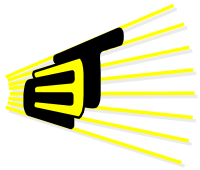
DESEMPEÑO COMPUTACIONAL SOBRE UNA ARQUITECTURA ALTERNATIVA DE ALTO RENDIMIENTO DE UN ALGORITMO DE TRANSFORMACIÓN APLICADO A DATOS SÍSMICOS

CARLOS ANDRÉS ANGULO JULIO



Escuela de Ingenierías
Eléctrica, Electrónica y
de Telecomunicaciones

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2016

DESEMPEÑO COMPUTACIONAL SOBRE UNA ARQUITECTURA ALTERNATIVA DE ALTO RENDIMIENTO DE UN ALGORITMO DE TRANSFORMACIÓN APLICADO A DATOS SÍSMICOS

Autor:
CARLOS ANDRÉS ANGULO JULIO

Trabajo de investigación presentado como requisito para optar al título de
Magíster en Ingeniería Electrónica

Director:
MSc. Carlos A. Fajardo Ariza

Codirector:
MSc. Jorge H. Ramón S.

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2016

A Amalia y Rafael: ustedes son quienes me guían.

AGRADECIMIENTOS

Gracias Dios por darme una nueva oportunidad para alcanzar este logro.

Este trabajo fue desarrollado con el apoyo de ECOPETROL y COLCIENCIAS como parte del proyecto de investigación No. 0266-2013.

Al cerrar este capítulo de mi vida tengo que reconocer que el mérito no es solo mío, este logro también es de todos aquellos que me acompañaron durante este proceso. Gracias a todos ustedes por permitir que este trabajo llegara a feliz término.

Agradezco a Carlos Fajardo por sus consejos y su invaluable y constante apoyo, no solo como director sino también como compañero de trabajo y especialmente como amigo.

Igualmente al profesor Jorge Ramón (q.e.p.d.) por su disposición, acompañamiento y la confianza depositada en mí durante tantos años.

A los miembros del grupo de investigación CPS por brindar tan excelente ambiente de trabajo y sus acertadas observaciones en el debido momento.

Mención especial a Fabián Sánchez, Iván Obregón, Julián Mantilla, Jairo Castelar, Christian Hernández, Gabriel Rincón, Marcos Caicedo, Manuel Pérez y Carlos Boada por sus aportes tan valiosos para esta investigación.

También a Laura Convers que con su excelente labor administrativa facilitó el desarrollo de las diversas actividades relacionadas con este trabajo de investigación.

A mi familia por su incansable apoyo y paciencia, creer siempre en mí, comprenderme en los momentos en que parecía ausente y darme la fortaleza para seguir adelante.

CONTENIDO

Pág.

INTRODUCCIÓN **11**

1 COMPRESIÓN DE DATOS **13**
1.1. Transformación 15
1.2. Cuantificación 16
1.3. Codificación 17
1.4. Trabajos relacionados 18

2 SELECCIÓN DE LA ESTRATEGIA DE TRANSFORMACIÓN **20**
2.1. Esquema *Lifting* 20
2.2. Comparación filtros Wavelet 22
2.3. Polinomios del esquema *lifting* 24

3 DESCOMPRESIÓN EN FPGA **27**
3.1. Compresión 27
3.2. Descompresión 29
3.2.1. Decodificación en FPGA 30
3.2.2. Cuantificación inversa en FPGA 31
3.2.3. Transformación inversa en FPGA 31
3.3. Resultados 34

4 DESCOMPRESIÓN EN GPU **36**
4.1. Compresión 36
4.2. Descompresión 37
4.2.1. Decodificación en GPU 38
4.2.2. Cuantificación inversa en GPU 39
4.2.3. Transformación inversa en GPU 39
4.3. Resultados 41

5 CONCLUSIONES **45**

REFERENCIAS **47**

BIBLIOGRAFÍA **51**

ANEXOS **55**

LISTA DE FIGURAS

Pág.

Figura 1 Esquema de compresión–descompresión. 14
Figura 2 Histograma de un conjunto de datos cuantificados. 15
Figura 3 Transformación Wavelet Discreta 1D 16
Figura 4 Transformación Waveled Discreta 2D. 16
Figura 5 Creación del diccionario Huffman. 18
Figura 6 Esquema *lifting* para la *DWT* 1D 21
Figura 7 *SNR* vs. *CR* para compresión sin transformación. 22
Figura 8 Compresión de datos sísmicos con filtros Wavelet *CFD*. 23
Figura 9 Compresión de datos sísmicos con filtros Wavelet *Daubechies*. . . . 23
Figura 10 Compresión de datos sísmicos con filtros Wavelet *Symlet*. 24
Figura 11 Tiempo de ejecución de la *IDWT* en CPU. 26
Figura 12 Esquema del algoritmo de compresión en CPU. 27
Figura 13 Esquema *lifting* para la *DWT 2D*. 28
Figura 14 Esquema de descompresión empleado. 29
Figura 15 Diagrama general del circuito de descompresión en FPGA. 30
Figura 16 Circuito Decodificador Huffman. 30
Figura 17 Circuito Cuantificación inversa. 31
Figura 18 Circuito general de la *IDWT 2D* mediante esquema *lifting*. 32
Figura 19 Esquema *lifting* para la *IDWT 2D*. 32
Figura 20 Circuito Actualización. 33
Figura 21 Circuito Predicción. 33
Figura 22 Número de ciclos de reloj empleados en la descompresión en FPGA. 35
Figura 23 Ejemplo de almacenamiento del diccionario Huffman. 36
Figura 24 Ejemplo de almacenamiento de los datos codificados. 37
Figura 25 Ejemplo de almacenamiento en paquetes de los datos codificados. . 37
Figura 26 Relación de compresión usando almacenamiento en paquetes. 38
Figura 27 Esquema de descompresión empleado. 38
Figura 28 Ejemplo del proceso de decodificación 39
Figura 29 Esquema *lifting* para la *IDWT* 2D. 40
Figura 30 Esquema *lifting* para la *IDWT* 1D. 40
Figura 31 Jerarquía de hilos 42
Figura 32 Tiempo de ejecución de un *kernel* con *threads* divergentes. 42
Figura 33 Tiempo de ejecución del *kernel* de decodificación. 43
Figura 34 Tiempo de descompresión para diferentes *datasets*. 44

RESUMEN

TÍTULO: Desempeño computacional sobre una arquitectura alternativa de alto rendimiento de un algoritmo de transformación aplicado a datos sísmicos*

AUTOR: Carlos Andrés Angulo Julio**

PALABRAS CLAVE: Compresión, Datos sísmicos, Descompresión, Esquema *Lifting*, FPGA, GPU, Transformación Wavelet, Trazas sísmicas.

DESCRIPCIÓN:

La industria del petróleo emplea imágenes del subsuelo para determinar la presencia de depósitos minerales. La construcción de estas imágenes requiere procesar una gran cantidad de trazas sísmicas en arquitecturas de alto desempeño. El tiempo de ejecución de los algoritmos sísmicos es penalizado por el cuello de botella en la transferencia de datos entre la memoria principal y la memoria de los nodos de cómputo debido a la cantidad de trazas involucradas.

Para reducir el tiempo de transferencia y almacenar eficientemente la información sísmica se pueden emplear técnicas de compresión de datos sísmicos. Generalmente, estas técnicas comprenden tres etapas: transformación, cuantificación y codificación. Uno de los retos que surge para obtener un algoritmo de compresión adecuado es hallar una transformación que permita concentrar la energía de las trazas sísmicas en unos pocos datos. Esto con el fin de mejorar la relación de compresión sin afectar considerablemente la calidad de la información.

En este proyecto se analiza la Transformación Wavelet Discreta con el fin de seleccionar una estrategia para implementar la etapa de transformación de un algoritmo de compresión de datos sísmicos. El análisis se centra en la relación de compresión de los datos y el tiempo de descompresión en arquitecturas de procesamiento en paralelo tales como FPGA y GPU. El objetivo es proponer una estrategia de transformación que favorezca simultáneamente la relación de compresión y el tiempo de descompresión.

Los resultados muestran que el esquema *lifting* reduce el tiempo de procesamiento ya que requiere menos operaciones matemáticas. Asimismo, los resultados sugieren usar el esquema *lifting* para el filtro wavelet *CDF 2.2* ya que este reduce el tiempo de procesamiento y ofrece uno de los mejores rendimientos en términos de Relación de Compresión vs. Relación Señal a Ruido.

* Trabajo de investigación.

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Maestría en Ingeniería Electrónica. Director: MSc. Carlos A. Fajardo Ariza.

ABSTRACT

TITLE: Computational performance on a high-performance alternative architecture of a transformation algorithm applied to seismic data*

AUTHOR: Carlos Andrés Angulo Julio**

KEYWORDS: Compression, Decompression, FPGA, GPU, Lifting Scheme, Seismic data, Seismic traces, Wavelet Transform

DESCRIPTION:

Subsurface images constructed by the oil industry to determine the presence of mineral deposits require processing a huge amount of seismic data on high performance architectures. The processing time of seismic algorithms is penalized by a data transfer bottleneck between the main memory and the memory of the computing nodes due to the amount of data involved.

Data compression techniques can be used to reduce the transfer time and to store the seismic information efficiently. Such techniques usually comprise three stages: transformation, quantization and coding. One of the challenges that arises is to find a transformation to achieve a suitable compression algorithm for seismic data that allows to concentrate their energy on few data in order to favor the compression ratio while maintaining the information quality.

The Discrete Wavelet Transform is analyzed to select a strategy to implement the transformation stage of a seismic data compression algorithm. The analysis focusses on the compression ratio of the seismic data and the execution time of the decompression algorithm on parallel architectures such as FPGA and GPU. The aim is to propose a transformation stage that simultaneously favors the compression ratio and the decompression time.

Results show that the lifting scheme reduces the amount of mathematical operations and therefore the processing time. Results also suggest using the lifting scheme for the *CDF 2.2* wavelet filter because it reduces the decompression time and it offers the best performance in terms of compression ratio vs. signal to noise ratio.

* Trabajo de investigación.

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Maestría en Ingeniería Electrónica. Director: MSc. Carlos A. Fajardo Ariza.

INTRODUCCIÓN

El presente proyecto está enmarcado dentro del programa de investigación "Migración sísmica pre–apilado en profundidad por extrapolación de campos de onda utilizando computación de alto desempeño para datos masivos en zonas complejas". Dicha investigación pretende aumentar la resolución de las imágenes del subsuelo requeridas por la industria del petróleo para identificar posibles reservas de hidrocarburos. Asimismo, dicha investigación busca reducir el tiempo de ejecución de las aplicaciones sísmicas usando arquitecturas de cómputo diferentes a las CPU.

La construcción de las imágenes del subsuelo es un proceso en el cual se generan grandes volúmenes de datos (del orden de decenas de Terabytes) [1, 2]. El uso de técnicas de compresión de datos al momento de su adquisición resulta útil puesto que aumenta la velocidad de transferencia de la información a los dispositivos de procesamiento y reduce los costos relacionados con la transmisión y almacenamiento de los datos [3–6].

Existen diversos métodos para comprimir datos, todos ellos basados en la reducción de la redundancia presente en los datos originales, debido a que la redundancia depende del tipo de datos (texto, audio, imagen, video, etc.) [7]. Uno de los aspectos a tener en cuenta a la hora de escoger un esquema de compresión de datos sísmicos es la selección una base adecuada con la cual se pueda representar eficientemente la información del subsuelo [8]. La nueva representación se consigue aplicando algún tipo de transformación a los datos originales con el fin de concentrar la energía de las trazas sísmicas* en pocos coeficientes. Con esto se busca mejorar los resultados de la compresión al aumentar la redundancia que se puede reducir.

La Transformación Wavelet Discreta ha resultado ser muy efectiva en diversas aplicaciones, entre las cuales se encuentra la compresión de datos sísmicos [3, 9, 10]. La virtud de la Transformación Wavelet que la convierte en una buena herramienta en el campo de la compresión está en la resolución espacio–frecuencia que ofrece. Esta multirresolución se obtiene al comparar sistemáticamente la señal de entrada con un conjunto de filtros hasta lograr el nivel deseado de descomposición en espacio y en frecuencia.

Una vez transferidos los datos comprimidos al dispositivo de cómputo, estos deben descomprimirse rápidamente para poder ser procesados y así obtener las imágenes del subsuelo [11]. Dispositivos de cómputo tales como FPGAs y GPUs son usados para disminuir el tiempo de ejecución de los algoritmos que permiten obtener dichas imágenes [12–14]. Los resultados obtenidos en [10] muestran que los algoritmos de compresión basados en la Transformación Wavelet Discreta son más rápidos que aquellos basados en

* Trazas sísmicas: Nombre con el cual se conocen los datos adquiridos en un experimento sísmico.

otra transformación. Asimismo, en [15] se realizaron pruebas con un conjunto de datos correspondientes a exploraciones marinas obteniendo mejores resultados al seleccionar filtros de mayor longitud.

Por lo tanto, un criterio de selección para la transformación de los datos sísmicos consiste en escoger un conjunto de filtros y una estrategia de descomposición que permitan identificar efectivamente la información más relevante para favorecer la relación de compresión que se puede alcanzar. De igual forma, la transformación debe permitir llevar a cabo la descompresión en el menor tiempo posible.

En el presente proyecto se evalúa el desempeño computacional, en términos de la relación de compresión y tiempo de descompresión, de un algoritmo de Transformación Wavelet Discreta Inversa implementado tanto en GPU como en FPGA usado en el contexto de la compresión–descompresión de trazas sísmicas. Los datos fueron suministrados por ECOPETROL y corresponden a conjuntos de 96 trazas sísmicas con 3584 muestras por traza.

Los resultados sugieren que la familia de filtros wavelet biortogonales *CDF 2.x* pueden conducir a mejores resultados que otras familias de wavelets. La implementación de la Transformación Wavelet Discreta se llevó a cabo empleando el esquema *lifting* y el filtro *CDF 2.2* ya que implican mayor velocidad de ejecución debido a su menor complejidad computacional.

El presente documento está organizado de la siguiente manera: En el capítulo 1 se presenta una breve explicación de la compresión de datos y las etapas empleadas para comprimir datos sísmicos. En el capítulo 2 se exploran estrategias para llevar a cabo la etapa de transformación. En los capítulos 3 y 4 se muestra la implementación de la descompresión de datos sísmicos en FPGA y GPU respectivamente. Asimismo se exponen los resultados obtenidos en cada arquitectura. Por último, en el capítulo 5 se presentan las conclusiones y recomendaciones.

1. COMPRESIÓN DE DATOS

El objetivo de la compresión es representar la información de un mensaje con menos bits que los usados en la representación original. Esto conlleva a la reducción de los costos asociados al tiempo de transferencia de la información y su almacenamiento.

La compresión es posible debido a que el mensaje puede contener datos redundantes que podrán ser eliminados sin afectar considerablemente información que éste tenga. De acuerdo con el teorema de Shannon [16], la entropía cuantifica la incertidumbre asociada a una variable aleatoria para determinar la cantidad de información de una fuente de datos. La entropía H estima el número promedio de bits requeridos para representar un mensaje de n símbolos $(x_1, x_2, \cdots, x_n)$ de acuerdo a la probabilidad de ocurrencia P_i del símbolo x_i usando la expresión

$$H = -\sum_{i=1}^{n} (P_i \cdot log_2 P_i) \,, \tag{1.1}$$

por lo tanto, el mensaje puede ser comprimido hasta $n \cdot H$ bits.

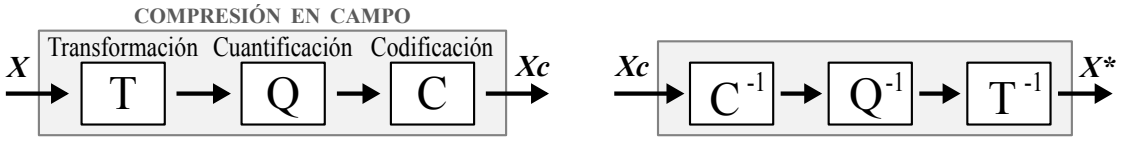
Existen diferentes métodos para llevar a cabo la compresión, basándose todos ellos en el mismo principio: reducir la redundancia existente en el conjunto de datos de entrada. La redundancia R se define como la diferencia entre el número de bits Hc usados en la representación original y la entropía [7].

$$R = H_c - H = H_c + \sum_{i=1}^{n} (P_i \cdot log_2 P_i) \,. \tag{1.2}$$

Dado que la redundancia depende del tipo de datos (texto, audio, imagen, etc.), no existe un algoritmo de compresión universal que resulte ser eficiente al momento de comprimir cualquier clase de dato, por lo tanto, es necesario desarrollar un método de compresión con el que se obtenga el mejor desempeño para un tipo de datos específico [7]. Para el caso de las datos sísmicos, la información corresponde a una representación de la imagen del subsuelo, razón por la cual se han empleado técnicas similares a la compresión de imágenes para llevar a cabo su compresión. Sin embargo, la compresión de datos sísmicos presenta un reto diferente por la presencia de muchas discontinuidades en las imágenes del subsuelo [15].

La Figura 1 muestra los procesos usados comúnmente para comprimir y descomprimir datos sísmicos [3, 17, 18]. En la primera etapa de la compresión se transforman los

Figura 1: Esquema de compresión–descompresión. Adaptada de [18].



(a) Compresión. (b) Descompresión.

datos originales a una representación en la cual se concentre la información más relevante. Con esta transformación se busca reducir la entropía y así aumentar la redundancia que puede ser eliminada.

En la siguiente etapa se representan los datos transformados mediante un conjunto finito de símbolos enteros empleando un proceso de cuantificación. Esto conlleva la introducción de cierta cantidad de error en los datos, por lo tanto, esta etapa debe ser omitida en los algoritmos de compresión sin pérdidas.

En la última etapa se lleva a cabo un proceso de codificación para disminuir la cantidad de bits que se deben emplear para representar el conjunto de datos. Por otro lado, la descompresión se logra realizando en orden inverso los procesos inversos de cada etapa(Ver Figura 1b).

En un algoritmo de compresión sin pérdidas, la información recuperada al momento de la descompresión será la misma que la original. Por otro lado, en la compresión con pérdidas se descarta aquella información que se considera no relevante con el fin de alcanzar compresiones más altas, siendo entonces la información recuperada distinta a la original.

Las métricas más usadas para medir la calidad de la compresión son la relación de compresión y la relación señal a ruido (CR y SNR respectivamente, por sus siglas en inglés) [5], las cuales se calculan usando las siguientes ecuaciones:

$$CR = \frac{\text{Tamaño de los datos sin comprimir}}{\text{Tamaño de los datos comprimidos}} , \tag{1.3}$$

$$SNR = 10 \cdot Log_{10} \left(\frac{\sum_n |\boldsymbol{X}_n|^2}{\sum_n |\boldsymbol{X}_n^* - \boldsymbol{X}_n|^2} \right) , \tag{1.4}$$

siendo $\boldsymbol{X}$ el conjunto de datos sin comprimir y $\boldsymbol{X}^*$ el conjunto de datos recuperado después de la descompresión.

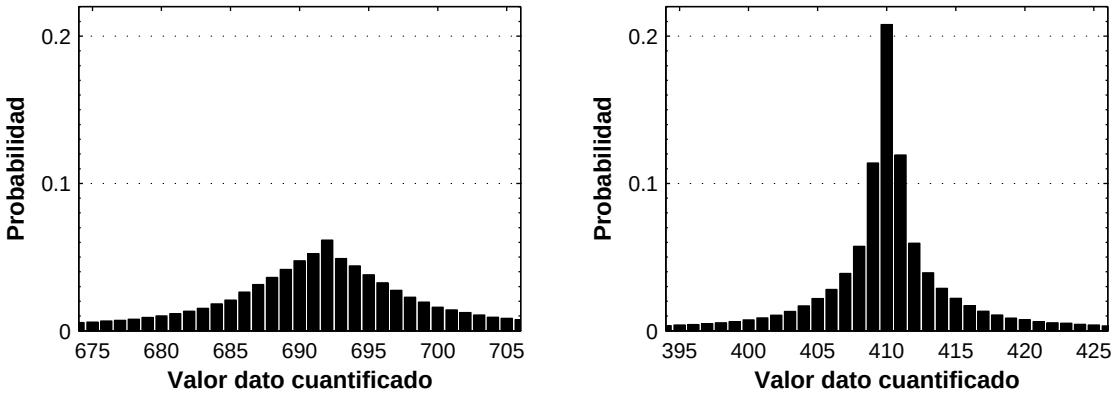
Las investigaciones en torno a las técnicas de compresión de datos sísmicos se han encaminado para obtener altos valores de CR con $SNR \geq 40dB$ [3, 17].

1.1. Transformación

La finalidad de la etapa de transformación en un algoritmo de compresión es representar la información de forma tal que permita obtener mejores resultados en las siguientes etapas de la compresión. La Transformación Wavelet Discreta (*DWT* por sus siglas en inglés), y en especial la familia de filtros *Cohen–Daubechies–Feauveau* (*CDF*), han sido usadas exitosamente para comprimir datos sísmicos en virtud de su análisis multirresolución en tiempo y frecuencia [4, 5, 19].

Al aplicar la *DWT* a un conjunto de datos, su energía se concentra en pocos coeficientes, lo cual ayuda a reducir su entropía. La Figura 2 muestra el efecto de la transformación en un conjunto de datos sísmicos compuesto por 344064 muestras. El rango de valores de los datos en la figura ha sido restringido con fines ilustrativos. El histograma de la Figura 2a corresponde al conjunto de datos luego de ser cuantificados usando 12 bits, mientras que el histograma de la Figura 2b corresponde al conjunto de datos que fueron cuantificados luego de ser transformados usando el filtro *CDF 2.2*.

Figura 2: Histograma de un conjunto de datos cuantificados.



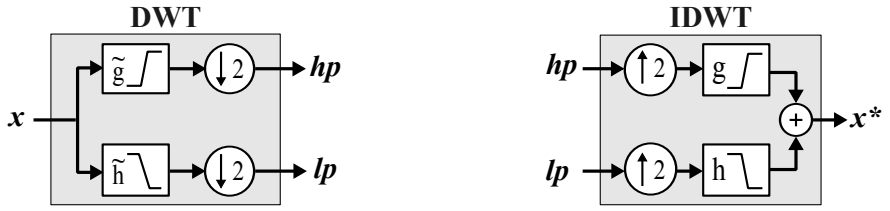
(a) Sin transformación

(b) Con transformación

En los histogramas se observa que al emplear la transformación se tiene una distribución más compacta, concentrando probabilidades más altas en menos datos. La entropía de los datos que fueron sido transformados es 4,71 bits por símbolo, mientras que la entropía de los datos que no fueron transformados es 6,31 bits por símbolo. Por lo tanto, al momento de comprimir el conjunto de datos transformados se podrá obtener un valor de CR más alto debido a que estos presentan más datos redundantes que pueden ser eliminados.

En la Figura 3a se muestra el método convencional para llevar a cabo la *DWT 1D*. Primero se realiza una convolución de la señal de entrada $\boldsymbol{x}$ con los filtros de análisis $\tilde{h}$ y $\tilde{g}$ y luego se hace una operación de diezmado. De esta manera, la transformación descompone la señal de entrada $\boldsymbol{x}$ en coeficientes de aproximación ($\boldsymbol{lp}$) y coeficientes de detalles ($\boldsymbol{hp}$) correspondientes a los componentes de baja frecuencia y alta frecuencia respectivamente.

Figura 3: Transformación Wavelet Discreta 1D. Adaptada de [20]



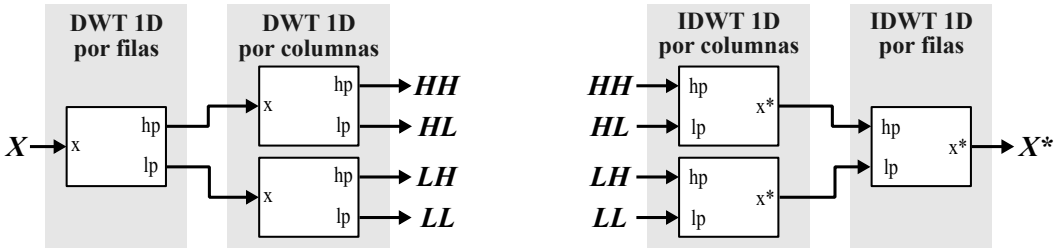
(a) Descomposición (b) Reconstrucción

En la Transformación Wavelet Discreta Inversa (*IDWT*) se aplican los filtros de síntesis h y g luego de interpolar los coeficientes $\boldsymbol{lp}$ y $\boldsymbol{hp}$ para lograr la reconstrucción de la señal (ver Figura 3b).

En el caso de señales bidimensionales se emplea la *DWT 2D* para descomponer la señal de entrada $\boldsymbol{X}$ en cuatro componentes: coeficientes de aproximación $\boldsymbol{LL}$, coeficientes de detalles horizontales $\boldsymbol{LH}$, coeficientes de detalles verticales $\boldsymbol{HL}$ y coeficientes de detalles diagonales $\boldsymbol{HH}$ [21]. La *DWT 2D* de una señal de N filas y M columnas puede ser implementada de la siguiente forma (ver Figura 4):

1. Realizar la *DWT 1D* a la señal de entrada en sentido horizontal o por filas. Como resultado se obtienen dos matrices de N filas y $\frac{M}{2}$ columnas.
2. Realizar la *DWT 1D* a las dos matrices anteriores pero esta vez en sentido vertical o por columnas. Como resultado se obtienen cuatro matrices de $\frac{N}{2}$ filas y $\frac{M}{2}$ columnas correspondientes a los coeficientes $\boldsymbol{LL}$, $\boldsymbol{LH}$, $\boldsymbol{HL}$ y $\boldsymbol{HH}$.

Figura 4: Transformación Waveled Discreta 2D.



(a) Descomposición (b) Reconstrucción

1.2. Cuantificación

La cuantificación puede ser considerada como una técnica de compresión con pérdidas ya que agrega ruido a la señal al reducir el número de bits usados para representar cada dato. En la etapa de cuantificación los coeficientes resultantes de la transformación son aproximados a un conjunto finito de valores enteros reduciendo la precisión de los

datos. Este proceso implica dividir el rango de los datos de entrada en un número finito de intervalos, asignar un símbolo a cada uno de estos intervalos y asociar a los datos dentro de cada intervalo el símbolo asignado a dicho intervalo.

La Cuantificación Uniforme es usada ampliamente en la compresión de datos sísmicos debido a que ofrece mejores resultados que otro tipo de cuantificación en términos de SNR [4, 5, 18, 22]. La Cuantificación Uniforme de un conjunto de datos $\boldsymbol{x}$, usando n bits para representar cada dato, puede ser llevada a cabo de la siguiente manera:

$$\begin{aligned} \boldsymbol{x}_{\text{min}} &= \boldsymbol{x} - \text{mínimo}\left(\boldsymbol{x}\right) \\ \boldsymbol{x}_{\text{cuantificado}} &= \text{round}\left[\frac{2^n - 1}{\text{máximo}\left(\boldsymbol{x}_{\text{min}}\right)} \cdot \boldsymbol{x}_{\text{min}}\right] . \end{aligned} \tag{1.5}$$

El proceso inverso se realiza de acuerdo a la siguiente expresión:

$$\boldsymbol{x}^* = \boldsymbol{x}_{\text{cuantificado}} \cdot \frac{\text{máximo}\left(\boldsymbol{x}_{\text{min}}\right)}{2^n - 1} + \text{mínimo}(\boldsymbol{x}) . \tag{1.6}$$

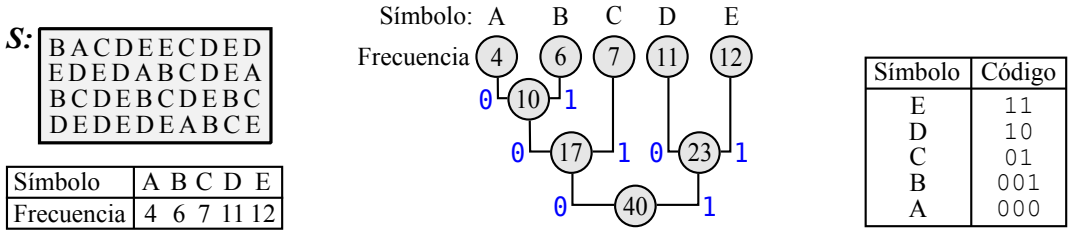
1.3. Codificación

El objetivo de la etapa de codificación es reducir el número de bits necesarios para representar todo el conjunto de símbolos. La Codificación Huffman [15, 17, 23, 24] y la Codificación Aritmética [4, 10, 25, 26] son los esquemas de codificación más usados en la compresión de datos sísmicos. Ambos presentan desempeño similar en términos de la relación de compresión que se puede lograr cuando la SNR es mayor que 40 dB [18], sin embargo, es preferible emplear la Codificación Huffman ya que su complejidad computacional es menor.

En la Codificación Huffman los símbolos son mapeados de acuerdo a un diccionario con códigos de longitud variable. La longitud del código asignado a cada símbolo dependerá de la cantidad de veces que aparezca el símbolo dentro del conjunto de datos, teniendo los símbolos más frecuentes un código más corto que aquellos símbolos poco frecuentes. De esta forma se logra reducir el número de bits necesarios para representar el conjunto de símbolos.

El diccionario resulta de la creación de un árbol binario de acuerdo a la frecuencia de aparición de los símbolos. Aquí, la raíz del árbol constituye el nodo final del proceso, mientras que las hojas correspondientes a cada uno de los símbolos diferentes constituyen los nodos iniciales. Para construir el árbol, los dos nodos menos frecuentes se unen repetidamente para formar un nodo de mayor frecuencia hasta tener un solo nodo. Los códigos correspondientes a cada símbolo se obtienen al asignar a cada ramificación el valor `0` o `1` empezando desde la raíz hasta llegar a cada hoja. La Figura 5 muestra un ejemplo de creación del diccionario Huffman para una señal $\boldsymbol{S}$ de 40 datos.

Figura 5: Creación del diccionario Huffman.



(a) Datos a codificar (b) Arbol binario (c) Diccionario

El proceso inverso, es decir, la decodificación, se realiza comparando los bits correspondientes a los datos codificados con los códigos de longitud variable almacenados en el diccionario Huffman. La longitud variable de los códigos hace de la decodificación un proceso altamente secuencial y que debe ser desarrollado a nivel de bits.

1.4. Trabajos relacionados

A continuación hacemos una revisión acerca de los trabajos de investigación relacionados con la compresión de datos, enfocándonos en los tipos de transformaciones que han sido utilizadas en la compresión de datos sísmicos.

Uno de los primeros trabajos publicados en el tema de la compresión de datos sísmicos fue el desarrollado por *Wood* en 1974 [27]. Su algoritmo se basó en la Transformación Walsh y un método en el dominio de tiempo para reducir la redundancia de la información sísmica.

A mediados de la década de 1980, *Joint Photographic Experts Group* desarrolló el método de compresión JPEG. Este método está basado en la Transformación del coseno y tuvo un gran auge especialmente en la compresión de imágenes.

A principios de 1990, Spanias, Jonsson y Stearns presentaron un estudio sobre la compresión de datos sísmicos [28]. Sus algoritmos se basaron en cuatro esquemas de transformación diferentes: la Transformación Discreta de Fourier (DFT), la Transformación Discreta del Coseno (DCT), la Transformación Walsh-Hadamard (WHT) y la Transformación Karhunen-Loeve (KLT). El mejor rendimiento se obtuvo mediante el uso de la KLT. Sin embargo, la base de esta transformación es dependiente de los datos, lo cual implica un alto costo computacional.

Fue en la década de 1990 cuando los algoritmos de compresión basados en la Transformación Wavelet [29] se volvieron populares.

En 1993 Bosman y Reiter [30] desarrollaron el primer algoritmo de compresión de datos sísmicos basado en la Transformación Wavelet. Sus resultaron evidenciaron que esta transformación ofrecía un mejor desempeño en la compresión al ser comparada con otro tipo de transformaciones tradicionales (e.g. Fourier, Coseno).

Más adelante, en 1996, Villaseñor desarrolló un algoritmo de compresión basado en la Transformación Wavelet 3D [17]. En este algoritmo se empleó el filtro biorthogonal 9-7 por su rendimiento en términos de relación de compresión *vs.* relación señal a ruido.

En el año 2000, *Joint Photographic Experts Group* creó el estándar JPEG 2000 [31]. En este estándar, la Transformación Wavelet reemplazó a la Transformación del coseno, lo cual mejoró considerablemente el rendimiento en términos de relación de compresión.

En la década de 2000, una nueva serie de transformaciones y esquemas de codificación se utilizaron con éxito en la compresión de datos sísmicos.

En 2004, Røsten et al. presentaron un algoritmo de compresión basado en un banco filtros [32], el cual ofreció mejores resultados que otros algoritmos basados en transformaciones de tiempo–frecuencia.

En el año 2012, Reddy et al. [33] utilizaron Análisis de Componentes Principales (*PCA*) para comprimir datos sísmicos. Los componentes principales fueron calculados usando una estrategia basada en redes neuronales artificiales.

En 2014, Wenmao desarrolló un esquema de compresión–descompresion de datos sísmicos basado en la transformación *SPITH* (*Set Positioning in Hierarchical Tree*) [34]. Los resultados obtenidos al emplear esta transformación fueron comparados con trabajos anteriores y se evidenciaron mejores resultados. Lamentablemente, el algoritmo implementado ofrece un alto costo computacional.

2. SELECCIÓN DE LA ESTRATEGIA DE TRANSFORMACIÓN

En el presente capítulo se exponen los criterios tenidos en cuenta para escoger la estrategia de transformación a emplear en el proceso de compresión–descompresión de los datos sísmicos. Las pruebas se llevaron a cabo empleando Cuantización Uniforme y Codificación Huffman para las demás etapas del proceso de compresión.

2.1. Esquema *Lifting*

La *DWT* convencional (es decir, aquella basada en la convolución) es ineficiente desde el punto de vista computacional. Esto debido a que primero se filtran dos veces todas las muestras de la señal de entrada (una vez con el filtro $\tilde{h}$ y otra con $\tilde{g}$) y luego se descartan la mitad de los resultados. De esta manera se obtienen los coeficientes ***lp*** y ***hp***, cada uno de ellos con tamaño igual a la mitad de los datos de la señal de entrada.

A mediados de la decada de los 90, W. Swelden propuso un método alternativo para calcular la *DWT* denominado *esquema lifting* [35]. En este método, las operaciones de convolución son reemplazadas por multiplicaciones con polinomios de Laurent, lo cual mejora el desempeño computacional de los algoritmos usados en el cálculo de la *DWT* [20, 36]. Entre las ventajas que ofrece el esquema *lifting* frente a la *DWT* convencional [36–38] se pueden destacar:

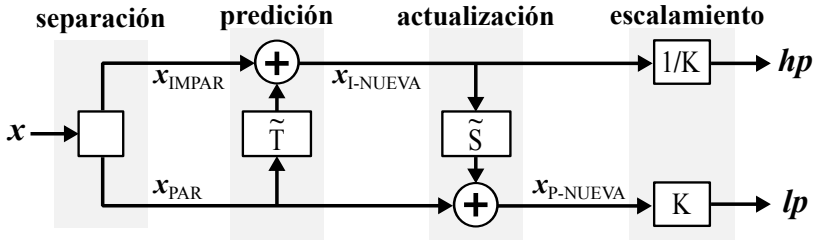
- Eficiencia computacional: es más rápido ya que la cantidad de operaciones matemáticas a realizar es menor.
- Uso de memoria: no necesita de un espacio de almacenamiento adicional para guardar resultados intermedios.
- Transformación Wavelet "*integer to integer*": es posible emplear una función de redondeo y lograr la reconstrucción perfecta si los datos de entrada están representados en formato entero.

Todas estas características hacen que el esquema *lifting* sea atractivo para emplear en aplicaciones donde la velocidad de procesamiento y el almacenamiento de datos son factores críticos.

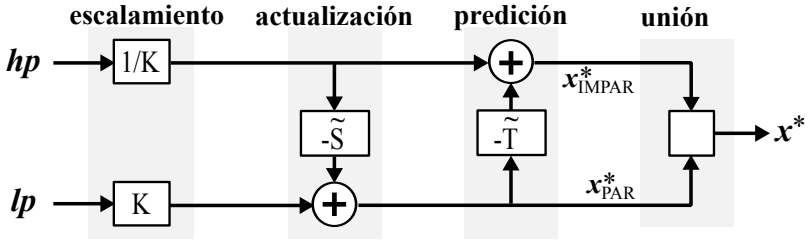
El esquema *lifting* es representado en la Figura 6. Los coeficientes ***lp*** y ***hp*** son calculados de acuerdo a los siguientes pasos:

1. *Separación*: La señal de entrada $\boldsymbol{x}$ es separada en muestras pares e impares. Esta operación también es conocida como *Lazy Wavelet Transform*.
2. *Predicción*: Se predicen las nuevas muestras impares mediante una combinación lineal que también involucra las muestras pares y el polinomio de predicción $\tilde{T}$. Esta operación tambien es conocida como *Dual lifting step*.
3. *Actualización*: Se actualizan las muestras pares mediante una combinación lineal que también involucra las nuevas muestras impares y el polinomio de actualización $\tilde{S}$. Esta operación también es conocida como *Primal lifting step*.
4. *Escalamiento*: Las nuevas muestras pares son multiplicadas por el factor de normalización K mientras que las nuevas muestras impares son divididas por dicho factor de normalización. Este paso debe omitirse en la Transformación Wavelet "*integer to integer*" [21].

Figura 6: Esquema *lifting* para la *DWT 1D*. Adaptada de [37]



(a) Descomposición



(b) Reconstrucción

Tal como se muestra en la Figura 6b, la *IDWT* se lleva a cabo invirtiendo el orden de los pasos de la *DWT* pero con las siguientes modificaciones:

- Intercambiar K y $1/K$ en el paso de *escalamiento*.
- Cambiar los signos de los polinomios $\tilde{S}$ y $\tilde{T}$ en los pasos de *actualización* y *predicción*.
- Reemplazar la operación de *separación* por una en la cual se unan las muestras de $\boldsymbol{x}^*_{PAR}$ y $\boldsymbol{x}^*_{IMPAR}$ para obtener $\boldsymbol{x}^*$.

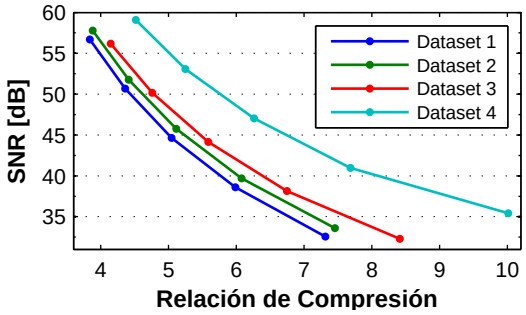
2.2. Comparación filtros Wavelet

Con el fin de establecer un punto de referencia, en primera instancia se realizó la compresión sin llevar a cabo la etapa de transformación. En la Tabla 1 se muestran los resultados obtenidos para un *dataset* o conjunto de datos sísmicos de 96 trazas con 3584 muestras cada una. Allí se observa que al emplear más bits en la cuantificación la CR disminuye mientras que la SNR aumenta. Este comportamiento se presenta para diferentes *datasets* tal como se muestra en la Figura 7. Por lo tanto, existe un compromiso para seleccionar el número de bits que se emplearán en la cuantificación.

Tabla 1: Resultados de la compresión sin transformación del *dataset 1*.

# BITS	CR	SNR [dB]
10	7.32	32.56
11	5.99	38.60
12	5.05	44.65
13	4.36	50.67
14	3.84	56.70

Figura 7: SNR vs. CR para compresión sin transformación.

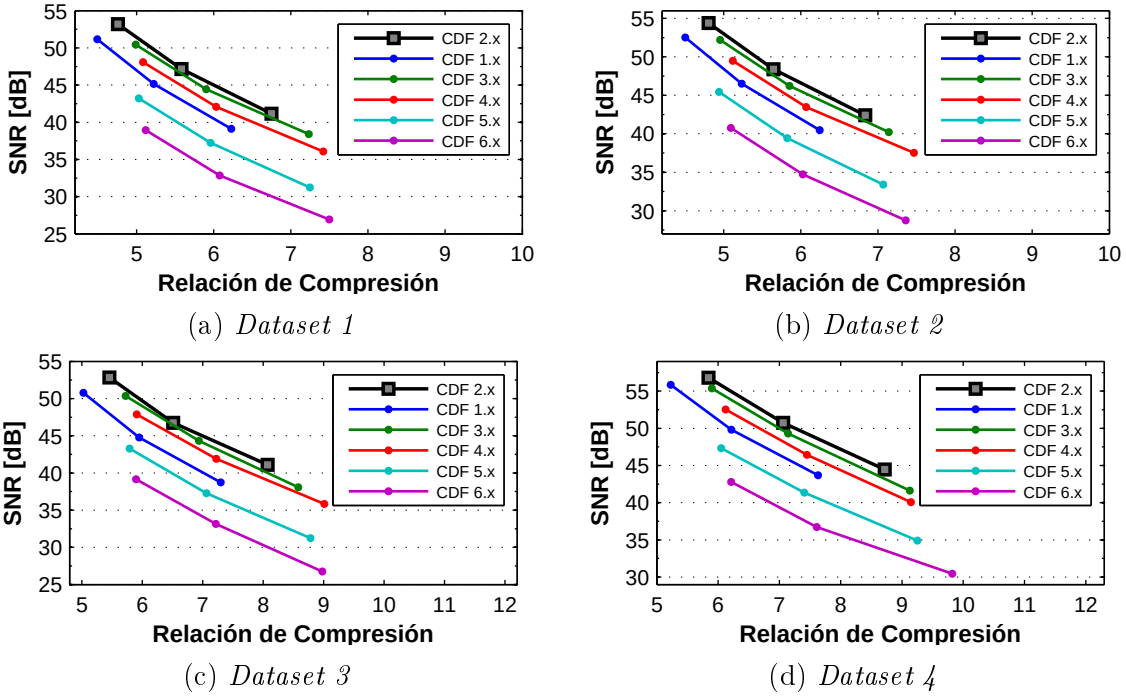


Posteriormente se realizó la compresión de los *datasets* incluyendo la etapa de transformación. En esta prueba se usaron entre 12 y 14 bits de cuantificación con el fin de obtener $SNR \geq 40dB$ y las siguientes familias de filtros Wavelet:

- *CDF 1.x*, *CDF 3.x* y *CDF 5.x* con $x = 1$, 3 y 5
- *CDF 2.x*, *CDF 4.x* y *CDF 6.x* con $x = 2$, 4 y 6
- *DB 2*, *DB 3*, *DB 4*, *DB 5* y *DB 6*
- *SYM 4*, *SYM 5*, *SYM 6* y *SYM 7*

Los resultados de esta prueba arrojaron un comportamiento similar para cada familia de filtros. Las Figuras 8 a 10 muestran los resultados obtenidos. Observe que en todos los casos la familia de filtros *CDF 2.x* ofrece los mejores resultados ya que con estos filtros se tiene la SNR más alta para determinada CR.

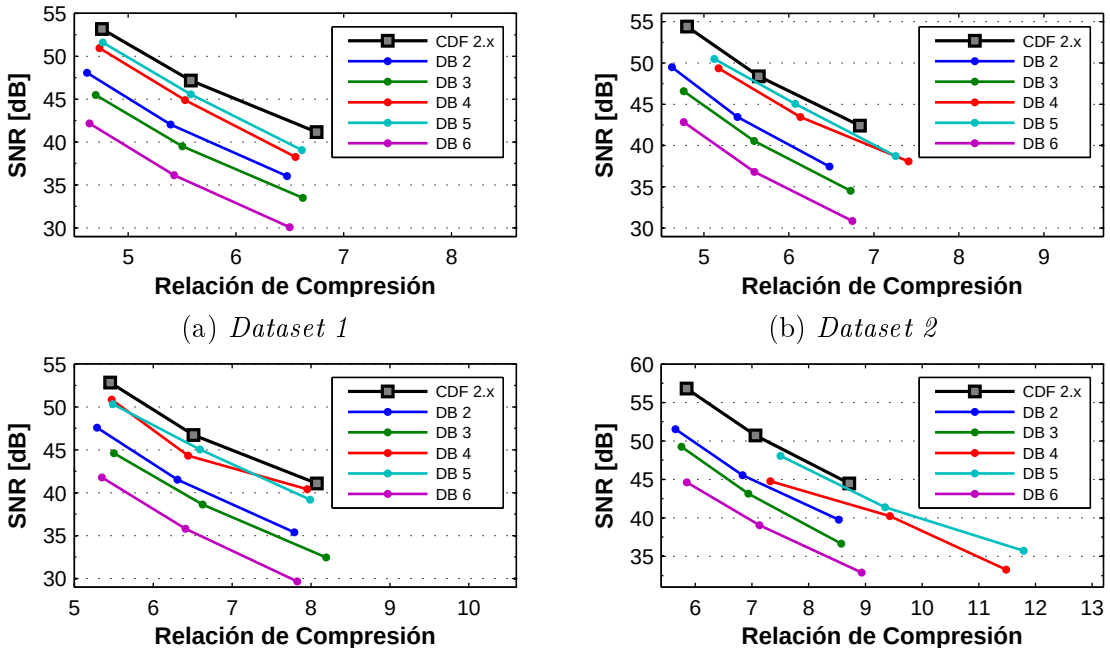
Figura 8: Compresión de datos sísmicos con filtros Wavelet *CFD*.



(a) *Dataset 1* (b) *Dataset 2*

(c) *Dataset 3* (d) *Dataset 4*

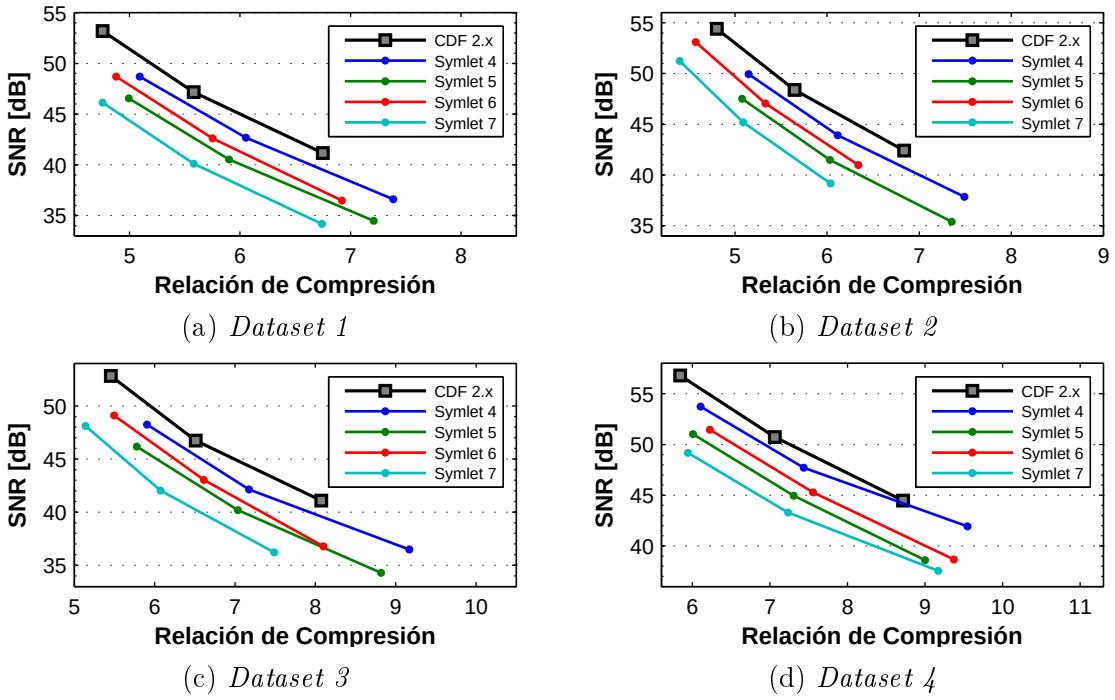
Figura 9: Compresión de datos sísmicos con filtros Wavelet *Daubechies*.



(a) *Dataset 1* (b) *Dataset 2*

(c) *Dataset 3* (d) *Dataset 4*

Figura 10: Compresión de datos sísmicos con filtros Wavelet *Symlet*.



(a) *Dataset 1* (b) *Dataset 2*

(c) *Dataset 3* (d) *Dataset 4*

2.3. Polinomios del esquema *lifting*

Los polinomios de predicción ($\tilde{T}$) y actualización ($\tilde{S}$) del esquema *lifting* son obtenidos a partir de la factorización de la matriz polifase $\tilde{P}$ de los filtros Wavelet $\tilde{h}$ y $\tilde{g}$ [21]. La matriz polifase de los filtros de análisis $\tilde{h}$ y $\tilde{g}$ está dada por:

$$\tilde{P} = \begin{bmatrix} \tilde{H}_e & \tilde{H}_o \\ \tilde{G}_e & \tilde{G}_o \end{bmatrix} , \tag{2.1}$$

siendo $\tilde{H}_e$ y $\tilde{H}_o$ los componentes par e impar del filtro pasa-bajas $\tilde{h}$, mientras que $\tilde{G}_e$ y $\tilde{G}_o$ son los componentes par e impar del filtro pasa-altas $\tilde{g}$.

Está demostrado que cualquier par de filtros complementarios $\tilde{h}$–$\tilde{g}$ pueden ser factorizados a un conjunto de matrices triangulares y una matriz diagonal [20, 37]. La matriz polifase $\tilde{P}$ puede reescribirse como

$$\tilde{P} = \begin{bmatrix} \tilde{H}_e & \tilde{H}_o \\ \tilde{G}_e & \tilde{G}_o \end{bmatrix} = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{m/2} \left(\begin{bmatrix} 1 & \tilde{S}_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{T}_i & 1 \end{bmatrix} \right) , \tag{2.2}$$

donde $\tilde{S}_i$ y $\tilde{T}_i$ son polinomios de Laurent, m corresponde al número de matrices triangulares en las que se puede factorizar $\tilde{P}$ y K es un factor de normalización.

Teniendo en cuenta lo anterior, la descomposición Wavelet de una entrada $\boldsymbol{x}$ puede expresarse como sigue:

$$\begin{bmatrix} LP \\ HP \end{bmatrix} = \tilde{P} \begin{bmatrix} X_e \\ X_o \end{bmatrix} = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{m/2} \left(\begin{bmatrix} 1 & \tilde{S}_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{T}_i & 1 \end{bmatrix} \right) \begin{bmatrix} X_e \\ X_o \end{bmatrix} , \tag{2.3}$$

donde X_e y X_o son los componentes par e impar de la entrada $\boldsymbol{X}$, LP son los coeficientes de aproximación y HP son los coeficientes de detalle.

En las ecuaciones (2.5) son presentados los polinomios de actualización y predicción correspondientes los filtros de análisis $\tilde{g}$ y $\tilde{h}$ de la familia de filtros Wavelet *CDF 2.x* mostrados en las ecuaciones (2.4) [38].

Filtros de análisis

$$\textbf{\textit{CDF 2.x:}} \quad \tilde{g} = \frac{\sqrt{2}}{4}\left(z^{-1} - 2 + z^{1}\right) , \tag{2.4a}$$

$$\textbf{\textit{CDF 2.2:}} \quad \tilde{h} = \frac{\sqrt{2}}{8}\left(-z^{-2} + 2z^{-1} + 6 + 2z^{1} - z^{2}\right) , \tag{2.4b}$$

$$\begin{aligned} \textbf{\textit{CDF 2.4:}} \quad \tilde{h} = \frac{\sqrt{2}}{128}\Big(&3z^{-4} - 6z^{-3} - 16z^{-2} + 38z^{-1} \\ &+ 90 + 38z^{1} - 16z^{2} - 6z^{3} + 3z^{4}\Big), \end{aligned} \tag{2.4c}$$

$$\begin{aligned} \textbf{\textit{CDF 2.6:}} \quad \tilde{h} = \frac{\sqrt{2}}{1024}\Big(&-5z^{-6} + 10z^{-5} + 34z^{-4} - 78z^{-3} - 123z^{-2} + 324z^{-1} \\ &+ 700 + 324z^{1} - 123z^{2} - 78z^{3} + 34z^{4} + 10z^{5} - 5z^{6}\Big) . \end{aligned} \tag{2.4d}$$

Polinomios *lifting*

$$\textbf{\textit{CDF 2.x:}} \quad \tilde{T}_{2.x} = -\frac{1}{2}\left(1 + z^{1}\right) , \tag{2.5a}$$

$$\textbf{\textit{CDF 2.2:}} \quad \tilde{S}_{2,2} = \frac{1}{4}\left(z^{-1} + 1\right) , \tag{2.5b}$$

$$\textbf{\textit{CDF 2.4:}} \quad \tilde{S}_{2,4} = \frac{1}{64}\left(-3z^{-2} + 19z^{-1} + 19 - 3z^{1}\right) , \tag{2.5c}$$

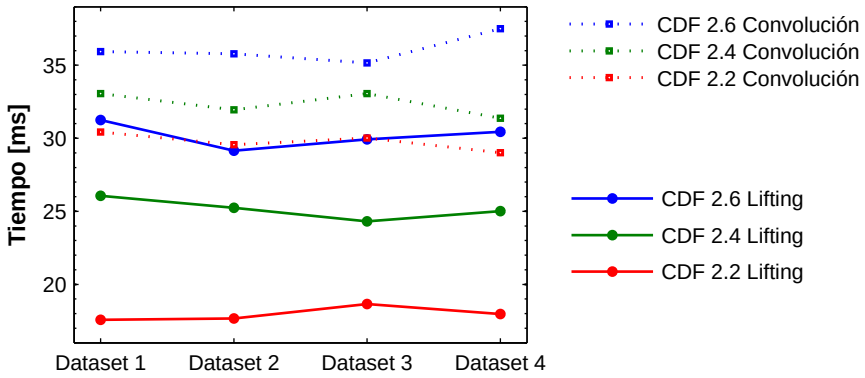
$$\textbf{\textit{CDF 2.6:}} \quad \tilde{S}_{2,6} = \frac{1}{512}\left(5z^{-3} - 39z^{-2} + 162z^{-1} + 162 - 39z^{1} + 5z^{2}\right) . \tag{2.5d}$$

Los tres filtros comparten el polinomio de predicción $\tilde{T}_{2.x}$ mostrado en la ecuación (2.5a). Observe que el filtro *CDF 2.2* es el menos costoso desde el punto de vista

computacional, ya que su polinomio de actualización ($\tilde{S}_{2,2}$) involucra menos operaciones de suma. Adicionalmente, los polinomios $\tilde{T}_{2.x}$ y $\tilde{S}_{2,2}$ solo involucran divisiones con 2 y 4, las cuales puede hacerse mediante corrimiento de los datos.

La Figura 11 muestra el tiempo necesario para ejecutar en una CPU un algoritmo de transformación Wavelet inversa aplicado a diferentes *datasets*. Observe que en todos los casos, el esquema *lifting* usando el filtro *CDF 2.2* se ejecuta en menos tiempo.

Figura 11: Tiempo de ejecución de la *IDWT* en CPU.



Por lo tanto, el esquema *lifting* y el filtro Wavelet *CDF 2.2* fueron seleccionados para la estrategia de transformación en el proceso de compresión de los datos sísmicos y su posterior descompresión en FPGA y GPU. A continuación se resumen las ventajas que ofrece esta selección:

- La familia de filtros Wavelet *CDF 2.x* ofrece mejor relación CR vs. SNR que los demás filtros analizados.
- Por medio del esquema *lifting*, los algoritmos para realizar la DWT se ejecutan más rápido y requieren menos capacidad de memoria que por el método convolucional.
- Los polinomios del esquema *lifting* correspondientes al filtro *CDF 2.2* implican menos sumas que los polinomios de los filtros *CDF 2.4* y *CDF 2.6*. Adicionalmente, al tener coeficientes que son potencias de 2, las divisiones pueden implementarse mediante corrimiento de los datos.
- Es posible usar la Transformación Wavelet "*integer to integer*" ya que los *datasets* suministrados por ECOPETROL corresponden a muestras representadas con números enteros. En este caso, la función de redondeo puede implementarse descartando los bits menos significativos que resultan de las divisiones.

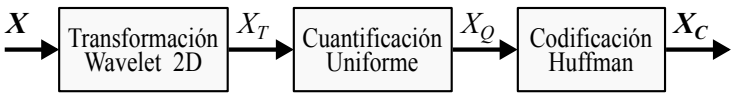
3. DESCOMPRESIÓN EN FPGA

En este capítulo se presenta la arquitectura computacional diseñada para descomprimir datos sísmicos en un FPGA. El capítulo inicia con una descripción del proceso de compresión empleado. Este proceso fue desarrollado en una arquitectura CPU teniendo en cuenta que la compresión de los datos sísmicos generalmente es desarrollada en este tipo de arquitectura mientras se realiza la adquisición. La segunda sección del capítulo describe la arquitectura computacional empleada para la descompresión en el FPGA, mientras que la tercera sección presenta los resultados obtenidos.

3.1. Compresión

La Figura 12 muestra el proceso usado para comprimir los datos, el cual está compuesto por la *DWT 2D*, la Cuantificación uniforme y la Codificación Huffman.

Figura 12: Esquema del algoritmo de compresión en CPU.



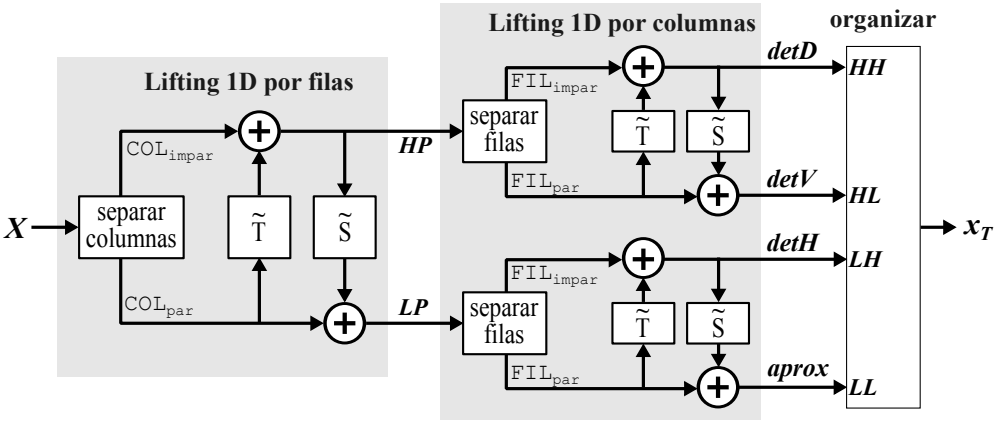
La primera etapa de la compresión se realizó mediante la transformación bidimensional teniendo en cuenta que un conjunto de datos sísmicos se puede representar como una matriz de dos dimensiones $\boldsymbol{X}$. De esta manera, cada columna de la matriz $\boldsymbol{X}$ tendrá una traza sísmica del conjunto de datos sísmicos. La *DWT 2D* se implementó mediante el esquema *lifting* usando el filtro *CDF2.2*. La Figura 13 muestra el esquema *lifting* para la transformación 2D, el cual se lleva a cabo de acuerdo a los siguientes pasos:

1. *Lifting* 1D por filas.
   - Separar la matriz de entrada $\boldsymbol{X}$ por columnas pares e impares. Para una entrada de N filas y M columnas se obtienen dos matrices con N filas y $\frac{M}{2}$ columnas.*
   - Hacer las operaciones de predicción y actualización a cada una de las filas de estas matrices usando los polinomios $\tilde{T}$ y $\tilde{S}$ correspondientes al filtro *CDF 2.2*, es decir,

$$\tilde{T} = -\frac{1}{2}\left(1 + z^1\right) \qquad \text{y} \qquad \tilde{S} = \frac{1}{4}\left(z^{-1} + 1\right). \tag{3.1}$$

* En el caso de M impar se tendrá una matriz con $\frac{M-1}{2}$ columnas y otra de $\frac{M+1}{2}$ columnas.

Figura 13: Esquema *lifting* para la *DWT 2D*. Adaptada de [21].



2. *Lifting* 1D por columnas.
   - Realizar la separación en filas pares e impares obteniendo 4 matrices con $\frac{N}{2}$ filas y $\frac{M}{2}$ columnas.
   - Hacer las operaciones de predicción y actualización pero ahora a cada una de las columnas de las matrices.
3. Organizar las matrices $\boldsymbol{aprox}$, $\boldsymbol{detH}$, $\boldsymbol{detV}$ y $\boldsymbol{detD}$ para formar el vector $\boldsymbol{x}_T$. Estas matrices corresponden a los coeficientes de aproximación y de detalles horizontal, vertical y diagonal, respectivamente.

La segunda etapa de la compresión se realizó mediante cuantificación uniforme empleando las expresiones

$$\begin{aligned} \boldsymbol{x}_{\min} &= \boldsymbol{x}_t - \text{mínimo}(\boldsymbol{x}_t) \\ \boldsymbol{x}_Q &= \text{round}\left[\frac{\boldsymbol{x}_{\min} \cdot (2^n - 1)}{\text{máximo}(\boldsymbol{x}_{\min})}\right], \end{aligned} \tag{3.2}$$

siendo n el número de bits empleados para representar cada coeficiente en $\boldsymbol{x}_T$. Esta ecuación puede reescribirse como se muestra a continuación:

$$\begin{aligned} K1 &= \frac{\text{máximo}(\boldsymbol{x}_{\min})}{(2^n - 1)} \\ K2 &= \text{mínimo}(\boldsymbol{x}_T) \\ \boldsymbol{x}_Q &= \text{round}\left[\frac{\boldsymbol{x}_T - K2}{K1}\right]. \end{aligned} \tag{3.3}$$

En la última etapa, los resultados de la codificación Huffman ($\boldsymbol{x}_C$) fueron almacenados en dos memorias. En la primera memoria se almacenaron los datos comprimidos.

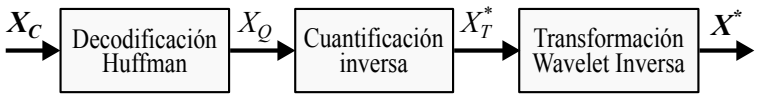
Esta memoria tiene solo un bit en cada posición. En la segunda memoria se almacenó el diccionario Huffman de la siguiente forma:

- El diccionario se organizó en orden descendente de acuerdo a la frecuencia de aparición de los símbolos. Con esto los códigos de los símbolos más frecuentes aparecen al principio del diccionario para que puedan ser encontrados en menos tiempo.
- Se agregó una bandera entre los códigos cada vez que la longitud de estos aumenta, para que así el decodificador identifique la cantidad de bits que debe comparar.
- Cada posición de memoria tiene 32 bits.
  - ▷ Los 12 bits menos significativos corresponden a la representación original de los símbolos.
  - ▷ Los 20 bits más significativos corresponden a los códigos Huffman. Los códigos de menos de 20 bits se rellenaron con el primer bit del código más corto.

3.2. Descompresión

En la Figura 14 se muestra el esquema usado para descomprimir un conjunto de datos sísmicos que ha sido comprimido con el esquema de la Figura 12.

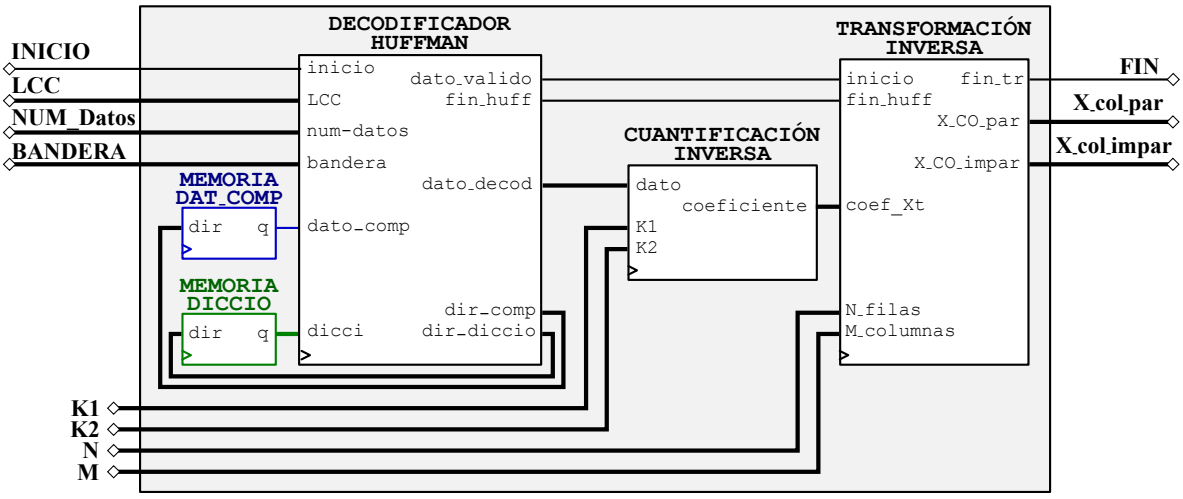
Figura 14: Esquema de descompresión empleado.



La arquitectura diseñada para realizar la descompresión en FPGA es mostrada en la Figura 15. Además de las memorias con el diccionario Huffman (MEMORIA_DICCIO) y los datos comprimidos (MEMORIA_DAT_COMP), el circuito de descompresión necesita los siguientes parámetros de compresión:

- NUM_DATOS: Número total de datos.
- M: Número de trazas.
- N: Número de muestras por trazas.
- LCC: Longitud del código más corto resultante de la codificación Huffman.
- BANDERA: Bandera de incremento de la longitud de los códigos en el diccionario Huffman.
- K1 y K2: Constantes resultantes de la etapa de cuantificación.

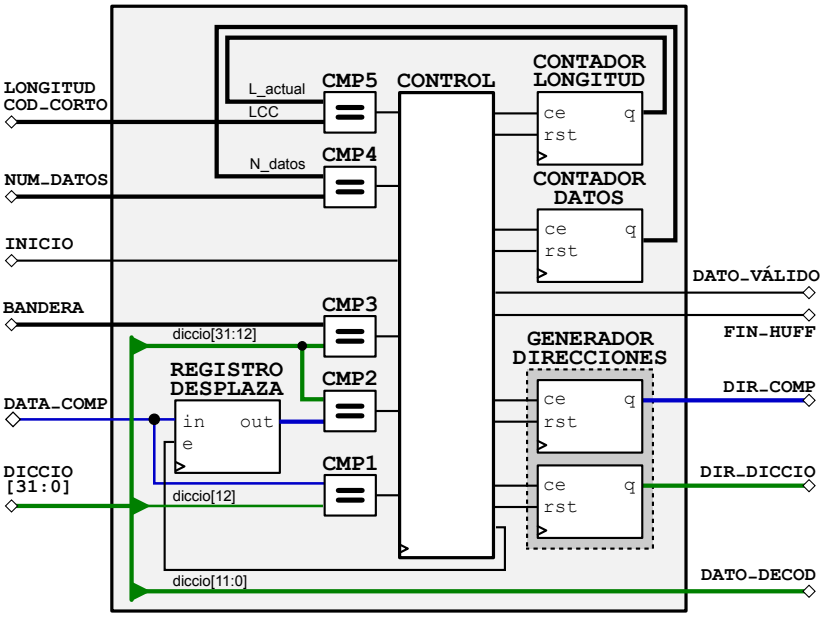
Figura 15: Diagrama general del circuito de descompresión en FPGA.



3.2.1. Decodificación en FPGA

La Figura 16 muestra la arquitectura empleada para llevar a cabo la decodificación. Este diseño está basado en nuestro trabajo anterior [39].

Figura 16: Circuito Decodificador Huffman.



El circuito CONTROL corresponde a una máquina de estados que gobierna el proceso de decodificación. La longitud efectiva de los códigos en el diccionario se determina por medio del comparador CMP3 y del circuito CONTADOR_LONGITUD. Este comparador activa su salida cuando encuentra una BANDERA de incremento en el diccionario, haciendo que CONTADOR_LONGITUD incremente su valor.

El circuito REGISTRO_DESPLAZA corresponde a un registro de desplazamiento que se encarga de concatenar un bit de los datos comprimidos cada vez que se active CMP3. Esta concatenación se hará hasta que el comparador CMP2 indique que el dato en REGISTRO_DESPLAZA es igual a uno de los códigos del diccionario. Cuando esto sucede, se activa la salida DATO_VÁLIDO para indicar que el dato presente en la salida DATO_DECOD corresponde a un dato que ha sido correctamente decodificado.

Cuando el código más corto solo tiene un bit (LCC=1), el circuito CONTROL verifica primero la salida del comparador CMP1 antes que la salida de CMP2. Con esto se logra que se necesite únicamente 1 ciclo de reloj para decodificar los códigos de un bit. Los códigos de más de un bit serán decodificados en al menos 5 ciclos de reloj cuando el código más corto tiene un bit.

Por otro lado, cuando el código más corto tiene más de un bit (LCC>1), CONTROL primero debe esperar que el comparador CMP5 le indique que REGISTRO_DESPLAZA ha almacenado LCC bits. Una vez realizado este almacenamiento, se da inicio al proceso de comparación. En este caso, el símbolo con el código más corto será decodificado en al menos LCC+3 ciclos de reloj, mientras que los otros códigos serán decodificados en al menos LCC+5 ciclos de reloj.

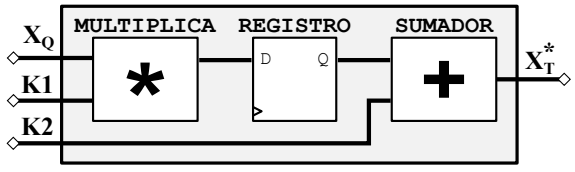
3.2.2. Cuantificación inversa en FPGA

Teniendo en cuenta las ecuaciones (1.6) y (3.3), esta etapa puede hacerse de acuerdo a la expresión

$$\boldsymbol{x}_T^* = (K1 \cdot \boldsymbol{x}_Q) + K2 \; . \tag{3.4}$$

La Figura 17 muestra el circuito diseñado para implementar la etapa de cuantificación inversa en el FPGA. El REGISTRO es utilizado para sincronizar el circuito y lograr su segmentación, lo cual permite aumentar su frecuencia máxima de operación.

Figura 17: Circuito Cuantificación inversa.



3.2.3. Transformación inversa en FPGA

La arquitectura mostrada en la Figura 18 permite llevar a cabo la *IDWT 2D* de acuerdo al esquema *lifting* de la Figura 19.

El circuito CONTROL consta de una máquina de estados que se encarga de habilitar cada etapa de la transformación 2D (*IDWT 1D* por columnas o por filas), así

Figura 18: Circuito general de la *IDWT 2D* mediante esquema *lifting*.

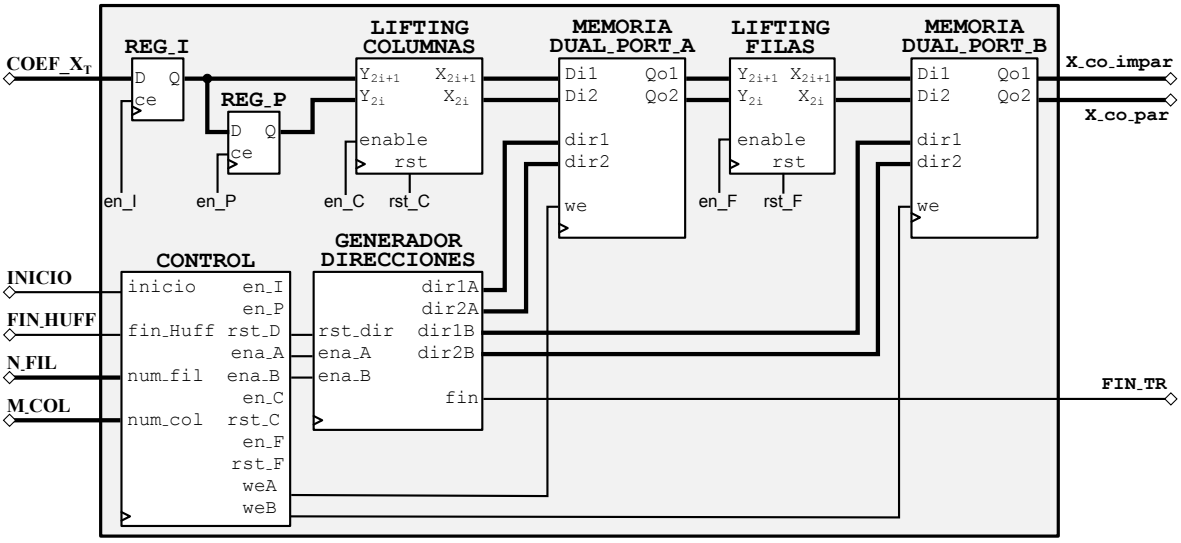
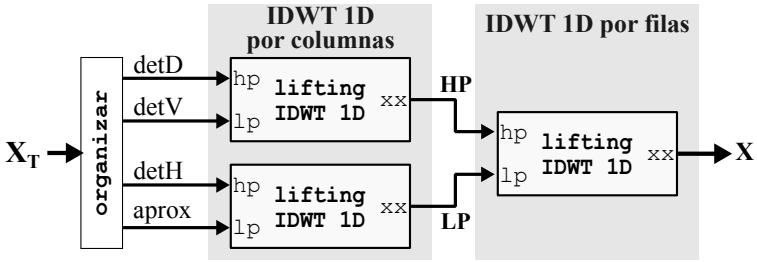
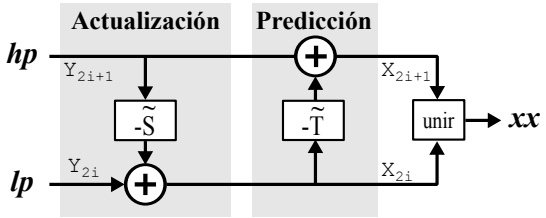


Figura 19: Esquema *lifting* para la *IDWT 2D*.



(a) Reconstrucción Wavelet 2D



(b) Esquema *lifting* para *IDWT 1D*

como también de habilitar los registros REG_I y REG_P. Adicionalmente, este circuito controla el almacenamiento de los resultados en memoria.

Los circuitos LIFTING_COLUMNAS y LIFTING_FILAS realizan la *IDWT 1D* de acuerdo al esquema *lifting* para la Transformación Wavelet "*integer to integer*" mostrado en la Figura 19b. Estos circuitos están basados en nuestro trabajo en [40] y fueron diseñados para procesar dos datos en cada ciclo de reloj, teniendo un retardo de 4 ciclos de reloj para entregar los primeros resultados. Por lo tanto, este circuito demora

en total $\left(\left\lceil \frac{1}{2}A \cdot B \right\rceil + 4\right)$ ciclos de reloj para reconstruir una matriz de tamaño $A \times B$. *

Los circuitos ACTUALIZACIÓN (Figura 20) y PREDICCIÓN (Figura 21) son los encargados de llevar a cabo las operaciones de actualización y predicción de la Figura 19b. Teniendo en cuenta los polinomios $\tilde{T}$ y $\tilde{S}$ mostrados en la ecuación (3.1), estas operaciones se realizan con las siguientes ecuaciones:

$$\text{Actualización:} \qquad \boldsymbol{X}_{2i} = \boldsymbol{Y}_{2i} - \left\lfloor \frac{1}{4}\left(\boldsymbol{Y}_{2i-1} + \boldsymbol{Y}_{2i+1}\right) \right\rfloor \tag{3.5a}$$

$$\text{Predicción:} \qquad \boldsymbol{X}_{2i+1} = \boldsymbol{Y}_{2i+1} + \left\lfloor \frac{1}{2}\left(\boldsymbol{X}_{2i} + \boldsymbol{X}_{2i+2}\right) \right\rfloor . \tag{3.5b}$$

Los desplazamientos en tiempo que aparecen en estas ecuaciones se logran con los registros REG_1 y REG_2. Las divisiones realizan mediante el corrimiento de los datos hacia la derecha descartando los bits menos significativos para redondear los resultados.

Figura 20: Circuito Actualización.

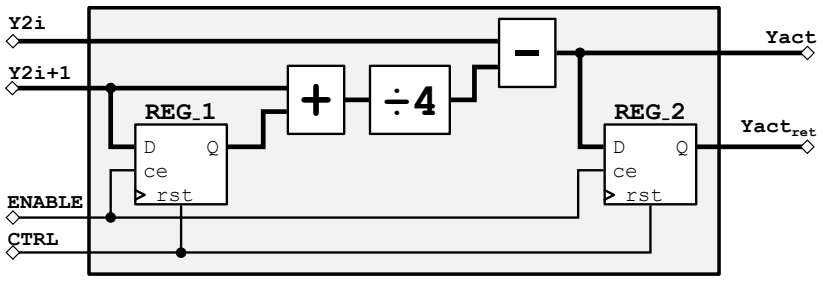
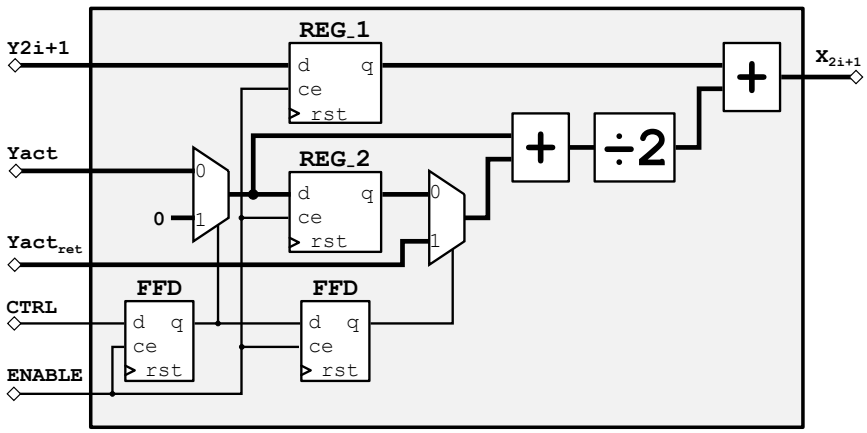


Figura 21: Circuito Predicción.



* $\lceil\cdot\rceil$ y $\lfloor\cdot\rfloor$ representan las funciones de redondeo hacia $+\infty$ y $-\infty$ respectivamente

3.3. Resultados

La arquitectura computacional expuesta en el presente capítulo fue desarrollada en el software ISE Design Suite 13.2 de Xilinx. El diseño fue implementado en el sistema de desarrollo ML507 el cual posee un FPGA Virtex5–XC5VFX70T. Teniendo en cuenta el reporte de implementación generado por ISE y la cantidad de bloques RAM disponibles, en este FPGA es posible descomprimir un conjunto de 65536 muestras con frecuencia máxima de operación de 85 MHz. Los resultados fueron capturados por medio de la herramienta Chipscope de Xilinx para verificar su validez al ser contrastados con los resultados de un *script* de descompresión en Matlab.

La tabla 2 resume los resultados obtenidos para las etapas de transformación y cuantificación inversa.

Tabla 2: Ciclos de reloj empleados en la transformación y la cuantificación.

Etapa	Tamaño datos	Ciclos de reloj
IDWT 1D por columnas	$N \times \left\lceil \frac{M}{2} \right\rceil$	$\frac{1}{2} \cdot N \cdot \left\lceil \frac{M}{2} \right\rceil + 4$
IDWT 1D por filas	$N \times M$	$\frac{1}{2} \cdot N \cdot M + 4$
Cuantificación inversa	$N \times M$	$N \cdot M$

La arquitectura propuesta para la transformación inversa 1D genera dos datos reconstruidos en cada ciclo de reloj con latencia de 4 ciclos. Para un conjunto de datos sísmicos de M trazas con N muestras por trazas, en la etapa de transformación por columnas se reconstruyen matrices de tamaño $N \times \left\lceil \frac{M}{2} \right\rceil$, por lo tanto, se necesitan $\left(\frac{1}{2} \cdot N \cdot \left\lceil \frac{M}{2} \right\rceil + 4\right)$ ciclos de reloj para completar esta etapa.

De igual forma, en la etapa de transformación por filas se reconstruye una matriz de tamaño $N \times M$, por lo tanto, empleará $\left(\frac{1}{2} \cdot N \cdot M + 4\right)$ ciclos de reloj para completar esta etapa.

A su vez, la cuantificación inversa toma $(N \cdot M)$ ciclos de reloj para procesar el conjunto de datos sísmicos.

Por otro lado, el número de ciclos de reloj que tarda el decodificador Huffman en decodificar un dato es variable y dependerá de la ubicación de cada símbolo al interior del diccionario Huffman, el cual cambia para cada conjunto de datos.

Así, el número de ciclos de reloj necesarios para descomprimir un conjunto de datos sísmicos de M trazas con N muestras por trazas es:

$$\begin{aligned} ciclos_{\text{ descompresión}} &= ciclos_{\text{ decodificación}} + ciclos_{\text{ cuantificación\_inversa}} \\ &\quad + ciclos_{\text{ lifting\_columnas}} + ciclos_{\text{ lifting\_filas}} \quad , \\ ciclos_{\text{ descompresión}} &\geq ciclos_{\text{ decodificación}} + N \cdot M + \tfrac{1}{4} \cdot N \cdot M + 4 + \tfrac{1}{2} \cdot N \cdot M + 4 \ , \\ ciclos_{\text{ descompresión}} &\geq ciclos_{\text{ decodificación}} + \tfrac{7}{4} \cdot N \cdot M + 8 \ . \end{aligned} \tag{3.6}$$

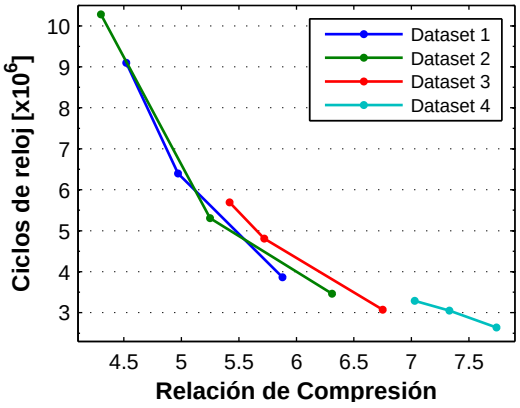
Teniendo en cuenta la latencia de cada circuito fue posible desarrollar una estrategia de segmentación en la cual no es necesario esperar a que el decodificador termine de decodificar todo el conjunto de datos. En este sentido, cada vez que el decodificador entrega un dato válido se activa el circuito de cuantificación inversa y un ciclo de reloj después se habilita el circuito de transformación inversa para que avance en la etapa de transformación por columnas. Una vez se tienen todos los datos decodificados se procede a habilitar la etapa de transformación por filas.

De esta manera el número de ciclos de reloj necesarios para descomprimir un conjunto de datos sísmicos de M trazas con N muestras por trazas está dado por:

$$ciclos_{\text{ descompresión}} \geq ciclos_{\text{ decodificación}} + \frac{1}{2} \cdot N \cdot M + 6 \, . \tag{3.7}$$

La Figura 22 muestra el número de ciclos de reloj empleados en la descompresión de varios conjuntos de datos sísmicos de 18 trazas sísmicas con 3584 muestras por traza. Se puede observar que existe una relación inversa entre el tiempo de descompresión y la relación de compresión ya que el número de ciclos de reloj disminuye a medida que aumenta la relación de compresión.

Figura 22: Número de ciclos de reloj empleados en la descompresión en FPGA.



4. DESCOMPRESIÓN EN GPU

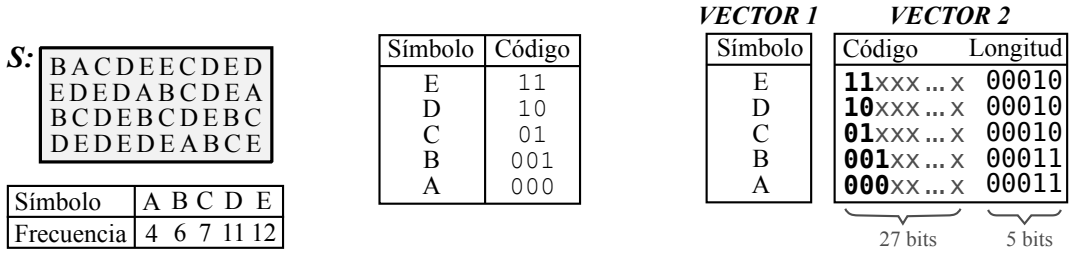
En el presente capítulo se expone el algoritmo diseñado para descomprimir los datos sísmicos en una GPU. El capítulo inicia con una descripción del proceso empleado para comprimir los datos en una CPU. La segunda sección del capítulo describe los *kernels* empleados para realizar la descompresión en la GPU, mientras que la tercera sección presenta los resultados obtenidos.

4.1. Compresión

Tal como se mencionó anteriormente, el proceso de compresión generalmente es desarrollado durante la adquisición de los datos sísmicos empleando CPUs. Aquí también se utilizó un algoritmo de compresión compuesto por tres etapas: *DWT 2D* empleando el esquema *lifting* y el filtro *CDF 2.2*, Cuantificación Uniforme y Codificación Huffman (ver Figura 12).

El diccionario Huffman fue almacenado en dos vectores cuyos elementos son variables de 32 bits. En el primer vector se guardaron los símbolos resultantes de la cuantificación. En el segundo vector, los 27 bits más significativos de cada uno de sus elementos fueron empleados para guardar los códigos Huffman, mientras que en los 5 bits menos significativos se guardó su longitud efectiva. La Figura 23 muestra los dos vectores del diccionario Huffman correspondiente a una señal $\boldsymbol{S}$ de 40 datos.

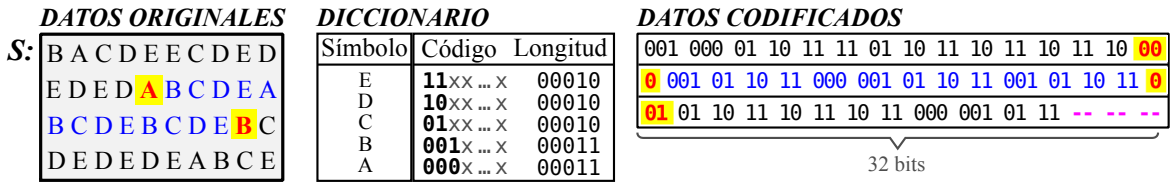
Figura 23: Ejemplo de almacenamiento del diccionario Huffman.



(a) Conjunto de datos (b) Diccionario (c) Almacenamiento diccionario

Por otro lado, los datos codificados se pueden almacenar en un vector en el cual cada posición tiene 32 bits tal como se muestra en la Figura 24. Note que, debido a la longitud variable de los códigos, existe la posibilidad que algunos los códigos sean divididos para ser almacenados en diferentes posiciones del vector, tal como ocurre con los códigos de los datos resaltados A y B.

Figura 24: Ejemplo de almacenamiento de los datos codificados.

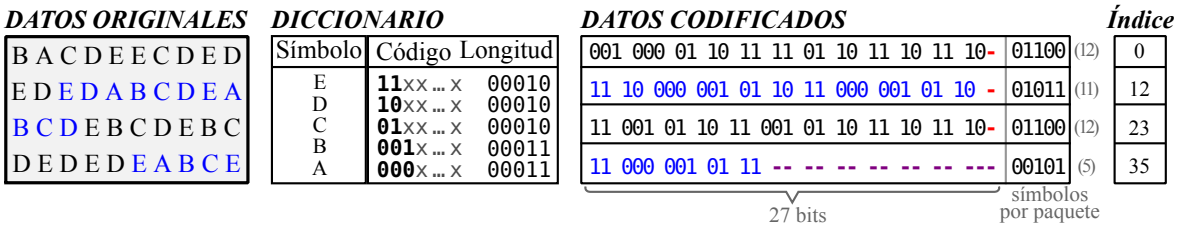


La decodificación es un proceso altamente secuencial ya que, en atención a la longitud variable de los códigos, se debe iniciar desde el primer bit de un código para poder decodificar correctamente el conjunto de datos. Esto impide ejecutar la decodificación en varios procesos corriendo en paralelo.

Con el fin de posibilitar la ejecución en paralelo del proceso de decodificación en GPU, los datos codificados fueron almacenados usando la estrategia desarrollada por nosotros en [41]. En esta estrategia, los códigos son almacenados en paquetes de 32 bits con la restricción de que ningún código se puede dividir para ser almacenado en dos paquetes diferentes. De los 32 bits del paquete, los 5 bits menos significativos son usados para indicar la cantidad de símbolos que hay en el paquete, mientras que los 27 bits más significativos son empleados para almacenar los datos codificados en sí. Esto permite que cada paquete pueda ser tratado como un pequeño conjunto de datos que puede ser decodificado independientemente de los otros paquetes.

La Figura 25 muestra cómo son almacenados los códigos correspondientes al conjunto de datos $\boldsymbol{S}$ empleando la estrategia propuesta. El vector *índice* es necesario para mantener el orden de los datos decodificados dentro del conjunto $\boldsymbol{S}$.

Figura 25: Ejemplo de almacenamiento en paquetes de los datos codificados.



Dado que algunos paquetes tendrán bits sin usar (representado como –) y sumado al vector *índice* adicional, la estrategia propuesta implica una disminución en la relación de compresión resultante, tal como se observa en la Figura 26.

4.2. Descompresión

La Figura 27 muestra el esquema empleado para elaborar los algoritmos que permitieron llevar a cabo el proceso de descompresión en la GPU. Estos algoritmos fueron desarrollados teniendo en cuenta los diversos procesos involucrados en cada etapa de la descompresión.

Figura 26: Relación de compresión usando almacenamiento en paquetes.

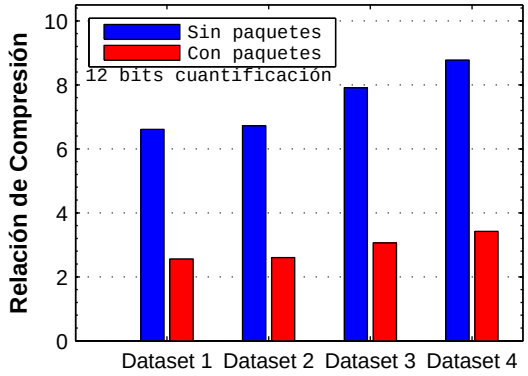


Figura 27: Esquema de descompresión empleado.

4.2.1. Decodificación en GPU

El algoritmo para esta etapa de la descompresión necesita la siguiente información:

- Diccionario Huffman.
- Paquetes con los datos codificados.
- Vector *índice* obtenido en la codificación.

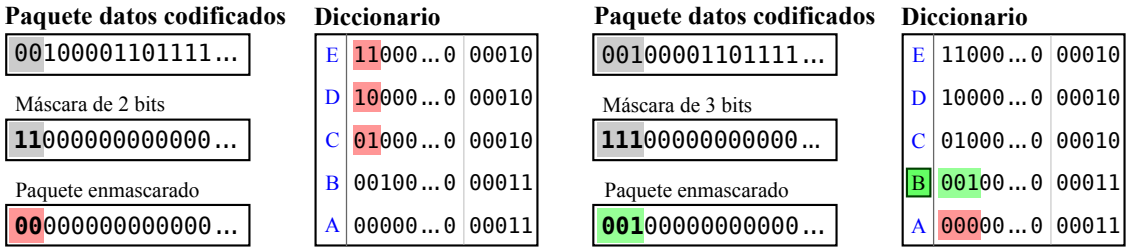
La decodificación de los datos almacenados en cada paquete de 32 bits se logra comparando sucesivamente los códigos en el diccionario con porciones de datos en los paquetes. A continuación se presenta el algoritmo usado para realizar estas comparaciones.

1. Leer la primera posición del diccionario para determinar la longitud L del primer código.
2. Crear una máscara de 32 bits compuesta por '1' en los L bits más significativos y '0' en los otros bits.
3. Aplicar la máscara al paquete con los datos codificados.
4. Comparar el paquete enmascarado con los códigos en el diccionario que tengan la misma longitud L.
5. Si el paquete enmascarado no concuerda con alguno de los códigos de longitud L (decodificación incompleta), entonces saltar al paso 8. Si el paquete enmascarado concuerda con un código (decodificación correcta), entonces leer el símbolo correspondiente y continuar con el paso 6.

6. Si no se han decodificado todos los símbolos del paquete, entonces continuar con el paso 7, en caso contrario, finalizar la decodificación.
7. Desplazar el paquete L bits hacia la izquierda y regresar al paso 1 para decodificar el siguiente código.
8. Leer la siguiente posición del diccionario para determinar la nueva longitud L del código y regresar al paso 2.

En la Figura 28 se muestra la decodificación con dos máscaras diferentes. La decodificación está incompleta en la Figura 28a debido a que el paquete enmascarado no concuerda con alguno de los códigos de 2 bits en el diccionario. En la Figura 28b se muestra que el símbolo ***B*** ha sido decodificado correctamente ya que su código concuerda con el paquete enmascarado.

Figura 28: Ejemplo del proceso de decodificación [41].



(a) Decodificación incompleta. (b) Decodificación correcta.

Los resultados de la decodificación de cada uno de los paquetes son ordenados de acuerdo a la información en el vector *índice* para así obtener el vector $\boldsymbol{x_Q}$ de la Figura 27.

4.2.2. Cuantificación inversa en GPU

Esta etapa de la descompresión se lleva cabo usando la ecuación

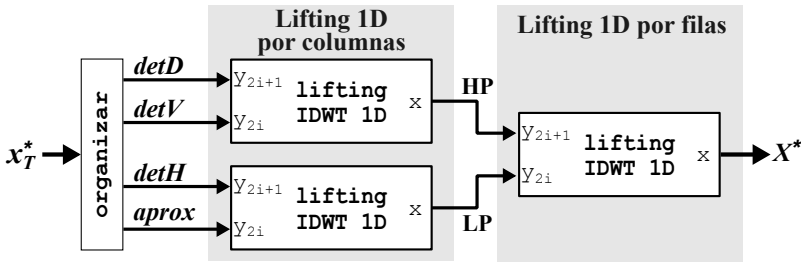
$$\boldsymbol{x_T^*} = (K1 \cdot \boldsymbol{x_Q}) + K2\ . \tag{4.1}$$

Por lo tanto, el *kernel* necesita que le sean suministrados los parámetros de compresión $K1$ y $K2$ obtenidos en la cuantificación al momento de la compresión.

4.2.3. Transformación inversa en GPU

En este caso se desarrolló un *kernel* con indexación 2D teniendo en cuenta que en la compresión se empleó una transformación 2D. Este *kernel* fue desarrollado de acuerdo al esquema *lifting* mostrado en la Figura 29.

Figura 29: Esquema *lifting* para la *IDWT* 2D.



La Figura 30 muestra el esquema *lifting* para implementar la *IDWT* 1D. Para el filtro *CDF 2.2* se tiene que $\tilde{T} = -\frac{1}{2}(1 + z^1)$ y $\tilde{S} = \frac{1}{4}(z^{-1} + 1)$, con lo cual las operaciones de predicción y actualización se realizan de acuerdo a las siguientes ecuaciones:

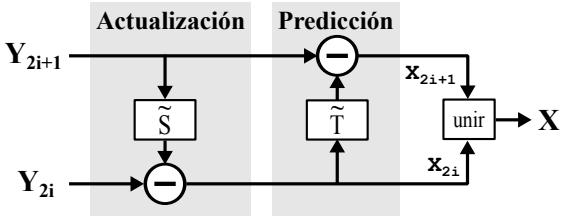
$$\text{Actualización:} \quad \boldsymbol{X}_{2i} = \boldsymbol{Y}_{2i} - \left\lfloor \frac{1}{4}\left(\boldsymbol{Y}_{2i+1} + \boldsymbol{Y}_{2i-1}\right) \right\rfloor \tag{4.2a}$$

$$\boldsymbol{X}_{first} = \boldsymbol{Y}_{2i} - \left\lfloor \frac{1}{4} \cdot \boldsymbol{Y}_{2i+1} \right\rfloor \ . \tag{4.2b}$$

$$\text{Predicción:} \quad \boldsymbol{X}_{2i+1} = \boldsymbol{Y}_{2i+1} + \left\lfloor \frac{1}{2}\left(\boldsymbol{X}_{2i} + \boldsymbol{X}_{2i+2}\right) \right\rfloor \tag{4.3a}$$

$$\boldsymbol{X}_{last} = \boldsymbol{Y}_{2i+1} + \left\lfloor \frac{1}{2} \cdot \boldsymbol{X}_{2i} \right\rfloor \ . \tag{4.3b}$$

Figura 30: Esquema *lifting* para la *IDWT* 1D.



Para obtener la matriz de coeficientes $\boldsymbol{HP}$ se debe realizar una transformación 1D por columnas con las entradas $\boldsymbol{detD}$ y $\boldsymbol{detV}$ (ver Figura 29). El algoritmo para realizar esta transformación se implementó de la siguiente forma:

- Actualización: La primera fila par de $\boldsymbol{HP}$ es calculada con la ecuación 4.2b, mientras que las demás filas pares se calculan usando la ecuación (4.2a). En estas ecuaciones las filas pares de $\boldsymbol{Y}$ corresponden a cada una de las filas de $\boldsymbol{detV}$ y las filas impares de $\boldsymbol{Y}$ corresponden a cada una de las filas de $\boldsymbol{detD}$.

- Predicción: La última fila impar de $\boldsymbol{HP}$ es calculada con la ecuación (4.3b) mientras que las demás filas impares se calculan usando la ecuación (4.3a). En estas ecuaciones, las filas impares de $\boldsymbol{Y}$ corresponden a cada una de las filas de $\boldsymbol{detD}$ y las filas pares de $\boldsymbol{X}$ corresponden a las filas pares de $\boldsymbol{HP}$ calculadas en la actualización.

Este algoritmo también se puede emplear con las entradas $\boldsymbol{detH}$ y $\boldsymbol{aprox}$ para obtener la matriz de coeficientes $\boldsymbol{LP}$ (ver Figura 29).

La salida $\boldsymbol{X}^*$ con los datos reconstruídos se obtiene al llevar a cabo una transformación 1D por filas con las matrices $\boldsymbol{HP}$ y $\boldsymbol{LP}$ previamente calculadas (ver Figura 29). El algoritmo para realizar esta transformación se implementó de la siguiente forma:

- Actualización: La primera columna par de $\boldsymbol{X}^*$ es calculada con la ecuación (4.2b) mientras que las demás columnas pares se calculan usando la ecuación (4.2a). En estas ecuaciones, las columnas pares de $\boldsymbol{Y}$ corresponden a cada una de las columnas de $\boldsymbol{LP}$ y las columnas impares de $\boldsymbol{Y}$ corresponden a cada una de las columnas de $\boldsymbol{HP}$.
- Predicción: La última columna impar de $\boldsymbol{X}^*$ es calculada con la ecuación (4.3b) mientras que las demás columnas impares se calculan usando la ecuación (4.3a). En estas ecuaciones, las columnas impares de $\boldsymbol{Y}$ corresponden a cada una de las columnas de $\boldsymbol{HP}$ y las columnas pares de $\boldsymbol{X}$ corresponden a las columnas pares de $\boldsymbol{X}^*$ calculadas en la actualización.

4.3. Resultados

En la GPU, los *threads* son agrupados en *bloques* que a su vez se organizan en una cuadrícula o *grid* (Ver Figura 31). La distribución de la cuadrícula y de los bloques es determinada por el programador y puede hacerse para tener indexación en 1, 2 o 3 dimensiones. Los *threads* de un *bloque* se comunican mediante la memoria compartida, mientras que todos los *threads* de la cuadrícula se comunican a través de la memoria global [42].

Adicionalmente, los *threads* de un *bloque* son agrupados en *warps* de 32 *threads* que ejecutan en paralelo la misma instrucción sobre diferentes datos. El desempeño de un *warp* disminuye cuando sus *threads* deban tomar diferentes caminos al presentarse saltos condicionales en el algoritmo. En estos casos, el *warp* trabajará secuencialmente activando primero los *threads* de un camino y luego los *threads* del otro camino [43].

Este fenómeno es conocido como divergencia y su efecto es mostrado en la Figura 32. En este caso, los *threads* divergen al llegar al salto condicional A ya que algunos *threads* deberán ir por la ramificación B, mientras que los otros deberán hacerlo por C. Por lo tanto, el *warp* activará primero los *threads* de la ramificación B, luego activará los

Figura 31: Jerarquía de hilos [42].

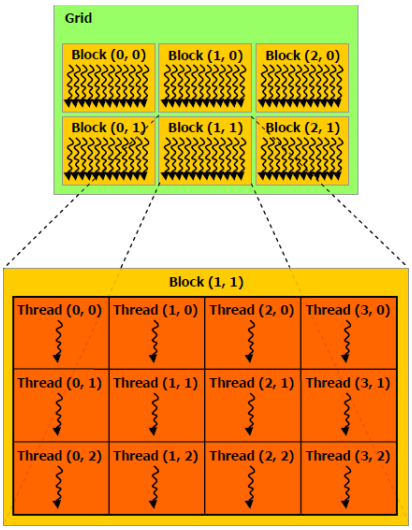
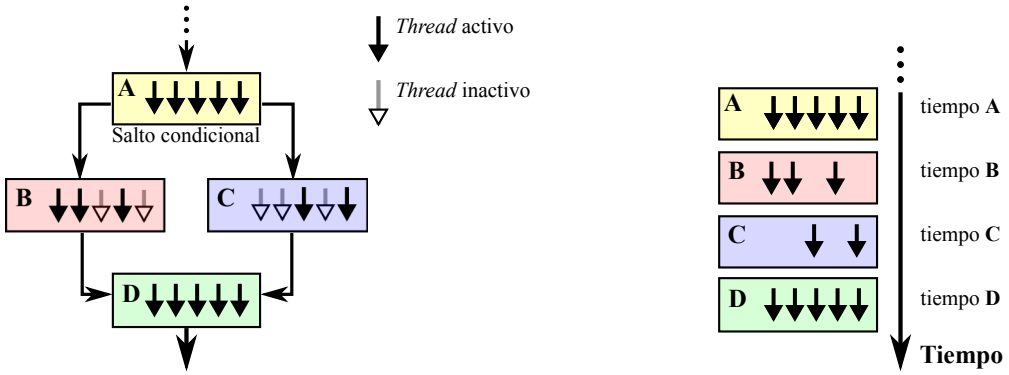


Figura 32: Tiempo de ejecución de un *kernel* con *threads* divergentes.



(a) Algoritmo. (b) Tiempo de ejecución.

threads de la ramificación **C**, y finalmente continua con la ejecución en paralelo luego de converger en **D**.

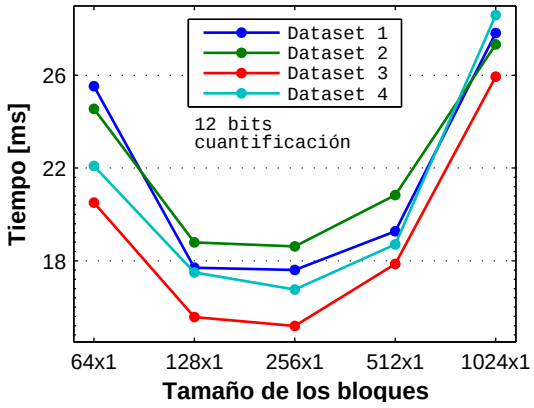
Cada una de la etapas de la descompresión fueron desarrolladas en diferentes *kernels* e implementadas en tres GPUs con diferentes arquitecturas: *GeForce 310m*, *GeForce GTX 660* y *GeForce GTX 760*.

La decodificación se desarrolló en un *kernel* con indexación 1D teniendo en cuenta que cada uno de los paquetes que conforman $\boldsymbol{x_C}$ puede verse como un conjunto de datos que no tiene dependencia con los demás paquetes. De igual manera, la cuantificación inversa se desarrolló en un *kernel* con indexación 1D ya que cada uno de los datos de $\boldsymbol{x_Q}$ puede ser procesado independientemente por un *thread*. Por otro lado, la *IDWT* 2D se desarrolló en un *kernel* con indexación 2D debido a que se necesita de los datos adyacentes en ambas direcciones para calcular cada uno de los datos de la matriz $\boldsymbol{X}^*$.

Las pruebas realizadas en cada GPU arrojaron resultados con similar comportamiento. A continuación se presentan los tiempos de ejecución obtenidos para cada *kernel* en la GPU *GeForce GTX 760*.

La Figura 33 muestra el tiempo de ejecución del *kernel* de decodificación de varios conjuntos de trazas sísmicas. El tiempo de decodificación de cada conjunto de trazas varía dado que la naturaleza de cada uno de ellas conlleva a un diccionario Huffman diferente y por lo tanto, tiempos de búsqueda diferentes. En la figura se observa que en todos los casos el mejor desempeño se obtuvo usando bloques de 256×1 *threads*.

Figura 33: Tiempo de ejecución del *kernel* de decodificación.



Por otro lado, el tiempo de ejecución del *kernel* de cuantificación inversa es el mismo para los conjuntos de trazas sísmicas que tengan igual cantidad de datos. Esto también se presenta con el tiempo de ejecución del *kernel* de transformación inversa.

En las Tablas 3 y 4 se muestra el tiempo de ejecución de los *kernels* de cuantificación inversa y transformación 2D inversa empleando diferentes tamaños de bloques de *threads*. El mejor desempeño para la etapa de cuantificación inversa se obtuvo al emplear bloques de 128×1 *threads*, mientras que para la transformación inversa se obtuvo con bloques 16×16 *threads*.

Tabla 3: Tiempo de ejecución del *kernel* de cuantificación inversa.

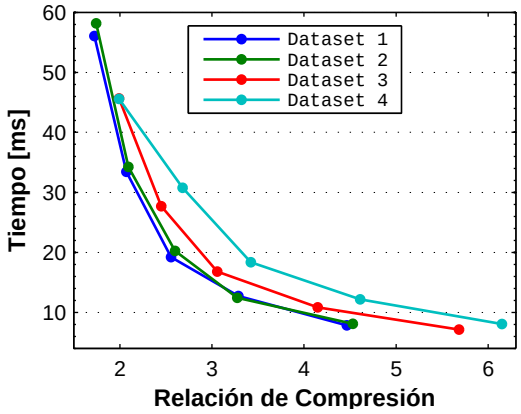
Tamaño bloques [*threads*]	Tiempo [μs]
64×1	171.58
128×1	104.63
256×1	106.99
512×1	118.25
1024×1	128.79

Tabla 4: Tiempo de ejecución del *kernel* de transformación inversa 2D.

Tamaño del bloque [*threads*]	Tiempo [*ms*]
4×4	5.75
8×8	1.59
16×16	1.49
32×32	1.95

La Figura 34 muestra el tiempo de descompresión de varios conjuntos de datos sísmicos empleando los tres *kernels*. Para cada uno de los *kernels* se escogió el tamaño de bloques que permite obtener el mejor desempeño, es decir, 256×1 *threads* para la etapa de decodificación, 128×1 *threads* para la etapa de cuantificación inversa y 16×16 *threads* para la etapa de transformación inversa. Se puede observar que existe una relación inversa ya que el tiempo de descompresión disminuye a medida que aumenta la relación de compresión.

Figura 34: Tiempo de descompresión para diferentes *datasets*.



La naturaleza secuencial del algoritmo de decodificación Huffman hace que este se convierta en el cuello de botella del proceso de descompresión. Se pudo observar que la diferencia en el orden y cantidad de datos comprimidos en los diferentes paquetes afecta considerablemente la velocidad de procesamiento del *kernel* de decodificación. Esto se debe a que cada *thread* tomará diferentes saltos en la ejecución del algoritmo de acuerdo al código que esté buscando dando lugar al fenómeno de divergencia y penalizando el tiempo de ejecución de los *kernels*.

5. CONCLUSIONES

El efecto en la relación de compresión y la relación señal a ruido al usar diferentes familias de filtros Wavelet en un algoritmo de compresión de datos sísmicos conformado por Transformación Wavelet, Cuantificación Uniforme y Codificación Huffman fue analizado en la presente investigación.

Este análisis nos permitió evidenciar cómo la etapa de transformación afecta la calidad del proceso de compresión en términos de *SNR* vs. *CR*. Adicionalmente, este análisis nos permitió seleccionar una estrategia de implementación computacional con el fin de reducir los requerimientos en términos capacidad de cómputo, tiempo de procesamiento y uso de memoria.

Las pruebas fueron realizadas en dos tipos de arquitectura ampliamente usadas en las aplicaciones geofísicas a saber: FPGA y GPU. El proceso de descompresión fue implementado en su totalidad en ambas arquitecturas, es decir, se implementaron las etapas de Decodificación Huffman, Cuantificación Uniforme Inversa e *IDWT 2D* tanto en FPGA como en GPU.

Los resultados obtenidos sugieren que usando la familia de filtros Wavelet biortogonales *CDF 2.x* se logra mejor desempeño en términos de *SNR* y *CR* que si se emplea otra familia de filtros.

Se escogió el esquema *lifting* como estrategia de implementación de la etapa de transformación con el fin de reducir el costo computacional de esta etapa. De igual manera, se seleccionó el filtro *CDF 2.2* con el fin de:

- Reducir el número de operaciones matemáticas necesarias para realizar la *IDWT*.
- Reducir la complejidad computacional ya que los coeficientes para este filtro en el esquema *lifting* permiten realizar las operaciones mediante corrimiento de los datos.

A partir del diseño realizado para llevar a cabo la descompresión en FPGA se estableció el número de ciclos de reloj necesarios para llevar a cabo las etapas de transformación y cuantificación en función del tamaño de los datos. Para un conjunto de datos sísmicos compuesto por M trazas con N muestras por traza, la FPGA necesitará $\frac{7}{4} \cdot N \cdot M + 8$ ciclos de reloj para realizar estas dos etapas. Este tiempo se disminuye a $\frac{1}{2} \cdot N \cdot M + 6$ ciclos si los datos se empiezan a procesar en la medida que van siendo decodificados. Sin embargo, los resultados no son concluyentes en cuanto al tiempo de ejecución de la etapa de decodificación.

De otro lado, las pruebas realizadas para descomprimir datos sísmicos en GPU evidenciaron la distribución de bloques de *threads* con la cual se podría obtener el

menor tiempo de ejecución. El menor tiempo de descompresión se obtuvo al emplear bloques de 256×1 *threads* en la decodificación, 128×1 *threads* en la cuantificación inversa y 16×16 *threads* en la transformación inversa.

Tanto en FPGA como en GPU se observó que el proceso de descompresión presenta un cuello de botella en la etapa de Decodificación Huffman. Esto debido a que la naturaleza secuencial del algoritmo de decodificación hace que la búsqueda de los códigos más largos en el diccionario tome más tiempo. Lo anterior en atención a que los códigos más largos están al final del diccionario y primero se debe recorrer y descartar los códigos más cortos.

También se observó que el tiempo de descompresión disminuye cuando la relación de compresión aumenta. Esto se debe a que la longitud promedio de los códigos es menor en la medida que la relación de compresión aumenta, lo cual resulta en menos tiempo de búsqueda en el diccionario Huffman.

La FPGA empleada en este trabajo permite almacenar datos sísmicos de hasta 65536 muestras. Para el caso de los datos sísmicos suministrados por ECOPETROL, entidad patrocinadora del proyecto, en esta FPGA se podrán procesar aproximadamente 18 trazas sísmicas con 3584 muestras por traza. Por lo tanto, con el fin de poder procesar más trazas sísmicas, recomendamos la implementación de memorias externas sin que los constantes accesos a memoria penalicen considerablemente el tiempo de descompresión.

En cuanto a la implementación en GPU, proponemos implementar *kernels* para cada etapa de transformación inversa 1D, es decir, *IDWT* por filas e *IDWT* por columnas y estimar el desempeño computacional de esta alternativa. Adicionalmente, sugerimos modificar el *kernel* de decodificación empleando técnicas que permitan reducir el efecto de la divergencia con el fin de disminuir el tiempo de descompresión. De igual forma, sugerimos que el vector índice sea calculado en la GPU. Con esto se aumentará la relación de compresión debido a que no será necesario el envió de dicho vector.

REFERENCIAS

[1] BAAZIZ Abdelkader and Quoniam Luc, "How to use big data technologies to optimize operations in upstream petroleum industry," *International Journal of Innovation - IJI*, vol. 1, no. 1, 2013.

[2] HE Chuan, Lu Mi, and Sun C., "Accelerating seismic migration using FPGA-based coprocessor platform," en *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, pp. 207–216, April 2004.

[3] AVERBUCH Amir Z., Zheludev Valery A., GuttmannN Moshe, and Kosloff Dan D., "LCT-wavelet based algorithms for data compression," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 11, no. 05, p. 1350032, 2013.

[4] WU Wenbo, Yang Zhigao, Qin Qianqing, and Hu Fuxiang, "Adaptive seismic data compression using wavelet packets," en *2006 IEEE International Symposium on Geoscience and Remote Sensing. IGARSS 2006. IEEE International Conference on*, pp. 787–789, July 2006.

[5] AL-MOOHIMEED M.A., "Towards an efficient compression algorithm for seismic data," en *Radio Science Conference, 2004. Proceedings. 2004 Asia-Pacific*, pp. 550–553, Aug 2004.

[6] DUVAL Laurent C, Røsten Tage, *et al.*, "Filter bank decomposition of seismic data with application to compression and denoising," en *2000 SEG Annual International Meeting Society Exploration Geophysicists*, pp. 2055–2058, Society of Exploration Geophysicists, 2000.

[7] SALOMON David, *Data Compression: The Complete Reference*. Springer-Verlag London, 3 ed., 2007.

[8] WANG Yongzhong and Wu Ru-Shan, "Seismic data compression by an adaptive local cosine/sine transform and its effects on migration," *Geophysical Prospecting*, vol. 48, no. 6, pp. 1009–1031, 2000.

[9] VILLASENOR J.D., Belzer B., and Liao J., "Wavelet filter evaluation for image compression," *Image Processing, IEEE Transactions on*, vol. 4, pp. 1053–1060, Aug 1995.

[10] AVERBUCH A.Z., Meyer F., Stromberg J.O., Coifman R., and Vassiliou A., "Low bit-rate efficient compression for seismic data," *Image Processing, IEEE Transactions on*, vol. 10, pp. 1801–1814, Dec 2001.

[11] FAJARDO Carlos A., Angulo Carlos A., Mantilla Julián G., Obregón Iván, Castillo Javier, Pedraza César, and Reyes Óscar M., "Computational Architecture for Fast Seismic Data Transmission between CPU and FPGA using Data Compression," en *2016 Data Compression Conference*, (Salt Lake, United States.), 2016.

[12] FAJARDO Carlos, Villar Javier Castillo, and Pedraza César, "Reducción de los tiempos de cómputo de la migración sísmica usando FPGAs y GPGPUs: Un artículo de revisión," *Ingeniería y Ciencia*, vol. 9, no. 17, pp. 261–293, 2013.

[13] ABREO C. Sergio A. and Ramírez S. Ana B., "Viabilidad de acelerar la migración sísmica 2D usando un procesador específico implementado sobre un FPGA," *Ingeniería e investigación*, vol. 30, no. 1, pp. 64–70, 2010.

[14] ABDELKHALEK Rached, Calandra Henri, Coulaud Olivier, Roman Jean, and Latu Guillaume, "Fast seismic modeling and reverse time migration on a GPU cluster," en *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, pp. 36–43, IEEE, 2009.

[15] VASSILIOU Anthony A. and Wickerhouser Mladen V., "Comparison of wavelet image coding schemes for seismic data compression," en *Proc. SPIE*, vol. 3169, pp. 118–126, 1997.

[16] SHANNON C.E., "A mathematical theory of communication," *Bell System Technical Journal, The*, vol. 27, pp. 379–423, July 1948.

[17] VILLASENOR J.D., Ergas R.A., and Donoho P.L., "Seismic data compression using high-dimensional wavelet transforms," en *Data Compression Conference, 1996. DCC '96. Proceedings*, pp. 396–405, Mar 1996.

[18] FAJARDO Carlos A., Reyes Oscar M., and Silva Ana Ramirez, "Seismic data compression using 2d lifting-wavelet algorithms," *Ingeniería y Ciencia | ing.cienc.*, vol. 11, no. 21, pp. 221–238, 2015.

[19] ZHENG Fan and Liu Shufen, "A fast compression algorithm for seismic data from non-cable seismographs," en *Information and Communication Technologies (WICT), 2012 World Congress on*, pp. 1215–1219, Oct 2012.

[20] DAUBECHIES Ingrid and Sweldens Wim, "Factoring wavelet transforms into lifting steps," *Journal of Fourier analysis and applications*, vol. 4, no. 3, pp. 247–269, 1998.

[21] ANGELOPOULOU Maria E, Masselos Konstantinos, Cheung Peter YK, and Andreopoulos Yiannis, "Implementation and comparison of the 5/3 lifting 2d discrete wavelet transform computation schedules on fpgas," *J. Signal Process. Syst.*, vol. 51, pp. 3–21, Oct. 2008.

[22] APARNA P. and David Sumam, "Adaptive Local Cosine transform for Seismic Image Compression," *2006 International Conference on Advanced Computing and Communications*, pp. 254–257, dec 2006.

[23] AQRAWI AA and Elster AC., "Bandwidth Reduction through Multithreaded Compression of Seismic Images," *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 1730–1739, May 2011.

[24] XI-ZHEN Wang, Yun-tina Then, Meng-tan Gao, and Hui Jiang, "Seismic data compression based on integer wavelet transform," *Acta Seismologica Sinica*, vol. 17, no. 04, pp. 123–128, 2004.

[25] XIE Xing and Qin Qianqing, "Fast Lossless Compression of Seismic Floating-Point Data," *2009 International Forum on Information Technology and Applications*, pp. 235–238, May 2009.

[26] LERVIK John M, Rosten T, and Ramstad Tor A, "Subband Seismic Data Compression: Optimization and Evaluation," *Digital Signal Processing Workshop Proceedings*, no. 1, pp. 65–68, 1996.

[27] WOOD Lawrence C, "Seismic data compression methods," *Geophysics*, vol. 39, no. 4, pp. 499–525, 1974.

[28] SPANIAS A.S., Jonsson S.B., and Stearns S.D., "Transform coding algorithms for seismic data compression," en *Circuits and Systems, 1990., IEEE International Symposium on*, pp. 1573–1576 vol.2, May 1990.

[29] DAUBECHIES Ingrid *et al.*, *Ten lectures on wavelets*, vol. 61. SIAM, 1992.

[30] BOSMAN Cheryl and Reiter Edmund, "Seismic Data Compression Using Wavelet Transforms," en *In 1993 SEG Annual Meeting*, pp. 1261–1264, Society of Exploration Geophysicists., 1993.

[31] "Overview of JPEG 2000." `http://jpeg.org/jpeg2000/`, 2002. [online] Accessed December-2015.

[32] RØSTEN Tage, Ramstad Tor A., and Amundsen Lasse, "Optimization of sub-band coding method for seismic data compression," *Geophysical Prospecting*, vol. 52, no. 5, pp. 359–378, 2004.

[33] REDDY T. Ashwini, Devi K. Renuka, and Gangashetty Suryakanth V., "Nonlinear principal component analysis for seismic data compression," *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*, pp. 927–932, mar 2012.

[34] YU Wenmao, Xie Kai, and Bai Zhijun, "Fast seismic data compression based on high-efficiency SPIHT," *Electronics Letters*, vol. 50, no. 5, pp. 365–367, 2014.

[35] SWELDENS Wim, "Lifting scheme: a new philosophy in biorthogonal wavelet constructions," en *Proc. SPIE*, vol. 2569, pp. 68–79, 1995.

[36] CALDERBANK A.R., Daubechies Ingrid, Sweldens Wim, and Yeo Boon-Lock, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332 – 369, 1998.

[37] ANDRA K., Chakrabarti C., and Acharya T., "A VLSI architecture for lifting-based forward and inverse wavelet transform," *Signal Processing, IEEE Transactions on*, vol. 50, pp. 966–977, Apr 2002.

[38] UYTTERHOEVEN Geert, Roose Dirk, and Bultheel Adhemar, "Wavelet transforms using the lifting scheme," *ITA-Wavelets Report WP*, vol. 1, 1997.

[39] ANGULO Carlos A., Fajardo Carlos, Reyes Oscar, and Castillo Javier, "FPGA implementation of a Huffman decoder for high speed seismic data decompression," en *2014 Data Compression Conference*, (Salt Lake, United States.), p. 396, IEEE Comput. Soc, 2014.

[40] SÁNCHEZ Fabián, Fajardo Carlos A., Angulo Carlos A., Reyes Óscar M, and Bouman Charles A, "A computational architecture for discrete wavelet transform using lifting scheme," en *2014 XIX Symposium on Image, Signal Processing and Artificial Vision*, IEEE, 2014.

[41] ANGULO Carlos A., Hernández Christian, Rincon Gabriel, Boada Carlos, Castillo J., and Fajardo Carlos, "Accelerating Huffman decoding of seismic data on GPUs," en *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, (Bogot Colombia), Sept 2015.

[42] "CUDA C Programming Guide." `http://docs.nvidia.com/cuda/`, 2015. [Online; accedido 20-06-2016].

[43] MENG Jiayuan, Tarjan David, and Skadron Kevin, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," *SIGARCH Comput. Archit. News*, vol. 38, pp. 235–246, June 2010.

[44] CASTELAR Jairo A., Angulo Carlos A., and Fajardo Carlos A., "Parallel decompression of seismic data on GPU using a lifting wavelet algorithm," en *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, Sept 2015.

BIBLIOGRAFÍA

ABDELKHALEK Rached, Calandra Henri, Coulaud Olivier, Roman Jean, and Latu Guillaume, "Fast seismic modeling and reverse time migration on a GPU cluster," en *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, pp. 36–43, IEEE, 2009

ABREO C. Sergio A. and Ramírez S. Ana B., "Viabilidad de acelerar la migración sísmica 2D usando un procesador específico implementado sobre un FPGA," *Ingeniería e investigación*, vol. 30, no. 1, pp. 64–70, 2010

AL-MOOHIMEED M.A., "Towards an efficient compression algorithm for seismic data," en *Radio Science Conference, 2004. Proceedings. 2004 Asia-Pacific*, pp. 550–553, Aug 2004

ANDRA K., Chakrabarti C., and Acharya T., "A VLSI architecture for lifting-based forward and inverse wavelet transform," *Signal Processing, IEEE Transactions on*, vol. 50, pp. 966–977, Apr 2002

ANGELOPOULOU Maria E, Masselos Konstantinos, Cheung Peter YK, and Andreopoulos Yiannis, "Implementation and comparison of the 5/3 lifting 2d discrete wavelet transform computation schedules on fpgas," *J. Signal Process. Syst.*, vol. 51, pp. 3–21, Oct. 2008

ANGULO Carlos A., Fajardo Carlos, Reyes Oscar, and Castillo Javier, "FPGA implementation of a Huffman decoder for high speed seismic data decompression," en *2014 Data Compression Conference*, (Salt Lake, United States.), p. 396, IEEE Comput. Soc, 2014

ANGULO Carlos A., Hernández Christian, Rincon Gabriel, Boada Carlos, Castillo J., and Fajardo Carlos, "Accelerating Huffman decoding of seismic data on GPUs," en *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, (Bogot Colombia), Sept 2015

APARNA P. and David Sumam, "Adaptive Local Cosine transform for Seismic Image Compression," *2006 International Conference on Advanced Computing and Communications*, pp. 254–257, dec 2006

AQRAWI AA and Elster AC., "Bandwidth Reduction through Multithreaded Compression of Seismic Images," *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 1730–1739, May 2011

AVERBUCH A.Z., Meyer F., Stromberg J.O., Coifman R., and Vassiliou A., "Low bitrate efficient compression for seismic data," *Image Processing, IEEE Transactions on*, vol. 10, pp. 1801–1814, Dec 2001

AVERBUCH Amir Z., Zheludev Valery A., GuttmannN Moshe, and Kosloff Dan D., "LCT-wavelet based algorithms for data compression," *International Journal of Wavelets, Multiresolution and Information Processing*, vol. 11, no. 05, p. 1350032, 2013

BAAZIZ Abdelkader and Quoniam Luc, "How to use big data technologies to optimize operations in upstream petroleum industry," *International Journal of Innovation - IJI*, vol. 1, no. 1, 2013

BOSMAN Cheryl and Reiter Edmund, "Seismic Data Compression Using Wavelet Transforms," en *In 1993 SEG Annual Meeting*, pp. 1261–1264, Society of Exploration Geophysicists., 1993

CALDERBANK A.R., Daubechies Ingrid, Sweldens Wim, and Yeo Boon-Lock, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332 – 369, 1998

CASTELAR Jairo A., Angulo Carlos A., and Fajardo Carlos A., "Parallel decompression of seismic data on GPU using a lifting wavelet algorithm," en *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, Sept 2015

"CUDA C Programming Guide." `http://docs.nvidia.com/cuda/`, 2015. [Online; accedido 20-06-2016]

DAUBECHIES Ingrid *et al.*, *Ten lectures on wavelets*, vol. 61. SIAM, 1992

DAUBECHIES Ingrid and Sweldens Wim, "Factoring wavelet transforms into lifting steps," *Journal of Fourier analysis and applications*, vol. 4, no. 3, pp. 247–269, 1998

DUVAL Laurent C, Røsten Tage, *et al.*, "Filter bank decomposition of seismic data with application to compression and denoising," en *2000 SEG Annual International Meeting Society Exploration Geophysicists*, pp. 2055–2058, Society of Exploration Geophysicists, 2000

FAJARDO Carlos, Villar Javier Castillo, and Pedraza César, "Reducción de los tiempos de cómputo de la migración sísmica usando FPGAs y GPGPUs: Un artículo de revisión," *Ingeniería y Ciencia*, vol. 9, no. 17, pp. 261–293, 2013

FAJARDO Carlos A., Reyes Oscar M., and Silva Ana Ramirez, "Seismic data compression using 2d lifting-wavelet algorithms," *Ingeniería y Ciencia | ing.cienc.*, vol. 11, no. 21, pp. 221–238, 2015

FAJARDO Carlos A., Angulo Carlos A., Mantilla Julián G., Obregón Iván, Castillo Javier, Pedraza César, and Reyes Óscar M., "Computational Architecture for Fast Seismic Data Transmission between CPU and FPGA using Data Compression," en *2016 Data Compression Conference*, (Salt Lake, United States.), 2016

HE Chuan, Lu Mi, and Sun C., "Accelerating seismic migration using FPGA-based coprocessor platform," en *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, pp. 207–216, April 2004

LERVIK John M, Rosten T, and Ramstad Tor A, "Subband Seismic Data Compression: Optimization and Evaluation," *Digital Signal Processing Workshop Proceedings*, no. 1, pp. 65–68, 1996

MENG Jiayuan, Tarjan David, and Skadron Kevin, "Dynamic warp subdivision for integrated branch and memory divergence tolerance," *SIGARCH Comput. Archit. News*, vol. 38, pp. 235–246, June 2010

REDDY T. Ashwini, Devi K. Renuka, and Gangashetty Suryakanth V., "Nonlinear principal component analysis for seismic data compression," *2012 1st International Conference on Recent Advances in Information Technology (RAIT)*, pp. 927–932, mar 2012

RØSTEN Tage, Ramstad Tor A., and Amundsen Lasse, "Optimization of sub-band coding method for seismic data compression," *Geophysical Prospecting*, vol. 52, no. 5, pp. 359–378, 2004

SALOMON David, *Data Compression: The Complete Reference*. Springer-Verlag London, 3 ed., 2007

SÁNCHEZ Fabián, Fajardo Carlos A., Angulo Carlos A., Reyes Óscar M, and Bouman Charles A, "A computational architecture for discrete wavelet transform using lifting scheme," en *2014 XIX Symposium on Image, Signal Processing and Artificial Vision*, IEEE, 2014

SHANNON C.E., "A mathematical theory of communication," *Bell System Technical Journal, The*, vol. 27, pp. 379–423, July 1948

SPANIAS A.S., Jonsson S.B., and Stearns S.D., "Transform coding algorithms for seismic data compression," en *Circuits and Systems, 1990., IEEE International Symposium on*, pp. 1573–1576 vol.2, May 1990

SWELDENS Wim, "Lifting scheme: a new philosophy in biorthogonal wavelet constructions," en *Proc. SPIE*, vol. 2569, pp. 68–79, 1995

UYTTERHOEVEN Geert, Roose Dirk, and Bultheel Adhemar, "Wavelet transforms using the lifting scheme," *ITA-Wavelets Report WP*, vol. 1, 1997

VASSILIOU Anthony A. and Wickerhouser Mladen V., "Comparison of wavelet image coding schemes for seismic data compression," en *Proc. SPIE*, vol. 3169, pp. 118–126, 1997

VILLASENOR J.D., Belzer B., and Liao J., "Wavelet filter evaluation for image compression," *Image Processing, IEEE Transactions on*, vol. 4, pp. 1053–1060, Aug 1995

VILLASENOR J.D., Ergas R.A., and Donoho P.L., "Seismic data compression using high-dimensional wavelet transforms," en *Data Compression Conference, 1996. DCC '96. Proceedings*, pp. 396–405, Mar 1996

WANG Yongzhong and Wu Ru-Shan, "Seismic data compression by an adaptive local cosine/sine transform and its effects on migration," *Geophysical Prospecting*, vol. 48, no. 6, pp. 1009–1031, 2000

WOOD Lawrence C, "Seismic data compression methods," *Geophysics*, vol. 39, no. 4, pp. 499–525, 1974

WU Wenbo, Yang Zhigao, Qin Qianqing, and Hu Fuxiang, "Adaptive seismic data compression using wavelet packets," en *2006 IEEE International Symposium on Geoscience and Remote Sensing. IGARSS 2006. IEEE International Conference on*, pp. 787–789, July 2006

XI-ZHEN Wang, Yun-tina Then, Meng-tan Gao, and Hui Jiang, "Seismic data compression based on integer wavelet transform," *Acta Seismologica Sinica*, vol. 17, no. 04, pp. 123–128, 2004

XIE Xing and Qin Qianqing, "Fast Lossless Compression of Seismic Floating-Point Data," *2009 International Forum on Information Technology and Applications*, pp. 235–238, May 2009

YU Wenmao, Xie Kai, and Bai Zhijun, "Fast seismic data compression based on high-efficiency SPIHT," *Electronics Letters*, vol. 50, no. 5, pp. 365–367, 2014

ZHENG Fan and Liu Shufen, "A fast compression algorithm for seismic data from noncable seismographs," en *Information and Communication Technologies (WICT), 2012 World Congress on*, pp. 1215–1219, Oct 2012

ANEXOS

A continuación se relacionan las publicaciones realizadas en el desarrollo del presente trabajo de investigación:

ANGULO Carlos A., Fajardo Carlos, Reyes Oscar, and Castillo Javier, "FPGA implementation of a Huffman decoder for high speed seismic data decompression," en *2014 Data Compression Conference*, (Salt Lake, United States.), p. 396, IEEE Comput. Soc, 2014

SÁNCHEZ Fabián, Fajardo Carlos A., Angulo Carlos A., Reyes Óscar M, and Bouman Charles A, "A computational architecture for discrete wavelet transform using lifting scheme," en *2014 XIX Symposium on Image, Signal Processing and Artificial Vision*, IEEE, 2014

ANGULO Carlos A., Hernández Christian, Rincon Gabriel, Boada Carlos, Castillo J., and Fajardo Carlos, "Accelerating Huffman decoding of seismic data on GPUs," en *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, (Bogot Colombia), Sept 2015

CASTELAR Jairo A., Angulo Carlos A., and Fajardo Carlos A., "Parallel decompression of seismic data on GPU using a lifting wavelet algorithm," en *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, Sept 2015

FAJARDO Carlos A., Angulo Carlos A., Mantilla Julián G., Obregón Iván, Castillo Javier, Pedraza César, and Reyes Óscar M., "Computational Architecture for Fast Seismic Data Transmission between CPU and FPGA using Data Compression," en *2016 Data Compression Conference*, (Salt Lake, United States.), 2016