Seismic Data Compression to reduce the PCIe Bandwidth Limitation

Carlos A. Fajardo

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2016

Seismic Data Compression to reduce the PCIe Bandwidth Limitation

Carlos A. Fajardo
Ingeniero Electrónico

Trabajo de investigación presentado para optar al título de
Doctor en Ingeniería

Director:
Javier Castillo
Doctor en Informática

Co-director:
Óscar M. Reyes
Doctor en Ingeniería

Universidad Industrial de Santander
Facultad de Ingenierías Físico Mecánicas
Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2016

To Margiory, who has been
the right companion for my journey

Acknowledgments

I have been extremely fortunate to work with Javier Castillo and Óscar Reyes. They gave me the support and guidance that I needed during my Ph.D. studies. Thanks very much, Javier and Oscar.

I am grateful to Charles Bouman to give me the chance to work with him at the Integrated Imaging Lab at Purdue University. My internship was an excellent experience, not only for me but also for my family. Thank you professor Bouman for everything.

The financial support for this research came from Colombian Oil Company – ECOPETROL – and – COLCIENCIAS. I also gratefully acknowledge CPS research group and Colombian Institute of Petroleum (ICP) for their constant support. Anita and Herling thank you so much.

I am also grateful to Carlos Angulo for his unconditional friendship and support. Thank you for your valuable contributions in the development of this research. I would like to thank my friends Yulieth and Paola, your friendship is a God's present for my life.

Various people at CPS research group have given me assistance. I thank Carlos Boada, Fabián Sánchez, Sergio Abreo, William Salamanca, Julián Mantilla, Iván Obregón, Manuél Pérez, Fabián Mantilla, Mayra Cárdenas, Jairo Castelar, Christián Hernández, Gabriel Rincón, Jheyston Omar, Juan S. López, Laurita.

No tengo palabras suficientes para decir gracias a mis padres, Felipe (Q.E.P:D) y Leonor, quienes con su apoyo, amor sacrificial y su ejemplo, me permitieron conseguir este importante logro. También agradezco a mi hermana Yidy Alexandra, quien ha sido un ejemplo para mi, por su fortaleza y su gran amor. A mis sobrinos Dani y Polito a quienes quiero mucho.

My daughter Isabella (Bella) and my son David Felipe (Pipe). They are my most important motivation to overcome the difficulties. They are the God's present to our marriage. They were

my perfect excuse to leave the lab early and to take holidays. They always remember me that the family and relationships are more important than research.

By far my most important support came from my wife. Margiory, you are the perfect companion that I have needed during my journey. You have been the best present that God has given me. I love you.

Finally, I thank God, who has made my dreams come true. I will be forever in debt with You.

Cafa

November 2016.

Contents

List of Figures

List of Tables

RESUMEN

TÍTULO: Compresión de datos sísmicos para reducir la limitación del ancho de banda del puerto PCIe*

AUTOR: Carlos Augusto Fajardo Ariza**

PALABRAS CLAVE: Compresión, Cuello de botella, Datos sísmicos, Entrada/Salida, FPGA, GPU, HPC, Huffman, Transformación Wavelet.

DESCRIPCIÓN:

Nosotros proponemos una estrategia para reducir el impacto del cuello de botella Entrada/Salida en un cluster heterogéneo, en el contexto de las aplicaciones sísmicas. La estrategia está basada en un proceso de compresión/descompresión optimizado. La estrategia comprime los datos en campo, mientras son adquiridos, usando un algoritmo de compresión optimizado. Las operaciones de transferencia desde la memoria principal hasta la memoria del nodo son ejecutadas usando los datos comprimidos para reducir el tiempo de transferencia. La descompresión de los datos es ejecutada dentro del nodo antes de que el dato sea procesado.

La estrategia se diseñó para dos tipos de clusters heterogeneos. El primer tipo de clúster usa GPUs y el segundo usa FPGAs. Por un lado, nuestros resultados muestran que las etapas secuenciales en el proceso de descompresión se convierten rápidamente en un cuello de botella en el cluster basado en GPUs. De otro lado, la implementación de la estrategia en un clúster basado en FPGAs, nos permitió proponer una arquitectura computacional específica, la cual se optimizó para las etapas secuenciales del proceso de descompresión.

La implementación de nuestra estrategia en un cluster con FPGAs puede acelerar el proceso de transferencia hasta $10\times$ para una relación de compresión de $16:1$ y hasta $3\times$ para una relación de compresión de $7:1$. Por consiguiente, nuestra estrategia efectivamente reduce el impacto del cuello de botella de Entrada/Salida de datos y puede mejorar el rendimiento general de un cluster basado en FPGAs.

---

* Trabajo de investigación.

** Facultad de ingenierías Fisico Mecánicas Doctorado en Ingeniería - Área Electrónica. Director: Javier Castillo.

ABSTRACT

TITLE: Seismic Data Compression to reduce the PCIe Bandwidth Limitation*

AUTOR: Carlos Augusto Fajardo Ariza**

KEYWORDS: Compression, FPGA, GPU, HPC, Huffman, Seismic data, Wavelet Transform.

DESCRIPTION:

We propose a strategy based on an optimized compression/decompression process to reduce the impact of the I/O bottleneck in a heterogeneous cluster, using seismic data as study case. Our strategy involves to compress the seismic data on-site while they are being acquired by a custom lossy compression algorithm. The transfer operations, from the *disk* to the *node memory*, are performed by using compressed data to reduce the I/O transfer time. Decompression occurs in the node before the data is processed.

We designed the strategy for two types of clusters widely used in computationally intensive algorithms, such as seismic applications. The first type of cluster is a GPU-based and the second one is an FPGA-based cluster. On the one hand, our results show that the use of a GPU-based cluster to perform the sequential stages of the decompression process generates a bottleneck because this architecture is not optimized for serial processes. On the other hand, the implementation of the strategy in an FPGA-based cluster allows us to propose a custom architecture, which is optimized for the sequential stages of the decompression process. This optimized architecture let us to overcome the bottleneck created by the sequential stages of the decompression process.

Our results show that the speedup in the transfer process (including the decompression process) strongly depends on the compression ratio: as the compression ratio increases, the speedup in the transfer process improves. The implementation of our strategy into an FPGA-based cluster can speed up the transfer operations up to 10× for a compression ratio of 16:1 and up to 3× for a compression ratio of 7:1. Therefore, our strategy effectively reduces the impact of the I/O bottleneck and can improve the cluster's overall performance.

* Trabajo de investigación.

** Facultad de ingenierías Físico Mecánicas Doctorado en Ingeniería - Área Electrónica. Director: Javier Castillo.

Introduction

This dissertation is motivated by the increasing need of computational power in the construction of subsurface images (i.e. seismic applications). These high-resolution images have demanded more computational power than has been available [3, 4, 5].

The oil and natural gas are currently the main energy sources around the world. About 55.9% of the total energy consumption comes from these primary energy sources [6]. Over the past 115 years, the oil companies have produced about one trillion of barrels of oil equivalent. Some conservatives projections suggest that in the next 20 years, the companies will have to find and produce another trillion of barrels of oil equivalent to supply the increasing demand [7].

The main technique used by oil companies to locate possible oil and gas reservoirs is the construction of subsurface images by seismic surveys. These images are interpreted by experienced geophysics and geologists, who establish whether there are reservoirs. However, the *easy* oil reservoirs around the world have now been located and exploited. These *easy* reservoirs were confined in relative unchallenging environments (e.g. shallow waters). Now, the companies have the challenge to find and produce oil and gas from *tough* areas with problematic geological formations, in deep waters or challenging environments [3, 8].

Motivation

The construction of subsurface images has motivated a considerable amount of research in the last years in the fields of both Geophysics and High Performance Computing (HPC). Several research works in HPC have aimed to reduce the processing time, which is now in the order of weeks or even months [3, 9, 10, 11, 12, 13].

Moreover, the localization of hydrocarbons in *tough* areas will require, among other things, to make subsurface images with a better resolution than those used in the past for the *easy* oil. The construction of these high-resolution images implies several computational challenges. For example, it will require acquiring and processing an enormous amount of seismic data, in the order of hundreds of Terabytes.

On the other hand, it is well-known that a significant challenge for any modern computational systems is the Input/Output bottleneck [14, 15, 16, 17]. This bottleneck arises in the construction of subsurface images because of the amount of required memory in a single node. For example, the implementation of seismic applications can easily require one terabyte of space in a single node. However, the currently available *on-chip* memory in one of these nodes (e.g. FPGAs and GPUs) is counted *only* in tens of Mega bytes [18, 19].

Consequently, when the seismic applications are implemented in heterogeneous clusters, it is required to make transferring operations between the *CPU main memory* and the *node memory* through the PCI Express bus. These transferring operations significantly reduce the overall performance, especially in applications that require transferring a considerable amount of data between the *CPU main memory* and the *node memory*. This penalty is due to the long latencies in *CPU main memory* and the low-speed in the PCIe bus (bandwidth). Thus, each time that the node requires making Input/Output operations on the *CPU main memory*, it has to stay idle waiting for data arrive. These idle waiting times generate a penalty on the overall performance of the computer system [16, 17, 20].

Thesis statement and Contributions

This dissertation presents a strategy to improve the transfer operations between the *disk* and the *node memory* through the PCI Express bus by using an optimized compression/decompression process, in the context of seismic applications.

The proposed strategy involves to compress the seismic data *on-site* while they are being acquired. Then, the compressed data are transferred to the head node (in the processing center), where they are sent to each *node memory*. Once the compressed data arrived at the node, they are decompressed before being processed.

It is important to highlight that we did not take into account the *compression time* because this process can be performed *on-site*. This compression process can be developed between

repetitions of the *seismic experiment* used by the oil companies to acquire the seismic data [3]. In our tests, compressing a seismic shot lasted a few tens of seconds. This time is assumable because it is quite small compared to the time between repetitions of the *seismic experiment*.

Figure 1 sketches a serial version of the time constrains in the proposed strategy. Note that, the strategy requires that the *compressed data transfer time* (t_1) plus the *decompression time* (t_2) has to be less than the *Traditional I/O transfer time* (t_{trad}):

$$t_1 + t_2 < t_{trad}. \tag{1}$$

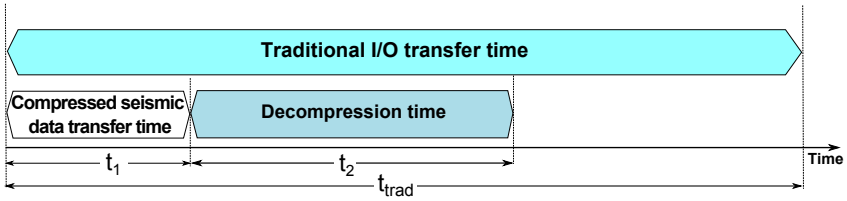


Figure 1: Time constrains in the proposed strategy (Serial version)

Compression algorithms have been traditionally designed to compress the seismic data as much as possible with no time constraints at all. In our case, however, we are limited by Equation 1.

As will be shown later, these algorithms seek to preserve the relevant geophysical information, to reduce the noise and to find *redundancy*. These three processes can not be implemented in just one stage. Thus, compressing seismic data requires several stages. On the other hand, the decompression algorithms perform the inverse process of each one of these stages. Achieving better compression ratios require to add new stages in the compression process or include new stages with major computational complexity. However, these new stages increasing the processing time.

Furthermore, the literature reports that better compression ratios are obtained by using some sequential stages. These stages make that the decompression of a single datum takes an enormous amount of processor cycles [21, 22, 23]. As a consequence of this trade-off between the compression ratio and the processing time, previous works have failed to speed up the transfer process by using a compression strategy [24, 25]. In these research works, when the sequential stages were implemented in parallel architectures, they quickly become the bottleneck in the decompression process.

Other research work overcame the sequential bottleneck by using a co-processing strategy (CPU + GPU) to reduce the impact of the I/O operations between the *disk* and a heterogeneous cluster [26]. The sequential stages were developed on CPU, and the parallel stages were developed on GPU. This co-processing strategy is not an option in our case, because we deal with the I/O bottleneck between the *disk* and the *node*. Therefore, we require to develop all the decompression process on the parallel architecture. Thus, one of the main challenges in this dissertation is to design and implement a compression-decompression strategy that has high performance in terms of both compression ratio (to reduce t_1) and decompression time (to reduce t_2) in a parallel architecture.

The proposed strategy was implemented in an FPGA-based cluster by using a custom architecture optimized for the sequential stages of the decompression process. The results showed that the proposed strategy reduces the impact of the I/O bottleneck in this type of cluster. The transfer operations are speedup up to 10× for a compression ratio of 16:1 and up to 3× for a compression ratio of 7:1. This speedup could improve the overall performance in the heterogeneous clusters, especially for those applications that are limited by I/O transfer operations.

Therefore, this thesis presents the first successfully implementation of a *compression-transfer-decompression* strategy to speedup the transfer process from the main memory to device memory in a heterogeneous cluster. Additionally, the thesis also presents the first computational architecture for a parallel Huffman decoding on an application whose dynamic range is unbounded.

Dissertation Outline

This dissertation is organized as follows: Chapter 1 gives a brief introduction about data compression theory to understand the terminology used in this dissertation. This chapter also presents an analysis of the most significant research works on seismic data compression and on those research works that have addressed the I/O issue by using a compression strategy.

Chapter 2 analyzes the transformation stage. This analysis aims to select a specific transformation scheme. This chapter also establishes some key parameters for the computational implementation of the transformation stage.

Chapters 3 and 4 describe the implementations of the decompression process in GPUs and FPGAs respectively. These chapters give details of the optimizations done in the decompression algorithms to improve its performance in both devices.

Chapter 5 presents and discuss the results of the proposed strategy. The conclusions and future work are drawn in chapter 6. Appendix 6 provides a description of the seismic datasets used in this dissertation. Finally, Appendix 6 lists the papers that have been published based on the research work performed in this dissertation.

Chapter 1

Seismic Data Compression

As mentioned, the oil companies make subsurface images to locate possible oil and gas reservoirs. The construction of these images requires seismic surveys over the area to explore. These surveys aim to collect the geophysical information by acquiring a huge amount of seismic data, which are processed to construct the subsurface image. These seismic data are stored in files up to hundreds of Terabytes. This amount of data demands a huge storage and transmission capacity. Hence, as the size of the seismic data has grown, the oil industry has been developing data compression algorithms to make the storage more efficient and to reduce both the transmission time and transmission costs.

In this chapter, we first give a brief introduction to data compression theory to understand the terminology used in this dissertation (for further details see [27, 28]). We also introduce the seismic data and present its main characteristics. Then, a review of the related work is presented, which is divided into two main parts: research works on seismic data compression and the efforts that have been done for reducing the I/O bottleneck by using a compression strategy. The chapter ends with a discussion about these research works.

1.1 Data compression

The compression process aims to reduce the number of bits per sample required to represent a given amount of information. Here is important to note that *information* and *data* are not synonymous. The data is the mean by which information is conveyed, thus we can use different data sizes to represent the same amount of information.

Let S be a string of n elements from an alphabet $\Sigma = \{a_1, a_2, \ldots a_\sigma\}$ of cardinality σ. Let

P_i be the probability of occurrences of a_i in S.

According to Shannon's source coding [29], S is optimally compressed by assigning to each symbol a code-word of length

$$\log_2 \frac{1}{P_i} \;=\; -\log_2 P_i \tag{1.1}$$

Thus, the string S can be compressed down to nH bits, on average, but no further, where

$$H \;=\; -\sum_{i=1}^{\sigma} P_i \;\log_2 P_i. \tag{1.2}$$

H is called the *entropy*, a quantity that determines the minimum average number of bits needed to represent each element in S [29]. In other words, it is required at least as many bits per element, on average, as the entropy to represent the string. The compressed string has the same quantity of information as the original, but represented with a fewer number of bits.

Note that the entropy is larger when all n probabilities are similar, and it is smaller as they become more and more different. This fact is used to define the *redundancy*, R, in the data, as

$$R \;=\; H_c - \left[-\sum_{i=1}^{\sigma} P_i \;\log_2 P_i\right] = H_c + \sum_{i=1}^{\sigma} P_i \;\log_2 P_i, \tag{1.3}$$

which indicates the difference between the current number of bits per symbol used (H_c) and the minimum number of bits per symbol required to represent the data set. In simple words, the redundancy is the amount of *wasted bits* –unnecessary bits– in the data representation. There are many strategies for data compression, but they are all based on the same strategy: reducing redundancy.

The compression algorithms examine the data to find redundancies and try to remove them. Finding redundancy can be viewed as a process of searching for patterns in the original data. Once these patterns have been identified, the compression process is achieved by merging or unifying these patterns.

Lossless and lossy compression

Lossless and lossy compression are terms that describe whether or not the original data is perfectly reconstructed. The lossless compression allows the perfect reconstruction from the compressed data, whereas in the lossy compression an approximation of the original data is reconstructed. Generally, lossy compression methods produce a much larger compression ratio than lossless compression ones.

In general, lossy compression algorithms have three stages: transformation, quantization and coding. The main difference between lossless and lossy compression methods is the quantization stage, which makes an approximation of the original data (e.g. floating-point data) to a set of integers. In others words, the quantization stage maps a *large* dataset on to a *small* one, which causes lossy compression. Figure 1.1 shows the lossy compression algorithms.

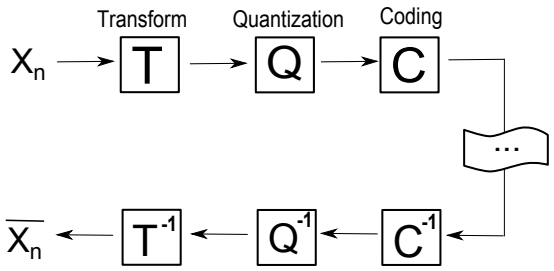


Figure 1.1: Lossy compression algorithms

Transformation stage

In the context of compression data, the transform stage is used to obtain *new* representation of the data, which has, on average, a smaller dynamic range than the original one. This reduction allows the representation of the transformed data using fewer bits, which improves the compression ratio.

In other words, this stage reduces the entropy (H) of the original data. As will shown in Chapter 2, this reduction is done by decorrelating the data, which allows a more compact representation in the transformed domain. Thus, the coefficients, in the transformed domain, are represented by a fewer number of bits.

Quantization stage

The transformation stage reduces the entropy of the seismic data, but no compression has occurred so far because the number of coefficients is the same as the number of samples in the original seismic data. In lossy compression algorithms, the stage that follows the transformation is the quantization, which makes an approximation of the floating-point transform coefficients in a set of integers. This stage reduces the dynamic range of the seismic data in some way.

In the quantization stage a floating-point number $x \in (a, b)$ is mapped into a finite set of

output levels $y_i, i = 1, 2, \ldots, N$, such that

$$Q(x) = y_i \tag{1.4}$$

Depending on how the levels y_i are distributed, the quantizer can be uniform (Figure 1.2a) or non-uniform (Figure 1.2b).

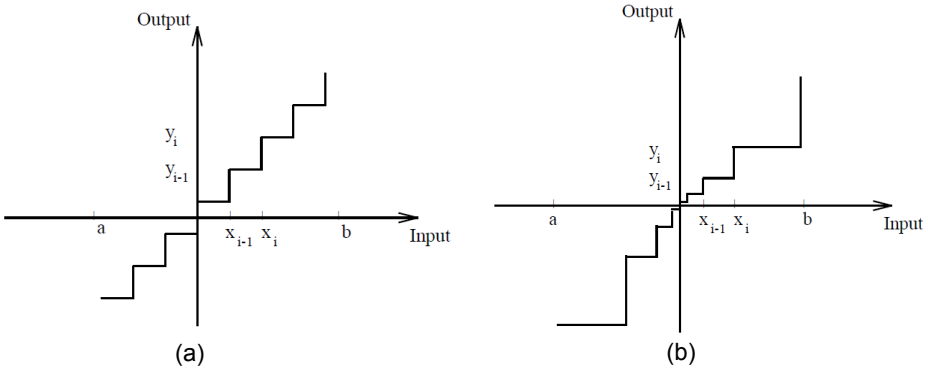


Figure 1.2: Uniform and non-uniform quantization [1]

A set the data $\mathbf{x}$ can be uniformly quantized by using

$$y_i = \text{round}\left[(x - \min(\mathbf{x})) * \frac{2^n - 1}{\max(\mathbf{x})}\right], \tag{1.5}$$

Where $\max(\mathbf{x})$ and $\min(\mathbf{x})$ are the maximum and minimum values of the input $\mathbf{x}$ respectively, and n is the number of bits used for the quantification.

The inverse process proceeds as follows:

$$\tilde{x} = y_i \frac{\max(\mathbf{x})}{2^n - 1} + \min(\mathbf{x}) \tag{1.6}$$

In a uniform quantization (Figure 1.2a), the number of quantization bits determines the number of output levels, which is 2^n. Thus, as the number of output levels is reduced the possibility to compress the data increases.

It is in this step where the *lossy* part of the compression process occurs because of the approximation process. Thus, after the quantization stage, a reduction in the SNR occurs.

Coding stage

The coding methods assign shorter code-words to the more frequent symbols and longer code-words to the less frequent symbols (Equation 1.1). The compression is achieved by replacing all characters from the input sequence with codes of smaller expected length.

Huffman [30] introduced a coding method, which assigns code-words of integer variable length. This coding method is optimal when the probabilities are integral powers of 2, which is rarely true for real data. However, Huffman is the best among the methods that use code-words of integer length. In 1978, it was shown by Gallager [31] that the maximum difference between the expected code length and the optimum is bounded by

$$p_m + \log_2 \frac{2 \log_2 e}{e} \approx p_m + 0.086 \tag{1.7}$$

Equation 1.1 suggest that we should assign code-words of non-integer length to achieve the best compression. The Arithmetic coding offers such a solution [32]. This method can be seen as a generalization of Huffman coding in which probabilities are not constrained to be integral powers of 2 and code-word lengths need not be integers. in the worst case, only 0.006 bits per symbol worse than the optimum [33].

Measure of compression quality

The performance of a compression algorithm can be measured by using various criteria, which depend on the nature of the application.

For lossless compression, common quantitative measures to express the performance of a compression method involve a comparison between the size of the input file before compression and the size of the output file after compression. Three of these measures are: the *Compression Ratio* (CR), the *Compression Factor* (CF), and the *Saving percentage* (SP). These quantities are defined as:

$$CR = \frac{\text{Number of bits before compression}}{\text{Number of bits after compression}} \tag{1.8}$$

$$CF = \frac{\text{Number of bits after compression}}{\text{Number of bits before compression}} \tag{1.9}$$

$$SP = \frac{\text{Number of bits before compression} - \text{Number of bits after compression}}{\text{Number of bits before compression}} \tag{1.10}$$

In lossy compression, the quality of reconstructed data is commonly measured by the *signal to noise ratio* (SNR), the *Peak signal to noise ratio* ($PSNR$) and the *Root Mean Squared Error* ($RMSE$), which are defined as:

$$SNR_{dB} = 10\, log_{10} \frac{\sum_{i=1}^{n} {S_i}^2}{\sum_{i=1}^{n} (S_i - \tilde{S}_i)^2} \tag{1.11}$$

$$PSNR_{dB} = 10 \log_{10} \frac{\max |S_i|^2}{\sum_{i=1}^{n} (S_i - \tilde{S}_i)^2} \tag{1.12}$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (S_i - \tilde{S}_i)^2} \tag{1.13}$$

where S_i is the original data and $\tilde{S}_i$ is the decompressed data.

1.2 Seismic Data

The construction of a subsurface image is done by a seismic survey over the area to explore. These surveys start with the localization of an array of sensors on the area to explore. Then a

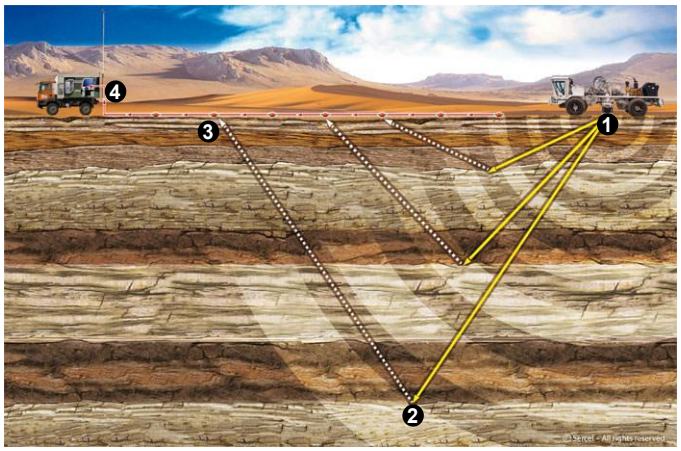


Figure 1.3: Seismic Acquisition [2]

low-frequency artificial seismic source is activated (e.g. an explosive charge). This *shot* creates a downward propagating wave-field (step 1 in Figure 1.3). Then, this wave-field is reflected from the geological boundaries (step 2 in Figure 1.3). Finally, the reflected wave-field is detected by

the sensors (step 3 in Figure 1.3) and recorded by a system recorder (step 4 in Figure 1.3). The recorded wave-field contains geophysical information, which is of interest to Geoscientists.

The recorded curve from a single sensor is called *seismic trace*. A section of a *seismic trace* is shown in Figure 1.4. Note that, the positive area of the curve is shaded black, while the negative side is left unfilled. This format is commonly used to enhance the visual display [34].

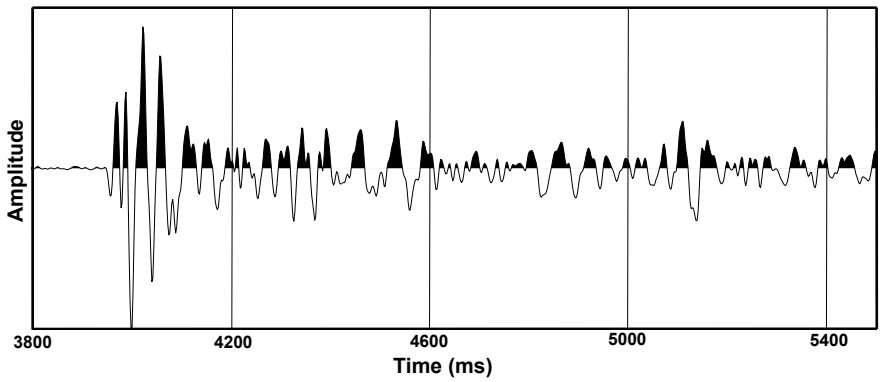


Figure 1.4: Section of a seismic trace

Figure 1.5 shows a section of a *seismic shot*, i.e. the reflections recorded by all sensors during a *shot*. Note that, the shaded area, in this case, is the right side of the curve. These *seismic shots* are stored by using different formats [35]. One of the most used is the SEG-Y format, which was developed by the Society of Exploration Geophysicists (SEG) [36]. This format adds a header to the seismic trace samples, which contains information related to the survey (e.g. shot number, source and receiver location, sampling frequency, etc.). In this dissertation we used 12 different *seismic shots*, which were provided by Ecopetrol Oil Company. For further details about these *seismic shots* see Appendix 6

Seismic trace samples are represented by using single-precision floating-point format. This format uses 32 bits per symbol, which is mandatory for processing purposes. However, from the Information Theory point of view, this format is longer than absolutely necessary, because the entropy of the seismic data (H) is small than 32 bits per symbol. In other words, this floating-point representation has redundancy.

Therefore, the seismic data can be considered as a combination of three types of components: geophysical information, redundancy in the signal representation, and uncorrelated and

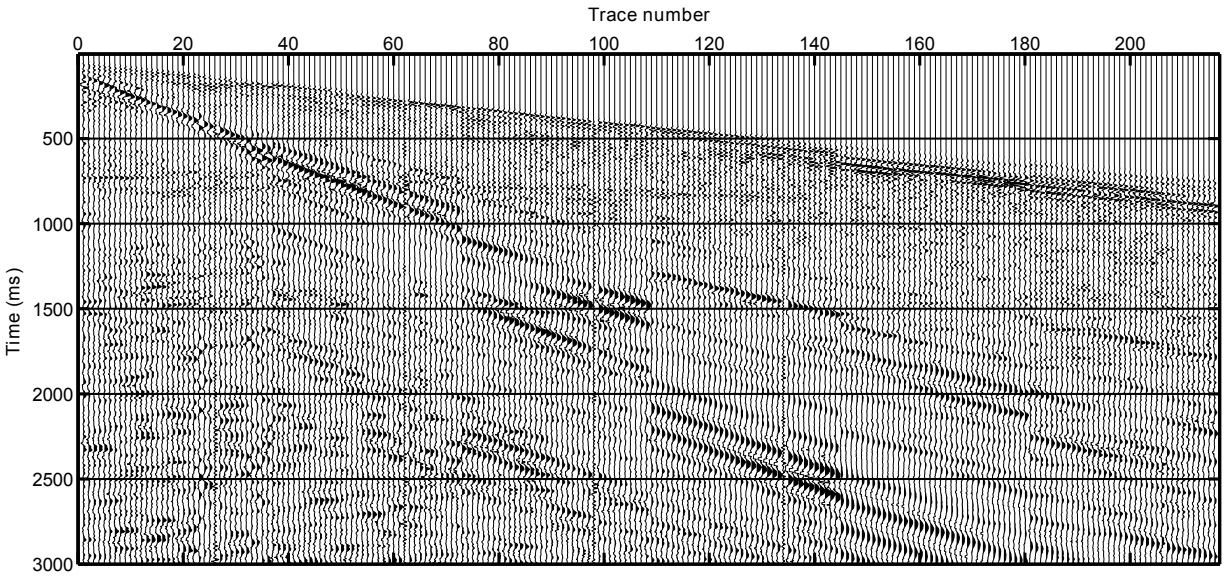


Figure 1.5: Shot Gather Section

broadband noise from different types of sources [37],

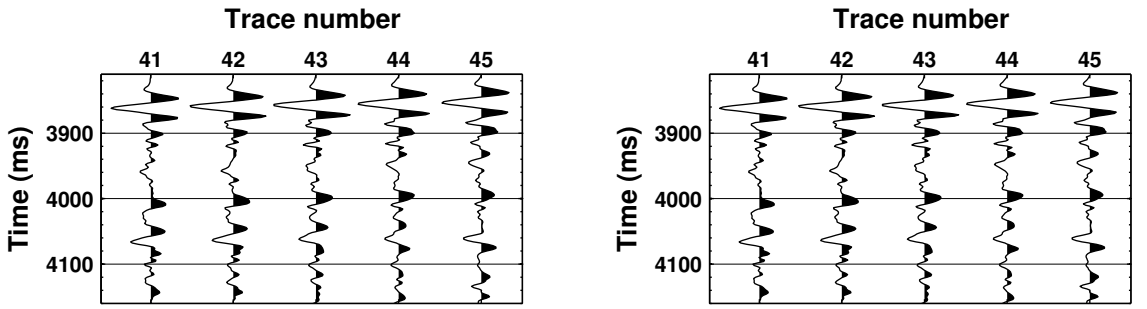
$$\text{Seismic trace} = \text{Information} + \text{Redundancy} + \text{Noise} \tag{1.14}$$

Seismic data compression algorithms seek to preserve the relevant geophysical information, to reduce the noise and to find redundancy. These algorithms aim to compress the seismic data as much as possible with a SNR above 40 dBs. This level of SNR is required to ensure that the compression-decompression process is not a noise source for the later processing such as Stacking or Migration [38, 39, 40, 41].

Figure 1.6 compares the decompressed version of our seismic shot 1 with the original one, when it has been compressed at 40 dB of SNR. Note that no artifacts are observed in the time domain. In the frequency domain (Figure 1.7), some changes are observed in the phase for high frequencies. However, the relevant geophysical information is concentrated under 100 Hz.

Limits on Seismic Data Compression

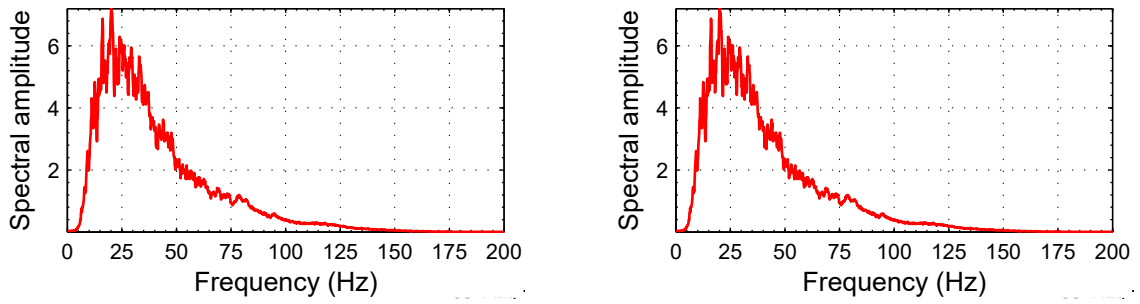
Compressing seismic data requires lossy compression methods to reach ratios greater than $2:1$. In other cases, for example, text compression (such as seismic trace headers), a higher compression ratio can be achieved by using lossless methods. These compression ratios are higher since the text files have frequently repeated symbols, while a seismic data is far less structured.



(a) Details of original traces

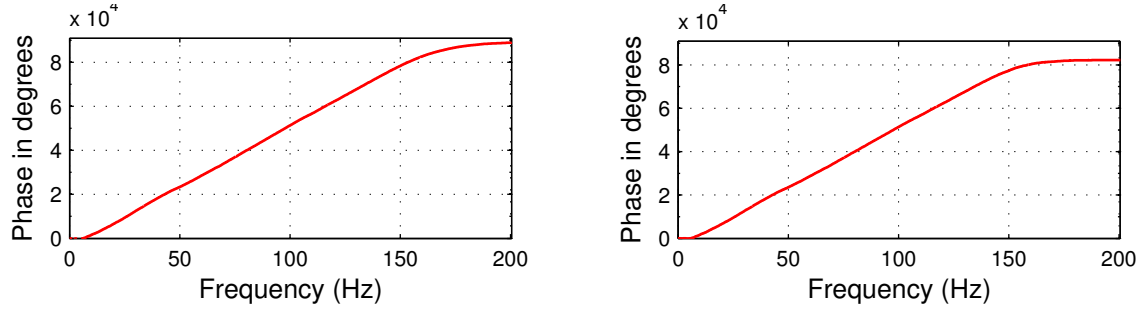
(b) Details of Decompressed traces

Figure 1.6: Seismic traces of shot 1



(a) Amplitude spectrum - Original traces

(b) Amplitude spectrum - Decompressed Traces



(c) Phase spectrum - Original traces

(d) Phase spectrum - Decompressed Traces

Figure 1.7: Spectrum of shot 1

1.3 Related Work

This dissertation is related to two main fields of knowledge: seismic data compression and I/O bottleneck. These issues have been well studied and addressed from many points of view. The first step in the development of this dissertation was a critical analysis of the most relevant research works in both fields. The results of this review are summarized in this section. The first part is dedicated to seismic data compression. The second part focus on those research works

that have addressed the I/O bottleneck by a compression strategy.

Seismic data compression

In the 1980s and 1990s decades, much of the original work in lossy compression were developed in the area of image compression [42, 43, 44, 45]. It was in the mid-1980s when the well-known JPEG method was development by *Joint Photographic Experts Group*, which later (1992) became a standard in lossy image compression. This standard is based on the cosine transform followed by a quantization scheme that uses *special* matrices (e.g. Losheller Matrix [46]), and a coding scheme composed by the *Run Length Encoding* and the *Huffman Encoding* [47]. However, this standard is not adequate for seismic data compression because it introduces artifacts even at low compression ratios [48].

In the early 1990s, Spanias *et al.*[49] stated that the Karhunen-Loeve Transform (KLT) [50] offers a better performance in terms of *data rates* vs. *normalized error*, when it is compared with other transform-based algorithms such as: Discrete Fourier Transform (DFT), Discrete Cosine Transform and Walsh-Hadamard Transform (WHT). However, the KLT has a high computational cost, which makes it impractical, especially for applications with time constrains.

It was in the 1990s decade when the Wavelet-based compression algorithms were a very active research area, especially in the field of image compression [51, 52, 53, 54]. Several wavelet-based algorithm have been proposed to compress seismic data [48, 41, 55, 56, 40]. The main change among these algorithms is the way in which the Discrete Wavelet Transform (DWT) is implemented. Several *flavors* of the DWT have been tested by changing parameters such as: the dimension (1D, 2D and 3D), the filter type and length and, the number of levels of decomposition, among others. These Wavelet-based algorithms showed a better performance over those based in the *traditional* transformations, because of the capabilities of the Wavelets on representing geophysical events in seismic data.

In the 2000s, a *new* series of transformations were successfully used in seismic data compression [57, 58, 59, 60, 61, 38, 62, 63]. These *new* transformations are General Filter Banks, which give an extra freedom in selecting the appropriate frequency decomposition. This freedom is used to suit the filter bank to seismic data, which improves the performance over Wavelet-based algorithms in terms of SNR at the same compression ratio.

In two works in 2001 [64] and 2002 [38], Duval and *et al.* presented algorithms for compression and denoising simultaneously. In these papers are shown how the Wavelet-based algorithms or

filters banks could be used for denoising and compression at a time. They used the *shrinkage* or *wavelet thresholding* to show how the lossy compression algorithms could be seen as a denoising tool, instead a source of noise. The core idea is to discard some coefficients termed as noisy by using one or several soft- or hard-thresholds.

Other research work has proposed to use *Principal Component Analysis* (PCA) to compress seismic data [65]. The limitation of this algorithm is the high computational cost to calculate the PCA.

Most recently, two compression algorithms based on the DWT and the *Embedded Zerotree Wavelet* [66] have been proposed. The algorithms classify and group the Wavelet coefficients according to specific patterns prior to quantizing and coding. These groups have a reduced entropy, which improves the compression ratio. However, the addition of the *Zerotree Wavelet* stage increases the computational complexity of the compression algorithm.

A uniform quantizer is generally used for seismic data compression [23, 49, 40, 63, 67, 68] because the larger errors in the non-uniform quantization scheme are concentrated in larger amplitudes, and usually, the larger amplitudes contain the relevant geophysical information. On the other hand, the small amplitudes data have a good chance to be noise.

In seismic data compression the most used coding schemes have been Huffman [69, 55, 48] and Arithmetic Coding [68, 63, 67, 39]. Sometimes a Run Length Encoding (RLE) [27] has been used prior to the entropy coding stage [68, 48].

The Huffman coding has been investigated during the last years [70, 71, 72, 73], these research works describe methods of storing and maintaining the Huffman tree. On the other hand, the Arithmetic coding has also been investigated during the last years [74, 75, 76, 77, 78], these research works deals with its high computational cost and how to store in an efficient way the changing probabilities of symbol occurrences.

Facing the I/O bottleneck by using a compression strategy

In the context of the heterogeneous cluster, the I/O issue has been well studied and addressed from many points of view. Despite all research in this area, the I/O latency continues to affect the overall performance of the modern computational systems.

The main strategy to reduce the impact of the I/O bottleneck has been the use of memory hierarchies. The goal in the designing of these memory hierarchies has been to provide memory systems that enable a fast access to the recently used data.

The reorganization of the stencil calculations to take advantage of the memory hierarchies has been the subject of much investigation over the years [79, 80, 81]. Other strategies include, for example, to hide memory latencies by switching between threads (on GPUs) or cores (on FPGAs). thus, while a thread (or core) is making transfer operations, some other one can be scheduled in its place [82, 83, 84].

In this review, we focus on those research works that have addressed this problem by using a compression strategy.

To the best of our knowledge, the first work that using a compression strategy, trying to reduce the I/O bottleneck, it was developed by Haugen in 2009 [20]. In his work, Haugen investigated the feasibility of using a lossy compression algorithm to improve the bandwidth between CPU and GPU through the PCIe bus. The strategy consists of compressing the seismic data before GPU transfer, and it was tested in a GPU-based cluster. Haugen used the compression algorithm developed by Rosten in 2000 [37], which is based on three stages: a filter bank, a uniform quantization and finally, a coding scheme formed by Huffman and RLE.

The strategy did not speed up the I/O operations between the CPU and GPU. The results showed that the transfer time by using the compression strategy was higher than the transfer time without compression. However, they stated that the strategy could speed-up the transfer speed over other slower buses, for example, over networks connections of $100Mbit/s$.

One of the major drawbacks of this work was how the decoding process was implemented. Huffman decoding has the highest computational cost in this algorithm, because of the variable length of its code-words [22, 85, 86]. The Huffman algorithm was implemented by porting a serial version, but not optimized to be implemented in parallel architectures. For this reason, the decoding process became the bottleneck of the decompression process.

In 2011, Aqrawi *et al.* [26] investigated how to reduce the limitations of disk I/O using a compression strategy. They aimed to improve these I/O limitations in a GPU-Based computational system, and tested the strategy using both HDDs (Hard disk drives) and SSDs (Solid state disks). The application that motivated this research was a *seismic filtering*, which is a process used to analyze the subsurface images. In this algorithm, the I/O operations spent most of the execution time. Their results show that only the 10% of the execution time was used to perform the filtering process itself, while the rest of time (90%) was spent in the transfer process.

They tested both lossless and lossy compression algorithms. On the one hand, the lossless algorithms were based on a modified RLE (Run Length Encoding) and Huffman Encoding. They

do not give specific details for their GPU implementation neither RLE nor Huffman coding. The compression ratios obtained by these lossless compression methods were 1.2 and 1.4 for RLE and Huffman respectively. These methods showed up to 1.4× faster than the traditional I/O transfer process for HDD disks, but a negative speed up for SSD disks.

On the other hand, the lossy compression schemes were based on the DCT (Discrete Cosine Transform) and the LOT (Lapped Orthogonal Transform). For the lossy compression methods, they tested both transformations (DCT and LOT) in several dimensions. The best compression ratio was 6.5, and it was achieved by using the $3D$-DCT and the modified RLE. These lossy methods showed up to 6× and 3.2× faster than the traditional I/O transfer process for HDD and SDD disks respectively.

It is important remark that both RLE and Huffman algorithms showed a lower performance on GPU than on CPU, due to its sequential nature. For this reason, these algorithms were finally implemented on CPU. On the other hand, the transform stage was performed on GPU because of its parallel nature.

This work differs from our work because the implementation used both the CPU and GPU capabilities to develop the compression/decompression process. By using the CPU capabilities to perform sequential algorithms, it was possible overcome the bottleneck caused by the sequential nature of the RLE and Huffman lossless compression methods. In our case, we need to perform the entire algorithm in a parallel architecture.

Then in 2012, Patel *et al.*[25] implemented the *bzip2* lossless algorithm on GPU to compress text. They stated that one motivation for their work it was to determine *whether on-the-fly compression is suitable for optimizing data transfer between CPU and GPU*. The algorithm consists of three stages: Burrows-Wheeler Transform(BWT) [87], Move-to-Front Transform (MFT) [88] and Huffman Coding.

The BWT is a particular case of string sort, which was implemented by using a sorting algorithm based on *merge sort* [89]. They did not give details for the reverse BWT implementation.

On the other hand, when MTF is applied to a string that has been transformed by BWT, the overall entropy is reduced [88]. The MTF was parallelized by using a *divide-and-conquer* strategy. They broke the input string in substrings and applied the MTF to these substrings. Then, the two partial transformed substrings are used to create a new *substring*, which, in turn, is transformed in a new substring, and so on. They left the parallel implementation of the reverse MTF for future work.

Finally, they parallelized the Huffman encoding by assigning a specific number of codes to each block on the GPU. In this case, they assigned 4096 codes to thread-block, where each block has 128 threads. The decoding process was parallelized by assigning each bit array to a thread in the GPU.

In the implementation, the string sort in the BWT and Huffman stages were the bottleneck in the implementation. They concluded that the implementation was not fast enough to optimize data transfers between CPU and GPU.

1.4 Discussion

Different algorithms have been proposed to compress seismic data. They have been focused on achieving higher compression ratios at a quality around 40 dB in terms of SNR. This level of SNR guarantees that later processing, such as stacking or migration, does not introduce errors caused by the compression process. Additionally, some works have stated that the compression process can be considered as a denoising tool instead of a noise source.

In the literature, we did not find a compression standard or a universal rule on seismic data compression. However, we found useful guidelines for the selection of the compression strategy, especially if we take into account the computational complexity of the algorithms used.

Regarding the quantization stage, there seems to be a consensus about the use of a uniform quantizer (Section 1.3). On the one hand, this quantizer minimizes the entropy for a fixed SNR [90]. Therefore, if the uniform quantizer is followed by an entropy coder, such as Huffman or Arithmetic, it will achieve the best compression ratio for a given SNR. On the other hand, the larger errors in the *non-uniform* quantizer are concentrated in larger amplitudes, and usually, the larger amplitudes contain the relevant geophysical information [90, 1].

Relating to the coding schemes, Huffman and Arithmetic have been the most used on seismic data compression, thanks to its performance in terms of compression ratio and its relatively low computational cost.

We tested both coding schemes with our seismic datasets. Figure 1.8 shows the performance in three of our datasets, in terms of compression ratio vs. SNR. In this case, we used a Wavelet-based compression algorithm with a uniform quantization (from 6 to 14 bits). Figure 1.8 also shows the optimum compression ratio, which is calculated by using the entropy of the quantized wavelet coefficients (Equation 1.2).

The Arithmetic coding exhibited a superior performance for low levels of SNR and was very close to the optimum compression ratio. For values of SNR around 40 dB, both Huffman and Arithmetic showed similar performance, which was very close to the optimum.

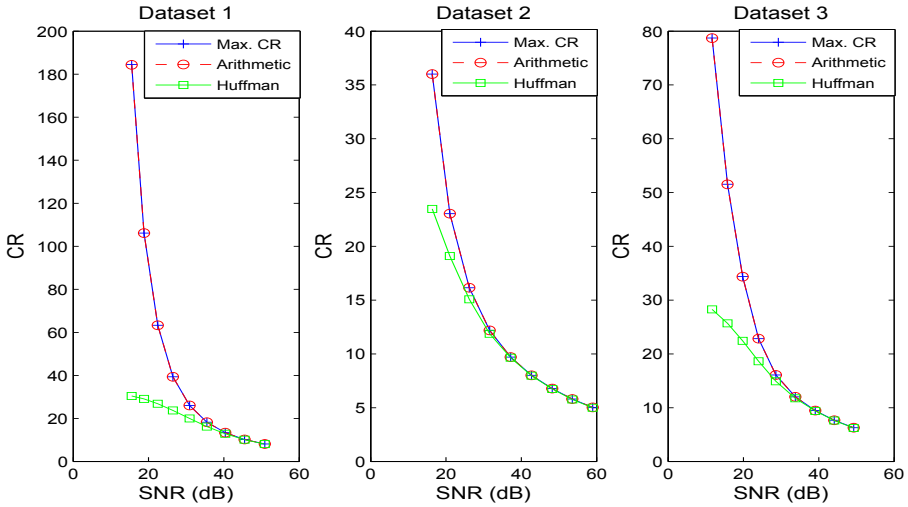


Figure 1.8: SNR vs CR for Huffman and arithmetic coding. Figure also shows the maximum CR achievable.

It is important to remark that the Arithmetic coding has a higher computational complexity than the Huffman coding. From the computational complexity perspective, the Huffman coding is a better option, when the aim is to achieve an SNR around 40 dB.

Regarding the transformation stage, its selection remains an open problem for the seismic data compression [91]. We analyze this stage in Chapter 2.

Chapter 2

Decorrelating seismic data

We aim to develop a compression strategy which has a high performance in terms of compression ratio and decompression time. The transformation stage plays an important role regarding these parameters because the transformation can improve the compression ratio and its computational cost affects the decompression time.

Several transformations have been used to compress seismic data (see section 1.3). However, the main objective of these works has been to compress seismic data as much as possible without time constrains. Thus, it is necessary to develop an analysis that also includes time constrains that are imposed by the proposed strategy.

In this chapter, we analyze the transformation stage. The analysis aims to select the transformation and to establish key parameters at the moment of implementing the whole transformation stage. The analysis is performed from both *compression ratio* and *decompression time* points of view.

2.1 Seismic Data Decorrelation

Seismic data compression aims to represent the seismic data in a way that requires fewer bits than the *original* way. This process has been achieved successfully because the seismic data have a form or structure, which is predictable. In other words, the seismic data have redundancy (see Section 1.1). Figure 2.1 shows a section of the Dataset 1 (Shot 1). Note that, there is a correlation between neighboring data, both in the vertical direction (time) and horizontal direction (space).

To illustrate this correlation, Figure 2.2 shows the *dispersion diagram* of pairs of adjacent seismic samples (in vertical direction). Each (x, y)-point is conformed by a sampled data and its

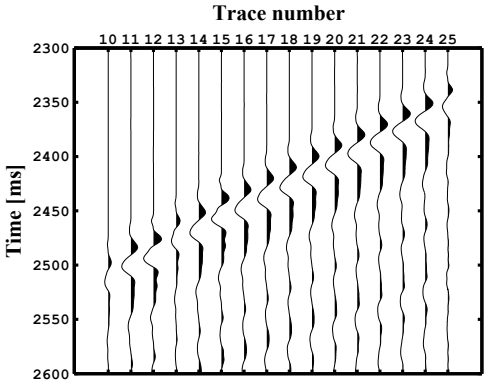


Figure 2.1: Shot Gather Section

neighbor. Note the relationship about the $y = x$ line, which illustrates the correlation between neighbor data samples.

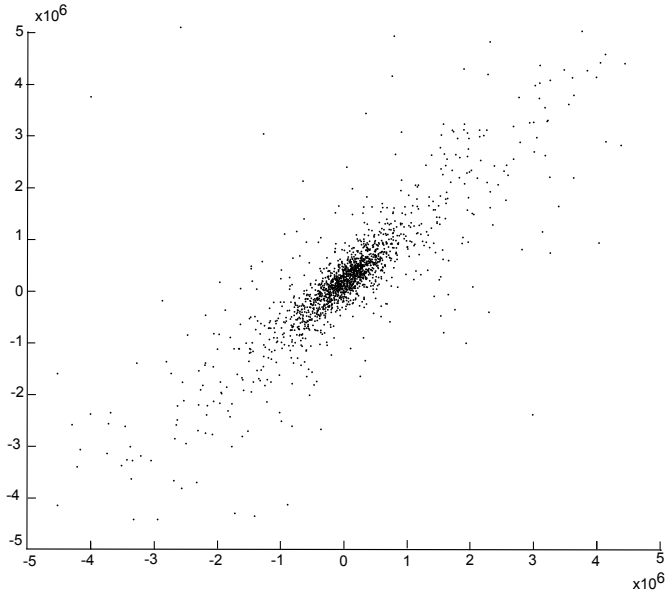


Figure 2.2: Adjacent seismic data

By rotating by 45° about the center the dispersion diagram shown in Figure 2.2, we can obtain a *new* dispersion diagram, as shown in Figure 2.3. Note that, this *new* distribution has a smaller dynamic range than the original one, and thus, we can encode these values with fewer bits.

Figure 2.4 shows the histograms obtained from a seismic data set before and after the rotation

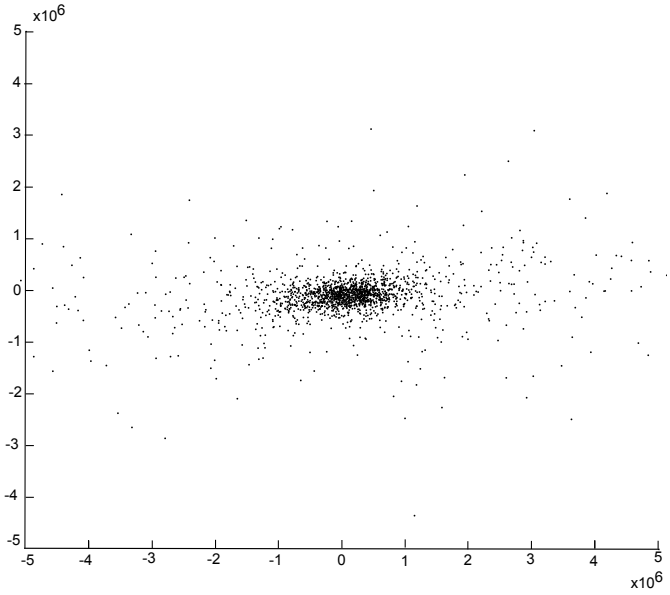


Figure 2.3: Rotation Decorrelation (Adjacent data)

respectively. The rotated data has a narrower distribution and therefore a better chance to reach an improved compression ratio.

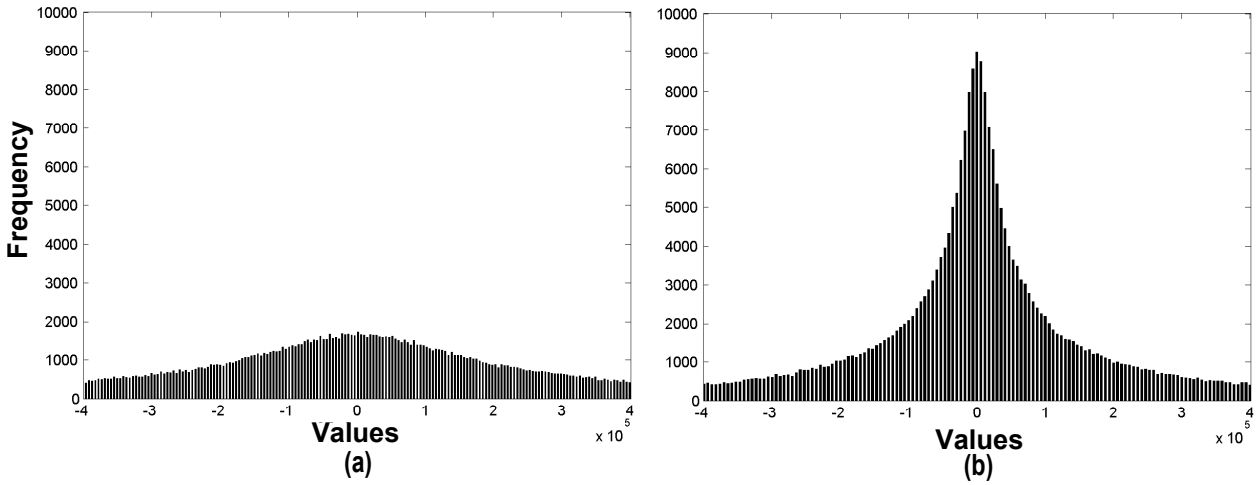


Figure 2.4: Histogram for a seismic data set. (a) Histogram of the data values of the seismic data. (b) Histogram of the rotated data.

We can analyze these dispersion diagrams from a statistical point of view. The dispersion diagram shown in Figure 2.2 has a positive covariance ($\sigma_{xy} > 0$), because of the lineal relationship

between neighbor sampled data. On the other hand, the dispersion diagram shown in Figure 2.3 has a covariance close to zero ($\sigma_{xy} \approx 0$) [92]. Thus, the rotation has decorrelated the data, i.e. the value of the first component gives few information to the second component. This decorrelation allows the reduction on both the dynamic range of the data and entropy (H).

Optimal decorrelation

We may define a seismic shot as a matrix $[\mathbf{X}]$ with N vectors, where each column-vector $\mathbf{x}_j$ is a seismic trace, i.e.

$$[\mathbf{X}] = [\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3\ \cdots\ \mathbf{x}_N] \tag{2.1}$$

In general, this seismic shot is decorrelated when the covariance between two different traces is zero, i.e.,

$$\mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 0 & i \neq j \\ \sigma^2 & i = j \end{cases} \tag{2.2}$$

Where ${\sigma_j}^2$ is the variance of $\mathbf{x}_j$. Thus, the covariance matrix $C_{n\times n}$ of the decorrelated process must be diagonal, such that:

$$C_{n\times n} = \begin{bmatrix} \mathrm{Cov}(\mathbf{x}_1,\mathbf{x}_1) & \mathrm{Cov}(\mathbf{x}_1,\mathbf{x}_2) & \dots & \mathrm{Cov}(\mathbf{x}_1,\mathbf{x}_n) \\ \mathrm{Cov}(\mathbf{x}_2,\mathbf{x}_1) & \mathrm{Cov}(\mathbf{x}_2,\mathbf{x}_2) & \dots & \mathrm{Cov}(\mathbf{x}_2,\mathbf{x}_n) \\ \vdots & & & \vdots \\ \mathrm{Cov}(\mathbf{x}_n,\mathbf{x}_1) & \mathrm{Cov}(\mathbf{x}_n,\mathbf{x}_2) & \dots & \mathrm{Cov}(\mathbf{x}_n,\mathbf{x}_n) \end{bmatrix} = \begin{bmatrix} {\sigma_1}^2 & 0 & \dots & 0 \\ 0 & {\sigma_2}^2 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & {\sigma_n}^2 \end{bmatrix} \tag{2.3}$$

The optimal transformation: *Karhunen-Loève Transform (KLT)*

It is possible to represent the transformation stage as a matrix operation as

$$\mathbf{Y} = [\mathbf{B}][\mathbf{X}] \tag{2.4}$$

Where $[\mathbf{Y}]$ is the transformed data, $[\mathbf{B}]$ is the linear transformation matrix, which is a basis for the new space and, $[\mathbf{X}]$ is the original seismic data. Alternatively, each transform column vector, $\mathbf{y}_i$ of $[\mathbf{Y}]$, can be calculated as

$$\mathbf{y}_i = [\mathbf{B}]\mathbf{x}_i \tag{2.5}$$

If it is assumed that [**B**] is an orthonormal basis, i.e. $[\mathbf{B}][\mathbf{B}]^T = [\mathbf{I}]$, which can be achieved by using a Gram-Schmidt process [93], the total energy under the transformation is preserved, i.e. ,

$$\begin{aligned} \|\mathbf{y}_i\|^2 &= {\mathbf{y}_i}^T \mathbf{y}_i \\ &= ([\mathbf{B}]\mathbf{x}_i)^T([\mathbf{B}]\mathbf{x}_i) \\ &= {\mathbf{x}_i}^T [\mathbf{B}] [\mathbf{B}]^T \mathbf{x}_i \\ &= {\mathbf{x}_i}^T \mathbf{x}_i \\ &= \|\mathbf{x}_i\|^2 \end{aligned} \tag{2.6}$$

For optimal decorrelation, it is required to find a linear transformation matrix, [**B**], which produces a diagonal covariance matrix for the transformed data.

If we assume that [**B**] is an orthonormal basis, the desired matrix $[C]_Y$ can be calculated from the original covariance matrix $[C]_X$ [50], as,

$$[C]_X[\mathbf{B}] = [\mathbf{B}][C]_Y \tag{2.7}$$

Since $[C]_Y$ is diagonal, Equation 2.7 can be re-written for each column vector as

$$[C]_X\mathbf{b}_j = \lambda_j\mathbf{b}_j \tag{2.8}$$

Where, the column vectors $\mathbf{b}_j$ are the basis of the new representation and the coefficients λ_j are the covariances of the transformed data (See equation 2.3).

Note that, Equation 2.8 is the *eigenvalue problem* [93], where the column vectors $\mathbf{b}_j$ and the coefficients λ_j are the eigenvectors and eigenvalues respectively of the covariance matrix, $[C]_X$, of the original data.

The transformation defined by the eigenvalues of the covariance matrix is the *Karhunen-Loève transform* (KLT) [50]. This method is also referred to as the *Hotelling Transform* or *Principal Components Analysis* (PCA) [94].

Thus, the KLT is the optimal transform for Gaussian data, in the sense of the decorrelation [95]. This transformation offers a *natural* basis, which allows an optimal decorrelation of the data. However, this transformation is computationally expensive and sometimes could be impractical because of the basis vectors are not constant but data dependent [49, 65, 96].

2.2 KLT vs. Discrete Wavelet Transform (DWT)

We are interested in testing the Discrete Wavelet Transform (DWT) in this dissertation because it offers some implementation alternatives that could help us to reduce the decompression time (see Chapter 3).

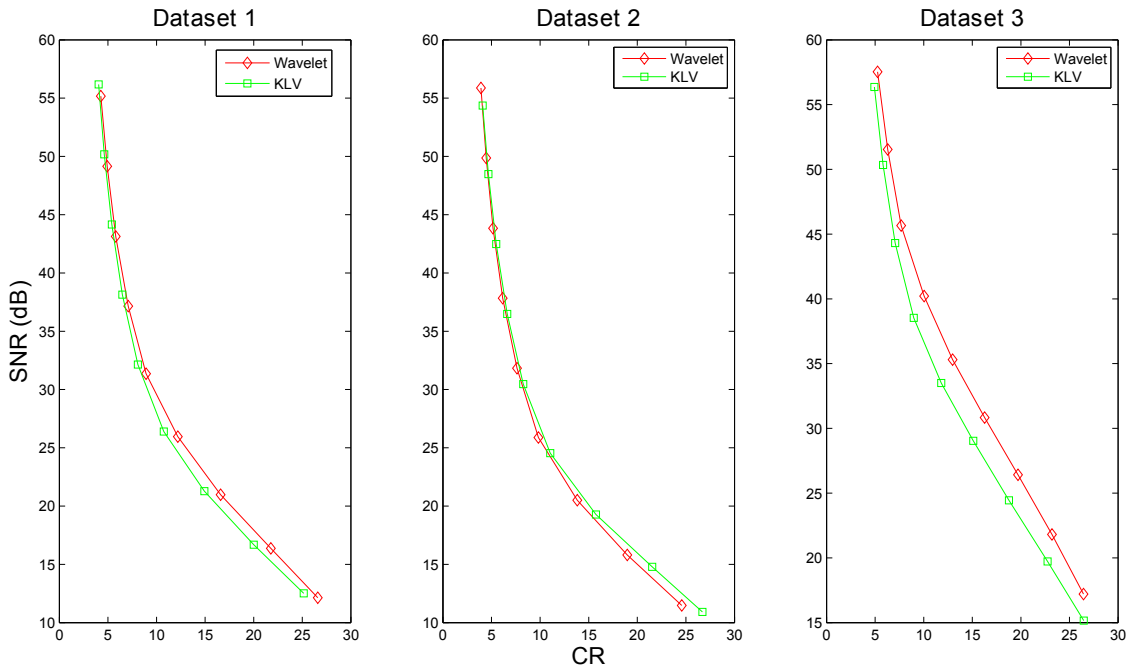


Figure 2.5: KLT vs. Discrete Wavelet Transform

To establish the performance of the DWT, we compare a KLT-based algorithm with a DWT-based algorithm, which is based on the CDF 2.2 filter (see section 2.4). The algorithms are the same except for the transformation stage. Figure 2.5 compares the performance of both algorithms in terms of SNR (in dBs) vs. Compression Ratio (CR) in three of our datasets. Note that the DWT performs similar to the KLT, and sometimes the DWT performance is better, because the KLT is optimal under the assumption of Gaussian data.

2.3 DWT by the Lifting Scheme

The DWT has traditionally been developed by convolutions, and its implementation demands a large number of computational and storage resources.

A *new* approach has been proposed by Swelden [97] for wavelet transformation, which is called the *lifting scheme*. This mathematical formulation requires less computational and storage resources than the convolutional implementation.

Basically, the *lifting scheme* changes the convolution operations into matrix multiplications of the image with complementary filters, which allows for *in-place* computation, reducing the amount of both computation and storage resources [98]. At each step of the lifting method, the image is filtered by a set of complementary filters (i.e. , a low-pass filter and its complementary high-pass filter) that allows perfect reconstruction of the image. In the next step of lifting the low-pass filtered image is again filtered by set of complementary filters, and so on. Each step of the lifting scheme can be inverted, so that the image can be recovered perfectly from its wavelet coefficients. A rigorous mathematical explanation of the wavelet transform using the lifting scheme can be found in [97, 99, 100].

2.4 Implementing the DWT

The DWT can be implemented in different *flavors* by using different filter types, filter lengths, number of decomposition levels and dimensions. To establish how these *different flavors* affect the performance regarding the compression ratio, we implemented several 2D lifting-based algorithms to compress different seismic data sets. Results for different filter type, filter length and number of decomposition levels were established.

The type of filter

To classify the compression ratio according to the selected type of filter, we used three different types and varied their lengths, and their number of vanishing moments. The types of filters used were: Biorthogonals (Bior), Cohen-Daubechies-Feauveau (CDF) and Daubechies (Db). The experiments were performed in the following order:

1. 2D lifting-wavelet decomposition (1-level) using different filters.
2. Uniform quantization from 6 to 14 quantization bits.
3. Huffman entropy coding.

Figures 2.6 to 2.8 show the performance of the algorithms based on Biorthogonals (Bior), Cohen-Daubechies-Feauveau (CDF) and Daubechies (Db). Each mark in theses figures corresponds to a number of quantization bits from 6 to 14. For simplicity we only show three filters for each dataset (low, medium and high performance). However, we observed that for each type of filter there is a range of vanishing moments that achieves a better performance.

The best performance was obtained by using: Daubechies filters with a number of vanishing moments either 4 or 5, Biorthogonal filters with a number of vanishing moments from 5 to 8, and CDF filters with a number of vanishing moments from 2 to 4. In all other cases, a lower CR was achieved. These number of vanishing moments were required for both the decomposition filter and the reconstruction filter. In all cases, when either fewer or more of these vanishing moments were used, a lower CR was achieved.

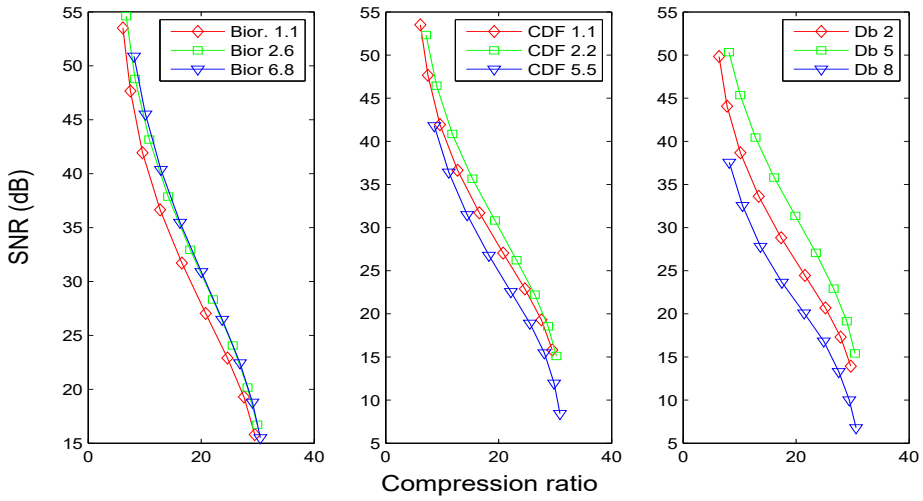


Figure 2.6: CR vs. SNR for the Dataset 1.

Figure 2.9 compares the best results among the three types of filters. Note that there are no significant differences between them.

The number of the decomposition levels

The discrete wavelet transform (DWT) can be applied in different levels of decomposition [101]. We compress the seismic data using different levels of the wavelet decompositions to determine the influence of the decomposition levels in the improvement of the compression ratio. The experiment was performed in the following order:

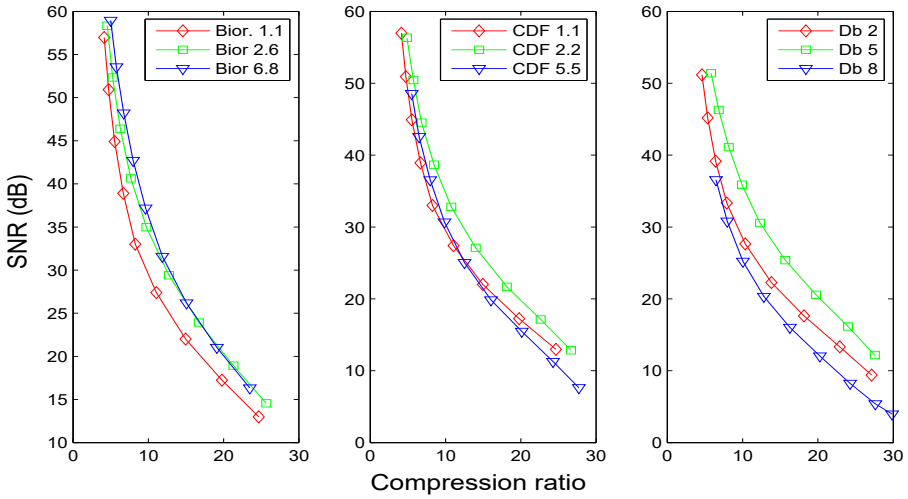


Figure 2.7: CR vs. SNR for the Dataset 2

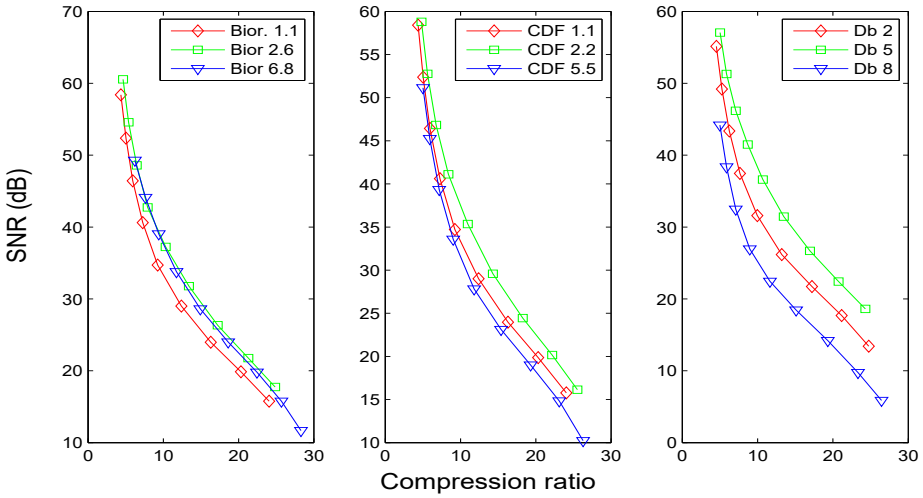


Figure 2.8: CR vs. SNR for the Dataset 3

1. 2D lifting-wavelet decomposition, using different levels of decomposition (From 1 to 7)

2. Uniform quantization using 12 quantization bits.

3. Huffman entropy coding.

Figure 2.10 shows the compression ratio vs the number of levels of decomposition. Note that

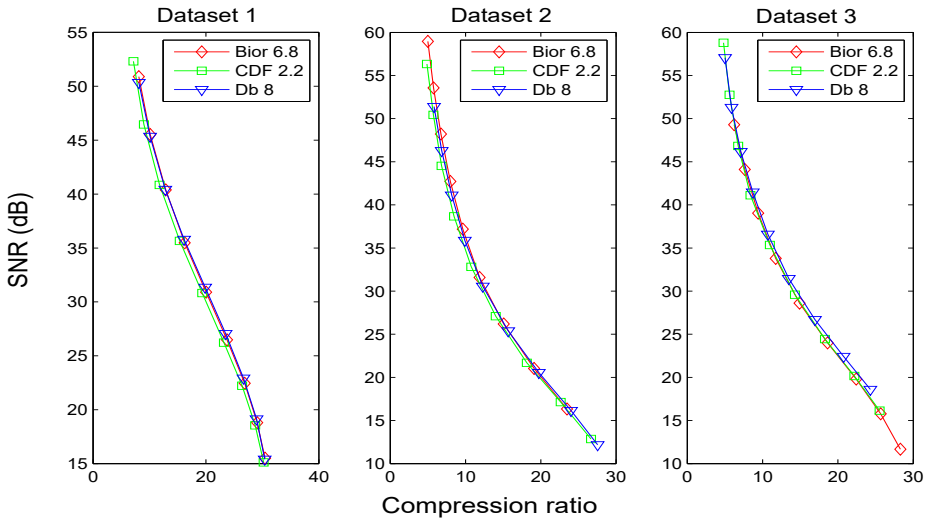


Figure 2.9: CR vs. SNR for all datasets. Best results for each type of filter at each dataset.

as the number of decomposition levels increases, the CR increases. This is true for all datasets, until a particular number of decomposition levels is reached. After that level of decomposition, the compression ratio holds almost constant. However, it is important to remark that as the number of decomposition levels increases, the SNR is reduced due to the loss of quality in the quantization process (Section 1.1). Therefore, it is necessary to choose the best compression ratio above 40 dB (in terms of SNR) among the different levels of decomposition. Table 2.1 summarizes our best results for three of our dataset.

Table 2.1: Better results above of 40 dB

Dataset	Level	SNR	CR
Data 1	3	42.45	13.56
Data 2	2	41.55	10.23
Data 3	2	42.08	13.92

2.5 Discussion

The KLT is considered the *optimal* transformation stage, in the context of compression data. However, this transformation is dependent on the statistics of the sequence, i.e. when the statistics change so also the KLT, which makes this transformation impractical because of its high computational cost.

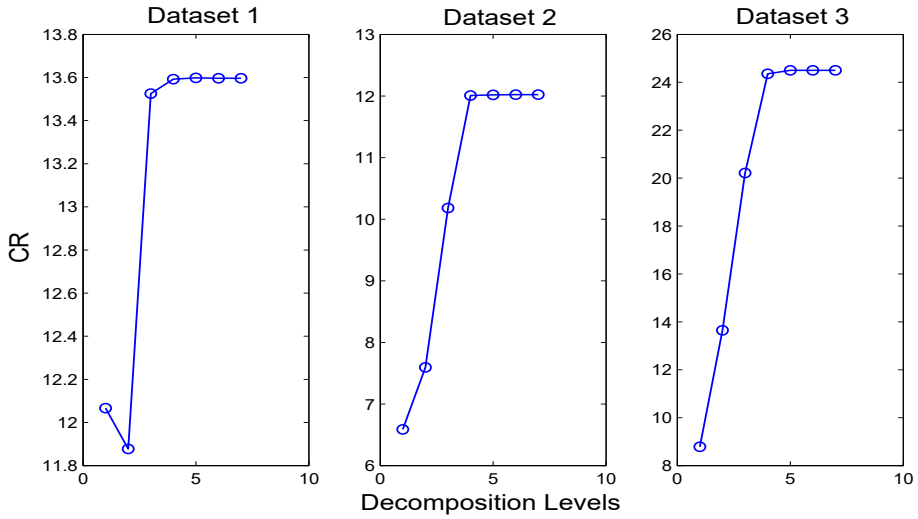


Figure 2.10: CR vs number of decomposition levels for three datasets.

We compared the DWT with the KLT and no significant difference was observed, which makes this transformation an option in our strategy.

The implementation of the DWT by using the lifting scheme reduces both the memory and computational resources. On the one hand, the lifting-based approach improves the computational performance by a factor of two [102, 97]. On the other hand, this scheme allows a fully in place calculation of the wavelet transform, i.e. no auxiliary memory is needed, and the data (e.g. the seismic shot) can be replaced with its wavelet transform [97].

Regarding the wavelet filter used, it seems that the performance –regarding the SNR obtained for a given compression ratio– is not strongly correlated with the type of filter [91]. It is possible to achieve the best performance using different types of wavelet filters (*e.g.*, Bior, CDF or Db) [91]. On the other hand, our results suggest that this performance is related to the number of vanishing moments of the wavelet filter.

The *most appropriate* number of levels of decomposition can not be established, since the compression ratio did not improve for a particular number of decomposition levels. Our results showed that a *moderate* number of decomposition levels (from 2 to 4) achieves the best compression ratio at the same level of SNR.

Chapter 3

GPU Implementations

As mentioned, the trade-off between the compression ratio and the processing time has not allowed the use of a compression strategy to speed up the transfer process between the main memory and the device memory.

In this chapter, we describe our efforts in optimizing the implementation of the decoding process into the GPU architecture. The chapter shows the throughput results by using different strategies of implementation. We also compare our throughput results with similar research works.

The chapter shows that despite our efforts to speed up the decoding process, the throughput obtained was no enough to improve the I/O transfer operations. Our tests showed that the time spent in the decoding process was longer than the time to transfer the data without compression.

3.1 Selecting the coding Algorithm

Huffman and Arithmetic coding schemes offer a high performance in terms of compression ratio (Section 1.1) and both have been widely used in seismic data compression (Section 1.3). However, as mentioned, the variable length of its code-words makes difficult to implement efficiently these coding schemes in parallel architectures, such that the decoding process becomes the main bottleneck in the decompression process.

On the other hand, the use of fixed-size codes coding schemes (e.g. Tunsdall Coding [103]) could be a better option for a parallel architecture. However, the performance of these fixed-size coding schemes, in terms of compression ratio, does not allows a notable reduction the *compressed data transfer time* (t_1 in Equation 1) to improve the transfer speed.

In Section 1.1, it was shown that both Huffman and Arithmetic coding schemes exhibit similar performance at values of SNR around 40 dB. Nevertheless, Huffman coding has a lower computational cost than the arithmetic coding.

For the reasons exposed above, we consider that the Huffman algorithm is the most appropriate coding scheme for achieving both a high compression ratio and the required throughput in the compression-decompression strategy.

3.2 Huffman Coding Implementation

We describe the CPU implementation of the coding process and the GPU implementation of the decoding process by an example. Let

$$W = [ACEBBCEBEEEBEABDACDBEDAECBCEBABEECECBDBEA] \tag{3.1}$$

be a string from an alphabet $\Sigma = \{A, B, C, D, E\}$ of cardinality 5. The frequency of occurrences of each symbol of Σ in W is shown in following table

Table 3.1: Frequency of each symbol in W

Symbol	Frequency
A	6
B	11
C	7
D	4
E	12

The Huffman encoding algorithm develops a binary tree (in order to create a dictionary) as follows:

1. The symbols are sorted in ascending order, according to their frequency (or probability) of occurrence. These symbols are written orderly in the bottom nodes, i.e. the tree leaves, as shown in Figure 3.1.
2. The nodes corresponding to the two less frequent symbols are grouped in order to create a new node. The frequency of the new node is the sum of the frequencies of the grouped nodes.

3. The previous step is repeated for all nodes until remains only one node, i.e. the tree root. At this node, the frequency of occurrence must be the number of data.

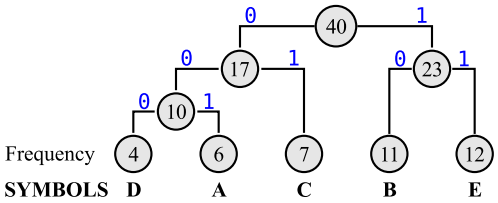


Figure 3.1: Huffman binary tree for the example data.

Once the binary tree is developed, the bits '1' and '0' are assigned to each branch (Figure 3.1). The code-words for each symbol are obtained grouping these bits from the root to each leaf. The dictionary consists of the symbols in the data and their respective code-word. Due to the variable length of the code-words, it is required to save these lengths into the dictionary to decode the information correctly. Table 3.2 shows the dictionary for the binary tree in Figure 3.1.

Table 3.2: Huffman dictionary for the example data.

Symbol	Code-word	Code-word length
E	11	2
B	10	2
C	01	2
A	001	3
D	000	3

The compression is achieved by replacing each symbol in the data with the corresponding Huffman code-word. As a result, it is obtained a bitstream that has the encoded data. On the other hand, the decoding is done by comparing bit to bit such bitstream with the code-words in the dictionary.

We propose a strategy that uses packets with headers, which seeks to force the alignment of code words at packet boundaries, allowing us to parallelize the decoding process.

We implemented a procedure to reduce the bits required to save the dictionary, taking into account the length of the code-words for our seismic data-sets. We saved the dictionary into two 32-bit vectors. The vector **s** contains the symbols and the vector **c** contains the code-words (using

the first 27 bits) and its length (using the last 5 bits). Table 3.3 shows the saved dictionary for our example using this procedure.

Table 3.3: Dictionary saved using two 32-bit vectors.

Vector **s**	Vector **c**	
Symbols	Code-words	& lengths
E	11xx ...x	00010
B	10xx ...x	00010
C	01xx ...x	00010
A	001x ...x	00011
D	000x ...x	00011

On the other hand, the encoded data must be saved into another vector whose elements are 32-bit or 64-bit variables. In a traditional coding, the encoded data is saved into the vector as a bitstream, no matter whether a code-word is saved into two different elements (highlighted code-words in Figure 3.2) or not.

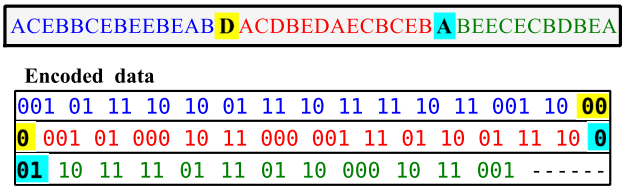


Figure 3.2: Encoded data using traditional coding for 32-bit variables.

Coding using 32-bit and 64-bit packets

As the GPUs are optimized to process the data using both 32-bit and 64-bit variables, we use 32-bit and 64-bit packets. Both strategies make use of the same dictionary.

In each 32-bit packet, 27 bits are used to save the encoded data, while 5 bits are used to save how many symbols are in the packet (Figure 3.3). This arrangement takes into account that it is possible to save up to 27 symbols encoded with 1-bit code-word. So, 5 bits are required to indicate that there are up to 27 symbols in the packet.

The proposed strategy compels that no code-word will be divided into two different packets, therefore, some packets will have *unused bits* (indicated as '–' in Figure 3.3). This way, each packet can be treated as a small data-set that could be decoded individually. However, it is required to create an additional vector (called *index*) to know the position of the encoded symbols

ACEBBCEBEEBE ABDACDBEDA ECBCEBABEECEC BDBEA

Encoded data	symbols per packet
001 01 11 10 10 01 11 10 11 11 10 11 --	01100
001 10 000 001 01 000 10 11 000 001 -	01010
11 01 10 01 11 10 001 10 11 11 01 11 01	01101
10 000 10 11 001 ----- ----- -----	00101

Figure 3.3: Encoded data using packets of 32 bits.

in the original data-set. Each element of *index* has the sum of symbols encoded in the previous packets. Table 3.4 shows the *index* vector for the encoded data in Figure 3.3.

Table 3.4: Index vector for our example.

Packet	Symbols per packet	*Index*
1	12	0
2	10	12
3	13	22
4	5	35

In the 64-bit packets, 58 bits of each packet are used to save the encoded data while 6 bits are used to save how many symbols are in the packets (Figure 3.4).

ACEBBCEBEEBEABDACDBEDAECB CEBABEECECBDBEA

Encoded data	symbols per packet
001 01 11 10 10 01 11 10 11 11 10 11 001 10...01 10-	011001
01 11 10 001 10 11 11 01 11 01 10 000 10 11 001-...-	001111

Figure 3.4: Encoded data using packets of 64 bits.

The packets allowed us to parallelize the decoding process, but they have a drawback. Due to the *unused bits* and the additional *index* vector, the compression ratio is lower compared to a traditional coding strategy (As shown in Section 3.4).

Figure 3.5 shows the compression ratio using these coding strategies. Prior to encode the seismic data, a uniform quantization was applied to each dataset using 12 bits of quantization to ensure an SNR above of 40 dB [91].

Note that the traditional coding provides better compression ratio, because all bits are used to encode the data and no additional files are needed for the decoding process. Likewise, the

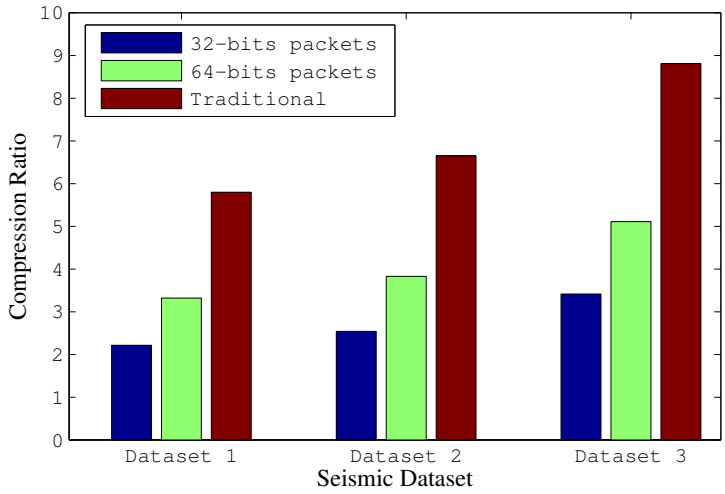


Figure 3.5: Compression ratio

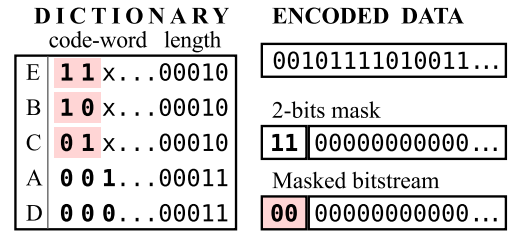
64-bit packets has better compression ratio than the 32-bit packets, since there are less *unused bits*, on average, with the 64-bit packets.

3.3 Decoding implementation

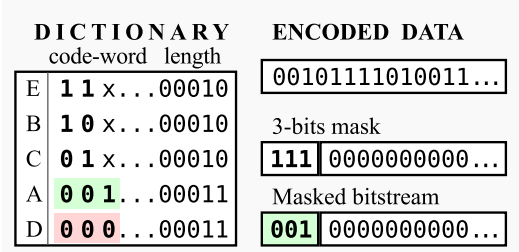
The decoding process is developed by comparing the encoded data with the Huffman code-words saved in the dictionary. The comparison is made taking into account the length of the code-words. The decoding algorithm is as follows (Figure 3.6):

1. Read the first row in the *dictionary* in order to know the length of the first code-word.
2. Mask the *encoded data* using the logical `AND` operation. The *mask* has as many `1`'s as bits has the code-word. The other bits are `0`.
3. Compare the *masked bitstream* with the code-words in the *dictionary* that have the same length. If there is a match, save the decoded symbol and go to step 5. Otherwise, continue with step 4.
4. Look for the next code-word length and return to step 2.
5. If all the symbols in the *encoded data* have been decoded, go to step 7. Otherwise, continue with step 6.
6. Shift the *encoded data* to the left as many bits have the previous code-word decoded. Then, go back to step 1 in order to decode the next symbol.

7. The decoding process is over.



(a) Failed matching.



(b) Successful matching.

Figure 3.6: Decoding process by masking the encoded data.

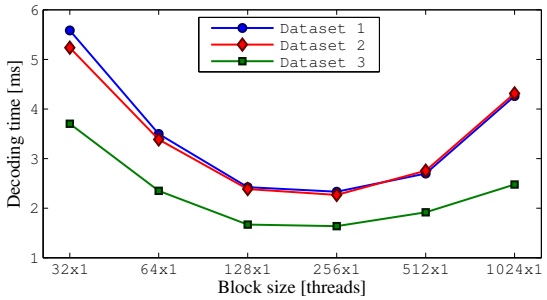
Figure 3.6 shows how different masks are used to decode our encoded data. In Figure 3.6a, no symbol is decoded, because the *masked bitstream* do not matches with any 2-bit code-word in the dictionary. Figure 3.6b shows how the symbol A can be decoded, because its code-word matches with the *masked bitstream*.

The decoding process can be parallelized assuming that one packet corresponds to an individual dataset. This way, each packet can be executed by one GPU thread, taking into account that symbols decoded by each thread must be organized according to the *index* vector.

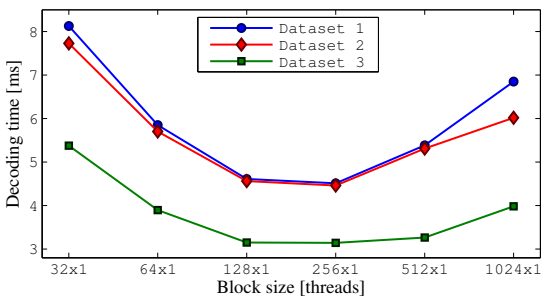
3.4 Throughput results

The creation of the Huffman dictionary and the coding process itself, was implemented on a CPU. The decoding algorithm was developed in CUDA 6.5 and implemented on a GeForce GTX660 GPU with 3.0 CUDA capability. The algorithms were tested with three of our seismic datasets.

The GPU implementation was done by using different threads-per-block distributions, i.e. different block sizes. Figure 3.7 shows the required time to decode the datasets encoded with

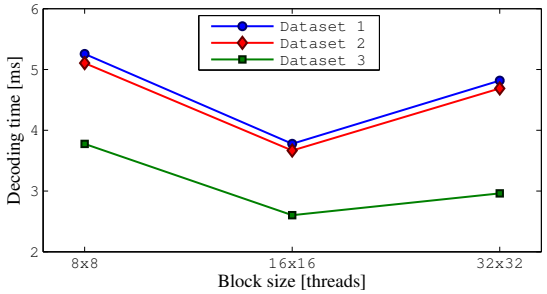


(a) Datasets encoded with 32-bit packets.

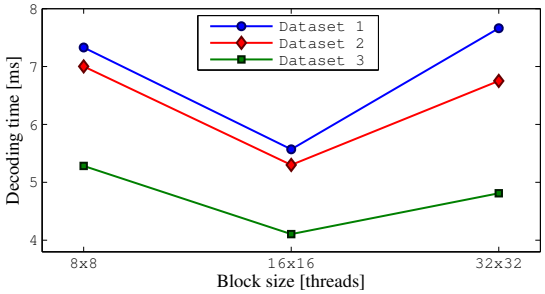


(b) Datasets encoded with 64-bit packets.

Figure 3.7: Decoding time using unidimensional blocks.



(a) Datasets encoded with 32-bit packets.



(b) Datasets encoded with 64-bit packets.

Figure 3.8: Decoding time using bidimensional blocks.

32-bit and 64-bit packets using unidimensional blocks, while Figure 3.8 shows the decoding time using bidimensional blocks. Note that, for 1D block sizes, the 256×1 distribution exhibits better performance, while for 2D block sizes, the best performance is obtained with 16×16 distribution.

In all cases, better performances were obtained when datasets were encoded using the 32-bit packets. So, the best performance was obtained using a block size of 256×1 threads to decode datasets encoded with 32-bit packets.

3.5 Discussion

The results show that packets that have less encoded symbols, i.e. such that have larger code-words, require more time to be decoded. This is because these code-words are at the end of the dictionary, and they demand more searching time. In this sense, the throughput improves as the compression ratio increases, because the average length of the code-words decreases.

On the other hand, the GPU executes the algorithm in groups of 32 threads running in parallel, which are called *warps*. As each packet is decoded by one thread, 32 packets are decoded simultaneously in the *warps*. For packets that have a different number of encoded symbols, the decoding time for each packet can be different into the *warp*. This difference in the decoding time generates *divergence* [104], which affects the GPU performance. The results show that the 32-bit packets strategy reduces the *divergence* compared with the 64-bit strategy.

It is difficult to compare our results with other Huffman decoders implemented on GPU because the nature of each dataset affects how it is encoded. Besides, the GPU technology is not necessarily the same from one work to another. However, some comparisons can be made if we take into account the throughput.

Table 3.5 compares our decoder with a predecessor work [24] that reports the decoding throughput. The results suggest that the performance of our decoder could be superior to the mentioned previous work.

Table 3.5: Predecessors Results

Author	GPU device	Compression Ratio	File Size $[MB]$	Throughput $[MB/s]$	Throughput $[Symbol * 10^6/s]$
Our decoder [86]	**GeForce GTX660 at 980 MHz**	**1.6**	**275.3**	**260.71**	**65.18**
Aqrawi 2010 [24]	Tesla C1060 at 1.3 GHz	1.4	244.14	116.26	29.07

The GPUs are successfully used with algorithms that exhibit a parallel behavior and are designed to work in float-point format. However, when the decoding process is implemented into the GPUs, the sequential stages quickly become the bottleneck in the overall process.

As mentioned, the throughput obtained for the decoding stage was no enough to speed up the transfer operations between the main memory and node memory. In this sense, the decompression process requires a custom architecture to overcome the sequential bottleneck that is generated by the decoding stage.

Chapter 4

FPGA Implementations

One of the main challenges in this dissertation is the implementation of the decoding process into a parallel architecture, because of its sequential nature. In this chapter, we describe the two computational architectures that were developed for the decoding process. We also present the transformation stage, explaining how it was developed, and giving the results regarding the processing time and the computational resources.

Finally, we summarize and discuss the throughput results obtained for both decoding and transformation stages.

4.1 Huffman decoder Version 1

The decoding process requires to be developed at bit level. The first approach seeks to develop a computational architecture that is optimized to perform the decoding process at bit level.

The general architecture of such decoder is shown in Figure 4.1. The decoder use a custom memory (*DATA MEMORY*) to store the string of compressed data. In this memory, each position has only one bit. The Huffman dictionary is stored in the *DICTIO MEMORY* as follows:

- Each memory position has 32 bits.
  - The 12 least significant bits are used to store the original representation of the symbols.
  - The 20 most significant bits correspond to the Huffman codewords.
  - The shortest codes are extended with the first bit of the codeword corresponding to the most frequent symbol.

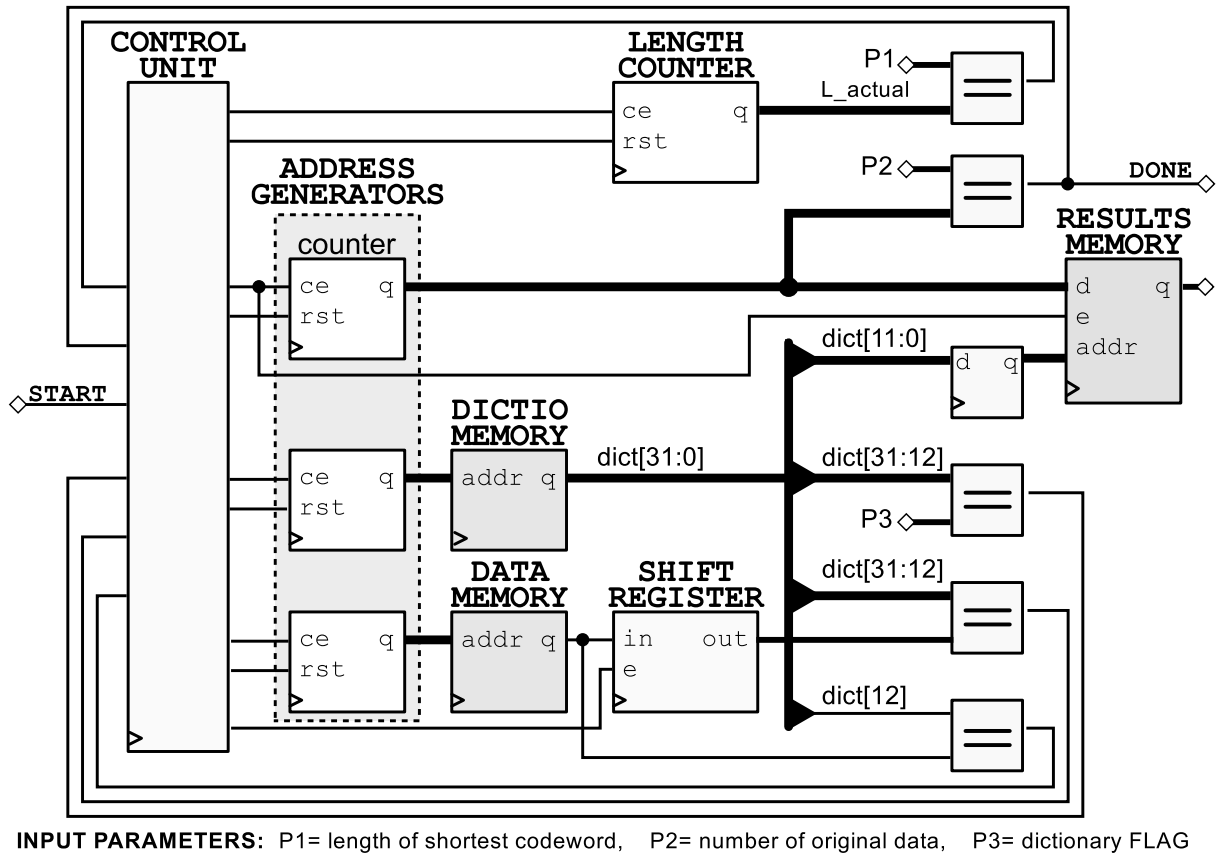


Figure 4.1: Block diagram of Huffman decoder version 1.

- The dictionary is sorted in descending order according to the frequency of the symbols.
- Each time the length of the code-word changes, many *FLAG*s are added to the dictionary as the number of increased bits in the length of the code-word. These *FLAG*s allow the decoder to know how many bits of the encoded data must be compared with the codewords.

The *CONTROL UNIT* is responsible for controlling the decoding process and the communication with the memories.

On the one hand, when the length of the shortest codeword, L_{SC}, is only 1 bit, the control unit disables the *shift_register*, so the decoder only takes one clock cycle to decode it. In this case, another clock cycle is needed to store the decoded symbol into *RESULTS MEMORY*. The other symbols (i.e. those encoded with more than one bit) are decoded and stored in at least 5 clock cycles.

On the other hand, when $L_{\mathrm{SC}} > 1$ bit, the control unit enables the *shift_register* to compare the

correct number of bits. In this case, the most frequent symbol is decoded in at least $L_{\text{SC}}+3$ clock cycles, while the other symbols need at least $L_{\text{SC}}+5$ clock cycles to be decoded.

This first version seeks to optimize the memory system and includes hardwired instructions at bit level to speedup the decoding process.

4.2 Huffman decoder Version 2

Once achieved an optimized architecture to perform the decoding process at bit level, our second version aims to improve the throughput by parallelizing the decoding process itself.

Several parallel Huffman decoders have been developed for applications where the dynamic range is known and bounded, such as image or video applications [105, 106, 107]. A bounded dynamic range helps to develop the computational architecture.

However, for seismic data, the dynamic range is wider and changes from one seismic dataset to other. Furthermore, the behavior of the code-word lengths can change significantly from one seismic dataset to the other.

We aim to develop a Huffman decoder that works for any of our seismic datasets, no matter the dictionary length and the behavior of the code-word lengths.

We determined the necessity of a preliminary study that allowed us to establish the behavior of the Huffman dictionaries generated for our seismic datasets. The study aimed to find all parameters required to design a computational architecture to develop a parallel decoder.

Table 4.1 shows the length of the Huffman dictionaries, which were obtained when we encoded the 12 seismic shot. Prior to encoding the seismic data, a uniform quantization was applied to each dataset, which uses a number of quantification-bits which guarantees a SNR around 40 dB. Table 4.1 also shows the result regarding compression ratio and SNR.

On the other hand, Table 4.2 shows the behavior of the code-words lengths. For example, for the seismic shot 1, there are three code-words of 3-bits, two code-words of 4-bits, and four code-words of 5-bits.

The preliminary study allowed us to establish that is hardly feasible to have full parallel Huffman decoder for seismic data, because of the amount of computational resources that a full parallel decoder would demand.

However, from Table 4.2 note that more than 50% of the data is encoded by using code-words with lengths between 1 and 5 bits. In some cases, up to 80% of the data is encoded by code-words

Table 4.1: Length of Huffman Dictionaries, CR and SNR

Dataset	Quantization Bits	Dictionary Length	Compression Ratio	SNR
Shot 1	12	1235	5.79	43.13
Shot 2	12	1260	5.70	45.40
Shot 3	12	1406	5.21	45.09
Shot 4	12	1238	5.16	43.83
Shot 5	12	1563	6.58	45.69
Shot 6	11	1617	7.72	40.10
Shot 7	12	1622	6.16	45.85
Shot 8	11	1629	7.87	40.18
Shot 9	12	1313	8.20	44.12
Shot 10	11	1578	8.26	40.19
Shot 11	12	1438	10.02	44.87
Shot 12	12	948	8.02	44.08

with these lengths.

Therefore, by decoding in parallel the five most frequent symbols, it is guaranteed that, in average, $67,51\%$ of the decoding process will be done in a parallel fashion.

Table 4.2: Code-word lengths

Seismic shot	1 bit	2 bits	3 bits	4 bits	5 bits	Representation percentage
Shot 1	0	0	3	2	4	58.26%
Shot 2	0	0	2	3	6	61.48%
Shot 3	0	0	0	5	6	53.16%
Shot 4	0	0	0	5	7	51.45%
Shot 5	0	1	2	2	2	65.09%
Shot 6	0	1	2	2	3	75.72%
Shot 7	0	1	2	2	2	61.51%
Shot 8	0	1	2	2	2	75.14%
Shot 9	1	0	2	0	2	73.87%
Shot 10	1	0	2	0	2	80.65%
Shot 11	0	1	2	2	2	76.12%
Shot 12	0	2	1	2	1	77.71%

Figure 4.2 shows the computational architecture of the Parallel Huffman decoder, which decodes in a parallel fashion the first five code-words. This architecture was implemented into an FPGA Virtex 5 XC5VFX70T.

The *PARALLEL COMPARATOR* is responsible for finding all code-words with lengths up to 5 bits in a parallel fashion. This module has both a register set and comparator set, which allowed us to decode all these code-words in just one clock cycle.

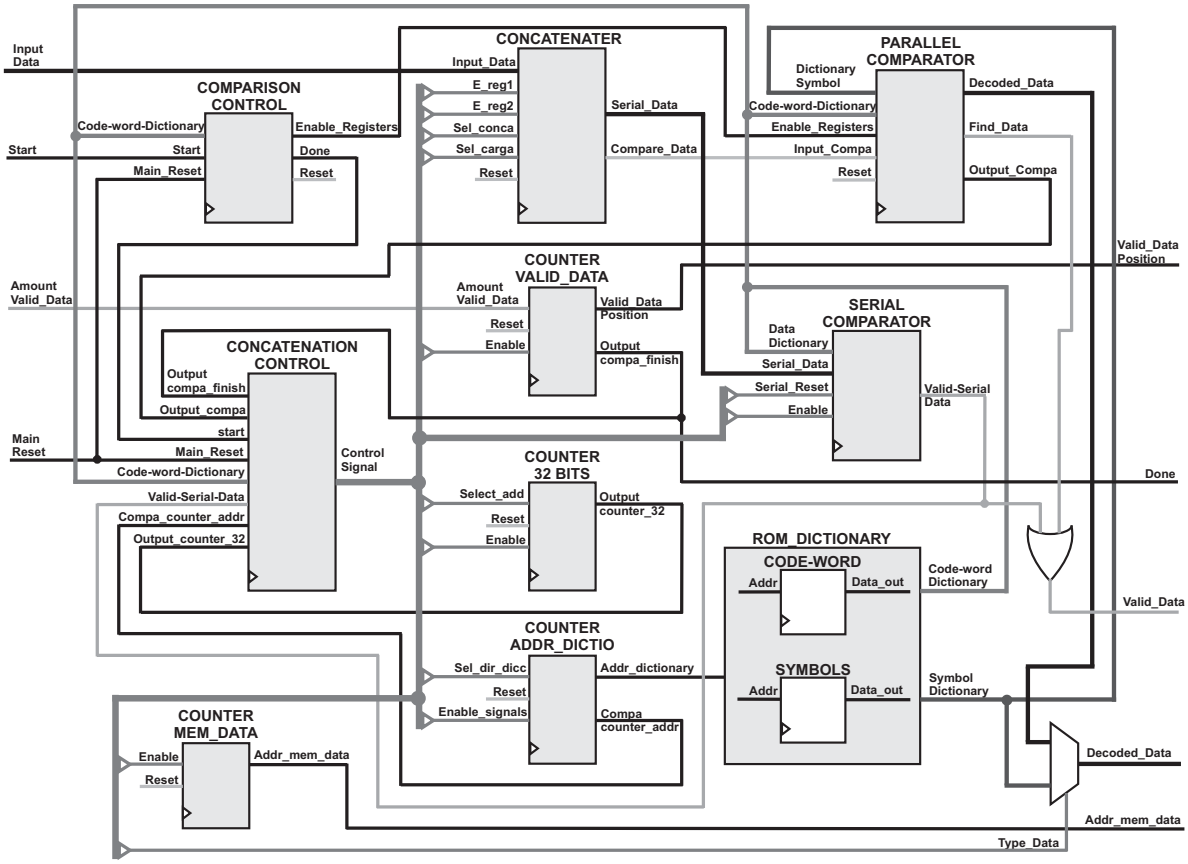


Figure 4.2: Computational Architecture for the Parallel Huffman Decoder

On the other hand, the *PARALLEL COMPARATOR* is disabled when the code-word has more than 5 bits of length. In this case, the comparison process is developed by the *SERIAL COMPARATOR* by searching directly on the *CODE-WORD* memory. In this case, at least three clock cycles are required to update the memory address and compare a new code word. An additional clock cycle is required when there is a change in the code word length.

The *CONCATENATER* is responsible for loading the encoded data string in the decoding process. This module loads a new 32-bit line when the decoding process has finished with a memory position. Additionally, the module concatenates the appropriate number of bits on the comparators.

The process is mainly controlled by two Finite State Machines (FSM). The first one (the *COMPARATOR CONTROL*) handles the loading process of the set register into the *PARALLEL*

COMPARATOR at the beginning of the process. The second FSM (the *CONCATENATER CONTROL*) handles both the parallel and serial comparison processes.

As mentioned, when the code-word length is greater than 5 bits, the second version requires at least four clock cycles to update the memory address and to compare a new code-word.

We found that these processes of update and compare are recurrent. In a new version (V2.1), we reduce from 4 to 2, the number of clock cycles required to update the memory address and to compare a new code-word. As shown in the next section, this reduction allowed us to improve the overall performance in the decoding process.

4.3 Throughput Results for Decoding Process

Figure 4.3 shows the Throughput (in MB/s) vs the number of quantization bits, for a single core. Note that, the version 2.1 offers a better performance, especially for high compression ratios.

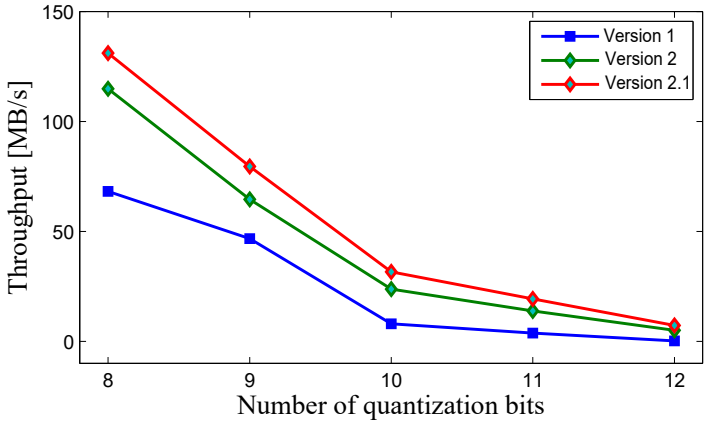


Figure 4.3: Throughput results for the decoding process (one core)

On the other hand, Figure 4.4 shows the average number of clock cycles for the three versions of the Huffman decoder. Note that, there is a significant difference between the versions regarding the average number of clock cycles.

4.4 Transformation stage

The DWT is traditionally developed by convolutions, which demand both a large number of mathematical operations and a large amount of storage. The mathematical formulation for

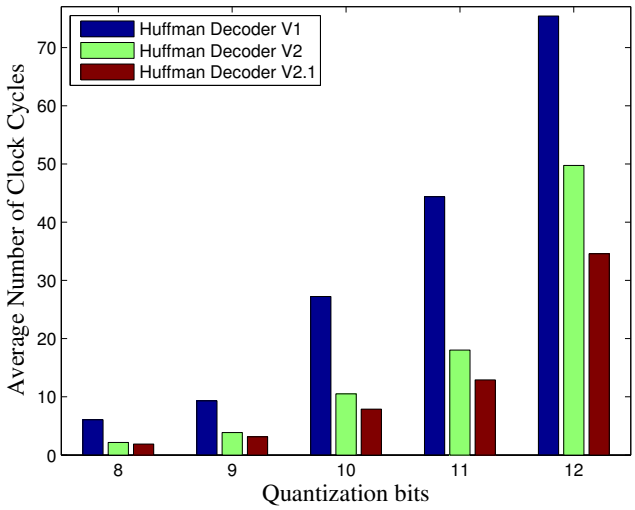
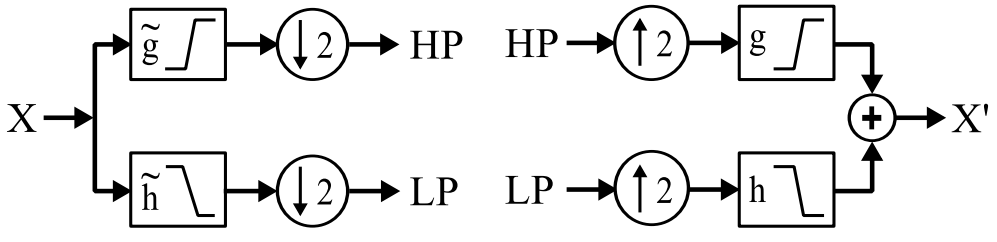


Figure 4.4: Average number of clock cycles for the Huffman Decoder versions

DWT proposed by Swelden [97] – called lifting scheme – requires less computation and storage resources than the traditional DWT. The lifting-based implementation seeks to improve the algorithm performance to calculate the wavelet transform by changing the convolution operations for multiplication operations of Laurent polymonials [108].

The single-level of the traditional DWT is shown in Figure 4.5. The forward DWT consists in applying the analysis filter pair $\tilde{h}$ – $\tilde{g}$ (low-pass and high-pass respectively), followed by a down-sampling operation (Figure 4.5a). As a result, the input signal is decomposed in approximation (LP) and detail (HP) coefficients, which correspond to the low and high-frequency components respectively. The inverse DWT (IDWT) consists in upsampling the LP and HP coefficients followed by applying the synthesis filters h and g (Figure 4.5b).



(a) Forward (b) Inverse

Figure 4.5: Discrete Wavelet Transform

The polyphase matrix representation of a wavelet filter bank is given by:

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} \tag{4.1}$$

$$P(z) = \begin{bmatrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{bmatrix},$$

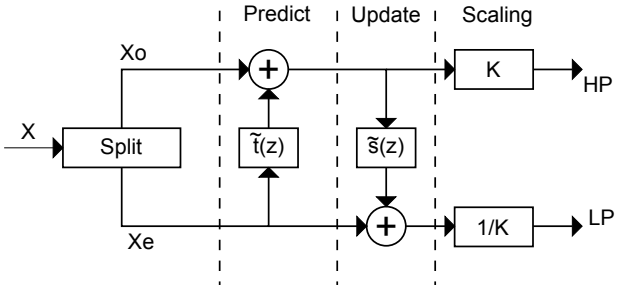
where $\tilde{h}_e(z)$, $\tilde{h}_o(z)$, $\tilde{g}_e(z)$ and $\tilde{g}_o(z)$ denote the even and odd components of the analysis filters, while $h_e(z)$, $h_o(z)$, $g_e(z)$ and $g_o(z)$ denote the even and odd components of the synthesis filters.

Since that any pair of complementary filters (h, g) can be factorized into a sequence of triangular matrices –upper and lower– and a diagonal matrix [108]. The matrix $\tilde{P}(z)$ can be factorized as:

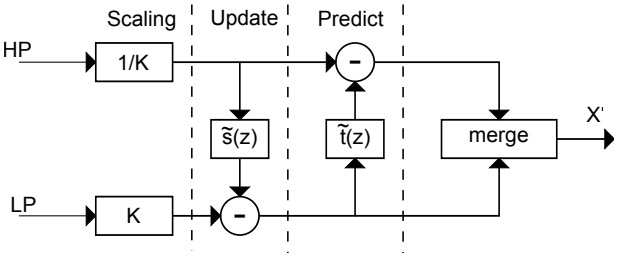
$$\tilde{P}(z) = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \prod_{i=1}^{m} \left(\begin{bmatrix} 1 & \tilde{s}_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{t}_i(z) & 1 \end{bmatrix} \right) \tag{4.2}$$

where $\tilde{s}_i(z)$ and $\tilde{t}_i(z)$ are Laurent polynomials, m is the number of lifting steps and K is a non-zero scaling factor.

The factorization of $\tilde{P}(z)$ allows a different way to decompose the signal –the lifting scheme– as shown in Figure 4.6.



(a) Forward



(b) Inverse

Figure 4.6: Lifting Scheme

The decomposition process using the lifting scheme can be summarized in the following four stages:

- *Split*: The signal X is split into odd component (X_o) and even component (X_e). This stage is also called *lazy wavelet*.
- *Predict*: The even component is multiplied by the *predict* operator $\tilde{t}(z)$ and added to the original odd component to obtain the new odd component. This stage is also known as *dual lifting*.
- *Update*: The new odd component is multiplied by the *update* operator $\tilde{s}(z)$ and added to the original even component to update the even component. This stage is also named *primal lifting*.
- *Scaling*: Both odd and even components are multiplied by $1/K$ and K respectively.

In this scheme, the reconstruction process is performed by running the forward scheme backwards as shown in Figure **??**. In the inverse scheme, the scaling factor is changed from K to $1/K$ and the signs of the coefficients in the Laurents polynomials $\tilde{s}_i(z)$ and $\tilde{t}_i(z)$ are flipped from positive to negative.

Selecting the Wavelet Filter

In chapter 1 was established that regarding the wavelet filter, the performance is not strongly correlated with the type of filter but the number of vanishing moments. In other words, it is possible to achieve the best performance with any type of filter (e.g. Cohen-Daubechies-Feauveau, Daubechies, etc.) by selecting the *appropriate* number vanishing moments for each type of filter.

In the lifting scheme, the filter CDF 2.2, which offers a high performance in terms of compression ratio vs SNR (see section 2.4), involves two multiplications, which can be implemented by using shift registers.

For the CDF 2.2 wavelet filter, the analysis low-pass filter has five coefficients while the analysis high-pass filter has three coefficients as follows:

$$\begin{aligned} \tilde{h}(z) &= -\tfrac{1}{8}z^{-2} + \tfrac{1}{4}z^{-1} + \tfrac{3}{4}z^{0} + \tfrac{1}{4}z^{1} - \tfrac{1}{8}z^{2} \\ \tilde{g}(z) &= -\tfrac{1}{2}z^{-2} + z^{-1} - \tfrac{1}{2}z^{0} \end{aligned} \tag{4.3}$$

The corresponding polyphase matrix is:

$$\begin{aligned} \tilde{P}(z) &= \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z & \frac{1}{4} + \frac{1}{4}z \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix}. \end{aligned} \tag{4.4}$$

This matrix can be factorized into two elementary matrices, as follows:

$$\tilde{P}(z) = \begin{bmatrix} 1 & \frac{1}{4}(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}(1+z^{-1}) & 1 \end{bmatrix}, \tag{4.5}$$

Note that the lifting scheme will only include one predict and update step (i.e. $m = 1$ in Equation 4.2).

This leads to the predict and update stages (in the time domain) being represented in the following equations:

$$\begin{aligned} HP &= x_{2i+1} - \tfrac{1}{2}(x_{2i} + x_{2i+2}) \\ LP &= x_{2i} + \tfrac{1}{4}(y_{2i+1} + y_{2i+3}) \end{aligned} \tag{4.6}$$

As mentioned, these multiplications can be implemented by using shift registers.

CDF 2.2 Lifting Wavelet architecture

We propose a computational architecture to calculate the inverse 1-D discrete wavelet transform using a lifting-based scheme for the CDF 2.2 wavelet filter. As shown in Figure 4.7, the architecture is divided into four main blocks: UPDATE, PREDICT, CONTROL and MEMORY.

The UPDATE and the PREDICT stages allow the reconstruction of both the odd and the even samples respectively. The MERGE stage was carried out by controlling the addresses of the dual-port memory while the results are being stored.

The control stage generates the addresses for the memory that contains the details and the approximations which are processed by the update and predict stages. This stage also deals with the control signals that allow to write and read the memory and the registers.

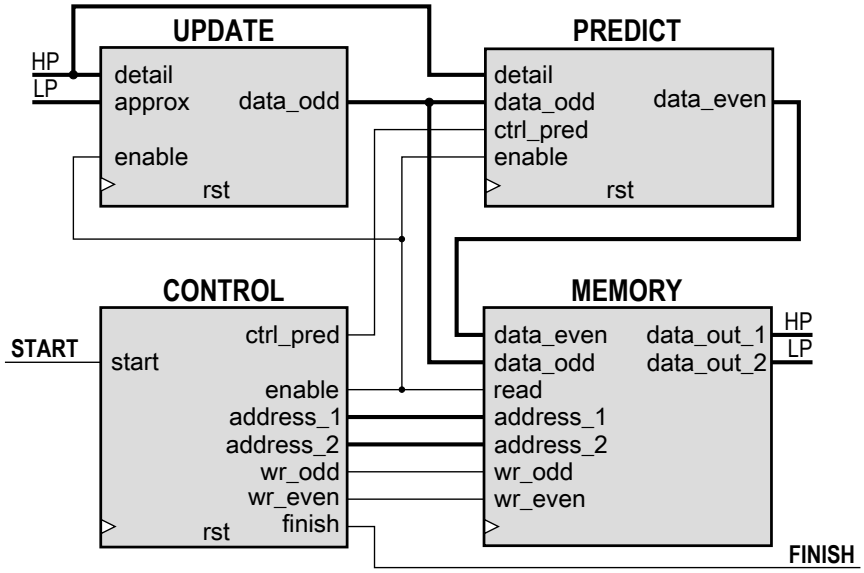


Figure 4.7: Implementation Block Diagram

Throughput results

The design was implemented into a Virtex $5-XC5VFX70T$ FPGA. In order to verify the correct operation of the architecture, both the detail and approximation coefficients were first loaded into the memory using Memory Initialization Files (.MIF).

The reconstruction of both odd and even samples requires one clock cycle. Therefore, one level of lifting transform, for an input data of size N, is computed in $\lceil N/2 \rceil$ clock cycles, two levels require $\lceil N/2 \rceil + \lceil N/4 \rceil$ cycles and so on. Thus, the number of clock cycles required to perform L levels amounts to:

$$\left\lceil \frac{x}{2} \right\rceil + \left\lceil \frac{x}{4} \right\rceil + \ldots + \left\lceil \frac{x}{2^L} \right\rceil \approx \sum_{i=1}^{L} \frac{N}{2^i} = N(1 - 0.5^L) \tag{4.7}$$

In addition, there are two clock cycles of latency to obtain the first odd sample and three clock cycles to obtain the first even sample. Thus, our design requires $3L + N(1 - 0.5^L)$ clock cycles to compute L levels of 1D lifting scheme for data of size N.

4.5 Discussion

As far as we know, our Huffman decoder (Version 1) is the first computational architecture published in the area of seismic data compression [22]. Other ones have been developed for

Table 4.3: Percentage of representation for different number of quantization bits

Seismic Shot	Quantization bits	CR	SNR	Percentaje of Representation
Shot 1	9	16.32	24.28	92.33 %
	10	12.73	31.31	88.09%
	11	9.68	37.11	80.37%
	12	7.50	42.78	74.73%
Shot 2	9	15.86	26.05	91.61%
	10	12.21	32.04	87.02%
	11	9.67	37.71	78.75%
	12	7.28	43.42	72.74%
Shot 3	9	12.38	25.99	87.27%
	10	9.55	32.71	81.52%
	11	7.64	39.11	75.70%
	12	6.23	45.10	65.06%

images and video applications [105, 109]. Therefore, it is difficult to compare our results against other Huffman decoder implemented on FPGA, since the nature of each dataset affects how it is encoded. Besides, the FPGA technology is not necessarily the same from one work to another.

However, some estimations can be made if we take into account: the throughput, and the logical resources used. A comparison of our Huffman decoder (Version 1) with some predecessors works suggests that ours overcomes in terms of throughput, and the logical resources used [22].

The performance of the Huffman decoder depends on the compression ratio. Table 4.3 shows the percentage of representation for a different number of quantization bits. Before encoding the seismic data, a CFD 2.2 wavelet transform and a uniform quantization were applied to each dataset. Note that as the number of quantization bits is increased, the percentage of representation become smaller. For this reason, the decoding time is strongly related to the compression ratio, because of the percentage of code-words that are decoding in parallel depends on the compression ratio.

In regard to the transformation stage, we compare our design with other similar implementations. The comparisons take into account the number of clock cycles required to carry out L levels of 1D lifting scheme. We also compared the resources used in the datapath, in terms of registers, multipliers, and adders. Table 4.4 summarizes both hardware and clock cycles comparisons.

On the other hand, the software ISE reports that the maximum operating frequency for the proposed architecture is $170MHz$, so if we compare this operating frequency with a similar FPGA

Table 4.4: Hardware and clock cycles comparisons

			Resources used		
Author	Filter	clock cycles	Reg.	Mult.	Adders
Our design [102]	5/3	$3L + N(1 - 0.5^L)$	3	2	4
Lian [110]	5/3	$4 + 2N(1 - 0.5^L)$	14	4	8
Huang [111]	9/7	$5 + 2N(1 - 0.5^L)$	16	4	8
Liao [112]	9/7	$N + L$	10	8	8

implementation [113], our implementation is around 1.75 times faster.

Chapter 5

Results

We tested the proposed strategy by implementing several decompression cores working in parallel. The parallel implementation was carried out using two different decompression cores. The first one consists of a uniform quantization and a Huffman coding. The second one is a transform-based algorithm, which consists of the Lifting-Wavelet Transform, a uniform quantization and a Huffman coding.

This chapter presents the speedup results in the transfer process between a CPU and an FPGA, through the PCI Express bus. Finally, a summary and a discussion about these speedup results is presented.

5.1 Parallel implementation

The platform used to test our strategy is shown in Figure 5.1. The system consists of a CPU and the *LXT ML507 Development Board*, which includes a *Virtex 5 XC5VFX70T FPGA*. The communication between CPU and FPGA was accomplished through PCIe bus.

Figure 5.2 shows the parallel architecture implemented, which has N decompression cores working in parallel.

This architecture has an optimized memory system consisting of three memory banks. The first bank is used to store the Huffman dictionary (*MEMORY BANK 0*). The second one is used to store the compressed data (*MEMORY BANK 1*), and the third bank is used to store the results of the decompression process (*MEMORY BANK 2*). Each bank has N dual port memories with 4096 memory positions each.

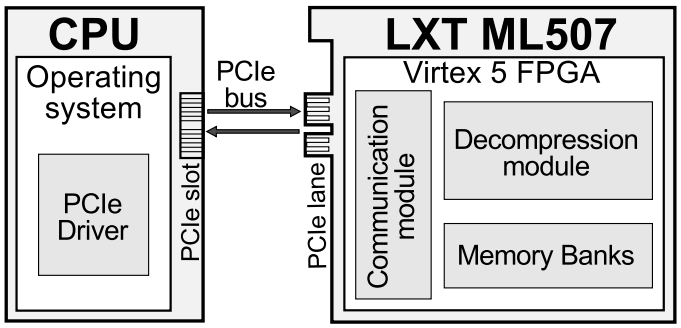


Figure 5.1: Co-processing platform

The ports of the memories are multiplexed and controlled by an *access logic*, which is responsible for controlling read and write operations over the memories.

Connecting the CPU with the FPGA

The first step to achieve the CPU-FPGA communication is the implementation of a driver on the CPU operating system, which is provided by Xilinx [114]. This driver must be modified because the original version only allows to write the whole memory, but our strategy requires to write in a particular memory position. To this end, an offset mechanism –called *llseek* [115]– was implemented.

On the other hand, in the FPGA, we used the *Endpoint Block* for PCI Express provided by Xilinx [116] as starting point. This module consists of a *PCIe core* and a *storage system*. The *PCIe core* offers the hardware description required by the PCIe communication protocol to transfer data via the PCIe bus. The *storage system* is used to store the data transferred.

In order to have direct access to the PCIe signals and increase the memory capacity, some modifications on the original *storage system* were required. These modifications include a register bank and connections to external memories (Figure 5.3). The register bank has 32 registers, which are used to store the compression parameters needed during the decompression process. Such parameters are sent by the CPU through the PCI bus.

5.2 Time schedule

Figure 5.4 shows the time schedule carried out to test the strategy. First, the Huffman dictionary is sent and stored in all N the memories of *MEMORY BANK 0*. Then, each section of the

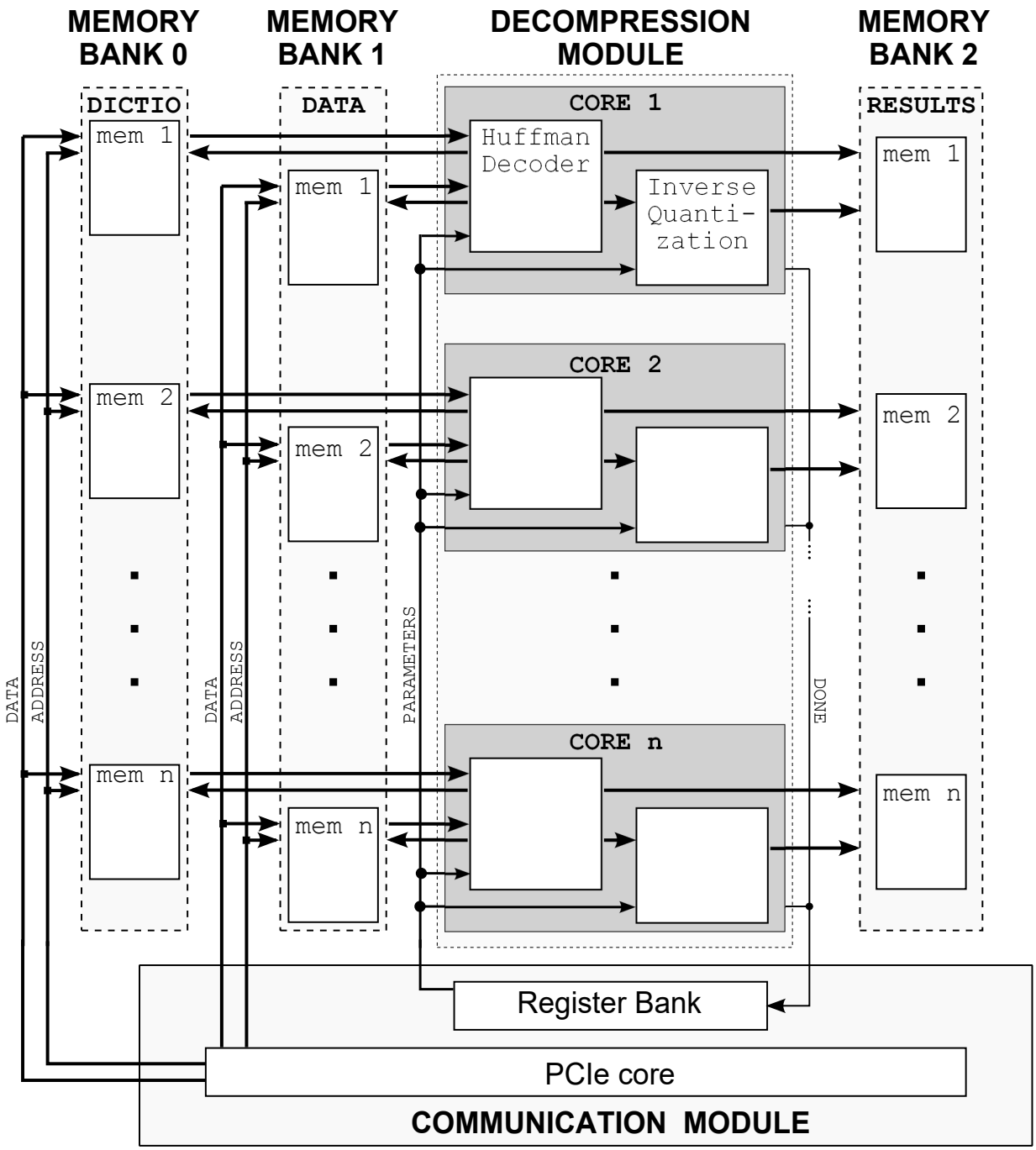


Figure 5.2: Parallel architecture for the decompression process

compressed data is sent sequentially to each memory of *MEMORY BANK 1*. Note that once a data section has been sent, the corresponding decompression core starts to operate.

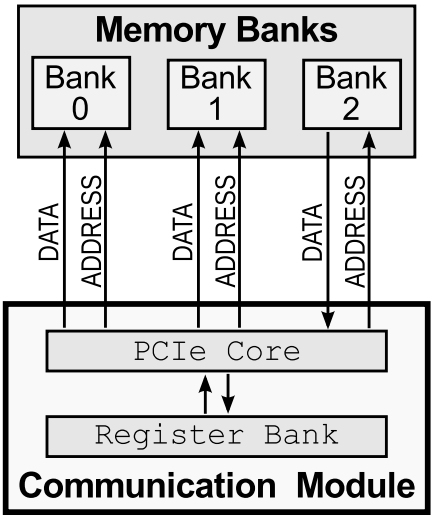


Figure 5.3: Communication module

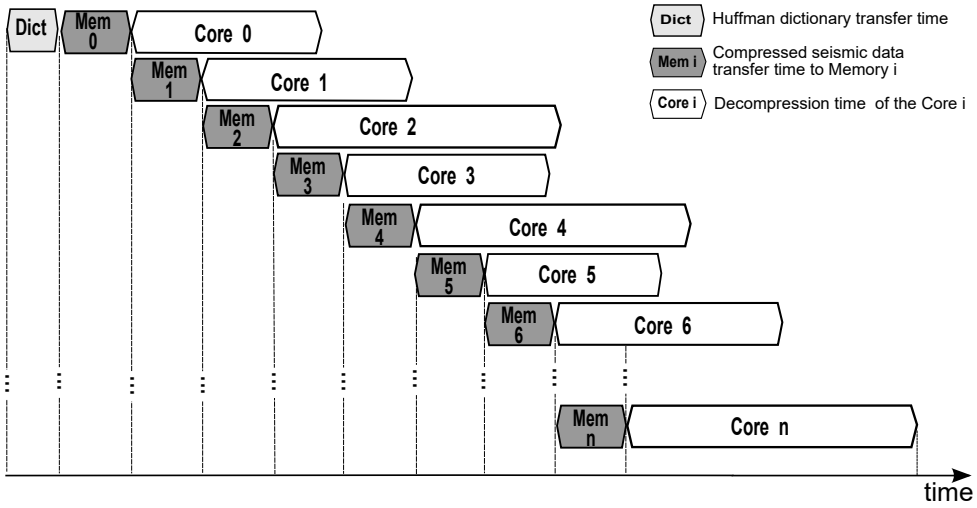
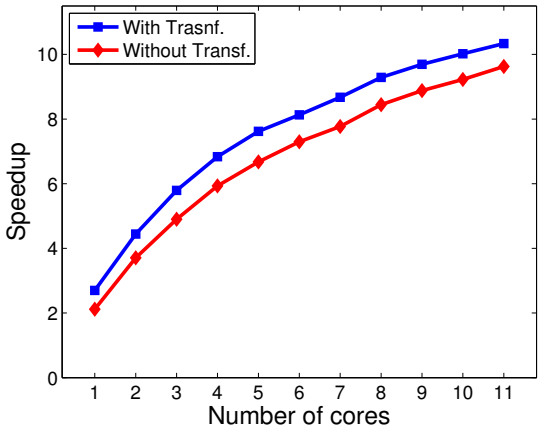


Figure 5.4: Time schedule carried out in the parallel architecture

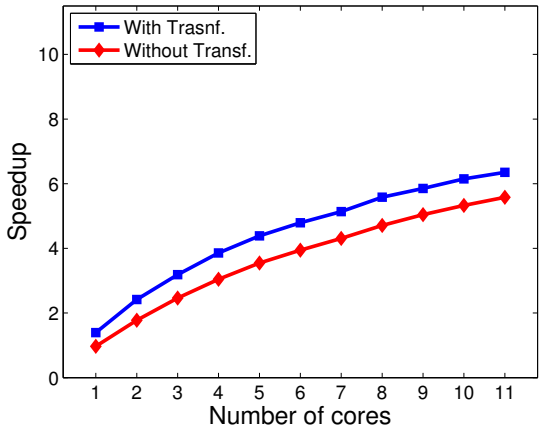
5.3 Speedup Results

We compared the speedup results between two different implementations. The first one is a compression-decompression algorithm without transformation stage (i.e. quantization and coding stages), which aims to reduce the decompression time by removing the transformation stage. The second one is a transform-based algorithm, which aim to improve the compression ratio by using the transformation stage.

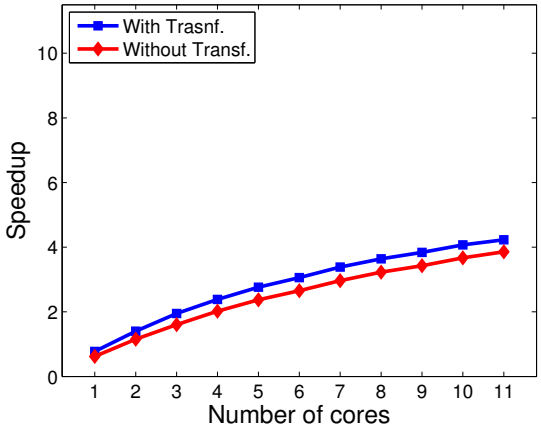
Both implementations use our Huffman decoder version 2.1 to speedup the decoding process



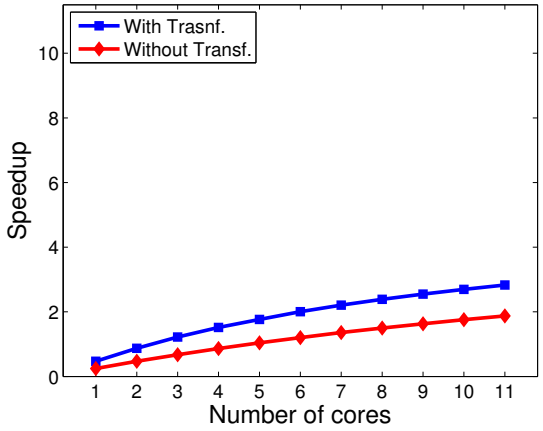
(a) compression ratio: 16.32 with transformation and 12.17 without transformation (9 bits of quantification).



(b) Compression ratio: 12.73 with transformation and, 8.90 without transformation (10 bits of quantification)



(c) Compression ratio: 9.68 with transformation and, 7.08 without transformation (11 bits of quantification).



(d) Compression ratio: 7.51 with transformation and, 5.79 without transformation (12 bits of quantification).

Figure 5.5: Speedup results

(see section 4.2). On the other hand, the transform-based algorithm uses our CDF 2.2 Lifting Wavelet architecture (see section 4.4) to implement the transformation stage.

Figure 5.5 shows the speedup results of both implementations, for different values of compression ratio.

The speedup was calculated as follows:

$$speed\_up = \frac{t_{trad}}{t_1 + t_2}, \tag{5.1}$$

where t_1 is the *compressed data transfer time*, t_2 is the decompression time and t_{trad} is the *traditional transfer time* (t_t).

As expected, the speedup improves as the number of decompression cores increases. As mentioned, the Huffman decoder performance depends on the compression ratio (see section 4.3). Therefore, the speedup improves as the compression radio increases, since the increase of the compression ratio reduces both the transfer and decompression times.

Including a transformation stage

The inclusion of a transformation improves the compression ratio (see chapter 2), which improves the Huffman decoder performance (see chapter 4). However, the inclusion of an inverse transformation stage could increase the decompression time, which compromises the overall performance.

Our CDF 2.2 Lifting Wavelet architecture [102] allowed us to implement the transform-based algorithm in a pipeline fashion. In such an implementation, the processing time is equal to the longest module in the decompression process. As the CDF 2.2 Lifting Wavelet architecture has a throughput of 2 datum per clock cycle and the inverse quantization stage has a throughput of 1 datum per clock cycle, the decompression time is governed by the decoding time. Therefore, the quantization and 1D-Transformation stages did not increase the decompression time, which explains the improved performance.

5.4 Discussion

Previous works have failed to speed up the transfer process by using compression strategies [20, 25] because of the trade-off between compression ratio and decompression time. We have implemented a optimized parallel architecture to face this trade-off.

The decompression process was implemented in a pipeline fashion, allowing the inclusion of the transformation stage without increasing the decompression time, i.e. decompression time is governed by the stage that takes more clock cycles.

In this case, the decompression time is governed by the decoding process. Therefore, the strategy depends on the compression ratio. As the compression ratio increases, the achieved speedup also increases.

Our results shows that the proposed strategy achieves a poor performance for a number of quantification bits above 11 (Figure 5.5). There are two principal reasons for this *poor* perfor-

mance.

With this number of bits of quantification, the percentage of code-words that can be decoded in parallel is reduced (see Table 4.3 in Chapter 4). In this case, the performance can be improved by extending the computational architecture to decode more code-words in parallel. This strategy increases the amount of computational resources and could decrease the operating frequency.

A second reason is due to the proposed strategy is dependent on the amount data that are transferred. As the amount of the decompressed data increases the performance is improved because t_{trad}, in Equation 5.1, is *proportional* to the amount of data. The results showed in Figure 5.5 were obtained by storing the compressed data at the maximum FPGA memory capacity.

The dotted line in Figure 5.6 estimates, by a spline interpolation, the speed up for a larger FPGA. Note that with 24 decompression cores, it will be achieved a speed up of $4\times$. Our results are inconclusive regarding the maximum number of cores that can operating currently into the parallel implementation.

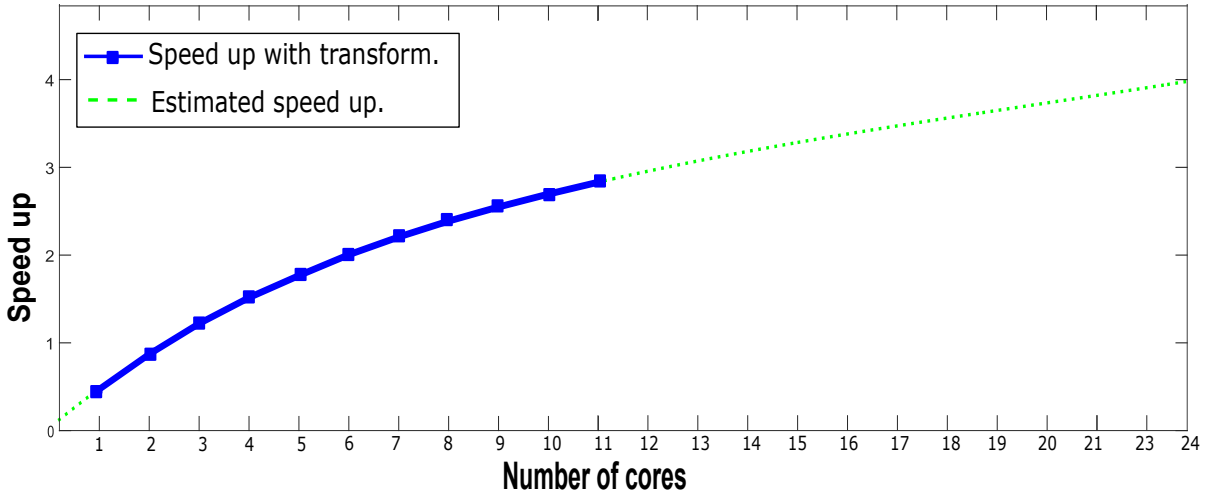


Figure 5.6: Estimated speed up for a compression ratio of 7.51 (12 bits of quantification)

Chapter 6

Conclusions

For intensive computing algorithms, heterogeneous clusters have demonstrated to be more efficient than homogeneous ones, when the algorithms can take advantage of the computing capability of parallel co-processors [117].

The implementation of these applications on heterogeneous clusters still have computational challenges. The I/O operations between *main memory* and *node memory* are one of the most limiting factors in the overall performance in these heterogeneous clusters. This I/O bottleneck is directly related to the well-known *memory wall* problem, which has been well studied [15, 118, 119] and addressed from many points of view [81, 83, 84].

This bottleneck has drawn attention over the last years because the overall performance in the heterogeneous clusters is mainly determined by the performance of these I/O operations rather than the processing speed of the computational co-processors [16, 17, 120].

We propose a strategy based on an optimized compression/decompression process to reduce the impact of the I/O bottleneck in a heterogeneous cluster. The strategy involves to compress the seismic data *on-site* while they are being acquired. The transfer operations from *disk* to *node memory* are done by using the compressed data to reduce both the load time from the *disk* and the transfer time through the PCIe bus.

The implementation of the strategy in a HPC heterogeneous cluster faced us against some challenges. It was required to propose a compression/decompression strategy that offered a high performance in terms of both compression ratio and decompression time. However, there is a trade-off between these two variables.

A key aspect to implement the decoding process into a GPU-based cluster was the use of a strategy that uses packets with headers. This strategy seeks to force the alignment of the

code words at packet boundaries, allowing us to parallelize the decoding process. However, the use of packets with headers reduces the compression ratio, which also reduces the throughput of the decoding process (see Chapter 3). Additionally, processing the code-words at bit level generates a time overhead in the GPU, because this architecture is optimized to work at single and double floating-point format. Thus, the obtained throughput was not enough to fulfill the time constrains established in the proposed strategy.

The GPU implementation allowed us to identify the necessity of a custom architecture to optimize the decompression process and its specific requirements. This computational architecture must include:

- A custom memory system to save the encoded data that optimize the storing of the string of compressed data without affecting the compression ratio.
- A custom bank registers that provides a fast storing of the temporal data required in the decompression process.
- Hardwired instructions to perform the decoding process at bit level.
- A pipeline fashion design to improve the throughput.

This custom architecture is mainly based on our Huffman decoder version 2.1 (see section 4.2) and our CDF 2.2 Lifting Wavelet architecture (see section 4.4). The optimized architecture allows a reduction in the number of clock cycles required to decompress a single sample. This reduction depends on the compression ratio. Our architecture improves the throughput several ten-folds when it is compared with our optimized GPU implementation or with similar research works (see section 4.3).

As mentioned, the performance of the compression strategy depends on the compression ratio. As floating-point format applications require lossy compression methods to reach ratios greater than 2:1 [37], our strategy is only feasible for those floating-point format applications that allow lossy compression.

The results show that our strategy requires compressing the seismic data with 9 bits of quantification to achieve a speedup above of 9×, for a parallel implementation with 11 decompression cores (Figure 5.5a). In this case, a compression ratio of 16.32 is achieved for this number of bits of quantification.

The strategy requires compressing the seismic data with 12 bits of quantification to double the performance of the transfer operations, in an implementation with 11 decompression cores (Figure 5.5d). In this case, a compression ratio of 7.51 is achieved for this number of bits of quantification.

As expected, the proposed strategy is limited by the I/O bottleneck itself, which means that the number of the decompression cores working concurrently is limited by the amount of seismic data that can be sent to the FPGA. Our results are inconclusive regarding the maximum number of cores that can operating currently into the parallel implementation.

The transformation stage was implemented by using the CDF 2.2 biorthogonal filter to reduce the amount of logical resources and improve the computational performance. Each transformation core offers a throughput of two samples per clock cycle. On the other hand, the decoding stage needs 10 clock cycles, in average, to decode a single sample. As these stages were implemented in pipeline, the transformation stage stays idle for 8 clock cycles, in average, because of the difference between these throughputs. In this sense, the inclusion of another type of filter can increase the compression ratio, at the same level of SNR (see section 2.4) without increasing the decompression time. This inclusion can improve the overall performance in the decompression strategy.

Implications and impact

The next generation of HPC nodes will include application-specific integrated circuits (ASICs) because of power and cooling constraints [121, 122, 123]. Our compression/decompression strategy can be implemented into these ASIC-based clusters by implementing the proposed architecture into the ASIC.

As an ASIC implementation has a higher speed than an FPGA implementation [124], the strategy in these ASIC-based clusters can offer a higher performance than in a FPGA-based cluster.

Towards the use of the compressed data directly

We use the best variable length coding scheme among the methods that use code-words of integer length (see section 1.1), which allows us to achieve a compression ratio very close to the optimum at a relatively low computational cost (see section 1.4). However, the compressed sequence, i.e.

the bit-stream with the code-words, is useful only for reducing the size of the data, but not for processing directly.

To find a compression strategy that allows processing the compressed data directly can achieve significant savings in I/O efforts and computational cost. This compression strategy must not be based on an entropy coding, because this compression scheme offers a bit-stream that does not have a mathematical relationship with the original data.

The use of a compression algorithm such as Matching Pursuit [125], which compress the data by a mathematical approximation could be an option in this sense.

Future Work

As the overall performance in the heterogeneous clusters is mainly determined by the performance of the I/O operations rather than the processing speed of the computational co-processors [16, 17, 120], our strategy can improve the cluster's overall performance because it reduces the impact of the I/O bottleneck. However, our results are inconclusive regarding the impact of the proposed strategy in the cluster's overall performance.

We are interested to test the impact of the proposed strategy in the overall performance in an FPGA-based cluster for a specific application, such as seismic migration. In this sense, it is required to select and implement a specific seismic migration (e.g. Reverse Time Migration) in an FPGA-based cluster and test the overall performance with and without the compression strategy.

On the other hand, we are interested in exploring the possibility to develop a seismic migration by using the compressed data directly. A seismic migration on a compressed domain raises several questions for further research: What is it the most appropriate compression algorithm?, Which seismic migration is feasible to develop in a compressed domain?, Is it required to develop a *new* migration operator?

References

[1] Tong Chen. Seismic Data compression: a tutorial. http://www.cwp.mines.edu/Documents/cwpreports/cwp-180.pdf, 1995.

[2] Sercel. What is Geophysics. `http://www.sercel.com/about/Pages/what-is-geophysics.aspx`. [online] Accessed January-2016.

[3] Carlos A. Fajardo, Javier Castillo Villar, and Cesar Pedraza. Reducción de los tiempos de cómputo de la Migración Sísmica usando FPGAs y GPGPUs: Un artículo de revisión. *Ingenería y Ciencia*, 9(17):261–293, 2013.

[4] Mauricio Araya-polo, Javier Cabezas, Mauricio Hanzich, Miquel Pericas, Isaac Gelado, Muhammad Shafiq, Enric Morancho, Nacho Navarro, Mateo Valero, and Eduard Ayguade. Assessing Accelerator-Based HPC Reverse Time Migration. *Electronic Design*, 22(1):147–162, 2011.

[5] Robert G. Clapp, Haohuan Fu, and Olav Lindtjorn. Selecting the right hardware for reverse time migration. *The Leading Edge*, 29(1):936 – 944, 2010.

[6] Key World Energy Statistics, 2014. International Energy Agency. Paris, 2014. Available: http://www.iea.org/publications/freepublications/publication/keyworld2014.pdf.

[7] Larry Hughes and Jacinda Rudolph. Future world oil production: Growth, plateau, or peak? *Current Opinion in Environmental Sustainability*, 3(4):225–234, 2011.

[8] Christine Jojarth. The end of easy oil: estimating average production costs for oil fields around the world. *Center on Democracy, Development, and The Rule of Law-Stanford, Published by Program on Energy and Sustainable Development Working Paper*, 72, 2008.

[9] SZ Sun, WK Yang, YM Ma, LJ Zhang, LW Yu, and XK Sun. Accelerating 3d prestack reverse time migration by the gpu-based parallel pseudospectral method. In *77th EAGE Conference and Exhibition 2015*, 2015.

[10] Rengan Xu, Maxime Hugues, Henri Calandra, Sunita Chandrasekaran, and Barbara Chapman. Accelerating Kirchhoff migration on GPU using directives. In *Accelerator Programming using Directives (WACCPD), 2014 First Workshop on*, pages 37–46. IEEE, 2014.

[11] HP Knibbe. *Reduction of computing time for seismic applications based on the Helmholtz equation by Graphics Processing Units*. PhD thesis, Delft University of Technology, Ph.D. Dissertation, 2015.

[12] Fei Han and Sam Z Sun. GPU acceleration of amplitude-preserved Q compensation prestack time migration. *Computers & Geosciences*, 82:214–224, 2015.

[13] Guofeng Liu, Yaning Liu, Li Ren, and Xiaohong Meng. 3D seismic reverse time migration on GPGPU. *Computers & Geosciences*, 59:17–23, 2013.

[14] Fang Lv, Lei Liu, Hui-min Cui, Lei Wang, Ying Liu, Xiao-bing Feng, and Pen-Chung Yew. Wisethrottling: a new asynchronous task scheduler for mitigating i/o bottleneck in large-scale datacenter servers. *The Journal of Supercomputing*, pages 1–40, 2015.

[15] Sally a. McKee. Reflections on the memory wall. *Proceedings of the first conference on computing frontiers on Computing frontiers - CF'04*, pages 162–167, 2004.

[16] Victor W. Lee, Per Hammarlund, Ronak Singhal, Pradeep Dubey, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, and Srinivas Chennupaty. Debunking the 100X GPU vs. CPU myth. *ACM SIGARCH Computer Architecture News*, 38(3):451, 2010.

[17] Chris Gregg and Kim Hazelwood. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. *ISPASS 2011 - IEEE International Symposium on Performance Analysis of Systems and Software*, pages 134–144, 2011.

[18] Mark Harris. Inside pascal: Nvidia's newest computing platform. https://devblogs.nvidia.com/parallelforall/inside-pascal/, jun 2016.

[19] Xilinx. Virtex ultrascale. http://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf, jun 2015.

[20] Daniel Haugen. *Seismic Data Compression and GPU Memory Latency*. Master thesis, Norwegian University of Science and Technology, 2009.

[21] Xing Xie and Qianqing Qin. Fast Lossless Compression of Seismic Floating-Point Data. In *2009 International Forum on Information Technology and Applications*, number 40204008, pages 235–238. Ieee, May 2009.

[22] Carlos Angulo, Carlos A. Fajardo, Oscar Reyes, and Javier Castillo. Fpga implementation of a Huffman decoder for high speed seismic data decompression Huffman Algorithm. In *Data Compression Conference, 2014*, pages 1–10, Salt Lake, United States., 2014. IEEE.

[23] Jairo A Castelar, Carlos A Angulo, and Carlos A Fajardo. Parallel decompression of seismic data on gpu using a lifting wavelet algorithm. In *XX Simposio Internacional de Tratamiento de Señales, Imágenes y Visión Artificial – STSIVA 2015*, 2015.

[24] Ahmed Adnan Aqrawi. *Effects of Compression on Data Intensive Algorithms*. Master thesis, Norwegian University of Science and Technology, 2010.

[25] Ritesh a. Patel, Yao Zhang, Jason Mak, Andrew Davidson, and John D. Owens. Parallel lossless data compression on the GPU. In *2012 Innovative Parallel Computing, InPar 2012*, 2012.

[26] Ahmed a. Aqrawi and Anne C. Elster. Bandwidth Reduction through Multithreaded Compression of Seismic Images. *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1730–1739, May 2011.

[27] David Salomon. *Data Compression The Complete Reference*. Springer, 4 edition, 2007.

[28] Kamisetty Ramam Rao and Patrick C Yip. *The transform and data compression handbook*. CRC press, 2000.

[29] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, XXVII(3):379–423, 1948.

[30] David A Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[31] Robert G Gallager. Variations on a theme by huffman. *Information Theory, IEEE Transactions on*, 24(6):668–674, 1978.

[32] Jorma Rissanen and Glen G Langdon. Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162, 1979.

[33] Sebastian Deorowicz. *Universal lossless data compression algorithms*. Doctor of philosophy dissertation, Silesian University of Technology, 2003.

[34] Kevin Reddy. Displaying Seismic Data: Part 7. Geophysical Methods. In Diana Morton-Thompson and Arnold M Woods, editors, *Development Geology Reference Manual*, chapter 10, pages 377–378. The American Association of Petroleum Geologist, 1992.

[35] R Tage and Tor A Ramstad. Efficient Lossless Compression of seismic trace headers using conditional Adaptive Arithmetic Coding. In *62nd EAGE conference*, pages 1–6, Glasgow, Scotland, 2000. European Association of Geoscientists and Engineers (EAGE).

[36] KM Barry, DA Cavers, and CW Kneale. (SEG-Y)Recommended standards for digital tape formats. *Geophysics*, 40(2):344–352, 1975.

[37] Tage Rosten. *Seismic Data Compression using subband coding*. PhD thesis, Norwegian University of Science and Technology, 2000.

[38] Laurent C. Duval. Simultaneous seismic compression and denoising using a lapped transform coder. In *IEEE International Conference on Acoustics Speech and Signal Processing*, pages 1269–1272. Ieee, 2002.

[39] Amir Z Averbuch, F Meyer, J O Stromberg, R Coifman, and A Vassiliou. Low bit-rate efficient compression for seismic data. *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, 10(12):1801–14, jan 2001.

[40] Mohammed A T. Towards an Efficient Compression Algorithm for Seismic Data. *Radio Science Conference, 2004.*, pages 550–553, 2004.

[41] Cheryl Bosman and Edmund Reiter. Seismic Data Compression Using Wavelet Transforms. In *In 1993 SEG Annual Meeting*, pages 1261–1264. Society of Exploration Geophysicists., 1993.

[42] Stephen D Shapiro. Use of the hough transform for image data compression. *Pattern Recognition*, 12(5):333–337, 1980.

[43] Norman B Nill. A visual model weighted cosine transform for image compression and quality assessment. *Communications, IEEE Transactions on*, 33(6):551–557, 1985.

[44] L Torres-Urgell and R Lynn Kirlin. Adaptive image compression using karhunen-loeve transform. *Signal processing*, 21(4):303–313, 1990.

[45] William R Zettler, John C Huffman, and David CP Linden. Application of compactly supported wavelets to image compression. In *Electronic Imaging'90, Santa Clara, 11-16 Feb'92*, pages 150–160. International Society for Optics and Photonics, 1990.

[46] Herbert Lohscheller. Vision adapted progressive image transmission. *Proceeding. r of ELISIPCO-83*, pages 191–194, 1983.

[47] Overview of JPEG. `https://jpeg.org/jpeg/index.html`, 1990. [online] Accessed December-2015.

[48] P. L. Dohono J. D. Villasenor, R. A. Ergas. Seismic Data Compression Using High-Dimensional Wavelet Transforms. In *Data Compression Conference, 1996.*, pages 396–405, 1996.

[49] a.S. Spanias, S.B. Jonsson, and S.D. Stearns. Transform methods for seismic data compression. *IEEE Transactions on Geoscience and Remote Sensing*, 29(3):407–416, may 1991.

[50] R.D. Dony. Karhunen-Loève Transform. In *The Transform and Data Compression Handbook*, chapter 1. CRC Press LLC, 2001.

[51] Marc Antonini, Michel Barlaud, Pierre Mathieu, and Ingrid Daubechies. Image coding using wavelet transform. *Image Processing, IEEE Transactions on*, 1(2):205–220, 1992.

[52] Ronald A DeVore, Björn Jawerth, and Bradley J Lucier. Image compression through wavelet transform coding. *Information Theory, IEEE Transactions on*, 38(2):719–746, 1992.

[53] Adrian S Lewis and G Knowles. Image compression using the 2-d wavelet transform. *Image Processing, IEEE Transactions on*, 1(2):244–250, 1992.

[54] Antonin Chambolle, Ronald A De Vore, Nam-Yong Lee, and Bradley J Lucier. Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet shrinkage. *Image Processing, IEEE Transactions on*, 7(3):319–335, 1998.

[55] Anthony A Vassiliou and Mladen V Wickerhouser. Comparison of wavelet image coding schemes for seismic data compression. In *Optical Science, Engineering and Instrumentation'97*, pages 118–126. International Society for Optics and Photonics, 1997.

[56] Wang Xi-zhen, Then Yun-tina, Gao Meng-tan, and Jiang Hui. Seismic data compression based on integer wavelet transform. *Acta Seismologica Sinica*, 17(04):123–128, 2004.

[57] Laurent C Duval and Truong Q Nguyen. Seismic data compression: a comparative study between GenLOT and wavelet compression. *Proc. SPIE, Wavelets: Appl. Signal Image Process.*, 3813:802–810, 1999.

[58] Laurent C Duval, Truong Q Nguyen, and Trac D Tran. Seismic data compression and QC using GenLOT. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, volume 3, pages 1765 – 1768 vol.3, 2001.

[59] L.G. Duval, T.Q. Nguyen, and T.D. Tran. GenLOT optimization techniques for seismic data compression. *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, 4:2111–2114, 2000.

[60] M.F. Khene and S.H. Abdul-Juawad. Adaptive seismic compression by wavelet shrinkage. In *Statistical Signal and Array Processing, 2000. Proceedings of the Tenth IEEE Workshop on*, pages 544–548, 2000.

[61] Laurent C. Duval and Takayuki Nagai. Seismic data compression using Gullots. *Proc. Int. Conf. Acoust. Speech Signal Process.*, 3:1765–1768, 2001.

[62] Tage Rosten, Tor A. Ramstad, and Lasse Amundsen. Optimization of sub-band coding method for seismic data compression. *Geophysical Prospecting*, 52(5):359–378, sep 2004.

[63] W. Wu, Z. Yang, Q. Qin, and F. Hu. Adaptive Seismic Data Compression Using Wavelet Packets. *2006 IEEE International Symposium on Geoscience and Remote Sensing*, (3):787–789, jul 2006.

[64] Laurent C. Duval and V. Buitran. Compression denoising: using seismic compression for uncoherent noise removal. *EAGE 63rd Conference & Technical Exhibition—Amsterdam, The Netherlands*, (June), 2001.

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

[66] Jerome M Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *Signal Processing, IEEE Transactions on*, 41(12):3445–3462, 1993.

[67] Xing Xie and Qianqing Qin. Fast lossless compression of seismic floating-point data. In *Information Technology and Applications, 2009. IFITA'09. International Forum on*, volume 1, pages 235–238. IEEE, 2009.

[68] John M Lervik, T Rosten, and Tor A Ramstad. Subband Seismic Data Compression: Optimization and Evaluation. *Digital Signal Processing Workshop Proceedings*, (1):65–68, 1996.

[69] Xi-zhen Wang, Yun-tian Teng, Meng-tan Gao, and Hui Jiang. Seismic data compression based on integer wavelet transform. *Acta Seismologica Sinica*, 17(1):123–128, 2004.

[70] Gordon V Cormack and R Nigel Horspool. Algorithms for adaptive huffman codes. *Information Processing Letters*, 18(3):159–165, 1984.

[71] Donald E Knuth. Dynamic huffman coding. *Journal of algorithms*, 6(2):163–180, 1985.

[72] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *Journal of the ACM (JACM)*, 34(4):825–845, 1987.

[73] Jeffrey Scott Vitter. Algorithm 673: dynamic huffman coding. *ACM Transactions on Mathematical Software (TOMS)*, 15(2):158–167, 1989.

[74] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[75] Paul G Howard and Jeffrey Scott Vitter. Analysis of arithmetic coding for data compression. In *Data Compression Conference, 1991. DCC'91.*, pages 3–12. IEEE, 1991.

[76] Paul G Howard and Jeffrey Scott Vitter. *Practical implementations of arithmetic coding.* Springer, 1992.

[77] Peter M Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24(3):327–336, 1994.

[78] Alistair Moffat. An improved data structure for cumulative probability tables. *Software: Practice and Experience*, 29(7):647–659, 1999.

[79] Rajeev Balasubramonian, David Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 245–257. ACM, 2000.

[80] Hongtu Jiang and V. Owall. FPGA implementation of real-time image convolutions with three level of memory hierarchy. In *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on*, pages 424–427, Dec 2003.

[81] H. Morishita, Y. Osana, N. Fujita, and H. Amano. Exploiting memory hierarchy for a Computational Fluid Dynamics accelerator on FPGAs. In *ICECE Technology, 2008. FPT 2008. International Conference on*, pages 193–200, Dec 2008.

[82] Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore GPUs. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[83] Yao Zhang and John D Owens. A quantitative performance analysis model for GPU architectures. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 382–393. IEEE, 2011.

[84] (performance modeling for highly-threaded many-core gpus.

[85] S. T. Klein. Parallel Huffman Decoding with Applications to JPEG Files. *The Computer Journal*, 46(5):487–497, may 2003.

[86] C.A. Angulo, C.D. Hernandez, G. Rincon, C.A. Boada, J. Castillo, and C.A. Fajardo. Accelerating huffman decoding of seismic data on GPUs. In *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, pages 1–6, Sept 2015.

[87] Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. Technical Report Technical Report SRC- RR-124, Digital SRC Research Report, 1994.

[88] Jon Louis Bentley, Daniel D Sleator, Robert E Tarjan, and Victor K Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[89] Andrew Davidson, David Tarjan, Michael Garland, and John D Owens. Efficient parallel merge sort for fixed and variable length keys. In *Innovative Parallel Computing (InPar), 2012*, pages 1–9. IEEE, 2012.

[90] A. Gercho and R. Gray. *Vector quantization and signal compression*. Kluwer Academic Publiser, 1992.

[91] Carlos A. Fajardo, Óscar M. Reyes, and Ana Ramirez. Seismic Data Compression Using 2D Lifting-Wavelet Algorithms. *Ingenería y Ciencia*, 11(21):221–238, 2015.

[92] Ronald E Walpole, Raymond H Myers, Sharon L Myers, and Keying Ye. Simple Linear Regression and Correlation. In *Probability and statistics for engineers and scientists*, chapter 11, pages 389–444. Macmillan New York, 2007.

[93] Ron Larson and David C. Falvo. *Elementary Linear Algebra*. Cengage Learning, Boston, sixth edition, 2009.

[94] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[95] S. Jana and P. Moulin. Optimality of klt for high-rate transform coding of gaussian vector-scale mixtures: Application to reconstruction, estimation, and classification. *IEEE Transactions on Information Theory*, 52(9):4049–4067, Sept 2006.

[96] P. Aparna and Sumam David. Adaptive Local Cosine transform for Seismic Image Compression. *2006 International Conference on Advanced Computing and Communications*, (x):254–257, dec 2006.

[97] Wim Sweldens. Lifting scheme: a new philosophy in biorthogonal wavelet constructions. In *SPIE's 1995 International Symposium on Optical Science, Engineering, and Instrumentation*, volume 2569, pages 68–79. International Society for Optics and Photonics, sep 1995.

[98] Kishore Andra, Chaitali Chakrabarti, and Tinku Acharya. A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform. *Signal Processing, IEEE Transactions on*, 50(4):966–977, 2002.

[99] Maarten H Jansen and Patrick J Oonincx. *Second generation wavelets and applications*. Springer Science & Business Media, 2005.

[100] Wim Sweldens. Wavelets and the Lifting Scheme : A 5 Minute Tour. *ZAMM-Zeitschrift fur Angewandte Mathematik und Mechanik*, 76(2):41–47, 1996.

[101] D. Lee Fugal. *Conceptual Wavelets In Digital Signal Processing*. Technologies, Space & Signal, San Diego, California, 2009.

[102] Fabian Sánchez, Carlos A Fajardo, Carlos A Angulo, Oscar M Reyes, and Charles A Bouman. A computational architecture for Discrete Wavelet Transform using Lifting Scheme. In *XIX Simposio Internacional de Tratamiento de Señales, Imágenes y Visión Artificial – STSIVA 2014*, pages 1–4, 2014.

[103] Brian Parker Tunstall. *Synthesis of noiseless compression codes*. Georgia Institute of Technology, Ph.D Thesis, , 1967.

[104] S. Cook. *Cuda Programming. A Developer's Guide to Parallel Computing with GPUs*. Elsevier, 1 edition, 2013.

[105] Laurentiu Acasandrei and Marius Neag. A fast parallel huffman decoder for fpga implementation. *Acta Technica Napocensis, Electronics and Telecommunications*, 49(1):8–15, 2008.

[106] Seunghyun Beak, B. Van Hieu, G. Park, Kyungtaek Lee, and Taikyeong Jeong. A New Binary Tree Algorithm Implementation with Huffman Decoder on FPGA. In *Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on*, pages 437–438. IEEE, 2010.

[107] S. Beak, B. Van Hieu, H. Lee, S. Choi, I. Kim, K. Lee, Y. Lee, and T. Jeong. Novel binary tree Huffman decoding algorithm and field programmable gate array implementation for terrestrial-digital multimedia broadcasting mobile handheld. *IET Science, Measurement & Technology*, 6(6):527, 2012.

[108] Ingrid Daubechies and Wim Sweldens. Factoring wavelet transforms into lifting steps. *Journal of Fourier analysis and applications*, 4(3):247–269, 1998.

[109] Zulfakar Aspar, Zulkalnain Mohd Yusof, and Ishak Suleiman. Parallel Huffman Decoder with an optimize Look up table option on FPGA. In *TENCON 2000*, pages 73–76, 2000.

[110] Chung-jr Lian, Kuan-FU Chen, Hong-HUi Chen, and Liang-gee Chen. Lifting Based Discrete Wavelet Transform Architecture for JPEG2000. In *IEEE International Symposium on Circuits and Systems, Sydney, Australia*, pages 445–448, 2001.

[111] Chao-Tsung Huang, Po-Chih Tseng, and L. G. Chen. Flipping structure: an efficient VLSI architecture for lifting-based discrete wavelet transform. *Signal Processing, IEEE Transactions on*, 52(4):1080–1089, April 2004.

[112] Hongyu Liao, Mrinal Kr Mandal, Senior Member, and Bruce F Cockburn. Efficient Architectures for 1-D and 2-D Lifting-Based Wavelet Transforms. *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, 52(5):1315–1326, 2004.

[113] Zhigang Wu and Wei Wang. Pipelined architecture for FPGA implementation of lifting-based DWT. In *2011 International Conference on Electric Information and Control Engineering*, pages 1535–1538. IEEE, apr 2011.

[114] John Ayer. Using the Memory Endpoint Test Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores. http://www.xilinx.com/support/documentation/application_notes/xapp1022.pdf, 2010.

[115] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc., 2005.

[116] Xilinx. LogiCORE IP Endpoint Block Plus v1.14 for PCI Express. http://www.xilinx.com/, 2010.

[117] Andre R Brodtkorb, Christopher Dyken, Trond R Hagen, and Jon M Hjelmervik. State of the art in heterogeneous computing. *Scientific Programming*, 18:1–33, 2010.

[118] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.

[119] Matthew Naylor, Paul J Fox, A Theodore Markettos, and Simon W Moore. Managing the fpga memory wall: Custom computing or vector processing? In *2013 23rd International Conference on Field programmable Logic and Applications*, pages 1–6. IEEE, 2013.

[120] Ravi Jain, John Werth, and JC Browne. I/O in parallel and distributed systems: An introduction. In *Input/Output in Parallel and Distributed Computer Systems*, pages 3–30. Springer, 1996.

[121] Jens Krueger, David Donofrio, John Shalf, Marghoob Mohiyuddin, Samuel Williams, Leonid Oliker, and Franz-Josef Pfreund. Hardware/software co-design for energy-efficient seismic modeling. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 73. ACM, 2011.

[122] Gloria Ortega López. *High performance computing for solving large sparse systems. Optical diffraction tomography as a case of study*, volume 335. Universidad Almería, 2015.

[123] Farzad Fatollahi-Fard, Dave Donofrio, and John Shalf. Emulating future hpc soc architectures using risc-v. In *RISC-V Workshop*, jan 2016.

[124] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 26(2):203–215, 2007.

[125] S.G. Mallat. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[126] Ecopetrol. `http://www.ecopetrol.com.co/wps/portal/es`, 2016. [online] Accessed November-2015.

Bibliography

Albertin, A., Kapoor, J., Randall, R., & Smith, M. (2002). La era de las imgáenes en escala de profundidad. Oilfield Review, 2–17.

Araya-polo, M., Cabezas, J., Hanzich, M., Pericas, M., Gelado, I., Shafiq, M., ... Ayguade, E. (2011). Assessing Accelerator-Based HPC Reverse Time Migration. Electronic Design, 22(1), 147–162. Chu, P. P. (2006). RTL Hardware Design Using VHDL. John Wiley & Sons.

Elad, M. (2010). Sparse and Redundant Representations. New York: Springer.

Fugal, D. L. (2009). Conceptual Wavelets In Digital Signal Processing. (S. & S. Technologies, Ed.). San Diego, California.

Gazdag, J., & Sguazzero, P. (1984). Migration of seismic data. Proceedings of the IEEE, 72(10), 1302–1315.

Gercho, A., & Gray, R. (1992). Vector quantization and signal compression. Kluwer Academic Publiser.

Hill, M. D., & Marty, M. R. (2008). Amdahl's Law in the Multicore Era. Computer, 41(7), 33–38.

John L. Hennesy, & David A. Patterson. (2012). Computer Architecture: A Quantitative Approach - David A. Patterson (5th ed.). Elsevier.

Kernighan, B. W., & Ritchie, D. M. (1988). C Programming Language (Second). Prentice Hall Software Series.

Larson, R., & Falvo, D. C. (2009). Elementary Linear Algebra (Sixth). Boston: Cengage Learning.

Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data coiupression, 30(6).

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). GPU Computing. Proceedings of the IEEE, 96(5), 879–899.

Pu, I. M. (2006). Fundamental Data Compression (1st ed.). Elsevier. Ravi Budruk, Don Anderson, & Tom Shanley. (2004). PCI Express System Architecture. Adison-Wesley.

Roos, M., & Rothe, J. (2010). Introduction to Computational Complexity (Vol. 1).

Salomon, D. (2007). Data Compression The Complete Reference (4th ed.). Springer.

Sayood, K. (2006). Introduction to Data Compression (3rd ed.). Elsevier.

Strang, G., & Nguyen, T. (1996). Wavelets and Filter Banks.

Wallwork, A. (2011). English for Writing Research Papers (1st ed.). New York: Springer.

Weeks, M. (2007). Digital signal processing using Matlab and Wavelets. Hingham, Massachusetts: Infinity Science Press.

Appendix A: Seismic data

The data sets used in this dissertation were provided by Ecopetrol Oil Company [126] that is a sponsor of this research project.

We used 12 seismic shots, where each shot is a collection of 96 seismic traces, having 3584 samples each. These seismic traces are in *common source*, i.e. the seismic data comes from a single shot and 96 receivers. The *common source* is the usually way as the seismic data are collected.

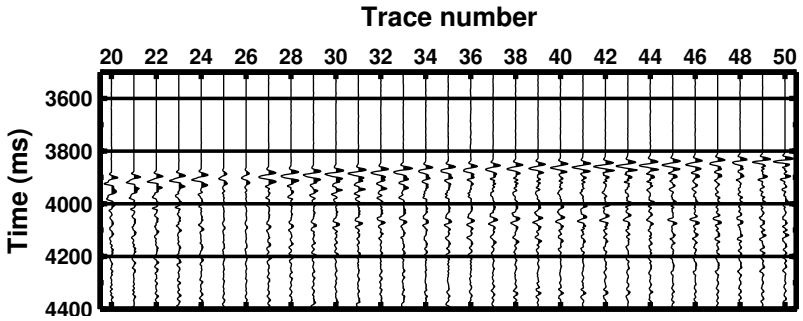
These seismic shots are part a marine seismic survey on the Caribbean coast of Colombia. The survey was carried out by Geosource Inc. Table 6.1 summarizes some technical details of this survey.

Table 6.1: Summary of technical details of the Marine Survey

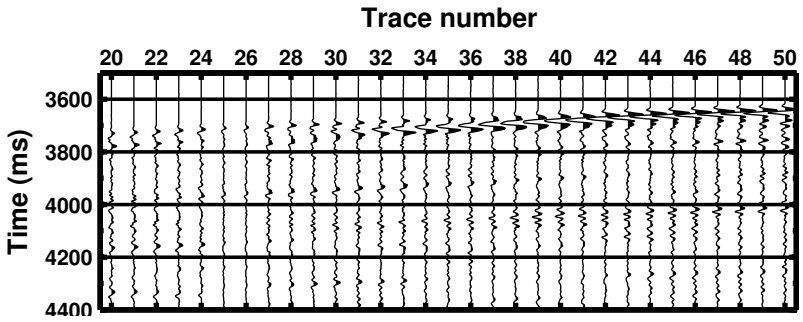
Number of total shots	3437
Number of receivers (hydrophones)	96
Length of the survey	86 Km
Offset Rank	225m – 2600m
Field Filters	8 Hz a 18db/oct
	128 Hz a 72db/oct
Acquisition time	7s
Δt (sampling period)	2ms
Source depth	9.1m
Receiver depth	15m

Figure 6.1 shows sections of four of these seismic shots. The vertical direction corresponds to the recorded curves from the sensors, in this case, hydrophones. Note that, the positive area of the curve is shaded black, while the negative side is left unfilled. This format is commonly used to enhance the visual display [34]. As shown in Figure 6.2, the relevant geophysical information

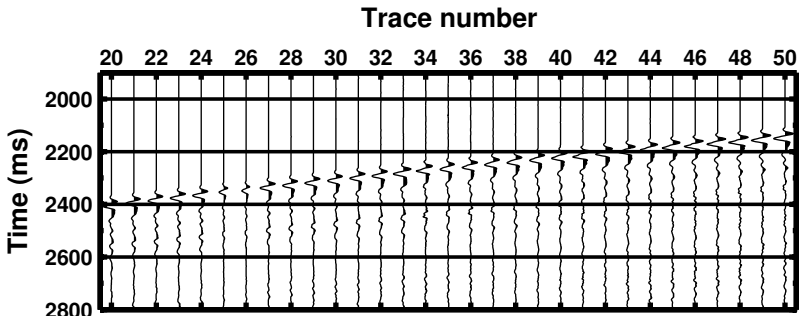
is concentrated at low frequencies.



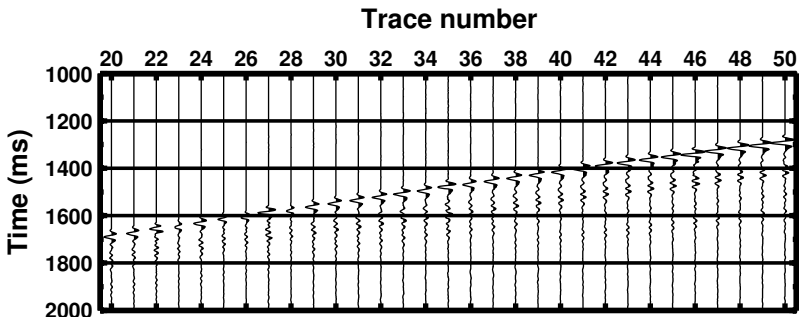
(a) Shot 1



(b) Shot 2

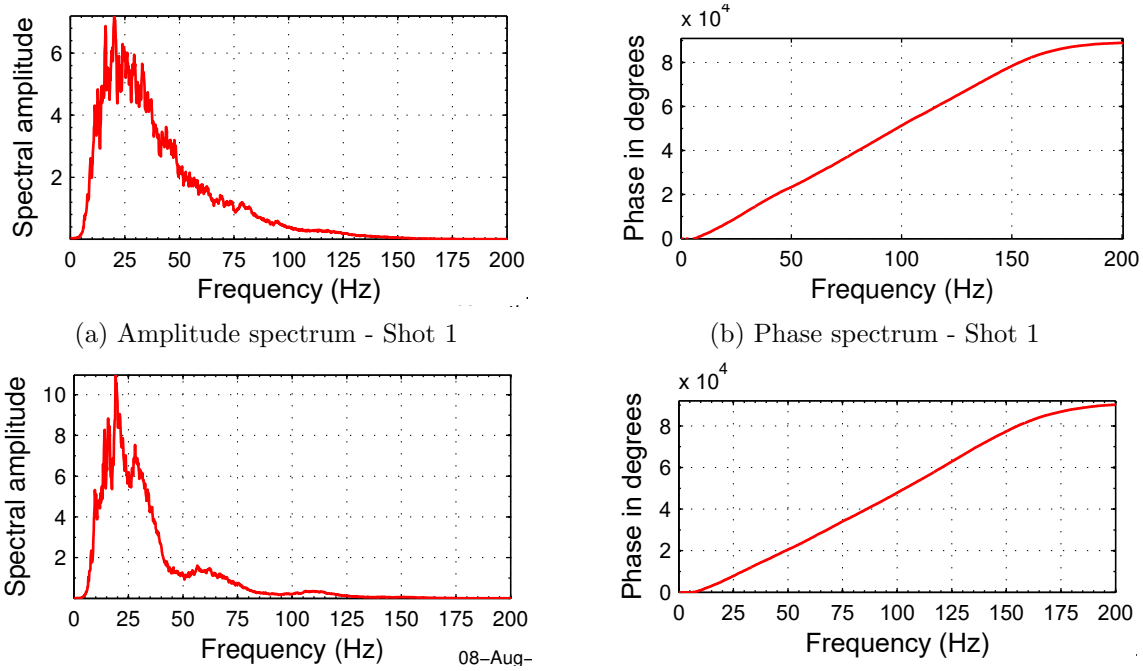


(c) Shot 3



(d) Shot 4

Figure 6.1: Seismic traces of different shots



(a) Amplitude spectrum - Shot 1

(b) Phase spectrum - Shot 1

(c) Amplitude spectrum - Shot 4

(d) Phase spectrum - Shot 4

Figure 6.2: Frequency Amplitude and Phase spectrum

Appendix B: List of Papers

The following papers that have been published based on the research work performed in this dissertation.

- C. A. Fajardo, et. al., "Computational Architecture for Fast Seismic Data Transmission between CPU and FPGA by using Data Compression", in *IEEE-Data Compression Conference*, 2016.
- C. A. Fajardo, O. M. Reyes, and A. Ramirez, "Seismic Data Compression Using 2D Lifting-Wavelet Algorithms", Ingenería y Ciencia, vol. 11, no. 21, pp. 221-238, 2015.
- C. A. Fajardo, J. Castillo, and C. Pedraza, "Reducción de los tiempos de cómputo de la Migración Sísmica usando FPGAs y GPGPUs: Un artículo de revisión", Ingenería y Ciencia, vol. 9, no. 17, 2013.
- J. A. Castelar, C. A. Angulo, and C. A. Fajardo, "Parallel decompression of seismic data on GPU using a lifting wavelet algorithm," in *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, 2015.
- C. Angulo, C. H. G. Rincón, C. A. Boada, J. Castillo, and C. A. Fajardo, "Accelerating Huffman Decoding of Seismic Data on GPUs," in *Signal Processing, Images and Computer Vision (STSIVA), 2015 20th Symposium on*, 2015.
- C. Angulo, C. A. Fajardo, O. Reyes, and J. Castillo, "FPGA implementation of a Huffman decoder for high speed seismic data decompression," in *2014 Data Compression Conference*, 2014.

- F. Sánchez, C. A. Fajardo, C. A. Angulo, O. M. Reyes, and C. A. Bouman, "A computational architecture for Discrete Wavelet Transform using Lifting Scheme," in *Signal Processing, Images and Computer Vision (STSIVA),2014 19th Symposium on*, 2014.

- S. Abreo, C. A. Fajardo, W. Salamanca, and A. Ramirez, "Alternative Computing Platforms to Implement the Seismic Migration", in *New Technologies in the Oil and Gas Industry*, Intech, 2012.