

A Partitioned Bitmask-based Technique for Lossless Seismic Data Compression

Weixun Wang
wewang@cise.ufl.edu

Prabhat Mishra
prabhat@cise.ufl.edu

CISE Technical Report 08-452
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611, USA

May 7, 2008

ABSTRACT

Seismic data costs companies and institutes millions of dollars for massive storage equipment as well as for huge data transfer bandwidth. At the same time, precision of seismic data is becoming critical for scientific analysis and research. Many existing techniques have achieved significant compression at the cost of accuracy (loss of information), or lossless compression at the cost of high computation complexity. This report addressed the problem by applying partitioned bitmask-based compression to seismic data in order to produce a significant compression without losing any accuracy. To demonstrate our approach, we compressed real world seismic data set and obtained an average compression ratio of 74% (1.35).

1. INTRODUCTION

Seismic data is used in seismology to study earthquakes and the propagation of elastic waves through the planet. It is usually collected by a series of seismometers from different locations. Seismic data compression is an area in which there are a lot of research and publications in the last decade. The motivation of compressing seismic data is that it takes enormous storage space and demands considerable transmission bandwidth. In a modern seismic acquisition survey, in which data is acquired by generating a loud sound at one location and recording through a receiver the resulting rumblings at another location, usually can produce up to 100 terabytes of data [1]. This number is still increasing in the range of multiple hundreds of terabytes nowadays due to acquisition of 3D or 4D survey instead of the traditional 2D one. Therefore, seismic data compression has become desirable and open to be exploited in order to save storage space and access time.

Before we introduce any specific technique, a primary metric for measuring the compression ratio should be decided. In the data compression research area, a widely accepted metric for measuring compression efficiency is compression ratio¹, which is used in this article, is defined as follows,

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}} \quad [1]$$

There are lots of existing researches regarding seismic data compression. Different transformations, filters and coding schemes are used to achieve great compression by losing a certain level of precision. These techniques exploit similarities of different pieces of seismic data, find approximate image partitions, condense information, and reduce statistical redundancy. However, in our observation, there is hardly any absolute repetition (of the entire 32-bit vector) even if millions of different data traces were analyzed. In order to maintain zero precision loss, those differences cannot be simply ignored or rounded. Lossless compression methods are also proposed and most of them used linear prediction as the major step. But they either require huge memory requirement or high computational complexity which will be shown in section 2.

In the code compression for embedded systems area, there is an efficient and lossless compression technique called bitmask-based method [2]. It uses dictionary-based compression to exploit existing repetitions as well as bitmask-matched repetitions by remembering the differences. Due to the similar precision requirement in seismic data compression, bitmask-based approach seems to be very promising. Unfortunately, as shown in our experiments, direct application of bitmask-based technique resulted in poor compression primarily due to the fact that there is rarely useful repetition in seismic data.

We propose an efficient seismic data compression technique based on bitmasks. It is a domain specific algorithm which can achieve moderate compression ratio without losing any precision. By choosing suitable partitions, bitmasks and dictionaries, different compression methods are applied to different partitions of every 32-bit data vectors. Heuristics are also applied to treat different parts discriminatively based on their repetitions. Our experiments demonstrate that our approach outperforms the existing bitmask-based approach by 23.8% without introducing any additional decompression overhead. The sample data used in this report is a 120-trace marine pre-stack data set recorded in the North Sea.

2. RELATED WORK

This section describes existing related work in three directions: lossy seismic data compression, lossless seismic data compression and bitmask-based code compression.

2.1 Lossy seismic data compression

Many seismic data compression researches have been carried out in the last decade. Y. Wang and R-S Wu [3] [4] applied adapted local cosine/sine transform to 2D seismic data set from which suitable window partition is generated. Duval et al. [5] used Generalized linear-phase Lapped Orthogonal Transform (GenLOT) to compress seismic data in which a filter bank optimization based on symmetric AR modeling of seismic signals autocorrelation is used. Duval et al. [6] proposed a compression scheme using Generalized Unequal Length Lapped Orthogonal Transform (GULLOT) which outperforms his former work. During the same period, wavelet-based image coding approach is also proved to be an effective compression for seismic data for its good localization in time and frequency. Various approaches [7] [8] [9] [10] have focused on this technique through different ways, making trade-offs between compression ratio and data quality in the SNR² perspective. Duval and Nguyen [11] did a comparative study between GenLOT and wavelet compression. Kiely and Pollara [12] first introduced sub-band coding methods to this area. This technique can be constructed as a progressive system in which required precision could be specified by end-users. Røsten et al. [13] presented an optimization and extension version of sub-band coding lossy seismic data compression.

¹ Larger compression ratio implies better compression result

² SNR stands for Signal-To-Noise ratio.

Although all the methods describe above got a high compression ratio, they lose some features of the original image, in another word, lose a certain level of precision. They are based on reduced redundancy due to the special property of seismic data and exploited irrelevancy, and therefore introduced information distortion. Because of this, lossy data compression methods generally represent data samples by fewer bits.

2.2 Lossless seismic data compression

As the demand of absolute accuracy grows, multiple lossless data compression techniques were proposed. Well-known examples like LZW and Deflates are not suitable for seismic data compression domain. It is primarily because that these techniques do not perform well in binary data scenario, but good at text data or text-like executable compressions. Another issue is that these techniques have high memory requirement during compressing and decompressing processes.

Stearns et al. [16] developed a lossless seismic data compression method using adaptive linear prediction. This approach is a two-stage process. In the first stage, a linear predictor with discrete coefficients is used to compress N samples of data into keywords (essentially are the first M samples, optimized coefficients and other parameters) and residue sequence, which is then further get compressed in the second stage. Several methods [15] [17] [18] also provide similar techniques by applying modifications to the used predictor or using specific coding methods. Although adaptive linear prediction has been accepted as a lossless compression and got wildly used, it requires complex computations during compression. Also these techniques require extremely complex (time-consuming) decompression mechanism. Therefore, they are not suitable where online (runtime) decompression is required.

2.3 Bitmask-based code compression

Due to similar requirements, it is intuitive to borrow some idea from the area of code compression in this scenario. Seong and Mishra [2] developed an efficient matching scheme using application-specific bitmasks and dictionaries that can significantly improve compression efficiency. It is necessary to develop a content-aware heuristic to select bitmask and dictionary entries to apply bitmask-based technique for lossless seismic data compression. According to our experiment, direct application of the algorithms in [2] to seismic data resulted in poor compression.

3. BACKGROUND

3.1 Seismic data format

Seismic data uses the SEG Y Data Exchange Format (version 0 published in 1975 and version 1 published in 2002 [14]) as its standard and dominated format. The official standard SEG-Y consists of the following components:

- a 3200-byte EBCDIC³ descriptive reel header record
- a 400-byte binary reel header record
- trace records consisting of
 - a 240-byte binary trace header
 - trace data

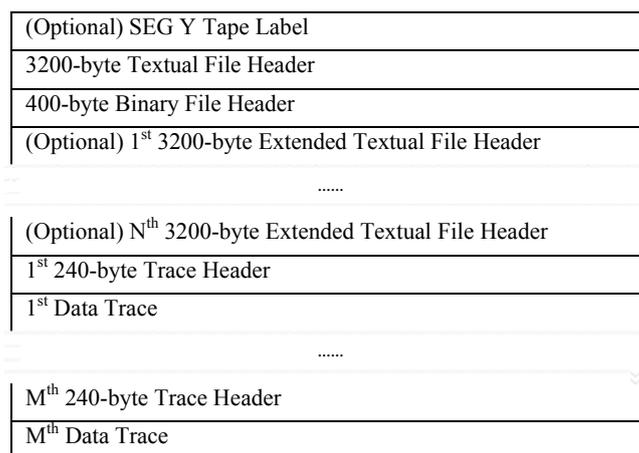


Figure 1. Byte stream structure of a SEG-Y file

³ EBCDIC : Extended Binary Coded Decimal Interchange Code (EBCDIC) is an 8-bit character encoding (code page) used on IBM mainframe operating systems.

File headers, although most of them are optional, are used to provide human-readable information about this data file. They can be compressed to a ratio larger than 2 using textual compression methods. Figure 1 shows the byte stream structure of a SEG-Y file. What we are trying to compress is the space-dominant part of a seismic data file: data traces. Specified by the reel header, the number of traces per record is usually at the magnitude of 103. A single trace is recorded from one receiver. The recording is sampled at a discrete interval, typically around 4 milliseconds, and then lasts for some duration, typically 4 or more seconds. Seismic data is almost always stored as a sequence of traces, each trace consisting of amplitude samples for one location.

3.2 Floating-point format for trace data

In a seismic trace, data is represented in IEEE Floating Point format or IBM single precision floating point format, with the later one been the most widely used nowadays. The following is a description of IBM single precision floating point format, which is also the one we used in this report.

Bit	0	1	2	3	4	5	6	7
Byte1	S	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
Byte2	Q ₋₁	Q ₋₂	Q ₋₃	Q ₋₄	Q ₋₅	Q ₋₆	Q ₋₇	Q ₋₈
Byte3	Q ₋₉	Q ₋₁₀	Q ₋₁₁	Q ₋₁₂	Q ₋₁₃	Q ₋₁₄	Q ₋₁₅	Q ₋₁₆
Byte4	Q ₋₁₇	Q ₋₁₈	Q ₋₁₉	Q ₋₂₀	Q ₋₂₁	Q ₋₂₂	Q ₋₂₃	0

Figure 2. IBM single precision floating point bit layout

Figure 2 contains the following information:

- *Sign bit (S)*: with 1 stands for negative number;
- *Hexadecimal exponent (C)*: has been biased by 64 such that it represents $16^{(CCCCCC - 64)}$ where CCCCCC can assume values from 0 to 127;
- *Magnitude fraction (Q)*: is a 23-bit positive binary fraction (i.e., the number system is sign and magnitude). The radix point is to the left of the most significant bit (Q₋₁) with the MSB being defined as 2^{-1} .

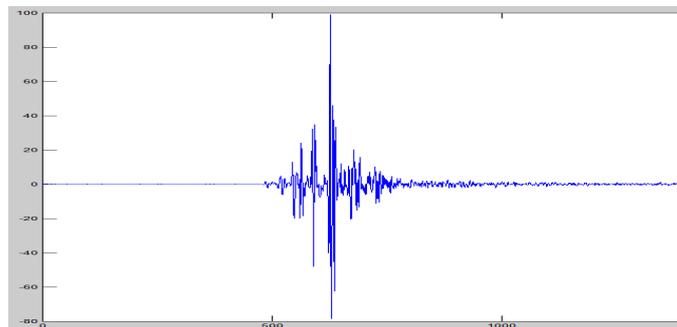


Figure 3. Seismic data trace plot diagram

Figure 3 is the plot diagram of the first data trace in the sample seismic data file. This is the first trace of all the 120 traces in that particular data record. Every single point on this curve stands for a floating point value. To apply our compression to these seismic data, we convert the original data into binary data in the IBM format. Table 1 shows the first three data out of 2560 in trace 1 of the sample seismic data file.

Table 1. Sample seismic trace data

IBM floating format value	Binary value
-0.072524	11000001000000010010100100001110
-0.037366	11000001000000001001100100001100
0.014643	01000001000000000011101111111010

As just pointed out, the first bit is the sign bit. Bits from 1st to 8th are the exponent bits and bits from 9th to 31st are magnitude bits.

4. PARTITIONED BITMASK-BASED SEISMIC DATA COMPRESSION

Based on the goal of achieving greater compression ratio without precision loss, we exploit repeating patterns as much as possible in seismic 32-bit binary data entries without affecting decompression efficiency. Our approach partitions a 32-bit vector into several parts then tries to find the best way to compress each part differently. By doing this, we can achieve the largest bit savings from each part of every seismic data entry. Table 2 shows our seismic data partition scheme used in our compression technique.

Table 2 Seismic data partition layout and examples

Sign (1 st)	Exponent (2 nd ~ 8 th)	Magnitude part 1 (9 th ~ 16 th)	Magnitude part 2 (17 th ~ 32 nd)
1	1000001	00000001	0010100100001110
1	1000010	00010011	1100010011011100
0	1000001	01000011	0101110101100010
1	1000001	00001101	0111110100100000
0	1000010	01001000	1001011000000100

4.1 Hexadecimal exponent (C₆ ~ C₀) part

We have analyzed the seismic data trace and observed that most of the exponents have a magnitude lower than 162, which means there are only two patterns for this part. Further observation revealed that among all the 120 traces in the seismic data record we used, the maximum number of different exponent part is 3. So we can use dictionary-based compression to compress this part of data using a very small dictionary whose length is at maximum 4. So the dictionary for exponent part is simply a list of all bit patterns. Figure 4 shows an example of exponent part compression. Note that the dictionary is generated, so the number of bits used as the index is decided, before each trace being compressed using our heuristic algorithm which will be introduced later.

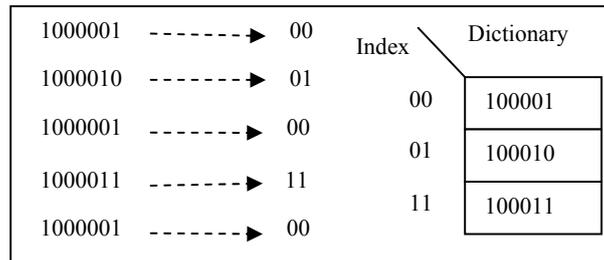


Figure 4. Exponent part compression

So the number of bits per data entry we can save from exponent part is (where n is the exponent dictionary's index length):

$$N = 8 - n \quad [2]$$

4.2 Magnitude part 1 (Q₋₁ ~ Q₋₈)

The first 8 bits in the magnitude part also shows some interesting repetition patterns. Most of them start with four 0s, which means we can use only 4 bits to compress these 8-bit vectors. For the remaining bit patterns which do not have this property, we just leave them as it is. So we have to use 5-bit to represent those who start with 0000 and 9-bit for the rest which do not. The extra bit put at the beginning of the compressed data is used to indicate whether this entry is compressed or not. Figure 5 shows an example compression for magnitude part 1.

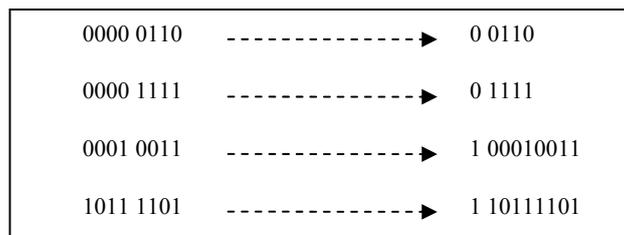


Figure 5. Magnitude part 1 compression

So the number of bits per data entry saved here is:

$$N = 8 - 5 * (C_1 / C_{total}) - 9 * (C_2 / C_{total}) \quad [3]$$

Here C_1 is the number of the vectors in the pattern of (0000xxxx); C_2 is the number of remaining vectors. C_{total} is the total number of data entries in the sample trace.

Table 3 presents three traces from the sample record file and shows how many bits we can save using the method described in section 4.1 and 4.2. For each trace, the table shows the possible bit patterns and bit savings of the exponent part and magnitude part 1. It also shows the frequency of each of these patterns. For instance, the exponent pattern “1000001” in data trace 1 appeared 98.7% of the time in the exponent partition and the patterns start with “0000” accounts for 87.0% of the magnitude part 1.

4.3 Magnitude part 2 (Q₉ ~ Q₂₃)

We use bitmask-based compression to compress the second part of the magnitude bits. Since it consists of 16 bits, a fixed 2-bit mask or a fixed 4-bit mask will be beneficial, each of which uses 5 (=2+3) or 6 (=4+2) bits respectively. In order to save bits, we are able to use dictionaries whose maximum length is 512 and 256 respectively.

We developed a new heuristic algorithm to generate the dictionary. As in the bitmask-based compression, each data vector is represented as a node. But here we attached two heuristic values to each node. One is the number of vectors which are the same as the one in node (Dictionary_Match), which means the matching count if this vector is picked as a dictionary entry. The other is the number of vectors which can be matched with the one in node using bitmask (Bitmask_Match). It depends on which bitmask pattern is used. A beneficial value is calculated for every node: (the compressed data should consist of 1 bit for compress decision, 1 bit for using bitmask or not; if a bitmask is used, then the value and location of that bitmask should also be added).

Table 3 Exploitable partition repetitions in a seismic data trace

Trace	Part name	Repetition pattern	Frequency	Bit saved
Trace 1	Exponent	1000001	98.7%	6.00
		1000010	1.3%	
	Magnitude part1	0000xxxx	87.0%	2.51
		others	13.0%	
Trace 2	Exponent	1000001	98.7%	6.00
		1000010	1.3%	
	Magnitude part1	0000xxxx	86.8%	2.47
		others	13.2%	
Trace 120	Exponent	1000001	90.9%	5.00
		1000010	8.8%	
		1000011	0.3%	
	Magnitude part1	0000xxxx	65.6%	1.62
		others	34.4%	

$$\text{Benefit} = (14 - \text{Dictionary_length}) * \text{Dictionary_Match} + (9 - \text{Dictionary_Match}) * \text{Bitmask_Match} \quad [4],$$

Here one 2-bit mask is used.

$$\text{Benefit} = (14 - \text{Dictionary_length}) * \text{Dictionary_Match} + (8 - \text{Dictionary_Match}) * \text{Bitmask_Match} \quad [5],$$

Here one 4-bit mask is used.

Each node also maintains its list of matching nodes using bitmask. Because in seismic data situation, dictionary matches are very rare. So we decided to discard the threshold scheme used in the algorithm in [2] and remove every node in a removed node's list. Figure 6 shows the dictionary generation algorithm for magnitude part 2. Note that we do not have to repeat step 2 every time because for the remaining nodes' Dictionary_Match values after every removal remain the same, for all the data entries in a same pattern are kept or discarded at the same time.

Algorithm 1: Magnitude part 2 dictionary generation

Inputs: 1. All 16-bit magnitude part 2 vectors.
2. Mask patterns.

Output: Optimized *dictionary*

Begin

Step 1: Construct a list **L** whose nodes represent each data vector in trace.

Step 2: Calculate *Dictionary_Match* for each node.

Step 3: Calculate the *Bitmask_Match* value using the input mask pattern for each vector. Generate the bitmask matching list of nodes which match this node.

Step 4: Allocate the *Benefit Value* for each nodes in the list, using equation [3] or [4].

Step 5: Select node **N** with the highest benefit value.

Step 6: Remove **N** from the list and insert into *dictionary*.

Step 7: Remove all nodes in **N**'s bitmask matching list.

Step 8: Repeat step 3 – 7 until *dictionary* is full or **L** is empty.

return *dictionary*

End

Figure 6. Magnitude part 2 dictionary generation

4.4 Complete compression algorithm

Figure 7 shows our compression algorithm. Note that the 4th step in Algorithm 2 can be executed serially or in parallel. If a parallel compression is used, attention must be paid on how to correctly connect every compressed part to get a whole compressed vector. Some synchronization should be applied to ensure the right compression sequence.

Algorithm 2: Seismic Data Compression

Inputs: 1. Original data (32-bit vectors).
2. Mask patterns.
3. Magnitude part 2 dictionary length

Output: Compressed data, dictionaries

Begin

Step 1: Generate the dictionary for exponent part.

Step 2: Analyze the two pattern counts for magnitude part 1. Calculate the bit-saving using equation [2].
if (bit-saving > 0) is_magnitude_1_compressed = true
else is_magnitude_1_compressed = false
endif.

Step 3: Generate the dictionary for magnitude part 2 using Algorithm 1.

Step 4: Compress 32-bit vectors by first partition them.
Keep the sign bit as it is.
Compress the exponent part using its dictionary.
if (is_magnitude_1_compressed)
 Compress magnitude part 1 using the method shown in Figure 5
endif.
Compress magnitude part 2 using its dictionary.
Connect four partitioned parts together

return Compressed data, dictionaries

End

Figure 7. Partitioned bitmask-based seismic data compression algorithm

For each compressed seismic data trace, an extra header is needed besides dictionaries. It should be used to reflect compression decisions made and dictionary length used when compressing the data in it. It is necessary for decompression. It contains the following three parts:

- a) Exponent part dictionary length
Length: 3 bits
Possible value: 001 ~ 111
- b) Magnitude part 1 compression decision
Length: 1bit
Possible value: “0” for uncompressed, “1” for compressed
- c) Magnitude part 2 compression decision (bitmask type / dictionary length)
Length: 4 bits
Possible value: first bit -- “0” for one fixed 2-bit mask, “1” for one fixed 4-bit mask; next three bits -- dictionary length

Figure 8 shows the format -- of a single compressed seismic data vector. As the above algorithm suggests, compression decisions are made before or during compression process. The length of compressed exponent equals to its dictionary index length. For magnitude part 1, possible lengths are 5 and 9 for compressed and uncompressed vectors respectively. The decision bits before each magnitude part 2 vector are necessary because they reveals the information of whether the data is uncompressed, compressed using bitmask or compressed using dictionary entry. The number of compressed bits is varying due to dictionary length and bitmask patterns.

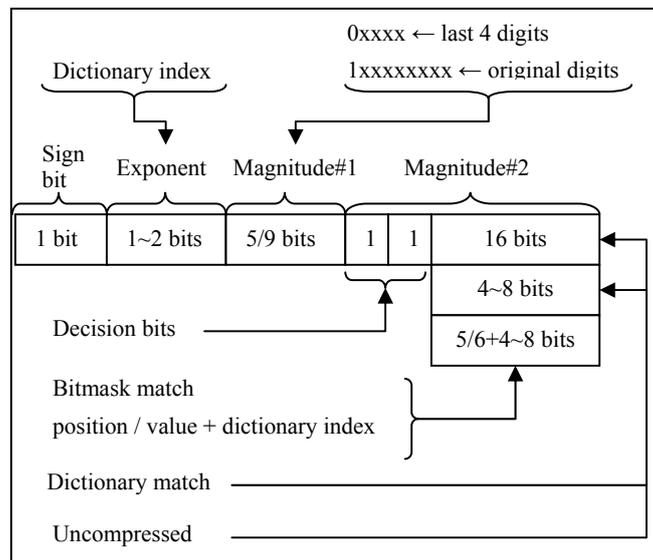


Figure 8. Final compressed seismic data layout

4.5 Decompression mechanism

The decompression process can be carried out in parallel since each trace is independent with others. So it is simple and efficient to develop a decompression unit that can run multiple decoders together during the decompression of a seismic data file which may consists of hundreds of traces.

Trace header, as described in last section, can be used to parameterize each decoder by providing the information like compression decisions and dictionary lengths. Then the decoder decompresses each partition accordingly. It simply keeps the first bit as the sign bit. It searches exponent part's dictionary, which is stored after trace header, using the index information (length is specified in header). According to magnitude part 1 compression decision, it either simply keeps all 8 bits (discards the decision bit) or adds “0000” to the front of it to decompress this part. For magnitude part 2, if there is a bitmask matching, the decoder parallelly searches for dictionary entry and construct a 16-bit mask vector then XORs those two together to get the original 16-bit data. Dictionary matching and uncompressed data is also handled accordingly.

5. EXPERIMENTS

We used a real world seismic data set recorded in the North Sea near Norway to set up our experiments. Sample data traces are exported from Matlab first in the decimal format. We then converted these data into binary using IBM floating point format.

To demonstrate our approach, first we apply bitmask-based compression alone to ten randomly chosen (1st, 13th, 27th, 36th, 42nd, 55th, 64th, 77th, 80th, 95th) traces out of all the 120 traces in the seismic data file. Each data trace has a fixed size which is 10KB if the header length is ignored. By using all the beneficial bitmask patterns and dictionary length combinations, we observed, as showed in Figure 9, that the compression ratios achieved by direct application of bitmask-based technique are unacceptable. The average compression ratio is 1.075. Obviously, these compression ratios are far from favorable enough to save storage space or transmission time. It is because there are hardly any repetition patterns which can be achieved from bitmasks in a single trace.

By using our lossless compression technique and the same ten data traces, we can achieve much better compression ratio. The sign bit, exponent part and magnitude part 1 are respectively compressed using different methods as described in section 4.1 and 4.2. The only length-varying part that can use different bitmask patterns and dictionary lengths is magnitude part 2 as stated in section 4.3. So in our experiments, we use all the beneficial (which can be deducted by equation [4] and [5]) combinations of bitmask pattern and dictionary length in that part's compression. Figure 10 shows the result of our approach. The average compression ratio is 1.331 with the maximum number of 1.353. So this experimental result reveals that, on average, our approach outperforms bitmask-based compression in seismic data compression by 23.8%.

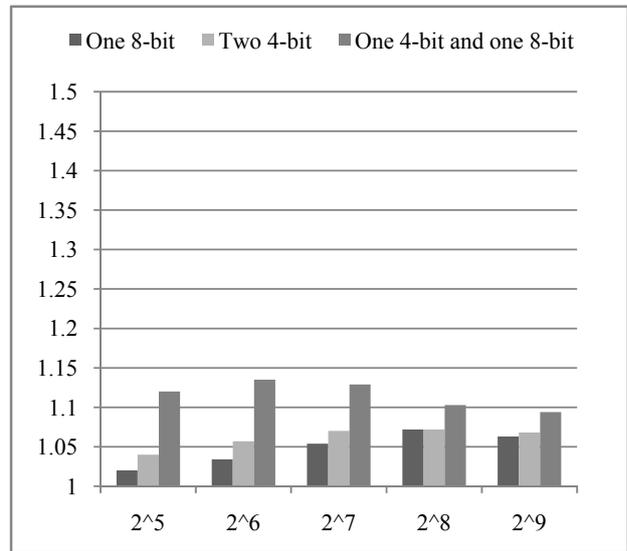
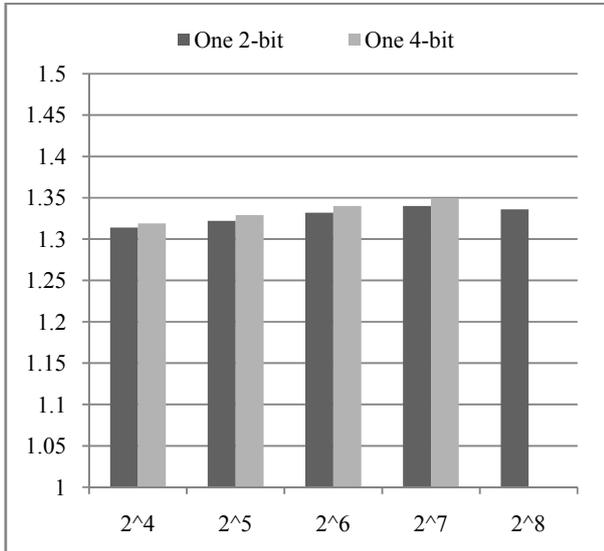


Figure 9. Seismic data compression using Bitmask directly Figure 10. Seismic data compression using our approach

6. CONCLUSION

Seismic data occupies terabytes of disk spaces. While achieving high compression ratio is desirable, maintaining an exact precision is also critical. The contribution of this report is that it develops an efficient and lossless seismic data compression technique which does not introduce any additional decompression overhead. Our partitioned seismic data compression approach outperforms the existing bitmask-based compression techniques.

7. REFERENCES

- [1] Paul L. Donoho, "Seismic data compression: Improved data management for acquisition, transmission, storage, and processing", *Proc. Seismic*, 1998
- [2] Seok-Won Seong, Prabhat Mishra, "Bitmask-Based Code Compression for Embedded Systems", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2007
- [3] Yongzhong Wang, Ru-Shan Wu, "2-D Semi-Adapted Local Cosine/Sine Transform Applied to Seismic Data Compression and its Effects on Migration", *Expanded Abstracts of the Technical Program, SEG 69th Annual Meeting, Houston*, 1999
- [4] Yongzhong Wang, Ru-Shan Wu, "Seismic Data Compression by an Adaptive Local Cosine/Sine Transform and Its Effects on Migration", *Geophysical Prospecting*, Nov., 2000

- [5] L. C. Duval, V. Bui-Tran, T. Q. Nguyen, T. D. Tran, "Seismic Data Compression Using GenLOT: Towards 'Optimality'?", *DCC* 2000
- [6] L. C. Duval, Takayuki Nagai, "Seismic Data Compression Using Gullots", *ICASSP* 2001
- [7] J. D. Villasenor, R. A. Ergas, P. L. Donoho, "Seismic data compression using high-dimensional wavelet transforms", *DCC* 1996
- [8] Anthony Vassiliou, M. V. Wickerhauser, "Comparison of Wavelet Image Coding Schemes for Seismic Data Compression" *Expanded Abstr., Int. Mtg., Soc. Exploration Geophys.*, 1997
- [9] Xi-Zhen Wang, Yun-Tian Teng, Meng-Tan Gao, Hui Jiang, "Seismic data compression based on integer wavelet transform", *Acta Seismologica Sinica*, 2004
- [10] M. A. Al-Moohimeed, "Towards an efficient compression algorithm for seismic data", *Radio Science Conference*, 2004
- [11] L. C. Duval, T. Q. Nguyen, "Seismic data compression: a comparative study between GenLOT and wavelet compression", *SPIE* 1999
- [12] A. Kiely, F. Pollara, "Subband Coding Methods for Seismic Data Compression", *DCC* 1995
- [13] T. Røsten, T. A. Ramstad, L. Amundsen, "Optimization of sub-band coding method for seismic data compression", *Geophysical Prospecting*, 2004
- [14] SEG Technical Standards Committee, "SEG Y rev 1 Data Exchange format" May, 2002
- [15] G. Mandyam, N. Magotra, W. McCoy, "Lossless Seismic Data Compression using Adaptive Linear Prediction", *GRSS*, 1996
- [16] S. D. Stearns, L. Tan, N. Magotra, "A Technique for Lossless Compression of Seismic Data", *GRSS*, 1992
- [17] Y. W. Nijim, S.D. Stearns, W. B. Mikhael, "Lossless Compression of Seismic Signals Using Differentiation", *IEEE Trans. on Geoscience and Remote Sensing, Vol 34, No. 1*, 1996
- [18] A. O. Abanmi, S. A. Alshebeili, T. H. Alamri, "Lossless Compression of Seismic Data", *Journal of the Franklin Institute*, 2006