Reducing the I/O Bottleneck by a Compression Strategy

Carlos A. Fajardo, Óscar M. Reyes, and Javier Castillo

***Abstract*—One of the main challenges in the HPC clusters is the I/O bottleneck between the main memory (disk storage system) and the node memory. This bottleneck is a consequence of the fact that the disk access speed is thousands-fold slower than the processing speed of the parallel co-processors. We propose a strategy to reduce the impact of this bottleneck in a heterogeneous cluster. We use a seismic application as case study, in this application the I/O transfer operations could consume above 80% of the overall processing time. The strategy puts forward a high-performance compression/decompression algorithm in terms of both compression ratio and processing time. The strategy is implemented in an FPGA-based cluster, which allowed us to implement a custom architecture. This architecture is optimized for both sequential and parallel stages of the decompression process. The proposed strategy can speed up the transfer operations up to $10\times$ for a compression ratio of 16:1 and up to $3\times$ 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.**

***Index Terms*—Compression strategy, FPGA, I/O bottleneck, Memory Wall.**

I. Introduction

IT is well-known that a significant challenge for any modern computational systems is the Input/Output bottleneck [1], [2], [3]. This bottleneck arises because of the amount of required memory in a single node.

The implementation of many 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.Consequently, when these 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 [1].

Manuscript received June 23, 2017; revised September 9, 2017. This research is supported by Colombian Oil Company (ECOPETROL) and the Administrative Department of Science, Technology and Innovation (COLCIENCIAS) as a part of the research projects No. 511 2010 and No. 531 2011.

C. A. Fajardo and O. M. Reyes are with Universidad Industrial de Santander, Bucaramanga, Colombia. C. A. Fajardo is also with Universidad Nacional Abierta y a Distancia, Bucaramanga, Colombia. e-mail: carlos.fajardo@unad.edu.co, omreyes@uis.edu.co

J. Castillo is with Universidad Rey Juan Carlos, Móstoles, España. e-mail: javier.castillo@urjc.es

A. Related works

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 [4].

The reorganization of the stencil calculations to take advantage of the memory hierarchies has been the subject of much investigation over the years. These investigations aim to reduce the data transfer time as much as possible, by using the faster memories closer of the co-processor. 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 [5], [6], [7].

Some research works have addressed this issue by using a compression strategy.

Aqrawi *et al.* [8] investigated how to reduce the limitations of disk I/O in a GPU-based cluster by using a compression strategy. The strategy was tested using both HDDs (Hard disk drives) and SSDs (Solid state disks). The lossy compression methods showed up to $6\times$ and $3.2\times$ faster than the traditional I/O transfer process for HDD and SDD disks respectively. It is important remark that decoding process 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.

Patel *et al.*[9] 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*. They concluded that the implementation was not fast enough to optimize data transfers between CPU and GPU.

It is important remark that in these works the decoding process was the bottleneck in the decompression process due to its sequential nature.

B. Contributions

We present a strategy to improve the transfer operations between the main memory (disk storage system) and the node

memory through the PCI Express. The strategy puts forward a high-performance compression/decompression algorithm in terms of both compression ratio and processing time.

We use a seismic application as case study. The proposed strategy involves to compress the seismic data *on-site* while they are being acquired (*off line*). 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.

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}), i.e. ,

$$t_1 + t_2 < t_{trad}. \quad (1)$$

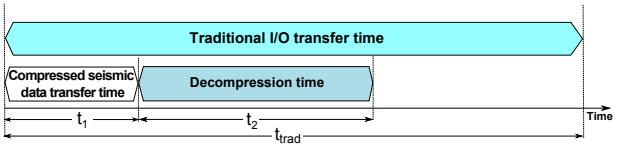


Fig. 1: Time constrains in the proposed strategy (Serial version)

It is important to highlight that we did not take into account the *compression time* because this process can be performed *on-site* (*off-line*). This compression process can be developed between repetitions of the *seismic experiment* used by the oil companies to acquire the seismic data. In our tests, compressing a seismic dataset 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* [10].

Previous works have showed 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 [9], [8].

We implemented the proposed strategy in an FPGA-based cluster by using a custom architecture optimized for both sequential and parallel stages of the decompression process.

Our 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 $10\times$ for a compression ratio of 16:1 and up to $3\times$ for a compression ratio of 7:1.

This reduction on the I/O bottleneck between the *disk* and the *node memory* could improve the overall performance in the heterogeneous clusters, especially for those applications that are limited by I/O transfer operations.

The rest of this paper is organized as follows. Section 2 shows the selection process of the compression/decompression strategy. Section 3 proposes a computational architecture for the proposed strategy. Section 4 describes the process to implement the strategy in an FPGA-based cluster. Sections 5 and 6 present present and discuss the results respectively. The conclusions are drawn in Section 7.

II. Selecting the compression-decompression scheme

Our goal is to implement a compression-decompression strategy with a high performance regarding both the compression ratio (to reduce t_1 in Equation 1) and the decompression time (to reduce t_2 in Equation 1). However, achieving better compression ratios requires to add new stages in the compression process or include new stages with major computational complexity, which increasing the processing time. Furthermore, better compression ratios are obtained by using variable-length coding schemes, like Huffman or Arithmetic coding. These coding schemes quickly become the bottleneck in the decompression process, because of its sequential nature [11], [12].

Several algorithms have been used to compress seismic data. In general, seismic data compression algorithms have three stages: transformation, quantization and coding. The main objective of these algorithms has been to compress seismic data as much as possible without time constrains. However, we are limited by Equation 1. In this sense, an analysis from both *compression ratio* and *decompression time* points of view was required in order to select the stages for the compression-decompression strategy. In our previous works such analysis has been performed [13], [12].

For the transformation stage, the Lifting-Wavelet transform was selected. The lifting scheme requires less computational and storage resources than the convolutional implementation, which reduces the processing time [14].

A uniform quantizer is generally used for seismic data compression [13] 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. Additionally, the uniform quantization has a lower computational complexity than the non-uniform one.

For the coding scheme we used the Huffman algorithm, which is the best among the methods that use code-words of integer length [15]. Additionally, this coding scheme has a low computational complexity than similar coding schemes.

III. Computational architecture for the strategy

The trade-off between the compression ratio and the processing time has not allowed the use of compression strategies to speed up the transfer process between the main memory and the device memory [8].

In previous works [16], [12], we describe our efforts in optimizing the implementation of the decompression process into the GPU architecture. The obtained throughput for the decompression process was not enough to fulfill the time constrains established in the proposed strategy [12].

These GPUs implementations allowed us to identify the necessity of a custom architecture to optimize the decompression process and its specific requirements.

The computational architecture must include:

- A custom memory system to 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.

To implement the custom architecture, we use a Field-Programmable Gate Array (FPGA). The FPGAs have been effectively used in high performance computing because they allow designers to create custom architectures, which can achieve a better performance. An FPGA implementation can be seen as a first version of an Application-specific Integrated Circuit (ASIC).

A. A computational architecture for the decoding process

One of the main challenges in this work is the implementation of the decoding process into a parallel architecture, because of its sequential nature.

Several parallel Huffman decoders have been developed for applications where the dynamic range is known and bounded, such as text, image or video applications [17], [18]. 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 another. We aim to develop a Huffman decoder that works for any seismic dataset, no matter the dictionary length and the behavior of the code-word lengths.

It was 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. In this study, we encoded 12 seismic shot. Prior to encoding the seismic data, a uniform quantization was applied to each dataset, which uses a number of quantification-bits that guarantees a SNR around 40 dB [13].

Table I shows the length of the Huffman dictionaries, which were obtained when we encoded 12 seismic shot. Note that how the dictionary lengths change from one seismic dataset to another. This table also shows results regarding compression ratio and SNR.

Table II shows the behavior of the code-words lengths and the percentage of data that is encoded by using code-words with lengths between 1 and 5 bits.

For example, Table II shows that 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. Additionally, the 58.26% of the seismic shot 1 is encoded by code-words with these lengths.

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.

It was established that more than 50% of the data were 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 with these lengths.

TABLE I: 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

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 II: 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%

We developed a computational architecture to decode in parallel the five most frequent symbols. This architecture is an improved version of our previous work [11], which was optimized to perform the decoding process at bit level.

Figure 2 compares the average number of clock cycles between our new architecture and our previous work [11]. Note that, there is a significant improvement regarding the average number of clock cycles.

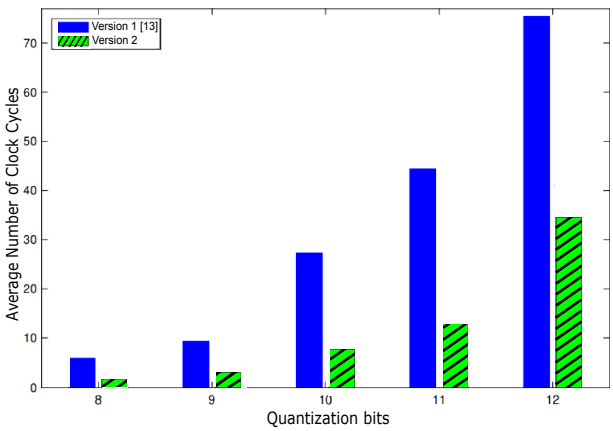


Fig. 2: Average number of clock cycles for the Huffman Decoder versions

Implementing the transformation stage

The implementation of the transformation stage was based on our previous work [19], which a computational architecture to calculate the inverse 1-D discrete wavelet transform using a lifting-based scheme for the CDF 2.2 wavelet filter.

The reconstruction of both odd and even samples requires one clock cycle [19]. This throughput allowed us to include this transformation stage in a pipeline fashion, i.e. decompression time is mainly governed by the decoding stage.

IV. Implementing the strategy

The platform used to test our strategy 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 3 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.

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.

Figure 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 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.

V. Results

Figure 5 shows the speedup in the transfer process by the proposed strategy. We compared 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.

The results are presented for different values of compression ratio and for different number of computational cores. The speedup was calculated as follows:

$$speed\_up = \frac{t_{trad}}{t_1 + t_2}, \tag{2}$$

where t_{trad} is the *traditional transfer time*, t_1 is the *compressed data transfer time* and, t_2 is the decompression time (Figure 1)

Figure 5a shows the speedup results for 12 bits of quantification. In this case a compression ratio of 7.51 is achieved for the transformation-based implementation, while a compression ratio of 5.79 is achieved for the non-transformation implementation. For the transformation-based implementation, a speedup of $2.82\times$ is achieved by 11 cores working in parallel, which gives a percentage increase of 51.3% when compared with non-transformation implementation.

Note that the speedup improves as the compression radio increases (Figures 5b,5c and, 5d), because the Huffman decoder performance depends on the compression ratio. For example, Figure 5d shows that the speedup for the transformation-based implementation is $10.33\times$ for a compression ratio of 16.32.

VI. Discussion

Previous works have failed to speed up the transfer process by using compression strategies because of the trade-off between compression ratio and decompression time [20], [9].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 mainly 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. There are two principal reasons for this *poor* performance.

At this level of compression ratio, the percentage of code-words that can be decoded in parallel is reduced. In this case, the performance can be improved by extending the computational architecture to decode more code-words in parallel.

A second reason is due to the proposed strategy is dependent on the amount data that are transferred (Equation 2). As the amount of the decompressed data increases the performance is improved because t_{trad}, in Equation 2, is *proportional* to the amount of data. Our results were obtained by storing the compressed data at the maximum FPGA memory capacity (Figure 5).

Figure 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 $4\times$.

On the other hand, 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.

VII. Conclusions

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. Our results show that the proposed strategy effectively reduces the impact of the I/O bottleneck and can improve the cluster's overall performance.

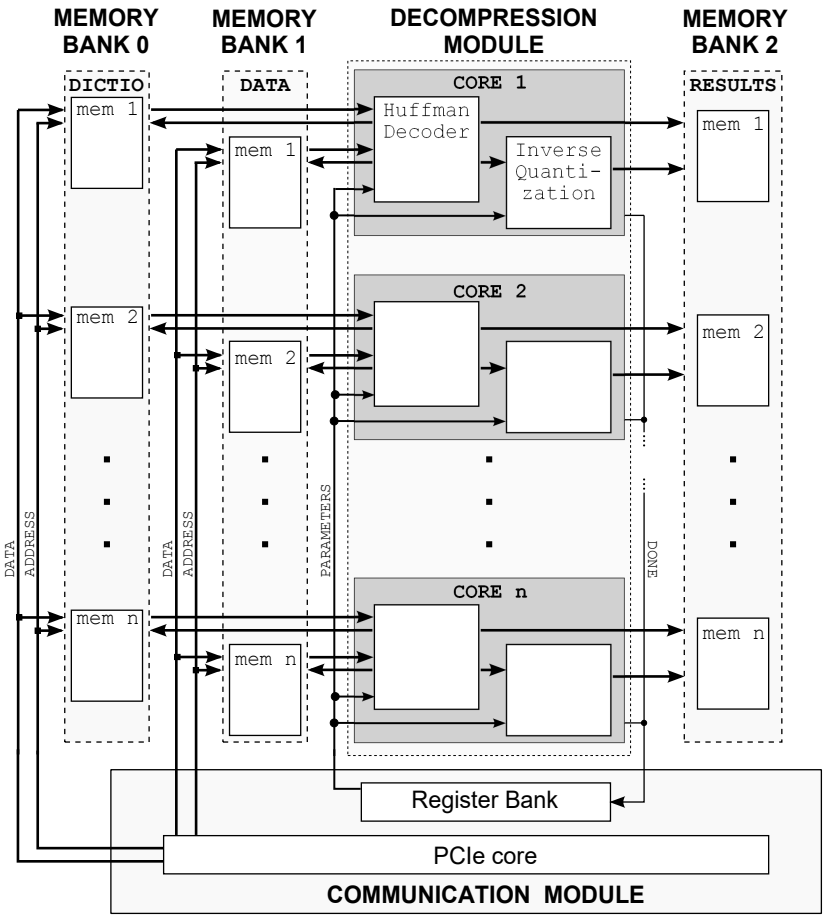


Fig. 3: Parallel architecture for the decompression process

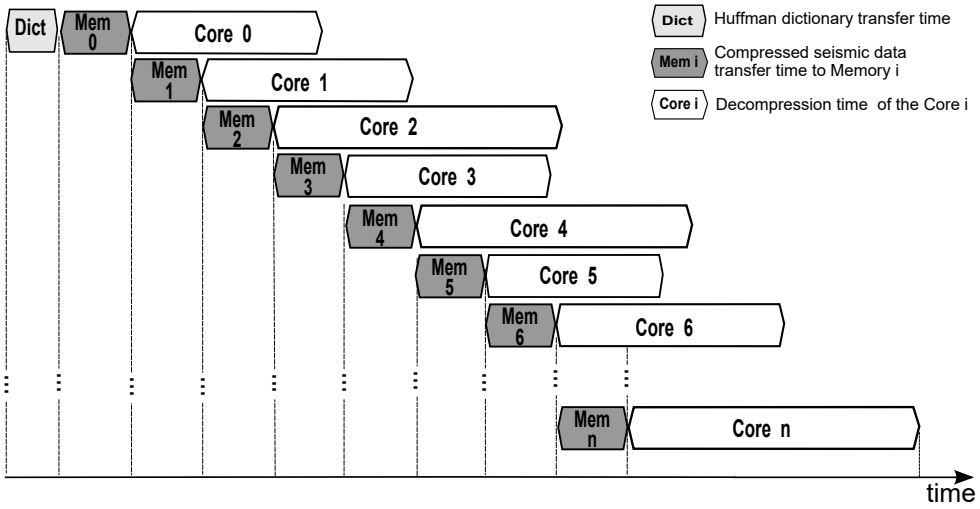


Fig. 4: Time schedule carried out in the parallel architecture

ACKNOWLEDGMENTS

We gratefully acknowledge CPS research group and Colombian Institute of Petroleum (ICP) for their constant support.

REFERENCES

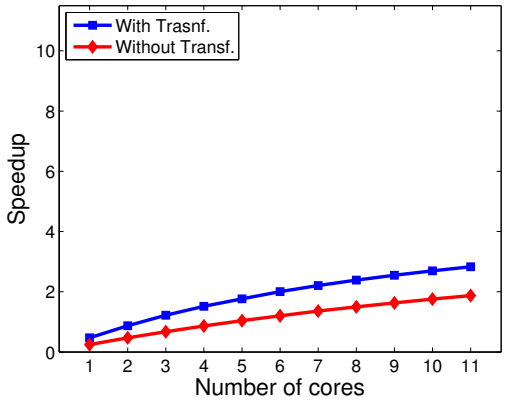
[1] C. Gregg and K. Hazelwood, "Where is the data? Why you cannot debate CPU vs. GPU performance without the answer," *ISPASS 2011 - IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 134–144, 2011.

[2] V. W. Lee, P. Hammarlund, R. Singhal, P. Dubey, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, and S. Chennupaty, "Debunking the 100X GPU vs. CPU myth," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, p. 451, 2010.

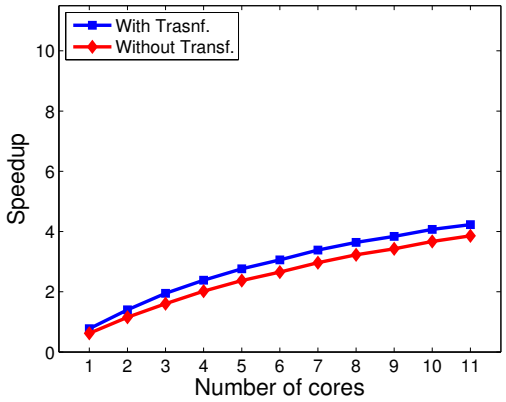
[3] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Towards high performance paged memory for gpus," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 345–357.

[4] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann, 2017.

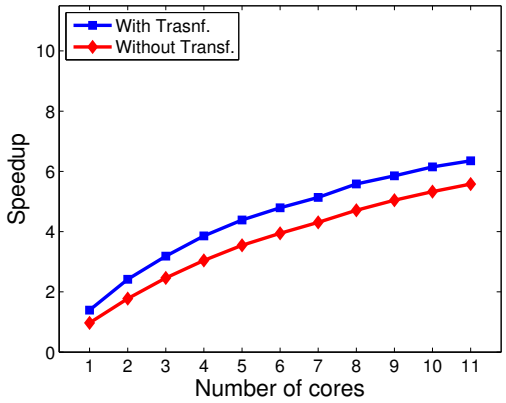
[5] M. M. Keshtegar, H. Falahati, and S. Hessabi, "Cluster-based approach for improving graphics processing unit performance by inter streaming multiprocessors locality," *IET Computers Digital Techniques*, vol. 9,



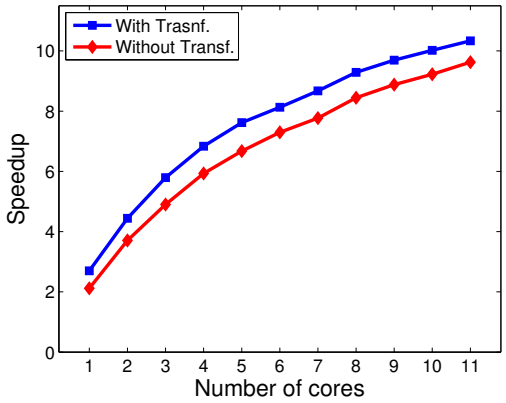
(a) Compression ratio: 5.79 without transformation and 7.51 with transformation (12 bits of quantification).



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



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



(d) Compression ratio: 12.17 without transformation and 16.32 with transformation (9 bits of quantification).

Fig. 5: Speedup results

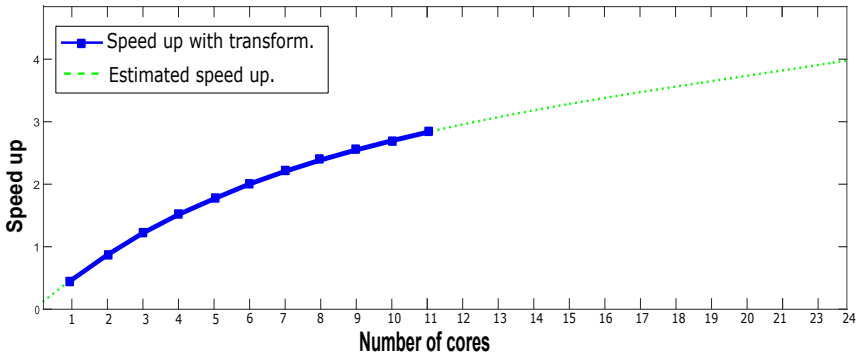


Fig. 6: Estimated speed up for a compression ratio of 7.51 (12 bits of quantification)

no. 5, pp. 275–282, 2015.

[6] T. Ao, P. Chen, Z. He, K. Dai, and X. Zou, "Rdcc: A new metric for processor workload characteristics evaluation," *IAENG International Journal of Computer Science*, vol. 40, no. 4, pp. 274–284, 2013.

[7] J. E. Garrido, E. Arias, D. Cazorla, F. Cuartero, I. Fernández, and C. Gallardo, "A distributed memory implementation of the regional atmospheric model promes," *IAENG International Journal of Computer Science*, vol. 36, no. 4, pp. 344–350, 2009.

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

[9] R. a. Patel, Y. Zhang, J. Mak, A. Davidson, and J. D. Owens, "Parallel lossless data compression on the GPU," in *2012 Innovative Parallel Computing, InPar 2012*, 2012.

[10] C. A. Fajardo, J. C. Villar, and C. Pedraza, "Reducción de los tiempos de cómputo de la Migración Sísmica usando FPGAs y GPGPUs: Un artículo de revisión," *Ingeniería y Ciencia*, vol. 9, no. 17, pp. 261–293, 2013.

[11] C. Angulo, C. A. Fajardo, O. Reyes, and J. Castillo, "Fpga implementation of a Huffman decoder for high speed seismic data decompression Huffman Algorithm," in *Data Compression Conference, 2014*. Salt Lake, United States.: IEEE, 2014, pp. 1–10.

[12] J. A. Castelar, C. A. Angulo, and C. 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.

[13] C. A. Fajardo, O. M. Reyes, and A. Ramirez, "Seismic Data Compression Using 2D Lifting-Wavelet Algorithms," *Ingeniería y Ciencia*, vol. 11, no. 21, pp. 221–238, 2015.

[14] W. Sweldens, "Lifting scheme: a new philosophy in biorthogonal wavelet constructions," in *SPIE's 1995 International Symposium on*

Optical Science, Engineering, and Instrumentation, vol. 2569. International Society for Optics and Photonics, sep 1995, pp. 68–79.

[15] R. G. Gallager, "Variations on a theme by huffman," *Information Theory, IEEE Transactions on*, vol. 24, no. 6, pp. 668–674, 1978.

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

[17] 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*, vol. 6, no. 6, p. 527, 2012.

[18] B. Abali and B. Blaner, "Parallel huffman decoder," Feb. 2 2016, uS Patent 9,252,805.

[19] 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 *XIX Simposio Internacional de Tratamiento de Señales, Imágenes y Visión Artificial STSIVA 2014*, 2014, pp. 1–4.

[20] D. Haugen, "Seismic Data Compression and GPU Memory Latency," Master Thesis, Norwegian University of Science and Technology, 2009.