := 0$ **to** $m - 1$ **hacer**
6. $w_{\langle 2i+j \rangle_n} = w_{\langle 2i+j \rangle_n} + h_j v_i$
7. $w_{\langle 2i+j \rangle_n} = w_{\langle 2i+j \rangle_n} + g_j v_{i+\frac{n}{2}}$
8. **fin para**
9. **fin para**
10. $v_{0:n-1} = w$
11. $n = 2n$
12. **fin para**

Una simple comparación de los algoritmos 2.1 y 2.2 permite inferir que el costo de calcular la DWT inversa es idéntico al de la directa.

En el caso bidimensional la DWT-2D, si se trata de la forma estándar que se basa en aplicar repetidamente la DWT-1D a cada fila primero y luego a cada columna de la matriz, el costo total quedaría:

$$C_{n,k}^2 = n\sum_{i=0}^{k-1} 2m\frac{n}{2^i} + n\sum_{i=0}^{k-1} 2m\frac{n}{2^i} = 8mn^2\left(1 - \frac{1}{2^k}\right) \tag{2.48}$$

Con lo que se concluye que la complejidad está en el orden de $O\left(n^2\right)$.

La complejidad de la transformada wavelet no-estándar y no estándar extendida definidas por las ecuaciones recursivas de la (2.43) a la (2.46) está determinada en cada nivel j de la recursividad por la misma complejidad de la estándar para $k = 1$ en cada nivel. Como en cada nivel las dimensiones de la matriz T_j se dividen entre 2 entonces la complejidad total quedaría como sigue:

$$C_{n,k}^2 = \sum_{j=0}^{k-1} 8m\left(\frac{n^2}{2^{2j}}\right) = 8mn^2\sum_{j=0}^{k-1}\frac{1}{2^{2j}} \tag{2.49}$$

Igual que en el caso anterior el orden de complejidad es de $O\left(n^2\right)$.

2.6. Wavelet packet

La *transformada wavelet packet* puede verse como una generalización de la transformada wavelet. Su desarrollo se debe a Coifman, Meyers, Quake y Wickerhauser en [24, 23]. En la transformada wavelet, el espacio funcional V_{j+1} se divide en los subespacios V_j y W_j, usando los filtros g y h y las relaciones (2.24) y (2.25). El espacio resultante V_j se vuelve a dividir, un proceso que se repite continuamente.

Podemos descomponer el espacio W_j haciendo uso de los mismos filtros. Se obtienen entonces dos espacios funcionales cuya suma directa sería igual a W_j. Si se continúa aplicando recursivamente la descomposición se obtiene un árbol binario de descomposición con raíz V_j. Las funciones bases de esos subespacios se llaman *wavelets packets*.

En la Figura 2.16 se muestra el árbol binario de descomposición para la trasformada wavelet y wavelet packet.

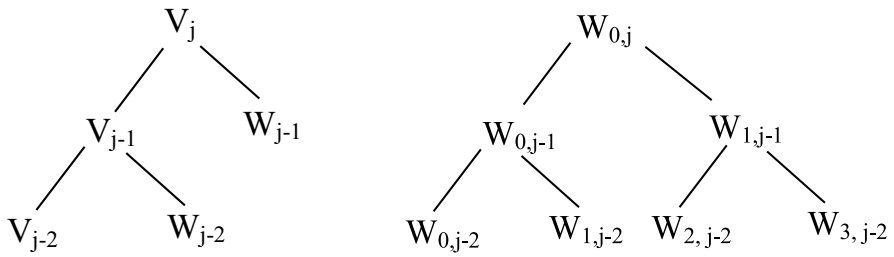


Figura 2.16: (a) Descomposición wavelet. (b) Wavelet Packet

Capítulo 3

Cálculo paralelo de la DWT en matrices densas

La complejidad de la aplicación de la DWT-2D en una matriz densa, tal y como se vio en la sección 2.5, es cuadrática. Cuando la matriz a transformar es de gran dimensión, se hace necesario buscar variantes de paralelización.

En este capítulo deseamos revisar los trabajos existentes sobre paralelización de la DWT, y proponer algunas variantes para mejorar la eficiencia de los esquemas propuestos.

Por razones de eficiencia, existe una restricción en el tipo de distribuciones que se pueden considerar. En efecto, considerando el caso 1D, la DWT es una operación que se aplica sobre componentes contiguas de un vector. De cara a que la operación distribuida se realice de forma eficiente, cualquier distribución de datos debe estar compuesta por subvectores dados por componentes consecutivas del vector original. Además, la longitud de los subvectores en cada procesador debe ser (también por eficiencia) mayor que la longitud L de los filtros wavelet. Denominaremos a este tipo de distribución "Distribución con contigüidad".

En el caso bidimensional, llamaremos también "Distribución con contigüidad" a aquella basada en bloques de filas y columnas "contiguos". Todos los resultados y análisis a partir de ahora se referirán a distribuciones con esta característica.

En el epígrafe 3.1 discutiremos los resultados propuestos por Nielsen y Hegland en [50]. En el epígrafe 3.2 introduciremos algunas variantes para lograr mejores presta-

ciones en el calculo de la DWT paralela introduciendo replicación de los datos en distintos niveles del cálculo y en particular propondremos una variante para cuando se use la distribución 2DBC.

3.1. La transformada wavelet discreta paralela

El objetivo, con una arquitectura paralela, es distribuir el trabajo entre varios procesadores para llegar a un resultado más rápidamente, o resolver problemas más grandes que los que son posibles resolver con solo un procesador. Sea $T(n)$ el tiempo de calcular la DWT con un algoritmo secuencial en un solo procesador. Idealmente el tiempo necesario para calcular la misma tarea en un sistema paralelo con p procesadores es $\frac{T(n)}{p}$. Sin embargo hay un conjunto de cuestiones, planteadas por Kumar et al. en [39] que impiden que se alcance ese estimado ideal:

Comunicación entre los procesadores Cualquier sistema paralelo no trivial requiere que sus elementos procesadores interactúen e intercambien datos. El tiempo que gastan los procesadores en el intercambio de datos es usualmente la principal fuente de sobrecarga en el procesamiento paralelo.

Inactividad Los elementos de procesamiento en un computador paralelo pueden estar inactivos por distintas razones, tales como: desequilibrio de carga, sincronización y presencia de fragmentos seriales. Cada uno de estos elementos inciden en que uno o varios procesadores tengan periodos de inactividad desaprovechando su capacidad de procesamiento.

En este epígrafe se expondrá un estado del arte de las distintas variantes de paralelización existentes de la transformada wavelet. Cada variante trata de disminuir los efectos negativos planteados anteriormente: el desequilibrio de carga, las comunicaciones y la sincronización. En cuanto a la sobrecarga por la presencia de elementos seriales, se asume que es muy pequeña y en general despreciable en el cálculo de la DWT.

Inicialmente se partirá de las variantes del cálculo de la DWT en una dimensión para entender mejor otras variantes en más dimensiones. La mayor parte del material que se trata en este epígrafe se puede encontrar en más detalle en [50].

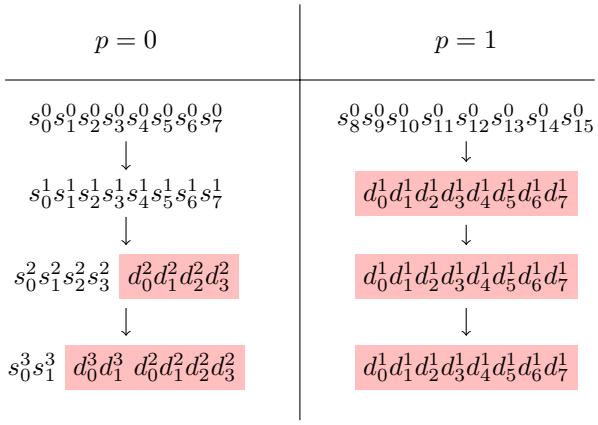


Figura 3.1: Distribución de datos intermedios en el cálculo de la DWT en dos procesadores, los sub-vectores sombreados indican que no se realizan más cálculos con ellos. $P = 2$, $N = 16$, $\lambda = 3$

3.1.1. Cálculo paralelo de la DWT-1D: Estado del arte

El problema en el cálculo paralelo de la DWT-1D de un vector x consiste en distribuir el trabajo entre P procesadores denotados por $p = 0, 1, \ldots, P-1$ para calcular el producto $y = Wx$, con W definido en (2.33). Se asume que los procesadores están organizados en una topología de anillo de forma que $\langle p-1 \rangle$ y $\langle p+1 \rangle$ son los vecinos izquierdo y derecho respectivamente de p. Por simplicidad también se asume que N, la dimensión de x, es múltiplo de P y que el vector x está distribuido de forma que cada procesador recibe la misma cantidad de elementos consecutivos. Así un procesador p cualquiera recibe los elementos:

$$\{s_n^0\}_N, \quad n = p\frac{N}{P}, p\frac{N}{P} + 1, \ldots, (p+1)\frac{N}{P} - 1$$

Denotaremos s_n^i como el componente n-ésimo de baja frecuencia en el nivel i de la transformada wavelet, en particular, en el nivel 0 se cumple que $s_n^i = x_n$. El problema ahora es cómo quedan distribuidos los resultados intermedios en los procesadores. Si aplicamos directamente la forma de cálculo que sugiere el algoritmo 2.1 estos quedaría distribuidos de forma consecutiva tal y como se muestra en la figura 3.1.

A simple vista se puede observar que el procesador 1 luego del cálculo del primer nivel queda desocupado. Este fenómeno se conoce como desbalance de carga y se enunció como uno de los problemas más comunes de ineficiencia en los algoritmos

paralelos. En [50] se propone otro orden de los datos intermedio y del resultado para lograr un mejor equilibrio de la carga. La propuesta se puede observar en la figura 3.2. En esta propuesta un procesador p va a calcular y almacenar los elementos $\{s_n^{i+1}\}_n$ y $\{d_n^{i+1}\}_n$ donde:

$$\begin{aligned} n &= p\frac{N}{P2^i}, p\frac{N}{P2^i}+1, \ldots, (p+1)\frac{N}{P2^i}-1 \\ i &= 0, 1, \ldots, \lambda-1 \end{aligned} \tag{3.1}$$

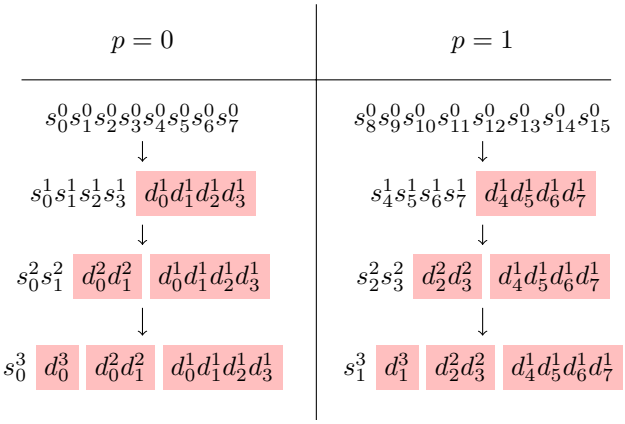


Figura 3.2: Distribución de datos intermedios en el cálculo de la DWT para lograr un mejor equilibrio de la carga, los sub-vectores sombreados significan que esa parte no requiere cálculos posteriores. $P = 2$, $N = 16$, $\lambda = 3$

Con esta nueva distribución, en esencia, el procesador calcula de forma local la DWT a un vector local. Además tiene la ventaja de que todos los procesadores involucrados comienzan y terminan al mismo tiempo y realizan la misma cantidad de cálculos con lo que alcanza un balance de carga perfecto. Este nuevo orden exige, sin embargo, que en el cálculo de un nivel cada procesador p comunique elementos hacia su vecino de la izquierda $\langle p-1\rangle_P$. En [50], Nielsen y Hegland plantean:

Proposición 3.1. *Considérese la paralelización de la DWT-1D, aplicada sobre un vector de longitud N, y fragmentado sobre P procesadores $(0, \ldots, P-1)$ en subvectores de longitud $\frac{N}{P}$. Si el orden de la wavelet (longitud de los filtros) es L, para calcular la DWT-1D del vector v, cada procesador debe recibir exactamente $L-2$ elementos desde sus vecinos de la derecha. (excepto para P_{p-1}, que debe recibir $L-2$ elementos*

de P_0)

De la proposición 3.1 se deriva, en particular, que si la longitud del filtro es 2 entonces no habría comunicaciones.

Para entender mejor analicemos el cálculo de la DWT de un nivel sobre un vector que se muestra en la Figura 3.3. Aquí se evidencia la dependencia de los datos en el cálculo de la DWT para un vector de longitud 16 distribuido en 2 procesadores usando un filtro wavelet de longitud $L = 4$. Por ejemplo para calcular los componentes de baja frecuencia, el procesador P_0 calcula los coeficientes s_0^1, s_1^1, s_2^1, s_3^1, la misma cantidad que el procesador P_1 que calcula los coeficientes s_4^1, s_5^1, s_6^1, s_7^1 (el proceso para calcular los coeficientes de alta frecuencia es el mismo). La dependencia de datos viene dada en que, por ejemplo, el procesador P_0 necesita para calcular el coeficiente s_3^1 recibir los elementos v_8 y v_9 que se encuentran en el procesador P_1 y de forma similar para calcular el coeficiente s_7^1 el procesador P_1 necesita recibir los valores v_0 y v_1 (en una DWT periódica) que están en el procesador P_0.

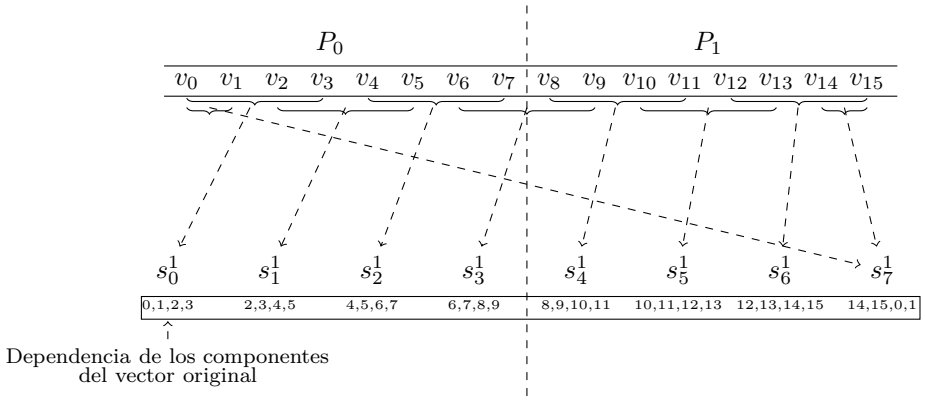


Figura 3.3: Un nivel de la DWT para un vector v distribuido sobre 2 procesadores, filtros con $L = 4$ (4 coeficientes)

3.1.2. Cálculo paralelo de la DWT-2D: Estado del arte

El caso matricial también fue discutido por Nielsen y Hegland en [50]. Los autores proponen dos modelos para el cálculo paralelo de la DWT-2D. Un primer modelo al que le llaman "DWT Replicada" y un segundo modelo llamado "DWT eficiente en comunicaciones".

El procedimiento utilizado por la "DWT Replicada" se basa en una distribución de datos donde cada procesador reciba la misma cantidad de filas consecutivas de la matriz a transformar. Se aplica la DWT-1D en cada fila, se traspone de forma paralela toda la matriz, y luego cada procesador calcula la DWT-1D a cada nueva fila. Lo anterior se puede ver gráficamente en 3.4.

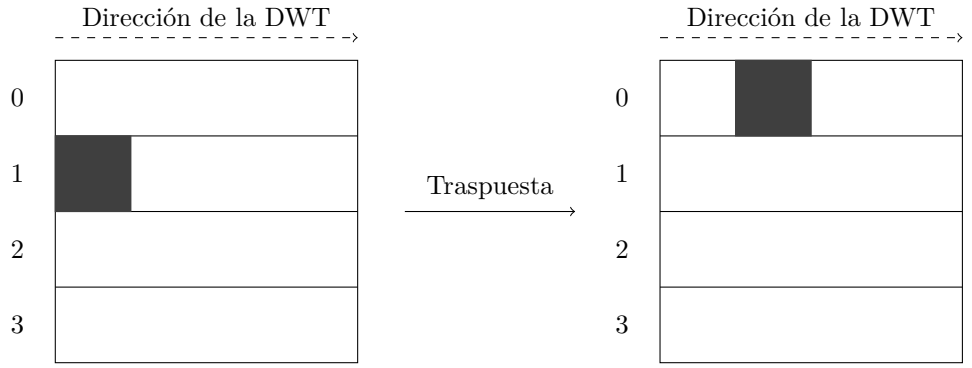


Figura 3.4: DWT Replicada. El bloque sombreado se mueve desde el procesador 1 al procesador 0 cuando se aplica la traspuesta de forma paralela.

Esta forma de proceder tiene la ventaja de que no se efectúan comunicaciones en los pasos en los que se calcula la DWT; sin embargo sí hay un monto grande de comunicaciones cuando se procede a trasponer la matriz en cuestión.

Una variante que evita el paso de trasposición también se propone en [50]. En ese trabajo la hacen llamar "DWT Eficiente en comunicaciones".

La "DWT Eficiente en comunicaciones" parte de una distribución de la matriz de la misma cantidad de columnas consecutivas a cada procesador. Se calcula la DWT-1D de cada fila, utilizando la técnica de la proposición 3.1; es decir, cada procesador debe recibir para cada fila $(L-2)$ elementos. Luego, se traspone localmente la submatriz de cada procesador y luego se aplica la DWT-1D local a cada fila de la matriz resultante. Esta idea se puede observar mejor en la figura 3.5.

La gran ventaja de esta variante es que la traspuesta de la matriz se puede hacer sin ninguna comunicación. Por otro lado las únicas comunicaciones que se necesitan son las requeridas por la DWT-1D multiple propuesta en 3.1.1. En total se necesita comunicar $M(L-2)$ datos, donde M es la cantidad de filas de la matriz a trasformar.

En Chaver et al. [18] se proponen dos variantes al trabajo de Nielsen y Hegland

para el caso de la DWT no Estándar. La primera modificación tiene que ver con usar el mismo esquema de la "DWT replicada" pero evitando el paso de trasponer la matriz a partir de hacer un mejor uso de la jerarquía de memoria y calcular la DWT por columnas de forma paralela. La segunda modificación propuesta en [18] tiene que ver con que se determine un "nivel crítico" a partir del cual el cálculo paralelo no puede mejorar el secuencial. Así el cálculo de la DWT se replica a partir de este "nivel crítico". En [18] se dice que este "nivel crítico" depende de la topología paralela y la cantidad de niveles wavelet pero no se expone una forma precisa para calcularlo. En el mismo trabajo se reconoce que la mayor parte de las aplicaciones de la DWT no exigen que se calculen todos los niveles posibles, y se propone la variante para cuando esto sea necesario.

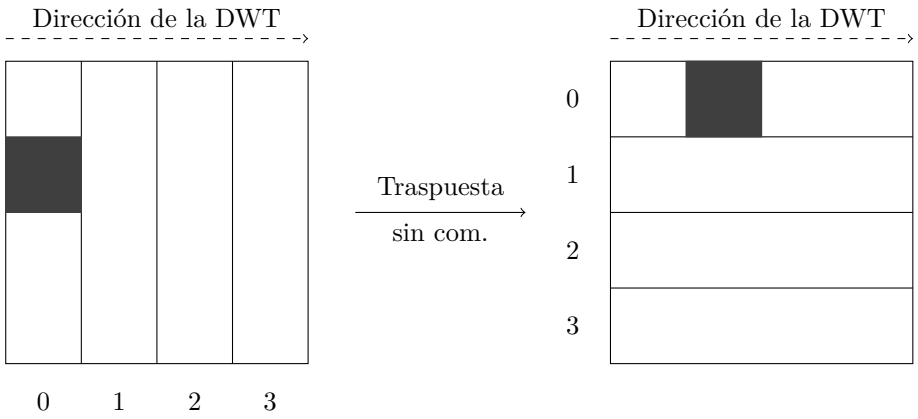


Figura 3.5: DWT Eficiente en comunicaciones. El bloque sombreado se mantiene en el mismo procesador lo que indica que la traspuesta puede hacerse sin comunicaciones.

3.2. Transformada wavelet discreta paralela con replicación parcial

En la sección 3.1 se presentó un estado del arte de las variantes de paralelización de la DWT. Se mencionaron los trabajos más importantes en este tema. En particular se mostró la mejor variante conocida para el calculo de la DWT-2D, la "Eficiente en comunicaciones" de Nielsen y Hegland. En este epígrafe se mostró que esa variante exige que se use una distribución de datos en la que cada procesador contenga una cantidad de columnas consecutivas de la matriz a trasformar. Sin embargo si la DWT

es solo una operación intermedia en alguna aplicación al Álgebra Lineal, por ejemplo si se tuviera que calcular la descomposición LU luego de aplicada la DWT entonces la distribución más adecuada ya no sería la dicha. Para el caso particular de la descomposición LU existe una distribución por bloque cíclicos que es la usada en librerías de álgebra lineal como ScaLAPACK [13].

En este epígrafe extenderemos los trabajos de Nielsen y Hegland en [50]. Mostraremos los resultados obtenidos por Acevedo y García en [7] donde se presentó la "DWT Eficiente en Comunicaciones" en una forma más general, para bloques de matrices y se demostró como se podían beneficiar otro tipo de distribuciones de datos, como la distribución ScaLAPACK. Extenderemos este trabajo proponiendo una variante de reordenamiento de la matriz que permita a su vez disminuir las comunicaciones.

Otra deficiencia de la "DWT Eficiente en Comunicaciones" es que tiene muchos pasos de comunicación esto es: para cada nivel λ_i de la DWT se necesitan comunicar $\frac{M}{2^i}(L-2)$ elementos. En este epígrafe probaremos que es más eficiente comunicar solo los elementos necesarios para calcular λ niveles de la DWT al inicio. Y daremos una vía para calcular exactamente esa cantidad de elementos.

3.2.1. Cálculo paralelo de la DWT-1D de varios niveles. Replicación parcial

Cuando se calcula un sólo nivel de la DWT es relativamente fácil conocer cuantos elementos extras necesita, un procesador dado, enviar o generar para hacer el cálculo de la DWT sin necesidad de comunicaciones posteriores. Para el caso en el que se calculan varios niveles de la DWT la predicción de la cantidad de elementos necesarios no es tan evidente. En [18] se propone replicar completamente la matriz original en todos los procesadores. Está claro que con esta estrategia no se necesitarán comunicaciones posteriores pero tiene la dificultad evidente de que si lo que vamos a calcular son pocos niveles de la DWT entonces estaremos utilizando memoria adicional de forma innecesaria.

Nuestra primera propuesta es la de calcular exactamente cuantos elementos extras necesita un procesador dado para calcular λ niveles de la DWT de tal forma que estos elementos se puedan envíar (generar) cuando se realice la distribución inicial de los datos. De esta forma, sólo se enviarán (generarán) los elementos de la matriz estrictamente necesarios, y se logrará evitar todas las comunicaciones posteriores.

Consideremos un vector distribuido sobre un conjunto de procesadores, queremos determinar el número $N_{L,\lambda}$ de elementos del vector original (aparte de los elementos originalmente en el procesador) que cada procesador necesitaría para la DWT de nivel λ usando filtros wavelet de longitud L.

Antes de probar el resultado principal necesitamos el siguiente Lema:

Lema 3.1. *Considere el cálculo paralelo de la DWT-1D de nivel k, aplicada sobre un vector de longitud N, y distribuido sobre P procesadores ($0, \ldots, P-1$) en subvectores de longitud $\frac{N}{P}$. Asumamos que cada procesador ha calculado la DWT de nivel k de su subvector (y para hacer esto ya ha recibido los $N_{L,k}$ elementos necesarios del vector original). Cada procesador para calcular M elementos adicionales de la DWT de nivel k sin comunicaciones posteriores, debe recibir $2^k \cdot M$ elementos adicionales del vector original.*

Demostración. Probemos en primer lugar el Lema para $k = 1$ (un nivel). Sea v el vector original y v_{dwt} el vector transformado DWT de nivel uno.

Consideremos un procesador p, que inicialmente contiene el subvector de v localizado entre los índices $[i_{ini}, i_{fin}]$, y que ya calculó su parte del subvector v_{dwt}, entre los índices $[iw_{ini}, iw_{fin}]$. Para esto el procesador p debió recibir $L-2$ elementos de otro procesador vecino, desde $v[i_{fin}+1]$ hasta $v[i_{fin}+L-2]$.

Lo que se desea es que este procesador calcule algunos elementos extras de v_{dwt} "a la derecha" de los ya calculados: $v_{dwt}[iw_{fin}+1], v_{dwt}[iw_{fin}+2], \ldots$. Necesitamos conocer cuantos elementos extras del vector v debe contener el procesador p para calcular estos elementos extras del vector trasformado v_{dwt}.

Comencemos por el elemento $v_{dwt}[iw_{fin}+1]$, para calcular este elemento el procesador p necesita los elementos $v[i_{fin}+1]$, $v[i_{fin}+2]$ hasta el elemento $v[i_{fin}+L]$ (recalcar aquí que el tamaño del filtro es L). Sin embargo, los primeros $L-2$ elementos han sido enviados a p, así que los únicos nuevos elementos que este procesador necesita son $v[i_{fin}+L-1]$, $v[i_{fin}+L]$.

Para poder calcular los próximos elementos $v_{dwt}[iw_{fin}+2]$, y tomando el subsampleo en consideración, el procesador p necesita $v[i_{fin}+3]$, $v[i_{fin}+4]$, hasta $v[i_{fin}+L+2]$. Nuevamente se necesitan dos elementos de v para calcular uno de v_{dwt}. En este momento debe estar claro que:

Un procesador p necesita $2M$ valores adicionales del vector original v para poder

calcular M elementos adicionales del vector transformado v_{dwt}.

Se ha probado para un solo nivel partiendo del nivel 0. Sin embargo el razonamiento es exactamente igual para ir desde un nivel cualquiera $k-1$ al nivel k.

Consideremos el caso propuesto en el lema 3.1 donde cada procesador ya ha calculado su parte de la DWT de nivel k, y necesita calcular M elementos extras de la DWT de nivel k localizados, como antes, "a la derecha" de los elementos ya calculados.

Usando el resultado obtenido, para calcular estos M elementos el procesador p necesita $2M$ elementos del vector transformado DWT de nivel $k-1$; para calcular los mismos se necesitan $4M$ del transformado DWT de nivel $k-2$. Repitiendo el proceso llegamos a que *para calcular los M elementos de la DWT de nivel k se necesitan $2^k \cdot M$ elementos del vector original v (nivel 0)* lo cual prueba el lema 3.1.

□

Podemos examinar nuevamente la figura 3.3 para comprobar el lema. En esta figura se ve que para calcular su parte en el primer nivel, el procesador p_0 recibió los elementos v_9 y v_{10}. Si se quiere que P_0 calcule también el elementos s_1^5, necesita tener acceso a los elementos v_9, v_{10}, v_{11} y v_{12} pero como ya los elementos v_9 y v_{10} están disponibles solo necesitaría v_{11} y v_{12}. Para calcular además el elemento s_1^6, necesitaría dos nuevos elementos v_{13} y v_{14}, y así sucesivamente.

De esta forma el número de elementos que debe ser enviado a cada procesador se obtiene fácilmente mediante la siguiente proposición.

Proposición 3.2. *Considérese el cálculo paralelo de λ niveles de la DWT-1D, aplicado sobre un vector de longitud N, y distribuido sobre P procesadores $(0, \ldots, P-1)$ en subvectores de longitud $\frac{N}{P}$. Para completar la tarea, sin comunicaciones posteriores, cada procesador debe recibir*

$$N_{L,\lambda} = (2^\lambda - 1)(L-2) \tag{3.2}$$

elementos adicionales del vector v localizados "a la derecha" de los ya disponibles en el procesador p.

Demostración. A continuación demostraremos esta proposición, utilizando inducción sobre el número de niveles λ.

Consideremos en primer lugar el caso base, con $\lambda = 1$. Este es el caso de aplicar

solo un nivel wavelet; aplicando la Proposición 3.1, la cantidad de elementos que se necesitan para un nivel de la DWT es $L-2$. Por lo tanto 3.2 se cumple para $\lambda = 1$.

$$N_{L,1} = \left(2^1 - 1\right) \cdot (L-2) = L-2$$

Suponiendo que (3.2) se cumple para cierto $\lambda = k$

$$N_{L,k} = \left(2^k - 1\right) \cdot (L-2)$$

demostremos que se cumple también para $\lambda = k+1$.

Para calcular la DWT de nivel $k+1$ cada procesador necesita haber calculado la de nivel k para lo que habrá recibido, por hipótesis de inducción $N_{L,k} = (2^k - 1)(L-2)$ **elementos adicionales del vector original**.

Aparte de estos $N_{L,k}$ elementos del vector original cada procesador necesita algunos elementos extras para completar la DWT de nivel $k+1$. Estos elementos se pueden determinar fácilmente ya que la DWT de nivel $k+1$ implica aplicar un nivel de la DWT sobre el vector DWT de nivel k. Así usando la proposición base 3.1, cada procesador debe recibir $L-2$ **elementos adicionales del vector trasformado DWT de nivel** k.

Así para calcular la DWT de nivel $k+1$ cada procesador necesita tener disponible $N_{L,k} = (2^k - 1)(L-2)$ elementos del vector original además de $L-2$ elementos adicionales correspondiente al vector DWT de nivel k.

Sin embargo queremos que el procesador complete esta tarea recibiendo elementos solo del vector original. Usando el lema 3.1, obtenemos que para calcular esos $L-2$ elementos del vector DWT de nivel k, se necesitan $2^k \cdot (L-2)$ elementos del vector original.

Manipulando, se obtiene rápidamente que la cantidad de elementos del vector original que se necesitan es:

$$(2^k - 1)(L-2) + 2^k(L-2) = (2^{k+1} - 1)(L-2) \tag{3.3}$$

Con lo que se prueba la proposición 3.2.

□

A manera de ejemplo en la Figura 3.6 se puede ver la dependencia de los datos para la DWT en dos niveles (nuevamente en el caso $L = 4$). Para calcular s_1^2, P_0 debe haber calculado previamente s_4^1 y s_5^1. Para realizar el cálculo de esos componentes de primer nivel, P_0 necesita los componentes relevantes del vector original. Esto significa que P_0 necesita $N_{2,4} = (2^2 - 1)(4 - 2) = 6$ componentes extras del vector original, desde el v_8 al v_{13}.

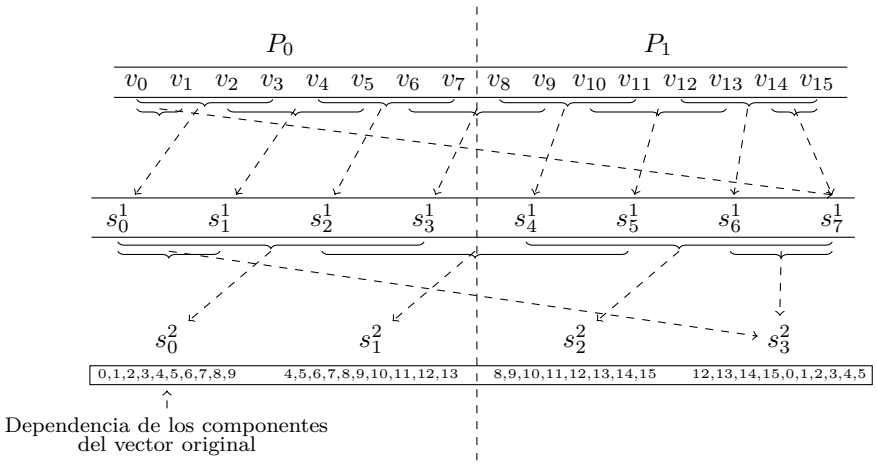


Figura 3.6: Dos niveles de la DWT a un vector v distribuido en 2 procesadores, filtro $L = 4$ (4 coeficientes).

Un detalle a tener en cuenta para usar el resultado 3.2 es que dicha ecuación crece exponencialmente con respecto a la cantidad de niveles λ. Si λ es grande habría que usar mucha memoria adicional, además de la replicación de los cálculos lo que haría inservible la propuesta. Aunque en la mayoría de las aplicaciones de la DWT en el álgebra lineal λ es pequeño, para minimizar la dificultad anterior podemos mezclar este resultado con el procedimiento clásico que es comunicar los valores necesarios cada vez que se aplique un nivel. Esto es buscar un convenio entre hacer varios niveles sin comunicaciones intermedias, usando el resultado 3.2, volver a comunicar para calcular varios niveles más con el mismo criterio y así sucesivamente.

3.2.2. Cálculo paralelo de la DWT-2D de varios niveles

El caso matricial se puede derivar directamente del resultado alcanzado en el epígrafe anterior. Considérese por ejemplo el bloque de la matriz A, delimitado por

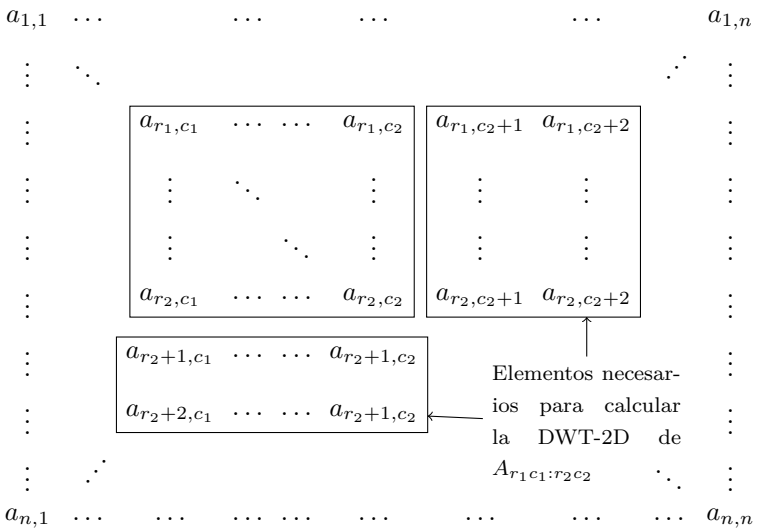


Figura 3.7: Elementos necesarios para calcular la DWT 2D de un nivel al bloque $A_{r_1:r_2,c_1:c_2}$ con $L = 4$

las filas r_1 y r_2 y por las columnas c_1 y c_2: $A_{r_1:r_2,c_1:c_2}$, asignado a un procesador dado. El cálculo de la DWT-2D comienza (por ejemplo) aplicando DWT-1D sobre cada una de las filas. Haciendo uso del resultado para vectores, el procesador dueño del bloque $A_{r_1:r_2,c_1:c_2}$ debe recibir el bloque compuesto por las $L-2$ columnas comprendidas entre r_1 y r_2: $A_{r_1:r_2,c_1:c_2+L-2}$. Lo mismo pasa para aplicar la DWT-1D sobre las columnas. En este caso el bloque que se necesita debe ser $A_{r_1:r_2+L-2,c_1:c_2}$. Esto se muestra en la Figura 3.7 para el caso en que $L = 4$ Esto generaliza fácilmente el caso de una matriz distribuida en bloques de filas consecutivas o bloques de columnas consecutivas.

Cuando debemos calcular varios niveles, el caso matricial es muy similar al unidimensional, ver la Figura 3.8 (recordemos que estamos usando transformada wavelet no estándar). Si queremos calcular dos niveles de la DWT con $L = 4$ del bloque $A_{r_1:r_2,c_1:c_2}$, para cada fila necesitamos $\left(2^2-1\right)(4-2) = 6$ elementos extra, y lo mismo para la columna. Es decir que cuando se aplica la DWT por filas al bloque $A_{r_1:r_1,c_1:c_2}$ se deben obtener de "la derecha" el bloque $A_{r_1:r_2,(c_2+1):(c_2+6)}$. Sin embargo para poder hacer posteriormente la transformada por columnas también debemos transformar por filas el bloque $A_{(r_2+1):(r_2+6),c_1:c_2}$ esto implica que necesitaremos comunicar, además de los bloques adyacentes por filas y columnas, el bloque diagonal

$A_{(r_2+1):(r_2+6),(c_2+1):(c_2+6)}$.

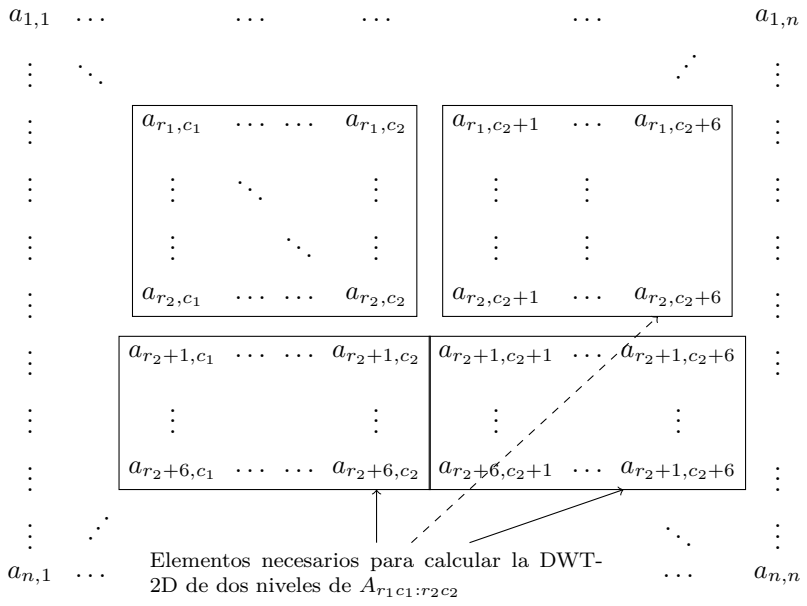


Figura 3.8: Elementos necesarios para calcular dos niveles de la DWT del bloque $A_{r_1:r_2,c_1:c_2}$, con $L = 4$

Por lo tanto, la replicación parcial para el cálculo paralelo de la DWT-2D se puede aplicar exactamente igual que en el caso 1D, determinando de antemano cuantos elementos extra necesita cada procesador (gracias a la proposición 3.2) y enviándolos (o generándolos) desde el principio, evitando así toda comunicación posterior.

3.3. Cálculo paralelo de la DWT-2D usando una distribución datos 2DBC

3.3.1. ScaLAPACK, distribución datos 2DBC

ScaLAPACK es una librería de álgebra numérica dedicada específicamente a las plataformas de cómputo paralelo en memoria distribuida, y es aceptada en general como la que implementa los mejores algoritmos paralelos existentes en cuanto a escalabilidad y optimización de rendimiento. La librería ScaLAPACK se presenta en

detalle en [13], en esta sección solo se hará una breve descripción que permita explicar las características del cálculo de la DWT-2D de una matriz distribuida para ser usada por la misma.

La figura 3.9 muestra los módulos sobre los que se soporta la librería ScaLAPACK. En particular el módulo BLACS permite mapear un arreglo lineal de procesos $N_0, \ldots, N_p$, como los que se manejan en la librería MPI (PVM), en una malla de procesos de Pr filas de procesos y Pc columnas de procesos, donde $Pr \times Pc \leq N_p$, ver el ejemplo que se muestra en la figura 3.10.

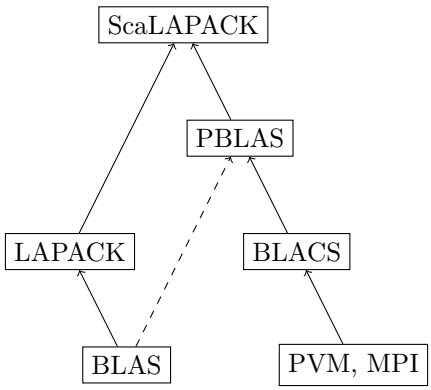


Figura 3.9: Módulos sobre los que se soporta la librería ScaLAPACK

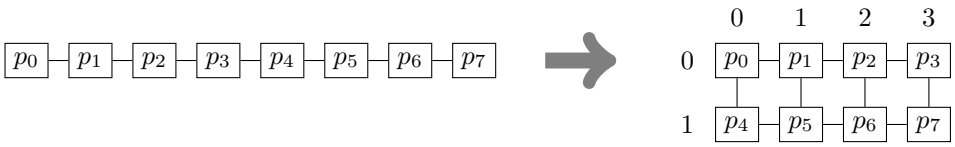


Figura 3.10: Mapeo de 8 procesos en un grid 2×4

Para aplicar las rutinas de la librería ScaLAPACK sobre una matriz esta debe estar distribuida usando la distribución 2DBC. Esta distribución tiene la característica de que los elementos de la matriz se asignan de forma cíclica y por bloques sobre los procesos de la malla definida en BLACS. El tamaño de bloque lo selecciona el usuario para cada aplicación en particular, y se debe elegir tomando un compromiso entre la eficiencia local, que debe ser mejor con bloques más grandes, y el balance de la carga, que debe mejorar con bloques más pequeños.

Consideremos por ejemplo, la malla bidimensional de procesos como en la figu-

ra 3.11 y una matriz 8×8 que se distribuye sobre esta malla usando un tamaño de bloque 2×2, tal y como se observa en la figura 3.12.

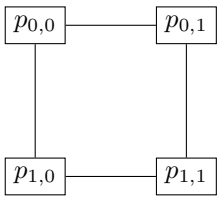


Figura 3.11: Malla bidimensional de procesos 2×2

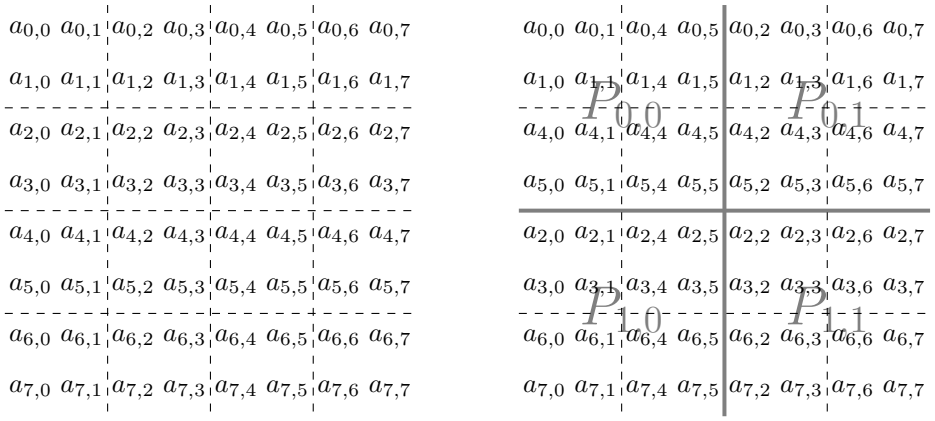


Figura 3.12: Ejemplo de una matriz (8×8) distribuida sobre la malla de procesadores (2×2), con tamaño de bloque 2×2 usando la distribución 2DBC de ScaLAPACK

En este ejemplo la matriz original se descompone en bloques de matrices de 2×2. Para distribuir la matriz en formato 2DBC se asignan los bloques a los procesos de forma cíclica, tomando una fila de bloques como inicial, en este caso la fila cero. La fila inicial se asigna de forma cíclica a los procesos de la fila 0 de la malla. En el ejemplo al procesador $P_{0,0}$ le corresponde el bloque $\begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix}$ al procesador $P_{0,1}$ le corresponde el bloque contiguo $\begin{pmatrix} a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,3} \end{pmatrix}$ luego, siguiendo el proceso de asignación cíclica, al procesador $P_{0,0}$ le corresponde el bloque $\begin{pmatrix} a_{0,4} & a_{0,5} \\ a_{1,4} & a_{1,5} \end{pmatrix}$ y para finalizar la asignación de la fila de bloques 0, al procesador $P_{0,1}$ le corresponde el bloque $\begin{pmatrix} a_{0,6} & a_{0,7} \\ a_{1,6} & a_{1,7} \end{pmatrix}$

Este proceso es el mismo para la fila 1 de bloques de la matriz original usando en este caso la fila 1 de procesos de la malla, etc.

3.3.2. DWT-2DBC

En la sección 3.3.1 se describe la distribución 2DBC como una distribución cíclica por bloques. Una variante trivial para realizar el cálculo de DWT-2D sería redistribuir la matriz desde la distribución 2DBC a una distribución por bloques de columnas, aplicar la DWT eficiente en comunicaciones explicada en la sección 3.1.2 y luego retornar la matriz a la distribución 2DBC. Esta variante tiene la dificultad evidente que se aplican dos procesos de redistribución costosos en comunicaciones.

El objetivo en esta sección es proponer una variante paralela de la DWT-2D aplicada a una matriz distribuida 2DBC, pero que no implique realizar un proceso de redistribución.

En la sección 3.2.2 se propone una forma paralela de calculo de la DWT-2D para bloques de matrices. En esta propuesta cada bloque de matriz se trata de forma independiente, si la longitud de filtro wavelet es mayor que 2 entonces se necesitan elementos de bloques consecutivos en fila, columna y diagonal en la matriz original. En particular si dichos bloques se encuentran en procesadores distintos entonces se generan comunicaciones.

Para dar una primera aproximación al problema, retomemos el ejemplo de la sección anterior sobre una matriz 8×8 distribuida sobre una malla de procesadores 2×2, ver figura 3.13.

En esta figura se resalta como ejemplo el cálculo de la DWT-2D de un nivel del bloque:

$$\begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix}$$

Para poder realizar la misma se necesita el bloque:

$$\begin{pmatrix} a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,3} \end{pmatrix}$$

que contiene los elementos consecutivos por fila

Además el bloque:

$$\begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix}$$

que contiene elementos consecutivos por columna. Estos dos bloques se encuentran, debido a la característica cíclica de la distribución 2DBC, en los procesadores $P_{0,1}$ y $P_{1,0}$ respectivamente. Una dependencia similar ocurre para el cálculo de la DWT-2D de todos los bloques de la matriz.

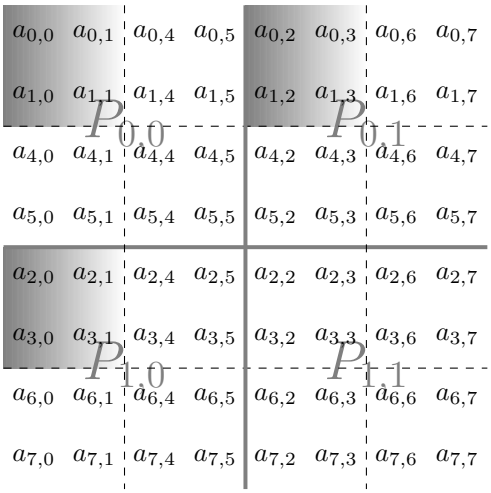


Figura 3.13: Cálculo de la DWT-2D una matriz (8×8) distribuida sobre la malla de procesadores (2×2), con tamaño de bloque 2×2 usando la distribución 2DBC de ScaLAPACK.

El algoritmo 3.1 muestra el esquema general para el cálculo de la DWT-2D de un nivel y longitud de filtro L, de una matriz A de dimensiones $N \times N$, distribuida 2DBC en una malla de $P \times P$ procesadores con tamaño de bloque $s \times s$.

La cantidad de "elementos necesarios" y "elementos extras" dependen del tamaño de filtro wavelet L y de las dimensiones $(s \times s)$ de los bloques $A_{i,j}$.

La cantidad de comunicaciones depende de la cantidad de bloques $A_{i,j}$ de la matriz $A_{P_{f,c}}$. La cantidad de bloques $A_{i,j}$ en la matriz local depende de s, el tamaño de bloque elegido para hacer la distribución 2DBC. En particular, un tamaño de bloques grande implica menos bloques de matrices en el procesador local y por ende menos comunicaciones, pero en la medida que vamos disminuyendo las dimensiones del bloque las comunicaciones pueden aumentar de una manera drástica.

Una buena estrategia sería escoger bloques grandes a la hora de hacer la distribución 2DBC. Sin embargo es conocido que muchos problemas tienen un tamaño de bloque óptimo; o sea que si la DWT se tomara como una operación inicial sobre la matriz, pero luego se aplicaran otras operaciones más complejas como por ejemplo una descomposición LU, entonces el tamaño de bloque que se escoja dependerá de la operación posterior.

Algoritmo 3.1 Transformada DWT-2D de un nivel y longitud de filtro L, de una matriz A de dimensiones $N \times N$, distribuida 2DBC en una malla de $P \times P$ procesadores con tamaño de bloque $s \times s$

1. Para cada procesador $P_{f,c}$ de la malla.
2. En $P_{f,c}$:
3. **para todo** Bloque $A_{i,j}$ de A que pertenece a la matriz local $A_{P_{f,c}}$ **hacer**
4. **enviar** "elementos necesarios al procesador que contiene el bloque $A_{i-1,j}$."
5. **enviar** "elementos necesarios al procesador que contiene el bloque $A_{i,j-1}$."
6. **recibir** "elementos extras del procesador que contiene el bloque $A_{i+1,j}$."
7. **recibir** "elementos extras del procesador que contiene el bloque $A_{i,j+1}$."
8. Calcular la DWT-2D del bloque $A_{i,j}$.
9. **fin para**

Por lo tanto necesitamos una estrategia que permita disminuir la cantidad de comunicaciones en el cálculo de la DWT, que sea independiente del tamaño de bloque, de una matriz distribuida al estilo ScaLAPACK.

3.3.3. Reordenamiento en la distribución 2DBC

En este epígrafe se propondrá un procedimiento para aplicar subrutinas de la librería ScaLAPACK a sistemas de ecuaciones a los que se le ha aplicado un nivel de la DWT. Para clarificar el procedimiento, se usará como ejemplo una matriz de dimensiones (8×8) "especial", la que se muestra en la figura 3.14 a la izquierda.

Supongamos que queremos realizar cálculos de ScaLAPACK con esta matriz; utilizando como tamaño de bloque (2×2), y distribuyendo sobre una malla de (2×2) procesadores tal y como se muestra en la figura 3.14.

Consideremos la matriz Ap globalmente distribuida; se puede considerar que pasamos de A a Ap mediante permutaciones de filas y columnas (y la posterior distribución); en este ejemplo concreto, se tendría que:

$$P = Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

y tendríamos $Ap = PAQ$. Por lo tanto P y Q representan en este caso las permutaciones de columnas y filas necesarias para obtener la distribución 2DBC.

$$\underset{A}{\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\ 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \end{pmatrix}} \xrightarrow[\text{2DBC}]{\text{Distribución}} \underset{Ap}{\begin{array}{c|c} \overset{P_{0,0}}{\begin{pmatrix} 1 & 2 & 5 & 6 \\ 9 & 10 & 13 & 14 \\ 33 & 34 & 37 & 38 \\ 41 & 42 & 45 & 46 \end{pmatrix}} & \overset{P_{0,1}}{\begin{pmatrix} 3 & 4 & 7 & 8 \\ 11 & 12 & 15 & 16 \\ 35 & 36 & 39 & 40 \\ 43 & 44 & 47 & 48 \end{pmatrix}} \\ \hline \underset{P_{1,0}}{\begin{pmatrix} 17 & 18 & 21 & 22 \\ 25 & 26 & 29 & 30 \\ 49 & 50 & 53 & 54 \\ 57 & 58 & 61 & 62 \end{pmatrix}} & \underset{P_{1,1}}{\begin{pmatrix} 19 & 20 & 23 & 24 \\ 27 & 28 & 31 & 32 \\ 51 & 52 & 55 & 56 \\ 59 & 60 & 63 & 64 \end{pmatrix}} \end{array}}$$

Figura 3.14: Ejemplo de una matriz A, de dimensiones (8×8), distribuida 2DBC sobre una malla de procesadores (2×2) con tamaño de bloque (2×2). Ap es la matriz A globalmente distribuida.

Supongamos ahora que tenemos una matriz distribuida "directamente" sobre una malla de procesadores (ver la figura 3.15), y que queremos realizar cálculos, por ejemplo resolver el sistema $Ax = b$ usando rutinas de ScaLAPACK.

La solución simple sería redistribuir la matriz A a una distribución 2DBC y luego resolver el sistema. Esto tiene el inconveniente que al redistribuir se produce una gran cantidad de comunicaciones. Nuestra propuesta para evitar la sobrecarga de comunicaciones es considerar la matriz A distribuida "directamente", como una permutación de filas y columnas de alguna matriz distribuida 2DBC.

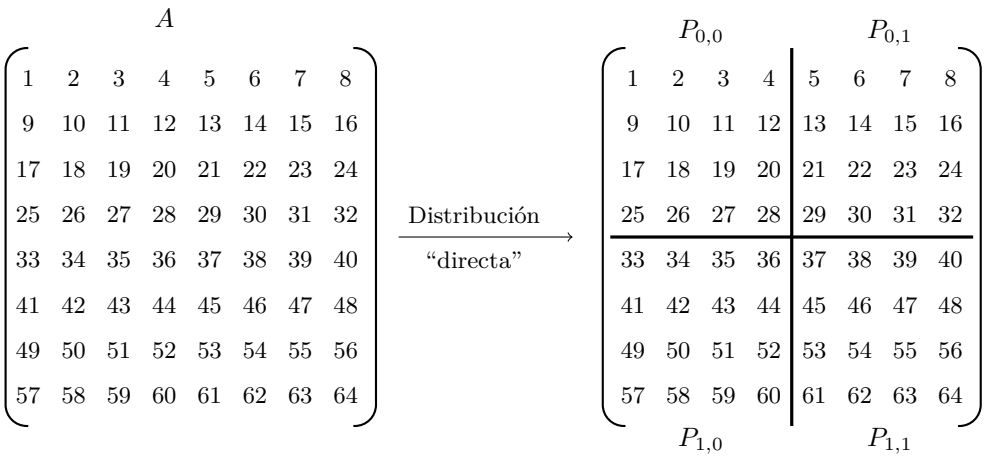


Figura 3.15: Ejemplo de una matriz A, de dimensiones (8×8), distribuida "directamente" sobre una malla de procesadores (2×2)

Lo que proponemos es resolver un sistema de forma paralela como si estuviera distribuido 2DBC y así evitar el paso de redistribución. En concreto estaríamos resolviendo el sistema equivalente:

$$P^{-1}AQ^{-1}y = P^{-1}b, \quad y = Qx \tag{3.4}$$

donde P y Q son las matrices de permutación ScaLAPACK y P^{-1}, Q^{-1}, las matrices de permutación inversa ScaLAPACK. Por lo tanto, podemos aplicar la rutina de ScaLAPACK a la matriz distribuida "directamente". Luego obtendríamos x con un pequeño paso de redistribución ($Q^{-1}y$), que para el caso de un vector es menos costoso.

En la figura 3.16 se muestra la distribución 2DBC con tamaño de bloque (2×2) de un sistema permutado inverso ScaLAPACK. Se puede verificar que el sistema permutado distribuido 2DBC es equivalente al sistema distribuido directamente.

Lo anterior significa que cuando resolvemos sistemas de ecuaciones con la librería ScaLAPACK, podemos pasar de una distribución "directa" a una distribución 2DBC sin costo alguno de comunicaciones, salvo el generado por la redistribución del vector de incógnitas calculado. Esto puede ser muy útil si consideramos que hay operaciones (que pueden ser anteriores a la solución del sistema) que son más eficientes con una distribución "directa". Ese es el caso del cálculo de la DWT por bloques que veremos en el subepígrafe siguiente.

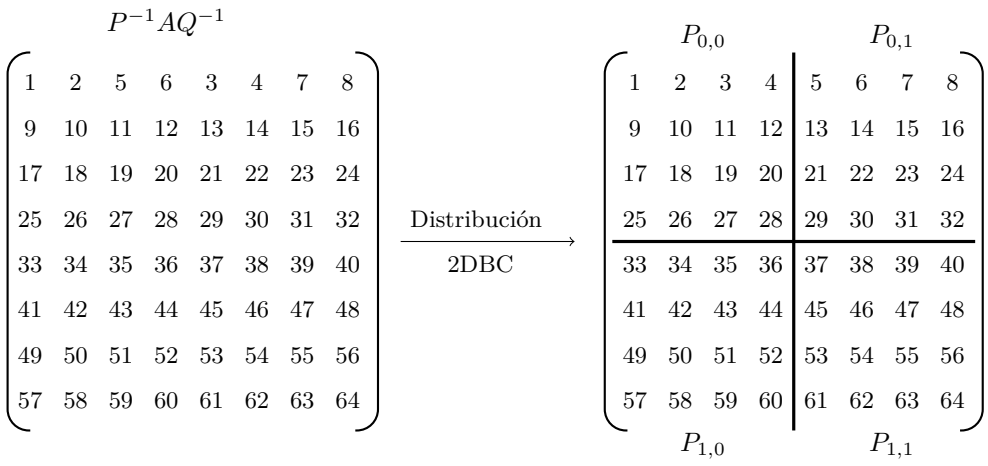


Figura 3.16: Ejemplo de una matriz A permutada inverso a ScaLAPACK distribuida 2DBC, con tamaño de bloque (2×2), sobre una malla de procesadores (2×2)

Cálculo paralelo eficiente de la DWT de una matriz distribuida 2DBC

Para calcular la DWT paralela y por bloques de una matriz, lo ideal es que los elementos estén distribuidos usando la distribución "directa" de la figura 3.15. En esta distribución los elementos son localmente consecutivos, por lo cual para completar la trasformada, solo se necesitarían comunicaciones en los extremos de los bloques locales, a diferencia de la distribución 2DBC donde se necesitan comunicaciones para cada bloque, dada su naturaleza cíclica, tal y como se consideró en el epígrafe 3.3.2.

Partiremos de una escenario donde se aplica la DWT a un sistema para luego resolver:

$$WAW^T(Wx) = Wb \tag{3.5}$$

Inicialmente el sistema está distribuido usando la distribución "directa", esto permite que el cálculo de la DWT sea menos costoso. Luego hay que resolver el sistema en sí. Para ello seguimos la idea planteada en el epígrafe anterior. Es decir consideramos que el sistema transformado wavelet es una permutación inversa ScaLAPACK de otro sistema equivalente, por lo tanto en la práctica resolveríamos el sistema equivalente:

$$P^{-1}WAW^TQ^{-1}(Q(Wx)) = P^{-1}Wb \tag{3.6}$$

3.4. Resumen de aportaciones

En este capítulo se estudiaron los algoritmos paralelos más relevantes de la literatura para el cálculo de la DWT no estándar. Se propuso una variante de paralelización que generaliza los ya existentes. Se propuso una forma de calcular la cantidad de elementos necesarios para comunicar valores solo una vez al inicio del algoritmo; o comunicar a intervalos más espaciados cuando la cantidad de niveles a calcular de la DWT es grande lo que evitaría las comunicaciones con un coste relativamente pequeño de replicación de los cálculos, estos resultados se han presentado en la ponencia "Partial Data Replication as a strategy for Parallel Computing of the Multilevel Discrete Wavelet Transform", aceptado para su presentación en la conferencia internacional "International Conference on Parallel Processing And Applied Mathematics, PPAM 2009", referencia [8].

Se estudió el cálculo paralelo de la DWT para la distribución 2DBC, cuyo uso se analiza también en el capítulo 5 en el epígrafe 5.1 y que dio lugar a la publicación "Compatibility of ScaLAPACK with the discrete wavelet transform" que fue presentado en la "International Conference on Computational Science -ICCS 2007" y publicado en las "Lecture Notes in Computer Science", en el año 2007, referencia [7]. Finalmente se mejoró este algoritmo buscando una permutación de los datos que disminuye la cantidad de comunicaciones.

Capítulo 4

Cálculo de la DWT sobre matrices dispersas

Una gran cantidad de sistemas de ecuaciones lineales surgen del proceso de solución de sistemas en derivadas parciales (PDE[1]). Muchos de estos sistemas aunque son de gran dimensión tienen la característica de tener poca densidad de elementos no nulos (sistemas dispersos). Las representaciones computacionales más eficientes para este tipo de sistemas se basan en almacenar sólo los elementos no nulos junto a distintos vectores de índices que permiten ubicarlos en la matriz según la representación que se use. Ejemplo de este tipo de representación puede ser el formato coordenado (COO) que usa MATLAB para representar sus matrices dispersas, los formatos comprimidos por columnas o por fila (CSC y CSR), etc.

4.1. Cálculo secuencial de la DWT en una matriz dispersa estilo CSR

En la bibliografía podemos encontrar algunos ejemplos de aplicación de la DWT en sistemas dispersos [16]. Generalmente se construye explícitamente la matriz (2.33) y se realiza el producto (2.37) usando cualquier rutina de multiplicación de matrices dispersas de las existentes. Si bien esta solución pudiera ser eficiente en cuanto a tiempo porque se usan librerías altamente fiables y rápidas, no lo es en cuanto al uso

[1]Tomado del inglés, Partial Differential Equations. Ecuaciones en derivadas parciales

de memoria. Para darse cuenta de ello basta analizar que para construir la matriz (2.33) se toman los dos filtros h y g y se replican en cada fila de H y G desplazando, cada vez, dos unidades más en la columna respecto a la fila anterior como se aprecia en (2.34) y (2.35). O sea que para calcular la DWT-2D utilizan innecesariamente $n^2 - n * L = n * (n - L)$ localizaciones de memoria.

En su forma habitual todos los elementos de una matriz se almacenan consecutivamente en memoria, ya sea por filas o por columnas. Esto tienen sentido cuando la gran mayoría de los elementos de la misma son representativos (distintos de cero). Para las matrices donde la gran mayoría de los elementos son ceros existen representaciones que solo almacenan los elementos distintos de cero. La diferencia entre estas representaciones está en la forma que indexan estos elementos no nulos. En general no hay consenso en cual de ellas usar, por ejemplo MATLAB usa una representación formada por tres vectores de igual dimensión, el primero para los elementos no nulos, y los dos restantes para almacenar el índice de la fila y columna respectiva del elemento no nulo. Esta representación se llama usualmente COO, nombre que proviene de que se tienen las coordenadas exactas de cada uno de esos elementos en la matriz. Las representaciones básicas de una librería como SPARSKIT [57] por su parte tratan de optimizar el formato COO comprimiendo los vectores para las filas o las columnas CSR y CSC. Los algoritmos, por ejemplo para el cálculo de una matriz dispersa por un vector, varían para el tipo de representación. Así las librerías que trabajan con matrices dispersas como SPARSKIT usan como base una representación, en este caso, CSR, y proveen de rutinas eficientes que cambian de una representación a otra.

En este capítulo nos centraremos en calcular la trasformada wavelet, para una matriz dispersa con formato CSR, sin construir explícitamente la matriz W.

Inicialmente veamos las características del formato CSR.

4.1.1. Formato de almacenamiento CSR

En una matriz dispersa almacenada en formato CSR, los elementos distintos de cero se almacenan en localizaciones continuas de memoria. El formato CSR utiliza 3 vectores con la siguiente notación:

a Almacena los elementos reales distintos de cero de la matriz en formato denso A.

ja Almacena los índice de columna correspondiente al elemento no nulo de a.

ia Se usa para almacenar las localizaciones en el vector a que inician una fila.

A modo de ejemplo consideremos la matriz A:

Forma densa de A.

$$A = \begin{pmatrix} 10 & 0 & 0 & -2 \\ 3 & 0 & 9 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 8 & 4 \end{pmatrix}$$

Forma Dispersa de A. (CSR)

a	10	-2	3	9	7	8	4
ja	1	4	1	3	2	3	4
	ia	1	3	5	6		

Hay que tener en cuenta los elementos siguientes para diseñar cualquier algoritmo usando este formato:

1. Cualquier fila i se encuentra localizada en el rango $[ia[i], ia[i+1]-1]$. La fila 1 en el ejemplo está localizada en $[ia[1]=1, ia[2]-1=2]$, o sea los elementos del vector a, 10 y -2.

2. No se puede acceder directamente a un elemento $[i, j]$. En su lugar hay que usar un índice k para recorrer la fila según el rango $[ia[i], ia[i+1]-1]$ y comparar con el índice de la columna almacenado en $ja[k] = j$ para acceder finalmente al elemento almacenado en $a[k]$.

3. La cantidad de elementos no nulos de la matriz se puede calcular como $ia[n]+1$ donde n es la cantidad de filas de la matriz. Así en nuestro ejemplo, la cantidad de elementos no nulos de A se puede calcular como $ia[4]+1$, o sea 7.

En particular, para almacenar un vector disperso usando este formato es necesario tener el vector de elementos no nulos y los índices de las columnas correspondientes. Por ejemplo para la fila 1 de la matriz A, la denotaremos v, tendríamos:

v	10	-2
jv	1	4

Es necesario también tener un valor adicional que indique la cantidad real de elementos del vector.

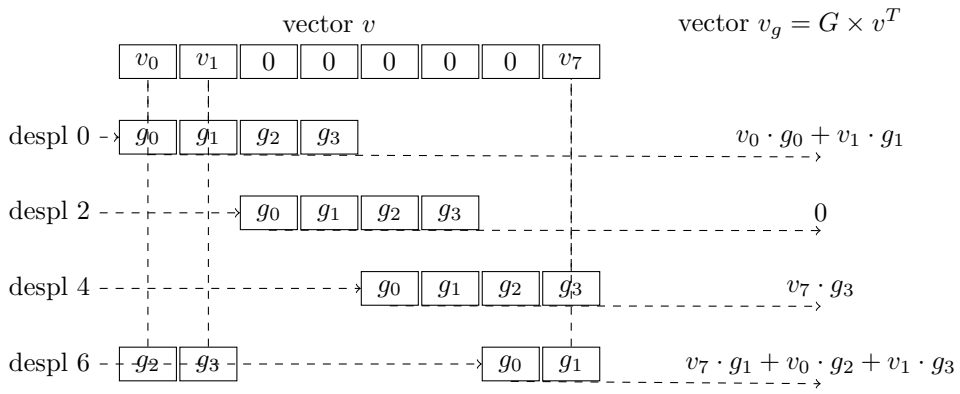


Figura 4.1: DWT-1D de baja frecuencia con filtro de dimensión 4 sobre un vector de longitud 8.

4.1.2. Transformada wavelet de un vector disperso

Analizaremos inicialmente el cálculo de la DWT de un vector disperso y luego generalizaremos el resultado a aplicar la DWT por filas a una matriz dispersa.

Supongamos inicialmente que tenemos un vector v denso, con muchos elementos nulos, y queremos aplicarle la DWT-1D, ver la figura 4.1. En esta figura se muestran los pasos de convolución del vector v con un filtro de baja frecuencia G. Hemos resaltado intencionalmente que tiene sentido realizar solo las operaciones con los elementos de v distintos de 0. Por ejemplo, para la posición 1 en v_g no se debe efectuar ninguna operación porque no hay ningún elemento distinto de cero en la posiciones $2, 3, 4, 5$ de v.

Cuando el vector a transformar tiene muchas entradas con valor 0 se hace necesario buscar una variante para calcular la transformada sin necesidad de operar con los elementos nulos. Para ello representaremos este mismo vector en un formato disperso y lo denotaremos $\langle v, jv, n\rangle$

v	v_0	v_1	v_7
jv	0	1	7
n	8		

donde v son los valores distintos de cero en el vector original, jv son los índices de las columnas correspondientes y n es la longitud del vector original.

Queremos ahora diseñar un algoritmo para hacer el cálculo de la DWT-1D de un

vector en formato disperso. Dentro de las características deseables de este algoritmo, queremos que su coste dependa de la cantidad de elementos no nulos del vector y no de la longitud del mismo.

La dificultad con esta representación radica en encontrar para cada columna donde aparezca un elemento no nulo en el vector disperso, qué columnas aparecen, resultado de la convolución, en el vector trasformado donde dicho valor está involucrado.

Tomemos por ejemplo la convolución de $\langle v, jv, n\rangle$ con el filtro G, de dimensión 4, a la que llamaremos $\langle v_g, iv_g, n_g\rangle$:

v_g	$v_0 \cdot g_0 + v_1 \cdot g_1$	$v_7 \cdot g_3$	$v_7 \cdot g_1 + v_0 \cdot g_2 + v_1 \cdot g_3$
iv_g	0	2	3
n_g	8/2=4		

Aquí el elemento v_7, de la posición 7 en el vector original v, está involucrado en el cálculo de los coeficientes de las posiciones 2 y 3 del vector transformado v_g. El elemento v_0 está involucrado en el cálculo de los índices 0 y 3 en el vector transformado v_g.

De forma general podemos determinar exactamente, dado el índice del elemento distinto de cero en el vector original, a qué posiciones contribuye en el vector transformado. Notar inicialmente que cualquier valor distinto de 0 en el vector original está involucrado en el cálculo de $\frac{L}{2}$ elementos en el vector trasformado, donde L es la longitud del filtro wavelet.

Dado un índice c del vector de índices jv, denotaremos como $ColTra(k, c, n_g)$ la posición a la que contribuye el índice jv_c, en el vector trasformado de longitud n_g, donde k toma valores en el rango $k = 0 \ldots \frac{L}{2} - 1$.

$$ColTra(k, c, n_g) = (\frac{jv_c}{2} + n_g - k)\,\%n_g \tag{4.1}$$

Nuevamente, en el ejemplo, la posición $c = 2$, $jv_c = 7$ está involucrada en el cálculo de los índices $ColTra(0, 2, 4) = (7/2 + 4 - 0)\,\%4 = 3$ y $ColTra(1, 2, 4) = (7/2 + 4 - 1)\,\%4 = 2$ y la posición 0 en el cálculo de las posiciones $ColTra(0, 0, 4) = (0/2 + 4 - 0)\,\%4 = 0$ y $ColTra(0, 0, 4) = (0/2 + 4 - 1)\,\%4 = 3$.

Otro problema es que una vez encontrada el índice $ColTra(k, c, n_g)$ en el vector transformado es necesario saber con cuál elemento del filtro se multiplica v_c, es decir,

qué índice del filtro le corresponde. Por ejemplo se puede observar que v_2 que está en la posición 7 del vector original, cuando se hace la convolución dicho valor contribuye a los valores de las posiciones 2 y 3 en el vector resultante, en la posición 2 se multiplica con el valor g_3 del filtro g y en la posición 3 por el valor h_1. La siguiente fórmula calcula, conociendo la columna a la que tributa un valor no nulo en el vector original, el índice del filtro wavelet por el que es necesario multiplicarlo.

$$IndFiltr_{ColTra(k,c,n_g)} = \begin{cases} (n + jv_c - 2 * ColTra(k,c,n_g))\,\%L, \\ \text{Si} \quad jv_c - 2 * ColTra(k,c,n_g) < 0 \\ jv_c - 2 * ColTra(k,c,n_g), \quad \text{En otro caso} \end{cases} \tag{4.2}$$

Retomando el ejemplo, v_7 que está originalmente en la posición 7, cuando se hace la convolución contribuye a los valores de las posiciones 2 y 3 en el vector resultante, en la posición 2 se multiplica con el índice $(8 + 7 - 2 \times 2)\,\%4 = 3$ o sea el valor g_3 del filtro g y en la posición 3 con el índice $(8 + 7 - 2 \times 3)\,\%4 = 1$ o sea por el valor g_1 del filtro.

Teniendo los índices $ColTra$ y $IndFiltr$, índices del elemento no nulo en la matriz transformada e índice del filtro correspondiente, que se calculan usando las ecuaciones (4.1) y (4.2) respectivamente, el procedimiento que se sigue para completar la trasformada es simple. Para cada elemento no nulo del vector disperso se calculan las columnas a las que tributa en el vector transformado $ColTra(k,c,n_g)$ y el índice del filtro con el que se multiplica $IndFiltr_{ColTra(k,c,n_g)}$ se efectúa la operación y el valor resultante se acumula en la casilla correspondiente a la columna $ColTra(k,c,n_g)$, ver el algoritmo 4.1.

Algoritmo 4.1 Trasformada Vector Disperso

Entrada: vector disperso v, jv, n, el filtro F de longitud L y nz la cantidad de elementos no nulos del vector v.

Salida: El vector convolución wv, jwv, wn y wnz.

1. `Inicializar un vector` iw `con longitud` $\frac{n}{2}$ {Esta variable se usa para tener referencia de las columnas que vayan surgiendo en el proceso de cálculo}
2. $len := 0$ {Esta variable se usa para tener un índice del último índice actualizado en el vector transformado}
3. **para** $c := 0$ **to** $nz - 1$ **hacer**
4. **para** $k := 0$ **to** $L/2 - 1$ **hacer**
5. $ct = ColTra(k, c, \frac{n}{2})$ (4.1)
6. $if = IndFiltr(k, c, \frac{n}{2})$ (4.2)
7. **si** $iw[ct] <> 0$ {Si ya se ha generado un no nulo para esta columna} **entonces**
8. $wv[iw[ct]] += v(j) * F[if]$
9. **si no**
10. $iw[ct] := len$
11. $jwv[len] := ct$
12. $wv[iw[ct]] := v(j) * F[if]$
13. $inc(len)$
14. **fin si**
15. **fin para**
16. **fin para**
17. $wnz = len$

4.1.3. Transformada wavelet por filas de una matriz dispersa

La transformada *wavelet* por filas de una matriz se basa en convolucionar cada fila de la matriz con un par de filtros digitales, representados por $\{g_i\}_{i=0}^{L-1}$, de baja frecuencia, y $\{h_i\}_{i=0}^{L-1}$, de alta frecuencia.

Para hacer el cálculo por filas de una matriz almacenada en el formato CSR, consideraremos cada fila de la matriz como un vector disperso y aplicaremos independientemente el **algoritmo 4.1** a cada una de estas filas. Este procedimiento se muestra en el **algoritmo 4.2**.

4.1.4. Convolución por columnas de M vectores dispersos

Para derivar un procedimiento de cálculo de la DWT de una matriz en la dirección de las columnas, analizaremos en primer lugar el caso de la convolución de M vectores dispersos representados en formato CSR.

Algoritmo 4.2 Trasformada wavelet por filas de una matriz dispersa

Entrada: a, ia, ja Matriz a transformar en formato CSR, F el filtro wavelet pasa-alto o pasa-bajo de longitud L.

Salida: La matriz wa, iwa, jwa equivalente a $A * H^T$ o $A * G^T$ según el filtro.

1. **para** $i := 0$ **to** $n - 1$ **hacer**
2. $v := a[ia[i]] : a[ia[i + 1]]$
3. $jv := ja[ia[i]] : ja[ia[i + 1]]$
4. Aplicar el **Algoritmo 4.1** con , v, jv, F, L para obtener el vector trasformado (wv, jwv, wn){Convolución de cada fila}
5. Agregar la fila (wv, jwv) a la matriz wa, iwa, jwa
6. **fin para**

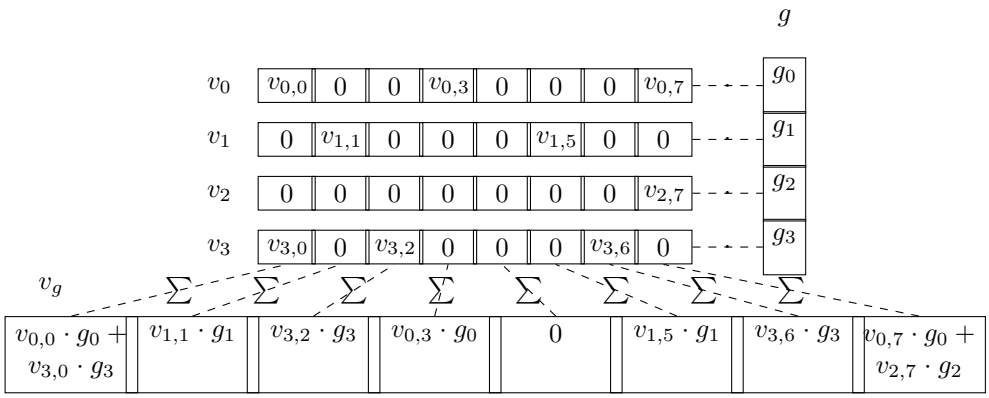


Figura 4.2: Ejemplo de la representación de 4 vectores dispersos en forma densa. Se convolucionan en la dirección de las columnas con g el resultados se almacena en v_g

Tomemos como ejemplo la convolución de 4 vectores con muchos elementos nulos ($v_0 \dots v_3$) y el filtro g de dimensión $L = 4$ en la dirección de las columnas. Cada elemento $v_{k,i}$, $k = 0 \dots 3$ e $i = 0 \dots 7$, se multiplica con su correspondiente g_k y se produce la suma $\sum_{k=0}^{3} v_{k,i} \cdot g_k$ que se coloca en la columna i del vector resultante v_g, ver la figura 4.2.

Sin embargo nuestro objetivo es operar y almacenar solo con los elementos no nulos. Supongamos ahora que tenemos los mismos 4 vectores almacenados de forma consecutiva, en formato CSR, en un solo vector disperso $\langle v_g, jv_g, iv_g \rangle$, ver la figura 4.3.

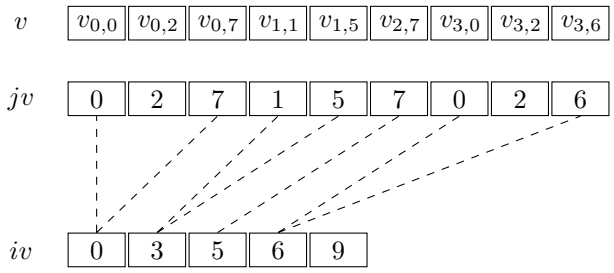


Figura 4.3: Ejemplo de 4 vectores dispersos en formato CSR.

Para hacer la convolución se usará un vector denso denotado por iw, de longitud n, para almacenar la posición de los elementos no nulos de v_g que vayan surgiendo en el proceso y que sirve además para identificar cuando se ha generado un no nulo en la columna correspondiente. Un cero en iw_k significa que no se ha generado aún ningún elemento distinto de cero en la posición k de v_g. Como estamos trabajando las posiciones en los vectores en base 0, tal y como lo hace el lenguaje "C", hay que aclarar que excepcionalmente este vector almacenará las posiciones en base 1 para no confundir un cero que significa que no se ha generado ningún elemento en esa columna con la posición 0. Iremos construyendo el vector v_g de la siguiente forma:

Tomamos el primer vector disperso v_0

v_0	$v_{0,0}$	$v_{0,2}$	$v_{0,7}$

En una sola pasada sobre cada vector iremos construyendo el vector resultante $\langle v_g, jv_g, 8\rangle$ y el vector auxiliar iw.

Primero con v_0 y con el elemento del filtro g_0:

iw	1	0	2	0	0	0	0	3

wa	$v_{0,0} * g_0$	$v_{0,2} * g_0$	$v_{0,7} * g_0$
ja	0	2	7

Luego con v_1 y con el elemento del filtro g_1:

iw	1	4	2	0	0	5	0	3

wa	$v_{0,0} * g_0$	$v_{0,2} * g_0$	$v_{0,7} * g_0$	$v_{1,1} * g_1$	$v_{1,5} * g_1$
ja	0	2	7	1	5

Hasta ahora, cada vez que se tiene un elemento distinto de cero, se multiplica por el filtro correspondiente y se agrega al vector resultante porque no han coincidido dos elementos con la misma columna. Veamos la pasada sobre v_2 y con el elemento del filtro g_2:

iw	1	4	2	0	0	5	0	3

wa	$v_{0,0} * g_0$	$v_{0,2} * g_0$	$v_{0,7} * g_0 + v_{2,7} * g_2$	$v_{1,1} * g_1$	$v_{1,5} * g_1$
ja	0	2	7	1	5

Aquí se ve que el elemento distinto de cero k_7 está en la columna 7 del vector v_2, pero ya se ha agregado un elementos para esa columna, $d_7 * g_0$, eso se puede saber porque $iw\,[7] \neq 0$ por lo tanto se modifica el valor acumulado para esta columna accediendo al mismo en aw a través del índice $iw\,[7] - 1$. El algoritmo 4.3 muestra el proceso formalizado para M vectores dispersos.

Algoritmo 4.3 Trasformada de M Vectores dispersos

Entrada: a, ia, ja M vectores en formato CSR, con $M = L$. F el filtro wavelet pasa-alto o pasa-bajo de longitud L.
Salida: Un vector wa, iwa, jwa, de longitud $\frac{n}{2}$.

1. $len = 0$
2. Inicializar el vector iw con tamaño n y valor cero
3. **para** $fila := 0$ **to** $M - 1$ **hacer**
4. $\quad InfFiltr = fila$
5. $\quad$ **para** $k := ia[fila]$ **to** $ia[fila + 1]$ **hacer**
6. $\quad\quad Col = ja[k]$
7. $\quad\quad$ **si** $iw[inn] <> 0$ {Si ya se ha generado un no nulo para esta columna} **entonces**
8. $\quad\quad\quad wa[iw[Col]] += a[k] * F[IndFiltr]$
9. $\quad\quad$ **si no**
10. $\quad\quad\quad iwa[Col] := len$
11. $\quad\quad\quad jwa[len] := ColTra$
12. $\quad\quad\quad wa[iwa[Col]] := a[k] * F[IndFiltr]$
13. $\quad\quad\quad inc(len)$
14. $\quad\quad$ **fin si**
15. $\quad$ **fin para**
16. **fin para**

4.1.5. Transformada wavelet por columnas de una matriz dispersa

La transformada wavelet por columna se basa en realizar la convolución de cada vector columna con un par de filtros wavelet, g y h. Por ejemplo para una matriz A es obtener las matrices $H * A$ usando un filtro *pasa-alto* en su forma matricial H y la matriz $G * A$ usando un filtro *pasa-bajo* representado por G.

Aplicar una transformada por columnas a una matriz en formato CSR tiene la dificultad que los elementos están almacenado por filas. El procedimiento de separar la columna, aplicar la transformada al vector y luego ir construyendo la matriz resultante sería muy costoso. En su lugar hay que idear una solución que nos permita aplicar las transformadas por columnas pero en la dirección de las filas.

En el epígrafe 4.1.4 se definió un procedimiento para calcular una convolución por columnas para M vectores dispersos con un filtro de longitud L en la dirección de las filas. El resultado de esta operación es un vector disperso.

Solo resta decir que para calcular la DWT por columnas de una matriz dispersa se toman grupos de L filas, donde L coincide con la longitud del filtro wavelet, comenzando por la fila 0, y desplazando la inicial en 2 filas cada vez, aplicar el algoritmo 4.3 sobre ese grupo de L filas e ir agregando los vectores obtenidos como filas de la matriz resultante. Hay que tener en cuenta que si $L > 2$ entonces cuando el índice de la fila inicial es $N - L + 2$ será necesario completar con filas de forma cíclica, es decir tomando del inicio de la matriz.

El **algoritmo 4.4** muestra el proceder a seguir.

Algoritmo 4.4 Trasformada wavelet por columnas

Entrada: a, ia, ja Matriz a transformar en formato CSR. F el filtro wavelet pasa-alto o pasa-bajo de longitud L

Salida: La matriz wa, iwa, jwa ($H*A$ ó $G*A$) según el filtro F.

1. **para** $i := 0$ **to** $n - 1$ **step** 2 {Solo filas alternas por la decimación} **hacer**
2. $v := a[ia[i]] : a[ia[i+L+1]-1]$ {Se toman L filas consecutivas de la matriz, considerando que cuando se llegue a la última fila se completa con las primeras de forma cíclica}
3. $jv := ja[ia[i]] : ja[ia[i + L + 1] - 1]$
4. Aplicar el **Algoritmo 4.3** con , v, jv, F, L {Convolución de L filas, por columna}
5. Agregar la fila (wv, jwv) resultado del algoritmo anterior a la matriz wa, iwa, jwa
6. **fin para**

4.2. Problema de rellenado: Algoritmos de reordenamiento

Un problema al aplicar la DWT-2D a una matriz dispersa es que la matriz transformada tiene un mayor porcentaje de elementos no nulos que la matriz original. Este es el mismo problema que ocurre cuando se realiza alguna factorización sobre la matriz dispersa, como la LU, y se le conoce como fenómeno de llenado (*fill-in*). Este es un fenómeno muy perjudicial, porque aumenta la cantidad de espacio de memoria para almacenar las matrices y además la cantidad de cálculos aritméticos en operaciones posteriores. Un ejemplo de este fenómeno lo podemos apreciar en la figura 4.4.

En este ejemplo se parte de una matriz (8×8) con 13 elementos distintos de cero, una vez que se aplica la DWT-2D la cantidad de elementos no nulos es de 44 en una matriz de las mismas dimensiones; esto hace que la densidad de no nulos aumente considerablemente en un solo nivel wavelet. Si luego se aplica otro nivel wavelet a la submatriz de aproximación entonces la matriz resultante será densa sin ninguna duda.

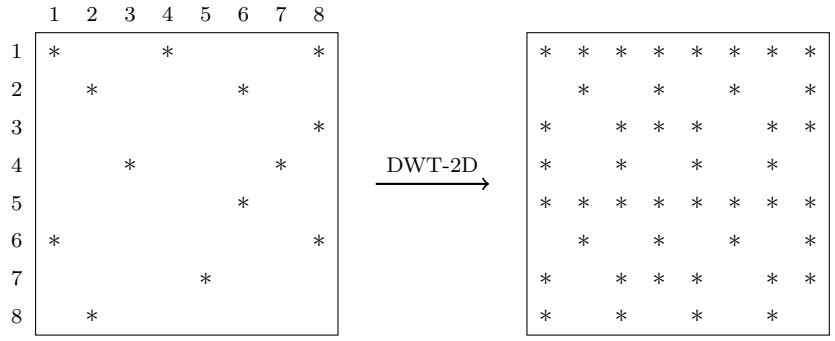


Figura 4.4: Ejemplo de rellenado al aplicar la DWT 2D a una matriz dispersa usando un filtro Haar.

La cantidad de no nulos que surgen en este proceso puede depender de la posición que ocupen los elementos distintos de cero en la matriz original y del tamaño del filtro wavelet que se use para aplicar la transformada.

Ahora transformemos la matriz (8×8) de la figura 4.4, usando el mismo filtro wavelet y con la misma cantidad de no nulos, pero en este caso aplicando una permutación de las filas y las columnas que trata de colocar los elementos distintos de cero

lo más cerca posible unos de otros. Este escenario se puede observar en la figura 4.5; nótese que aplicando esta permutación el llenado disminuye a 31 elementos.

Partiremos de la tesis de que cuando se calcula la DWT-2D es deseable que los elementos en la matriz estén lo más cerca posible unos de otros. La idea es que cuando se aplique la convolución tanto por filas como por columnas con los filtros wavelet surjan la menor cantidad de elementos distintos de cero.

La permutación de filas y columnas de una matriz para lograr cierta configuración que disminuya el rellenado es un tema bastante tratado en la literatura, usualmente para mejorar el desempeño de operaciones sobre matrices dispersas como las factorizaciones de *Cholesky* y LU. Lo más representativo entre las técnicas de reordenamiento son los algoritmos de reducción en banda que derivan del algoritmo de *Cuthill-McKee*[25], y las heurísticas basadas en el criterio de *Markowitz* [46] como los algoritmos *Minimum Degree*, *Approximate Minimum Degree*[9][29], entre otras [58]. Otros enfoques tratan de reordenar la matriz para obtener bloques de filas y columnas densos y así minimizar los fallos de cache en la operación producto matriz-vector [53].

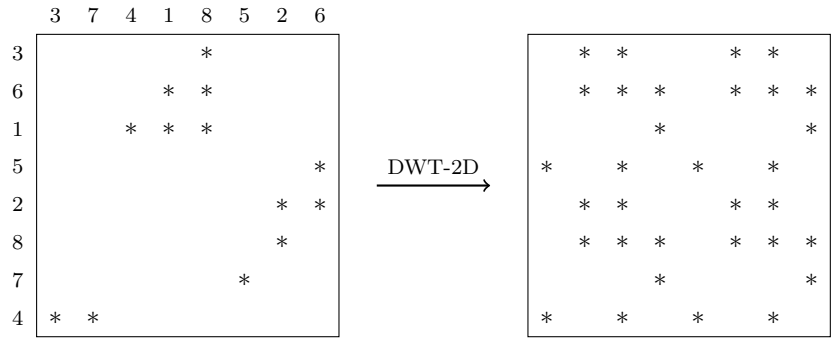


Figura 4.5: Ejemplo de rellenado al aplicar la DWT 2D a una matriz dispersa usando un filtro Haar permutando los elementos de la matriz original a una configuración más conveniente.

Utilizaremos estos dos enfoques para afrontar el problema del cálculo de la DWT en matrices dispersas, un problema del que no hay referencias en la literatura.

4.2.1. Criterio de *Markowitz*

El criterio de *Markowitz* [46] se diseñó para disminuir el efecto de rellenado cuando se realiza la descomposición LU de una matriz dispersa. Esta estrategia "voraz" de optimización local se puede describir en forma resumida como sigue. Después de k pasos de la eliminación Gaussiana, denotaremos r_i^k y c_j^k, como el número de elementos no nulos en la fila i y la columna j de la submatriz restante $(n-k)\times(n-k)$. El criterio de *Markowitz* es seleccionar como próximo pivote la entrada distinta de cero a_{ij} que tenga el mínimo contador de *Markowitz* $\left(r_i^k - 1\right) \times \left(c_i^k - 1\right)$. Con esto se trata de minimizar la cota superior de rellenado que se produciría en el paso $k+1$.

La implementación del criterio de *Markowitz* se hace ineficiente en el sentido en que en cada paso se deben actualizar las entradas no nulas de la submatriz restante.

4.2.2. Algoritmos de *Cuthill-McKee*

Uno de los métodos más utilizados para la transformación en banda de una matriz dispersa es el conocido con el nombre de algoritmo de *Cuthill-McKee* (CM) [25] y el algoritmo *Cuthill-McKee* inverso (RCM) [3] que básicamente es el mismo algoritmo pero numerando los índices resultantes de forma inversa y en la práctica da los mejores resultados [42].

Este algoritmo se basa en la teoría de grafos y su objetivo es encontrar una permutación que disminuya el ancho de banda de una matriz dispersa y simétrica, para tratar de atenuar el efecto de rellenado al aplicar una eliminación gaussiana sobre la matriz.

La distribución de las entradas no nulas de una matriz se representa mediante un grafo. Dada una matriz simétrica A, si la entrada a_{ij} de A es distinta de cero, los nodos i y j están conectados en el grafo mediante una arista. La eliminación gaussiana en la matriz queda reflejada en el grafo al eliminar el nodo correspondiente y aparecer nuevas conexiones entre los nodos restantes, correspondiendo cada una de ellas a un nuevo elemento distinto de cero en A.

La idea del algoritmo es muy simple: Si a es un vértice ya renumerado y b es un vértice no renumerado aún, conectado al vértice a mediante una arista en el grafo, para minimizar la anchura de la fila asociada a b es evidente que el vértice b se debe renumerar lo antes posible, inmediatamente después del vértice a.

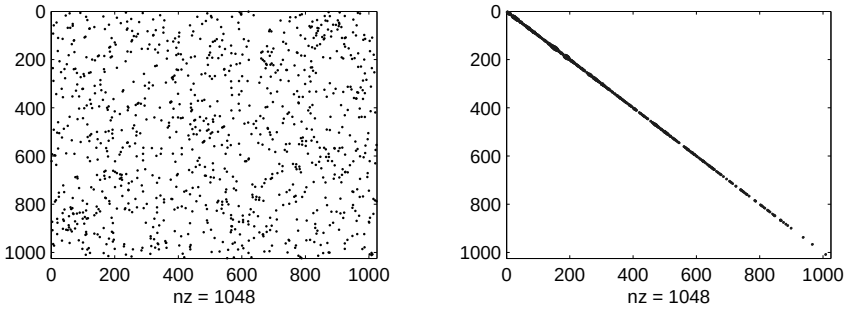
El esquema del algoritmo es el siguiente:

1. Se construye una tabla indicando el número de conexiones (que se suele llamar grado y se representa $N(x)$) de cada vértice.

2. Se elige un nodo inicial (en un extremo del grafo y con pocas conexiones), que se renumera x_1.

3. Se renumeran a continuación los vértices conectados a x_1, en orden ascendente de grado.

En la figura 4.6 se representa una matriz simétrica A de dimensiones (1024×1024) generada aleatoriamente con la rutina `sprandsym` de MATLAB. A la derecha, se representa la matriz $A(p, p)$, donde p es un vector de permutación obtenido mediante el algoritmo de ordenamiento *Cuthill-McKee* inverso (symrcm). Se puede apreciar el reordenamiento en forma de banda de dicha matriz $A(p, p)$.



(a) `randsym1024_001` (b) `randsym1024_001`Perm

Figura 4.6: En (a), una matriz dispersa simétrica generada aleatoriamente con densidad de elementos no nulos de 0.001 y en (b) la misma matriz permutada con la rutina `symrcm` de MATLAB

4.2.3. Resultados experimentales

Para realizar la totalidad de los experimentos se hizo uso de MATLAB como entorno de programación. Como métodos de ordenamiento basados en el criterio de *Markowitz* se escogieron las funciones `colamd` que calcula la permutación de grado

mínimo aproximada para matrices no simétricas y `symamd` que implementan la permutación de grado mínimo para matrices simétricas. También se escogió la función `symrcm` que calcula la permutación usando el algoritmo *Cuthill-McKee* inverso.

Las matrices seleccionadas para los experimentos vienen de distintas fuentes. Para realizar las pruebas iniciales se generaron aleatoriamente 4 matrices de prueba usando la función `sprandsym` de dimensiones 1024 y 2048 con densidad de elementos no nulos 0,1 y 0,01 en ambos casos.

Seguidamente se escogieron varias matrices desde el repositorio "Matrix Market" [1]: `fidapm37`, `plat1919`, `bcsstk27` y `e40r5000`. Finalmente se seleccionaron 4 matrices de gran dimensión desde una colección de matrices de la Universidad de Florida [28]: *Epb*0 y *Epb*1, *Epb*2 y *Epb*3. Un resumen se puede ver en la Tabla 4.1.

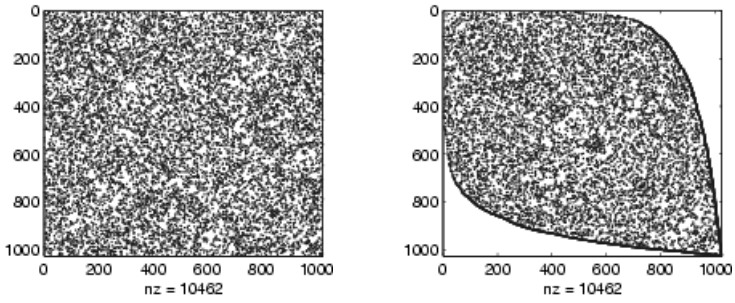
Matrices	Dimensión	Número de no Ceros
`randsym1024_01`	1024	99837
`randsym1024_001`	1024	10462
`randsym2048_01`	2048	399087
`randsym2048_001`	2048	41767
`bcsstk27`	1224	56126
`fidapm37`	9152	765944
`plat1919`	1919	32399
`e40r5000`	17281	553562
`epb0`	1794	7764
`epb1`	14734	95053
`epb2`	25228	175027
`epb3`	84617	463625

Tabla 4.1: Matrices de prueba

Para el cálculo de la DWT-2D se seleccionaron 6 juegos de filtros de "daubechies": {`db1`, `db2`, `db3`, `db4`, `db5`, `db6`}, de ordenes {1, 2, 3, 4, 6, 6} respectivamente, ver apéndice A.

En la figura 4.7 se muestra la estructura dispersa de la matriz `randsym1024_001` contrastada con las permutadas usando las funciones `symamd` y `symrcm` de MATLAB. Intuitivamente se puede estimar que tanto para la matriz `randsym1024_001` reordenada usando la función de mínimo grado simétrica 4.7(b) y la matriz 4.7(c) van a tener un menor porcentaje de elementos no nulos que 4.7(a). La razón de esto es que aparentemente hay más zonas "vacías", llamando así a espacios en la matriz donde no hay ningún elemento no nulo. Esto favorecería que cuando se calcule la DWT-2D no surjan elementos distintos de cero producto de la convolución de los filtros con esas

áreas.



(a) Orden Natural

(b) Permutada `symrcm`

(c) `symamd`

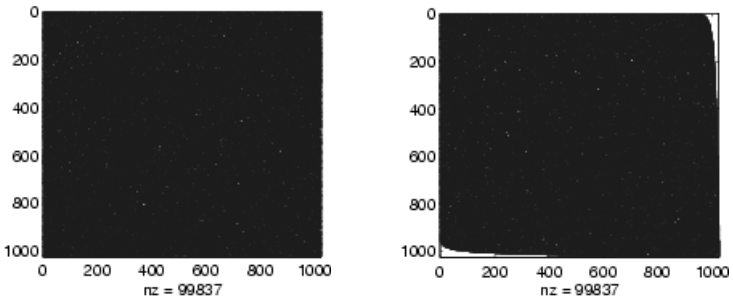
Figura 4.7: En (a) se muestra la matriz `randsym1024_001`, en (b) la misma matriz permutada con la rutina `symrcm` de MATLAB y en (c) permutada con la función `symamd`

La tabla 4.2 muestra una comparación de la cantidad de elementos no nulos que surge luego de aplicar la DWT-2D no estándar de varios ordenes. En esta comparación se puede corroborar que efectivamente la cantidad de ceros que surgen al aplicar la DWT-2D es menor cuando se aplican las permutaciones.

Filtros / Perm	db1	db2	db3	db4	db5	db6
Natural	41206	155000	316400	493796	659528	796780
symrcm	37824	134520	268952	415152	551444	664092
symamd	39526	142724	278356	415940	534772	625220

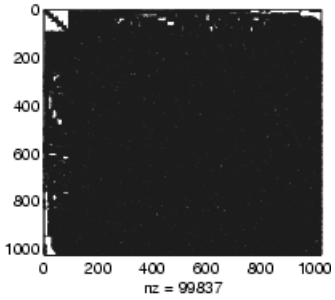
Tabla 4.2: Elementos no nulos al aplicar la DWT-2D no estándar de varios ordenes a la matriz `randsym1024_001`

La figura 4.8 muestra la representación dispersa de la matriz `randsym1024_001` contrastada con las permutadas usando las funciones `symamd` y `symrcm` de MATLAB.



(a) Orden Natural

(b) Permutada `symrcm`



(c) `symamd`

Figura 4.8: En (a) se muestra la matriz `randsym1024_01`, en (b) la misma matriz permutada con la rutina `symrcm` de MATLAB y en (c) permutada con la función `symamd`

Se puede observar que dado que la densidad de elementos no nulos es mayor, siendo la dispersión de los mismos uniforme por toda la matriz, la cantidad de "vacíos" es menor.

La tabla 4.3 muestra la comparación en cuanto a la cantidad de elementos no nulos de la matriz `randsym1024_01` en su orden natural contra la misma reordenada con `symrcm` y `symamd`. Se puede observar cierta mejora en cuanto a la cantidad de no nulos que surgen luego de aplicada la DWT-2D. No obstante la diferencia es menor que la observada con la matriz `randsym1024_01` debido a que las permutaciones no son ya tan efectivas dada la mayor densidad de no nulos.

Las tablas de 4.4(a) y 4.4(b) muestran las comparaciones entre la cantidad de no nulos de la matrices `randsym2048_001` y `randsym2048_01` respectivamente. El comportamiento es similar a los visto anteriormente, la mayor ganancia cuando la densidad de no nulos es menor.

Perm \ Filtros	db1	db2	db3	db4	db5	db6
Natural	346156	837120	1020084	1046804	1048512	1048572
symrcm	342528	823016	1003608	1033480	1037996	1040248
symamd	343874	821624	998800	1029456	1035484	1038832

Tabla 4.3: Elementos no nulos al aplicar la DWT-2D no estándar de varios ordenes a la matriz randsym1024_01

(a) randsym2048_001

Perm \ Filtros	db1	db2	db3	db4	db5	db6
Natural	164464	619912	1267328	1980392	2644416	3190872
symrcm	157052	574480	1162836	1809600	2414628	2913600
symamd	161842	597900	1195548	1827520	2390792	2832892

(b) randsym2048_01

Perm \ Filtros	db1	db2	db3	db4	db5	db6
Natural	1384056	3346596	4079524	4187108	4194088	4194304
symrcm	1374566	3316156	4047040	4162044	4174440	4178724
symamd	1374940	3306828	4031640	4148736	4165236	4172060

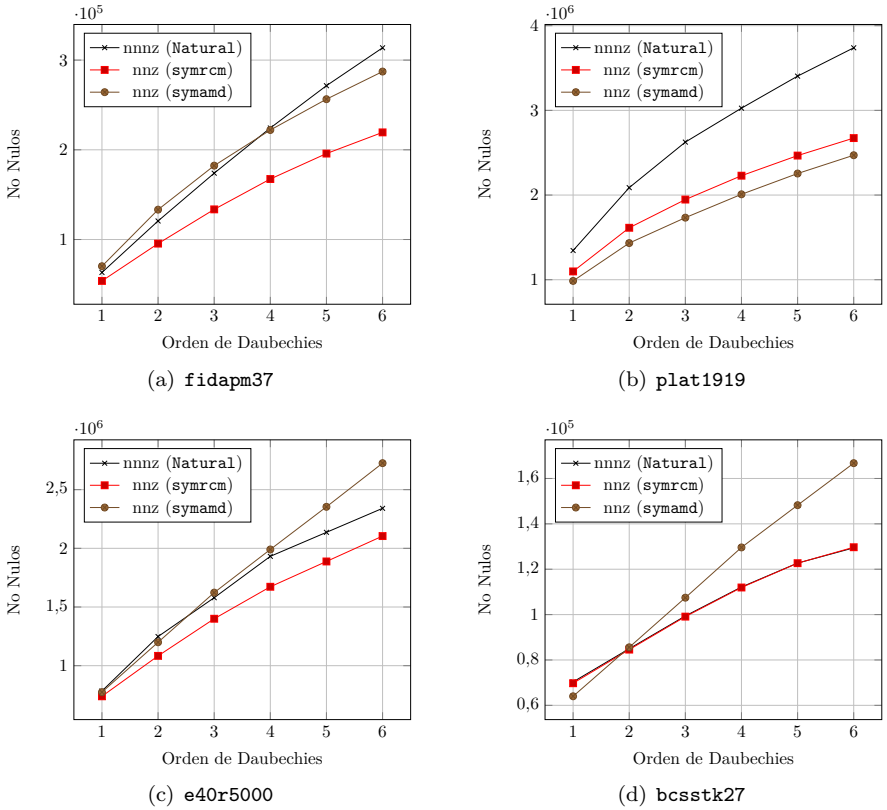
Tabla 4.4: Elementos no nulos al aplicar la DWT-2D no estándar de varios ordenes a la matriz randsym2048_001 y randsym2048_01

También se puede observar que hasta el momento la ganancia de usar un orden u otro en los ejemplos anteriores no es significativa aunque se puede decir que hay una ligera mejora en usar el reordenamiento producido por la rutina symrcm.

La figura 4.9 muestra gráficas comparativas en cuanto a la cantidad de elementos no nulos de las matrices, bcsstk27, fidapm37, plat1919 y e40r5000 en su orden natural y las mismas permutadas usando las funciones symrcm y symamd de MATLAB. Se puede observar en todos los casos ganancias permutando las matrices, aunque las diferencias más significativas se alcanzan cuando se usa el reordenamiento producido por la función symrcm de MATLAB.

Con el objetivo de verificar cómo se comportan las características de compresión de la DWT-2D contrastamos la norma de Frobenius de las matrices resultado de la descomposición DWT-2D no estándar, $A_1 = QTQ^T$, $B_1 = QTP^T$, $C_1 = PTQ^T$, $T_1 = PTP^T$, contra las producidas usando la permutación p, obtenida con la ruti-

na `symrcm` de MATLAB, $QT(p,p)Q^T$, $QT(p,p)P^T$, $PT(p,p)Q^T$, $PT(p,p)P^T$, de la matrices `fidapm37`, `plat1919` y `e40r5000`.



(a) `fidapm37` (b) `plat1919`

(c) `e40r5000` (d) `bcsstk27`

Figura 4.9: Comparación de la cantidad de elementos no nulos luego de aplicada la DWT tipo "daubechies" de ordenes $\{db1(Haar), db2, db3, db4, db5, db6\}$, de las matrices: {`fidapm37`, `plat1919`, `e40r5000`, `bcsstk27`} en su orden natural contra la matriz reordenada usado el algoritmo de *Cuthill-McKee* en reverso (`symrcm`) y el algoritmo de grado mínimo (`symamd`)

Recordemos la descomposición:

$$WTW^T = \begin{pmatrix} A_1 & B_1 \\ C_1 & T_1 \end{pmatrix}$$

Con

$$W = \begin{pmatrix} Q \\ P \end{pmatrix},$$

donde Q representa la submatriz que contiene los coeficiente de detalle y P representa la submatriz que contiene los coeficientes de aproximación. A la matriz $A_1 = QTQ^T$ recordemos que se le llama matriz de detalle y a $T_1 = PTP^T$ se le conoce como matriz de aproximación.

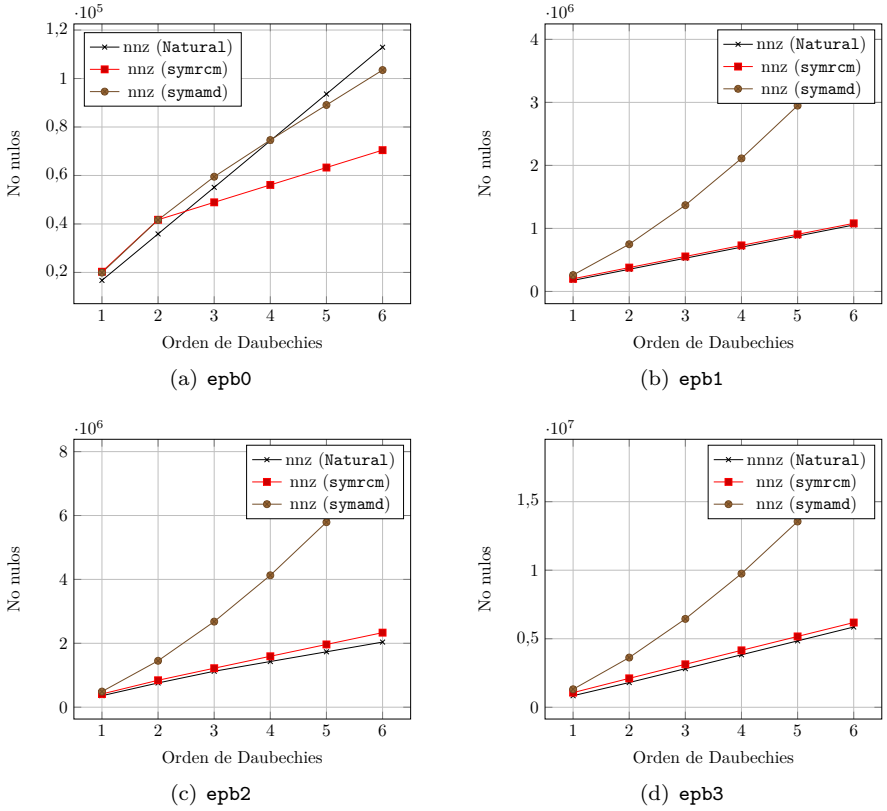
Los resultados alcanzados se pueden ver en la tabla 4.5. En todos los casos se observa una redistribución de la norma. Lo interesante es que la norma de la matriz de detalle $A_1 = QTQ^T$ es mayor que la que la correspondiente permutada en todos los casos. El peso de la matriz parece desplazarse mejor hacia las matrices de aproximación cuando se aplica un reordenamiento como `symrcm` previo a aplicar la DWT-2D. Este resultado va a ser tenido en cuenta en el capítulo 6 cuando se explique la convergencia de los métodos multimalla en algunas matrices.

	$\|A_1\|_{fro}$	$\|A_{1p}\|_{fro}$	$\|B_1\|_{fro}$	$\|B_{1p}\|_{fro}$
`fidapm37`	$6{,}12 \cdot 10^7$	$5{,}88 \cdot 10^7$	$4{,}56 \cdot 10^7$	$4{,}62 \cdot 10^7$
`plat1919`	18.35	14.81	4.21	6.84
`e40r5000`	1257.50	1115.21	744.46	952.60
	$\|C_1\|_{frod}$	$\|C_{1p}\|_{frod}$	$\|T_1\|_{frod}$	$\|T_{1p}\|_{frod}$
`fidapm37`	$4{,}56 \cdot 10^7$	$4{,}62 \cdot 10^7$	$5{,}65 \cdot 10^7$	$5{,}80 \cdot 10^7$
`plat1919`	4.21	6.84	10.90	13.36
`e40r5000`	755.32	955.90	1276.18	1126.85

Tabla 4.5: Norma de Frobenius de las matrices resultado de la descomposición DWT-2D no estándar, $A_1 = QTQ^T$, $B_1 = QTP^T$, $C_1 = PTQ^T$, $T_1 = PTP^T$, contra las producidas usando la permutación `symrcm` de MATLAB, $A_{1p} = QT(p,p)Q^T$, $B_{1p} = QT(p,p)P^T$, $C_{1p} = PT(p,p)Q^T$, $T_{1p} = PT(p,p)P^T$ de las matrices: `fidapm37`, `plat1919`, `e40r5000`. Con el filtro `db2`

Seguidamente se muestran experimentos con las matrices `epb0`, `epb1`, `epb2`, `epb3`. En la figura 4.10 se muestra una comparación de la cantidad de elementos no nulos al aplicar la DWT tipo "daubechies" de orden $\{1, 2, 3, 4, 5, 6\}$ de estas matrices en su orden natural contra la matriz reordenada usando el algoritmo de *Cuthill-McKee* en reverso (`symrcm`) y el algoritmo de grado mínimo (`symamd`). Se puede observar que solo hay ganancias en cuanto a la cantidad de no nulos que surgen en la matriz `epb0`, la explicación está en que mientras las matrices `epb1`, `epb2`, `epb3` tienen una estructura de banda bastante marcada, en la matriz `epb0` los elementos distintos de cero están más aislados por lo que ganan más espacios "vacíos" al aplicar los reordenamientos, ver las figuras 4.12 y 4.13. No obstante en estas matrices también se observan cambios en la distribución de la norma al aplicar la DWT-2D a la matriz en el orden natural y la matriz reordenada. La figura 4.11 muestra estos cambios en particular para la

matriz `epb1`.



(a) `epb0` (b) `epb1`

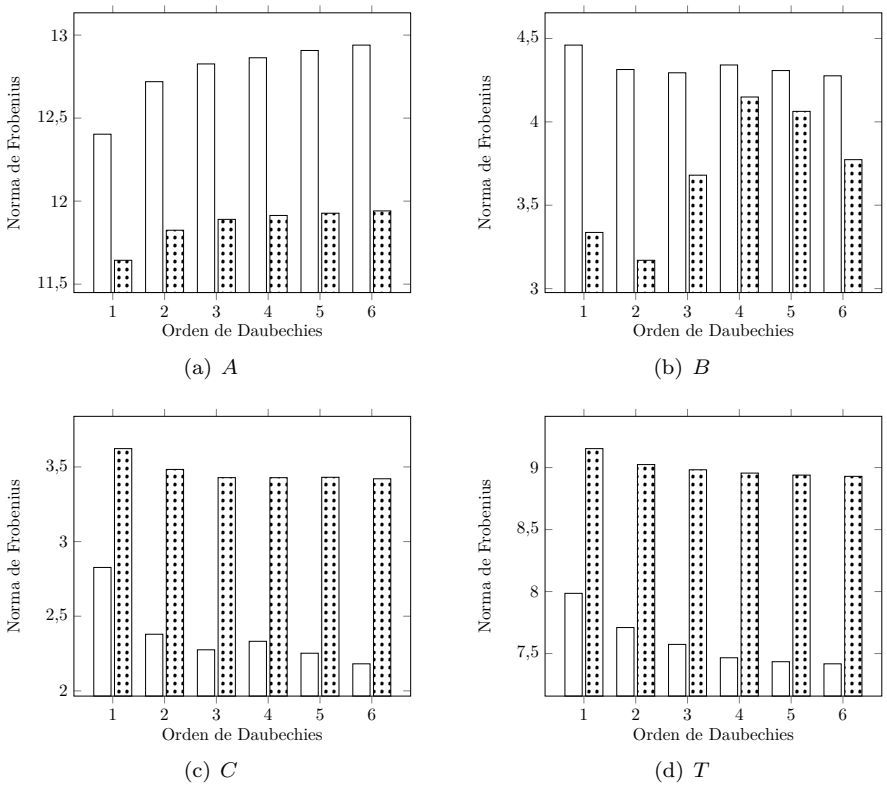
(c) `epb2` (d) `epb3`

Figura 4.10: Comparación de la cantidad de elementos no nulos al aplicar la DWT tipo "daubechies" de orden $\{1, 2, 3, 4, 5, 6\}$ de las matrices {`epb0`, `epb1`, `epb2`, `epb3`} en su orden natural contra la matriz reordenada usando el algoritmo de *Cuthill-McKee* en reverso (`symrcm`) y el algoritmo de grado mínimo (`symamd`).

4.3. Resumen de aportaciones

Este capítulo se compone de una primera parte, fundamentalmente técnica, donde se proponen algoritmos para calcular la DWT-2D no estándar en matrices dispersas sin necesidad de construir explícitamente la matriz de trasformación. Estos algoritmos usan la forma de representación CSR. Además se hace un estudio preliminar del fenómeno de rellenado que se produce cuando se calcula la DWT-2D de matrices dispersas donde se demuestra la influencia de los reordenamientos. Adicionalmente

se sugieren este tipo de reordenamiento para mejorar la convergencia de los métodos multimalla algebraicos basados en la DWT.

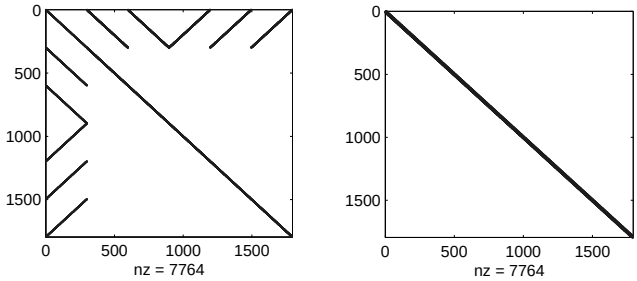


(a) A (b) B

(c) C (d) T

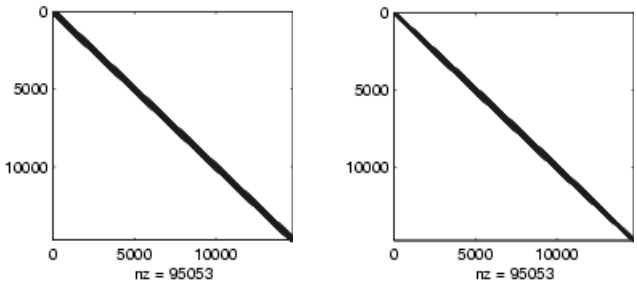
Figura 4.11: Norma de las matrices generadas al aplicar la DWT no estándar tipo "daubechies" de orden $\{1, 2, 3, 4, 5, 6\}$ de la matriz `epb1` en su orden natural (columnas en blanco) contra la matriz reordenada usado el algoritmo de *Cuthill-McKee* en reverso(columnas punteadas)

.



(a) `epb0` (b) `epb0` Permutada

Figura 4.12: Matriz `epb0` y `epb0` permutada con la rutina `symrcm` de MATLAB



(a) `epb1` (b) `epb1` Permutada

Figura 4.13: Matriz `epb1` y `epb1` permutada con la rutina `symrcm` de MATLAB

Capítulo 5

Aplicaciones de la DWT en la solución de sistemas de ecuaciones lineales

Hay varias líneas de investigación, orientadas hacia la solución eficiente de sistemas de ecuaciones lineales usando wavelets [12, 17, 19]. Muchas de estas investigaciones toman como base la descomposición (2.37), por ejemplo, el algoritmo propuesto por Chan y Chen en [16]. Este algoritmo usa la descomposición en bloques (2.37) producida por la DWT, combinando el método del complemento de *Schur* y métodos iterativos (GMRES, *Richardson*)[30] para construir un precondicionador multinivel. En la sección 5.1 se resume este método y se propone una variante de paralelización.

Otra linea de investigación relacionada con la descomposición multi-nivel tiene que ver con el "Álgebra No Estándar" definida en [56], la sección 5.2 describe en que consiste y se propone un método que utiliza la descomposición LU no estándar como precondicionador para el método BICGSTAB.

5.1. Precondicionador *Wavelet Schur*

5.1.1. Algoritmo secuencial

Un sistema de ecuaciones lineales $Ax = b$ se puede transformar con un nivel de la DWT, para obtener $\widetilde{T_0}\widetilde{x} = \widetilde{b_0}$, donde $\widetilde{T_0} = W_n A W_n^T$, $\widetilde{x} = W_n^T x$ y $\widetilde{b_0} = W_n b$. $\widetilde{T_0}$ tiene una estructura como la que se plantea en (2.37). Una restricción de la DWT para los sistemas lineales es que su dimensión debe ser $n = q \cdot 2^k$ para algún entero positivo k. Tomaremos esta suposición de ahora en adelante.

Queremos resolver este sistema transformado con algún método iterativo. Como es usual, necesitaremos algún precondicionador para obtener una velocidad de solución aceptable.

En el algoritmo propuesto por Chan y Chen en [16] se construye un precondicionador sustituyendo A_1, B_1 y C_1 en (2.37) por aproximaciones banda: Dado un entero positivo μ, se define una aproximación banda de una matriz A con semi banda μ como:

$$\overline{A}_{i,j} = \begin{cases} A_{i,j} & si \ \ |i-k| < \mu \\ 0, & en\ otro\ caso \end{cases} \tag{5.1}$$

Por tanto el precondicionador se define como:

$$\overline{T}_0 = \begin{pmatrix} \overline{A}_1 & \overline{B}_1 \\ \overline{C}_1 & T_1 \end{pmatrix} \tag{5.2}$$

El uso de este precondicionador con algún método iterativo (como GMRES, CG, etc) significa que en cada iteración del método iterativo se debe resolver un sistema lineal con la matriz de coeficientes $\overline{T}_0$:

$$\begin{pmatrix} \overline{A}_1 & \overline{B}_1 \\ \overline{C}_1 & T_1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \tag{5.3}$$

Dada su estructura a bloques y dado el hecho de que las matrices B_1, C_1 y especialmente A_1 son matrices banda se puede usar el complemento de *Schur*:

Algoritmo 5.1 Complemento-*Schur*

1. Resolver el sistema lineal $\overline{A}_1 z_1 = r_1$
2. Calcular $z_2 = r_2 - \overline{C}_1 z_1$
3. Resolver el sistema lineal $\left(T_1 - \overline{C}_1 \overline{A}_1^{-1} \overline{B}_1\right) y_2 = z_2$
4. Calcular $y_1 = z_1 - \overline{A}_1^{-1} \overline{B}_1 y_2$

Los sistemas lineales que aparecen en los pasos 1 y 4 se pueden resolver con un bajo coste, debido a que la matriz $\overline{A}_1$ es una matriz banda. El principal sistema a resolver es el que aparece en el paso 3. Este sistema debe ser resuelto nuevamente usando un método iterativo, usando T_1 como precondicionador, y así podemos aplicar exactamente la misma técnica de resolver el sistema ahora con T_1 como matriz de coeficientes: aplicar un nivel de descomposición wavelet a T_1 (para obtener A_2, B_2, C_2 y T_2), aproximar A_2, B_2, C_2 como matrices banda y aplicar nuevamente el complemento de *Schur*, para el nuevo sistema descompuesto; en el paso 3, debe aparecer un nuevo sistema $\left(T_2 - \overline{C}_2 \overline{A}_2^{-1} \overline{B}_2\right) y_3 = z_3$, este se resuelve iterativamente usando T_2 como precondicionador y así sucesivamente.

En cada nuevo nivel, las dimensiones de la matriz se dividen por 2. Esto significa que, si la dimensión del sistema original es $n = q \cdot 2^k$ dicho procedimiento se puede repetir como máximo k veces (o colapsa en el nivel k).

Sea porque se ha llegado al último nivel posible o no sea eficiente descender más niveles, tendremos en algún nivel "más grueso" en el cual la solución del sistema se debe encontrar con un método directo.

Una vez que se haya encontrado el nivel "más grueso" debemos proceder a subir hasta el nivel más fino. El procedimiento completo forma un algoritmo multi-nivel, similar a los métodos multimalla que se verán en el capítulo 6.

Los detalles se pueden encontrar en [16, 20].

5.1.2. Implementación paralela del algoritmo *Wavelet-Schur*

El precondicionador *wavelet-schur*, descrito en 5.1, es un buen ejemplo de algoritmo que se puede beneficiar del uso de la librería ScaLAPACK. Necesita calcular varios productos matriz-vector y, sobre todo, necesita que se resuelva en el nivel más bajo un sistema lineal denso.

Como aplicación de la distribución de datos ScaLAPACK para la DWT introducida en el capítulo 3 hemos desarrollado un algoritmo paralelo iterativo basado en GMRES y precondicionado con el precondicionador *wavelet-schur*.

El método GMRES para matrices densas se implementa fácilmente sobre una distribución 2DBC usando funciones de PBLAS y ScaLAPACK para los productos matriz-vector, las actualizaciones escalar-vector, producto escalar, norma euclidiana: pdgemv, pdaxpy, pddot y pdnorm2.

La implementación del precondicionador *wavelet-schur*, tal y como se describe en [16, 20], tiene muchas variantes; en nuestro caso hemos decidido usar dos iteraciones del GMRES como método iterativo aproximado para el sistema que aparece en el paso 3 del algoritmo 5.1, para el cálculo del complemento de *Schur*. En el nivel más grueso se usa un método directo para el cálculo de la descomposición LU utilizando la rutina PDGESV de ScaLAPACK.

La aproximación banda que se escogió para las matrices A, B y C fue la diagonal.

Finalmente, usamos la wavelet de *Haar* para la descomposición de los sistemas. Esto significa que no se utilizarán comunicaciones en el calculo de la DWT (debido a que la dimensión de los bloques es múltiplo de 2).

Para evaluar el algoritmo paralelo propuesto se usa una matriz generada de la siguiente forma:

$$a_{i,j} = \begin{cases} 2, & \text{si } i = j \\ \dfrac{1}{|i-j|} & \text{En otro caso} \end{cases} \tag{5.4}$$

Los tamaños de la matriz serán $768, 1152, \ldots, 4024$. Estos tamaños aseguran que las dimensiones de los sub-bloques locales sean múltiplos de 2, y por tanto, no hayan comunicaciones en el calculo de la DWT. El tamaño de bloque escogido para la distribución 2DBC fue de 32.

Todas las pruebas se llevaron a cabo en un cluster compuesto por 20 biprocesadores *Intel Xeon*, cada uno con 1 Gbyte de memoria RAM, dispuestos en una malla 4×5 de topología toro, e inter-conectados a través de una red SCI. Las mallas de procesos que usamos fueron de 2×2, 3×3 y 4×4 con tamaño de bloque 32.

5.1.3. Resultados numéricos

Las tablas 5.1 y 5.2 muestran los tiempos de CPU de las versiones secuencial y paralelas del algoritmo *wavelet-schur*, usando 1 y 2 niveles en las mallas propuestas y la distribución 2DBC de ScaLAPACK.

N	1 Proc	2×2 Procs	3×3 Procs	4×4 Procs
768	0.15	0.13	0.08	0.05
1152	0.36	0.29	0.14	0.1
1536	0.68	0.4	0.2	0.12
1920	1.14	0.62	0.29	0.18
2304	1.76	0.8	0.42	0.25
2688	2.45	1.07	0.52	0.32
3072	3.41	1.4	0.69	0.43
3456	4.76	1.8	0.93	0.56
3840	6.19	2.29	1.17	0.72
4224	7.93	2.76	1.42	0.85

Tabla 5.1: Tiempo de CPU (seg.) para el algoritmo con 1 nivel wavelet, distribución ScaLAPACK

N	1 Proc	2×2 Procs	3×3 Procs	4×4 Procs
768	0.13	0.12	0.07	0.05
1152	0.26	0.23	0.12	0.08
1536	0.47	0.3	0.14	0.09
1920	0.76	0.47	0.21	0.13
2304	1.13	0.64	0.29	0.17
2688	1.65	0.78	0.39	0.23
3072	1.96	0.98	0.43	0.27
3456	2.8	1.17	0.61	0.35
3840	3.37	1.37	0.66	0.4
4224	4.22	1.63	0.79	0.46

Tabla 5.2: Tiempo de CPU (seg.) para el algoritmo con 2 niveles wavelet, distribución ScaLAPACK

El mismo algoritmo se implementó calculando la DWT en paralelo usando la distribución Eficiente en Comunicaciones (EC). Hubiera sido interesante aplicar directamente el algoritmo para la solución del sistema lineal usando directamente la distribución EC pero las rutinas de ScaLAPACK para la descomposición LU exigen que la matriz esté distribuida usando bloques cuadrados, por eso no se puede usar una

distribución por bloques de columnas similar a la usada por la DWT Eficiente en Comunicaciones. Por tanto la matriz debe redistribuirse a la distribución 2DBC después de calculada la DWT (La redistribución se hace usando la rutina de ScaLAPACK pdgemr2d).

Las diferencias entre los tiempos de ejecución pueden atribuirse completamente a la redistribución de la matriz de la distribución EC a la distribución 2DBC. Los resultados en estos casos se pueden observar en las tablas 5.3 y 5.4.

N	1 Proc	2×2 Procs	3×3 Procs	4×4 Procs
768	0.15	0.22	0.25	0.26
1152	0.36	0.49	0.38	0.4
1536	0.68	0.7	0.58	0.52
1920	1.14	1.08	0.87	0.77
2304	1.76	1.5	1.13	0.98
2688	2.45	2.03	1.47	1.33
3072	3.41	2.62	1.92	1.66
3456	4.76	3.3	2.44	2.13
3840	6.19	4.14	3.07	2.64
4224	7.93	4.98	3.66	3.15

Tabla 5.3: Tiempo de CPU (seg.) para el algoritmo con 1 nivel wavelet, distribución Eficiente en Comunicaciones

N	1 Proc	2×2 Procs	3×3 Procs	4×4 Procs
768	0.13	0.14	0.18	0.27
1152	0.26	0.27	0.25	0.34
1536	0.47	0.38	0.31	0.41
1920	0.76	0.59	0.42	0.52
2304	1.13	0.81	0.56	0.53
2688	1.65	1.04	0.7	0.6
3072	1.96	1.31	0.85	0.69
3456	2.8	1.54	1.07	0.86
3840	3.37	1.83	1.2	0.99
4224	4.22	2.21	1.43	1.15

Tabla 5.4: Tiempo de CPU (seg.) para el algoritmo con 2 niveles wavelet, distribución Eficiente en Comunicaciones

Es bastante claro (y obvio) que los tiempos alcanzados usando directamente la distribución de datos ScaLAPACK son significativamente mejores que los tiempos del algoritmo secuencial y del que se obtiene usando la distribución EC. Si conside-

ramos que el cálculo de la DWT es más rápido usando la distribución EC que con la distribución 2DBC, significa que el tiempo necesario para redistribuir la matriz (desde la distribución EC a la distribución 2DBC) es bastante grande. Las eficiencias alcanzadas no son muy buenas en ninguno de los dos casos, debido al alto costo en comunicaciones de que requieren los productos matriz vector y la descomposición LU. Sin embargo, la eficiencia mejora cuando el número de procesadores que se usan y el tamaño del problema se incrementan; en el caso de la distribución 2DBC, la mejora es más notable.

Los resultados de este epígrafe fueron descritos en el artículo "Transformada wavelet paralela en la resolución de sistemas lineales densos" publicado en la "Revista Cubana de Ciencias Informáticas", referencia [6].

5.2. Precondicionado de sistemas densos usando la DWT no estándar

En esta sección definiremos formalmente la representación de un operador lineal dada en forma no estándar como aparece en [56]. Seguidamente introduciremos las operaciones básicas del álgebra no estándar dadas en [56] y algunas operaciones necesarias para definir un método de precondicionado basado en la descomposición LU no estándar y el método iterativo BICGSTAB.

5.2.1. La forma no estándar de un operador lineal

La forma no estándar de un operador consiste en una representación multinivel de una matriz cuando se le aplica la DWT-2D no estándar que se introdujo previamente en la sección 2.4.5.

Definición 5.2.1 (Forma no estándar de una matriz)**.** *Sea T_0 una matriz $n \times n$. Se definen las matrices P_i y Q_i de la misma forma que en (2.38) y (2.39) respectivamente, para $i = 1, 2, ...l$, donde l es la cantidad de niveles wavelet a aplicar.*

Por ejemplo la forma no estándar de un operador T_0 para 3 niveles sería:

<table>
<tr><td rowspan="2">A_1</td><td colspan="3" rowspan="2">B_1</td></tr>
<tr></tr>
<tr><td rowspan="3">C_1</td><td>A_2</td><td colspan="2">B_2</td></tr>
<tr><td rowspan="2">C_2</td><td>A_3</td><td>B_3</td></tr>
<tr><td>C_3</td><td>T_3</td></tr>
</table>

donde:

$$T_i = P_i T_{i-1} P_i^T,\ A_i = Q_i T_{i-1} Q_i^T,\ B_i = Q_i T_{i-1} P_i^T,\ C_i = P_i T_{i-1} Q_i^T \tag{5.5}$$

Es importante destacar que una matriz representada en forma no estándar no es como una matriz ordinaria en el sentido en que no se aplican de la misma forma las operaciones del álgebra matricial. Por lo tanto es necesario definir esas operaciones.

Definición 5.2.2 (Forma no estándar extendida de un vector)**.** *La representación extendida v_e de un vector $v = s_0$, es una extensión de la transformada* wavelet *de v con el objetivo de usar la forma no estándar en las operaciones con v. La misma incluye vectores de todos los niveles de la transformada wavelet de v.*

$$\widetilde{v}_e = \begin{pmatrix} d_1 \\ s_1 \\ d_2 \\ s_2 \\ \vdots \\ d_l \\ s_l \end{pmatrix} \tag{5.6}$$

donde

$$d_i = Q_i s_{i-1},\ s_i = P_i s_i,\ i = 1, 2, ... \tag{5.7}$$

Definición 5.2.3 (Multiplicación no estándar de una matriz y un vector)**.** *La multiplicación de una matriz y un vector en forma no estándar está definida por vectores*

en forma extendida. Sea v un vector, teniendo su representación extendida $\widetilde{v}_e$, sea M la representación no estándar de una matriz T, su producto multiresolución produce un vector que es la representación no estándar extendida del vector Tv. Las siguientes ecuaciones recursivas muestran el proceder:

$$\overline{d}_1 = \overline{s}_1 = 0, \tag{5.8}$$

Para $j = 1, 2, ..., l$

$$\breve{d}_j = A_j d_j + B_j s_j, \tag{5.9}$$

$$d_j = \breve{d}_j + \overline{d}_j, \tag{5.10}$$

$$s_j = \breve{s}_j + \overline{s}_j, \tag{5.11}$$

$$\overline{d}_j = Q_j \left(\breve{d}_{j-1} + \overline{d}_{j-1} \right) \tag{5.12}$$

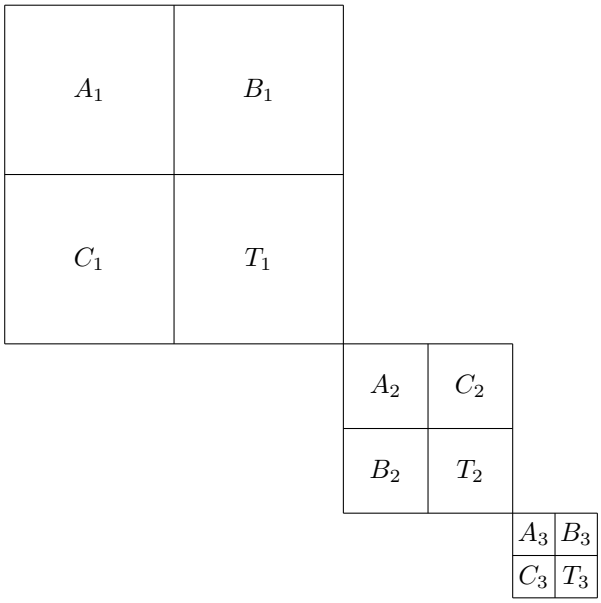
$$\overline{s}_j = Q_j \left(\breve{s}_{j-1} + \overline{s}_{j-1} \right) \tag{5.13}$$

En el último nivel

$$s_n = \overline{s}_n + \breve{s}_j = C_n d_n + T_n s_n + \overline{s}_n \tag{5.14}$$

Para definir la multiplicación de dos matrices es necesario definir primero la forma no estándar extendida de un matriz.

Definición 5.2.4 (Forma no estándar extendida de una matriz)**.** *La forma no estándar extendida de una matriz se define igual que la estándar pero teniendo en cuenta también los T_j. La forma no estándar de un operador T_0 para* 3 *niveles sería:*



Ahora podemos definir el producto matricial de dos matrices en forma no estándar extendida.

Definición 5.2.5 (Producto matricial no estándar)**.** *Sean $\hat{T}_0$ y $\breve{T}_0$ dos matrices de $n \times n$, con forma no estándar $\hat{T}$ y $\breve{T}$. Entonces la forma no estándar de T_0 viene dada por:*

$$T = \hat{T} \bullet \breve{T} \tag{5.15}$$

Donde $\bullet$ es el operador de multiplicación en el álgebra no estándar. El operador $\bullet$ se define en términos de submatrices de $\hat{T}$ y $\breve{T}$:

$$\overline{A}_1 = \overline{B}_1 = \overline{C}_1 = \overline{T}_1 = 0, \tag{5.16}$$

Para $j = 1, 2.., l$

$$\breve{A}_j = \hat{A}_j \tilde{A}_j + \hat{B}_j \tilde{C}_j, \tag{5.17}$$

$$\breve{B}_j = \hat{A}_j \tilde{B}_j + \hat{B}_j \tilde{T}_j, \tag{5.18}$$

$$\breve{B}_j = \hat{C}_j \tilde{A}_j + \hat{T}_j \tilde{C}_j, \tag{5.19}$$

$$\breve{T}_j = \hat{C}_j \tilde{B}_j, \tag{5.20}$$

$$A_j = \breve{A}_j + \overline{A}_j, \tag{5.21}$$

$$B_j = \breve{B}_j + \overline{B}_j, \tag{5.22}$$

$$C_j = \breve{C}_j + \overline{C}_j, \tag{5.23}$$

$$T_j = \breve{T}_j + \overline{T}_j, \tag{5.24}$$

$$\overline{A}_j = Q_j \left(\breve{T}_{j-1} + \overline{T}_{j-1} \right) Q_j^T, \tag{5.25}$$

$$\overline{B}_j = Q_j \left(\breve{T}_{j-1} + \overline{T}_{j-1} \right) P_j^T, \tag{5.26}$$

$$\overline{C}_j = P_j \left(\breve{T}_{j-1} + \overline{T}_{j-1} \right) Q_j^T, \tag{5.27}$$

$$\overline{T}_j = P_j \left(\breve{T}_{j-1} + \overline{T}_{j-1} \right) P_j^T, \tag{5.28}$$

En el último nivel

$$T_l = \breve{T}_l + \overline{T}_l = \hat{C}_l \tilde{B}_l + \hat{T}_l \tilde{T}_l + \overline{T}_l \tag{5.29}$$

5.2.2. Descomposición LU en forma no estándar

Podemos factorizar la forma no estándar T, de una matriz T_0 en dos matrices $\hat{T}$ y $\tilde{T}$ tal que se satisfaga:

$$T = \hat{T} \bullet \tilde{T} \tag{5.30}$$

Donde $\hat{T}$ es una matriz en forma no estándar *triangular inferior* y $\tilde{T}$ es una matriz en forma no estándar *triangular superior*. Esto significa que para $j = 1, 2, ...l$ que $\hat{A}_j$ es *triangular inferior* y $\hat{B}_j$ es cero, $\tilde{A}_j$ es *triangular superior* y $\tilde{B}_j$ es cero, en el nivel l, $\hat{T}_l$ es *triangular inferior* y $\tilde{T}_l$ es *triangular superior*. Las submatrices que forman

a $\hat{T}$ y a $\tilde{T}$ se pude calcular haciendo uso de la definición 5.2.5. En cada nivel, $\breve{A}_j$ se factorizan usando la descomposición LU estándar y los factores $\hat{A}_j$ y $\tilde{A}_j$ se usan para calcular $\hat{C}_j$ y $\tilde{B}_j$. En [56] se explica el procedimiento a seguir en más detalle.

5.2.3. Solución de los sistemas triangulares

Dada la factorización no estándar de una matriz T_0, $\hat{T} \bullet \tilde{T}$ la ecuación:

$$T_0 x = b \tag{5.31}$$

Se puede resolver aplicando el siguiente algoritmo:

Algoritmo 5.2 Solución de un sistema en forma no estándar

1. Calcular la DWT, $\tilde{b}$ de b.
2. Usar la sustitución multiresolución hacia adelante para calcular el vector $\tilde{y}$. Resolviendo: $\hat{T}\tilde{y} = \tilde{b}$. (Algoritmo 5.3)
3. Usar la sustitución multiresolución hacia atrás para calcular el vector $\tilde{x}$. Resolviendo: $\tilde{T}\tilde{x} = \tilde{y}$. (Algoritmo 5.4)
4. Aplicar la DWT inversa a $\tilde{x}$ para obtener x. Y con ello la solución de (5.31).

Los algoritmos de sustitución multiresolución se pueden encontrar en [56]. A continuación los resumimos en los algoritmos 5.3 y 5.4.

Algoritmo 5.3 Solución de un sistema triangular inferior en forma no estándar, con sustitución hacia adelante

1. $\overline{d}_1 = \overline{s}_1 = 0$
2. **para** $j := 0$ **to** l **hacer**
3. Calcular $\overline{d}_j$, $j \neq 1$
4. Calcular $\tilde{d}_j$
5. **fin para**
6. En el último nivel calcular $\tilde{s}_j$

Algoritmo 5.4 Solución de un sistema triangular superior en forma no estándar, con sustitución hacia atrás

1. En el último nivel calcular $\tilde{s}_j$.
2. **para** $j := l$ **to** 1 **hacer**
3. Calcular $\tilde{s}_j$, $j \neq l$
4. Calcular $\tilde{d}_j$
5. **fin para**

5.2.4. Compresión del operador

Informalmente decimos que una matriz es "suave", tiene una parte "suave", si existen un conjunto grandes de entradas consecutivas en la matriz de valor muy cercano entre sí, por ejemplo la matriz definida en (5.4). Si A es lo suficientemente "suave", su forma no estándar contendrá un conjunto grande de elementos por debajo de cierto umbral. En particular si la parte no "suave" de A se concentra en una banda de la diagonal principal, entonces las submatrices que conforman la forma no estándar tendrán también una estructura diagonal. Si la matriz A tiene estas características entonces el algoritmo 5.2 se puede usar para calcular la solución de (5.31) de manera muy eficiente, ver [56]. Si embargo esto depende fuertemente de la estructura que tenga la matriz A.

Si la matriz A no tiene una estructura banda entonces se puede aplicar un umbral para hacer dispersa la forma no estándar. Este umbral se puede elegir como:

$$Umbral = \varepsilon \|A\| \tag{5.32}$$

Se descartan todos los elementos que están por debajo de ese umbral. El algoritmo 5.5 muestra la resolución de (5.31) cuando la matriz tiene muchas partes suaves.

Este procedimiento falla cuando la matriz A no tiene una estructura suave y hay que recurrir a un ε grande para obtener un índice de dispersidad aceptable. En muchos casos el error de la solución obtenida es malo. En el siguiente epígrafe propondremos usar la descomposición LU no estándar como precondicionador del método iterativo BICGSTAB con el objetivo de mejorar la solución inicial alcanzada con el algoritmo 5.5.

Algoritmo 5.5 Solución de un sistema en forma no estándar

1. Escoger un umbral ε.
2. Calcular la DWT, $\tilde{b}$ de b.
3. Calcular la LU no estándar y obtener $\hat{T}$ y $\tilde{T}$, descartar todos los elementos que están por debajo del umbral dado por (5.32).
4. Usar la sustitución multiresolución hacia adelante para calcular el vector $\tilde{y}$. Resolviendo: $\hat{T}\tilde{y} = \tilde{b}$. (Algoritmo 5.3)
5. Usar la sustitución multiresolución hacia atrás para calcular el vector $\tilde{x}$. Resolviendo: $\tilde{T}\tilde{x} = \tilde{y}$. (Algoritmo 5.4)
6. Aplicar la DWT inversa a $\tilde{x}$ para obtener x. Y con ello la solución de (5.31).

5.2.5. Precondicionador usando la LU no estándar

La forma no estándar es una forma poco costosa de resolver (5.31) pero su precisión depende de la estructura inicial que tenga la matriz del sistema. Cuando esa estructura no es conocida en algunas ocasiones debemos recurrir a umbrales demasiado finos para obtener la precisión deseada, sin embargo esto disminuye la eficiencia de los algoritmos debido a que los índices de dispersidad disminuyen. Lo que sí podemos hacer es obtener una buena aproximación inicial de la solución y luego intentar mejorarla usando algún método iterativo. Además se puede aprovechar la descomposición LU no estándar ya calculada como precondicionador del método iterativo que hallamos escogido. Esto es una propuesta, pero no se ha probado suficientemente.

5.3. Resumen de aportaciones

Se estudiaron distintas variantes de construcción de precondicionadores basados en la DWT para acelerar la convergencia de métodos iterativos en la resolución de ecuaciones lineales. Se propuso una variante de paralelización que usa el algoritmo paralelo de cálculo de la DWT con replicación parcial, este resultado fue publicado en la "Revista Cubana de Ciencias Informáticas", en el 2007, referencia [6].

También se propone una alternativa para la solución de sistemas lineales densos de gran dimensión que utiliza un precondicionador basado en el álgebra no estándar.

Capítulo 6

Variantes de los métodos multimalla basados en la transformada wavelet

El algoritmo central para el cálculo de la DWT visto en la sección 2.4.5, tiene una estructura multi-nivel que recuerda los métodos multimalla para la resolución de sistemas lineales. Es natural por lo tanto, que se encuentren en la literatura varios artículos dedicados a buscar la conexión entre estos dos métodos, proponiéndose una gran variedad de métodos Wavelet-Multimalla.

Muchos de estos artículos describen algoritmos multimalla que usan la transformada wavelet como base para la discretización en la resolución de ecuaciones en derivadas parciales. Estos métodos pueden ser muy eficientes pero no son generales debido a que, para aplicarlos, se necesita un conocimiento completo de la discretización y del problema que le da origen. Por otro lado, existen algoritmos tipo multimalla, que se pueden aplicar a cualquier sistema lineal no importa de donde surja. Solo se necesita conocer la matriz y la parte derecha del sistema. Estos métodos multimalla se conocen como "algebraicos".

El presente capítulo trata los Métodos Wavelet Multimalla Algebraicos (Wavelet Algebraic Multigrid Methods, WAMG), algoritmos que combinan los métodos multimalla con las wavelet, y que no necesitan de ningún conocimiento del problema a resolver; solo la matriz de coeficientes y la parte derecha del sistema.

Existen, que conozcamos, pocos trabajos dedicados a los métodos WAMG. Desde nuestro punto de vista el algoritmo principal fue propuesto por Wang, Dutton y Hao en [34] y retomado por Pereira y otros autores en [52]; otro algoritmo importante fue el propuesto por D. Leon en [40] (discutiremos estos algoritmos en detalle más adelante en la sección 6.2). Sin embargo hay variantes muy interesantes de estos algoritmos que no se discuten en los artículos citados. En este capítulo propondremos dos variantes de los algoritmos WAMG. La primera se basa en la descomposición inducida del sistema lineal al aplicar la DWT. La segunda reduce el costo de los ciclos del algoritmo multimalla saltando operaciones en algunos niveles o mallas.

Una observación importante (la cual discutiremos en detalle al final de este capítulo) de estos métodos multimallas basados en la transformada wavelet es que son especialmente adecuados para problemas en los que se necesita resolver varios sistemas de ecuaciones con matrices desplazadas $A - hI$. Esto resulta muy relevante para la solución de Ecuaciones en Derivadas Parciales (PDEs) y sistemas de ecuaciones diferenciales ordinarias dependientes del tiempo.

6.1. Métodos multimalla

Los métodos Multimalla[1] están diseñados para resolver eficientemente sistemas de ecuaciones lineales

$$Tx = b \tag{6.1}$$

que surgen de la discretización de ecuaciones en diferencias parciales. El nombre de "multimalla" parte del supuesto de que el problema original se ha discretizado en varias mallas, cada vez más gruesas y en una forma progresiva, que se usan para construir iterativamente la solución del sistema (6.1). Para describir los métodos multimalla es suficiente con considerar solo dos mallas, la malla original que llamaremos "fina" y otra a la que llamaremos "gruesa". Una vez discretizado el problema original en estas dos mallas, las dos observaciones siguientes nos pueden dar una idea global:

- Los métodos iterativos simples como Gauss-Seidel o Jacobi (a los que llamaremos, como es usual, suavizadores o de relajamiento) son muy efectivos en eliminar los componentes de alta frecuencia del error [14], mientras que se consideran más lentos en eliminar los componentes de baja frecuencia del error.

[1] Del vocablo en Inglés multigrid

- Los componentes de baja frecuencia del error pueden convertirse en componentes de alta frecuencia (y por tanto, más fáciles de descartar) si el mismo problema puede ser interpolado a una malla más gruesa.

Esas dos observaciones nos remiten al algoritmo general 6.1. A partir de este momento el superíndice f se usará para las variables asociadas a la malla fina, y c para la malla gruesa, h para magnitudes de alta frecuencia y l para magnitudes de baja frecuencia.

Algoritmo 6.1 Algoritmo multimalla de dos mallas

1. Aplicar algunos pasos de suavizado al sistema más fino para obtener una aproximación inicial x^f.
2. Calcular el residual para esta aproximación. $r^f = b^f - T^f x^f$ (Dando como resultado la ecuación residual, $T^f e^f = r^f$).
3. Restringir el residual a la malla gruesa, $r^f \rightarrow r^c$.
4. Resolver la ecuación residual en la malla gruesa. $T^c e^c = r^c$.
5. Proyectar la solución en el nivel grueso de vuelta al nivel más fino: $r^c \rightarrow r^f$.
6. Actualizar la solución en el nivel más fino, $x^f = x^f + e^f$.
7. Aplicar varias iteraciones de relajación al sistema en el nivel fino, usando como aproximación inicial x^f.

Para muchos problemas este algoritmo simple reduce notablemente el error con un costo relativamente pequeño. Así mismo es la base de todos los métodos multimalla; muchos de ellos son solo versiones recursivas de este, aplicando el método en dos mallas para resolver la ecuación gruesa en la linea (4). Por lo tanto, en este capítulo, para todo algoritmo que se describa o proponga se usará el modelo de algoritmo en dos mallas.

Para definir de manera correcta este algoritmo es necesario especificar claramente la operación de restricción en el paso (3) y de proyección del paso (5), así como la manera de construir la matriz gruesa T^c. Tal y como hemos mencionado, en los métodos multimalla generales, esas operaciones se efectúan rediscretizando el problema sobre una malla más gruesa. Sin embargo esto restringe su aplicabilidad a problemas que surgen de discretizaciones.

Los métodos multimalla algebraicos son algoritmos que tratan de simular la idea clásica de los métodos multimalla, pero sin necesidad de tener conocimiento del problema que le dio origen; los operadores de restricción y proyección junto a la matriz gruesa se definen usando solamente las entradas de la matriz original de coeficientes.

6.2. Algoritmos wavelet multimalla algebraicos: Estado del arte

La mayoría de los algoritmos wavelet multimalla comienzan aplicando algunos pasos de relajamiento al sistema original, obteniendo así una aproximación inicial de la solución x_0. A partir de esta aproximación inicial se calcula el residuo y se obtiene la ecuación residual:

$$Te = r, \qquad donde \qquad r = b - Tx_0 \tag{6.2}$$

Entonces, se aplica un nivel wavelet con lo que se obtiene una descomposición análoga a (2.37):

$$Te = r, \quad \Rightarrow \left(W_n T W_n^T\right)(W_n e) = (W_n r) \quad \Rightarrow \begin{pmatrix} A_1 & B_1 \\ C_1 & T_1 \end{pmatrix} \begin{pmatrix} e_h^c \\ e_l^c \end{pmatrix} = \begin{pmatrix} r_h^c \\ r_l^c \end{pmatrix} \tag{6.3}$$

A partir de esta ecuación hay muchas maneras de obtener operadores de restricción y de relajamiento así como matrices gruesas. Uno de los primeros algoritmos en este campo fue propuesto por Doreen de Leon en [40]. Aunque el mismo fue descrito usando problemas elípticos unidimensionales y bidimensionales, el algoritmo sólo necesita las entradas de la matriz para ser aplicado, por lo tanto es un algoritmo "multimalla algebraico".

El algoritmo propuesto en [40] (que asume que la matriz es simétrica) surge de aplicar una descomposición LDU por bloques al sistema (6.3). Luego de algunas transformaciones, se escoge el operador de malla gruesa (Para el caso simétrico) como:

$$T_1 - B_1 A_1^{-1} B_1 \tag{6.4}$$

el cual es el complemento de *Schur* de A_1 en T. Los resultados mostrados son buenos aunque presentados en términos de v-ciclos; sin embargo el mayor problema para la aplicación de esta técnica es el cálculo de la inversa, ya sea exacto o aproximado, de A_1 para cada nivel. Para muchos problemas A_1^{-1} no será dispersa por lo que el costo de calcular la inversa puede ser muy alto, comparable a resolver el sistema original con un método directo.

Un algoritmo más general lo presentaron Wang, Dutton y Hao en [34]. El algoritmo propuesto en este trabajo es, al parecer, el principal "algoritmo wavelet multimalla

algebraico"(WAMG). El mismo se obtiene seleccionando T_1 como la matriz gruesa, la matriz de filtros de baja frecuencia G como operador de restricción y su traspuesta G^T como operador de prolongación. Esto es equivalente a tomar la parte de alta frecuencia de la solución del sistema (6.3), e_h^c, como un vector nulo.

El algoritmo en dos mallas propuesto en [34] se presenta en 6.2.

Algoritmo 6.2 Algoritmo en dos mallas, WAMG

1. Aplicar algunos pasos de suavizado al sistema más fino para obtener una aproximación inicial x^f.
2. Calcular el residual para esta aproximación. $r^f = b^f - T^f x^f$ (Dando como resultado la ecuación residual, $T^f e^f = r^f$).
3. Aplicar la trasformada wavelet a la ecuación residual dando como resultado el sistema (6.3).
4. Resolver la ecuación residual "gruesa": $T_1 e_l^c = r_l^c$.
5. Aplicar la DWT inversa a la solución "gruesa" e_l^c hacia el nivel más fino:

$$e^f = W_n^T \begin{pmatrix} 0 \\ e_l^c \end{pmatrix} = G_n^T e_l^c. \tag{6.5}$$

6. Actualizar la solución en el nivel más fino, $x^f = x^f + e^f$.
7. Aplicar varias iteraciones de relajación al sistema en el nivel fino, usando como aproximación inicial x^f.

Lamentablemente, los resultados experimentales alcanzados en este artículo estaban relacionados con un problema integral electromagnético, discretizado con una versión especial del Método de Fronteras, el cual no es suficientemente general ni fácil de reproducir.

El mismo algoritmo, con muchos más detalles, lo presentó recientemente Pereira y otros en [52]. Estos autores probaron experimentalmente que el algoritmo WAMG es un precondicionador muy útil. También mostraron que el cálculo de la DWT en L niveles se puede restringir al cálculo de la secuencia de matrices de baja frecuencia T_i. Esto significa que el tiempo de puesta a punto se puede reducir notablemente, ya que no es necesario calcular la transformada wavelet completa en cada nivel. Finalmente se muestra como adaptar este método cuando los coeficientes de la matriz T (o la dimensión de cualquiera de la matrices calculadas T_i) no es par. También se discuten algunas implementaciones interesantes.

Estos artículos han abierto el campo. En los próximos epígrafes, propondremos dos algoritmos similares, que comparten las mismas ideas, pero con algunas diferencias

de implementación.

6.3. Variaciones del algoritmo WAMG

6.3.1. Implementación eficiente de la DWT-2D de Haar en MATLAB

Para todas las implementaciones de algoritmos wavelet multimalla de este capítulo se ha utilizado el wavelet de Haar ($G_0 = \frac{\sqrt{2}}{2}$, $G_1 = \frac{\sqrt{2}}{2}$ y $H_0 = -\frac{\sqrt{2}}{2}$, $H_1 = \frac{\sqrt{2}}{2}$). Para las implementaciones en MATLAB se ha usado el algoritmo 6.3 que ha probado ser muy eficiente para matrices dispersas.

Algoritmo 6.3 DWT-2D de Haar en MATLAB.
Entrada: Matriz A.
Salida: Submatrices $T1$, $B1$, $C1$ y $T1$ como en la ecuación (2.37).

```
(1) fact=sqrt(2)/2;
(2) aux1=A(1:2:n,:)*fact;aux2=A(2:2:n,:)*fact;
(3) wa1=aux1+aux2; wa2=aux1-aux2;
(4) T1=wa1(:,1:2:n)*fact+wa1(:,2:2:n)*fact;
(5) B1=wa1(:,1:2:n)*fact-wa1(:,2:2:n)*fact;
(6) C1=wa2(:,1:2:n)*fact+wa2(:,2:2:n)*fact;
(7) A1=wa2(:,1:2:n)*fact-wa2(:,2:2:n)*fact;
```

6.3.2. Algoritmo WPAMG

Consideremos nuevamente el algoritmo 6.2, después del paso (2). Recordemos que en el paso (1) se han llevado a cabo algunas iteraciones de relajamiento. De las propiedades de los suavizadores estándares (Jacobi, Gauss-Seidel) el error residual (aún por calcular) mantiene la mayoría de sus componentes de baja frecuencia, mientras descarta la mayoría de los componentes de alta frecuencia. Aplicando un nivel wavelet, el sistema adquiere la estructura de (6.3), y los vectores e y r se dividen en sus componentes de alta y baja frecuencia:

$$We = \begin{pmatrix} e_h^c \\ e_l^c \end{pmatrix}, \quad Wr = \begin{pmatrix} r_h^c \\ r_l^c \end{pmatrix} \tag{6.6}$$

Si prestamos atención solo a la parte baja del sistema (6.3), obtenemos:

$$C_1 e_h^c + T_1 e_l^c = r_l^c. \tag{6.7}$$

Como sabemos, el error e (aún por calcular) se divide por la descomposición (6.3) en componentes de alta frecuencia e_h^c y componentes de baja frecuencia e_l^c. Como el suavizador elimina los componentes de alta frecuencia, se puede esperar (y usualmente se confirma en los experimentos) que la norma de e_h^c es pequeña o, al menos, menor que la norma de e_l^c.

Por este motivo, tiene sentido tomar

$$T_1 e_l^c = r_l^c \tag{6.8}$$

como ecuación de malla gruesa, lo que se confirma con los resultados experimentales mostrados en [34, 52], ambos usando el algoritmo en si mismo o como precondicionador. Sin embargo, el resultado depende fuertemente de que la norma del vector e_h^c sea realmente pequeña. Cuando esto no se cumple, es posible la siguiente variante para mejorar el método, usando la parte superior del sistema trasformado wavelet:

$$A_1 e_h^c + B_1 e_l^c = r_h^c. \tag{6.9}$$

Podemos usar este subsistema para calcular aproximadamente los componentes de alta frecuencia del error. Para hacer esto, usamos en primer lugar la aproximación de baja frecuencia (calculada resolviendo (6.8)) e_l^c, para calcular una nueva parte derecha, $\hat{r}^c$, para la ecuación (6.9).

$$A_1 e_h^c = r_h^c - B_1 e_l^c = \hat{r}^c. \tag{6.10}$$

y entonces resolver (al menos aproximadamente) el sistema $A_1 e_h^c = \hat{r}^c$; al que podemos llamar "sistema de alta frecuencia", mientras que el sistema (6.8) podemos nombrarlo "sistema de baja frecuencia".

Cuando ya se ha calculado la aproximación de alta frecuencia del error, se puede aplicar la transformada wavelet inversa para obtener el error correcto en el nivel más fino.

$$e^f = W_n^t \begin{pmatrix} e_h^c \\ e_l^c \end{pmatrix} \tag{6.11}$$

El nuevo subsistema de alta frecuencia se puede resolver usando la idea recursiva de los métodos multimalla, esto es, aplicar un nuevo algoritmo en dos mallas al sistema (6.10). Sin embargo esto ha resultado ser muy costoso (especialmente la fase de inicialización, explicada anteriormente) y en muchos casos el error de alta frecuencia es pequeño, de forma que no hay necesidad real de un calculo tan preciso. En lugar de ellos se pueden aplicar uno o dos pasos de suavizado a este sistema para obtener un valor razonable de e_h^c a un costo pequeño.

Examinamos también la posibilidad de conocer la norma del nuevo residual $\hat{r}^c$ y aplicar los pasos de suavizado sobre el sistema de alta frecuencia solo si dicha norma es relativamente grande en comparación con la norma del residuo original r^c. Sin embargo en nuestros experimentos no encontramos ninguna ventaja en hacer esto.

El algoritmo propuesto opera simultáneamente sobre la parte de baja y alta frecuencia lo que es característico de la descomposición *Wavelet Packet*, por eso nombramos al algoritmo como algoritmo Wavelet Packet Multimalla Algebraico (WPAMG), ver Algoritmo 6.4.

Algoritmo 6.4 Algoritmo en dos mallas, WPAMG

1. Aplicar algunos pasos de suavizado al sistema más fino para obtener una aproximación inicial x^f.
2. Calcular el residual para esta aproximación. $r^f = b^f - T^f x^f$ (Dando como resultado la ecuación residual, $T^f e^f = r^f$).
3. Aplicar la trasformada wavelet a la ecuación residual dando como resultado el sistema (6.3).
4. Resolver la ecuación residual "gruesa": $T_1 e_l^c = r_l^c$.
5. Calcular el "nuevo residual de alta frecuencia" $\hat{r}^c$ como $r_h^c - B_1 e_l^c$.
6. Aplicar algunos pasos de suavizado en (6.9) para obtener e_h^c.
7. Aplicar la DWT inversa a la solución "gruesa" e_l^c hacia el nivel más fino:

$$e^f = W_n^T \begin{pmatrix} e_h^c \\ e_l^c \end{pmatrix} = G_n^T e_l^c. \tag{6.12}$$

8. Actualizar la solución en el nivel más fino, $x^f = x^f + e^f$.
9. Aplicar varias iteraciones de relajación al sistema en el nivel fino, usando como aproximación inicial x^f.

Es posible diseñar iteraciones más complicadas. Por ejemplo, una vez que se calcule una aproximación para el error de alta frecuencia e_h^c, es posible retomar la parte baja del sistema $C_1 e_h^c + T_1 e_l^c = r_l^c$, calcular y actualizar la parte derecha: $(\hat{r})_u = r_l^c - e_h^c$ y hacer iteraciones de relajamiento en el nuevo sistema $T_1 e_l^c = (\hat{r})_u$ usando para ello la

aproximación inicial e_l^c obtenida anteriormente. Sin embargo, en ninguno de nuestros experimentos, esos pasos de suavizado mejoran suficientemente la exactitud de forma que justifiquen ese gasto extra.

Tal y como se mencionó anteriormente, la inicialización de WAMG usando L niveles consiste en calcular las matrices de baja frecuencia $T_1, T_2, \ldots, T_L$. Si analizamos el algoritmo WPAMG nos percatamos que para los dos niveles que proponemos necesitamos calcular las matrices T_1, A_1, B_1. Por lo mismo el paso de inicialización de WPAMG se puede hacer usando el algoritmo 6.3 pero sin el paso de calcular las matrices C.

6.3.3. Algoritmo WAMG2

El algoritmo WPAMG trata de obtener mayor precisión en un ciclo completo, con algún trabajo extra. Una variante complementaria trataría de disminuir el costo de un ciclo completo a costa de tener menor precisión.

En el algoritmo original WAMG, el relajamiento se hace en cada nivel tal y como se detalla en la sección 6.2. Sin embargo es posible limitar el número de niveles en los cuales se llevan a cabo pasos de relajamiento, debido a que todas las matrices T_i de los L niveles de descomposición pueden ser consideradas como aproximaciones gruesas de la matriz original T.

Tomando el sistema original en el nivel 0, proponemos aplicar el suavizado en los "niveles pares", saltando los "niveles impares". Por supuesto hay más posibilidades pero esta parece ser la más sencilla. Llamamos a este algoritmo WAMG2 y su definición se muestra en 6.5. El algoritmo se puede adaptar fácilmente si la cantidad de niveles es impar.

6.4. Resultados experimentales

Aunque los métodos descritos en los epígrafes anteriores pueden ser usados por si solos como técnicas de resolución, los mejores rendimientos se han obtenido utilizándolos como precondicionadores de métodos iterativos estándares como el GMRES [28, 30] (Como ya se ha mencionado en [52]). Para evaluar los métodos, los hemos comparados con los mejores métodos conocidos (y probablemente los más usados), la

familia de precondicionadores ILU.

Algoritmo 6.5 Algoritmo en dos mallas, WAMG2

1. Aplicar algunos pasos de suavizado al sistema más fino para obtener una aproximación inicial x^f.
2. Calcular el residual para esta aproximación. $r^f = b - T^f x^f$ (Dando como resultado la ecuación residual, $T^f e^f = r^f$).
3. Aplicar la trasformada wavelet a la ecuación residual dando como resultado el sistema (6.3).
4. Aplicar la transformada wavelet a la ecuación residual "gruesa": $T_1 e_l^c = r_l^c$; extraer la nueva ecuación residual $T_2 e_l^{c_2} = r_l^{c_2}$.
5. Resolver la segunda ecuación residual "gruesa": $T_2 e_l^{c_2} = r_l^{c_2}$.
6. Aplicar la DWT inversa a la solución "gruesa" $e_l^{c_2}$ dos veces hasta el nivel más fino.
$$e_l^c = W_{\frac{n}{2}}^T \begin{pmatrix} 0 \\ e_l^{c_2} \end{pmatrix} = G_{\frac{n}{2}}^T; \quad e^f = W_n^T \begin{pmatrix} 0 \\ e_l^c \end{pmatrix} = G_n^T e_l^c \tag{6.13}$$
7. Actualizar la solución en el nivel más fino $x^f = x^f + e^f$.
8. Aplicar algunos pasos de suavizado al sistema fino, usando como aproximación inicial x^f.

En general hemos comparado el algoritmo WAMG propuesto en [34, 52], los algoritmos WPAMG y WAMG2 propuestos previamente, el precondicionador ILU-0 sin rellenado [58], y el precondicionador ILU con umbral, descrito en también en [58]; usamos tres umbrales diferentes, 0.1, 0.01 y 0.001.

Todos los métodos se usan como precondicionadores de GMRES usando como parámetro de reinicio 10 (la máxima dimensión del subespacio de Krylov). Todas las implementaciones y experimentos se han llevado a cabo con MATLAB [4]. Las implementaciones usadas de GMRES, los precondicionadores ILU-0 e ILU son las versiones estándares de los paquetes en MATLAB. El vector de la parte derecha se generó de forma tal que la solución de los sistemas fuera un vector de unos y la solución inicial se tomó como un vector generado de forma aleatoria.

Los experimentos con los algoritmos WAMG, WAMG2 y WPAMG se hicieron usando las wavelet de Haar. El número de niveles de estos algoritmos se tomó de forma tal que la dimensión del nivel más grueso fuera menor que 16 usando la expresión $levels = ceil(\log_2 \frac{n}{15})$, donde n es la dimensión de la matriz. Para resolver el sistema de ecuaciones en el nivel más grueso se usó la descomposición LU de MATLAB. Las iteraciones de relajamiento se hicieron haciendo uso del método iterativo Gauss-Seidel.

En todos los casos el algoritmo se detiene cuando la norma del residuo se reduce

en un factor $1e-6$ con respecto al vector de la parte derecha.

Las matrices seleccionadas para los experimentos vienen de distintas fuentes. En primer lugar usamos SPARSKIT para generar tres de las matrices usadas en los experimentos llevados a cabo en [52]; estas son las matrices de diferencias finitas $F2da$, $F2db$, y $F3d$. La matriz $F2da$ es un caso sencillo de dos dimensiones, $F2db$ es un caso difícil (coeficientes discontinuos) también en dos dimensiones y $F3d$ es un caso relativamente fácil en tres dimensiones.

Seguidamente se escogieron varias matrices desde el repositorio "Matrix Market" [1]: $Orsirr1$ (Usada también en [52]), $Orsirr2$, $Sherman1$, $Thermal$, $Nos3$, $Nos7$. Se seleccionaron además dos matrices de gran dimensión desde una colección de matrices de la Universidad de Florida [28]: $Epb1$ y $Epb2$. Los resultados para la última de ellas se obtuvieron reordenándola previamente con la función *colamd* de MATLAB. Un resumen se puede ver en la Tabla 6.1.

La Tabla 6.2 muestra la cantidad de iteraciones del método GMRES con los diferentes precondicionadores. En algunos casos, marcados como N, las iteraciones no convergen usando el máximo número de iteraciones (1000), mientras que en otros casos, marcados con X, los precondicionadores usados fueron "Singulares".

El desempeño de los precondicionadores ILU es característico; el precondicionador ILU-0 tiene un desempeño relativamente pobre, mientras que los precondicionadores ILU con umbral mejoran (y se hacen más costosos de calcular) cuando el umbral disminuye.

Matrices	Dimensión	Número de no Ceros
f2da	1024	4992
f2db	1024	4992
f3d	4096	27136
orsirr1	1030	6858
orsirr2	886	5970
sherman1	1000	3750
thermal	3456	66528
nos3	960	15844
nos7	729	4617
epb1	14734	95053
epb2	25228	175027

Tabla 6.1: Matrices de prueba

Matrices	WAMG	WAMG2	WPAMG	ILU 0	ILU 0.1	ILU 0.01	ILU 0.001
f2da	12	16	11	30	31	14	6
f2db	13	18	13	N	630	20	7
f3d	7	9	7	18	19	10	6
orsirr1	44	45	39	33	34	29	15
orsirr2	38	39	34	30	31	27	14
sherman1	49	72	47	X	90	19	7
thermal	5	6	4	3	9	5	3
nos3	20	18	24	132	184	50	9
nos7	10	11	10	N	N	20	10
epb1	41	55	46	169	195	62	21
epb2	21	28	21	91	57	17	7

Tabla 6.2: Iteraciones GMRES

Los precondicionadores WAMG, WAMG2 y WPAMG se comportan relativamente bien en términos de iteraciones; los tres usan una cantidad de iteraciones similar, y en general tienen un desempeño semejante a ILU(0.01). Estos resultados deben ser interpretados cuidadosamente, debido a que el costo de un v-ciclo y el costo de un sistema triangular (para los precondicionadores ILU) depende del número de ceros en el precondicionador. El costo de un v-ciclo debe ser mayor que el costo de dos sistemas triangulares; si el número de ceros en ambos precondicionadores es similar.

El costo de cada v-ciclo en WAMG2 es más pequeño que el costo de un v-ciclo en WAMG, el cual por su parte tiene un costo menor que WPAMG. Lógicamente el número de iteraciones de WPAMG (en promedio) es más pequeño, seguido por WAMG. De cualquier forma, independientemente de algún caso, las diferencias son pequeñas. Como es usual en estos casos, escoger entre alguno de ellos no es claro y depende del problema que se intente resolver.

El costo de inicio y el costo por iteración tienen una relación directa con el número de elementos distintos de cero del precondicionador. En el caso de WAMG y WPAMG, esto se calcula sumando el número de elementos distintos de cero de todas las submatrices generadas; el inicio de WAMG2 es el mismo que para WAMG, por tanto no se incluye. El número de elementos distintos de cero se muestra en la Tabla 6.3

El número de elementos distintos de cero parece indicar que el costo de WAMG y de WPAMG son, a grueso modo, de igual magnitud que los precondicionadores ILU probados. Sin embargo el tiempo de CPU parece dar una clara ventaja (para matrices grandes) a los algoritmos WAMG y WPAMG sobre ILU(0.001).

Matrices	WAMG	WPAMG	ILU0	ILU (0.1)	ILU(0.01)	ILU(0.001)
f2da	4661	13983	4992	4992	10521	26113
f2db	4661	13983	4992	4873	10591	22876
f3d	24868	66984	27136	25186	58482	197546
orsirr1	741614	19280	6858	2678	2942	4957
orsirr2	6541	16904	5970	2365	2767	4804
sherman1	5450	15683	3750	2548	4939	10630
thermal	48277	144831	66528	8064	30453	47743
nos3	7525	22283	15844	7033	21997	48305
nos7	6675	19645	4617	3780	8051	19468
epb1	92635	277905	95053	76112	165998	412629
epb2	236813	703813	175027	137299	362837	812956

Tabla 6.3: Elementos distintos de cero en el precondicionador

Por supuesto, el tiempo de CPU es muy propenso a errores, debido a que depende mucho de la implementación, el problema y la máquina. Además, para la mayoría de las matrices probadas el tiempo de inicio fue menor. Sin embargo decidimos dar los resultados para matrices grandes, ver la Tabla 6.4.

Estos tiempos se obtuvieron usando MATLAB v 7.1, utilizando la función *luinc* para los precondicionadores ILU y el algoritmo 6.3 (Iterativamente sobre las matrices T_i) como base para los precondicionadores WAMG, WPAMG y WAMG2. Los resultados se obtuvieron usando un procesador Pentium IV a 3.0 GHz y 512 MB de RAM.

Matrices	WAMG	WPAMG	ILU0	ILU (0,1)	ILU (0,01)	ILU (0,001)
f3d	0.0310	0.0470	0.5150	0.1410	0.2030	0.3910
thermal	0.0630	0.0940	1.0000	0.0310	0.4220	1.0630
epb1	0.0940	0.1880	6.4530	0.2350	0.2960	0.5310
epb2	0.2970	0.2970	20.9220	0.1410	0.4220	0.9370

Tabla 6.4: Tiempos de CPU

Al parecer el precondicionador ILU-0 para MATLAB tiene algún tipo de deficiencia en problemas grandes, ya que muestra un crecimiento de costo muy exagerado. El costo de los ILU con umbral es típico (crece cuando disminuye el umbral). El tiempo de inicio de los precondicionadores wavelet (basado en 6.3) es pequeño y no crece en la misma medida que lo hace el de los precondicionadores ILU.

Debemos recalcar que el tiempo de inicio de los precondicionadores wavelet se obtuvo en MATLAB con el algoritmo 6.3 como base; en otras implementaciones para

calcular la DWT, basados en calcular explícitamente la matriz W, si el usuario trata de calcular la DWT de una matriz dispersa como $Epb2$, se alcanzan tiempos de cálculo inmensos e incluso en errores por falta de memoria.

Mirando de forma global los resultados, parece ser que los algoritmos basados en wavelet funcionan bien y se obtienen resultados similares a los obtenidos por las descomposiciones ILU. Cuando el umbral que se usa para la descomposición ILU es relativamente grande, el algoritmo WAMG se comporta comparativamente bien. ILU(0.001) se comporta mejor que WAMG en términos del número de iteraciones. Sin embargo su tiempo de inicio puede resultar relativamente costoso.

Una conclusión general para esta comparación es que el algoritmo WAMG puede ser una alternativa a los precondicionadores ILU cuando el costo de inicio de los precondicionadores ILU de gran precisión (umbral pequeño) se hace prohibitivo.

Sin embargo hay una clase muy común de problemas para los cuales los algoritmos multimalla basados en wavelet descritos en este capítulo tienen una ventaja clara: el problema donde se necesita resolver muchos sistema de la forma $A - hB$ (donde A, B son matrices y h es un escalar; con mucha frecuencia $B = I$), que describiremos en el próximo epígrafe.

6.5. Aplicación: Sistemas lineales desplazados en problemas de valores propios dependientes del tiempo y PDEs.

La solución eficiente de sistemas dispersos de la forma $(A - hI)x = b$, donde h es un escalar, es un tema importante en muchos campos de la ciencia. Inicialmente haremos un estado del arte de los problemas más importante relacionados con estos sistemas.

1. Solución implícita de ecuaciones ordinarias en diferencias parciales (ODEs). Considérese un sistema de ODEs

$$\frac{dY}{dt} = T(t, Y), \quad donde \quad Y \in R^n \tag{6.14}$$

Este sistema de ODEs se puede resolver con diferentes métodos, pero para pro-

blemas muy grandes se convierten usualmente a "rígidos" y necesitan de un método implícito para obtener una solución. Todos los solucionadores implícitos modernos de sistemas de ODEs necesitan resolver sistemas lineales de tiempo de la forma $(J - hI)x = b$, donde J es el Jacobiano del sistema. Por otra parte, la eficiencia y exactitud se aseguran variando el paso de tiempo, esto conlleva a variar el escalar h. Esto significa que se deben resolver varios sistemas lineales con diferentes valores de h, mientras que (usualmente) J es fijo o cambia muy lentamente.

2. PDEs dependientes del tiempo. Este sistema se puede reducir al anterior pero algunas veces se usan otro tipo de técnicas. Para cualquier PDE dependiente del tiempo a resolver por métodos implícitos, se necesita resolver varios sistemas de la forma $(A - hI)x = b$.

3. Calculo de valores propios. Muchos procedimientos para el cálculo de valores propios están basados en estrategias "Desplazar e Invertir"; esto significa que en lugar de calcular los valores propios de una matriz A directamente, intentan calcular los valores propios de $(A - hI)^{-1}$. Esta estrategia en combinación con algunos métodos iterativos adecuados, obtiene los valores propios más cercanos a "desplazamiento" h. Sin embargo, este algoritmo requerirá resolver varios sistemas de ecuaciones con matriz de coeficientes $A - hI$. La misma técnica se aplica para calcular los valores propios generalizados de un par de matrices: $(Ax = \lambda Bx)$; en este caso, el sistema a resolver tiene matriz de coeficientes $A - hB$.

   Consideremos inicialmente la solución de un sistema de ecuaciones desplazado usando métodos directos: Asumamos que se calcula la descomposición LU de la matriz $A - hI$ para un valor inicial de h, con lo que resolveríamos el primer sistema. Esa descomposición LU se puede reusar hasta tanto no cambie el valor de h, pero cuando esto pase, se debe recalcular la descomposición por completo; si h cambia con mucha frecuencia esto se torna un proceso muy costoso.

   Cuando se resuelve el sistema usando un método iterativo, surge el mismo problema con los precondicionadores más usuales como la descomposición LU incompleta; tan pronto como h cambie se debe recalcular el precondicionador aún si las matrices no cambian. En un artículo presentado por Benzi y Bertaccini en [11] se aborda este problema, y se construye un precondicionador Inverso Aproximado adaptado a este problema, pero aún este nuevo precondicionador

tiene un tiempo de inicio significativo, y se presentó solamente para el caso simétrico y definido positivo.

6.5.1. Métodos multimalla basados en la transformada wavelet para la resolución de sistemas lineales desplazados

Nuestro planteamiento de que los Métodos Multimalla Algebraicos basados en la transformada wavelet son adecuados para resolver sistemas lineales desplazados parte del hecho que la transformada wavelet que usamos es una transformación ortogonal, y por la siguientes observaciones obvias; cuando aplicamos la DWT ortogonal a una matriz desplazada $A - hI$, se convierte en:

$$(W_n(A-hI)W_n^T) = W_nAW_n^T - W_nhIW_n^T = W_nAW_n^T - hI = \begin{pmatrix} A_1 - hI & B_1 \\ C_1 & T_1 - hI \end{pmatrix} \tag{6.15}$$

Lo que significa que, una vez calculada la DWT de la matriz A, el cálculo de la DWT de cualquier matriz desplazada de A es trivial, ya que solo necesitaríamos añadir h a la diagonal de A_1 y de T_1. Si se calculan más niveles de la DWT, (aplicando DWT sobre T_i) es igualmente obvio que las submatrices calculadas se actualizan de la misma forma.

Esto significa que los algoritmos multimallas basados en wavelet tienen una ventaja importante sobre la mayor parte de los precondicionadores (incluyendo la descomposición LU incompleta), en la situación en que se necesita resolver muchos sistemas de la forma $A - hI$ (con h cambiante), los mismos solo necesitan de una fase de inicialización, para calcular los niveles deseados de la DWT de la matriz A. Una vez que se haya calculado, la fase de inicialización para cualquier matriz desplazada $A - hI$ es trivial e insignificante ya que solo se necesita actualizar la diagonal de la matriz.

Si el sistema a resolver es de la forma $A - hB$, $B \neq I$, la actualización incluye la matriz B completa. Esto de alguna manera es más costoso pero seguramente aún menos costosos que recalcular otros precondicionadores.

6.6. Resumen de aportaciones

Se propusieron dos nuevos algoritmos multimalla algebraicos competitivos basados en la DWT. El primero llamado WPAMG, resuelve aproximadamente el sub-sistema de alta frecuencia. El segundo, WAMG2, es básicamente el mismo algoritmo pero evita pasos de suavizado en algunos niveles. Los algoritmos propuestos y el algoritmo original WAMG, se comparan como precondicionadores para el GMRES con cuatro precondicionadores de tipo ILU obteniéndose resultados satisfactorios. Una conclusión general para esta comparación es que el algoritmo WAMG puede ser una alternativa a los precondicionadores ILU cuando el costo de inicio de los precondicionadores ILU de gran precisión (umbral pequeño) se hace prohibitivo.

Por último se discute la posibilidad de usar WPAMG, WAMG2 o WAMG para resolver eficientemente sistemas lineales desplazados de la forma $A - hI$ o $A - hB$, donde h es un escalar que toma distintos valores.

Estos resultados fueron publicados en el artículo "Variants of algebraic wavelet-based multigrid methods: Application to shifted linear systems" en la revista "Applied Mathematics and Computation", en el 2008, referencia [31].

Capítulo 7

Conclusiones y trabajos futuros

Las principales aportaciones de este trabajo se pueden resumir en los siguientes puntos:

- Se estudiaron los algoritmos paralelos más relevantes de la literatura para el cálculo de la DWT no estándar. Se propuso una variante de paralelización que generaliza los métodos ya existentes. Se propuso una forma de calcular la cantidad de elementos necesarios para comunicar valores sólo una vez al inicio del algoritmo; lo que evitaría las comunicaciones con un coste relativamente pequeño de replicación de los cálculos. Estos resultados se han presentado en la ponencia "Partial Data Replication as a strategy for Parallel Computing of the Multilevel Discrete Wavelet Transform", aceptada para su presentación en la conferencia internacional "International Conference on Parallel Processing And Applied Mathematics, PPAM 2009", referencia [8]. También se estudió el cálculo paralelo de la DWT para la distribución 2DBC, cuyo uso se analiza también en el capítulo 5 en el epígrafe 5.1 y que dio lugar a la publicación "Compatibility of ScaLAPACK with the discrete wavelet transform". Esta ponencia fue presentada en la conferencia internacional "International Conference on Computational Science -ICCS 2007" y publicado en las "Lecture Notes in Computer Science", en el año 2007, referencia [7]. Finalmente se mejoró este algoritmo buscando una permutación de los datos que disminuye la cantidad de comunicaciones.
- Se propusieron algoritmos para calcular la DWT-2D no estándar en matrices

dispersas sin necesidad de construir explícitamente la matriz de trasformación. Estos algoritmos usan la forma de representación CSR. También se propusieron variantes para disminuir el fenómeno de rellenado que se produce en este tipo de matrices cuando se calcula la DWT; basados en reordenar la matriz para obtener una configuración donde los elementos distintos de cero estén cercanos unos con otros. Adicionalmente se sugieren estos reordenamientos para mejorar la convergencia de los métodos multimalla algebraicos basados en la DWT.

- Se estudiaron distintas variantes de construcción de precondicionadores basados en la DWT para acelerar la convergencia de métodos iterativos en la resolución de ecuaciones lineales. Se propuso una variante de paralelización que usa el algoritmo paralelo de cálculo de la DWT con replicación parcial. Este resultado fue publicado en la "Revista Cubana de Ciencias Informáticas", en el 2007, referencia [6]. También se propone una solución efectiva para la solución de problemas densos de gran dimensión que utiliza un precondicionador basado en el álgebra no estándar.

- Finalmente se propusieron dos nuevos algoritmos multimalla algebraicos competitivos basados en la DWT. El primero llamado WPAMG, resuelve aproximadamente el sub-sistema de alta frecuencia. El segundo, WAMG2, es básicamente el mismo algoritmo pero evita pasos de suavizado en algunos niveles. Los algoritmos propuestos y el algoritmo original WAMG, se comparan como precondicionadores para el GMRES con cuatro precondicionadores de tipo ILU obteniéndose resultados satisfactorios. Por último se discute la posibilidad de usar WPAMG, WAMG2 o WAMG para resolver eficientemente sistemas lineales desplazados de la forma $A - hI$ o $A - hB$, donde h es un escalar que toma distintos valores. Los resultados de este último aporte fueron publicados en el artículo "Variants of algebraic wavelet-based multigrid methods: Application to shifted linear systems" en la revista "Applied Mathematics and Computation", en el 2008, referencia [31].

Trabajos Futuros

Los trabajos futuros que se pueden derivar de la presente tesis son los siguientes:

- Analizar con detalle los reordenamientos para matrices dispersas vistos en el

capítulo 4. Una posibilidad es considerar otros tipos de reordenamiento, cómo el reordenamiento en bloques densos propuesto en [53], donde se reorganiza la matriz para hacer eficiente el producto matriz vector disperso en cuanto al uso de la cache. El objetivo de este tipo de reordenamiento es obtener una matriz dispersa con los elementos lo más cercanos unos de otros reduciendo el mismo al problema del viajero vendedor *TSP*[1]. También se podría considerar como trabajo futuro seguir estudiando la influencia de la redistribución de la norma al calcular la DWT de una matriz reordenada en la convergencia de los métodos wavelet multimalla vistos en el capítulo 6.

- Se propone también estudiar los métodos iterativos usando el álgebra no estándar descrita en el capítulo 5 en problemas como: el problema MoM[2] [55]. Los problemas MoM han probado ser particularmente factibles de simplificar usando la trasformada wavelet y la transformada wavelet Packet [64, 38, 65, 61, 66, 33]. Sería interesante utilizar el álgebra no estándar en combinación con algunos métodos iterativos clásicos, tal y como se propone en el capítulo 5 epígrafe 5.2, para resolver este tipo de problemas MoM.

Soporte de la tesis

Este trabajo ha recibido soporte financiero de los siguientes proyectos e instituciones:

- Proyecto CICYT TIC2003-08238-C02-02 “Desarrollo y optimización de código paralelo para sistemas de Audio 3D”.
- Proyecto CICYT TIN2008-06570-C04-02 “COPABIB-UM: Construcción y optimización automáticas de bibliotecas paralelas de computación científica”.
- Proyecto UPV 2008 0009 “Software científico”.
- Generalitat Valenciana bajo el proyecto 20080811.
- Oficina de Acción Internacional de la Universidad Politécnica de Valencia.
- Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

[1] Travelling Saleman Problem
[2] Del inglés Method of Moments

- Dirección de Formación Posgraduada y Dirección de Cooperación Internacional de la Universidad de las Ciencias Informáticas, Habana, Cuba.

Apéndice A

Filtros de Daubechies

La tabla A.1 muestra los coeficientes de la función escala de Daubechies para los ordenes de 1 al 6. Se usa la notación de MATLAB para los filtro de Daubechies dbx, donde db es el tipo de filtro, en este caso de tipo Daubechies, x es el índice del filtro y $2x$ la longitud del mismo. Los coeficientes wavelet se pueden obtener a partir de los coeficientes de la función escala invirtiendo el orden los mismos y luego cambiando el signo por cada dos (p.e, $db2 = \{-0{,}0915064, -0{,}1584937, 0{,}5915064, -0{,}3415064\}$). Matemáticamente cada coeficiente wavelet de índice k se obtiene $h_k = (-1)^k c_{2N-1-k}$, donde h son los coeficientes de la secuencia wavelet, c los coeficientes de la secuencia de escala. N es el índice wavelet (p.e $N = 2$ para $db2$).

$db1$ ($Haar$)	$db2$	$db3$	$db4$	$db5$	$db6$	$db\ldots$
0.5	0.3415064	0.2352336	0.1629017	0.1132095	0.0788712	
0.5	0.5915064	0.5705585	0.5054729	0.4269718	0.3497519	
	0.1584937	0.3251825	0.4461007	0.5121635	0.5311319	
	-0.0915064	-0.0954672	-0.0197875	0.0978835	0.2229157	
		-0.0604161	-0.1322536	-0.1713284	-0.1599933	
		0.0249088	0.0218082	-0.0228006	-0.0917590	
			0.0232518	0.0548513	0.0689440	
			-0.0074935	-0.0044134	0.0194616	
				-0.0088959	-0.0223319	
				0.0023587	0.0003916	
					0.0033780	

Tabla A.1: Coeficientes de daubechies ortogonales hasta el índice 6. La notación dbx es tomada de MATLAB, x es el índice del filtro y $2x$ la longitud.

Índice de tablas

Índice de figuras

Índice de algoritmos

Lista de abreviaturas

WSQ	Del Inglés Wavelet Scalar Quantization, cuantización escalar de wavelets.
FFT	Transformada Rápida de Fourier.
STFT	Trasformada Corta de Fourier.
DWT	Transformada Wavelet Discreta.
CWT	Transformada Wavelet Continua.
DWT-1D	Transformada Wavelet Discreta en una dimensión.
DWT-2D	Transformada Wavelet Discreta en dos dimensiones.
CSR	Tomado del Inglés. Compressed Sparse Row. Almacenamiento comprimido por filas.
CSC	Tomado del Inglés. Compressed Sparse Column. Almacenamiento comprimido por Columnas.
MRA	Tomado del Inglés. Multi-Resolution analysis. Análisis multi-resolución.
2DBC	Distribución Bidimensional Cíclica y por Bloques. Es la distribución típica de la librería ScaLAPACK.

Lista de símbolos

H	Filtro wavelet de descomposición de alta frecuencia.
G	Filtro wavelet de descomposición de baja frecuencia.
H'	Filtro wavelet de reconstrucción de alta frecuencia.
G'	Filtro wavelet de reconstrucción de baja frecuencia.
W	Matriz de coeficiente wavelet, incluye tanto los de alta como los de baja frecuencia.
cA	Componentes de aproximación de una señal.
cD	Componentes de detalle de una señal.
cA_i	Componentes de aproximación de una señal en el nivel i.
cD_i	Componentes de detalle de una señal en el nivel i.
$\downarrow$	Cuando aparece dentro de un círculo, representa el proceso de subsampleo. Se descartan componentes redundantes de una señal. En la DWT uno de cada dos componentes son descartados.
$\uparrow$	Cuando aparece dentro de un círculo, representa el proceso inverso al subsampleo.
$\{g_i\}_{i=1}^{L}$	Filtro pasa bajo de dimensión L. Para obtener la información de baja frecuencia.
$\{h_i\}_{i=1}^{L}$	Filtro pasa alto de dimensión L. Para obtener la información de alta frecuencia.

Bibliografía

[1] National Institute of Standards and Technology, "Matrix Market". Available from: http://math.nist.gov/MatrixMarket. Citado en p. 76, 109

[2] The Wavelet Digest. wavelet.org. Citado en p. 1

[3] *Several Strategies for Reducing the Bandwidth of Matrices*, Papers of the Symposium on Sparse Matrices and their Applications, New York, 1972. IBM Thomas J. Watson Research Center. Citado en p. 3, 74

[4] MATLAB R14 Natick MA, 2004. Citado en p. 108

[5] F. Abramovich, T. Bailey, and T. Sapatinas. Wavelet analysis and its statistical applications. *JRSSD*, 48:1–30, 2000. Citado en p. 1

[6] L. Acevedo, V. M. García, and A. M. Vidal. Transformada wavelet paralela en la resolución de sistemas lineales densos. *Revista Cubana de Ciencias Informáticas*, 1:32–46, 2007. Citado en p. 91, 98, 118

[7] L. Acevedo, V. M. Garcia, and A. M. Vidal. Compatibility of ScaLAPACK with the discrete wavelet transform. *Lecture Notes in Computer Science*, 4487:152–159, 2007. Citado en p. 44, 59, 117

[8] L. Acevedo, V. M. Garcia, A. M. Vidal, and P. Alonso. Partial Data Replication as a strategy for Parallel Computing of the Multilevel Discrete Wavelet Transform. Eighth International Conference on Parallel Processing and Applied Mathematics, PPAM 2009, Poland., 2009. Citado en p. 59, 117

[9] P. R. Amestoy, T. A. Davis, and I. S. Duff. An Approximate Minimum Degree Ordering Algorithm. *SIAM Journal on Matrix Analysis and Applications,*, 17:886–905, 1996. Citado en p. 3, 73

[10] A. Antoniadis. Wavelets in statistics: a review. *J. It. Statist. Soc.*, 1999. Citado en p. 1

[11] M. Benzi and D. Bertaccini. Approximate Inverse Preconditioning for Shifted Linear systems. *BIT Numerical Mathematics*, 43:231–244, 2003. Citado en p. 4, 113

[12] G. Beylkin, R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms. *Commun. Pure Appl. Math.*, XLIV:141–183, 1991. Citado en p. 4, 85

[13] L. S. Blackford, J. Choi, A. Cleary, E. D. J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *Software, enviroment, tools. ScaLAPACK Users Guide.* Society for Industrial and Applied Mathematics, 1997. Citado en p. 44, 51

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial (2nd ed.).* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. Citado en p. 5, 100

[15] S. Burrus, R. Gophinath, and H. Guo. *Introduction to Wavelet and Wavelet Transforms.* Prentice Hall, New Jersey, 1998. Citado en p. 17

[16] T. F. Chan and K. Chen. On Two Variants of an Algebraic Wavelet Preconditioner. *SIAM Journal on Scientific Computing*, 24:260–283, 2003. Citado en p. 3, 4, 5, 7, 31, 61, 85, 86, 87, 88

[17] T. F. Chan, W. Tang, and W. Wan. Wavelet sparse approximate inverse preconditioners. *BIT*, 37:644–660, 1997. Citado en p. 4, 85

[18] D. Chaver, M. Prieto, L. Piuel, and F. Tirado. Parallel Wavelet for Large Scale Image Processing. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS'2002)*, 2002. Citado en p. 3, 42, 43, 44

[19] K. Chen. An analysis of sparse approximate inverse preconditioners for boundary integral equations. *SIAM J. Matrix Anal. Appl.*, 22:1058–1078, 2001. Citado en p. 4, 85

[20] K. Chen. *Matrix Preconditioning Techniques and Applications.* Cambridge University Press, 2005. Citado en p. 4, 87, 88

[21] C. Chiann and P. A. Morettin. A wavelet analysis for time series. *Journal of Nonparametric Statistics*, 1(10):1–46, 1999. Citado en p. 1

[22] W. Cochran, J. Cooley, D. Favin, H. Helms, R. Kaenel, W. Lang, J. Maling, G.C., D. Nelson, C. Rader, and P. Welch. What is the fast Fourier transform. *Proceedings of the IEEE*, 55(10):1664– 1674, Oct 1967. Citado en p. 13

[23] R. Coifman and M. Wickerhauser. Entropy-based algorithms for best basis selection. *Information Theory, IEEE Transactions on*, 38(2):713–718, Mar 1992. Citado en p. 26, 35

[24] R. R. Coifman, Y. Meyer, S. R. Quake, and M. V. Wickerhauser. *Signal processing and compression with wavelet packets*. In Y. Meyer and S. Roques, 1993. Citado en p. 35

[25] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. *Proceedings of the 24th National Conference of the ACM*, 1969. Citado en p. 3, 73, 74

[26] I. Daubechies. *Ten Lectures on Wavelets.* SIAM: Society for Industrial and Applied Mathematics, 1992. Citado en p. 1

[27] I. Daubechies. Orthonormal Bases of Compactly Supported wavelets II. variations on a Theme. *SIAM J. Math. Anal.*, 24(2):499–519, 1993. Citado en p. 27

[28] T. Davis. University of Florida sparse matrix collection. *NA Digest*, 97, 1997. Citado en p. 76, 107, 109

[29] T. A. Davis, J. R. Gilbert, S. Larimore, and E. Ng. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30:353 – 376, 2004. Citado en p. 3, 73

[30] J. Dongarra, R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, V. Eijkhout, R. Pozo, C. Romine, and H. Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1993. Citado en p. 85, 107

[31] V. M. García, L. Acevedo, and A. M. Vidal. Variants of algebraic wavelet-based multigrid methods: Application to shifted linear systems. *Applied Mathematics and Computation*, 2021(202):287–299, 2008. Citado en p. 115, 118

[32] D. Gines, G. Beylkin, and J. Dunn. Lu Factorization of Non-standard Forms and Direct Multiresolution Solvers. *Applied and Computational Harmonic Analysis*, 5(2):156 – 201, 1998. Citado en p. 4, 5

[33] J. C. Goswami and M. Tentzeris. Wavelets in Electromagnetics. *Encyclopaedia of Microwaves Engineering*, 1:377–396, 2005. Citado en p. 119

[34] G.Wang, R. W. Dutton, and J. Hou. A Fast wavelet Multigrid Algorithm for solution of Electromagnetic Integral Equations. *Microwave and Optical Technology Letters*, 24(2), 2000. Citado en p. 6, 100, 102, 103, 105, 108

[35] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 277–286, New York, NY, USA, 1995. ACM. Citado en p. 1, 4

[36] S. Kasaei, M. Deriche, and B. Boashash. Fingerprint compression using a modified wavelet transform andpyramid lattice vector quantization. *TENCON 96. Proceedings. 1996 IEEE TENCON. Digital Signal Processing Applications*, 2:798–803, 1996. Citado en p. 1

[37] D. Keim and M. Heczko. *Wavelets and their applications in databases*. Tutorial Notes of ICDE 2001, 2001. Citado en p. 1

[38] M. Krumpholz and L. P. B. Katehi. MRTD: New Time-Domain SchemesBased on Multiresolution Analysis. *IEEE Transactions On Microwave Theory And Techniques*, 44(4), Abril 1996. Citado en p. 119

[39] V. Kumar., A. Grama., A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and analysis of algorithms*. Addison Wesley, January 2003. Citado en p. 38

[40] D. R. N. D. Leon. Wavelet Operators Applied to Multigrid Methods (Ph Tesis). Technical Report CAM Report 00-22, UCLA Mathematics Department, 2000. Citado en p. 6, 100, 102

[41] Li, Tao, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explor. Newsl.*, 4(2):49–68, December 2002. Citado en p. 1

[42] J. Liu and A. Sherman. Comparative analysis of the Cuthill-Mckee and the reverse Cuthill-Mckee ordering algorithms for sparse matrices. *SIAM Journal of Numerical Analysis.*, 13:198–213, 1975. Citado en p. 74

[43] C. V. Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial Mathematics, January 1987. Citado en p. 14

[44] S. Mallat. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. *PAMI*, 11(7):674–693, July 1989. Citado en p. 24

[45] S. Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 1999. Citado en p. 1, 20

[46] H. M. Markowitz. The Elimination Form of the Inverse and Its Application to Linear Programming. *Management Science*, 3(3):255–269, Apr 1957. Citado en p. 73, 74

[47] P. Meerwald and A. Uhl. A survey of wavelet-domain watermarking algorithms. *Proceedings of SPIE, Electronic Imaging*, 4314, 2001. Citado en p. 1

[48] M. Misiti, Y. Misiti, G. Openheim, and J. M. Poggy. Wavelet Toolbox, User' s Guide. *The Math Works*, 2000. Citado en p. 1, 18

[49] G. P. Nason and R. von Sachs. Wavelets in time series analysis. *Philosophical Transactions of the Royal Society of London A*, 1760(357):2511–2526, 1999. Citado en p. 1

[50] O. Nielsen and M. Hegland. A scalable parallel 2D wavelet transform algorithm. Technical Report TR-CS-97-21, Australian National University, 1997. Citado en p. 2, 7, 37, 38, 40, 41, 42, 44

[51] D. Percival and A. T. Walden. Wavelet Methods for Time Series Analysis. *Cambridge University Press,*, 2000. Citado en p. 1

[52] F. H. Pereira, S. L. Verardi, and S. I. Nabeta. A wavelet-based algebraic multigrid preconditioner for sparse linear systems. *Appl Math. and Comput*, 182:1098–1107, 2006. Citado en p. 6, 100, 103, 105, 107, 108, 109

[53] A. Pinar and M. T. Heath. Improving performance of sparse matrix-vector multiplication. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, number 30, 1999. Citado en p. 73, 119

[54] L. Prasad, S. S. Iyengar, and S. S. Ayengar. Wavelet Analysis with Applications to Image Processing. *CRC Press,*, 1997. Citado en p. 1

[55] Ray and Mittra. *Computational Methods of Electromagnetic Scattering Peterson.* WileyBlackwell, 1997. Citado en p. 119

[56] V. Rokhlin. Diagonal Forms of Translation Operators for Helmholtz Equation in Three Dimensions. *Applied and Computational Harmonic Analysis*, 5(2):156–201(46), Apr 1998. Citado en p. 85, 91, 96, 97

[57] Y. Saad. *SPARSKIT: A basic tool-kit for sparse matrix computations.* University of Minnesota, Department of Computer Science and Engineering, 200 Union Street S.E., Minneapolis, MN 55455 USA, 2 edition, June 1994. Citado en p. 62

[58] Y. Saad. *Iterative methods for sparse linear systems.* PWS Publishing Company, 1996. Citado en p. 73, 108

[59] B. Steinberg and Y. Leviatan. On the use of wavelet expansions in the method of moments. *Antennas and Propagation, IEEE Transactions*, 41:610 – 619, May 1993. Citado en p. 4

[60] D. S. Taubman and M. W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice.* Springer, 2001. Citado en p. 1

[61] M. Tentzeris. Multiresolution time-domain (MRTD) adaptive schemes using arbitrary resolutions of wavelets. *IEEE Trans. Microwave Theory Tech*, 50, 2002. Citado en p. 119

[62] P. Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: a tutorial. *In Proc. IEEE*, pages 56–93, 1990. Citado en p. 1

[63] B. Vidakovic. *Statistical Modeling by Wavelets.* John Wiley & Sons, New york, 1999. Citado en p. 1

[64] R. Wagner and Chew. A Study of Wavelets for the Solution of Electromagnetic Integral Equations. *IEEE Transactions on antennas and propagation*, 43(2), Agosto 1995. Citado en p. 4, 119

[65] L. G. Wojciech. Wavelet Packets for Fast Solution of Electromagnetic Integral Equations. *IEEE Transactions on antennas and propagation*, 46(5), Mayo 1998. Citado en p. 119

[66] Y.Tertakiov. On sampling-biorthogonal time-domain scheme based on Daubechies compactly supported wavelets. *Progress In Electromagnetics Research*, 47:213–234, 2004. Citado en p. 119