TOHOKU UNIVERSITY
Graduate School of Information Sciences

Bandwidth Compression Hardware for High-Performance Stream Computing of Real-World Applications

（実応用問題の高性能ストリーム計算のための
帯域圧縮ハードウェアに関する研究）

A dissertation submitted for the degree of Doctor of Philosophy
(Information Sciences)

Department of Computer and Mathematical Sciences

by
Tomohiro UENO

Febraury 22, 2016

Bandwidth Compression Hardware for High-Performance Stream Computing of Real-World Applications

Tomohiro UENO

Abstract

Large-scale numerical computation is one of the most important techniques today. It has been an essential key technology for various fields such as automobile, aircraft, biotechnology, and pharmacy. Numerical computation especially allows the research or development to be efficient by reducing complicated and expensive experiments for them. From requirements of high-precision and large-scale for the numerical computation, there are strong demands of high-performance and large-scale computation.

Since the computational requirements for large-scale numerical computation increased, only supercomputers had met the requirements until recently. In addition to the performance, the power efficiency of computing has got more important than before from the points of view of economic efficiency and effect on the environment. Instead of general-purpose processors, FPGA (field programmable gate array) are often used for high performance and power efficient computing.

FPGA is a device which can be reconfigured with any circuits as we want. By using hardware dedicated to a target application, FPGA achieves both high performance and high power efficiency for computation. In addition, the computing performance of FPGA has grown rapidly in recent years. Therefore, FPGA is now recognized to be a promising device for high performance and power efficient computing.

Many studies employ stream computing to exploit the performance of FPGA, which processes the data stream with a deep computing pipeline by efficient memory read and write. However, the lack of memory bandwidth often prevents the dedicated hardware from achieving the performance because of short supply of data. Since a higher memory bandwidth is required

for high-performance computing, there is a large demand for memory bandwidth enhancement. Since the memory bandwidth is not easy to be improved directly, bandwidth compression on hardware has been proposed. The bandwidth compression reduce the required bandwidth of the data stream, therefore, it improves the data supply to the dedicated hardware as if an available bandwidth increases. Moreover, hardware implementation can operate fast and efficiency.

Despite of these advantages, there are few studies about the hardware-based bandwidth compression for stream computing. Most of them are focused on data compression itself, and the concrete reports on the bandwidth enhancement by the data compression are very rare. Moreover, there is no example of the bandwidth compression for the hardware-based stream computing. Since the purpose of the compression is a performance improvement of stream computing, this dissertation shows the bandwidth compression hardware for a real-world application in order to show that the bandwidth compression is useful.

In chapter 2, it describes the algorithm and hardware design of the data compression and decompression. The compression algorithm for numerical data streams employs prediction-based method which exploits continuity of the data. This chapter also shows designs and an implementation of the data compressor and decompressor which realize the algorithm on FPGA. The data compressor and decompressor achieve good compression performance for the numerical data and they can achieve high-throughput processing. On the other hand, the area is still a problem to apply to real-world numerical applications.

Chapter 3 shows solutions for the large area of data compression hardware. This chapter presents a smaller hardware than the previous design which requires large hardware resource so as to convert a variable-length compressed datum into a fixed-length output data block. Although the amount of hardware resources is usually inversely proportional to the compression performance, the improved one achieves a large area reduction with a little decline of the compression performance thanks to the adoption of a novel encoding. The proposed encoding exploits uneven distribution of a predictive accuracy in the data compression, which is specific to each data. Moreover, the proposed method can deal with various types of data by selecting parameters at a design-time.

Chapter 4 describes the multiple-channel compressor in order to apply to real-world applica-

tions. The proposed hardware achieves the bandwidth compression for multiple channels with careful attention to the requirements from the numerical computation on FPGA. The improved hardware allows us to implement several dozen compressors and decompressor at the same time on a FPGA to handle multiple channels. The bandwidth compression achieves to enhance the memory bandwidth up to 2.5 times of a physical bandwidth. The demonstration shows the performance improvement of the computing by the enhancing the I/O bandwidth of the dedicated hardware.

The contributions of the thesis are a proposition of the hardware-based bandwidth compressor which improves the FPGA-based stream computing, and a presentation of the performance improvement of the real-world application by the bandwidth compression.The bandwidth compressor is high-throughput and small to apply to various applications on hardware. In addition, there is no need to improve physically for the bandwidth enhancement. Therefore, the bandwidth compressor is very useful for various cases of hardware-based computing. To achieve an effective bandwidth compression, we design the hardware-based data compressor and decompressor. The demonstration of the proposed method shows that the bandwidth compressor improves the computation performance even if the available bandwidth is insufficient.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Backgrounds

High-performance computing is a very important technology in today's world. It deals with huge and complicated calculations such as numerical simulation which often employs large-scale and high-performance computers. For example, machine designs sometimes needs large-scale numerical simulation based on fluid dynamics and material mechanics. It has be an essential technology in other various fields, such as designing of buildings, developments of new medicines, and analyses of natural phenomenon. Large-scale simulations require large amounts of operations and data transmissions to obtain productive results [1, 2, 3, 4]. Since they require long computation time and large resource utilization, there is a strong demand for large-scale and high-speed computation, especially for numerical simulations, such as computational fluid dynamics (CFD) or computational electromagnetics (CEM)[2, 5, 6, 7].

Supercomputers are generally used for large-scale computing. Recent supercomputers consist of a large number of computing nodes connected by their interconnection network. For example, K-computer developed by RIKEN consists of more than 80,000 computing cores [8] and Tofu interconnect which is a six-dimensional torus network [9]. K-computer achieved a performance of 8,162 TFLOPs in LINPACK benchmarks thanks to the large number of processors and the efficient network architecture [10]. However, such many-node architectures require huge electricity for large-scale computation. An energy consumption of K-computer was 9.899 MW when the LINPACK benchmarks were measured [10]. If the power efficiency is not going to be improved for successors of K-computer, the power consumption becomes more than 1000 MW for an exascale computing which is expected to be available around 2020 at latest [11, 12]. Therefore, it is strongly required to reduce the power consumption of the high-performance computing.

On the other hand, high-performance computing often suffers from insufficient memory bandwidth in general-purpose computers based on von Neumann architecture [13, 14, 15, 16]. The arithmetic performance of a general-purpose processor cannot fully be exploited when memory

reads and writes are required for many data per operation. [17]. Here the number of operations per unit data read from an external memory is referred to as an operational intensity. It has been reported that insufficient memory bandwidth often limits the sustained performance of many computing applications to a fraction, which is sometimes a few percent, of the peak performance due to their low operational intensity. To alleviate the memory bottleneck, recent general-purpose processors usually rely on a hierarchical memory system, which are composed of multi-level cache memories. The memory system holds and reuses data read from an external memory for reduction of data transfer between the processor and the memory. However, the cache memory does not always work effectively for all kinds of applications because the cache memories are designed for the average features of applications, and do not satisfy the requirements of all the applications completely. Additionally, parallel computers, which is now the mainstream of high-performance computing, become inefficient for particular applications because an internode communication limits the entire performance. [18, 19]. Thus, it is getting more and more difficult to exploit computing resources for performance especially in a large-scale system due to the inefficiency caused by the limited memory bandwidth for a processor and the communication overhead among nodes. Please note that the inefficiency arises out of the general-purpose feature of the computers. For these reasons, general-purpose processors are often hard to exploit their peak performance for large-scale computation because of their too much flexibility.

There are alternative devices instead of general-purpose processors for large-scale numerical computing. Because of the demand to reduce the power consumption and to enhance memory bandwidth, power efficient devices with sufficient memory bandwidths are ideal for high performance computing. For the numerical computing, graphics processing units (GPUs) have been used in many studies and for practical applications. Although GPUs are originally designed for image processing, they can achieve high performance for floating-point operations. Field programmable gate array (FPGA) is also a power efficient device, and there are several studies which propose to use FPGAs for high-performance floating-point computation [20, 21, 22]. FPGA is a reconfigurable device on which we can implement various digital circuits, therefore, it allows us to use an optimal circuit in accordance with a target computation. Both GPUs and FPGAs are expected to achieve higher power efficiency than general-purpose processors.

Figs. 1.1 and 1.2 compare performances and power efficiencies among computational devices of CPUs, GPUs and FPGAs [23, 24]. Fig. 1.1 shows processing performances and power efficiencies of CPU, GPU, and FPGA for a random number generation, and Fig. 1.2 shows power efficiencies of information filtering implemented in OpenCL. The results show that FPGAs achieve the highest performance and efficiency in both cases. Compared with GPUs, FPGAs are much more power efficient, while flexibility is also given for various computations by their

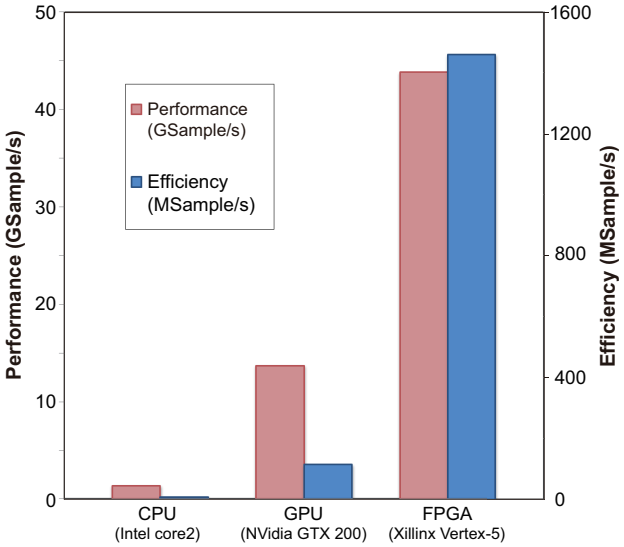


Figure 1.1: Performance and power efficiency of CPU, GPU, and FPGA for generating random numbers.

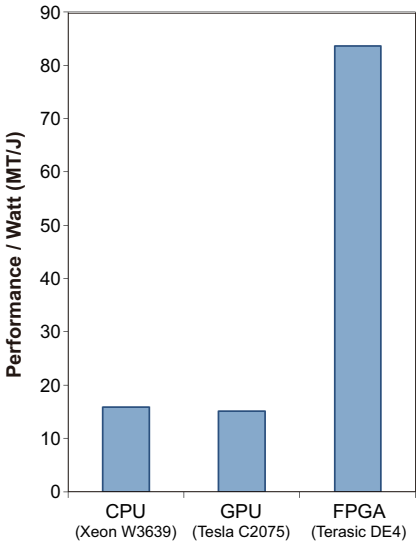


Figure 1.2: Power efficiency of CPU, GPU, and FPGA by OpenCL implementation.

reconfigurability. Moreover, FPGAs are recently getting higher performance even for numerical computation, so that many studies are conducted for high-performance computing with FPGAs. Thus FPGAs have both characteristics of high power efficiency and flexibility, which can be used for high-performance but low-power computation of various applications.

Until 1990s, FPGAs had been originally and mainly used for a prototype implementation or emulation of ASICs (Application specific Integrated Circuits), which are proprietary semiconductor chips fabricated for specific applications. Dedicate circuits implemented on ASICs are very fast, small, and low power achieving high performance processing by an effective use of hardware resources. However, their enormous initial cost, which is required for a photomask, is an obstacle in making ASICs. To justify the fabrication of ASICs, they have a large demand for mass production of the same chips from the economic viewpoint. In contrast to ASICs, FPGAs do not have such high initial cost because it is possible to implement circuits on them without any photomasks. Although FPGAs have higher cost per chip than ASICs in the case of mass production, FPGAs have cost advantages for high-mix low-volume production with various computing applications. In other word, the state-of-the-art semiconductor devices with a less than 20 nm process rule are too expensive to fabricate. Furthermore, a long development term makes ASICs weaker than FPGAs. Therefore, if extremely high performance and low power processing is not necessary, FPGAs are feasible solutions better than general-purpose processors including CPUs, DSPs, and GPUs.

Moreover, the potential performance of FPGAs has been improved drastically and the gap

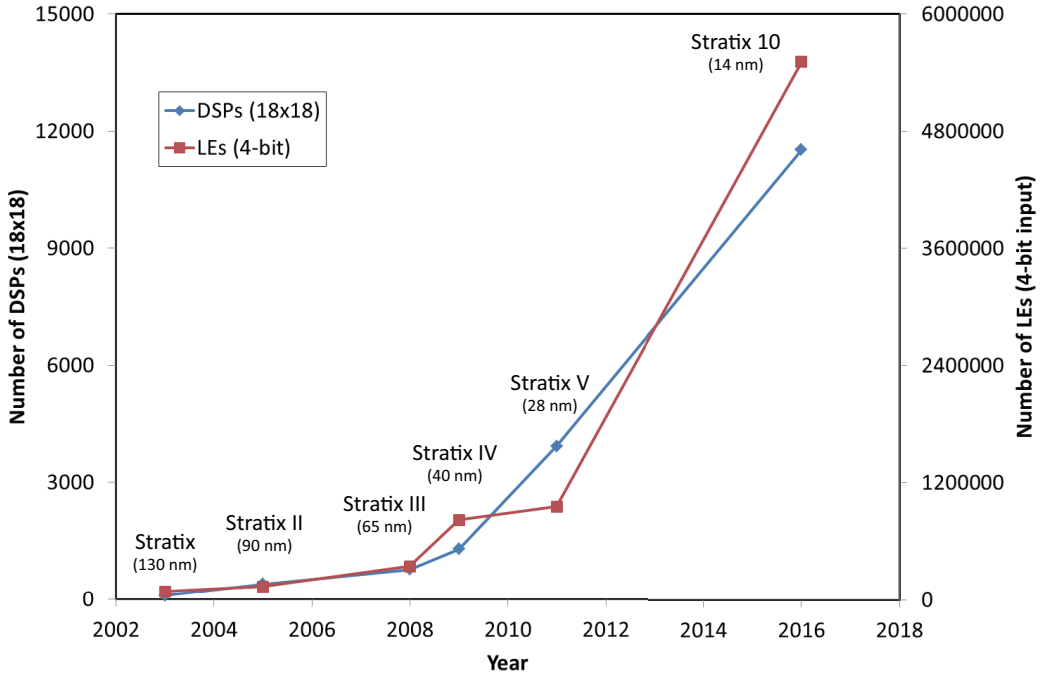


Figure 1.3: Number of DSP blocks and LEs in high-end FPGAs.

between FPGAs and ASIC are being reduced. Fig. 1.3 shows the technological advancement of FPGAs. Stratix FPGA series are high-end FPGAs of Altera corporation which is one of major FPGA vendors. DSP (digital signal processor) is a dedicated circuit which is generally embedded in FPGA as high-speed integer multipliers while DSP blocks of the recent Stratix10 FPGAs have floating-point operation modes. LE (logic element) is a logic resource unit to implement combinational circuits and sequential circuits. The increase of hardware resources on FPGAs allows us to implement larger and more functional hardware than ever before. Therefore, recent FPGAs are much more economical devices for dedicated processing than ASICs, and they are also capable of low-power and high-performance computing.

To exploit the performance of FPGAs for high-performance computing, a stream computing is one of the promising methods. With a sequence of data elements, referred to as a data stream, stream computing continuously processes a large amount of data in the stream. In many real-world applications such as CFD, high-performance computing requires to apply a certain process to the entire data. The operations applied to each element of a data stream are referred to as a processing kernel, or just a kernel. By pipelining a large kernel to increase a throughput of stream computing, high-performance computation can be achieved with a data stream. In FPGA-based stream computing, circuits of a deeply-pipelined kernel are usually implemented on an FPGA with external memories. This study also implements direct memory access (DMA) hardware modules to feed a data stream to the kernel pipeline by successively reading data from

the memories. The data read from the memory is input into the kernel circuit, then the pipeline processing is applied to them. Then the processing results are output as a data stream to be written to the external memory. Customized kernel circuits allow the stream computing with FPGAs to achieve power-efficient processing even at a high throughput. For a large FPGA, we can also rely on coarse-grain parallelism to further increase performance as far as resources on an FPGA allow. Moreover, stream computing is good at using memories efficiently. Its successive data access allows us to exploit available bandwidth of DRAMs which are usually used as external memories.

With a large amount of data to process, the performance of stream computing is determined almost exclusively by throughput because it is a pipeline process. When the number of the pipeline stages is smaller enough than the number of data elements to be processed, the throughput of pipeline is the most important factor for performance. The throughput of stream computing is also determined by some factors such as operation frequency, I/O bit width of a circuit, and memory bandwidth. Since high throughput is required to achieve higher performance in FPGA-based stream computing, these factors are keys to the performance improvement. Even deeply pipelined custom circuits implemented on an FPGA, their peak performance can not be fully exploited if the memory does not satisfy the bandwidth requirement of the pipeline.

Since a lot of computational elements can be implemented with today's large FPGAs, it is possible to apply higher parallelism with the elements to improve the overall performance. However, with multiple cores on single FPGA, the parallel processing often requires wide memory bandwidth. When the circuit on a FPGA requires greater bandwidth than an available memory bandwidth, the computational performance is limited by the memory bandwidth because of an insufficiency of a data supply from the memory. Along with the performance improvement of FPGAs, the required bandwidth of the dedicated circuit has also been increasing, which enhances an importance of the memory bandwidth in the entire performance [25]. Therefore, it is impossible to improve the computational performance without sufficient memory bandwidth even if we employ a high-performance FPGA with an appropriate circuit.

1.2 Hardware-based bandwidth compression

There are several solutions to enhance the memory bandwidth of the FPGA-based stream computing. The bandwidth is determined by two factors, an I/O bit width of both FPGA and memory, and a data transmission rate of an I/O part. In general, it is difficult to directly improve these factors because we need to improve physical layers of data transfer. Instead of trying to improve the physical layers to directly increase the bandwidth, data compression which can improve the bandwidth without any physical improvement is expected as an promising method. Data compression is a technique to reduce an amount of digital data by exploiting redundancy in data. Therefore, it is possible to enhance the memory bandwidth of FPGA-based stream computing by compressing a transmission data stream between FPGA and a memory. The compression of transmitted data reduces the memory bandwidth required for a computing kernel on an FPGA. On the other hand, the data compression also requires some computational cost as processing time and computational resources.

From the point of view of the entire performance, especially the processing time of the data compression must be short enough not to extend the entire computational time. To satisfy this requirement, some previous studies employed hardware-based data compression which processes much faster than software processing by general-purpose processors. Therefore, this study uses data compression hardware for preventing a performance decrement due to the insufficient bandwidth. In this study, the bandwidth reduction by applying data compression technique as bandwidth compression is referred. The hardware-based bandwidth compression reduces the required bandwidth by a circuit designed specifically which processes within a very short period of time.

The FPGA-based stream computing is very suitable for being applied the bandwidth compression hardware to demonstrate its effectiveness. Hardware implementation is suitable for data compression between a computing kernel on an FPGA and a memory. If we use a processor for software compression out of the FPGA, it is impossible to compress the bandwidth between the processor and the FPGA. Accordingly, it is feasible and most promising for us to implement hardware for bandwidth compression between a computing kernel and a memory interface on an FPGA. For applying the bandwidth compression to the memory bandwidth of this computing, therefore, it is necessary to implement the compression hardware with a dedicated circuit which performs specific computing on an FPGA.

Although bandwidth compression employs data compression, the purpose of the bandwidth compression is not only to reduce an amount of data but also to reduce of required bandwidth. In the FPGA-based stream computing, the bandwidth compression reduces the required bandwidth of a transmitted data stream. The target of the bandwidth compression is a data stream of

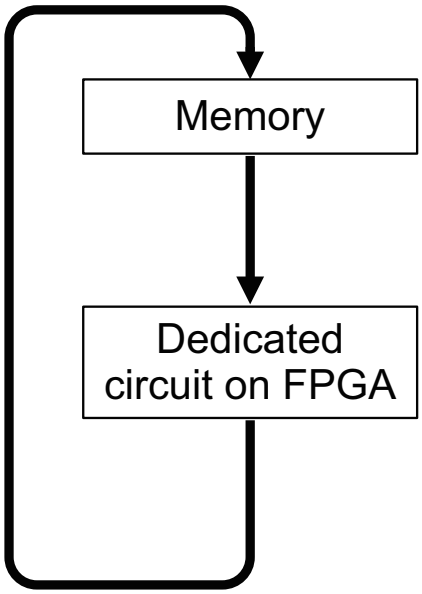


Figure 1.4: Stream computing.

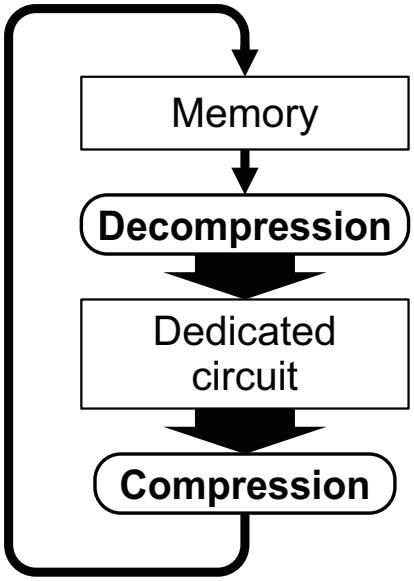


Figure 1.5: Stream computing with bandwidth compression.

numerical computation in this study because the stream contains a large amount of data and several variables, which often cause an increase in the required bandwidth. For applying to the FPGA-based numerical computation, the bandwidth compression has several requirements as follows:

1. Suitable compression performance.
2. Lossless data compression.
3. Processing for multiple channels.
4. High throughput compression.
5. Small hardware.

The first, second and third requirements are due to the numerical computation, and the fourth and fifth are due to an operation of the hardware-based implementation. These requirements are for applying to real-world numerical simulations which are used to solve actual problems. This study describes such numerical applications as real-world applications, and sets them as targets of the bandwidth compression.

Fig. 1.4 show an overview of the FPGA-based stream computing. This computing requires a concurrent execution of the data transmission and the data processing. Since dedicated circuits are usually pipelined, the stream computing requires sustained input and output of data streams. The data stream alternates between the memory and the FPGA many times because the real-world application usually requires an iterative computation.

To apply the bandwidth compression to Fig. 1.4, the data compression and decompression hardware should be implemented as shown in Fig. 1.5. The bandwidth compressor has no choice

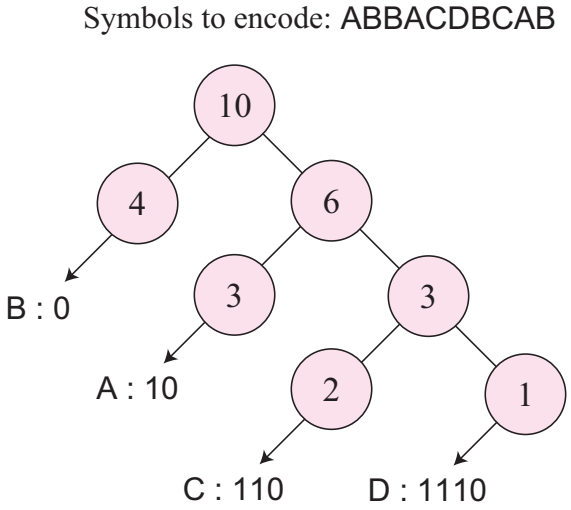
but to implement on the FPGA. In addition, since the numerical circuit needs uncompressed data to obtain exact results, the data decompression hardware is also needed. In Fig. 1.5, an output stream from the dedicated circuit compress by the bandwidth compressor, and it is transferred to the external memory. The memory stores the compressed data stream temporarily, then it transfers the stream to the FPGA again. The transferred data stream is input to the decompressor and reconstruct as uncompressed data, then the stream is input to the dedicated circuit. The computing with the bandwidth compression repeats these series of operations until the end of the computation.

Since the system shown in Fig. 1.5 reduces the required bandwidth, the effect of the bandwidth compression depends on the performance of the data compression ratio strongly. The performance of the data compression is represented by a compression ratio, R_{comp}, which is defined as

$$R_{comp} = \frac{S_{orig}}{S_{comp}}, \tag{1.1}$$

where S_{orig} and S_{comp} are the data sizes of original and compressed, respectively. The effect of the bandwidth compression can be roughly estimated from the average of the compression ratio of an entire data.

However, factors to determine the performance of the bandwidth compression are not only the compression ratio. A hardware design, a property of data, and an implementation environment such as performances of FPGAs and memories are also considered to affect the performance of the bandwidth compression. Especially, since the bandwidth compression of multiple channels has not been implemented and evaluated for the real-world application so far, concrete evaluations have been required as actual cases of the bandwidth compression to find out characteristics of this method. This study presents the multiple channels bandwidth compression for real-world applications as an objective to be realized. In addition, this study also presents the area reduction design of the bandwidth compressor which is essential for applying to the multiple channels. Moreover, it shows a detailed design of the data compressor and decompressor used for the bandwidth compression, the algorithm of which has been proposed in previous works [26, 27].



Symbol	Frequency	code
B	4	0
A	3	10
C	2	110
D	1	1110

Figure 1.6: Huffman tree and Huffman coding.

1.3 Related work

The concept of the bandwidth compression is very important in data communication because an amount of transmitted data increases rapidly in recent years. This trend is not only for communication via the Internet but also for internode communications of large-scale parallel computation and memory accesses in data intensive processing [28]. As mentioned in the previous sections, the bandwidth compression of data transfer to or from a memory is required to be much faster and higher-throughput processing than the long-distance communication such as the Internet. This requirement is the same for FPGA-based stream computing in the data transmission between external memories and FPGAs. Therefore, the bandwidth compression needs fast and high-throughput operations with an enough compression performance to improve the performance.

In such a challenge, it is necessary to find out what is needed to realize by referring related studies. Following subsections describe related studies of the bandwidth compression by a hardware implementation for numerical data. This section describes those studies dividing into three fields, data compression algorithms for numerical data, data compressions on hardware, and encoding methods of multiple channels. In addition, it describes previous studies of our research group to clarify objects and tasks to achieve the bandwidth compression for real-world applications.

1.3.1 Data compression algorithms for numerical data

Firstly, this subsection describes about the data compression algorithm. This study requires the compression to be lossless, suitable for hardware implementation, and suitable for numerical

data streams.

Lossless data compressions are used for some kinds of data which require that decompressed data are identical to original data [29, 30, 31]. For example, programs, text documents, and numerical data with high accuracy correspond to this. Commonly used lossless data compression formats are ZIP, gzip, LZH, which are based on LZ77 compression algorithm [32]. Lossless compression generally exploits redundancy in data as a sequence of symbols. It is based on information theory propounded by Claude Shannon [33]. In the information theory, concepts of entropy and redundancy in the information are defined, which can be used to reduce the amount of data without any loss of necessary information. Shannon-Fano coding [34] is a representative method of entropy codings, and Huffman coding [35] which employs an optimal prefix code is commonly used for lossless compression. Fig. 1.6 shows an example of Huffman tree which is used in Haffman coding when it encodes data contain four symbols into binary codes. In Haffman coding, symbols with higher occurrence frequencies are encoded into shorter bits to increase the compression performance. Symbols are sorted in order of appearance frequencies, and allocated from the shorter code in the order.

Considering the hardware implementation, there are some choices of lossless data compression. Typical techniques of the lossless data compression other than entropy codings are dictionary methods, data transform, and prediction based methods. The entropy coding, such as Huffman coding and arithmetic coding [36, 37], needs to regard individual data as symbols. Therefore, entropy coding does not suit for data compression with too many symbols and without meaningful probability deviation [38]. The dictionary method handles a string of symbols to be applied to the entropy coding, which allows us to use variable-size codes by holding input strings of symbols in a dictionary. The data transform method is intended to compress data of two or three dimensional space. Most of other lossless compression algorithms are combination of these techniques.

The efficient and effective lossless compression for large amount of data are required especially in the fields of big data processing and deep learning. Maruyama et al. proposed lightweight lossless data compression algorithms based on dictionary methods [39, 40]. Their methods are general-purpose and used in online, which are able to apply to wide range such as text data or scientific data. In addition, some of them have been implemented on FPGA [41], which are applied to various data, such as English text, MIDI, and DNA gene. These studies shows a high versatility in data compression, however, they do not suit for the bandwidth compression of stream computing because they require too big hardware for implementation. Furthermore, the methods are not suitable for compression of floating-point data in terms of compression ratio.

For compression of floating-point data in numerical computing, these techniques does not suit to apply directly because it is difficult to obtain frequency of appearance in symbols or their

sequences directly from such numerical data. Therefore, it is necessary to employ a different method to compress floating-point data. Numerical floating-point data usually have numerical continuity in a discretely represented data stream. The continuity appears in spatial and temporal spreading quantities. The spatial continuity enable us to predict a particular datum by a certain equation with adjacent data with some precision. Therefore, it is possible to compress the numerical data effectively by subtracting between the predicted and original value and encoding the difference adequately.

Several studies have proposed compressions using hash tables to exploit fluctuation patterns of data [42, 43], which use the dictionary method. Ratanaworabhan et al. proposed a data compression for double precision floating-point numbers based on context-based prediction [42]. And Sukhwani et al. proposed an adaptive method which selects an optimal dictionary dynamically [43]. Some studies employed predictors based on the numerical computing. Lindstrom et al. and Ibarria et al. employed Lorenzo predictor which predicts the value of a computational grid point in arbitrary dimension [44, 45]. Fout et al. proposed the adaptive predictor [38]. This compression selects the optimal predictor from among several options. Prediction-based data compressions always calculates the difference or the XOR between the predicted and the original data to reduce the data amount. The differences are encoded into shorter bit strings than the original data. These studies achieve high compression ratios for numerical data on a three dimensional or two dimensional computational grid. Although these algorithms achieve effective data compressions, their implementations are mainly for software processing which does not provide a sufficient throughput required for FPGA-based stream computing. This study employs an effective algorithm suitable for hardware-based bandwidth compression by extending these algorithms. The detail of the algorithm is described in the next chapter.

1.3.2 Data compression on hardware

Secondly, this subsection describes the hardware-based data compression, which is necessary for high-throughput compression in stream computing. The hardware-based data compression has been researched in for various types of data, including audio [46], and video data [47]. The hardware compression techniques for audio and video data are already established and are commonly used. As seen in these studies, these data compressions have achieved for lossy data compression. There are also propositions for arbitrary data in a memory [48], and in streaming transfer [49, 43]. These techniques cannot directly compress floating-point data at a high compression ratio because they don't exploit characteristics in numerical data.

For the compression of numerical data, there are only a few studies of hardware-based data compression. Hardware-based decompression of double-precision floating-point was proposed in [50]. This study employs some commonly-used algorithms and an original method which is

based on the dictionary method. They also showed a hardware design of the decompressor. Sakuwani et al. presented data compression hardware based on context-based prediction using a hash table described above [43]. These studies show hardware designs for a small area with a potentially high throughput. However, the compression ratio is relatively low. Moreover, there was no complete design capable of bandwidth compression for real-world applications.

As other examples, Angulo et al.[51] presents a seismic data decompression on FPGA, which requires high speed processing, and shows Keymeulen et al. lossless hyperspectral data compression on FPGA to transmit through limited bandwidth [52]. These studies have reasons to use FPGA, the former needs fast operations to minimize the damage, and the latter processing are under unusual environments. Although these implementations are reasonable, they do not suit for numerical data and stream computing. As another hardware based compression, Maxeler Technologies provides by incorporating data compression system for high-level synthesis of FPGA applications [53]. This data flow computing platform allows us to use lossless and lossy data compression into application data flows. They also proposes hardware compressor and decompressor of run length encoding for the fast processing [54]. These proposals take advantages of the FPGA, however, these are not able to enhance memory bandwidth effectively.

Our study requires both data compressor and decompressor for the stream computing, which achieve both high compression ratio and high throughput.

1.3.3 Encoding multiple channels

Thirdly, this subsection describes related work on the serialization of multiple channels. In particular, realistic applications use multiple variables in their computation, the bandwidth compressor needs to compress multiple channels on hardware-based stream computing. Time-division multiplexing (TDM) is commonly used to encode multiple channels into a single transmission channel.

Some studies, such as multicore processing on FPGA, employed TDM to bundle multiple channels [21]. On the other hand, to the best of our knowledge, there have been no study on hardware-based encoding multiple channels with lossless data compressions. TDM allocates an output bandwidth just equally to all channels. Therefore, the simple TDM cannot handle multiple channels which require different bandwidths when the compression ratio fluctuates among the channels. In the prediction-based data compression, each channel has different compression ratio. This means that the serialization which we employ is required to distribute an available memory bandwidth to the channels according to their compression ratios. Therefore, this study propose an original method to handle the multiple compressed channels so as to satisfy the requirements of real-world applications.

1.3.4 Previous work

A previous work already have been made, which is about data compression algorithm for the numerical data stream. Katahira et al. proposed a data compression algorithm for floating-point data stream, which achieved a comfortable compression performance for data of numerical simulation [26]. The algorithm is simple enough to achieve high throughput by hardware implementation. It is also possible to be implemented by a smaller hardware than other related work above. Sano et al. proposed a high-throughput predictor for the prediction-based data compression [27]. A prediction accuracy of the predictor is sufficient, and it is realized by very small and fast hardware. In spite of these achievements, their works did not design a complete data compressor and decompressor.

The dissertation presents an entire design of the FPGA-based bandwidth compressor. Although the data compression algorithm is based on techniques that have been proposed so far, this achieves a novel real-time bandwidth compression for multiple channels processing. It presents complete design of the bandwidth compressor which can be applied to real-world numerical applications on FPGA. It also shows a demonstration of the bandwidth compressor with real-world application to improve the throughput and computational performance.

1.4 Objectives

The objective of this dissertation is to present the bandwidth compression can be applied to stream computation for real-world applications in order to improve its performance by enhancing an effective bandwidth without any change of the physical layers. The bandwidth compression provides us a trade-off between bandwidth improvement and hardware resource consumption. To use this technique efficiently, the cost of the compression, that is especially a delay and an area overhead, need to be as small as possible considering the finite hardware resources available on FPGA. The target of the bandwidth compression in this study is large-scale numerical simulations, referred to as real-world applications in the text, which suffer from inefficiency due to the limited memory bandwidth in software-based computation. We assume that this kind of computation should be implemented with FPGA-based custom hardware for stream computation to exploit its potential of low-power but high-performance computation with state-of-the-art FPGAs. The proposed technique for bandwidth compression allows FPGA-based stream computation to achieve higher performance with improvement of an effective memory bandwidth. There are requirements and problems to achieve the bandwidth compression. Following chapters present their implementations and solutions, and finally demonstrates numerical computing with the bandwidth compression for several benchmark computations.

Firstly, this thesis presents the data compression hardware for the bandwidth compression of real-world applications. The data compression algorithm for the numerical data already has been proposed in the previous work. This study realizes the data compressor and decompressor on FPGA, and evaluate their performance. Based on the evaluation, it shows discussions whether the implemented hardware is suitable for the bandwidth compression. As a result, the data compression hardware has too large area to apply to the real-world applications, whose large area prevent us from implementing with dedicated circuits on a recent FPGA. Moreover, the hardware is just not able to deal with multiple channels.

The large area of the hardware must be solved to apply to real world applications. A cause of this problem is an operation to convert from variable-length compressed data to fixed-length output data blocks. This conversion is essential for exploiting an available bandwidth efficiently. And the lossless data compression forces that each compressed datum has a variable length of bits. To solve the problem, this thesis introduce a novel quantized encoding which reduce the area of the hardware dramatically. The solution also exploits deviations of distribution of prediction accuracies to prevent a decrement of the compression performance. Since the solution provides a selectable design of the compressor, we are able to select its design depending on a characteristic of data. As a result, this study presents an improved compressor which has a very small area and a sufficient compression performance. The small compressor makes it possible to

apply the bandwidth compression to the real-world applications.

The dedicated circuit on FPGA, which operates the real-world application, has multiple channels corresponding to the variables in a simulation. There is a requirement of a multi-channel bandwidth compressor which satisfy requirements of the numerical computing. This thesis shows designs of a multi-channel serializer (MCS) and multi-channel deserizlizer (MCD) to control the multiple channels. Then it is possible to assemble the bandwidth from the compressor, decompressor, MCS, and MCD. For a demonstration of the bandwidth compressor, this study applies it to a benchmark computing and evaluate the bandwidth and the computing performance. The result shows that the bandwidth compression enhances the available bandwidth, and improves the computing performance.

This paper is organized as follows. Chapter 2 shows data compression algorithm for numerical data stream and design of the compressor and decompressor. The contributions of the chapter are the complete designs of these hardware and finding out that the compressor cannot be applied to real-world applications because of its hardware area. Chapter 3 presents area efficient data compressor for a numerical data stream. The contributions of the chapter are introductions of area-oriented designs of the data compressor and decompressor which enable us to select suitable design depending on target data and available hardware area. Chapter 4 presents a multiple channels bandwidth compressor for real-world applications. The contributions of the chapter are the designs of the bandwidth compressor and an entire system, and the demonstration of the bandwidth compression which improves the computing performance of the real-world application. Finally, chapter 5 gives conclusions and future work.

Chapter 2

Algorithm and hardware design of data compression

2.1 Introduction

Data compression exploits the redundancy of data. In digital processing, data are represented by fixed-length bit strings. The basic idea of lossless data compression is that the number of bits is reduced to a minimum capable of holding the original information. Entropy coding is a typical example thereof, which compresses data effectively in accordance with occurrence frequencies. The lengths of lossless compressed data are generally variable as shown in Fig. 1.6, and the bit lengths of compressed data can be longer than that of the original data when the occurrence frequency of the symbol is very low. In data compressions of floating-point data, the entropy coding does not work well because it is difficult to find the deviation of the symbol occurrence when we regard each value as a symbol. The redundancy of numerical floating-point data is different from such as integer or text data which we can compress a datum itself as a symbol. The compression of numerical floating-point data requires a method based on a numerical continuity rather than a deviation of the occurrence frequencies.

The numerical data often similar to image data because pixel data also have a certain level of continuity. Although typical image compressions are usually lossy, there are some lossless compression techniques which take advantage of the continuity [55]. Since the image compression does not require strict conditions generally, they often employ high-compressive and complex algorithms such as JPEG which consists of discrete cosine transform, quantization, and huffman coding [56].On the other hand, the bandwidth compression for stream computing should not employ such complex methods because of severe constraints in processing time. Moreover, an algorithm for the bandwidth compression should also be so simple that it can be implemented as small hardware.

Hardware-based stream computing on hardware employs a sequential memory access to

exploit the peak bandwidth by simplifying operations. Since numerical computation needs adjacent data on a computational grid, the data placement in the memory is based on the arrangement of the computational grid. Our previous studies proposed the prediction-based data compression for numerical data streams [26, 27]. They employed a simple polynomial for the prediction in consideration of an implementation of a high-throughput and small hardware. Since the prediction-based algorithm also achieved a good compression performance for practical numerical data, this study also employ this algorithm.

We design the entire hardware of the data compressor and decompressor and evaluate them for applying to the real-world applications. The compressor and decompressor are pipelined to be high throughput, and the predictor can be implemented with very small area. In the evaluation, the hardware achieves higher compression ratio than a general-purpose data compression format, bzip2, for actual numerical data. The evaluation also shows that the hardware can operate about 200MHz for single-precision floating-point data, which is enough to apply real-world applications.

On the other hands, the areas of the compressor and decompressor are not enough small to applying multiple channels computing. Since the prediction-based algorithm requires data with continuities, data compression for multiple channels needs many hardware compressors and decompressors. In addition, the bandwidth compressor is needs to be implemented with dedicated numerical circuits. Therefore, the area of these modules should small.

This chapter presents data compression algorithm for numerical data streams in the stream computing. We propose a prediction-based data compression algorithm which exploits numerical continuity of floating point data. We show that the compression ratios are more than three in this algorithm with a compression of real numerical data streams. We also show the hardware designs of the data compressor and decompressor for implementing the proposed algorithm. Evaluations show the data compression hardware is capable to operate at high operating frequency, therefore, throughput of the hardware is also high. In addition, we develop a parameterized data compressor and decompressor and evaluate them. This evaluation shows that the compressors and decompressors applied to multiple channels cause huge area overhead for real-world numerical applications.

The objective of this chapter is to present entire hardware for the data compressor and decmpressor and evaluate the implementation of the hardware. The implementation of these modules should be high throughput and small hardware. For this goal, we employ the prediction-based algorithm which has been proposed in our previous work and design the compressor and decompressor. We also evaluate the algorithm and the hardware implementation of these modules. The evaluation shows that the algorithm achieves sufficient compression performances for numerical data and the implementation achieves high throughput. However, for applying it to the multiple channels, the area must be a problem in our estimation. The contributions of

this chapter are:

1. Entire designs of the compressor and decompressor.
2. Good compression performance of the algorithm.
3. High throughput of the hardware implementation.
4. Large area of the implementation.

The organization of the chapter is as follows. Section 2.2 describes the data compression algorithm. Section 2.3 shows a principle of the prediction. Section 2.4 presents the entire design of the data compressor and decompressor. Section 2.5 shows a parameterized designs. In section 2.6, we evaluate the compression ratio, throughput and hardware area. Then section 2.7 presents the problem of the hardware implementation. Finally section 2.8 gives conclusions of this chapter.

2.2 Data compression algorithm for numerical data streams

This section describes the data compression algorithm in our previous work [26], which is suitable for an FPGA implementation and applying real-world applications. In FPGA-based stream computing, several studies showed that the achievable performance is limited by I/O bandwidth instead of arithmetic performance of dedicated circuits, even if regularity of stream computation exploits the physical bandwidth efficiently [57, 58]. Since it is not easy to drastically increase the I/O bandwidth, future chips will be perpetually suffering from the insufficiency of bandwidth for on-chip computing with remarkable increase of hardware resources on FPGA. To solve the problem, we have proposed bandwidth enhancement by employing lossless data compression for numerical floating-point data in our preceding research [27, 26]. The bandwidth compression excludes of bits in data streams, which enables us to use the available bandwidth efficiently at *compressed bandwidth*.

The lossless data compression guarantees complete reconstruction of original data, thus causing no computational error unlike lossy compression. If data compression reduces the size of data to $1/r$ of the original data on average by placing the compressor and decompressor to the input and output of the dedicated circuit, the achievable bandwidth can be as r times wider than the physical bandwidth. Although the decompressor and compressor slightly increase the delay, stream computing is inherently tolerant to latency because stream computing is usually pipelined with many stages. Therefore we focus on only throughput of compression and decompression for high-performance stream computation.

2.2.1 Requirements for bandwidth compression

For bandwidth compression proposed in our preceding study, we made a choice of a data compression algorithm based on the following requirements [26].

1. Lossless data compression,
2. Direct compression of numerical data,
3. Single-pass compression,
4. Acceptable compression performance,
5. High-throughput.

Data compression should be lossless so as to reconstruct the original data from compressed data to avoid unnecessary errors. For the lossless and direct compression, the prediction-based lossless algorithms [44, 59, 45, 42, 60] have been proposed, which achieve better compression ratios than general-purpose compressors like BZIP2[61] for direct compression of floating-point data. In

these algorithms, predictors calculate the values of the next input data by previous input data in a sequence. Then they encode the difference between the predicted value and the actual input value. The single-pass compression means that the entire data to compress cannot be traversed in advance because it must process in the stream computing. The prediction-based compression algorithms also satisfies the single-pass. For requirements of compression performance and high-throughput, arithmetic predictors [44, 59, 45] is suitable to be employed. For these reasons, we adopted prediction-based lossless compression with 1D arithmetic predictor to compress a floating-point data stream directly [27, 26].

2.2.2 Prediction-based compression algorithm

Since our target application is mainly numerical simulations, the algorithm that we apply to bandwidth compression directly compresses a sequence of IEEE754 floating-point data [27, 26]. Each IEEE754 datum represents the following number:

$$(-1)^s 2^{e-2^{n_e-1}-n_m+1}(2^{n_m}+m), \tag{2.1}$$

where s, e and m show a sign bit, an exponent and an mantissa, respectively. For single (32 bit) and double (64 bit) precision, (n_e, n_m) are (8, 23) and (11, 52), respectively.

We assume that a data stream represented by IEEE754 floating-point, $S = \{..., f_{i-2}, f_{i-1}, f_i, ...\}$, is a target of the compression, where f_i denotes the current input. generally, a data stream is generated by traversing a 2D or 3D computational grid of an object space. For f_i, the predictor calculates a predicted value, p_i, with some of the previous input data recorded in a buffer memory. When the prediction is made with good accuracy, p_i has a very similar bit pattern to the f_i. By a difference calculation or exclusive OR between p_i and f_i, we get a bit string where a lot of bits from MSB are zeros. By encoding these zeros with their length, we can represent the difference between the predicted value and the original value in fewer bits.

Predictor

The computational results of numerical simulation such as CFD have some spatial and temporal continuity. Since these results are the solutions of the partial differential equations, we can find continuity in discrete solutions on computational grids. Proposed prediction-based compression generally exploit these continuity for accurate predictions, for example, [45] employ Lorenzo predictor which is an extended parallelogram predictor as shown in 2.1. This predictor achieves good compression performance [62], however, there is a problem for hardware implementations. Since the stream computing employs a pipeline processing, it needs to hold consecutive data which arranged in one-dimensional by scanning the computational grid. To hold these data, it needs large buffer including the wasteful part shown in Fig. 2.2. Fig. 2.2 shows the case of two

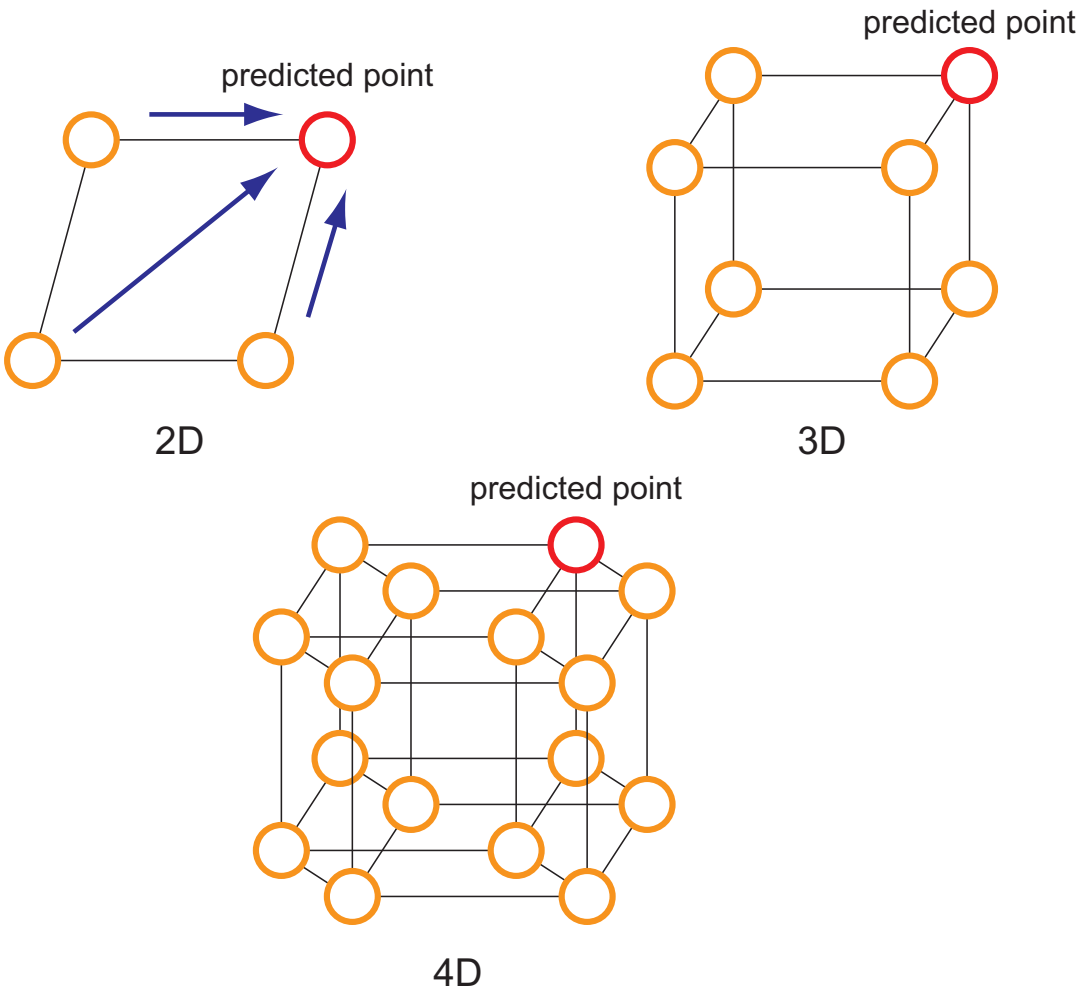


Figure 2.1: Lorenzo predictor.

dimensional grid, in addition, three dimensional computing requires to hold a huge number of data, which causes too large buffer to implement on one FPGA.

Therefore, we employed one-dimensional polynomial function to calculate predicted values, which requires only several recent input without the wasteful part of buffers. This is a technique to predict using the continuity of only one direction regardless of the number of dimensions 2.3. This prediction based on 1D polynomial functions allows us to assume that several consecutive data can be well locally-approximated by polynomial functions, which is similar to the proposal of [63]. Under this assumption, the *polynomial predictors* obtains good prediction for data given by such computations.

Assuming that a numerical input sequence is $S = \{..., f_{i-1}, f_i\}$, and the next input is f_i,

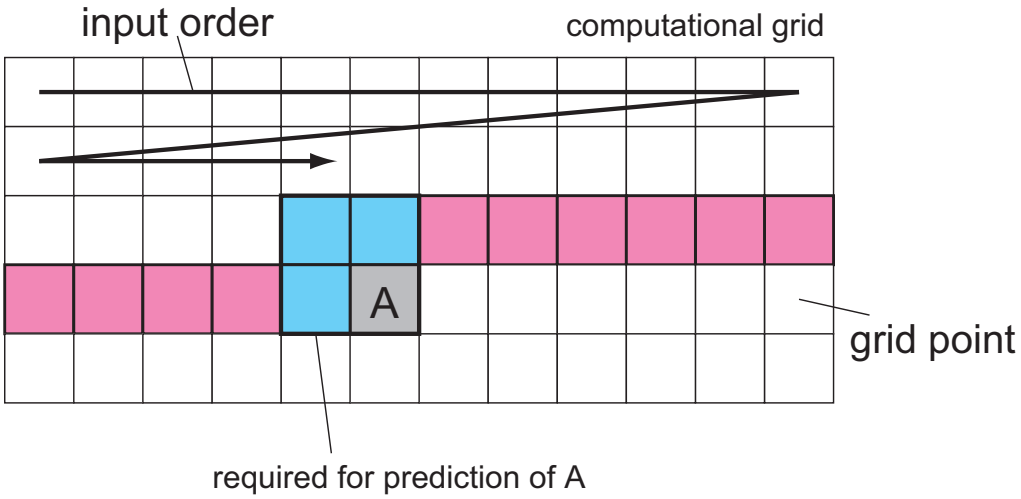


Figure 2.2: Required data on the computational grid for the 2D Lorenzo predictor.

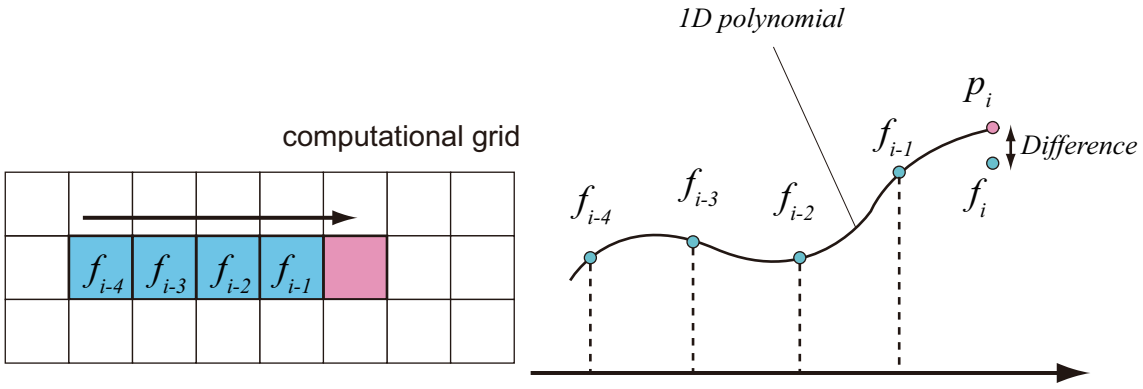


Figure 2.3: 1D polynomial predictor.

following equations are used in the polynomial predictions.

$$p_i = \begin{cases} f_{i-1} & (n=1) \\ 2f_{i-1} - f_{i-2} & (n=2) \\ 3f_{i-1} - 3f_{i-2} + f_{i-3} & (n=3) \\ 4f_{i-1} - 6f_{i-2} + 4f_{i-3} - f_{i-4} & (n=4) \\ 5f_{i-1} - 10f_{i-2} + 10f_{i-3} - 5f_{i-4} + f_{i-5} & (n=5) \\ 6f_{i-1} - 15f_{i-2} + 20f_{i-3} - 15f_{i-4} + 6f_{i-5} - f_{i-6} & (n=6). \end{cases} \tag{2.2}$$

We also refer to each of the predictors as *Constant*, *Linear*, *Quadratic*, *Cubic*, *Quartic*, and *Quintic* predictors for $n = 1$, 2, 3, 4, 5, and 6 respectively. Thus, the 1D polynomial prediction p_i of f_i is formulated as

$$p_i = c_{-1}f_{i-1} + c_{-2}f_{i-2} + ... + c_{-n}f_{i-n}, \tag{2.3}$$

where c_k are coefficients. Higher order predictors in these require more data for calculations,

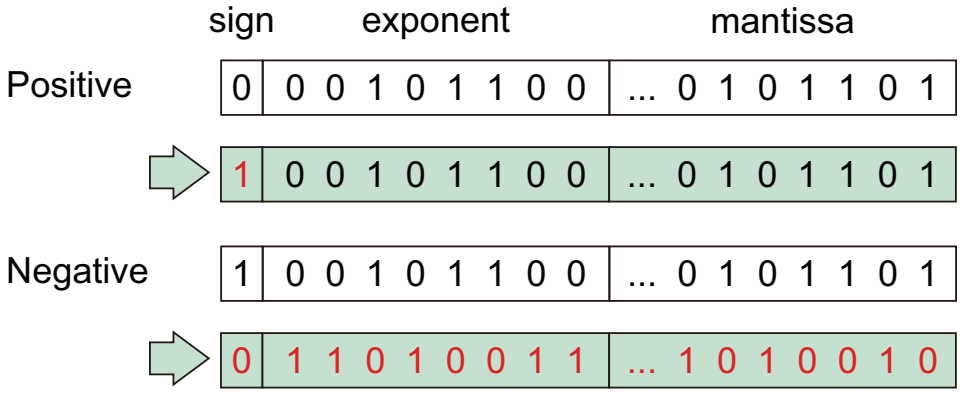


Figure 2.4: Integer conversion of floating-point bit strings.

which causes both a more precise prediction and an increase of area. Note that we can apply other extrapolation method by giving different coefficients.

Compared to higher-dimensional predictors such as Lorenzo predictor in hardware implementation, the 1D polynomial predictors require very few data to hold on the hardware. For example, the 1D polynomial predictor requires only n buffers for any number of dimensions of the computing. On the other hand, Lorenzo predictor requires $X_{res}+2$ for 2D, and $X_{res}Y_{res}+Xres+2$ for 3D. X_{res} and Y_{res} indicate resolutions of x and y direction, respectively. Therefore, the 1D polynomial predictor has much higher adaptability than predictors tailored to the number of dimensions.

Difference encoder

After the prediction, we encode the difference between p_i and f_i with its length of residual bits (LRB) which shows the number of the remaining bits except successive zeros from MSB, and the remaining bits themselves. We refer to the remaining bits as *residual bits* and denoted by r. Many studies employed leading-zero count (LZC) which is the number of successive zeros from MSB for prediction-based lossless algorithms [63, 44, 59, 42, 60]. We employ LRB in order to simplify the processing at the decompression instead of LZC.

The prediction-based algorithms are classified into two groups for the encoding: arithmetic-based one [44, 59, 45] and context-based one [42, 60]. The arithmetic-based algorithm uses an arithmetic predictor to obtain the prediction by calculation. The context-based algorithm uses a hash table to look up a datum that appeared after the same input phrase to predict the next input. We select the arithmetic-based algorithm because it allows hardware to be faster and smaller than that for the context-based algorithm. The hash table requires a large on-chip memory and the memory update for every prediction limits an operating frequency.

We employ an *integer prediction* and *subtraction* for a small hardware and a high throughput.

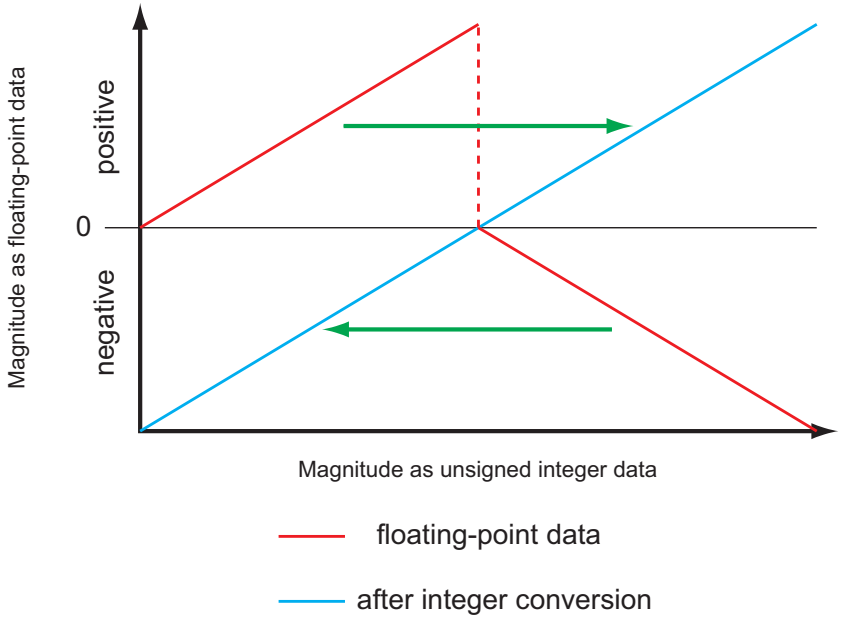


Figure 2.5: Continuous mapping of integer conversion.

The integer-subtraction calculates the difference between p_i and f_i which are converted to unsigned integers, therefore, the accurate prediction always give small difference even as converted integers. The integer conversion is performed by flipping the sign bit for positive FP numbers, or all bits for negative numbers [44] shown in Fig. 2.4. As a result, the positive and negative FP numbers are mapped continuously to the higher and lower space of unsigned integers, respectively, as shown in Fig. 2.5.

We apply the integer conversion to the input data in advance before the prediction, therefore, all numerical data are handled as unsigned integer. While p_i and f_i denote numerical values in the floating-point format, P and F denote converted integers of p and f, respectively. Then we compute the difference, $D = P - F$ for $P > F$, or $D = F - P$ for $P \leq F$. Since an additional bit is required to indicate a magnitude relation between P and F, we define an extra bit ex, where

$$\mathrm{ex} = \begin{cases} 1 & \text{for} \quad P > F \\ 0 & \text{for} \quad P \leq F. \end{cases} \tag{2.4}$$

As the results of the operations made at this moment, we have three bit strings which are denoted by ex, LRB, and residual bits, respectively. We can reconstruct the original F completely with these bit strings as ex, LRB, residual bits. Therefore, the compressor must send these three bit strings every one of them. To output these bit strings with fixed bit-width, the most simple and easy method is concatenating these bit string and cut out a output block from the concatenated bit string according to the output width. Then the output stream from the compressor contains these three bits completely. Although the decompressor receives the compressed stream without any additional information of the length of each bit string, it can

reconstruct the original data with expansions from the beginning of the stream because ex and LRB have fixed length of bits. The decompressor can obtain the length of residual bits by examining LRB. After the decompressor extracts residual bits, it reconstructs the original bit string with an original bit length as D, then we employ the predictor again to calculate P_i which is completely the same with the P_i in the compressor.

4-bit coding

The single-precision floating-point data consists of 32 bits per one datum. When we handle 32-bit datum, *4-bit coding* achieves an effective data compression on hardware [42, 60, 27]. The obtained LRB can be recorded as it is, however we used the 4-bit coding for easier handling of a variable length of the residual. Here we assume that an FP number has 32 bits for single precision. In this case, LRB can be 1 to 32, which is represented in 5 bits. The residual has LRB bits. On the other hand, the 4-bit coding represents LRB with a multiple of 4. For example, $LRB = 15$ naively gives 15 residual-bits. In the 4-bit coding, the LRB is truncated to 16, and the residual becomes 16 bits being padded with unnecessary 0s. The 4-bit coding is expected to have the advantage of 4-bit alignment that allowed us to more simply output residuals with variable bit-length.

For IEEE754 single-precision floating-point data, the procedure of the integer subtraction and the 4-bit coding is summarized as follows. We converted p_i and f_i to their unsigned integer, P and F, respectively. We computed D by subtraction, and obtained ex $= 1$ or 0 for $P > F$ or not, respectively. Then we obtained the LRB of D. Instead of using 5 bits to represent the LZC, we encode $\lceil \mathrm{LRB}/4 \rceil$ with 3 bits so that the LZC is a multiple of 4. We padded necessary 0s to r. The length of encoded bits is $(1 + 3 + 4\lceil \mathrm{LRB}/4 \rceil)$ for $\{\mathrm{ex}, \mathrm{LRB}, r\}$.

2.3 Prediction based on 1D polynomial function

This chapter describes a derivation of 1D polynomial functions shown in equations 2.2. The polynomial functions for the prediction are based on Lagrange interpolation. When (n=2) in equations 2.2, the polynomial achieves a linear interpolation for the next value. Therefore, the polynomials with (n+1) consecutive values achieve n-th order Lagrange interpolation.

Lagrange interpolation is defined as

$$\begin{cases} p_n(x) = \sum_{k=0}^{n} a_k l_k^{(n)}(x) \\ l_k^{(n)}(x) = \prod_{i=0, i\neq k}^{n} \frac{x-x_i}{x_k-x_i} \quad (k = 0, 1, ..., n). \end{cases} \tag{2.5}$$

$p_n(x)$ is n-th order polynomial, and $a_k (k = 0, 1, ..., n)$ are coefficients of $p_n(x)$. $l_k^{(n)}(x) (k = 0, 1, ..., n)$ are called Lagrange polynomials, which are n-th order polynomials and independent of each other. With assuming $l_0^{(0)}(x) = 1$, it has following properties,

$$l_k^{(n)}(x_i) = \begin{cases} 1 & (i = k) \\ 0 & (i \neq k). \end{cases} \tag{2.6}$$

Therefore, the following equation is satisfied.

$$p_n(x) = \sum_{k=0}^{n} a_k l_k^{(n)}(x) = a_i \quad (i = 0, 1, ..., n). \tag{2.7}$$

If it assumes $a_k = f(x_k)$, we obtain Lagrange interpolation formula as

$$p_n(x) = \sum_{k=0}^{n} f(x_k) l_k^{(n)}(x) = a_i. \tag{2.8}$$

Please note that the order of the polynomial is at most n-th.

With Lagrange interpolation, the prediction is enabled by obtaining the coefficients a_k for various orders of polynomials. For example, 1-st order polynomial ($n = 1$) is described as

$$p_n(x) = f(x_0)\frac{x - x_1}{} + f(x_1)\frac{x - x_0}{x_1 - x_0}. \tag{2.9}$$

In addition, since it assumes that numerical data are sampled at equal interval, and assuming $(x_0 - x_1 = 1)$, we obtain the coefficients as shown in 2.2. Therefore, it is possible to obtain predicted values by constant multiplications and simple additions. For the hardware implementation, Lagrange interpolation can be achieved by a smaller area of the hardware comparing other interpolations such as Newton interpolation.

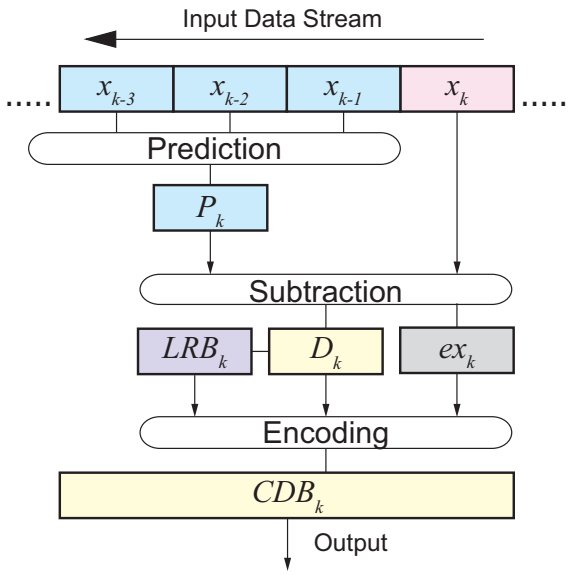


Figure 2.6: Data compression algorithm.

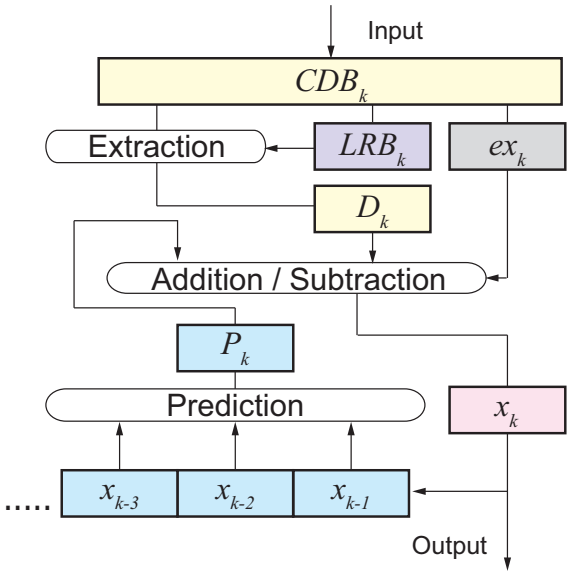


Figure 2.7: Data decompression algorithm.

2.4 Hardware design

Based on the compression algorithm mentioned above, we designed hardware compressor and decompressor for a single channel of single-precision floating-point numbers [26]. Figs. 2.6 and 2.7 show the overviews of the data compression and the decompression. One datum in a floating-point data stream is input to the compressor and it outputs a compressed data block as a stream every clock cycle. We employ *block coding* to encode method for hardware data compression because the output of the compressor needs to fit the I/O of the memory controller on a FPGA. A *compressed data block (CDB)* is the output data block which contains a number of compressed data, whose bit length is determined by an I/O bit width from or to the memory. The decompressor receives the CDB, and extracts individual elements from the CDB, then it operates in the same way as the compressor to reconstruct original data.

2.4.1 Compressor

Fig. 2.8 shows the overview of the compressor, which is pipelined and consists of several modules which have their specific functions. The compressor is composed of *binary translation unit (BTU)*, *a predictor with buffers*, *a difference-computing unit (DCU)*, *an LRB unit (LRBU)* and *a variable-to-fixed length converter (VFC)*. The input floating-point datum f is firstly converted to an unsigned integer F by flipping the sign bit for a positive number or all bits for a negative number. The converted integer is stored in the buffer. The buffer stores a necessary number of previous inputs.

The predictor computes equations (2.2) to get P for the current input Fs by integer operation.

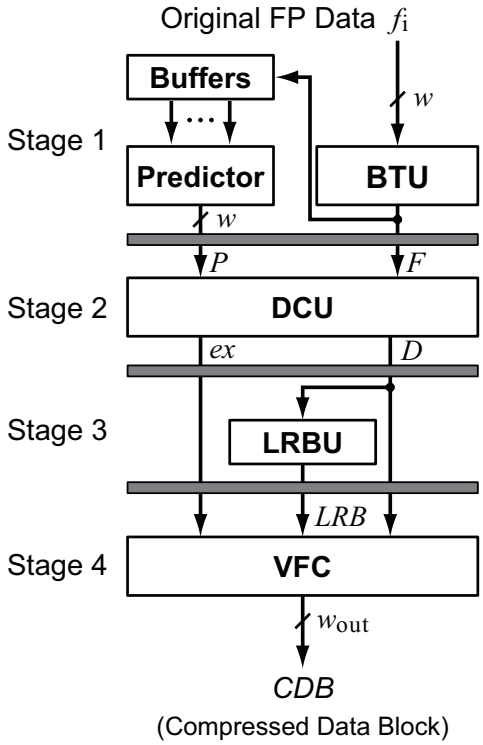


Figure 2.8: Hardware design of the compressor.

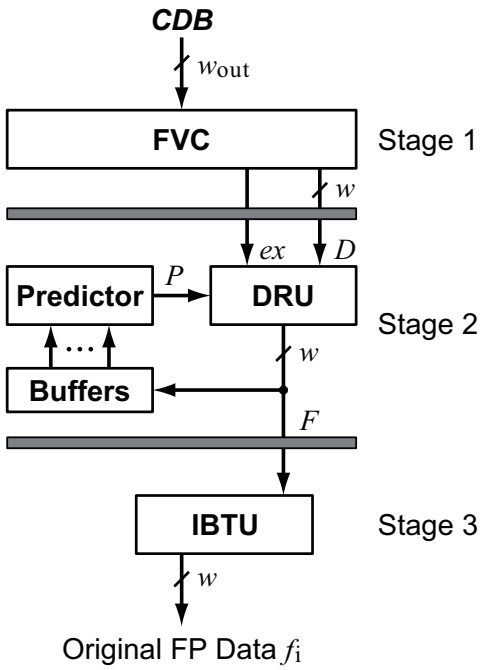


Figure 2.9: Hardware design of the decompressor.

Fig.2.10 shows the structure of the cubic predictor with n=4, which takes the last four inputs from the buffer in parallel for the prediction. The predictor uses integer operations for prediction with the cubic equation (2.2) instead of floating-point operations as described in section 2.2. Since integer operations are less complicated than floating-point operations, integer operations are more suitable for fast and small hardware implementations. Our preceding work [26] show that the integer prediction provides approximately equal compression performance as that by the floating-point prediction. A floating-point operation needs to divide bit strings of each datum and operate separately because of its structure consisting of three parts, sign, exponent, and mantissa. As shown in Fig. 2.4, converted bit strings still keep continuities as in floating-point, which consists of sign, exponent, and mantissa from MSB. If there are no change in exponent bits among the data used for prediction, the integer conversion has only a small impact on the prediction because converted data have been kept the same magnitude relationships as in the case of floating-point. In the case of numerical computing by CFD, the integer prediction achieves a comparable performance that by the floating-point prediction because the computational results do not have large fluctuation.

DCU computes the difference, D, between P and F by subtracting the smaller from the larger after swapping P and F if necessary. DCU also outputs the exchange signal, ex, which

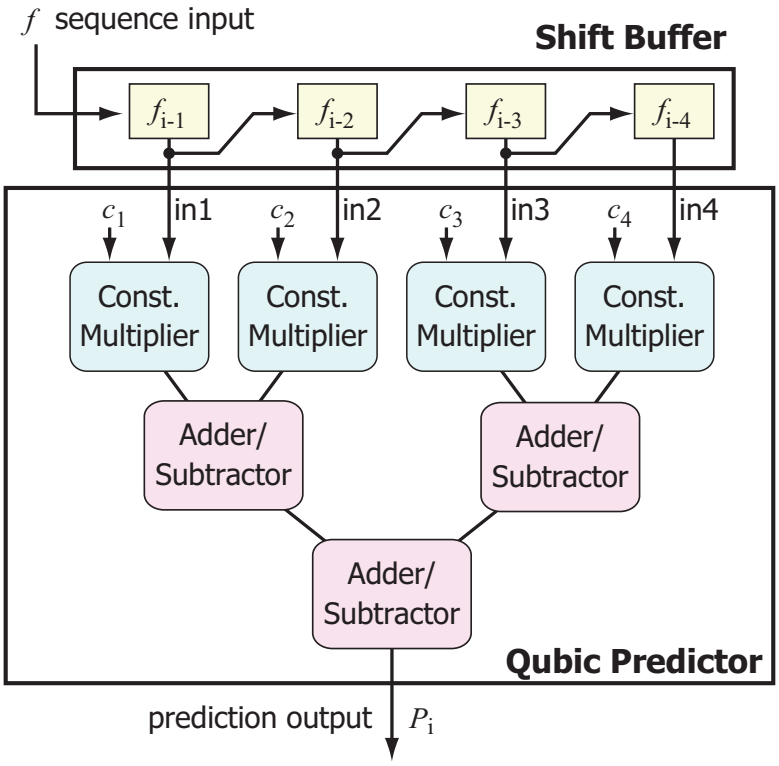


Figure 2.10: Cubic predictor.

is asserted when P and F are swapped. LRBU computes (LRB) of D. Finally VFC outputs words of compressed data, which are composed of $\{r, \mathrm{LRB}, \mathrm{ex}\}$.

2.4.2 Decompressor

Fig. 2.9 shows the overview of the decompressor. The decompressor consists of *a fixed-to-variable length converter (FVC)*, *the predictor with the buffer*, *a data-reconstruction unit (DRU)* and *an inverse binary translation unit (IBTU)*. The predictor and the buffer are the same as those of the compressor. The decompressor outputs the original floating-point data stream with the input of the compressed data stream. FVC generates D with ex, LRB and r. The predictor gives prediction P with the previously decompressed numbers stored in the buffer. DRU reconstructs the integer of the original data, F, with D, P and ex by performing the inverse operation of the difference-computation. Finally IBTU converts F to its floating-point number f.

As shown in Figs. 2.8 and 2.8, the compressor and the decompressor are pipelined with the four stages and the three stages, respectively, to process one floating-point datum every cycle at a high operating frequency. The details of the units in the compressor and decompressor except VFC and FVC were presented in [26].

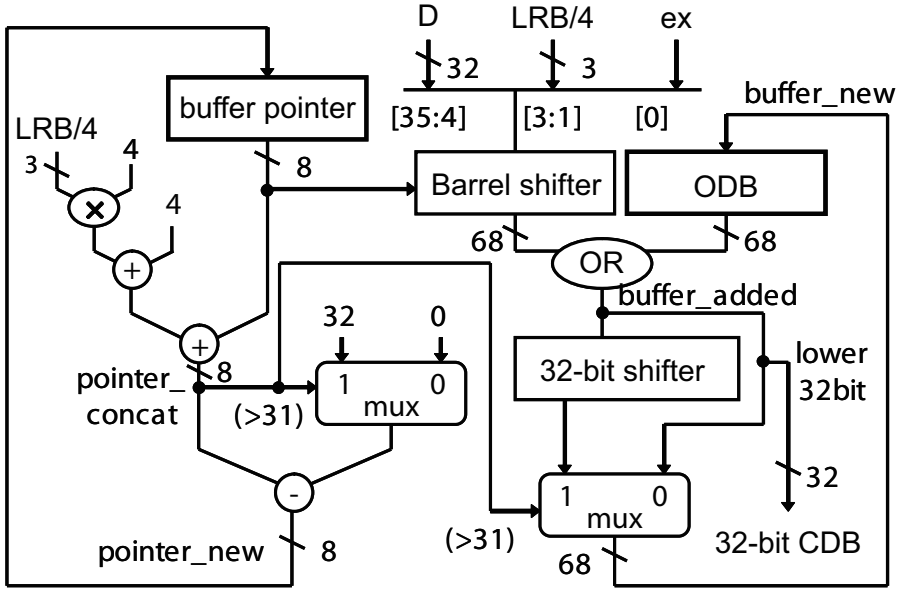


Figure 2.11: Variable-to-fixed length converter.

2.4.3 VFC and FVC

The variable-to-fixed length converter (VFC) and the fixed-to-variable length converter (FVC) are designed to handle variable length compressed data. These modules are also important for conversion between variable-length compressed and fixed-length CDBs. Therefore, both output of VFC and input of FVC are intermittent due to generating and extracting CDBs. The following sentences describe the detailed design and the behavior of VFC and FVC.

Fig.2.11 shows the design of the variable-to-fixed length converter (VFC) with 32-bit CDBs. VFC consists of *a 68-bit output data buffer (ODB)* and *a buffer pointer*. The buffer pointer specifies the number of bits accumulated in *ODB* at the moment. *ODB* accumulates the inputs in *ODB* and outputs a 32-bit word when the buffer pointer is greater than or equal to 32.

VFC is in the stage 4 of the compressor. It receives ex and D from DCU and (LRB/4) from LRBU, and outputs words of compressed data. The input is of 8 to 36 bits with a variable length of a multiple of 4, while the output is a fixed 32-bit word. In order to decode compressed data, our compression employs LRB to calculate compressed data length at the decompressor. When the decompressor decodes compressed data, information of lengths of every compressed datum is necessary. Because the CDBs are concatenated bit strings of several {ex, LRB/4 and r}, the decompressor always finds LRB/4 and calculate the right length of the compressed data.

First, VFC concatenates ex, LRB/4 and D into a block, so that they are aligned from LSB to MSB. Next, the block is shifted left so that the block exists in the next empty bits of ODB. The shift amount is specified by *Pointer*. The shifted block is inserted into the next empty positions of the present ODB by OR operation, giving *buffer_added*. The width of the block is obtained

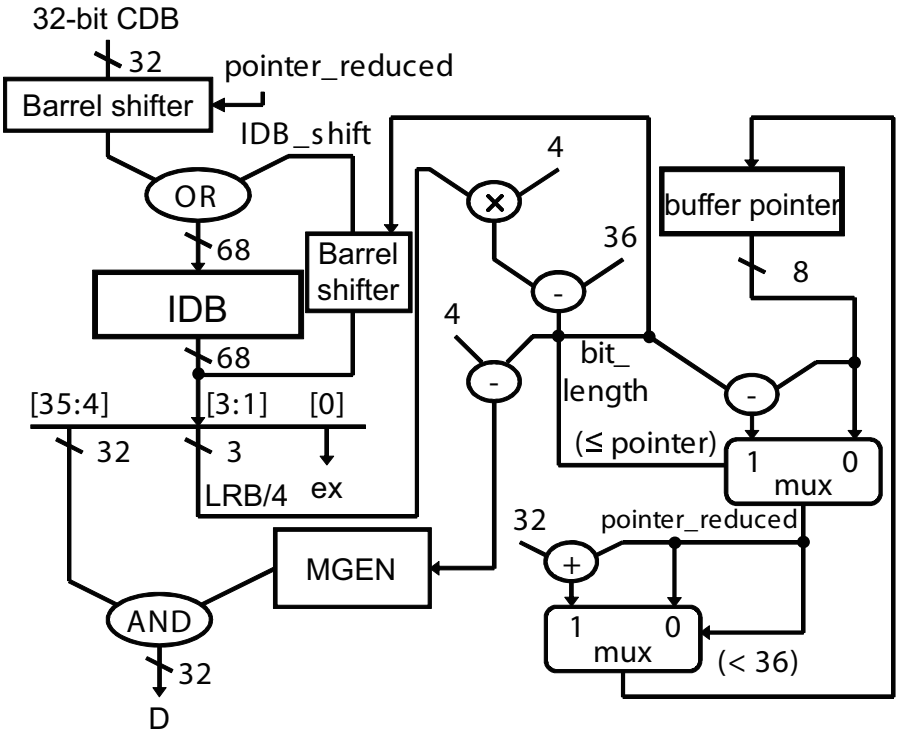


Figure 2.12: Fixed-to-variable length converter.

with LRB/4, then *pointer_concat* is computed by adding *Pointer* and the width of the block.

If *pointer_concat* is greater than or equal to 32, the lowest 32 bits of *buffer_added* are output with a valid signal as the CDB. Simultaneously, *Buffer_added* is shifted 32-bit right to form *Buffer_new*, which is used to update *ODB* for the next cycle. *Pointer* is also updated with the value given by subtracting *pointer_concat* with 32. Then the next datum is input to VFC. If *Pointer* exceeds 67, *ODB* and the buffer pointer are not updated and a signal is output to stall the entire pipeline of the compressor. When the last datum is input into VFC, *ODB* can have remaining bits shorter than 32 bits. We flush them out at the end of compression with zeros padded to form a 32-bit word.

Fig.2.12 shows the design of the fixed-to-variable length converter (FVC) with 32-bit CDBs. FVC is in the first stage of the decompressor. It receives words of the compressed data containing ex, (LRB/4) and *residual bits*, and outputs ex and D to DRU for each encoded number. FVC consists of *a 68-bit input data buffer (IDB)*, *a buffer pointer* and *a Mask Generation Unit (MGEN)*. Along with VFC, the buffer pointer indicates the number of bits accumulated in *IDB*. *IDB* receives and accumulates the next word when the buffer pointer is less than 32. *MGEN* makes a 32-bit mask to clear unnecessary bits from the output of D.

IDB stores ex, LRB/4 and *residual bits* for multiple compressed numbers. By reading LRB/4, the length of the *residual bits* is computed. If the entire *residual bits* exist in *IDB*, they are output

as D and IDB is shifted. Then the next input word is inserted into IDB.

The input word is shifted left so that it fits the next empty positions in IDB. The shift amount is given with $pointer\_reduced = (Pointer - bit\_length)$, where bit_length is the length of *residual bits*, (LRB/4), ex in Fig.2.12. If $pointer\_reduced$ is less than 36, the shifted bits are inserted into IDB by using OR operation, and the buffer pointer is updated with $(pointer\_reduced + 32)$. In order to extract D from IDB, it needs to get the bit length of each compressed data. Since the bits of (LRB/4), ex are always at the LSB of IDB, we can directly read (LRB/4) and use it to obtain the bit length of *residual bits* by computing $(bit\_length - 4)$. Then $MGEN$ makes a 32-bit mask with bit_length. The 32-bit D is generated by performing AND operation with the mask and the 32-bit of IDB [35:4].

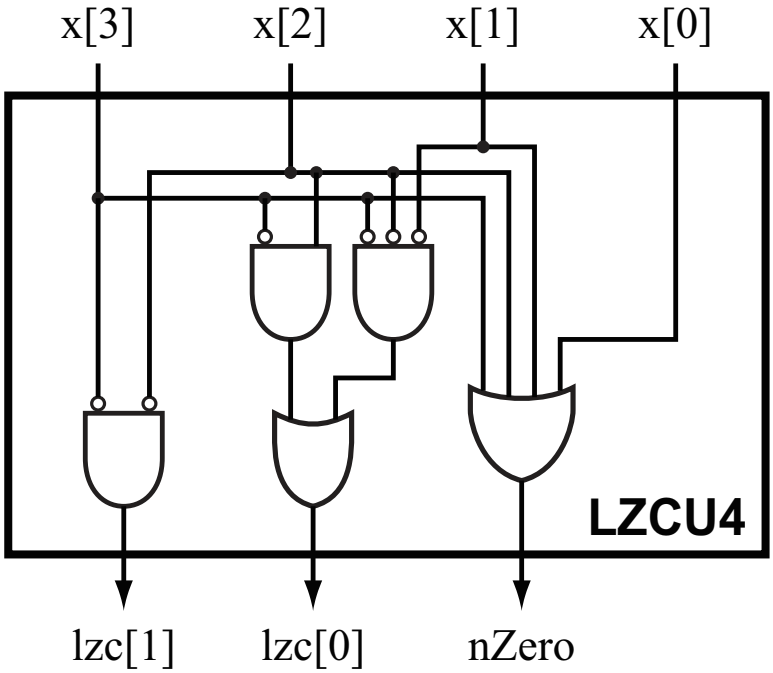


Figure 2.13: LZCU for 4 bits input.

2.5 Parametrized design of the compressor and decompressor

For the compressor and decompressor to be able to process various data and various lengths of CDBs, we parameterize their design. Here we describe structures of the major components: the predictor, VFC, FVC, and the LRBU. The other components are simple, and easy to be parameterized.

As shown in Fig. 2.10, the width of the buffer and operators in the predictor is parameterized with an input data width, w. Fig. 2.11 shows the structure of the VFC. The VFC converts variable-length inputs to w-bit fixed-length outputs by using the pointer and the buffer. The pointer manages the MSB position of the data in the buffer. According to the pointer, the left barrel shifter adjusts the bit position where the input is inserted in the buffer. The VFC is composed of the two data-paths to update the pointer and the buffer, respectively. Here we define three parameters, w_0, w_1, and w_2, which are the length of the raw LRB, width of the buffer, and the length of concatenated data of D, and LRB/4, and ex. They are obtained by the following equations: $w_0 = \log 2w$, $w_1 = 2w + w_0 - 1$, and $w_2 = w + w_0 - 1$.

Fig. 2.12 shows the FVC in the decompressor, which also has the pointer and buffer for conversion. The pointer is updated according to the update status of the buffer. The data in the buffer is shifted by the right barrel shifter when a datum is output from the buffer. The FVC is also parameterized with the parameters, w, w_0, w_1, and w_2.

To generate LRB, we employ LZC unit which generates LZC of bit strings, then we obtain

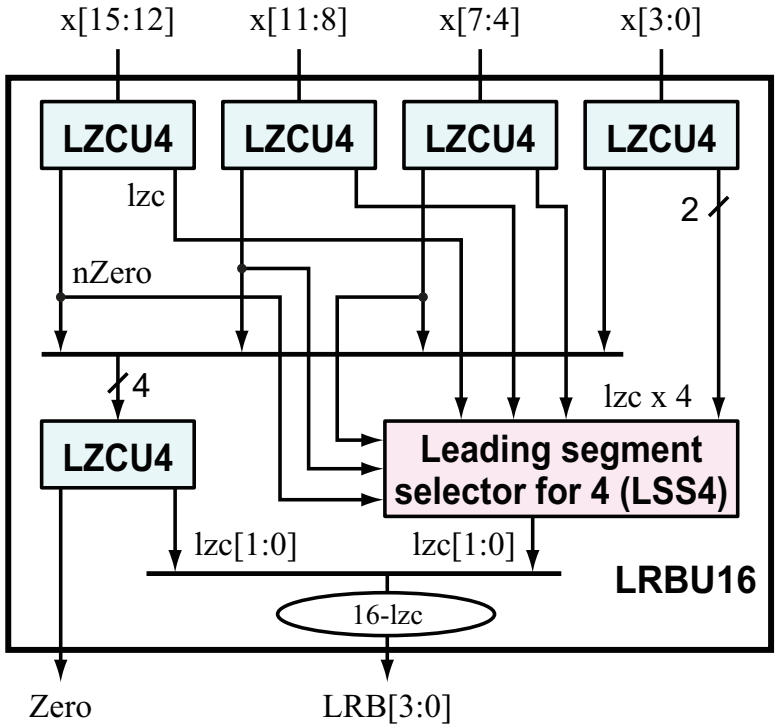


Figure 2.14: LRB unit for 16-bit input.

LRB by subtracting of $(w - LZC)$. We designed LRBUs for various bit width by combinating several 4-bit LZC unit, called LZCU4 as shown in Fig. 2.13. The LZCU4 has a 4-bit input and outputs of 2-bit lzc and 1-bit non-zero (nZero) signal. We built an LRBU for a 16-bit input, called LRBU16, with five LZCU4s as shown in Fig. 2.14. In the LRBU16, we also have the leading-segment selector for 4 (LSS4) to select one from four lzc from the four LZCU4s. The 4-bit lzc of the LRBU16 is generated by concatenating the 2-bit lzc from the second-stage LZCU4 and the LSS4. Thus we can compose a four times wider LRBU by combining four LRBUs. We also employ an LSS2 and two LRBU16s for a 32-bit LRBU, and a LSS4 and four LRBU16s for 64-bit LRBU. We realize wider input LRBUs in the same way as mentioned above. LSS provides an overall LZC in LRBU, which counts the successive zero bits from MSB to LSB. LZCU is designed for 2- and 4-bit inputs, and LSS is designed for two and four input LZC, which cope with various data types by combining these modules. Therefore, it is possible to apply the LRBU to various data which have powers of 2 bits.

The parameterized design allows us to apply the data compressor and decompressor to real-world implementations on FPGA because the bit length of CDBs must be the same with I/O bit width of a memory controller. Since required length of a CDB is determined by specifications of memory and FPGA, the parameterized design can be applied to various systems on different devices. Moreover, it can handle a variety of data type such as double-precision floating-point

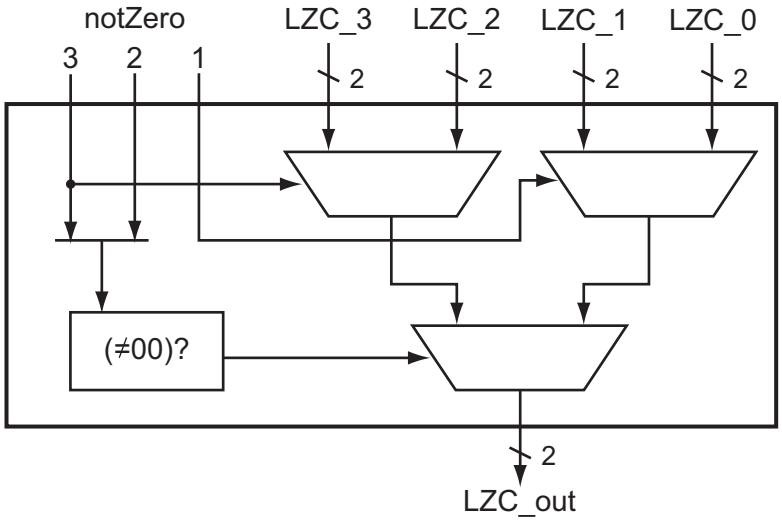


Figure 2.15: Leading-segment selector for 4 inputs.

as well as single-precision.

2.6 Evaluation of the compressor and decompressor

The proposed data compressor and decompressor need to satisfy the requirements described in section 2.2. Therefore, we evaluate the hardware implementation of the compressor and decompressor in terms of the compression ratio, the hardware area, and the maximum operating frequency (Fmax). In related studies and our preceding studies, the entire designs were not presented, especially the output formats were ignored. An appropriate output format is necessary to exploit an available bandwidth by the compressed data. Therefore, the data compression mentioned in previous sections is unique and essential for the bandwidth compression. To apply to the bandwidth compression, we need to evaluate the hardware implementation of the data compressor and decompressor. Firstly, we evaluate the hardware with 4-bit coding which designed for single-precision floating-point data and output 32-bit CDBs, in which we measure the compression ratio, hardware area, and Fmax. Then we also evaluate the parameterized design with various CDB sizes in terms of the area and Fmax.

We implement the compressor and the decompressor in the FPGA-based prototype system as shown in Fig. 2.16. The system consists of the PCI-Express Gen.2 x4 controller, the two external memory controllers, and the compressor and the decompressor with Altera Stratix IV FPGA on DE4 development board. The FPGA is one of the 40-nm series, which has 182400 ALUTs (adaptive lookup tables), which are logic cells to realize various logic circuits. Since the prototype system is designed just for verification and experiments, non-compressed data are read and written from/to the external memories to evaluate the performance of the compression and hardware.

We also evaluate the performance of the hardware with various data widths and CDB widths with the parameterized design. We set that both of the input and output widths of the compressor and decompressor are the same as 32, 64, 128, and 256 bits.

The prototype implementation is controlled by a software on the host PC. A data stream to compress is transmitted between the two external memory. Every processes for the stream is pipelined on FPGA. We used Altera's Qsys system integration tool [64] to generate the prototype implementation with the I/O controllers. The compressor and the decompressor operate at 125MHz. We implemented the four-stage compressor and the three-stage decompressor described above for IEEE754 single precision floating-point numbers. The compressor and the decompressor operate at 125MHz. The implemented circuits are written in Verilog-HDL, and compiled with Altera's Quartus II software ver.11.1 where "Speed" option is specified.

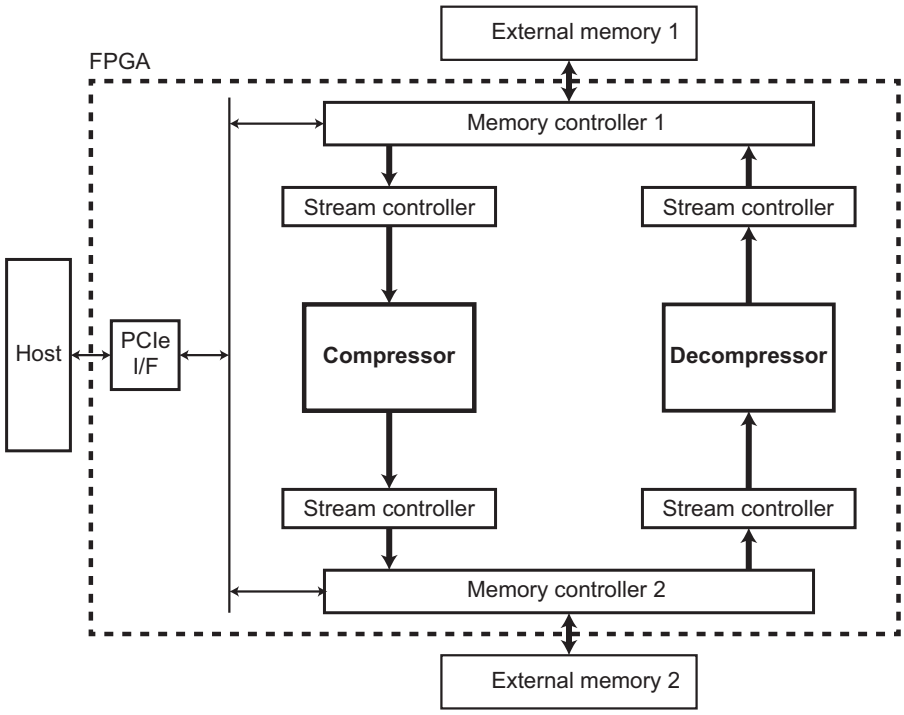


Figure 2.16: Prototype implementation of the compressor and decompressor.

Table 2.1: Resource usage of the polynomial predictor.

Polynomial order	Logic cells	Registers
n=1	33	32
n=2	97	64
n=3	223	96
n=4	252	128
n=5	406	160
n=6	560	192

2.6.1 Prediction accuracy and area of predictor

First, we evaluate the prediction accuracy which affects the compression performance for a numerical data set of a real-world simulation. The predictor employs the 1D polynomial functions as shown in Eq. (2.3). We evaluate the predictors from 0-th to 5-th which are shown in Eq. (2.2) to investigate the prediction accuracy and the circuit area to select a predictor. We use the computational results of the two dimensional lattice Boltzmann method (2DLBM), which is one of the CFD schemes. The 2D-LBM data are obtained at 120,000 time steps on 1600x960 lattice. We predict single-precision floating-point data of 2D-LBM computational results in every 5000 time steps by these six predictors.

Fig 2.17 shows the graph of an average *LRB* in each time step. Since LRB influences

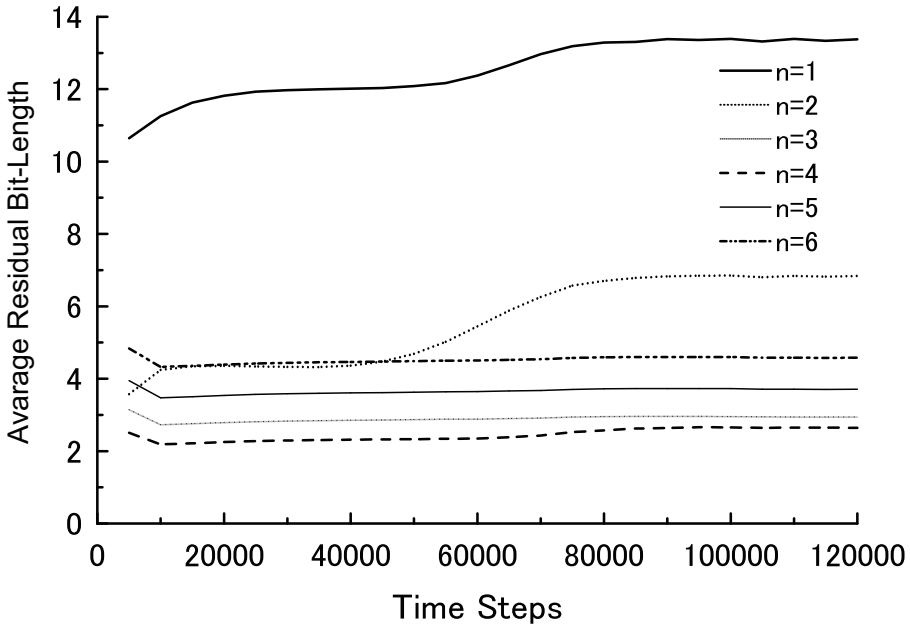


Figure 2.17: LRB of 2D-LBM Data Compression.

the length of a compressed datum, the prediction with small LRB leads the high compression performance. The shorter the residual bits are, the higher the compression ratios are obtained with in the encoding. The result shows that the predictor with $n = 4$ achieves the most accurate prediction. The higher order predictions have tendency to achieve higher accuracy. However, with $n = 6$ and $n = 5$, they are not shorter than $n = 4$. These higher order predictions achieve more consistent performance than lower order predictions. The prediction functions are based on the assumption that the data obey the $(n-1)$-th order function, thus the predicted values are on the function. Since the higher order functions are likely to be affected by small fluctuations, the prediction accuracy is not so high with a higher order function.

Table 2.1 shows the resource usage of these six predictors on a FPGA, which are almost in proportion to the order of predictors. In the result, the area of predictor is small giving an insignificant effect to the area of the entire system. From these points of view, we employ cubic $(n = 4)$ predictor with emphasis on compression performance considering the balance between the prediction accuracy and the area.

2.6.2 Resource consumption

Table 2.2 shows the resource consumption of the proposed compressor and the decompressor which employ the 4-bit coding with 32-bit output CDBs. The compressor uses only 909 ALUTs and 612 dedicated registers, which correspond to 0.5 % and 0.34 % of the total resources on EP4SGX230 FPGA, respectively. The biggest module in the compressor is VFC, which uses

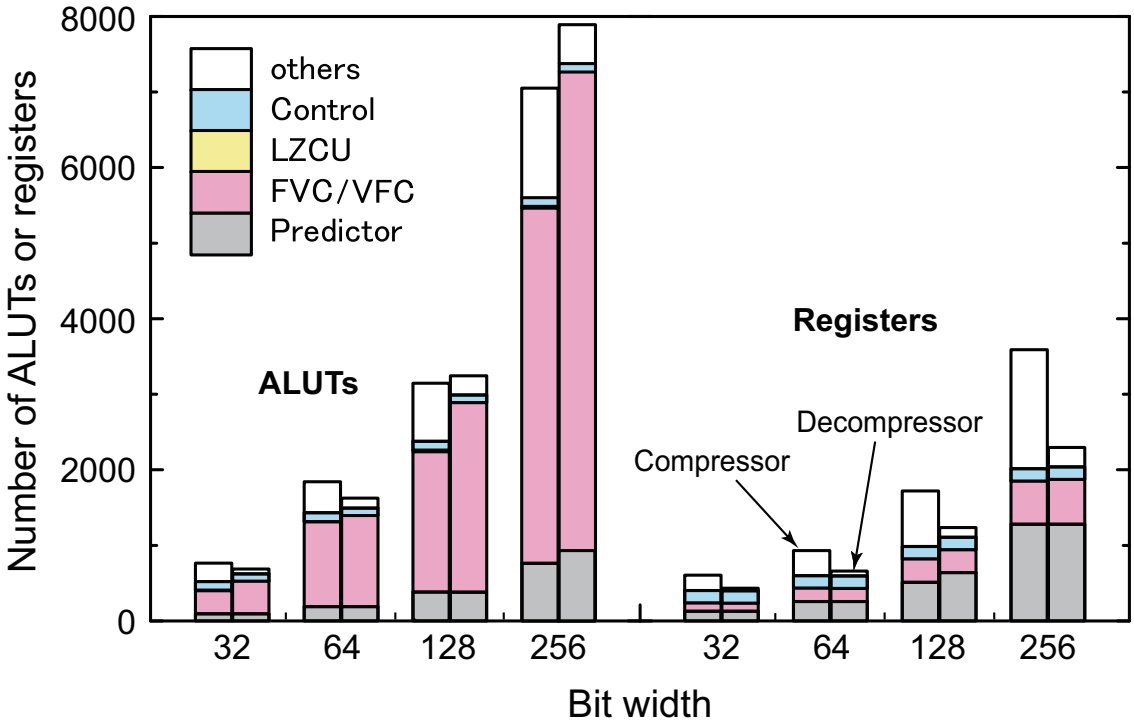


Figure 2.18: Resource usages of the parameterized compressor and decompressor.

313 ALUTs (0.17 %) and 110 registers (0.06 %). The decompressor is also very small, which consumes only 822 ALUTs and 439 registers which are 0.45 % and 0.24 % of the total resources, respectively. FVC is the largest module in the decompressor, using 428 ALUTs (0.23 %) and 106 registers (0.058 %).

These results show that the compressor and decompressor are so small that they account for less than 1 % of the total resources of this FPGA, which are less resources than those of [43]. In addition, they require no block memory and no DSP block. This is very important for auxiliary utilization of compression because such embedded hard macros should be used by major computing modules.

Fig. 2.18 shows resource usage of the compressors and decompressors with various data widths in the parameterized design. The data width shows input and output bit widths of the compressor and decompressor, and the input and output widths of both modules are the same. The "others" in the graph means other than shown in graph such as pipeline registers and some logics to connect the modules. The result shows that the number of ALUTs increases almost linearly. In the case of 32-bit width, the compressor and decompressor use 764 and 689 ALUTs, respectively, which corresponds to only 0.42% and 0.38% of the total ALUTs. This 32-bit hardware is very small, however, the wider the data width are, the larger the hardware area is. 64-bit width hardware is still small, consuming only about 2% of the total ALUTs, while the 256-

Table 2.2: Resource consumption of the compressor and decompressor.

Modules	ALUTs	Dedicated registers	Block memory bits
Compressor	909 (0.50%)	612 (0.34%)	0 (0.0%)
Predictor	195 (0.11%)	132 (0.72%)	0 (0.0%)
LRBU	4 (0.002%)	0 (0.0%)	0 (0.0%)
VFC	313 (0.17%)	110 (0.060%)	0 (0.0%)
Others	278 (0.15%)	204 (0.11%)	0 (0.0%)
Decompressor	822 (0.45%)	439 (0.24%)	0 (0.0%)
Predictor	228 (0.12%)	132 (0.072%)	0 (0.0%)
FVC	428 (0.23%)	106 (0.058%)	0 (0.0%)
Others	66 (0.036%)	35 (0.019%)	0 (0.0%)
Stratix IV EP4SGX230	182400	182400	14625792

bit designs require 3.87% and 4.33% ALUTs for the compressor and decompressor, respectively. The important modules for the hardware area is VFC and FVC, which occupy more than half area in the compressor and decompressor in the case of 128 and 256-bit, respectively. Other modules are less affects by the data width with the exception of the predictor which is much smaller than VFC and FVC.

The number of registers also increases as bit width increases. However, their numbers are moderate prepared to the ALUTs. The predictor is the largest module in terms of the number of registers in many cases, because the predictor requires four data buffer to hold previous data for the cubic prediction.

This result shows that the bit width of the input and output strongly affect the area. For real-world applications, the input of the compressor is usually 32 or 64-bit width, and output is wider than input in many cases. Therefore, the output width, which is the same with the length of CDB, is considered to affect the entire hardware area. We discuss and evaluate this impact in the next chapter.

2.6.3 Throughput

Fig. 2.19 shows the maximum operating frequencies of the parameterized compressors and decompressors. Both hardware can operate at higher than 232MHz and 201MHz for both 32-bit and 64-bit, respectively, while the frequencies decreases as increasing data widths. In the 256-bit case, frequency higher than 120MHz is still available for both compression and decompression. This result shows that we can apply this compressor and decompressor to dedicated circuits whose Fmaxs is lower than the compressor and decompressor.

Table 2.3 shows latencies of pipeline stages. The compressor has a critical path in the stage 4 including VFC for 32-bit and 64-bit, while the stage 1 including the predictor becomes a critical

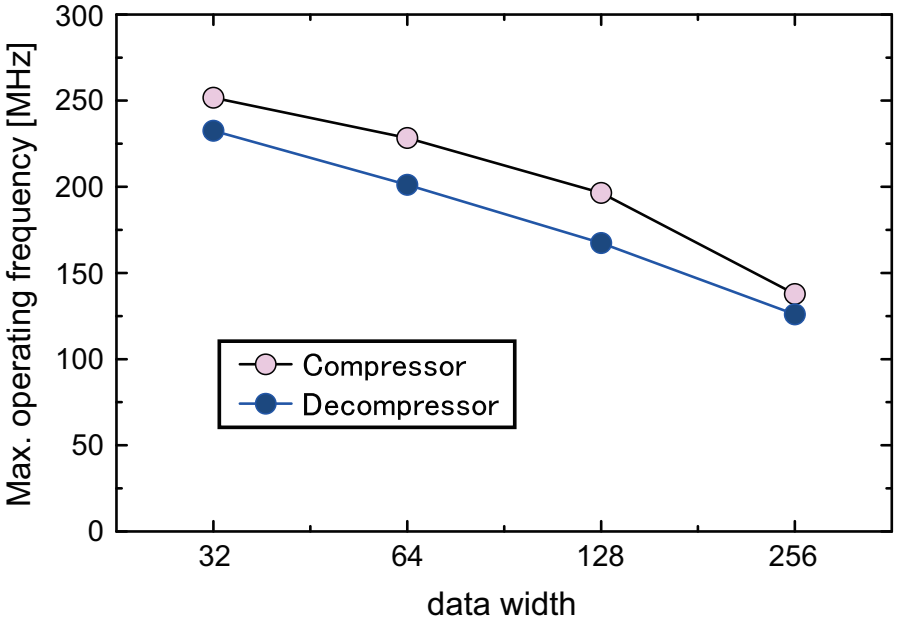


Figure 2.19: Maximum operating frequencies of the parameterized compressor and decompressor.

Table 2.3: Critical path delay and maximum frequency ($F_{\max}$) of each pipeline stage.

	Compressor				Decompressor		
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 1	Stage 2	Stage 3
Delay of critical path [ns]	**4.141**	3.946	3.596	4.045	4.550	**5.052**	3.439
$F_{\max}$ [MHz]	**241.5**	253.4	278.1	247.2	219.8	**197.9**	290.8

path for 128-bit and 256-bit. This is because the latency of the adders in the predictor gets longer than that of the barrel shifter in VFC for wider bits. On the other hand, the decompressor has a critical path in the stage 2 with the predictor for shorter bits, while the stage 1 including FVC gets to have a critical path for 256-bit. This is because the stage 2 has a long critical-path for the DRU, buffers, and the predictor. In both hardware, the feedback loop is responsible for increasing of delay, the stage 1 of the compressor and the stage 2 of the decompressor. To improve the throughput of the hardware, we need to employ faster predictor, or redesign without the feedback loops.

2.6.4 Compression ratio

To confirm that the data compression have effect for real-world numerical data, we evaluate the compression performance of the hardware by using 2DLBM data For verification, we used the computational results of the 2DLBM [26]. Each computational result is composed of nine

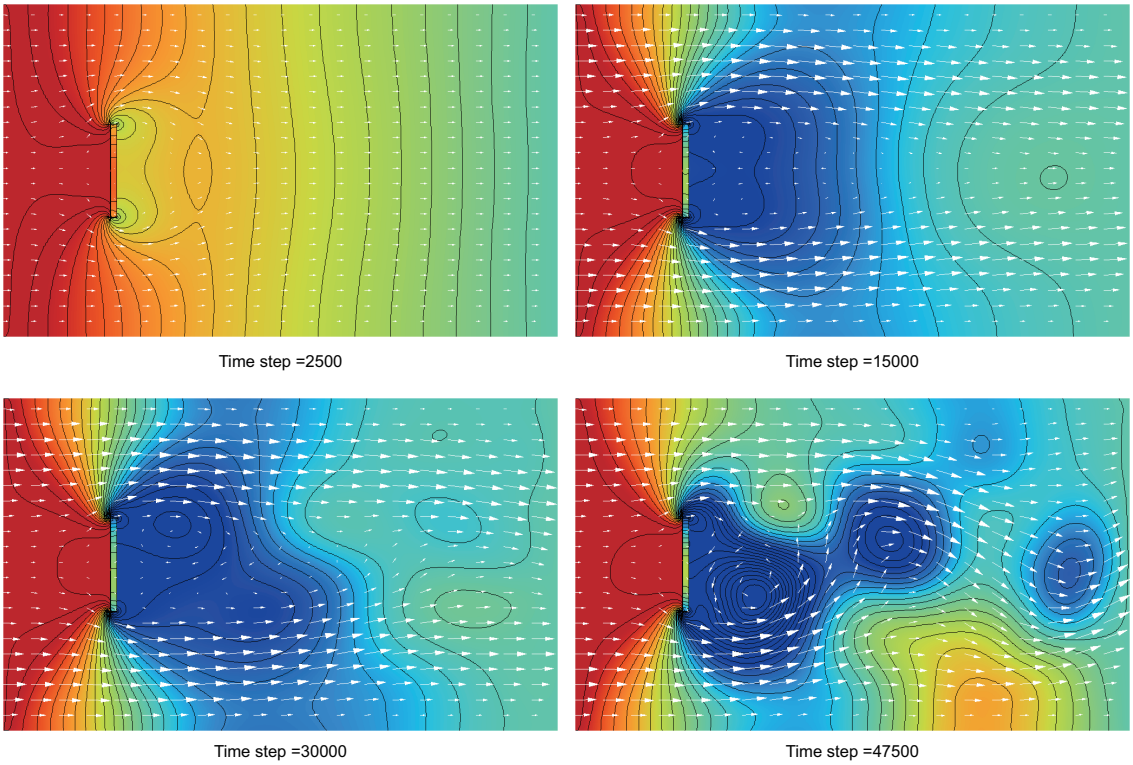


Figure 2.20: Numerical simulation of 2DLBM.

variables, each of which contains 800×480 floating-point numbers in single precision. Then we evaluated *a compression ratio* which is defined with $R_{\text{comp}} = \frac{\text{(Size of original data)}}{\text{(Size of compressed data)}}$. Figs.2.20 is numerical data of the results of 2DLBM for evaluation. The average compression ratio is $R_{\text{comp}} = 3.7$ in this evaluation, where most of floating-point numbers are compressed at the ratio of 4 while some few numbers intermittently have lower ratios from 2 to 3. Since the theoretical maximum compression ratio is $32/8 = 4$ in this algorithm, this result shows the compression is very effective for the 2DLBM data. The compression of the stream data means that the compressor and decompressor reduce a required bandwidth.

Finally, we measure compression ratios for 32 and 64-bit data which correspond to single-precision and double-precision of floating-point data, respectively. For evaluation, we used the test data and the CFD data shown in Figs. 2.21 and 2.22, and we do not employ the 4-bit coding which affects only a 32-bit compression. The test data are obtained by sampling a simple function, $f_i = \sin\left(\frac{2\pi\alpha x^2}{32768^2}\right) + \beta$, at 32768 points. We used GNU MPFR (multiple-precision floatin-point computations with correct rounding) library [65] to compute it for 32-bit and 64-bit floating-point numbers. The CFD data are obtained by a CFD which computes the fluid dynamics in 2D Laval nozzle. The data are of a 101×301 orthogonal grid in generalized curvilinear coordinates. This CFD data contain velocity, pressure, tempreture and specific heat value at constant pressure as variables. We also measure the compression ratios by bzip to

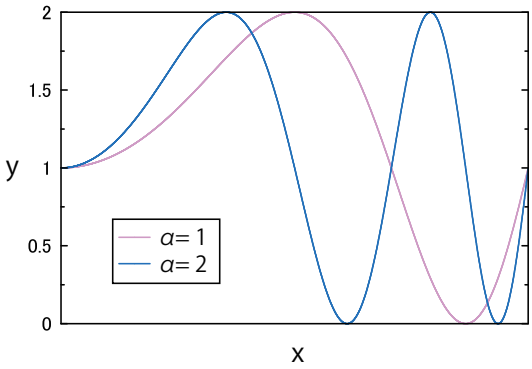


Figure 2.21: Test data generated by sampling a function for $\beta = 1$.

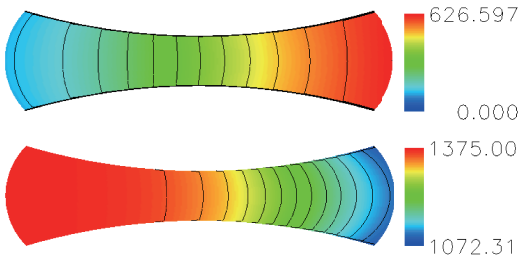


Figure 2.22: CFD result of a 2D Laval nozzle. The color bar shows velocities and pressures.

compare to the hardware compressor.

Fig. 2.23 shows the result. The compression ratios of the test data tend to be higher for smaller α and β, because such parameters make smaller prediction errors between adjacent samples in compressing the test data. As a result, the 32-bit and 64-bit compressions achieve the maximum compression ratio of 3.9 and 5.2, respectively. Since the theoretical maximum compression ratios are 4.57 and 8 for 32 and 64-bit data compression, respectively, this results shows that the compression performs well especially in 32-bit data. On the other hand, the CFD data compression is not effective, which are 3.1 and 1.8 for 32-bit and 64-bit compressions. This is due to the less continuity in the CFD data than the test data because of the curvilinear grid and numerical discontinuity points by shock waves. These ratios are not so high, but our compression technique is still effective in such disadvantageous cases. In all cases, our compressor achieves equal or better ratios than those by the bzip2, a general-purpose software compressor. Since these bit widths are commonly used in numerical computation, the hardware-based data compression is especially useful for bandwidth compression in typical computation with single-precision or double-precision floating-point data streams.

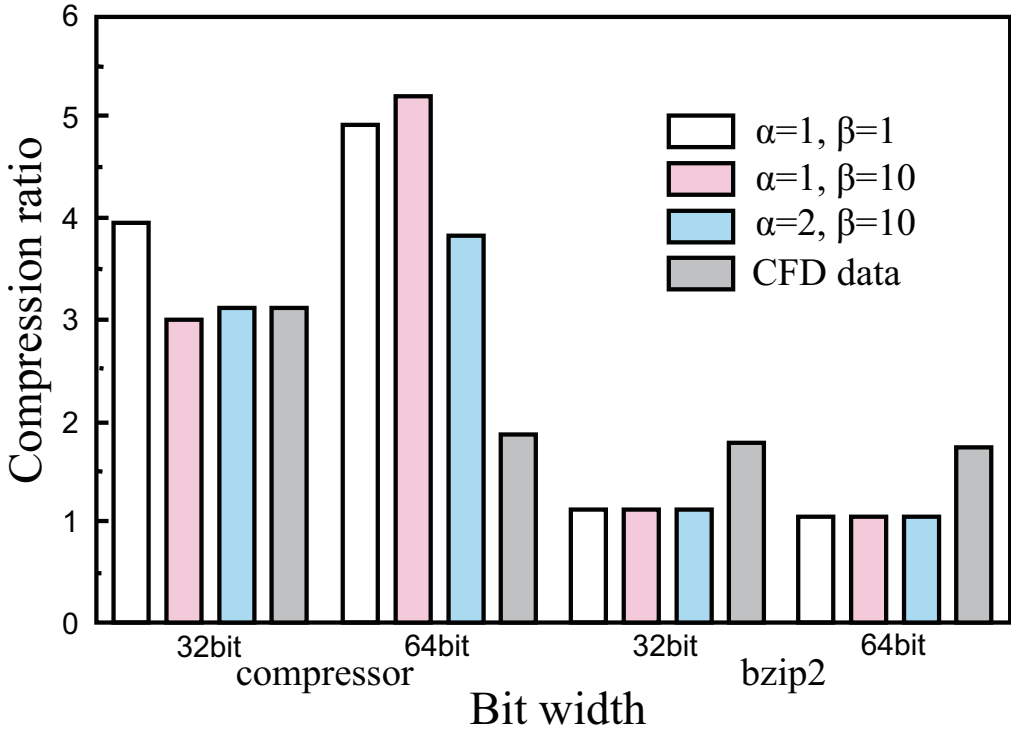


Figure 2.23: Data compression ratios of 32 and 64-bit floating-point data by the compressor and bzip2.

2.7 Problems of the previous design

The last section presents the evaluations of the data compressor and decompressor. The results shows that the throughput is sufficiently high because the compressor and decompressor can operate more than 200 MHz for single-precision 32-bit data. In addition, the compression ratio is also high especially for single-precision data, and is almost the same in performance with bzip2 for double-precision data by more simple algorithm.

On the other hand, in the area evaluation, both the compressor and decompressor require more than 7,000 ALUTs with the 256-bit I/O design. Therefore, a pair of the compressor and decompressor needs about 15,000 ALUTs, which is much more than registers required. Since we need a lossless compression, it is necessary to apply one pair of the compressor and decompressor to a single channel in the FPGA. According to the studies of FPGA-based stream computing [66, 22, 58], the numerical computing is required to process in multiple channels. Therefore, the bandwidth compressor for n channels processing requires $n \times 15000$ ALUTs at least, which limits to apply the compression to computing with many channels.

The number of the channels is determined by computations themselves, especially by the number of variables and the parallelism of the circuit. Since we cannot determine the number of

channels, the bandwidth compressor must be designed specifically for an objective application, including the number of channels. However, the limitation of the design caused by area prevents to apply the bandwidth compressor to various applications. For example, the Stratix IV has 182,400 ALUTs on which we can implement up to twelve pairs of the compressor and decompressor without any modules. The implementation needs modules for computational cores, an encoder and a decoder to handle multiple channels for the numerical computing at least. From this point of view, the areas of the compressor and decompressor are too large to apply to real-world applications on available FPGAs. We will discuss this problem and give the solutions in chapter3.

Moreover, the Compressor and decompressor need to control for multi-channel operations. The real-world applications in this dissertation have multiple variables which correspond to the multiple channels. These variables needs to be calculated synchronously in hardware with multiple channels. On the other hand, a compression ratio of each channel is different and also fluctuating. Therefore, the compression of multiple channels needs to deal with these fluctuating compression ratio and guarantee the synchronized processing on the dedicated circuit. We will give the solutions and show the hardware implementation of the multi-channel compression in chapter4.

2.8 Conclusions

The objective of this chapter is to evaluate the data compressor and decompressor on hardware for applying to the real-world applications. For them, we have described the data compression algorithm which proposed in our preceding work, and presents the hardware designs and the evaluation of the hardware implementation.

The algorithm is based on the prediction of the next input value. Since the floating-point data are not suitable for compressions based on the occurrence frequencies of symbols such as an entropy coding, we employed the algorithm which consists of the prediction and the subtraction. The prediction is carried out by 1D polynomial function, which achieves good performance for the numerical data by exploiting their continuities. The prediction-based compression satisfies the requirements of the bandwidth compression on FPGA, such as small latency and enough compression ratio.

We have also shown the hardware designs of the compressor and decompressor. They are pipelined in 4 and 3 stages, respectively, and consist some modules such as the predictor, VFC, and FVC. The parameterized designs of these modules allow us to apply to various data including single and double precision floating-point data.

By implementing the compressor and decompressor on FPGA, we evaluated the resource utilization, frequency, and compression ratio. The compression ratios in the evaluation are 3.1 and 1.8 for 32 and 64-bit CFD data, respectively. And proposed hardware achieves higher compression ratios than bzip2 which is a general-purpose data compression method. In addition, the maximum operating frequencies are more than 200 MHz in the cases of 32 and 64-bit data compression, which are sufficient to operate on the FPGA. With 32-bit width design, the compressor and decompressor require small area thanks to 4-bit coding.

These results showed that the data compressor for single and double-precision floating-point numbers is very useful for FPGA-based bandwidth compression. On the other hand, the large area of the hardware in the cases of wide data widths had a possibility that we cannot apply the bandwidth compression with a large number of channels. And we also needed to design the hardware for multiple channels including the compressor and decompressor for the real-world applications.

Chapter 3

Area efficient data compressor for stream computing

3.1 Introduction

Chapter2 described a lossless compression algorithm for a floating-point data stream in numerical computing, and we designed high-throughput pipelined compressor and decompressor. The hardware implementation achieves good compression performance and high-throughput processing. However, the hardware area becomes a problem when we try to use the compressor and decompressor to real-world numerical applications.

Since the numerical simulation generally computes complex phenomena based on governing equations with multiple variables, its stream hardware requires multiple channels [57, 67]. In addition, to increase the performance of FPGA-based computing, we need more channels which accommodate multiple cores to exploit coarse-grain parallelism. Since each of the compressor and decompressor can handle only a single channel, we need plenty of the compressor and decompressor modules for FPGA-based high-performance stream computing. For example, 2D CFD with LBM, 2D-LBM, needs 10 variables to compute on each cell [58, 27]. Therefore, the parallel processing of 2D-LBM requires ($parallelism \times 10$) compressors and decompressors, which are the same as the total number of the channels. Accordingly, we need smaller compressors and decompressors to save the area for such auxiliary hardware.

Moreover, there is another factor that a CDB which is an output data block from the compressor needs to be the same width as the output port of FPGA. Recent FPGAs have wide I/O bit widths such as 256 or 512-bit, which causes further area increasing, and also have a large numbers of resources to allow us highly parallel computing. Therefore, we need new method to reduce the area of the compressor and decompressor with little decrease in compression performance.

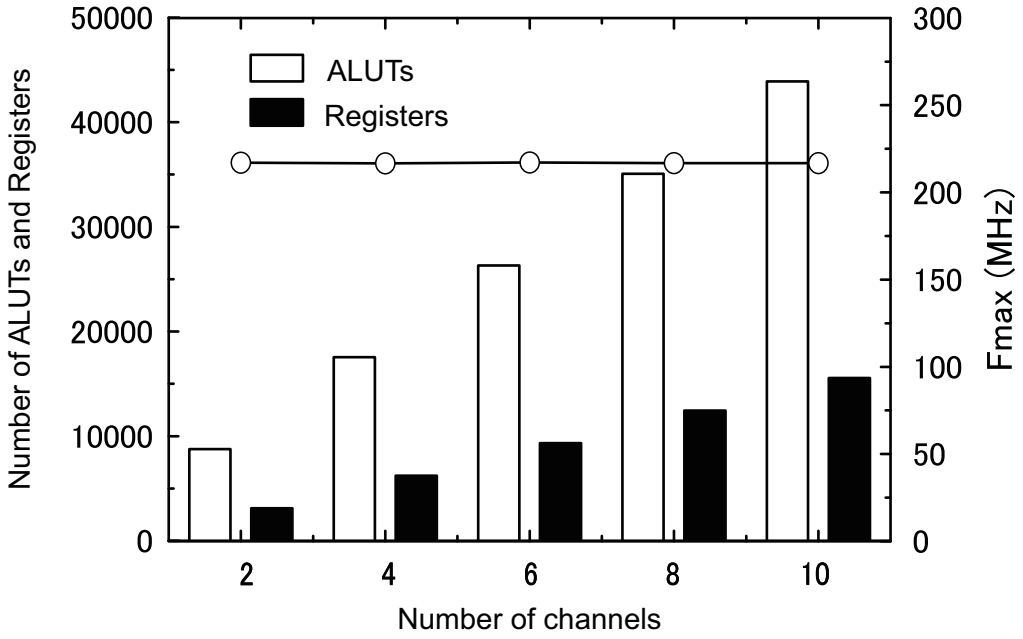


Figure 3.1: Areas and maximum frequencies of the data compressor with various numbers of channels.

To reduce the area of each the compressor and decompressor, we propose an *area-oriented encoding*. In this chapter, we analyze the cause of this large area. This detailed analysis of the hardware points out that the VFC and the FVC are largely responsible for most of the area consumption. The VFC and the FVC are both modules to convert between variable-length compressed data and fixed-length CDBs. Since the compressor generates a fixed-length CDB in contrast, we need to concatenate compressed data to generate CDBs. The VFC gives the concatenation of variable-length compressed data, and the FVC extracts individual compressed data from concatenated data blocks. Since the VFC and FVC are area-consuming, we improve their structure and designs with the newly proposed encoding method. Our approach is based on the idea of 4-bit coding but further simplifies of hardware processing more than the naive 4-bit coding. We propose the limited-LRB (l-LRB) and the improved CDB for area improvement, design new hardware compressor and decompressor, and evaluate their area and compression performance.

This chapter presents a design of very small compressor and decompressor as an area-oriented design which are realized a drastic reduction of the hardware area with a slight decrease of the compression performance. It shows the ideas and specifications of the l-LRB and an improved CDB for reduction of the area of VFC and FVC. Along with this, we show the area-oriented algorithm and design, which simplifies the operations and limits the range of possible values of LRB. We also show an estimation of the compression ratio with modeled LRB distributions which represent the characteristics of the various data. The evaluation shows that the compressors

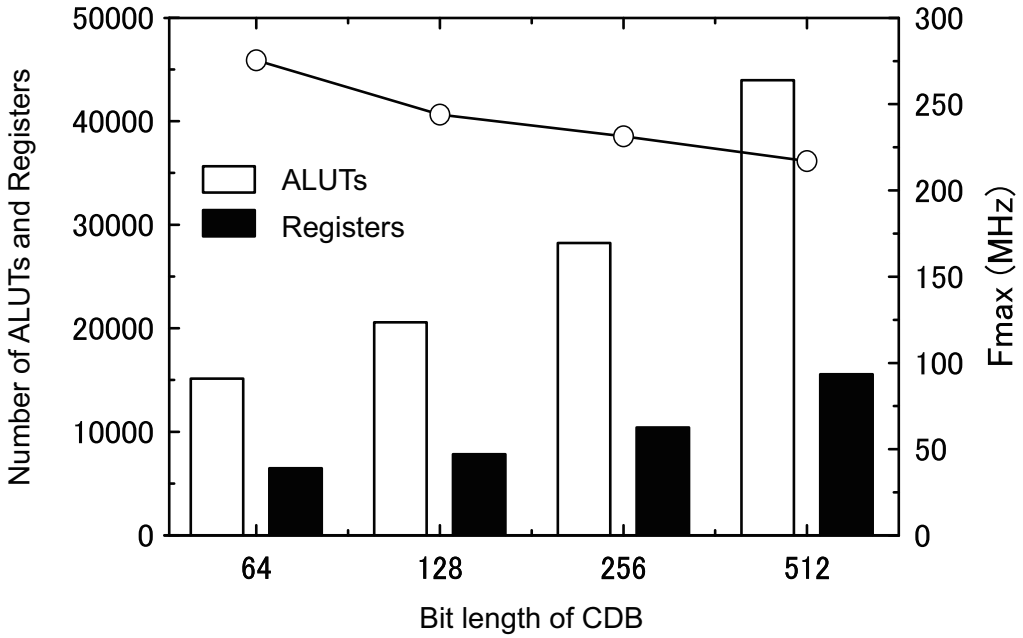


Figure 3.2: Areas and maximum frequencies of the data compressor with various lengths of CDBs.

and decompressors are sufficiently small to apply to dozens channels like real-world numerical applications.

The objective of this chapter is proposing new encoding method for area saving to apply to the real-world applications which requires multiple channels. For this goal, we present the detailed analysis of the area of the compressor and decompressor. And we shows the new encoding method, design the improved compressor and decompressor based on the new encoding, and evaluate the proposed hardware for their area and compression performance. The contributions of this chapter are:

1. Detailed analysis of the previous hardware,
2. Proposing new encoding method,
3. Designing hardware based on new encoding,
4. Evaluation of proposed compressor and decompressor.

The organization of the chapter is as follows. Section 3.2 describes the causes of the large area. Section 3.3 proposes the new encoding method. Section 3.4 shows the design of the improved compressor and decompressor. Section 3.5 presents evaluations. In section 3.6, we discuss about a selection of design for various applications. Finally section 3.7 gives conclusions of this chapter.

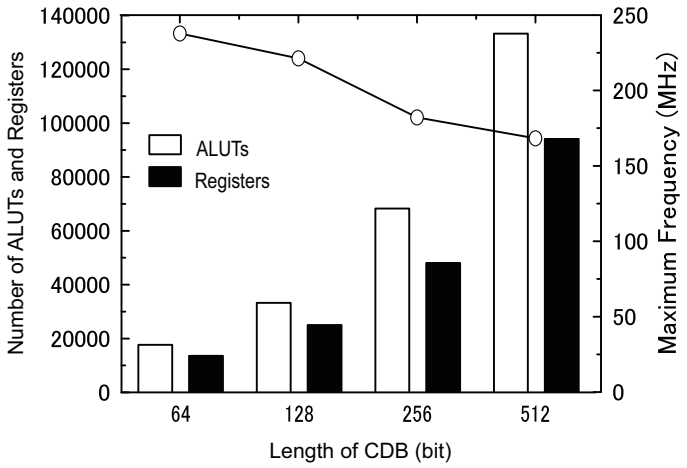


Figure 3.3: Area and Frequency of decompressor.

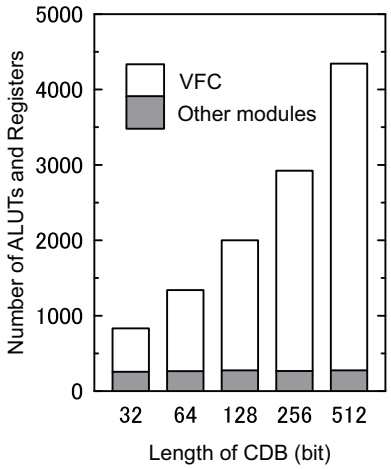


Figure 3.4: Breakdown of resource usage of a compressor

3.2 Causes of the large area problem

In the previous chapter, . the evaluation results are good for the compression performance and the throughput, however, the area becomes a problem when we apply the data compression for real-world applications. Fig. 3.1 shows the area and maximum frequency of the data compression hardware for various numbers of the channels when the length of CDB is 512 bits. The area becomes bigger as the number of channels increases. On the other hand, the maximum frequency stays almost constant.

Fig. 3.2 also shows the area and maximum frequencies of the compressor for various lengths of CDBs when the number of channels is 10. In contrast to Fig. 3.1, the area increases and the maximum operating frequency decreases as the output block size increases.

These results show that the maximum operating frequency is determined mainly by the bit width, not by the number of channels. The usage of ALUTs is much bigger than registers, and the compressors uses about 25% of the total ALUTs in Stratix IV when the number of channels is 10 and the output block size is 512. Fig. 3.4 shows the breakdown of ALUT usages by VFCs and others for various output block size. As the length of CDB increases, the amount of ALUTs for the compressor also increases due to the growing VFC. In contrast, the other parts of the compressor do not increase the area significantly. These results show us that the length of CDB is the major factor in determining area of the compressor.

Fig. 3.3 shows an evaluation result of the decompressor for 10 channels with various CDB sizes. The result denotes the same tendency of the compressor, however, we found that the area of the decompressor is significantly larger than the compressor. This is because that the decompressor contains two barrel shifters which are used for variable length shifts while the compressor contains only one barrel shifter.

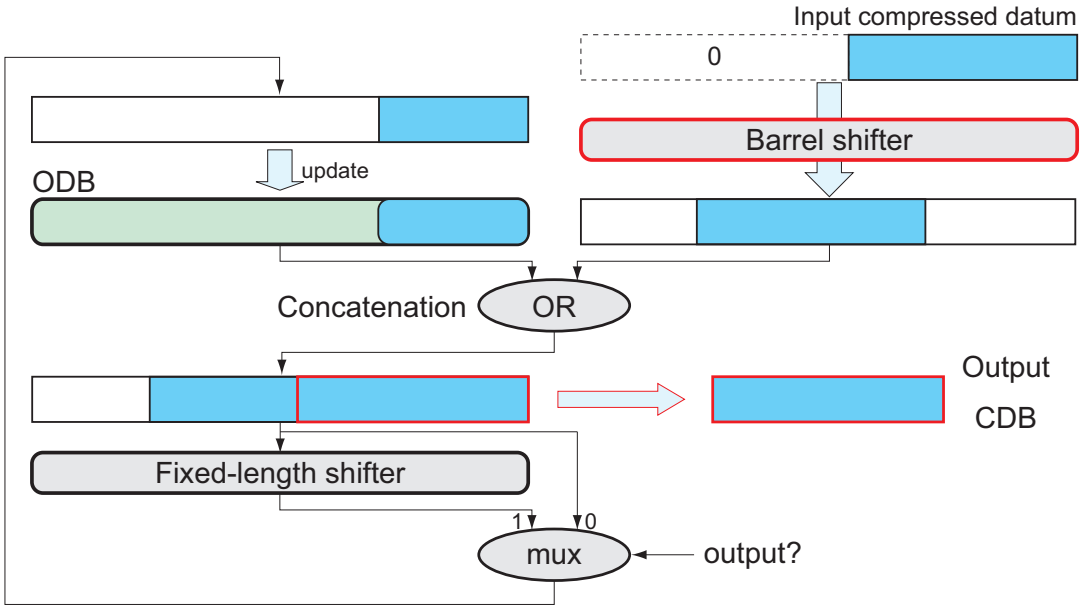


Figure 3.5: Generating CDB by concatenating compressed data.

3.2.1 Impact of the number of channels on area

The numerical computing requires multiple compressors and decompressors for the bandwidth compression of multiple channels. Therefore, the resource consumption of the compressors and decompressors increase in proportion to the number of channels as shown in Fig. 3.1. In the previous design of the preceding study, the number of ALUTs required per channel is about 10,000 with 512-bit CDBs, which allow us to implement only 18 pairs of the compressor and decompressor on Stratix IV FPGA. In terms of application to real-world problems, the hardware area of the previous design is too large to make sufficient room for computing hardware to be implemented with on state-of-art FPGAs. Therefore, the area of the compressor and decompressor should be as small as possible.

Additionally, since the bandwidth compression improves the performance of applications which requires wider bandwidth than available bandwidth, the bandwidth compressor is required to be sufficiently small as compared with the dedicated numerical circuits. For example, when the I/O width of the FPGA is w_{out} bits and the data width is w bits, the bandwidth compression needs more than $\frac{w_{out}}{w}$ channels to produce an effect. Therefore, the bandwidth compression consistently improves a marginal performance of the FPGA-based stream computing rather than increases a general performance. Hence the more channels the dedicated circuit has, the more effective the bandwidth compression becomes. Since we cannot choose the number of channels freely, it is impossible to reduce the area by the reduction of channels. To reduce the area of the bandwidth compressor, we have no choice but to reduce the area of the compressor

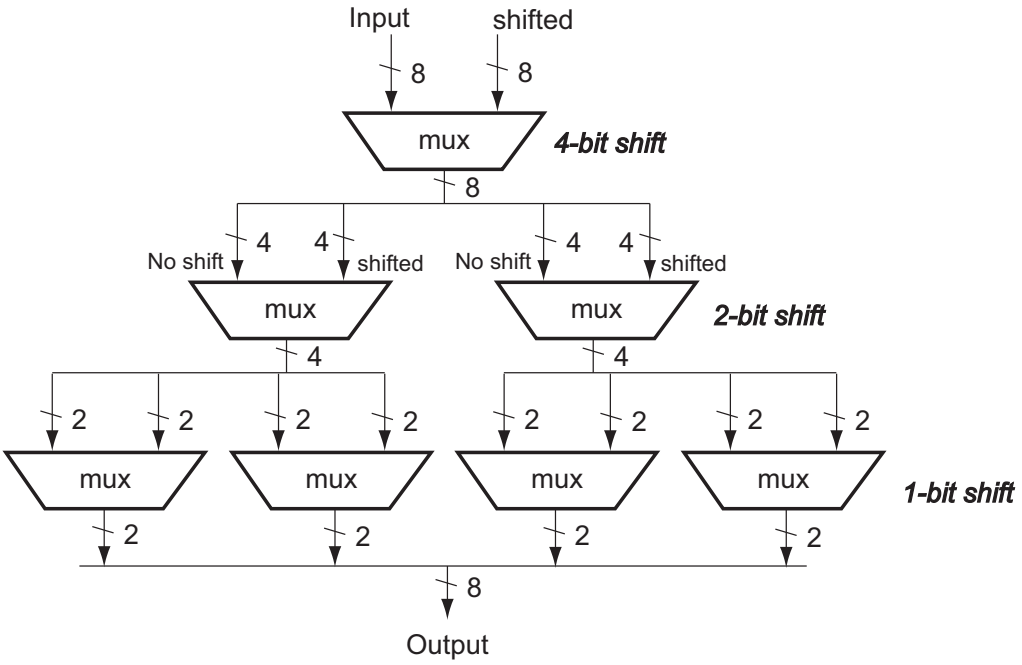


Figure 3.6: Structural overview of an 8-bit barrel shifter.

and decompressor themselves.

3.2.2 Impact of the length of CDB on area

Figs. 3.2 and 3.4 show that the number of ALUTs increases with the length of CDB mainly due to the increasing ALUTs of the VFC. On the other hand, the cause of the increase of used ALUTs is the VFC as is clear from Fig. . In the case of the VFC in Fig. 2.11 with 32-bit CDB, it contains a 68-bit buffer as an output data buffer (ODB), which concatenates $\{r, \mathrm{LRB}, \mathrm{ex}\}$ of the successive compressed data. The length of ODB is more than twice of the length of the CDB. Since the VFC concatenates compressed data as a bit string and isolates a CDB from the bit string in ODB as shown in Fig. 3.5, the ODB has to have enough bit width so that any bits are not lost. In the case of the 32-bit CDB, the ODB needs 68 bits so that it always holds the compressed data which have up to 36 bits. Therefore, with 512-bit CDBs, the ODB needs to have 550 bits at least, such along buffer requires a huge barrel shifter.

A barrel shifter makes an arbitrary shift by combining 2^n-bit shifts. The structural overview of an 8-bit input barrel shifter is shown in Fig. 3.6. For 8-bit input, the barrel shifter contains a three stages which correspond different shift amounts from each other, 4-bit, 2-bit, and 1-bit shift, respectively. It can achieve 1 to 8-bit shifts by combining of 4, 2, and 1-bit shifts. Since every stage of the barrel shifter requires 8 multiplexors, there are 24 multiplexors in the 8-bit input barrel shifter. Therefore, the number of multiplexors for an n-bit shifter is of $O(nlog_2n)$. Fig. 3.7 shows the increasing number of multiplexors in a n-bit barrel shifter. As shown in the

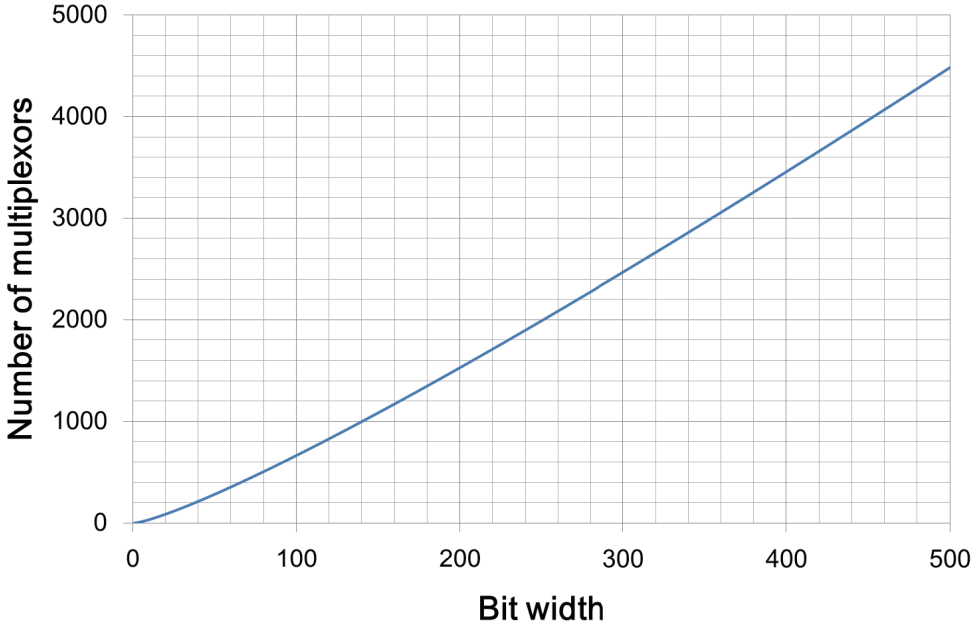


Figure 3.7: Hardware cost of a barrel shifter.

graph, we need much more multiplexors in the barrel shifter according to increase of n. According to this theory, a 550-bit barrel shifter needs 10240 multiplexors. Actually, an optimization in a compiler reduces the resource usage of the barrel shifter, which exploits that the input is 38 bits at a maximum. Even so, a wide barrel shifter still requires a large amount resources.

The VFC contains one barrel shifter while the FVC contains two barrel shifters. Since the FVC has two barrel shifters for IDB update and shift input data, the FVC needs more resources than the VFC. Therefore, to reduce the hardware area of the compressor and decompressor, it is necessary to make the barrel shifters smaller. To make the barrel shifters smaller, we can rely on either splitting the operation or reducing bit width. However, the barrel shifters cannot be pipelined to divide the process because the VFC and FVC contain feedback loops. Moreover, we cannot divide the available bandwidth by a partitioning CDB in each channel because of fluctuations of the compression ratios in channels.

Thus the encoding method and the structure of the previous design .are not suitable for small implementation To solve the problem, we need to improve VFC and FVC especially. We propose the solution of this problem in the next section.

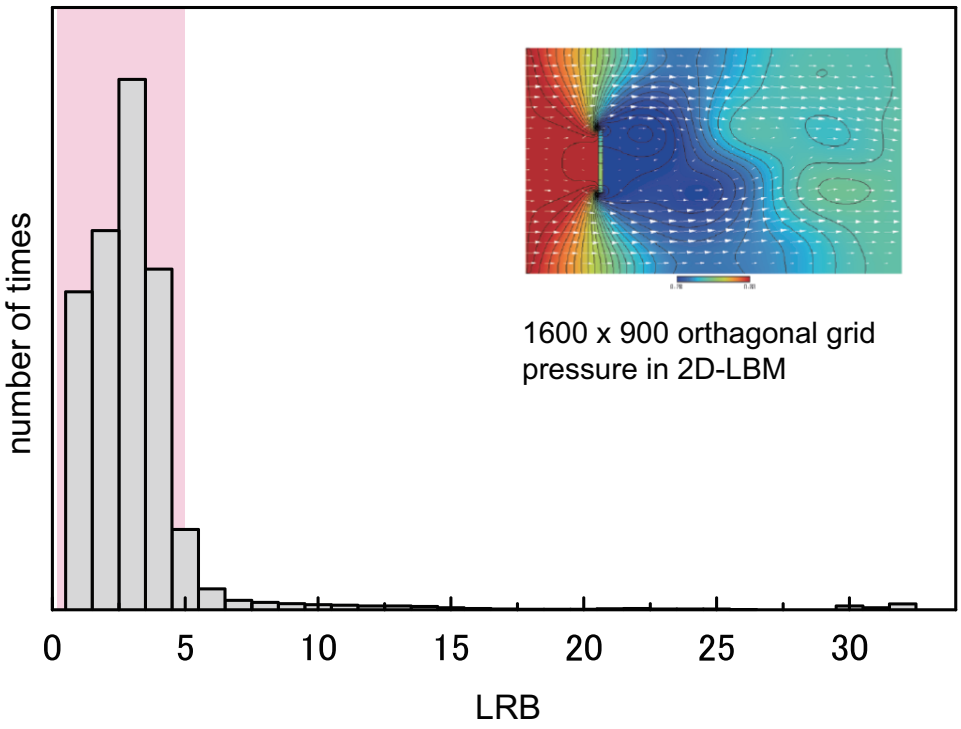


Figure 3.8: LRB distribution of 2D LBM data.

3.3 Area-oriented encoding

Since the previous design requires too many resources to apply to the real-world applications, we need to reduce the area of the compressor and decompressor drastically. Several researches proposed *n-bit encoding* which handles the residual bits in units of n bits to simplify the process [42, 60]. Our preceding study also proposed the naive 4-bit encoding in order to easily handle variable-length compressed bits [68]. The n-bit encoding limits the length of residual bits to be in multiples of n. Our preceding use $n = 4$ for compression of only a single precision floating-point data which allows the compressed bit to be of multiples of 4 bits. By limiting the length of the residual bits to $4n$, LRB is expressed in 3 bits. With one more additional bit for ex, the set of the residual bits, LRB, and ex fits the 4-bit alignment. We generalize and further simplify this approach for other lengths of an input datum to reduce the area of the compressor and decompressor.

The idea of the bandwidth compression means that we accelerate the computing at the expense of an extra hardware resource. In terms of the compression performance, it also obtains good performances by using large hardware resources generally. Therefore, it is believed that the reduction of area involves a decline of the compression performance. So we should employ

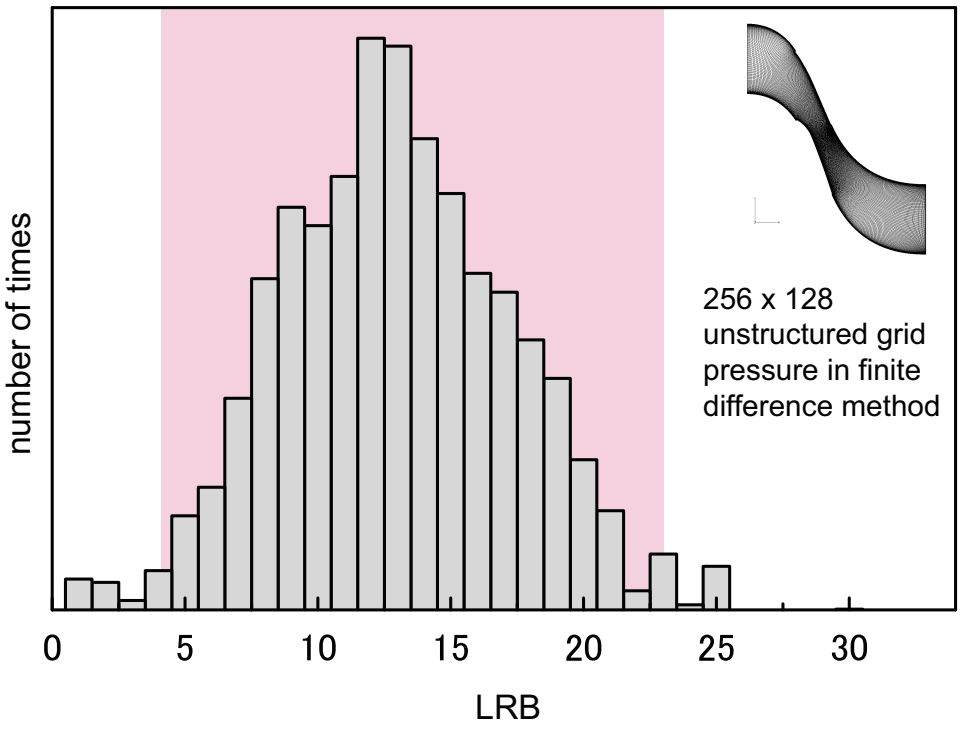


Figure 3.9: LRB distribution of numerical data with an unstructured grid.

an area reduction method which does not decrease the compression performance too much. Following subsections show area reduction methods which are based on the idea of n-bit encoding and exploiting deviations of the numerical data to reduce the degradation of compression performance.

3.3.1 Uneven distribution of LRB in numerical data

We have found that compression of typical data gives uneven distribution of residual length, which exist in a certain ranges. The length of residual bits, LRB, indicates the prediction accuracy in the compression, which depends on the characteristic of data. Generally, the the characteristic of data is influenced by computation itself and computational grid.

Figs. 3.8 and 3.9 show examples of LRB distribution. In these graph, the smaller LRBs show that the prediction accuracy is higher. In the Fig. 3.8, a computational grid is large and orthogonal, which allows the prediction more accurate because the data get closer to constant. On the other hand, in Fig. 3.9, since a computational grid is small, and containing rapid change of physical quantities given by such as a shock wave, the distribution of the LRB becomes bigger than that of Fig. 3.8 because of inaccuracy of the prediction. The LRB distributions are different depending the continuity of the data, however, in both cases, LRBs are clustered in certain regions.

Therefore, we should exploit these deviations to keep the compression performance with the

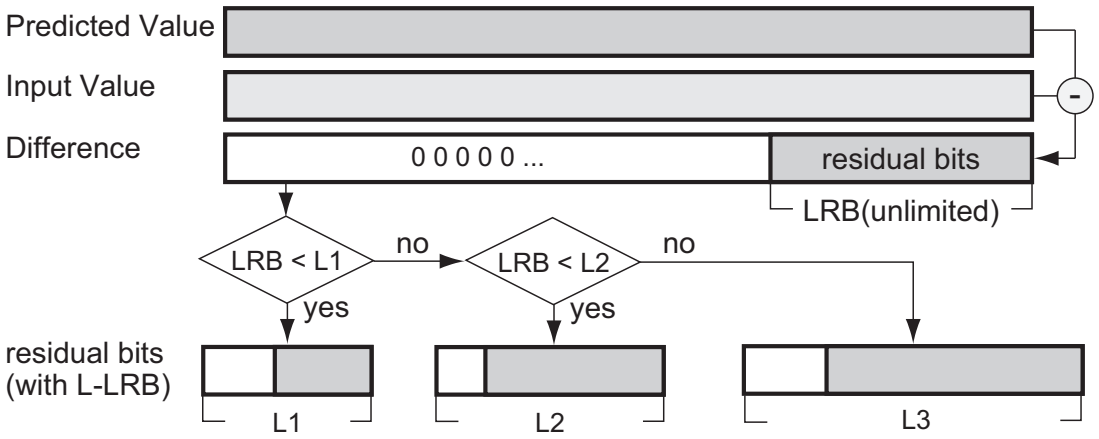


Figure 3.10: Limited LRB for area efficiency.

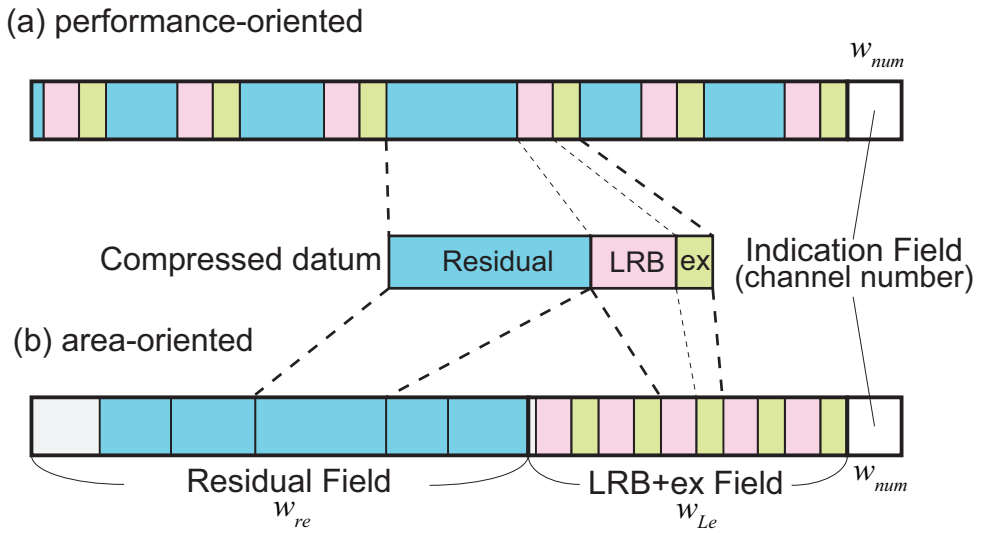


Figure 3.11: Structures of compressed data block (CDB).

reduction of the areas. The uneven distribution of LRB means that we can reduce a complexity of the encoding because there are some regions both in Figs. 3.9 and 3.8 where the compression does not affect because the LRBs are almost non-existent. On the other hand, we focus to compress effectively the regions in which the LRB is many such as the colored area in Figs. 3.8 and 3.9. With this concept, we propose for an area-efficient encoding method which limits the possible values of LRB.

3.3.2 Limited LRB

First, instead of a variable bit length, we use only three lengths to express the residual length to make the bit string of LRB shorter. We refer to these selected lengths as *limited LRB (L-LRB)*.

We make the three lengths in multiple of 2 like 2, 4, and 32. Let L1, L2, and L3 denote the three lengths that $L1 \leq L2 \leq L3$. Fig.3.10 shows the detail of L-LRB. We make L3 the same length as the original data to keep the entire bits without lossy compression when $LRB \geq L2$. When $L1 \leq LRB \leq L2$, we keep L2 bits including the original residual and padded zeros. Or when $LRB \leq L1$, we keep L1 bits including padded zeros. This L-LRB approach gives additional zeros in stored data, however requires only 2 bits to encode the three lengths, resulting in much simplified hardware.

The L-LRB allows us to employ much smaller barrel shifters than the previous encoding. We select $L1$, $L2$, $L3$ to have the big greatest common divisor (GCD) for an effective area reduction. In this case, since the GCD works as n in n-bit encoding, the lengths of the bits in the ODB are always multiple of the GCD. Therefore, we can handle the ODB with small barrel shifters because the shift amount for the CDB generation always becomes the multiples of the GCD. When we denote the GCD of $L1$, $L2$, $L3$ with w_{GCD}, the number of required multiplexors in the barrel shifter becomes w_{GCD} times smaller than the original.

3.3.3 Improved CDB

Second, we split the fields of CDBs into the field of *L-LRB* and *ex*, and the field of the residual bits as shown in Fig. 3.11(b). Fig. 3.11(a) is an original method which is referred to as the performance-oriented encoding thereafter. This is because the L-LRB and ex have a fixed bit-length of 3, where 2 for L-LRB and 1 for ex. Since sets of LRB and ex are at a fixed location, simple hardware with multiplexors can be used to extract one of them. On the other hand, the residual field is occupied with multiples of the greatest common divisor (GCD) of the three lengths. For example, the number of bits is multiple of 2 for lengths of 2, 4, and 32.

Regarding the hardware, this improvement can reduce the amount of its resources dramatically, as described in the next section. We employ a barrel shifter to generate CDB with compressed bits, which can shift input data for any shift amounts in one clock cycle. Barrel shifters require many multiplexers when the shift amounts can take the wide range of values. The number of multiplexors is now given by $(w_s \log_2 w_s)$, where w_s is the width of data to be shifted. For example, the I/O data width of FPGAs is 512 bits which causes that the CDB has 512-bit length to exploit physical bandwidth. Therefore, the range of possible shift amounts is from 1 to 512. On the other hand, in this reformation, the number of multiplexers decreases to $1/d$, where d is GCD of the selected values. So the reformation reduces this range of possible shift amounts. As a result, it also reduces the usage of multiplexers. We call this area effective approach as *area-oriented* encoding method.

The area-oriented encoding method is generalized with parameters as follows. The size of each field is optimized depending on the combination of the selected residual lengths. Let w_{out}

denotes the bit width of the CDB, and w_I, w_{Le}, and w_{re} denote the widths of the fields of indication, *L-LRB* with *ex*, and residual, respectively. The indication field contains a channel number to indicate the channel in which the CDB is generated. These parameters have to satisfy the following conditions.

$$w_I + w_{Le} + w_{re} = W. \tag{3.1}$$

$$(L_{L-LRB} + L_{ex} + L_{Rmin})N_{comp} < W - w_n. \tag{3.2}$$

The L_{L-LRB}, L_{ex}, and L_{Rmin} are the bit length of the *LRB*, *ex*, and the residual bits, respectively. N_{comp} is the number of the compressed data that can exist in CDB. We set the maximum N_{comp} under the above conditions. In the case that $w_{out} = 512, \quad w_I = 5$, $L_{L-LRB} = 2$, $L_{ex} = 1$, and $L_{Rmin} = 2$, we can define that $w_{Le} = 285$, $w_{re} = 222$ and $N_{comp} = 95$.

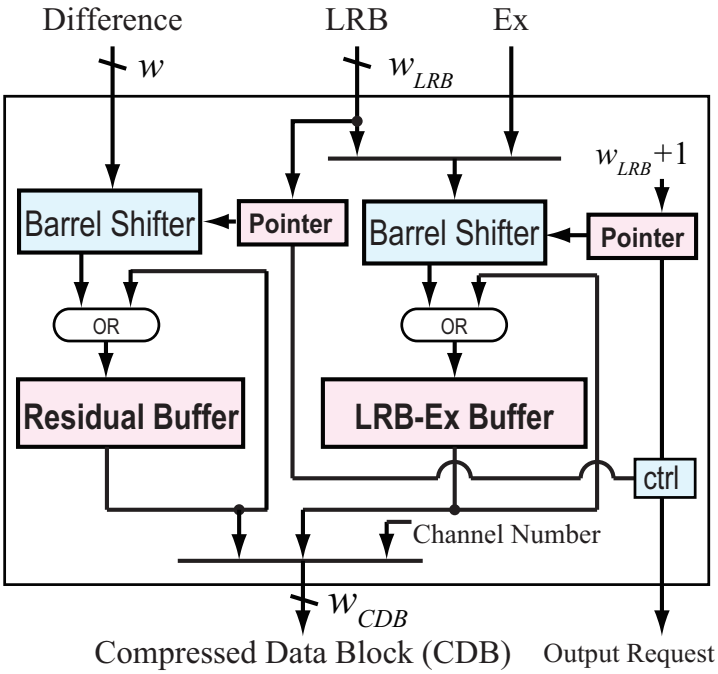


Figure 3.12: Variable to fixed length converter(VFC).

3.4 Hardware design of area-oriented data compression

The area-oriented encoding reduces the area by avoiding inefficient coding and simplifying the hardware operation. Since VFC and FVC occupy most of the hardware resources of the compressor and decompressor, we focus on improvement of VFC and FVC. Especially, we try to reduce the hardware of shift operation, which requires huge barrel shifters to shift according to an accumulating buffer to concatenate in the original design. With the proposed encoding method, we improve them by removing the barrel shifters from FVC, and dividing and reducing the size of the barrel shifter in VFC.

3.4.1 Improved VFC

VFC contains a buffer for accumulating of variable-length bit strings with a large barrel shifter to generate CDBs. Fig. 3.12 shows the area efficient design of the improved VFC. The area-oriented design has two smaller barrel shifters for the residual and LRB-Ex in the CDB instead of one large barrel shifter shown in Fig. 2.11. Both accumulation buffers, for *residual* and *LRB-Ex*, are controlled by the buffer pointers and these barrel shifters. The pointers used for input and output controls to indicate the position of the next coming bit-string in the buffer.

The two buffers operate in the same way. Firstly, the input bit strings are shifted according

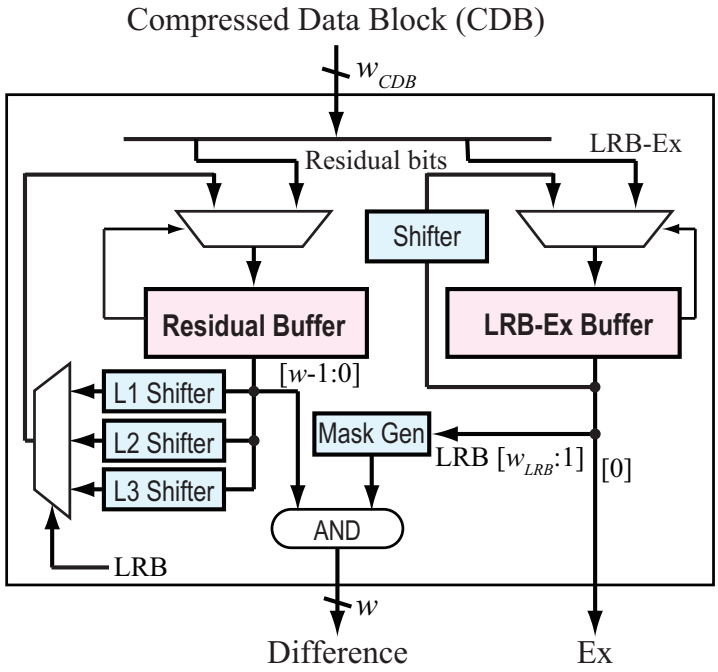


Figure 3.13: Fixed to variable length converter(FVC).

to the pointer's values. The shifted bit strings are concatenated with the existing bits in the accumulation buffers by OR operation. The length of each buffer is the same with the length of the corresponding field in CDB. Therefore, the output CDB is the concatenated bit strings from these buffers with the channel number. If the pointer shows that the buffer cannot accept any more inputs, the VFC asserts an output request signal to the following bundler, at the same time, it stops accepting input until the buffer becomes free. VFC actually outputs CDB when it receives an output request signal from the down stream.

3.4.2 Improved FVC

Fig.3.13 shows the structure of FVC. FVC extracts the *ex* and *difference* from the received CDB. It has also the two buffers to store CDBs, one is for the *ex* and *LRB*, and the other is for the residual bits. For the naive coding method, the original FVC needs two barrel shifters to update the buffers and input control. On the other hand, the proposed encoding method does not require the barrel shifters because requiring a small number of fixed shifters.

The improved FVC has two buffers which are identical with those of the improved VFC. The three bits from LSB of LRB-Ex buffer are always *ex* and *LRB* for the next reconstruct datum. *Ex* is output directly, and *LRB* is used to a reconstructing *difference*. With the *LRB* information, FVC generates masked bits to reconstruct the *difference*. The masked bits are used to extract w bits from the LSB of the residual buffer, w is the bit length of original data, then it generates the *difference* by AND operation. Since the length of residual bits is limited to

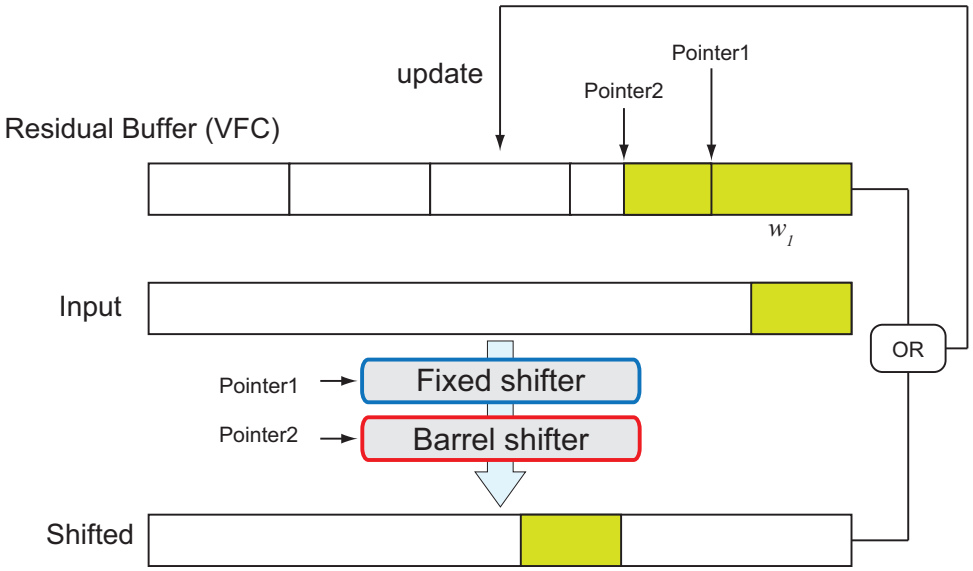


Figure 3.14: Improvement of the operation in VFC for reduction of barrel shifter.

the selected three values, FVC needs only three fixed shifters to update of the residual buffer. Then FVC selects a correct shifted bit string by *LRB* value. The improved FVC does not have pointers to indicate the bit amounts of the buffers because *LRB* can show the end of CDBs directly. As mentioned before, although two bits *LRB* can express four different values of the residual length, one of them is used for the control, which describes the end of CDB. If the next *LRB* is 00, FVC allows to input because there is no data in current buffers. Then if the next *LRB* is not 00, FVC allows to output and disallows to input.

3.4.3 Simplification of the processing in VFC and FVC

We make another effort to reduce the area of the improved VFC and FVC. In the improved designs, we have only two medium barrel shifters in the improved VFC and no barrel shifter in the improved FVC. As described in section 3.2, the number of multiplexors in the barrel shifter is represented as $w \log_2 w_{buff}$, where w is the data word of the computing and w_{buff} is the length of the buffer. As this formula shows, the barrel shifter becomes smaller when we divide the buffer and handle each divided region separately. Fig. 3.14 shows the improved buffer operation. We divide the buffer into multiple regions, each of which length is w_1, and we employ the two pointers to indicate the amount of bits in the buffer. It is necessary that w_1 is larger than *residual bits* so as to adopt a fixed-length shifter. With this improvement, the barrel shifter in the VFC becomes smaller by taking over a part of the shift operation with the fixed shifters.

3.5 Evaluation

In this chapter, we proposed the area-oriented encoding method for the data compression according to various cases. We describe the previous original method as performance-oriented after this. We expect that the performance-oriented encoding achieves better compression performance with larger area than the area oriented. On the other hand, the area-oriented may also reduce the compression performance. Therefore, we investigate the trade-off between the compression performance and the area in a quantitative way. Since the performance of data compression depends on data themselves, evaluation only with some specific numerical data sets is insufficient. To understand a tendency of the compression performance for various data sets, we model the LRB distribution of a numerical data set and investigate the relationship between compression performance and the statistical features of the data set.

We focus on the encoding method of data compression in terms of hardware area and compression performance. In this evaluation, we compare the performance-oriented and the area-oriented designs to show which is more suitable to be implemented for real-world applications with FPGA. We show that the proposed area-oriented encoding method can obviously reduce the area compared to the performance-oriented. Furthermore, we compare the compression performances to evaluate a trade-off between the area and the compression performance.

Since it is difficult to evaluate the compression performance qualitatively by experiential evaluations with practical numerical data sets, we obtain the compression performance by a distribution model of residual lengths which is represented by canonical distribution. To model the LRB distributions, we employ a canonical distribution where probability density is clustered as the examples above.

We use the data modeled with the canonical distribution of LRBs as is shown in Fig. 3.15 (a) so as to estimate compression performance with various *L-LRBs*. A probability density function of canonical distribution is given as follows:

$$\frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2}),$$

where σ is an expectation, and μ is a standard deviation. We estimate compression ratios for the performance-oriented method and area-oriented encoding methods with three combinations of L-LRBs, (2, 4, 32), (4, 8, 32), and (8, 16, 32). We use of single-precision floating-point data, that have the modeled distribution of LRBs These selected sets of values correspond to 2-bit, 4-bit, and 8-bit encoding which are the GCDs of their values respectively.

3.5.1 Compression ratio

Figs. 3.15 (b), (c), and (d) show the compression performances of the proposed encoding method when μ is 1, 3, and 5. The x-axis shows expectations, σ, and the y-axis shows compression ra-

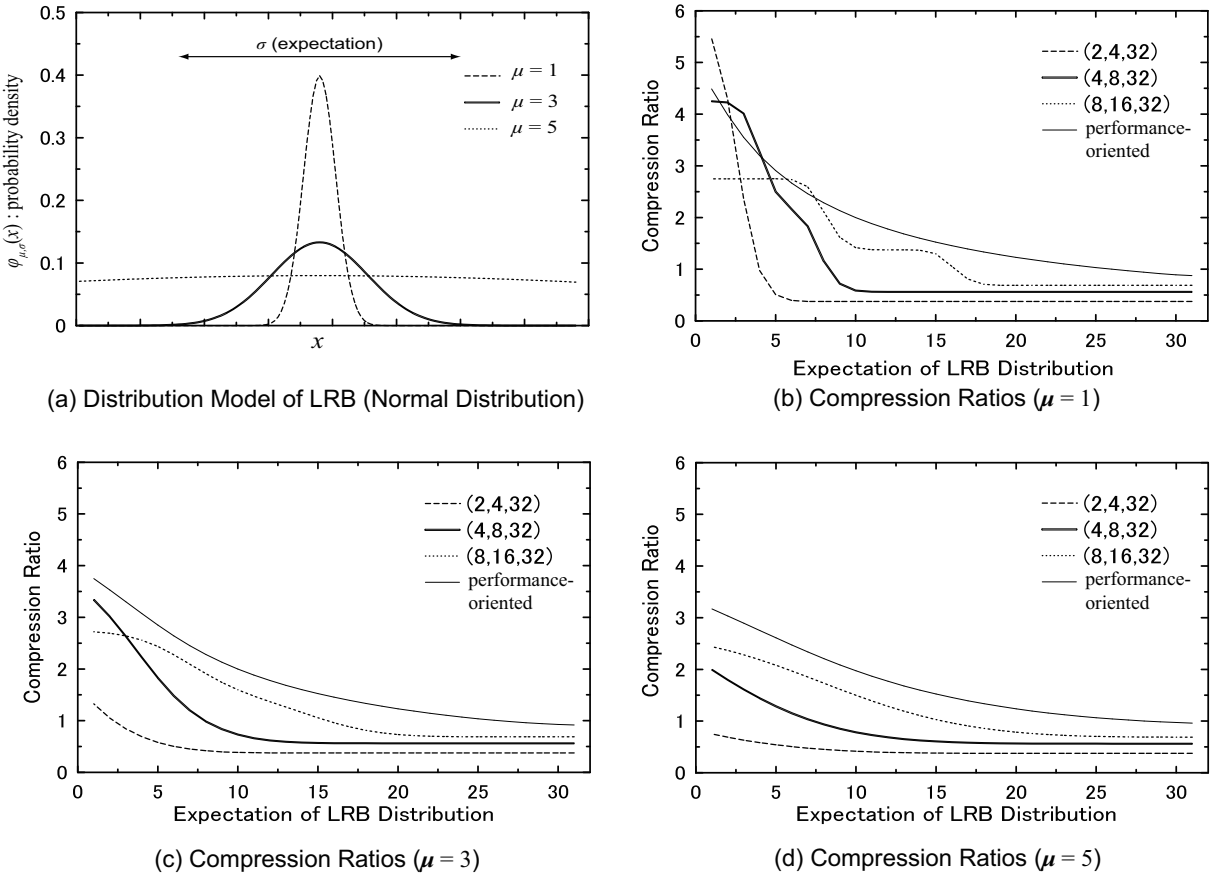


Figure 3.15: Evaluation of Area Efficient Compressions by LRB Distribution Model.

tios. As a matter of course, smaller LRBs raise the compression ratio, and the area-oriented design with L-LRB encoding is effective with smaller μ when the LRB distribution has a higher deviation. In every case, the performance-oriented encoding method achieves the best compression performance overall. However, the area-oriented encodings achieves higher compression ratio in several cases when the encoding exploits the deviation adequately. In conclusion, the area-oriented encoding method is effective for distribution with strong deviation. Moreover, an effective range of the compression decreases as smaller L-LRBs are selected.

Fig. 3.16 shows the compression ratios for real-world numerical data as shown in Figs. 3.8 and 3.9. The 2D-LBM data are computed using an orthogonal grid with a high resolution. In contrast, the nozzle data are obtained by using an unconstructed grid with a low resolution. Therefore, their LRB distributions have completely different characteristics. Accordingly, the graph also shows the huge difference as the compression performances. The 2D-LBM data are compressed very well in all cases. Especially, (2,4,32) and (4,8,32) shows better compression performances than the performance-oriented encoding method. On the other hand, for the nozzle data on the course grid, encoding with (8,16,32) gives a slightly higher compression ratio. This

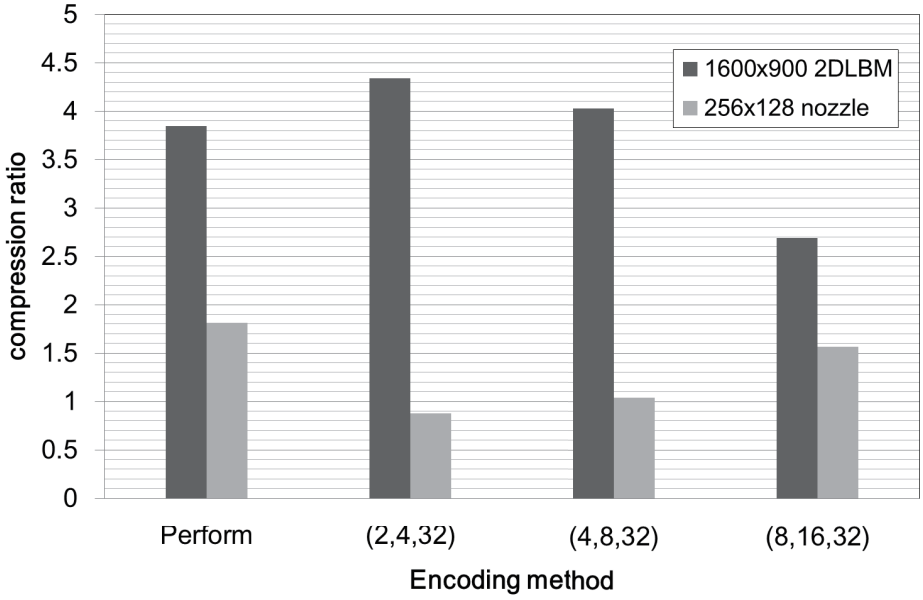


Figure 3.16: Compression ratios for real-world numerical data.

evaluation shows that we should select the encoding method of the data compression depending on the objective data.

3.5.2 Hardware area

We also evaluate the hardware area of the compressor and the decompressor for a single channel with the several L-LRBs of parameterized implementations [69]. Fig. 3.17 shows the hardware area in usages of logic elements, shown as ALUTs (adaptive look up table) which are logic units on Altera's FPGA, and registers. The results of the area-oriented designs show that they require much smaller area than the performance-oriented design. For the number of registers, every case of the area-oriented design shows the unchanged result because the number of registers is not affected by the length of CDB. On the other hand, the numbers of logic elements in the compressors have different characteristics from our expectation where it is determined only by the greatest common divisor of selected values. The largest area among the three cases is required for (4, 8, 32), while we expected that (2, 4, 32) would require the largest area because its GCD is the smallest of the three. The results of the decompressors show that the area-oriented designs dramatically reduce the area, because there are no barrel shifter in the area-oriented designs compared to the performance-oriented which has two barrel shifters.

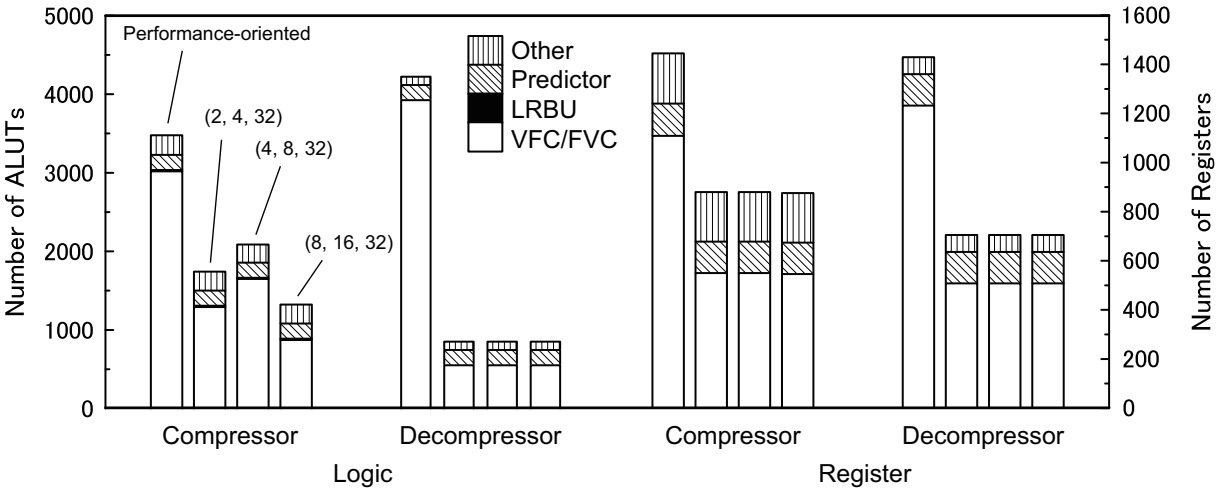


Figure 3.17: Hardware Area of Single Compressor and Decompressor.

3.5.3 Area efficiency for compression performance

To evaluate area-performance efficiency, we calculate a compression ratios per resource usages. The evaluation investigates compression ratios per one ALUT and one register with $\mu = 1$ and 5. Although this is not a quantitative and absolute indicator because it can not always be an accurate evaluation when the compression ratio is less than 1, it can be used for relative comparison under the equal condition.

Figs. 3.18 and 3.19 show the area efficiency of encoding cases of L-LRB and performance-oriented for ALUTs. "Perform" in the legend shows the result of the performance-oriented encoding method. Both graphs shows that the encoding with (8,16,32) achieves the highest area efficiency in most cases. In addition, with an extremely high prediction accuracy, the encoding with (2,4,32) and (4,8,32) achieves high area efficiencies in the case that $\mu = 1$. According this results, we should employ the area-oriented encoding with (8,16,32) as L-LRB when we emphasize the efficiency of ALUT consumption. Figs. 3.20 and 3.21 show the area efficiency for registers. Similar to the results for ALUTs, they show that the encoding with (8,16,32) achieves the highest area efficiency.

Based on these results and discussions, we decide to employ (8, 16, 32) the area-oriented design with for FPGA-based implementation and demonstration, because it has the smallest area and relatively good compression performance for a wide range of LRB distribution.

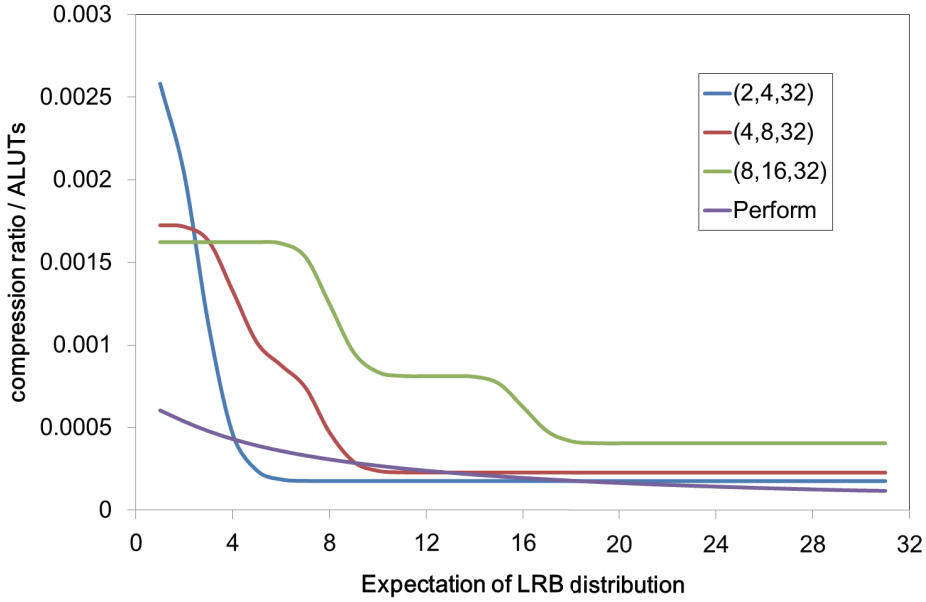


Figure 3.18: Area efficiency by compression ratio per ALUTs ($\mu = 1$).

3.6 Discussion for Optimal L-LRB Selection

This section provides criterions for selecting an optimal L-LRB in accordance with the target applications and implementation environments. In the area-oriented design we must select L-LRB for lengths of residuals. A selection of L-LRB is initially for reducing area of the bandwidth compressor, moreover, it also exploits the deviation of the distribution of prediction accuracies so that the improved encoding prevent a significant drop in performance. However, since this deviation is data-dependent, it needs some statistical information of the target data to determine the selection of suitable L-LRB in advance.

To provide users a guidance for LLRB selection according to the target data, this section evaluates and discusses distributions of compression performance by using the distribution model of prediction accuracies in section 3.5. It also shows estimations of the areas for selected L-LRBs. Since there are some constraints for FPGA applications due to the environment such as required compression performance and limitation of available resource, compression users select an optimal design for the target by reference to the data given in this section.

Firstly, we shows area estimations of the compressor and decompressor. The L-LRB has different impacts on the area of the compressor and decompressor. This method reduces the area by reducing barrel shifters in the compressor. On the other hand, the decompressor with L-LRB has multiple fixed shifters instead of two large barrel shifters.

Fig. 3.22 shows the numbers of required multiplexors for the barrel shifter of the residual field shown in Fig. 3.11(b) in the compressor. The resource usage of the compressor depends heavily on the greatest common divisor (GCD) of the selected L-LRB because the shift amounts

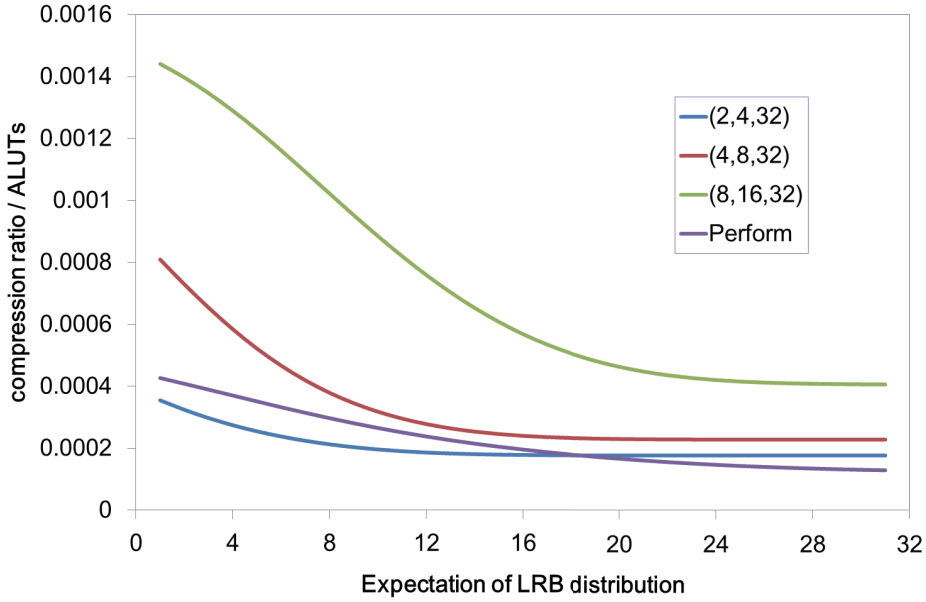


Figure 3.19: Area efficiency by compression ratio per ALUTs ($\mu = 5$).

of the barrel shifter are always multiples of the GCD. It should select a large GCD for a small compressor. On the other hand, the number of selected L-LRB does not affect too much for the compression ratio because the barrel shifter in the compressor is unaffected other than the shift amount. The length of LRB is determined by the selected number of L-LRB, it has two bits when it selects three or less numbers, or it has three bits when it selects more than four numbers. There are almost no differences between the 2-bit LRB and the 3-bit LRB, as the aforementioned theory tells.

Fig. 3.23 shows the numbers of required multiplexors for the shifters in the decompressor. The area-oriented design has no barrel shifter in the decompressor, and employ multiple fixed shifters as alternated. Since the number of the shifters is as high as the selected L-LRB, the number of required multiplexors increases with increasing of the number of L-LRB. Basically the area of the shifters bears a proportionate relationship to the number of L-LRB, the relationship disappears when the number of bits in LRB changes. These graphs describe that it should select small number of L-LRB with large GCD of the selected L-LRB so that we obtain small hardware implementations.

Second, we evaluate the compression performance for various LRB distributions. Figs. 3.25-3.35 shows distributions of data compression ratios for various selected L-LRBs, and Fig. 3.24 shows by the original encoding. In these pictures, the vertical axis represent standard deviations and the horizontal axis represent expected values or means of LRB distributions. The small vertically-long boxes in the graphs show the clusters of the mean and the standard deviation. The color intensity shows the compression ratio for the corresponding LRB distributions.

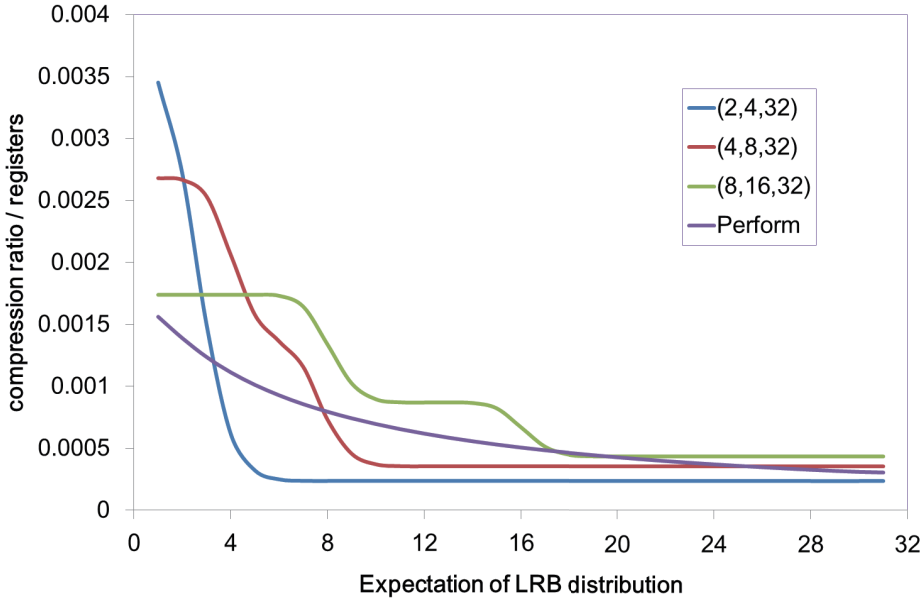


Figure 3.20: Area efficiency by compression ratio per registers ($\mu = 1$).

It evaluates compression ratio with three, four, or five selected L-LRBs. The distributions of LRB are determined by the properties of data as shown in Figs. 3.8 and 3.9. A picture with large area that is colored indicates that selected L-LRB effects for a variety of data. These results show that the compression performances are greatly changed by selection of L-LRBs. With selecting small L-LRB such as two, the compression performances in a small region in the lower left corner are very high. On the other hand, with large L-LRB, the largest the compression performance is not so high, however, the compression is effective for a large region. The original one achieves effective data compressions for the largest region and also high compression ratio. For the standard deviation, area-oriented designs tend to achieve good compression performance with small standard deviation. Therefore, the L-LRB, especially with selecting small L-LRB, is effective when the deviations of LRB distribution is very strong.

For these results, we obtain the criteria for selecting L-LRB. If a large amount of hardware resource is available, it should select the original encoding which achieves the most effective compression for a wide region. If we can use an insufficient hardware resource, it should employ the area-oriented design and select L-LRBs according to LRB distribution. As shown in last section, the LRB distribution is determined by the environment of computation. Therefore, if you suspect that the LRB distribution have strong deviation such as shown in Fig. 3.8, you should select small L-LRBs which is very effective for a small region of the results. On the other hand, when the LRB distribution does not have very strong deviation such as shown in Fig. 3.9, you should select large L-LRBs such as 8, 16, 32 which achieves sufficient compression performances for a large region.

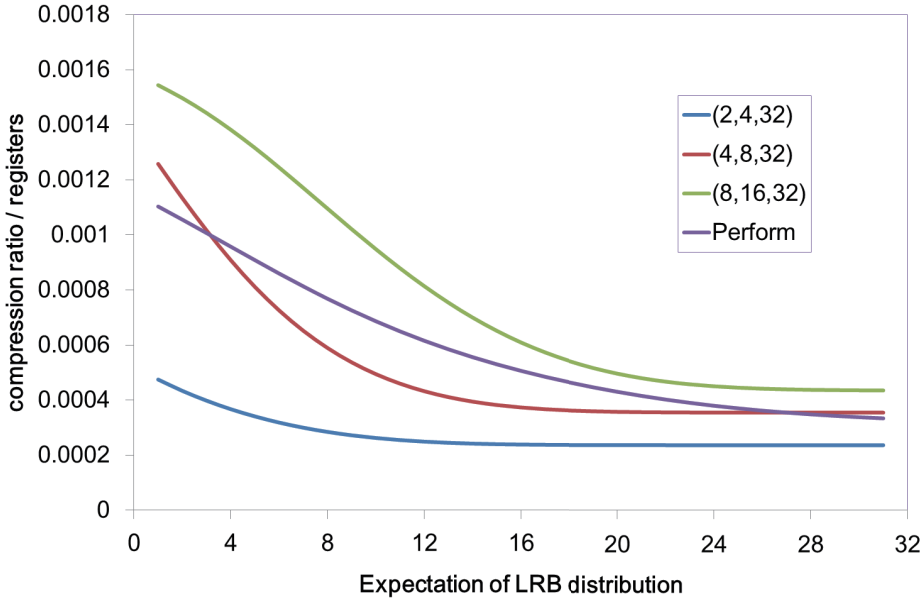


Figure 3.21: Area efficiency by compression ratio per registers ($\mu = 5$).

In addition, the number of selected L-LRB does not affect for the compression performance. We expected that many L-LRBs increases the compression performance because it can provide suitable compression to a large region. However, the results deny this expectation, in which compression ratios are differ little from the cases with fewer selected L-LRBs. This is because that a lot of selected L-LRB cause increase of bits for LRB. For example, it needs two bits for LRB with three L-LRBs, on the other hand, with four L-LRBs, LRB needs three bits. Therefore, the latter needs one extra bit per one compressed datum. From the result, this study employ selecting three L-LRBs for the hardware implementation.

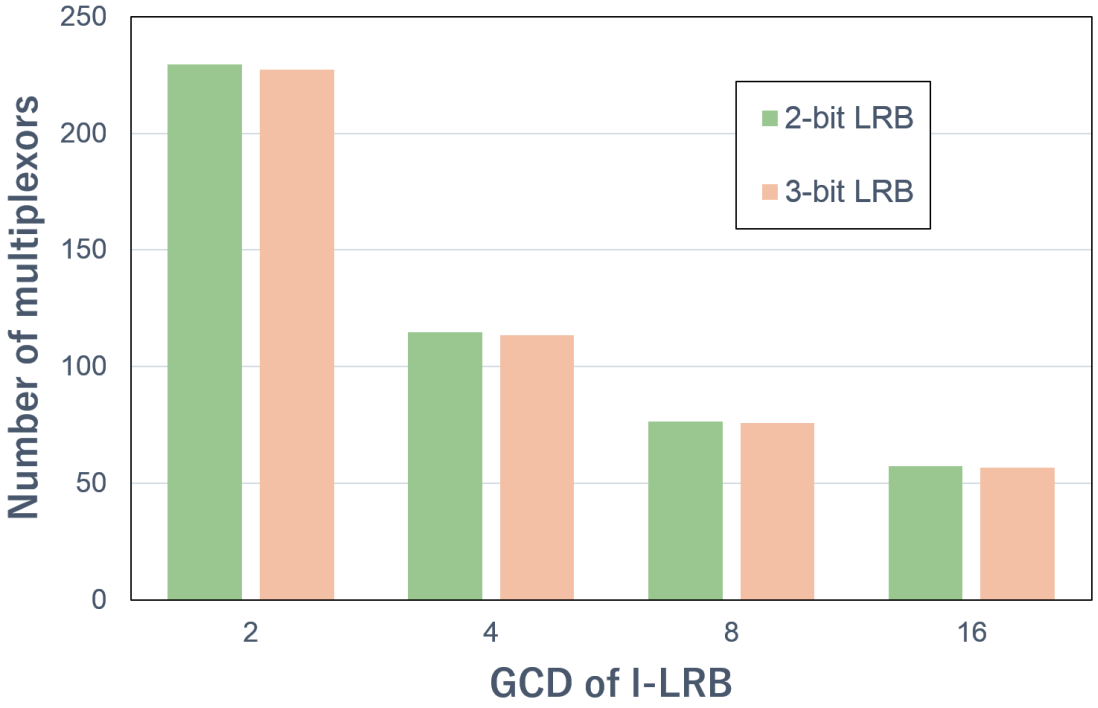


Figure 3.22: Required multiplexors for the barrel shifter in the compresser.

3.7 Conclusions

In this chapter, we proposed the area-oriented encoding method to reduce the hardware area so as to apply the bandwidth compressor to real-world applications with a lot of stream channels. Since the hardware-based numerical applications requires multiple channels processing, we need to apply the compressors and decompressors to every channels one to one. Therefore, the area of the compressor and decompressor should be small, in addition, we need a flexible design in a variety of conditions. We should use the hardware resources mainly for the computation itself rather than the bandwidth compression.

Since the reason of the huge hardware area is the number of possible LRB values, we proposed L-LRB to limit the possible range of LRB to reduce it. In addition, to simplify the hardware processing, we improved the structure of CDB. Through these improvements, we reduce the hardware area of VFC and FVC especially. Their improved designs allow us to select the optimal design in accordance with a purpose. We also optimized the structure of VFC by splitting the barrel shifter to further reduce the area.

In the evaluation, we found the decrease of the compression ratio with the area-oriented design. On the other hand, the evaluation with data generated by numerical simulations, which are of real-world applications, shows that the proposed encoding method is effective for various

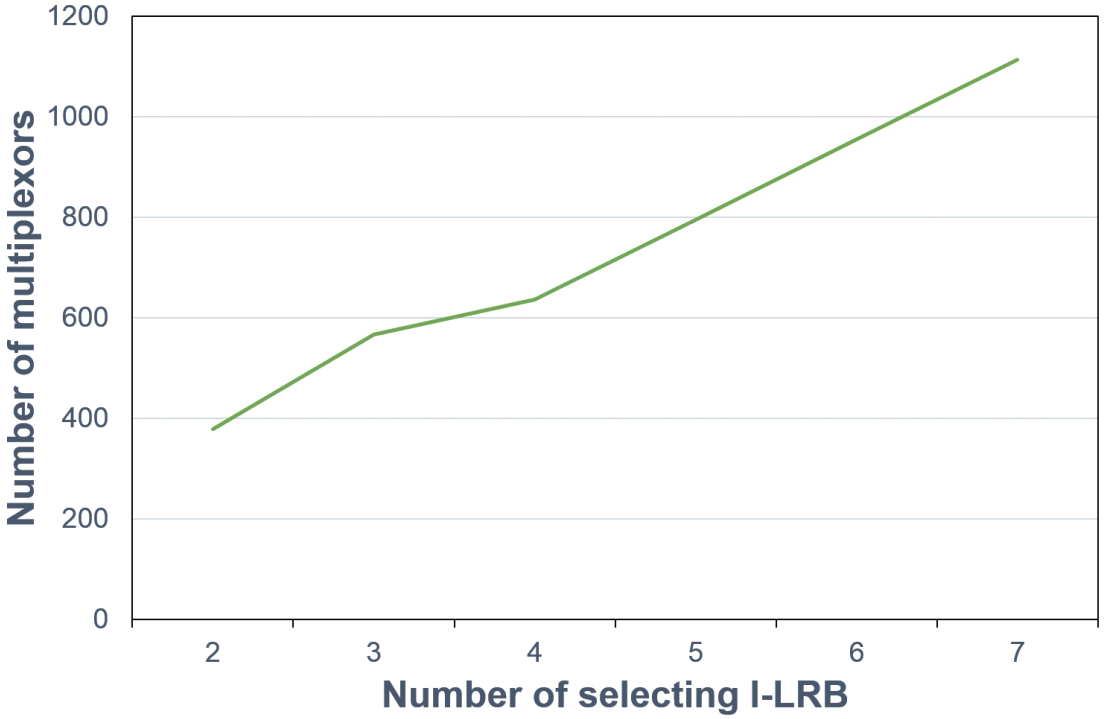


Figure 3.23: Required multiplexors for shifters in the decompresser.

types of data. The improved design achieves to reduce the hardware area dramatically. We made sure that the area-oriented design overcome the performance-oriented design in compression ratio per area. Moreover, we also made sure that the area-oriented design can be applied to various data with different characteristics by properly.

With these results and discussions, we conclude that the proposed encoding design and its new hardware design provide us efficient and feasible solution for bandwidth compression to be applied to FPGA-based high-performance stream computation. In the next chapter, we will apply the new compressor and decompressor to real-world numerical applications with FPGA.

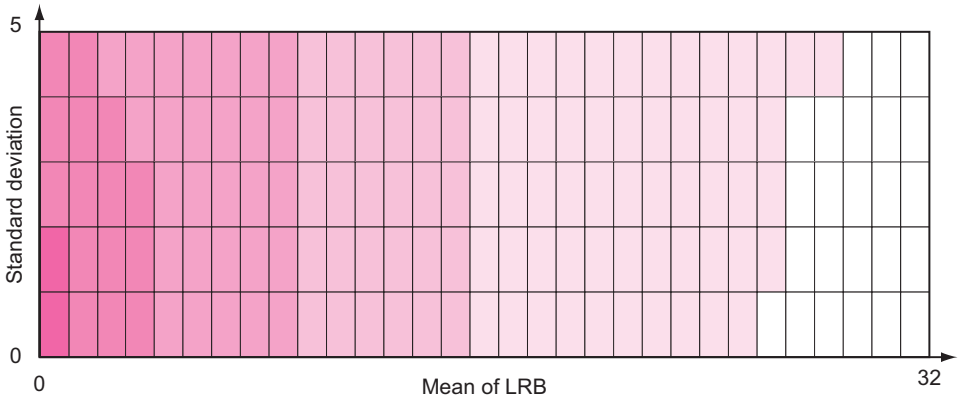


Figure 3.24: Distribution of compression ratios with the original encoding.

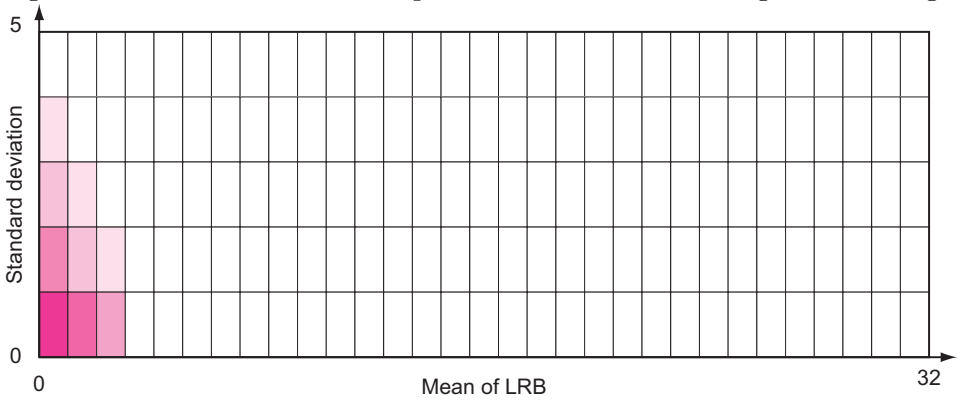


Figure 3.25: Distribution of compression ratios with l-LRBs {2, 4, 32}.

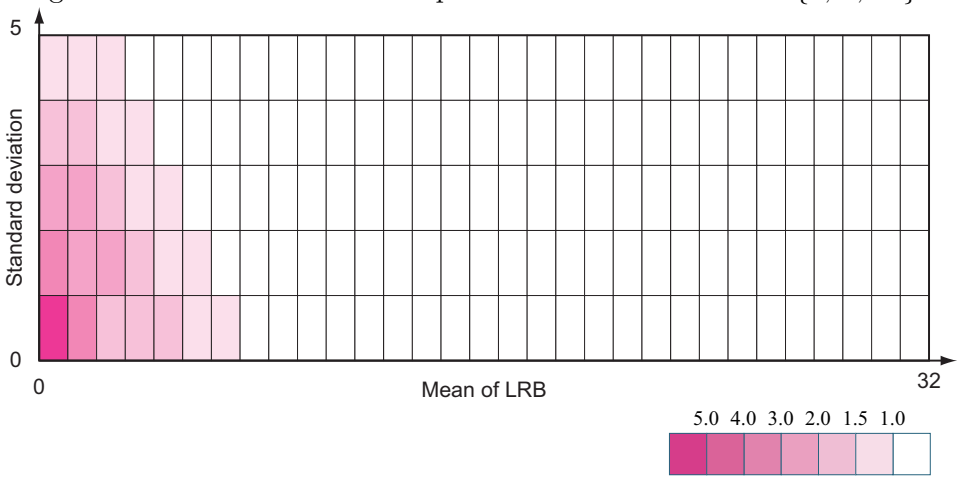


Figure 3.26: Distribution of compression ratios with l-LRBs {2, 8, 32}.

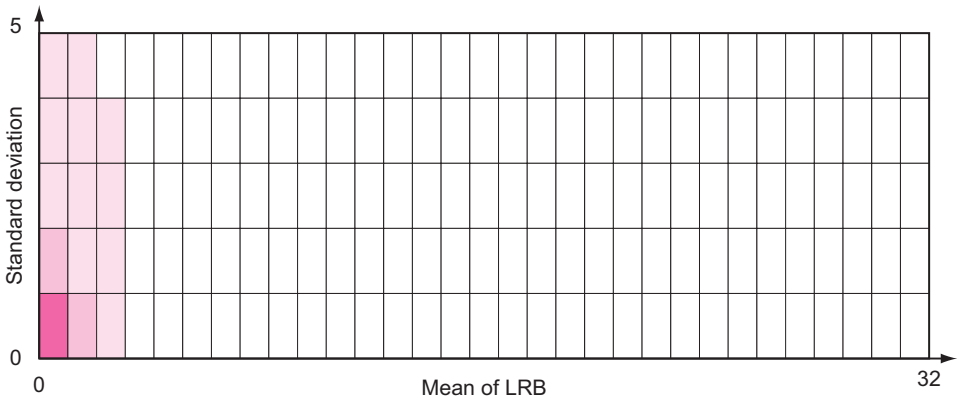


Figure 3.27: Distribution of compression ratios with l-LRBs {2, 16, 32}.

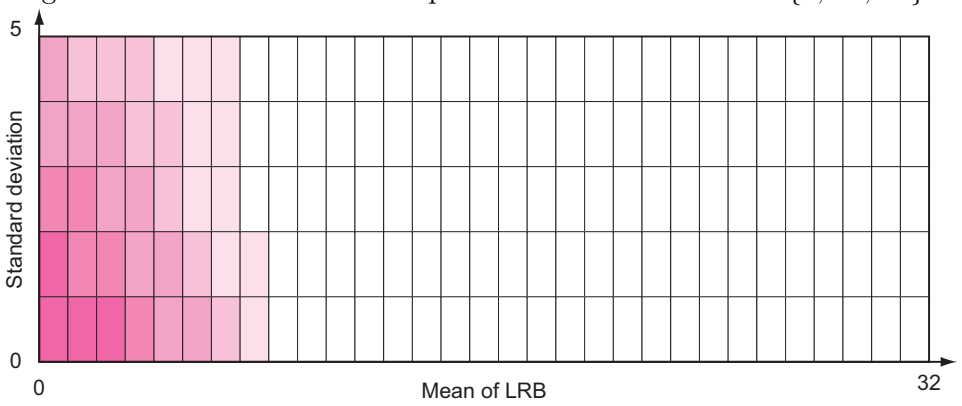


Figure 3.28: Distribution of compression ratios with l-LRBs {4, 8, 32}.

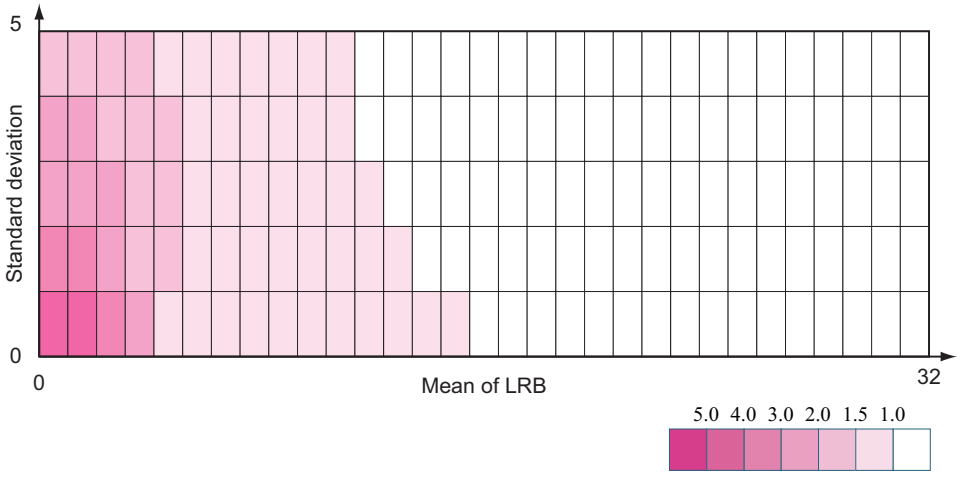


Figure 3.29: Distribution of compression ratios with l-LRBs {4, 16, 32}.

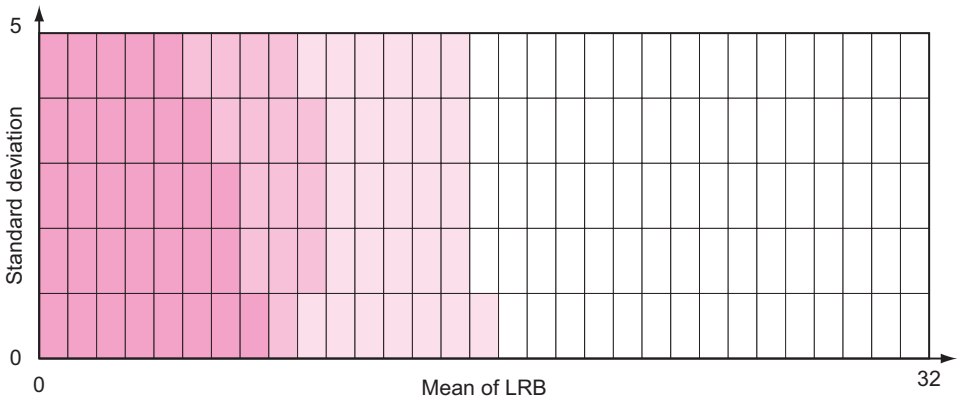


Figure 3.30: Distribution of compression ratios with l-LRBs {8, 16, 32}.

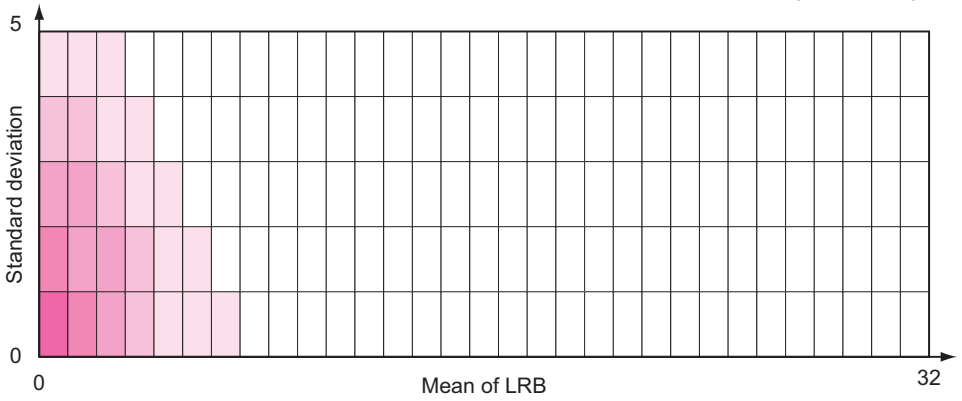


Figure 3.31: Distribution of compression ratios with l-LRBs {2, 4, 8, 32}.

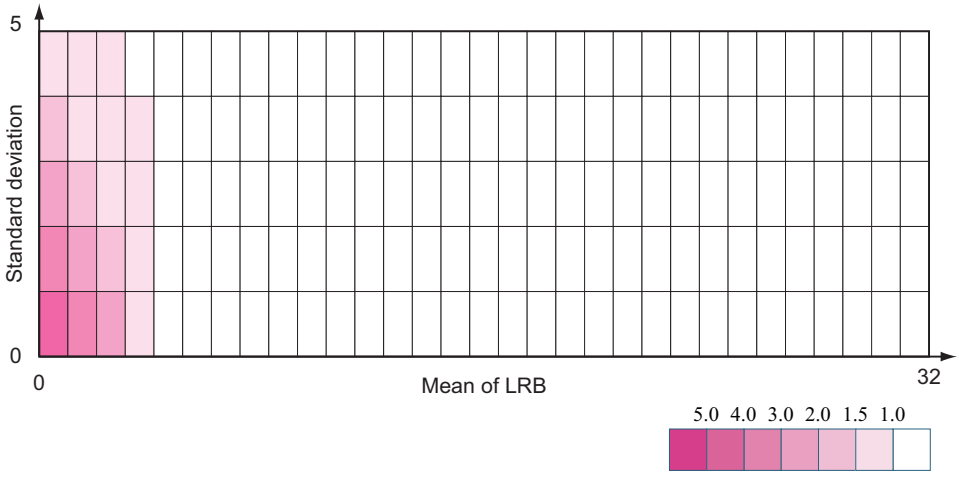


Figure 3.32: Distribution of compression ratios with l-LRBs {2, 4, 16, 32}.

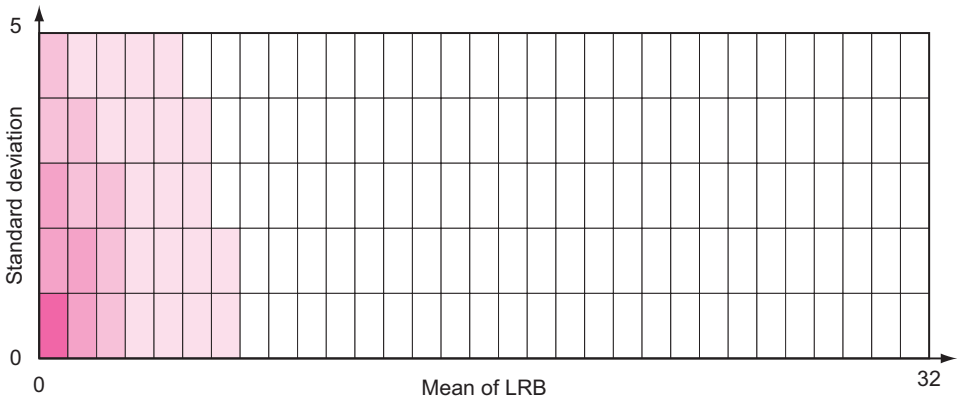


Figure 3.33: Distribution of compression ratios with l-LRBs {2, 8, 16, 32}.

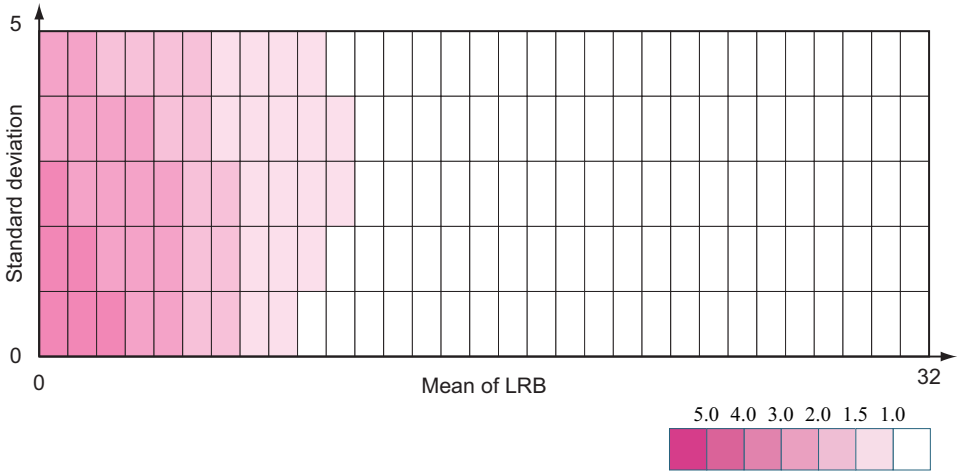


Figure 3.34: Distribution of compression ratios with l-LRBs {4, 8, 16, 32}.

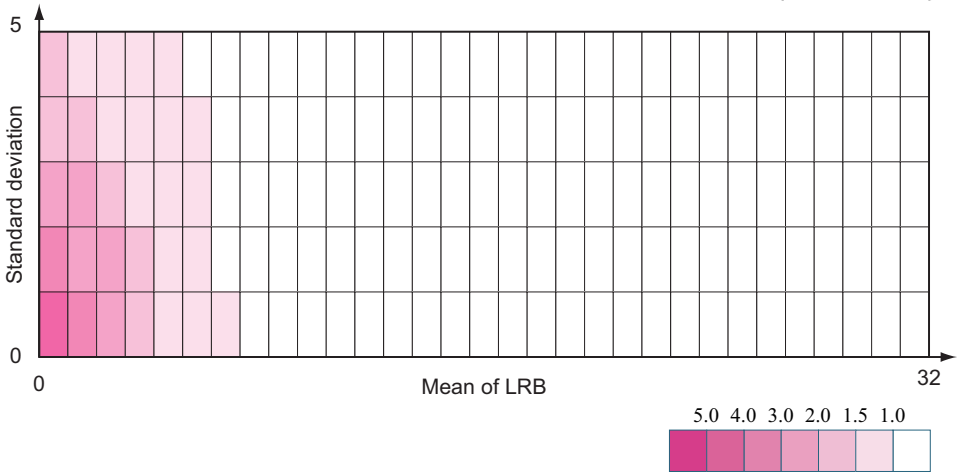


Figure 3.35: Distribution of compression ratios with l-LRBs {2, 4, 8, 16, 32}.

Chapter 4

Multiple-channel bandwidth compressor for real-world applications

4.1 Introduction

The goal of this study is to accelerate FPGA-based numerical computation for real-world applications by enhancing the memory bandwidth with our bandwidth compression technique. Since we solved the area problem in chapter3, now we are ready to apply our bandwidth compression to increase the performance of stream computing on FPGA. This chapter presents the idea and design of multi-channel data compression and decompression. There are also several requirements for handling the multi-channel of numerical stream. There are some studies about parallelized data compression, using segment parallel prediction [27], for audio compression [70], and EEG (electroencephalogram) compression [71].

First, the numerical computing requires synchronization among channels because each of all channels is required to give one datum for a unit computation. For example, LBM computation for a cell requires one scalar datum from each of 10 channels [22]. Therefore, the input data for the numerical core are also required synchronization. On the other hand, since the compression ratios of channels can be different from others, it is difficult to keep the synchronization if we equally read CDBs among all channels. Such a channel that has a low compression ratio can fail to output a decompressed datum.

Second, since a transmission path to a memory is usually given a single channel, we have to bundle multiple compressed channels into a single channel. In decompression, we have to distribute single channel from the memory to the multiple channels. This is because an external memory which is usually implemented with DRAMs. DRAMs are good at regular and successive access for a single channel.

In this chapter, we develop modules which are called a multi-channel serializer (MCS) and

a multi-channel deserializer (MCD) to satisfy these requirements. MCS serializes the multiple channels into a single channel, and MCD deserializes the single channel to the multiple channels. We present the approach to handle multiple channels with uneven compression ratios, and then describe the hardware design and its behavior of MCS and MCD.

Finally, we apply the bandwidth compressor to real-world numerical application on FPGA. The compressor internally enhances the effective memory bandwidth, which leads to improvement of computing performance. For evaluation, we implement a prototype system with our multi-channel bandwidth compressor and decompressor with numerical computing module on an FPGA. The evaluations show not only the area of entire system, but also the bandwidth enhancement and computing performance with the bandwidth compression. The contributions of this chapter are

1. Multi-channels data compression method,
2. Designs of MCS and MCD,
3. Entire design of the bandwidth compressor,
4. Evaluation with real-world application.

The organization of the chapter is as follows. Section 4.2 describes the requirements of multi-channel compression and proposes the method. Section 4.3 shows the hardware design of multi-channel bandwidth compressor. Section 4.4 presents evaluations and demonstration the bandwidth compression with real-world application. Finally section 4.5 gives conclusions of this chapter.

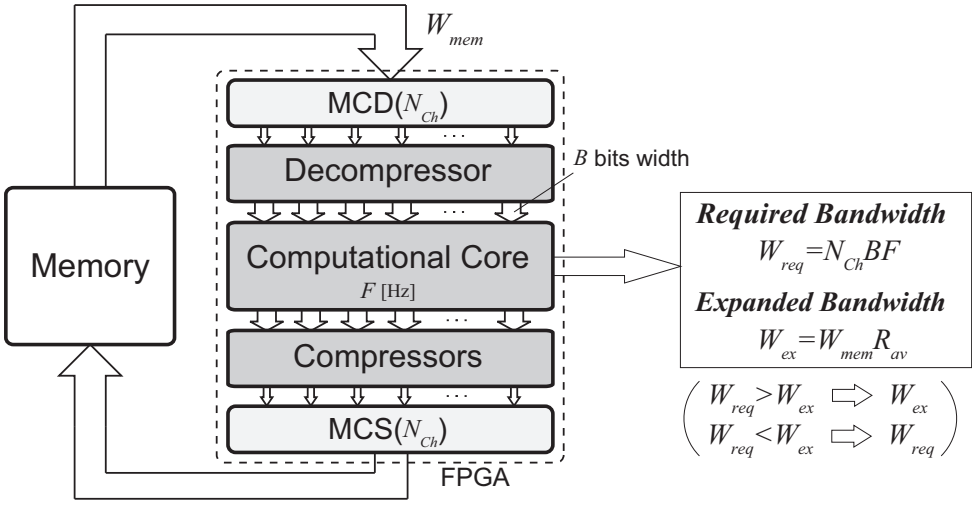


Figure 4.1: Bandwidth Compression in stream computing.

4.2 Multi-channel bandwidth compressor

The bandwidth compressor presented in previous chapters handles only a single channel of a numerical data stream. To apply it to a real-world application which has multiple variables, we need to handle multiple channels and compress them simultaneously. Since each channel has its own data continuity, it is natural to apply the data compressor and decompressor to each channel and perform compression of all channels in parallel as shown in Fig.4.1.

On the other hand, it is favorable to use a single DMA (direct memory access) module to read and write an external memory in terms of hardware-resource consumption and efficient utilization of a memory bandwidth. Multiple DMAs for one FPGA can be inefficient due to some loss of available memory bandwidth for access conflicts. Thus we need to encode and decode CDBs of channels to and from a single data stream for an external memory.

In addition, the memory bandwidth is not so constant especially transmitting a large amount of data. Fig 4.2 shows the actual memory bandwidth with various stride width accesses. The model of the memory is DDR3-1600, which can achieves 12.8 GB/s for its peak. However, the result shows that the peak memory bandwidth cannot be achieved easily, and the bandwidth largely reduces when the stride width is large. With the sequential access, the result does not effect for the performance, however, for large scale numerical computing, especially three dimensional, the stride access is necessary, which causes a decrease of performance. Therefore, the bandwidth compression is required to increase practical performances.

We achieve this function by introducing time-division multiplexing (TDM). In this section, we propose a multi-channel serializer (MCS) and a deserializer (MCD) to serialize and deserialize CDBs, respectively. Considering the requirement of synchronous inputs to a computing core and

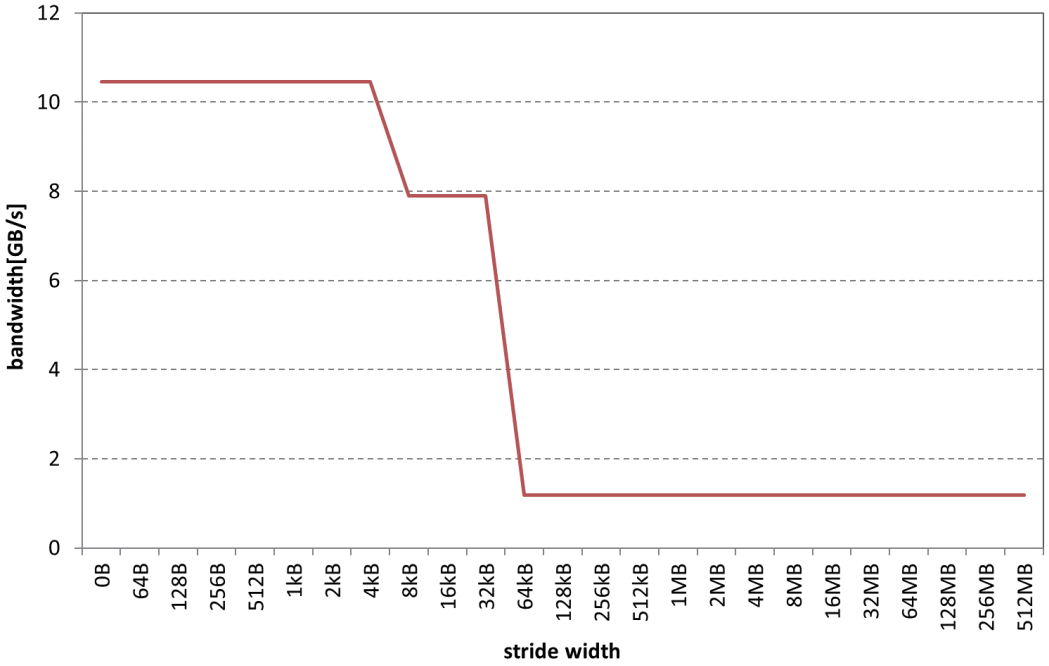


Figure 4.2: Memory bandwidth for various stride widths.

uniformity of compression ratios among channels, we need to deploy an appropriate mechanisms to MCS and MCD. In the following subsections, we describe a model of a compressed bandwidth for multiple channels, a basic idea and a hardware design of MCS and MCD.

4.2.1 Bandwidth of stream computing

For stream computing with bandwidth compression, we can expand a memory bandwidth for computing according to a compression ratio. Let W_{mem} and R_{av} denote an available memory bandwidth and an average compression ratio. The maximum expanded bandwidth is theoretically given with $W_{mem}R_{av}$. Hence we should design a computing hardware core so that it can operate with the maximum expanded bandwidth to fully utilize the available memory bandwidth. Or the computing performance is limited by the memory bandwidth or the core itself.

Here we give a model to distinguish performance limitation. as shown in Fig.4.1. Let a computing core have N_{ch} channel for inputs and outputs. Each channel has a width of B bits and operates at a frequency of F [Hz]. Then the bandwidth required by the core, W_{req}, is given as

$$W_{req} = N_{ch}BF \quad [\text{bits}\,/\,\text{s}]. \tag{4.1}$$

For the most efficient design of the core, we balance the required bandwidth with the expanded

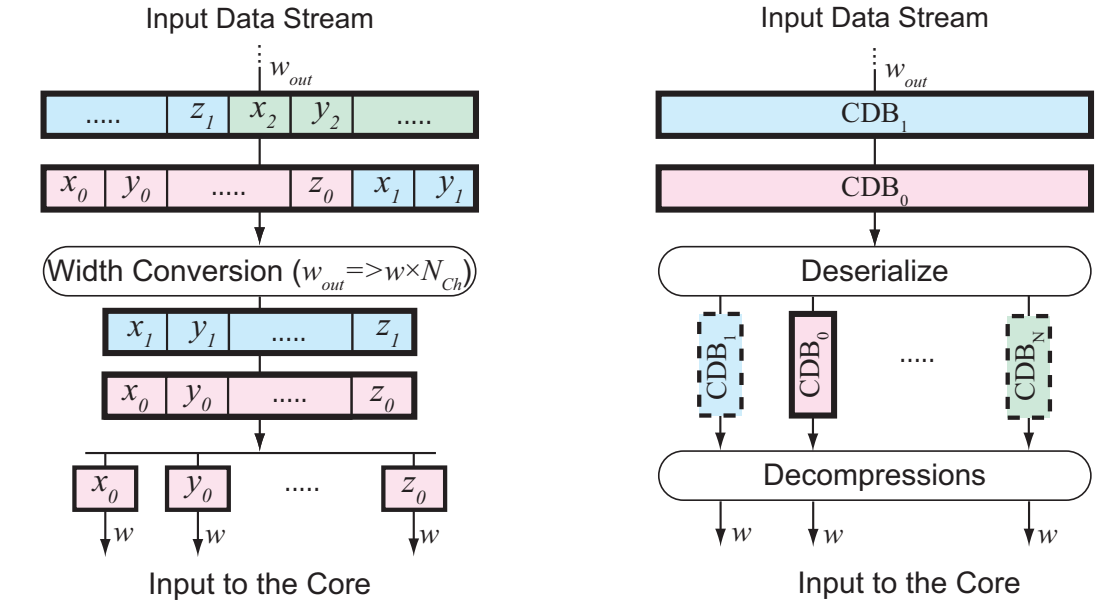


Figure 4.3: Input data format of stream computing with/without bandwidth compression.

bandwidth so that

$$W_{req} = W_{mem} R_{av}. \tag{4.2}$$

If $W_{req} < W_{mem} R_{av}$, a performance bottleneck is in the core itself. If $W_{req} > W_{mem} R_{av}$, the sustained performance is limited by the memory bandwidth, and therefore we cannot fully exploit the peak performance of the core.

When we know the average compression ratio for computation, we can estimate the necessary number of channels to fully utilize both the memory bandwidth and the peak performance of the core. By substituting Eq.(4.1) to Eq.(4.2), we obtain

$$N_{ch} = \frac{W_{mem} R_{av}}{BF}, \tag{4.3}$$

which means that we can exploit more parallelism with more channels for a higher compression ratio. We also obtain

$$R_{av} = \frac{N_{ch} BF}{W_{mem}}, \tag{4.4}$$

which means that a higher compression ratio is required for a smaller memory bandwidth available.

4.2.2 Scheme of multi-channel compression

Fig.4.3(a) shows an input data format to the computational core with raw data streams, which our proposed bandwidth compression targets at. The input data streams are sequentially stored in a memory region. Fig.4.3(a) shows an array of a structure, where data elements such as $x_i, y_i, ...$ are packed into a group first, and then the groups are arranged in a certain order. The array is efficiently read by using a single DMA controller. Since the width of read data can be different from that of the inputs to a computing core, it requires a width conversion. Then, the input words are sent to the core by multiple channels, where the elements of the channels are synchronously consumed for computation. We also convert their width with another converter for an output data stream to be written to the external memory.

On the other hand, Fig.4.3(b) shows an input data format for compressed stream computing, where we apply the single decompressor and compressor to each channel. Furthermore, we deploy a multi-channel serializer (MCS) and a multi-channel deserializer (MCD) to read/write CDBs by a single data stream from/to an external memory. We give a CDB the same bit-width as that of the memory interface, so that we can read or write one CDB every cycle. The read CDB is deserialized and distributed to its corresponding channel. Since one CDB consists of compressed data of one channel, it cannot input to the core until all the channels have CDBs to satisfy the requirement of synchronous input in every channel. Once all the decompressors have their CDBs, they start decoding the original data elements to be fed to the computing core. The output elements of the core are separately compressed into CDBs. The output CDBs are then serialized into an output data stream by MCS.

Here, due to ununiformity of compression ratios among channels, we have to make a suitable mechanism for MCD and MCS to adequately distribute frequencies to transmit CDBs among channels. Or we would fail in synchronously feeding all the data elements to the core every cycle. Let's assume that some channel has a lower compression ratio than the others. A CDB generated for such a channel contains a smaller number of original elements than CDBs for other channels do. Therefore we need more CDBs for the channel to decode the same number of original elements as those for the other channels. Thus, if we simply apply a round-robin method to equally distribute a memory bandwidth to the channels, we can have an insufficient number of CDBs for some channels due to ununiformity of compression ratios. Instead of such a naive approach, we propose MCS and MCD to adaptively distribute the bandwidth according to the compression ratios of the channels.

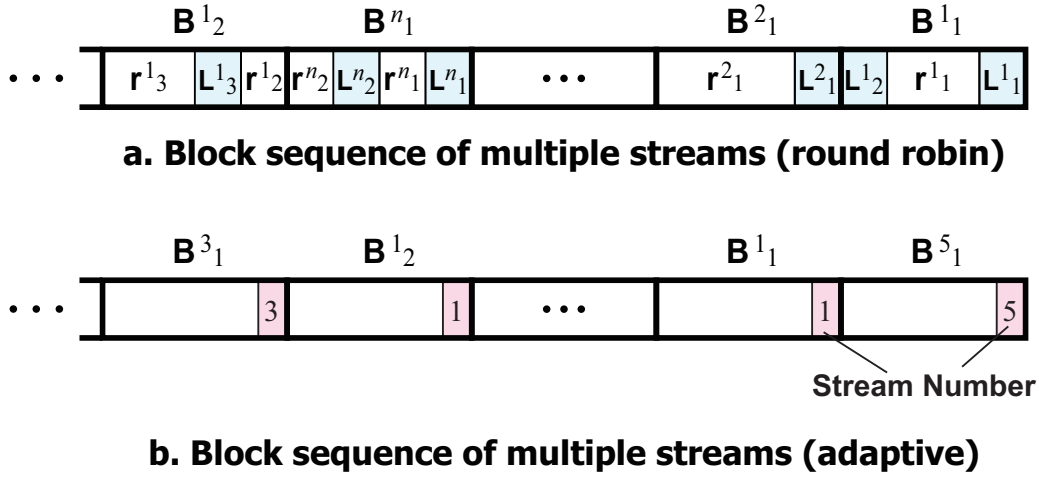


Figure 4.4: Approaches to encode multiple streams.

4.2.3 Output arrangement

Numerical computing circuits require data of the same grid point to be inputted at the same time. The round-robin arrangement of blocks shown in Fig. 4.4a cannot satisfy this requirement. Let's suppose that compressed data of each stream are decomposed into blocks with the size of 256 bits. In the case of compressing 32-bit floating-point datum, the minimum and maximum sizes of compressed datum are 6 bits and 38 bits, respectively. Accordingly, each 256-bit block can contain 7 to 42 compressed floating-point data. Since the round-robin arrangement uniformly distributes the bandwidth of the channel to n_{str} streams, the decompressor with the low compression ratio consumes the block data more quickly than that of a highly-compressed stream. Decompressing 7 data takes only 7 cycles while 42 data require 42 cycles. Therefore, we must allocate the bandwidth to each stream so as to transmit data before compression equally in every stream.

To meet this requirement among streams with different compression ratios, we need to allocate wider bandwidth to streams with lower compression ratios. We can give a higher priority to a less compressed stream, so that more blocks of the stream are adaptively selected and transmitted. Therefore, the order of output must be determined by the compression ratios instead of round-robin. Without round-robin arrangement, some information is necessary for received blocks to be correctly distributed to their destination decompressors. We add the stream number to the output blocks, and the output stream becomes as shown in Fig. 4.4b. The block size, w_{out}, should be big in order to increase the proportion of compressed data. Therefore, we set the block size, w_{out}, as the output width of the FPGA.

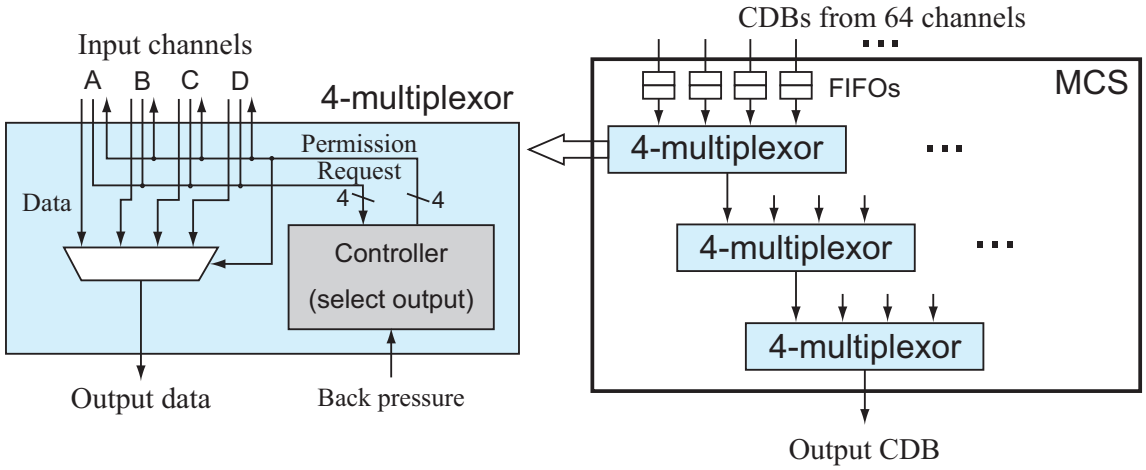


Figure 4.5: Multi-channel serializer (MCS).

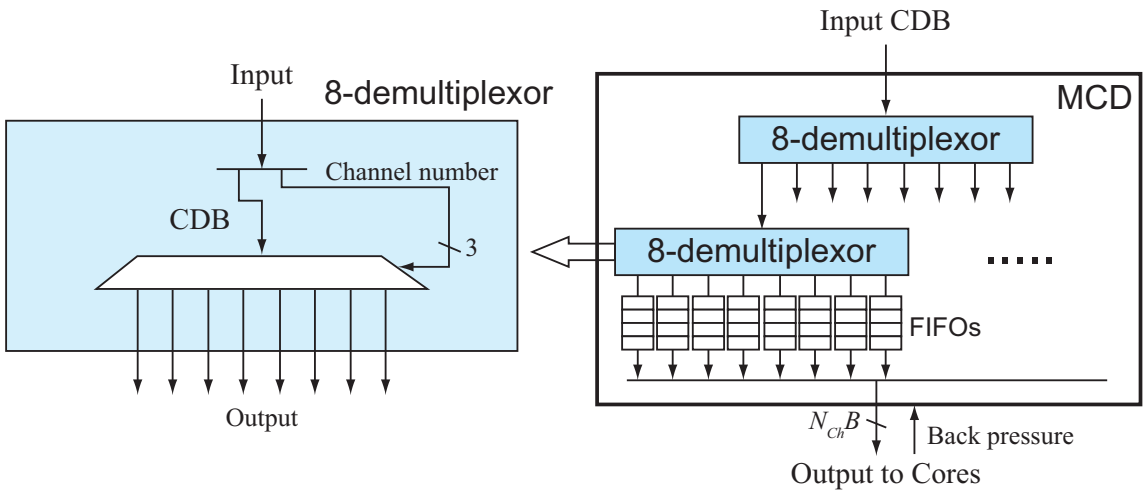


Figure 4.6: Multi-channel deserializer (MCD).

4.3 Hardware design

MCS and MCD serializes and deserializes CDBs for multiple channels into a single data stream for an external memory. MCS operates like a multiplexor with a selection controller. MCD is a demultiplexor to distribute an input CDB to its corresponding channel according to the channel indication field shown in Fig.3.11(b).

For MCS and MCD, we employ a tree structure for high-throughput and scalable hardware by localizing control. Fig.4.5 shows the structure of MCS, which consists of 4-channel selectors as common units. The 4-channel selector forms one pipeline stage. It is an example for 64 channels, where 21 selectors give three pipeline stages in a tree. We have also designed selectors for 2 and 3 channels, therefore, it can manage a wide range of numbers of channels by a combination of

these selectors.

The 4-channel selector selects and outputs one among four inputs with its internal controller. In order to control selection on a time-bases, we design MCS with the following mechanism. First, once we get multiple output request inputs at a time, they go to a selection process while the other input ports are locked until all the request inputs are output. Suppose we have three request inputs at port 1, 3, and 4. For these three, we select and output one by priority of port numbers. In this example, we sequentially output the words of input ports 1, 3, and 4. When the first input words are all output in three cycles, all the input ports are released for the next inputs.

In the case of a 512-bit CDB, a compressor generates CDBs at intervals of 11 or more cycles because each CDB can contain at least 11 data elements. This means that we commonly have request inputs only for a few ports, rather than having full inputs to all the ports. Thus MCS can normally transmit CDBs of input port as they arrive, resulting in more bandwidth for ports required. When multiple ports have inputs occasionally, they are prioritized to avoid uneven selection frequencies.

Fig. 4.6 shows the structure of MCD, which is a distribution tree with 8-channel demultiplexors. The mechanism of MCD is straight forward. Input CDBs are demultiplexed by using a channel number recorded in the indication field of the CDB. The control logic of MCD is also localized to each selector. MCD also contains multistage FIFOs in every output channel to guarantee synchronous outputs. Therefore, in our design, all compressors and decompressors also process synchronously to keep the synchronous input to the computational core.

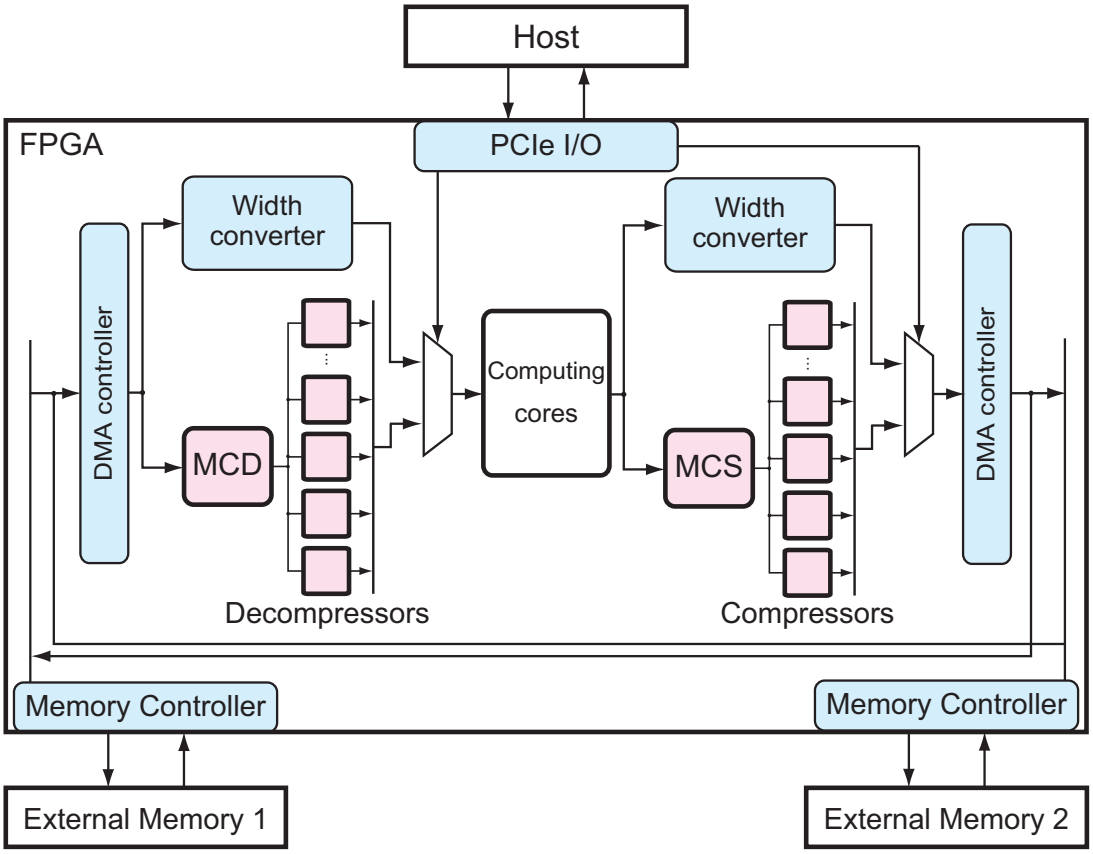


Figure 4.7: Entire system implemented on FPGA.

4.4 Evaluation and demonstration

We show an actual implementation of multi-channel bandwidth compression with a numerical computation on a FPGA. We employ a numerical core of 2-dimentional Lattice Boltzmann Methods (2D-LBM) [22] to implement with the multi-channel bandwidth compressor. The 2D-LBM is one of computational fluid dynamics (CFD) methods which computes fluid dynamics by a movement of particles. It contains nine variables as particle distribution functions for nine spatial directions, hence one 2D-LBM core has ten channels, of which nine channels are for numerical data and one channel is for attribute data. We show the computational throughput exceeds the memory bandwidth by the bandwidth compression for the LBM numerical data. Additionally, we also evaluate an impact of the bandwidth compressor by measuring computational performance of a real application with or without the numerical core.

4.4.1 Implementation

We design and implement the entire system for numerical stream computation with the bandwidth compression, shown in Fig. 4.7. In this implementation, since a memory has uncompressed

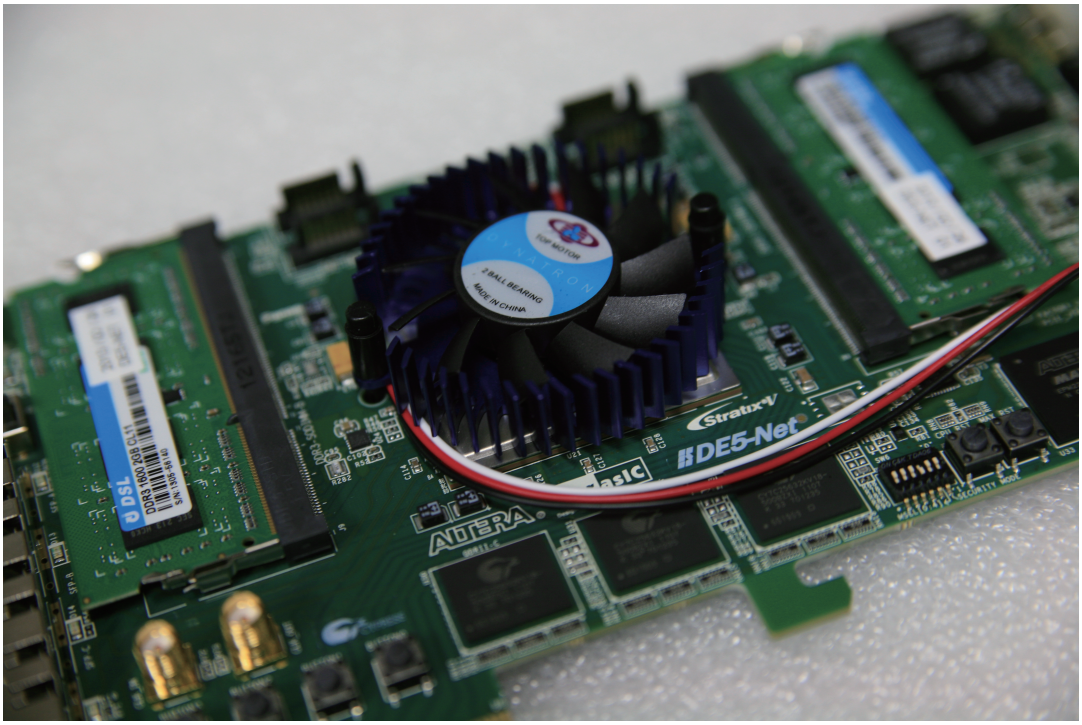


Figure 4.8: Stratix V FPGA board using by the implementation.

data at the beginning, it needs the route without compression and decompression to divert the data stream at the beginning and the end of the computation. First, we input original data without decompressions into the core. Then, the data always pass through the compressors and decompressors except the first and last iterations. In the last iteration, it outputs computed data stream without the compression to check the result.

We implemented this system with ALTERA Stratix V FPGA which is on TERASIC DE5-NET development-board shown in Fig 4.8. All the logics are written in Verilog- HDL, and compiled with ALTERA Quartus II compiler ver.14.1. The FPGA contains 234,720 adaptive logic modules (ALMs) and 939,000 registers. In the demonstration, numerical data are represented by 32-bit single precision floating-point, and the input and output width of FPGA is 512-bit, thus $w = 32$ and $w_{out} = 512$. For the evaluation, we implement two cycle counter modules to record a number of cycles. The bandwidth calculated by the recorded data in the cycle counters which can record a number of operating cycles and validation cycles. Therefore, we can infer the actual bandwidth from the data and a width of the data path between the decompressors and compressors. We also measure the actual compression ratio in the operation. The operating frequency of the core and bandwidth compressor is 150 MHz. The employed memories have the peak bandwidth of approximately 8.0 GB/s, which is actually measured on the implemented system.

Table 4.1 shows resource utilization of the entire system and individual modules. The table

Table 4.1: Resource usage of the numerical cores with bandwidth compression

Cores	Module	ALMs	Registers	Block memory bits	DSP blocks
1 (10 chs.)	–	109630 (47%)	160824 (17%)	157747 (3%)	48 (19%)
	Comp. and MCS	12848 (5.3%)	19748 (2.1%)	20380 (0.04%)	0 (0%)
	Decomp. and MCD	7826 (3.3%)	19343 (2.1%)	40640 (0.1%)	0 (0%)
	Computational core	32717 (13.9%)	61907 (6.6%)	1042986 (2.0%)	48 (18.8%)
2 (20 chs.)	–	152317 (65%)	219135 (23.3%)	2229043 (4%)	96 (38%)
	Comp. and MCS	21333 (9.1%)	21601 (2.3%)	40960 (0.1%)	0 (0.0%)
	Decomp. and MCD	10993 (4.7%)	17304 (1.8%)	81280 (0.2%)	0 (0.0%)
	Computational core	58785 (25.0%)	87939 (9.4%)	1078166 (2.0%)	96 (37.8%)
3 (30 chs.)	–	195151 (83%)	302751 (32%)	2252422 (4%)	144 (56%)
	Comp. and MCS	30582 (13.0%)	31356 (3.3%)	61440 (0.1%)	0 (0.0%)
	Decomp. and MCD	16231 (6.9%)	25511 (2.7%)	121920 (0.2%)	0 (0.0%)
	Computational core	75013 (32.0%)	147790 (15.7%)	1039178 (2.0%)	144 (56.3%)

shows the usages of ALMs (adaptive logic modules), registers, block memory, and DSP blocks, and the values in parentheses are the percentages to the total available resources. It shows that the resource usage of the bandwidth compression increases in proportion to the number of channels. With the area-oriented design, we can implement up to three LBM cores which include 30 channels to compress and decompress separately on the FPGA. Therefore, the scale of which can be implemented in a FPGA is limited by the number of ALMs.

4.4.2 Bandwidth enhancement

We conduct experiments on memory bandwidth enhancement without LBM computation cores. We increase the number of channels 10 by 10, and stream the result of the 2D LBM computation to each set of 10 channels. Fig. 4.11 shows the enhanced bandwidth between the compressors and the decompressors. Since the evaluation does not use the computational core, every 10 channels data of input are the same, which are the interim results of 2D LBM computation. The graph shows bandwidth in a vertical axis, and a number of channels in a horizontal axis. Black circles with outlined lines show theoretical maximum bandwidths with actual compression ratios, which is calculated by (Compression ratio $\times$ Available memory bandwidth). White circles and black lines show achieved bandwidth calculated from the measured values on the cycle counters. The available memory bandwidth is 8.0 GB/s, which is shown as the horizontal broken line.

In the case of 10 channels, a required memory bandwidth is ($4\text{Byte}\times10\times150\text{MHz} = 6.0\text{GB/s}$) which is narrower than the available memory bandwidth, therefore, the compression is not necessary. On the other hand, when the required bandwidth is wider than 8.0 GB/s a available memory bandwidth, with greater than 13 channels on 150 MHz, the bandwidth compressor enhances the available memory bandwidth in this case. The compression ratio on the operation is 2.59 and it is close to theoretical maximum compression ratio, 2.6875. In the best case, 50 channels, it achieves 2.33 times wider than the available memory bandwidth. Therefore, the

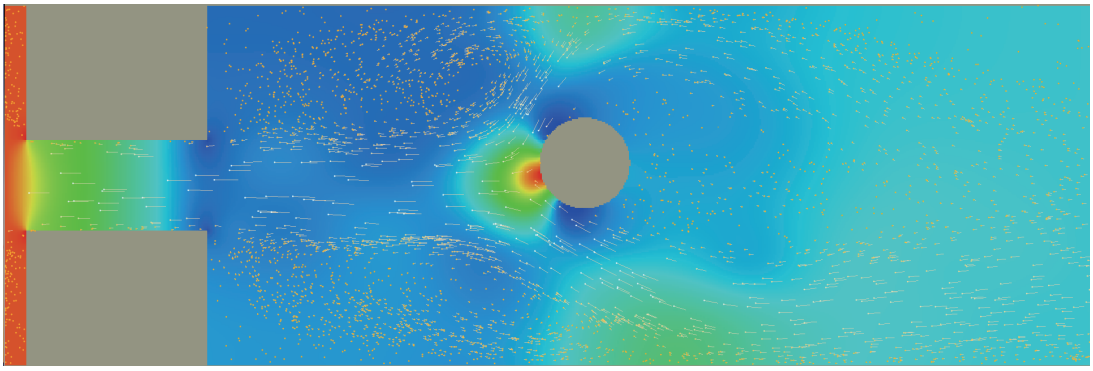


Figure 4.9: Computational result of 2D LBM simulation for a sudden expansion chamber with an obstacle.

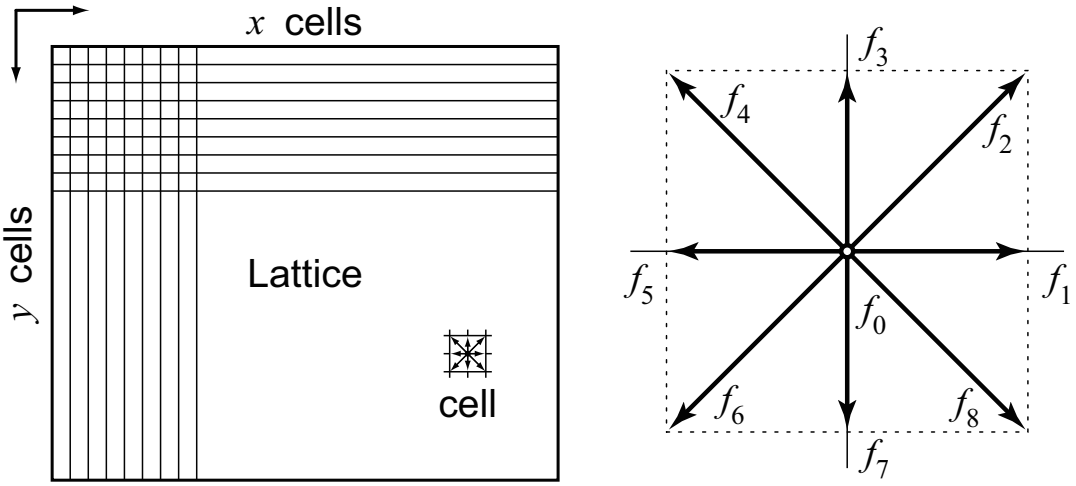


Figure 4.10: Computational grid and grid point of 2DLBM.

bandwidth compression achieves great effect in this demonstration.

Under the compression ratio as 2.59, the theoretical maximum bandwidth is approximately 20.7 GB/s. Since the maximum bandwidth is fixed, the bandwidth increases up to 35 channels and becomes flat thereafter, in an ideal case. The result roughly shows this tendency, however, the bandwidth decreases between the case of 50 and 60 channels. This bandwidth decrement is caused by a large number of channels. Too many channels cause an increase of pipeline stalls in MCS and decompressors. Since the computational core requires that all channels have input data consistently, it also needs to stall the pipeline if there are any channels with no input data. We allocate some FIFOs to prevent such stalls, however, these stalls occur by uneven output among the channels, regardless of the design and compression algorithm.

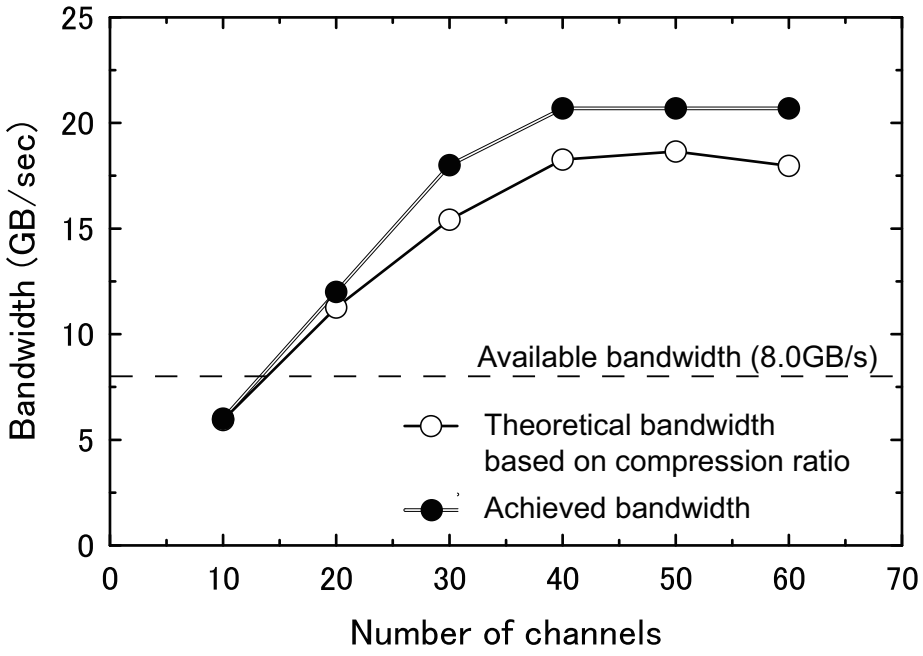


Figure 4.11: Bandwidth enhancement by compression.

4.4.3 Demonstration with numerical computation on FPGA

Finally, we demonstrate the numerical computation with bandwidth compression on a FPGA. We evaluate computing performances of 2D fluid simulation by LBM with the bandwidth compression. The 2DLBM core contains 10 channels, 9 of them are probability distribution density, and the other is an attribution of the grid point. Fig. 4.10 shows the computational grid and grid point of the 2DLBM. The demonstration uses one, two, or three LBM cores which have 10, 20, or 30 channels respectively. We implement the system as shown in Fig. 4.7. As is the case with actual numerical computing, the demonstration calculates and updates values in multiple iterations. This means that we need to repeat calculations as many as a required iterations. Other conditions are the same as the above evaluations.

An application of the demonstration is a flow simulation of abrupt pipe expansion and a cylinder shown in Fig. 4.9. In the demonstration, we use a DSL-based LBM calculation module represented in [72]. This module can calculate a single data stream in parallel. It contains n cores, and each of them receives every n-th data. For example, in the case of $n = 2$, there are two cores in the module, and the module receives data of two continuous grid points at the same time. Each core receives every second data in the data stream and calculates independently.

Table 4.2 shows the average computing performances of 20,000 iterations with and without the bandwidth compression. We obtain the performance by,

$$(\text{Performance}) = nN_{Flops}F_{GHz}R_{op}\ [\text{GFLOP/s}]. \tag{4.5}$$

Table 4.2: Computational performance with/without compression

Cores (channels)	Compression	performance (GFlop/s)
1 (10)	No	19.64
	Yes	19.43
2 (20)	No	26.24
	Yes	**34.45**
3 (30)	No	26.27
	Yes	**45.51**

where n is number of cores in parallel, and N_{Flops} and F_{GHz} are the number of Floating-point operators in one core and the operating frequency, respectively. R_{op} is an operating rate of cores, which is obtained by number of operating cycles and validation cycles measured by the cycle counter. Our LBM core has $N_{Flops} = 131$ and $F_{GHz} = 0.15$. Table 4.2 also shows a required memory bandwidth in each case. They are almost close the bandwidth enhancement shown in Fig. 4.11, therefore, the computational performance is improved by exploiting the extended bandwidth.

The result shows that the bandwidth compression improves the performance in two and three cores. In the case of one LBM core, the bandwidth compression does not work because required memory bandwidth is narrower than the physical bandwidth. When the required bandwidth is wider than the physical bandwidth in the case of two and three cores, the results of without compression are almost the same because of the limitation of available memory bandwidth. On the other hand, it also shows the cases with the compression achieve performances that exceed the limit of physical bandwidth. The bandwidth compressor improves the performances about 31.3% for 20 channels and 73.2% for 30 channels. Although these results depend on compression ratios, it demonstrates that it is possible to realize a performance beyond the bandwidth limit. Therefore, the result shows that the bandwidth compressor improve the performance of custom stream computing.

4.5 Conclusions

This chapter shows the bandwidth compression for real-world numerical application on FPGA. With small compressor and decompressor proposed in chapter3, We implement and demonstrate the bandwidth compressor for real-world applications.

The multi-channel compression for numerical computing requires to keep the synchronized processing in the numerical computing core. To handle the problem, we designed and implemented the MCS and MCD which are encoding multiple channels into a single channel and decoding a single channel into multiple channels, respectively. We employ the area-oriented compressor and decompressor which allow us to implement with a large number of channels. The encoding in MCS employ the determination of output order in accordance with the compression ratio of the channel. Since the output order is dynamically changed, we also employ the indication bits to be distributed to the correct channel.

With prototype implementation, we demonstrated that the bandwidth compressor allows to stream data at a bandwidth higher than the memory bandwidth available. We made sure that the custom stream computing of 2D-LBM achieves 1.73 times higher performance which is provided by the bandwidth enhancement. In this demonstration, the bandwidth compressor achieves the memory bandwidth enhancement and the improvement of computational performance by the bandwidth enhancement. If the compression ratio is R_comp, the bandwidth compressor can enhance the memory bandwidth up to R_comp times theoretically. Although the result shows this tendency, the bandwidth compression for too many channels reduce the effect because there are many stalls in the hardware to allocate data correctly.

The results of the evaluation and demonstration shows that the proposed bandwidth compressor enable to improve the numerical applications on FPGA with insufficient bandwidth. Future FPGAs will have much more resources and wider memory bandwidth. Even in such case, the bandwidth compression is often effective because the increase of cores on FPGA also enlarges required memory bandwidth.

Chapter 5

Conclusions

This dissertation presents the practical bandwidth compression for insufficient memory bandwidth of real-world numerical applications on FPGA. There are huge demands to improve computing performance for large-scale numerical simulations. The bandwidth compressor improves the performance of FPGA-based stream computing for such real-world applications, which has both high power efficiency and high performance. The compression reduces the required bandwidth of computation by employing data compression technique. Since there are many applications that require more than available memory bandwidth, the bandwidth compressor is very useful and applicable to various computing problems.

The objective of this study is performance improvement of the FPGA-based stream computing for practical applications by bandwidth compression. We target numerical simulation with a computational grid, which is referred to as real-world application in this dissertation. For this goal, we consider several requirements due to the numerical computation and the hardware implementation. We divide the objective into the three major challenges: design and evaluation of the data compression hardware, improvement of the data compression hardware for flexibility and smaller area, and design of multi-channel synchronization hardware for bandwidth compression in real-world applications.

In chapter 2, we investigated applicability of the previous data compression that is proposed in our preceding study, and point out their problems in using them to high-performance computation for real-world applications. For data compression of numerical floating-point data, our preceding study proposed the prediction-based algorithm instead of general purpose compression such as entropy coding and prefix coding. The prediction exploits the continuity of the numerical data which we can suppose that the data sequences follow the one dimensional polynomial function. The algorithm is composed of the prediction and the subtraction, both of which are processed in unsigned integer operations to reduce computational cost. We designed and implemented the data compressor and decompressor and evaluated them in terms of the compression ratio, the hardware area, and the maximum operational frequency. We made sure

that they achieve good compression performance and high throughput, however, they have too big area to be implemented with FPGA-based stream computation of real-world applications. Based on the results and discussions, we summarized the requirements for improvement of the data compressor and decompressor for practical applications, which are hardware-area reduction and multi-channel synchronization.

In chapter 3, we improved the data compressor and decompressor to reduce the hardware area. First of all, we investigated the cause of the hardware area growth. The hardware mainly consumed for the data conversion from variable-length compressed data to fixed-length compressed data blocks (CDB). The variable-to-fixed and fixed-to-variable length conversions requires big barrel shifters to concatenate compressed data into CDB and extract a datum from CDB. To reduce the area, we focused on the length of residual bits (LRB) which is a part of the compressed data to indicate the length of residual bits by subtractions to indicate the length of residual bits for the decompression. Based on the fact that the size of the barrel shifters are strongly affected by the dynamic range of LRB, we proposed the quantized encoding method with limited LRB (L-LRB). The L-LRB exploits the deviation of LRB distribution, which is given by the data and prediction accuracy, to reduce the series of the barrel shifters at the limited sacrifice of compression performance. For L-LRB, we also improved the structure of the CDB to simplify the processing in variable-to-fixed length converter (VFC) and fixed-to-variable length converter (FVC). Both L-LRB and CDB improvement reduced the area of the compressor and decompressor dramatically. In addition, we generalized the encoding method with L-LRB by introducing parameters. We provide a criterion to select a set of appropriate parameters according to the statistical characteristic of target data.

In chapter 4, we proposed the mechanism to allow bandwidth compression to synchronously handle multiple channels of compressed data with ununiform compression ratios, and demonstrated improvement of computational performance with the prototype system of our proposal. With small and high-throughput data compressor and decompressor presented in chapter 3, we designed bandwidth compressor for real-world applications using multiple channels. We designed MCS (multi-channel serializer) and MCD (multi-channle deserializer) to handle multiple channels. MCS and MCD control the input and output of CDBs in order to keep the synchronization for stream computing. Through a demonstration with the dedicated 2D-LBM core, we made sure that the bandwidth of the I/O port of the core is improved. Simultaneously, we made sure that the computational performance is also increased even if the available memory bandwidth is insufficient without bandwidth compression. Finally the bandwidth compressor achieved 1.73 times higher performance than the implementation without the compression.

In summary, the dissertation presents that proposed bandwidth compressor can improve the computational performance of the FPGA-based stream computing. Even if the I/O bandwidth

can not physically increased, the bandwidth compressor provides the solution to enhance the bandwidth in exchange for small hardware resource. The new generation FPGAs will have the much larger amount of hardware resources but the growth of I/O bandwidth will be relatively small. In such cases, the bandwidth compression technique presented in this study is useful for more applications to enjoy the hardware resources for high-performance computation with the physically-limited I/O bandwidth. improvement because the proportion of the hardware resource to the memory bandwidth will not change dramatically. In addition, the bandwidth compression can be used for not only the memory system but also the inter communication network. We are sure that the proposed technique push the FPGA-based high-performance computation to more popular and feasible solutions.

Bibliography

[1] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of AFIPS joint computer conference*, pages 483–485, April 1967.

[2] Jens Harting, Jonathan Chin, Maddalena Venturoli, and Peter V. Coveney. Large-scale lattice boltzmann simulations of complex fluids: advances through the advent of computational grids. *Philosophical Transactions of Royal Society A*, 363:1895–1915, August 2005.

[3] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Distributed resource discovery in large scale computing systems. In *Proceedings of the 2005 Symposium on Applications and the Internet*, pages 320–326, January 2005.

[4] Eric E. Schadt, Michael D. Linderman, Lawrence Lee, and Garry P. Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657, September 2010.

[5] Subhash Saini, Dale Talcott, Dennis Jespersen, Jahed Djomehri, Haoqiang Jin, and Rupak Biswas. Scientific application-based performance comparison of sgi altix 4700, ibm power5+, and sgi ice 8200 supercomputers. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, November 2008.

[6] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *Proceedings of 9th International Meeting on High-Performance Computing for Computational Science*, pages 1–25, July 2010.

[7] Satoshi Miyake, Hironori Miyazawa, and Satoru Yamamoto. Unsteady wet-steam flows through low pressure turbine final three stages considering blade number. In *Proceedings of*

ASME Turbo Expo 2015: Turbine Technical Conference and Exposition, pages 1–11, June 2015.

[8] Mitsuo Yokokawa, Fumiyoshi Shoji, Atsuya Uno, Motoyoshi Kurokawa, and Tadashi Watanabe. The k computer japanese next-generation supercomputer development project. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pages 371–372, 2011.

[9] Yuuichirou Ajima, Tomohiro Inoue, Shinya Hiramoto, and Toshiyuki Shimizu. Tohu: interconnect for the k computer. *Fujitsu Scientific and Technical Journal*, 48(3):280–285, July 2012.

[10] top500.org. Top500 lists june 2011. `http://www.top500.org`, 2015. [Online; accessed 25-november-2015].

[11] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. *High Performance Computing for Computational Science VECPAR 2010*, 6449, 2011.

[12] open supercomputer.org. hpci-roadmap. `http://open-supercomputer.org/wp-content/uploads/2012/03/hpci-roadmap.pdf`, 2012. [Online; accessed 15-february-2016].

[13] Doug Burger, James R. Goodman, and Alain Kagi. Memory bandwidth limitations of future microprocessors. In *Proceedings of 23rd Annual International Symposium on Computer Architecture*, pages 78–89, May 1996.

[14] David Wonnacott. Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In *Proceedings of 14th International Symposium on Parallel and Distributed Processing*, pages 171–180, 2000.

[15] Chen Ding and Ken Kennedy. The memory of bandwidth bottleneck and its amelioration by a compiler. In *Proceedings of 14th International Symposium on Parallel and Distributed Processing*, pages 181–189, 2000.

[16] Reiner Hartenstein. A decade of reconfigurable computing: a visionary retrospective. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 642–649, 2001.

[17] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, April 2009.

[18] Aaron Sawdey, Matthew O ' Keefe, and Rainer Bleck. The design, implementation, and performance of a parallel ocean circulation model. In *Proceedings of 6th ECMWF Workshop on the Use of Parallel Processors in Meteorology*, pages 523–550, 1995.

[19] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of 22nd International Conference on Field Programmable Logic and Applicaions*, pages 5–12, 2012.

[20] Kentaro Sano, Wang Luzhou, Yodhiaki Hatsuda, Takanori Iizuka, and Satoru Yamamoto. Fpga-array with bandwidth-reduction mechanism for scalable and power-efficient numerical simulations based on finite difference methods. *ACM Transactions on Reconfigurable Technology and Systems*, 3(4), November 2010.

[21] Michael Pellauer, Michael Adler, Michel Kinsy, Angshuman Parashar, and Joel Emer. Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing. In *Proceedings of High Performance Computer Architecture (HPCA)*, pages 406–417, February 2011.

[22] Yoshiaki Kono, Kentaro Sano, and Satoru Yamamoto. Scalability analysis of tightly-coupled FPGA-cluster for lattice boltzmann computation. In *Proceedings of the 22nd International Conference on Field-Programmable Logic and Applications*, pages 120–127, August 2012.

[23] Doris Chen and Deshanand Singh. Using opencl to evaluate the efficiency of cpus, gpus and fpgas for information filterling. In *Proceedings of 6th ECMWF Workshop on the Use of Parallel Processors in Meteorology*, pages 5–12, 2012.

[24] David B. Thomas, Lee Howes, and Wayne Luk. A comparison of cpus, gpus, fpgas, and massively parallel processor arrays for random number generation. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 63–72, 2009.

[25] Zheming Jin and Jason D. Bakos. Memory interface design for 3d stencil kernels on a massively parallel memory system. *ACM Transactions on Reconfigurable Technology and Systems*, 8(4), September 2015. Article 24.

[26] Kazuya Katahira, Kentaro Sano, and Satoru Yamamoto. FPGA-based lossless compressors of floating-point data streams to enhance memory bandwidth. In *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors*, pages 246–253, July 2010.

[27] Kentaro Sano, Kazuya Katahira, and Satoru Yamamoto. Segment-parallel predictor for FPGA-based hardware compressor and decompressor of floating-point data streams to enhance memory i/o bandwidth. In *Proceedings of the Data Compression Conference*, pages 416–425, March 2010.

[28] C. L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences, Elsevier*, 275:314–347, August 2014.

[29] David Salomon. *Data Compression: The Complete Reference.* Springer-Verlag, London, 4th. edition, 2007.

[30] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital System Research Center, 1994.

[31] Terry A. Welch. A technique for high-performance data compression. *IEEE Computer.*

[32] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Infromation Theory*, 23(3):337–343, May 1977.

[33] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[34] Robert M. Fano. The transmission of information. Technical report, Research Laboratory of Electronics, Massachusetts Institute of Technology, 1949.

[35] David A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the I.R.E.*, pages 1098–1101, September 1951.

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

[37] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*.

[38] Nathaniel Fout and Kwan-Liu Ma. An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Transactions on Visualizaton and Computer Graphics*, 18(12):2295–2304, December 2012.

[39] Shirou Maruyama, Hiroshi Sakamoto, and Masayuki Takeda. An online algorithm for lightweight grammar-based compression. *Algorithms 2012*, 5:214–235, April 2012.

[40] Shirou Maruyama, Masaya Nakahara, Naoya Kishiue, and Hiroshi Sakamoto. Esp-index: A compressed index based on edit-sensitive parsing. *Journal of Discrete Algorithms*, 18:100–112, 2013.

[41] Shinichi Yamagiwa, Koichi Marumo, and Hiroshi Sakamoto. Stream-based lossless data compression hardware using adaptive frequency table management. *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, 9495:133–146, January 2016.

[42] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. Fast lossless compression of scientific floating-point data. In *Proceedings of Data Compression Conference*, pages 133–142, March 2006.

[43] Bharat Sukhwani, Bulent Abali, Bernard Brezzo, and Sameh Asaad. High-throughput, lossless data compression on FPGAs. In *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 113–116, May 2011.

[44] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visual and Computer Graphics*, 12(5):1245–1250, September 2006.

[45] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22(3):343–348, September 2003.

[46] Jae S. Lim and Alan V. Oppenheim. Enhancement and bandwidth compression of noisy speech. *Proceedings of the IEEE*, 67(12):1586–1604, December 1979.

[47] Donald J. Healy and O. Robert Mitchell. Digital video bandwidth compression using block truncation coding. *IEEE Transactions on Communications*, COM-29(12):1809–1817, 1981.

[48] R. B. Tremaine, P. A. Franaszek, J. T. Robinson, C. O. Schulz, T. B. Smith, M. E. Wazlowski, and P. M. Bland. Ibm memory expansion technology (mxt). *IBM Journal of Research and Development*, 45(2):271–285, March 2001.

[49] Peter A. Franaszek, Luis A. Lastras-Montaño, Song Peng, and John T. Robinson. Data compression with restricted parsings. In *Proceedigns of the Data Compression Conference*, 2006.

[50] Hisanobu Tomari, Mary Inaba, and Kei Hiraki. Compressing floating-point number stream for numerical applications. In *2010 First International Conference on Networking and Computing*, pages 112–119, November 2010.

[51] Carlos Angulo J., Carlos Fajardo A., Oscar M. Reyes., and Javier Castillo V. Fpga implementation of a huffman decoder for high speed seismic data decompression. In *Proceedings of the Data Compression Conference (DCC)*, page 396, March 2014.

[52] Didier Keymeulen, Nazeeh Aranki, Alireza Bakhshi, Huy Luong, Charles Sarture, and David Dolman. Airborne demonstration of fpga implementation of fast lossless hyperspectral data compression system. *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 278–284, July 2014.

[53] Maxeler Technologies Ltd. `https://www.maxeler.com/`, 2016. [Online; accessed 15-february-2016].

[54] Oliver Pell and Stephen Girdlestone Henning Meyer. Sistems and methods for data compression and parallel, pipelined decompression. *United States Patent*, (US 8,847,798 B2), September 2014.

[55] Amir Said and William A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 6(3):243–250, June 1996.

[56] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM - Special issue on digital*, 34(4):30–44, April 1991.

[57] Kentaro Sano, Oskar Mencer, and Wayne Luk. FPGA-based acceleration of the lattice boltzmann method. In *Proceedings of the International Conference on Parallel Computational Fluid Dynamics*, May 2007. CDROM (paper-041).

[58] Kentaro Sano, Oliver Pell, Wayne Luk, and Satoru Yamamoto. FPGA-based streaming computation for lattice boltzmann method. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pages 233–236, December 2007.

[59] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink. Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869–877, January 2005.

[60] Martin Burtscher and Paruj Ratanaworabhan. FPC: a high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computer*, 58(1):18–31, January 2009.

[61] bzip.org. bzip2. `http://www.bzip.org`, 2016. [Online; accessed 15-february-2016].

[62] Martin Isenburg, Ioannis Ivrissimtzis, Stefan Gumhold, and Hans-Peter Seidel. Geometry prediction for high degree polygons. In *Proceedings of the 21st Spring Conference on Computer Graphics*, pages 147–152, 2005.

[63] Vadim Engelson, Dag Fritzson, and Peter Fritzson. Lossless compression of high-volume numerical data from simulations. In *Proceedings of Data Compression Conference (DCC)*, pages 574–586, September 2000.

[64] Altera Corporation. Qsys - altera ' s system integration tool. `https://www.altera.com/products/design-software/fpga-design/quartus-prime/quartus-ii-subscription-edition/qts-qsys.html`. [Online; accessed 15-february-2016].

[65] mpfr.org. The gnu mpfr library. `http://www.mpfr.org`, 2016. [Online; accessed 15-february-2016].

[66] George A. Constantinides, Nicola Nicolici, and Adam B. Kinsman. Numerical data representations for fpga-based scientific computing. *IEEE Design Test of Computers*, 28(4), May 2011.

[67] Kentaro Sano, Takanori Iizuka, and Satoru Yamamoto. Systolic architecture for computational fluid dynamics on FPGAs. In *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 107–116, April 2007.

[68] Tomohiro Ueno, Yoshiaki Kono, Kentaro Sano, and Satoru Yamamoto. FPGA-based implementation of compact compressor and decompressor of floating-point data-stream for bandwidth reduction. In *Proceedings of the 2012 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'12)*, July 2012.

[69] Tomohiro Ueno, Yoshiaki Kono, Kentaro Sano, and Satoru Yamamoto. Parameterized design and evaluation of bandwidth compressor for floating-point data streams in fpga-based custom computing. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*, pages 90–102, March 2013.

[70] Han gil Moon, Jeong il Seo, Seungkwon Baek, and Koeng-Mo Sung. A multi-channel audio compression method with virtual source location information for mpeg-4 sac. *IEEE Transactions on Consumer Electronics*, 51(4):1253–1259, November 2005.

[71] Yodchanan Wongsawat, Soontorn Oraintara, Toshihisa Tanaka, and K. R. Rao. Lossless multi-channel eeg compression. 2006.

[72] Kentaro Sano. Dsl-based design space exploration for temporal and spatial parallelism of custom stream computing. In *Proceedings of the 2nd International Workshop on FPGAs for Software Programmers*, pages 29–34, September 2015.

Published Papers

Reviewed Papers

[1] Tomohiro Ueno, Yoshiaki Kono, Kentaro Sano, Satoru Yamamoto, Implementation and Evaluation of FPGA-based Compressor and Decompressor of Floating-Point Data-Stream for Bandwidth Reductionh, Proceedings of The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA12), CSREA Press, pp.119-126, July, 2012 (chapter 2)

[2] Tomohiro Ueno, Yoshiaki Kono, Kentaro Sano, Satoru Yamamoto, Parameterized Design and Evaluation of Bandwidth Compressor for Floating-Point Data Streams in FPGA-based Custom Computing, Reconfigurable Computing: Architectures, Tools and Applications, Springer, pp.90-102, 2013 (chapter 2)

[3] Kentaro Sano, Yoshiaki Kono, Hayato Suzuki, Ryotaro Chiba, Ryo Ito, Tomohiro Ueno, Kyo Koizumi and Satoru Yamamoto, Efficient Custom Computing of Fully-Streamed Lattice Boltzmann Method on Tightly-Coupled FPGA Cluster, Proceedings of International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, Volume 41 Issue 5, pp.47-52, December, 2013 (chapter 4)

[4] Tomohiro Ueno, Ryo Ito, Kentaro Sano, Satoru Yamamoto, Bandwidth Compression of Multiple Numerical Data Streams for High Performance Custom Computing, Proceedings of The 25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP2014), pp.190-191, June 18-20, 2014 (chapter 3 and 4)

[5] Kentaro Sano, Hayato Suzuki, Ryo Ito, Tomohiro Ueno, Satoru Yamamoto, Stream Processor Generator for HPC to Embedded Applications on FPGA-based System Platform,

Proceedings of International Workshop on FPGAs for Software Programmers (FSP 2014), pp.43-48, August 2014 (chapter 4)

[6] Tomohiro Ueno, Kentaro Sano, Satoru Yamamoto, Memory Bandwidth Compressor for FPGA-based High-Performance Custom Stream, ACM Transactions on Reconfigurable Technology and Systems, submitted (chapter 3 and 4)

[7] 上野　知洋, 佐野　健太郎, 山本　悟, 予測残差長の量子化を用いた可逆データ圧縮ハードウェアの性能評価, 2016 年ハイパフォーマンスコンピューティングと計算科学シンポジウム (chapter 3)

Technical Papers

[8] 上野　知洋, 高野 芳彰, 佐野　健太郎, 山本　悟, 異なるビット幅を扱う浮動小数点データストリーム圧縮ハードウェアの性能評価, 電子情報通信学会リコンフィギャラブルシステム研究会，2013 年

[9] 上野　知洋, 佐野　健太郎, 山本　悟, 複数データストリームの帯域向上のための圧縮ハードウェアの実装と評価， 電子情報通信学会リコンフィギャラブルシステム研究会，2013 年

[10] 上野　知洋, 伊藤 涼, 佐野　健太郎, 山本　悟, 複数ストリームのための帯域圧縮ハードウェアの実装と評価， 電子情報通信学会リコンフィギャラブルシステム研究会，2014 年

[11] 上野　知洋, 佐野　健太郎, 山本　悟, 予測残差長の偏りを利用した小面積帯域圧縮ハードウェアの提案， 電子情報通信学会リコンフィギャラブルシステム研究会，2015 年

[12] 上野　知洋, 佐野　健太郎, 山本　悟, メモリ帯域圧縮ハードウェアを用いた数値計算の高性能化， 第 151 回 HPC 研究発表会，2015 年

[13] 上野　知洋, 佐野　健太郎, 山本　悟, 帯域圧縮による数値計算ハードウェアの高性能化，第 29 回数値流体力学シンポジウム，2015 年

Acknowledgments

I am deeply grateful to my doctoral adviser Prof. Kentaro Sano for his helpful guidance and encouragement over the past 7 years. He gave me many advices from various aspects which are not only for my study but also for my life. Without his guidance and persistent help, this thesis would not have been possible.

I am deeply grateful to Prof. Hiroaki Kobayashi, Prof. Ayumi Shinohara and Prof. Satoru Yamamoto for suggestions and advices about this study and dissertation. In addition, I would like to thank Prof. Satoru Yamamoto for his supports and advice for fluid dynamics in this study.

I would like to express my gratitude to Dr. Takashi Furusawa for his support and encouragement of my study and life.

I would like to thank Mr. Daichi Tanaka and Mr. Kohei Nagasu for their support of my study.

I would like to thank Japan Society for the Promotion of Science for generous support.

Finally, I am deeply grateful to my parents and my family for their support and encouragement.