Exploiting Trade-offs in Memory, Storage, and Communication Performance and Accuracy in High-Capacity Computing Systems

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Qianqian Fan

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. David J. Lilja
Prof. Sachin S. Sapatnekar

August, 2019

Acknowledgements

First, I would like to express my sincere gratitude to my advisors, Prof. David J. Lilja and Prof. Sachin S. Sapatnekar, for the continuous support of my Ph.D study. Their patience, guidance and encouragement helped me all the time of research and overcome challenges. As my incredible advisors, they taught me how to create the big picture thinking and also pay attention to the detail in real work. They have played an important role in my educational development, and I feel very fortunate to be associated with such talented mentors.

Besides my advisors, I would like to thank the rest of my thesis committee: Prof. Ulya Karpuzcu and Prof. Antonia Zhai, for their insightful comments and suggestions.

I am grateful for the resources from the University of Minnesota Supercomputing Institute and the support from C-SPIN, a Semiconductor Research Corporation program, and from Nation Science Foundation grant no. CCF-1438286.

I want to thank Zhaoxin, Farhana, Vivek, Cong, Bingzhe, and Amogh for being not only offering guidance on my research, but also help and advice on my life and career. Many thanks to Deepashree, Meghna, Masoud, Susmita, Tonmoy, Vidya, Kishor, Yaobin, Jinfeng, Mohammad, Arvind, and Geraldo for being wonderful labmates as well. I want to thank my friends Qiannan, Dingyi and Jianjun for the get-togethers and happy hours. Finally, I would like to thank my husband, Tengtao, without whose support and constant encouragement, I could not have finished my PhD.

Dedication

To my parents and my husband.

Abstract

Error resilient applications are becoming more common in large-scale computing systems. These types of applications introduce the possibility of balancing cost and performance in new ways by trading-off output quality with performance. To exploit this new opportunity, this thesis introduces three approximation techniques that trade-off the accuracy of memory, storage, and communication for gains in efficiency and performance in large-scale, high-capacity computing systems.

First, we employ the notion of approximate memory, which exploits the idea that some memory errors are not only nonfatal, but can be leveraged to enhance power and performance with minimal loss in quality. The traditional approach for increasing yield in large memory arrays has been to eliminate all hard errors using repair mechanisms. However, the cost of these mechanisms can become prohibitive at higher error rates. Instead of completely repairing faulty memories, we introduce new approximate memory repair mechanisms that only partially repair both CMOS DRAMs and STT-MRAMs. By combining redundant repair with unequal protection, such as skewing the limited spare elements available for repairing faults towards the k most significant bits, and a hybrid bit-shuffling and redundant repair scheme, the new mechanisms maintain excellent output quality while substantially reducing the cost of the repair mechanism, particularly for increasingly important cluster faults.

Second, we investigate the use of approximate storage, which is defined as cheaper, lower reliability storage with higher error rates. In the past few years, ever-increasing amounts of image data have been generated by users globally, and these images are routinely stored in cold storage systems in compressed formats. Since traditional JPEG-based schemes that use variable-length coding are extremely sensitive to error, the direct use of approximate storage results in severe quality degradation. We propose an error-resilient adaptive-length coding (ALC) scheme that divides all symbols into two classes, based on their frequency of occurrence, where each class has a fixed-length codeword. This provides a balance between the reliability of fixed-length coding schemes, which have a high storage overhead, and the storage-efficiency of Huffman coding schemes, which show high levels of error on low-reliability storage platforms. Further, we use data partitioning to determine which bits are stored in approximate or reliable storage to lower the overall cost of storage. We show that ALC can be used with general non-volatile storage, and can substantially reduce the total cost compared to traditional JPEG-based storage.

Finally, approximate communication as a new opportunity has arisen for improving the communication efficiency in parallel systems, which can significantly reduce the amount of communication time by transmitting partial or imprecise messages. Communication overheads in distributed systems constitute a large fraction of the total execution time, and limit the scalability of applications running on these systems. We propose a Discrete Cosine Transform (DCT)-based approximate communication scheme that takes advantage of the error resiliency of several widely-used applications, and improves communication efficiency by substantially reducing message lengths. Our scheme is implemented into the Message Passing Interface (MPI) library. When evaluated on several representative MPI applications on a real cluster system, it is shown that our approximate communication scheme effectively speeds up the total execution time without much loss in quality of the result, even accounting for the computational overhead required for DCT encoding.

In summary, the partial-repaired memory scheme, error-resilient ALC scheme, and DCT-based approximate communication scheme are proposed in this thesis and allow the system to maintain an acceptable output quality while substantially reducing the cost of the system.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A growing set of applications that involve social media, machine learning, and scientific computing gain prominence. A common characteristic of these applications is that an approximate or less-than-optimal result is sufficient. With these error resilient applications becoming more popular, a new trade-off between output quality and performance is introduced. Therefore, approximation research has recently emerged as a promising approach to the energy-efficient design of systems [1, 4–16]. In this thesis, three approximation-based techniques are investigated, which trade off accuracy of memory, storage, and communication for gains in efficiency and performance.

Approximate memory: Several decades of device scaling has substantially reduced the cost per bit of memories, yet yield loss still remains a significant issue. Furthermore, process technology is still evolving for emerging memories, such as spin-transfer-torque magnetic RAMs (STT-MRAMs), which further impacts their yield. Hard errors in CMOS dynamic RAMs (DRAMs) typically dominate over soft errors [17, 18], while the emerging nature of STT-MRAM technology suggests that process-related hard errors are more likely than transient retention errors. The traditional approach to handling memory faults, and thereby improving yield, has been to root them out completely using various off-line redundancy mechanisms, such as spare rows and columns [19–21]. These repair schemes define yield as the probability that no faults occur [22]. As the probability of failure increases, though, these conventional redundant repair approaches will lead to tremendous increases in overhead to repair the memory to 100 percent correctness.

For a set of emerging applications, a new opportunity has arisen for coping with errors instead of completely repairing them. We employ the notion of approximate memory [5], which exploits the idea that some memory errors are not only nonfatal, but can be leveraged to enhance power and performance with minimal loss in quality [6]. Our work addresses image application for approximate memory evaluation, which shows inherent error-tolerance.

The economics of memory dictates that the cost of memory increases nonlinearly with reliability. As a result, memories with few errors are very expensive, and the cost reduces greatly with higher error rates. We develop an approach to employ these lower-cost, higher-error memories for image applications.

In our work, we evaluate a set of approximate memory repair mechanisms that only partially repair memory to reduce the cost of the repair mechanism at the expense of some loss in output quality. We extend prior conventional memory repair methods that use limited spare elements to several new schemes, including the k-MSB redundant repair scheme that skews the redundant repair elements to the first k most significant bits of a byte, and compressed bit-shuffling which reduces the overhead of prior bit-shuffling approaches [6] while maintaining comparable quality. We further show that the bit-shuffling related schemes do not perform well in the presence of row, column, and cluster faults, which are important in real-world applications [20,23]. We also propose a new hybrid bit-shuffling and redundant repair scheme. Our work ensures that even these lower-cost, lower-reliability memories produce results where the quality degradation is controlled, leading to an opportunity to trade off cost with memory quality.

Approximate storage: Approximation can also enable more effective use of long-term storage resources [24, 25]. The quantity of image data on cloud-based storage has skyrocketed in recent years. For example, in 2019 annual trends report, over 1 trillion images were uploaded to the social media for one year [26], a number that has surely increased since. Furthermore, images are currently stored in most sites at multiple resolutions to support different devices and contexts, further increasing storage overheads [27]. These uploads require a significant amount of storage, which can be expensive. Therefore, cost-effective image storage is an active field of research. In [28], they reduced the metadata of photo to obtain lower cost storage with higher throughput. Progressive JPEG with customized encoding parameters for dynamic resizing was used to reduce bandwidth and storage overheads in [27]. Recent works applied cheaper, lower-reliability image and video storage at dramatically lower cost [1, 7–9].

Individual images are typically saved in compressed form in long-term storage rather than in a raw format where every pixel is stored. Apple is adopting the High Efficiency Image File (HEIF) format as its new photo format [29], but this format is still only a small part of the market. Dropbox uses Lepton to re-compress JPEG files, which replaces the lowest layer of the baseline JPEG compression with a parallelized arithmetic code (variable-length) to improve speed [30]. The PTC encoding algorithm is used in [7] for the purpose of approximate storage, which is a seldom-used image format and has low compression efficiency. To reach our target, the JPEG encoding algorithm is a good candidate because of its popularity [31] and high compression ratio.

The JPEG standard uses a discrete cosine transform (DCT) and saves the coefficients of the dominant spatial frequencies using variable-length coding (VLC), a Huffman coding scheme that further improves compression efficiency [32]. Modern image storage providers, such as Google Photos, typically re-compress JPEG files to reduce the image size. While this loses some information, it does so without any perceptual difference [30, 33]. Similarly, when approximate storage is used, the new storage scheme should keep each encoded image with no visually perceptible loss in quality. However, an error in compressed image data can change key attributes of the VLC-encoded image, and even with variable error correction mechanism, can still result in obvious distortions in the image [1]. Even a single error could completely corrupt part of the image (Fig. 1.1(a)). The direct use of error-prone low-reliability storage for saving VLC-encoded JPEG images is thus risky as it can cause unacceptable levels of error. Therefore, despite their low cost, error-prone storage is not considered viable for storing compressed VLC-encoded images.

In this work, a novel adaptive-length coding (ALC) scheme is proposed to replace the error-sensitive Huffman coding in JPEG compression. Our ALC scheme is shown to be inherently resilient to errors. ALC divides all the symbols into two classes: more frequently occurring symbols, which are encoded to a shorter fixed length, and more infrequent symbols that are mapped to a longer fixed length code. Since the codeword in each class has the same length, ALC is an encoding scheme where every word uses one of two allowable lengths. This differs from Huffman coding as used in VLC where

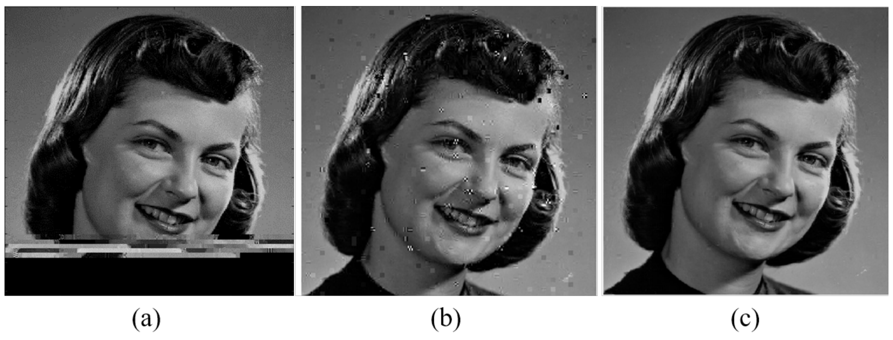


Figure 1.1: (a) VLC-based storage where a single error corrupts the the remaining blocks of image (b) VLC-based storage with variable error correction mechanism [1] under a 1% error rate, showing obvious degradation (c) Our ALC-based scheme with a same percentage of data stored in approximate storage as (b) under a 1% error rate, showing no visible quality loss.

each symbol could be mapped to a variable length encoding [32]. By limiting the number of code lengths to two, we obtain the best of both worlds, balancing the error-resilience of fixed-length encoding with the reduced storage needs of VLC. According to the statistics of an image, the algorithm adaptively and efficiently adds additional bits to compensate for the quality loss. We demonstrate that ALC maintains excellent compression efficiency while improving error resilience inherently. An example result is shown in Fig. 1.1(c).

The increased error resilience of ALC allows most of the data bits of an image to be stored in low-cost approximate storage, but some critical parts of the image may still require reliable storage. We develop a data partitioning scheme that segments the ALC-encoded data into reliable and approximate storage, minimizing the total cost of storage with the quality degradation constrained. From source-channel separation theorem [34] aspect, we can optimize ALC and ECC of storage separately and can simply combine them in a cascaded manner. In this work, we mainly focus on the compressed image data instead of other applications with high error resilience [35]. Our scheme can be implemented for JPEG files recompression as [30], which only replace Huffman coding with error-resilient ALC compared to conventional JPEG scheme. Furthermore, ALC scheme is compatible with more sophisticated JPEG techniques, e.g. JPEGmini, which use image-specific quantization matrix to further improve compression efficiency [36].

Approximate communication: Nowadays, a growing number of applications are implemented by using parallel computation on a computer cluster to achieve better performance. These applications of high-performance computing (HPC) can distribute computation across many nodes, and each node processes only a part of the total workload. During the operation, the work done by each node is not independent, and some data may have to be transferred among nodes for additional processing and analysis. The message passing interface (MPI) is a widely used communication protocol for cluster computation that provides a standard interface for communication [37].

In large-scale parallel systems, efficient communication is a major challenge for scalability [38]. Communication-intensive parallel applications transfer a large amount of data among nodes of a cluster via an interconnection network. In [2], the fraction of time spent on communication increased significantly with the number of processors for representative applications of HPC, as shown in Fig. 1.2. This large communication overhead limits the scalability of parallel applications.

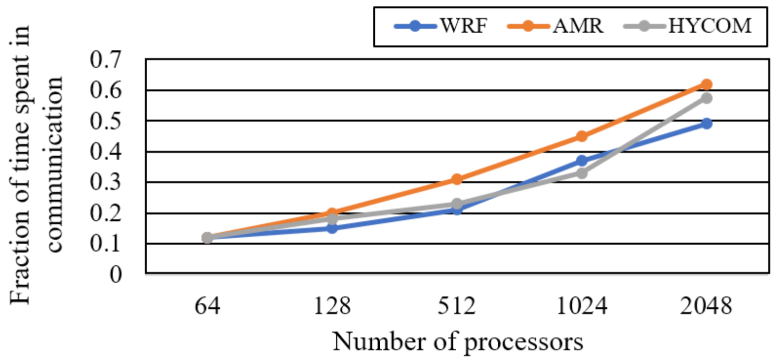


Figure 1.2: Fraction of time spent in communication as the number of processors increase for representative high performance computing applications [2].

With the growing popularity of error-resilient applications, a new trade-off between the quality and speed has been introduced. These error-resilient applications can improve the efficiency of the system while retaining an acceptable level of accuracy. Therefore, approximate communication [38] has arisen as a new opportunity for improving the efficiency of communication in parallel systems, which can significantly reduce the time needed for communication by transmitting partial or imprecise messages.

We propose a DCT-based new approximate communication scheme based on discrete cosine transform (DCT) that allows some errors during communication to substantially reduce overhead. In the proposed scheme, the subband decomposition [39] of the message is first used, and the original DCT with length N can be approximately computed by a half-length DCT. We then propose a fast and recursive DCT with a piecewise-constant approximation to speed-up processing while maintaining a small space overhead, especially for long DCT transformations. Moreover, the zero run-length coding scheme is used to improve the efficiency of compression. Specifically, some applications have similarities among messages at the same node. Two compression strategies are used here by making use of this characteristic to take differential analysis: compressing the entire message, or compressing the difference between the given message and a reference message. We implemented the DCT-based approximate communication scheme in the OpenMPI library, compiled the applications using this modified library, and executed them on a real distributed cluster system. Eight representative error-resilient MPI applications were used to explore the benefits of approximate communication. Several parameters can be tuned for each application to achieve its best performance. The performance of the MPI applications was evaluated on a real cluster system. Compared with schemes that apply other lossy compression schemes used in HPC applications, the results show that the DCT-based approximate communication scheme obtained a significant reduction in the cost of communication time with a smaller overhead in terms of the time needed for compression. For the communication-intensive applications, it is shown that our approximate communication scheme effectively speeds up the total execution time without much loss in quality of the result.

Thesis organization: In this thesis, we have conducted a detailed study of taking approximation in memory, storage, and communication process. The thesis is organized as follows:

- Chapter 2 presents a set of related works in these three related areas.
- Chapter 3 details the memory organization and describes fault models for DRAMs and STT-MRAMs. Next, a set of conventional and approximate repair schemes are discussed that are evaluated in this work. Finally, an experimental evaluation is conducted, followed by the results to show the cost-quality trade-offs of approximate memory repair mechanisms for image data.

- Chapter 4 details the fundamentals and error resilience limitations of JPEG-based storage schemes. Next, the adaptive-length coding (ALC) scheme and partitioning data between reliable and approximate storage strategy are discussed.

- Chapter 5 details the MPI-based error-resilient applications and high message energy compaction of DCT. The DCT-based approximate communication scheme and differential analysis of messages strategy are proposed.

- Chapter 6 presents a final discussion of the analysis presented in the thesis and draws the conclusion of the thesis.

Chapter 2

Related Work

Approximate computing is an emerging design approach that is able to exploit the error tolerance of some applications for gains in performance and efficiency. Approximation research has explored various fields with different trade-offs [40].

2.1 Approximate memory

Several works have proposed some techniques to present the idea that memory errors are not only non-fatal, but can be leveraged to enhance power and performance with minimal loss in quality. Flikker [41] reduces refresh rates in DRAM memories to reduce energy consumption at the cost of approximate data. By exploring that skewing errors towards the most significant bit (MSB) of a word tend to be more serious than those towards the least significant bit (LSB), a recently proposed scheme [6] detects MSB errors and rotates the bits of a word to store LSBs in these bit positions instead. The concept of approximate memory has also been applied for non-volatile memories. In [8], quality-energy tradeoff in STT-RAM is explored to gain some improvement in energy efficiency by tuning the supply voltage. The density can be improved for a fixed power budget for a given MLC memory at the cost of approximate writes [5]. In this work, we focus on the use of redundant memory repair mechanisms to realize approximate memory with substantially reducing the cost of the repair mechanism, particularly for increasingly important cluster faults.

2.2 Approximate storage

Recent works applied cheaper, lower-reliability image and video storage at dramatically lower cost [1, 7–9]. This concept has been explored for raw images [9] and client devices presuming the availability of a high-fidelity copy in the cloud [7]. However, these prior works do not address the issue of reducing the cost of the long-term storage in the cloud, which is the subject of our work.

Individual images are typically saved in compressed form in long-term storage and the JPEG encoding algorithm is a good candidate because of its popularity [31] and high compression ratio. However, the conventional JPEG is extremely sensitive to error. To solve this problem, the networking community has proposed a set of techniques for image transmission over lossy networks to improve the error resilience of image coding.

From the point of view of source coding, to overcome the resynchronization limitation of VLC, the error resilient entropy coding (EREC) method was proposed in [42]. In this code, blocks of various lengths are reorganized into fixed-length blocks to prevent error propagation beyond block boundaries. The approaches in [43–45] adopt the same idea to improve the error resilience of JPEG by preventing the errors propagating across the block boundaries. However, this cannot prevent quality degradation within blocks at the higher error rates seen in low-cost storage. In [46], the embedding and side-match vector quantization (VQ) is used to conceal the corrupted block, but their discussion shows that it is challenging for this scheme to achieve acceptable quality degradation under large error rates. The Hybrid Variable Length Code (HVLC) [47] uses multiple VLC coding structures to reduce the error propagation distance within one codeword, but the boundaries of the codeword are based on VLC, and if errors are introduced in these boundary code bits, the image data can be corrupted.

Channel coding also can be applied intelligently to improve the error resilience. An error-resilient unequal protection can robustly cope with packet loss in transmission [1, 48]. The Wyner-Ziv Error-Resilient scheme emphasizes protecting the Region of Interest area in the frame [49]. However, the goal of all these methods localize the error within blocks or larger segments and to prevent it from impacting the entire bit-stream. As we stated earlier, the impact of this block-based error on perceptual quality is still large for purposes of image storage, as shown in Fig. 1.1(b).

Instead of using VLC, SoftCast [50] compresses data by discarding zero and near-zero DCT components. The codewords preserve the numerical properties of the original pixels, so the error resilience of image or video coding can be improved compared to the VLC-based scheme. Fixed-length coding (FLC) [51] ensures that errors do not propagate beyond the corrupted codeword, but at a much lower compression efficiency than VLC. Variable-to-Fixed-length codes, e.g. the Tunstall coding algorithm [52], map symbols to a fixed number of bits and can also avoid resynchronization issues in VLC. However, the compression efficiency of Tunstall coding is severely affected by rarely-used symbols compared with Huffman coding. No existing scheme can overcome the error sensitivity of VLC with comparable compression efficiency.

2.3 Approximate communication

To improve the efficiency of communication in parallel applications, several works have proposed some techniques [38] to reduce the total number of messages and the size of each message.

Reducing total number of messages: Relaxing the data dependency (e.g., read-after-write data dependency) in parallel applications can help reduce the time needed for communication among the threads [53]. Thread fusion was used in [54]. It assumes that the outputs of adjacent threads are similar to one another, and uses the output of one thread to represent others. To reduce the cost of communication based on more communication patterns, Paraprox [55] used a subset of values in the input array to construct an approximate version of the input to reduce communication among nodes. These strategies apply approximate communication, where the accuracy of the result is traded for higher speed in parallel applications. However, a non-negligible error is introduced by directly discarding some communication.

Reducing the size of each message: Compression has been commonly used for reducing the cost of communication by reducing the length of messages. The transmission time of MPI message increases with message size [56] [57].

By taking advantage of the similarities between spatial and temporal neighbors, Ref. [58] evaluated a method of lossless compression on a large-scale climate simulation dataset. Because different compression algorithms deliver varying performance, the adaptive compression system [59,60] can adaptively select the compression algorithm to compress the given message. The selection is based on internal models developed by the authors or previous experimental results to estimate the compression performance of different algorithms. However, these dynamic strategies require the collection of a large volume of performance data for different compression algorithms to build the selection criteria. Moreover, once the pattern of a message changes, the effectiveness of estimation of these compression algorithms significantly decreases. The lossless compression algorithm can preserve all information but perfect communication is not necessary for error-tolerant applications. Lossy compression on messages can perform better in terms of efficiency of compression to further reduce the cost of communication.

Instead of using replicates to represent messages transmitted by nodes, lossy compression provides an option using the notion of approximate communication. Lossy compression algorithms can be divided into prediction-based compression and transform-based compression. For prediction-based lossy compression algorithms, ISABELA [61] applies B-splines curve fitting to predict the sorted input data but its efficiency of compression is limited because each data item has an index to record its position in the original unsorted input, and this spatial index map occupies a large part of the final compressed output. SZ [62] uses a multidimensional model to predict the next data point and designs adaptive error-controlled quantization for each point value. For transform-based algorithms, ZFP [63] develops an orthogonal block transform-based compression algorithm to compress 3D floating-point data. Wavelet transformation has also been applied to HPC applications [64,65]. SSEM [65] first applies 2D wavelet transformation and compresses only the high-frequency band by quantization to maintain the quality of the results. These lossy compression algorithms have been evaluated on scientific data with an emphasis on pointwise compression error between the original and the reconstructed datasets. Instead of considering point-to-point errors in each message transmitted through intermediates in MPI applications, we focus on obtaining a final output of quality comparable to that of the original by using a lossy runtime compression algorithm.

Chapter 3

Cost-Quality Trade-offs of Approximate Memory Repair Mechanisms

In this chapter, we focus on the use of schemes that repair hard errors through the addition of redundant hardware. As mentioned in Chapter 1, the cost of the conventional redundant repair mechanisms become prohibitive at high error rates to eliminate all hard errors in memory arrays. Therefore, in addition to evaluating conventional repair methods, we propose two nonuniform protection techniques to improve the repair efficiency that only partially repair faulty memories [66]. In our work, we consider two types of memories, CMOS based DRAMs and spin-transfer-torque magnetic RAMs (STT-MRAMs) in the evaluation.

3.1 Memory organization and fault models

3.1.1 Data array organization

Our memory model is based on the structure in CACTI [3], as depicted in Fig. 3.1. The memory is organized into multiple identical banks, each of which can be accessed in parallel, chosen by the bank address input. A bank is composed of multiple identical subbanks, which are further divided into multiple mats. All mats are activated during an access, and bits within a word are interleaved across mats. A mat has four subarrays that share the predecoding and decoding circuitry, and each subarray has its own associated peripheral circuitry.

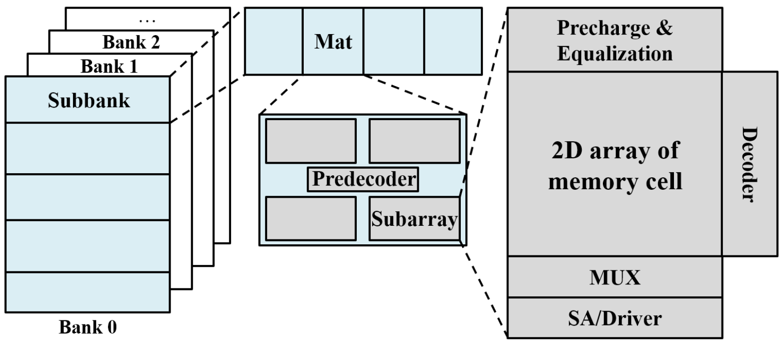


Figure 3.1: Organization of the memory array [3].

3.1.2 Fault models

Fault distribution

If λ is the mean number of faults in a memory array of size M, the failure probability of a bit-cell, $P_{cell} = \lambda/M$. The probability of n failing cells in the array follows the binomial distribution. This approximation holds for our typical use case where M is very large and P_{cell} is small [22]. A sample from this distribution provides the total number of hard faults in a specific memory array. In our experiments, we allocate this total number of faults to be *single cell failures* that affect an isolated cell, or *column failures*, *row failures*, and *cluster failures*, which affect an entire column, row, and cell cluster, respectively [67, 68], according to a set probability. We further set the fault type to a specific functional fault according to a user-specified probability, where the set of fault types for DRAMs and STT-MRAMs are described in the remainder of this section.

DRAM fault types

The functional degradations that could be brought about by a DRAM fault fall into several categories [69]. For *stuck-at faults* (SAFs), the value in the cell is fixed at logic

1 or logic 0, while for *stuck-open faults* (SOFs), the cell is completely inaccessible. In *transition faults* (TFs), a rising or falling transition cannot be realized. These faults are similar to SAFs, but the difference is that the stuck-at logic level is unknown. *Coupling faults* may be manifested as *inversion faults* (CFin), where a transition in one cell will invert one of its neighbors, *idempotent faults* (CFid), where a transition in one cell will set one of its neighbors to a fixed value, or *state faults* (CFst), where a specific value in one cell will set one of its neighbors to a fixed value. More comprehensive fault models [68] can also be applied but they are unlikely to introduce any meaningful changes for final results.

STT-MRAM fault types

Failures in STT-MRAMs [21] can be classified into persistent errors, which are similar to hard faults, and transient errors, which are similar to soft errors. We focus here on persistent errors, primarily those caused by process variations. The behavior of these errors can be expressed in several ways. *Transition faults* (TF0/TF1) are write errors that are caused by insufficient MTJ write current, which prevents a switching event from being completed. *Read disturb faults* (RDFs) are caused when the read current is too high and inadvertently flips the cell value during a read operation. *Incorrect read faults* (IRFs) occur when, due to insufficient read sense margin, an incorrect value is read from the cell based on its sensed value and the response of the sense amplifier.

3.2 Memory repair schemes

3.2.1 Conventional repair schemes

Two-dimensional (2D) redundant repair

Two-dimensional redundant repair, illustrated in Fig. 3.2, involves the insertion of spare rows and/or columns into a memory array. When a faulty cell is detected, a spare element may be used to replace it. Note that spare elements have the same failure probability as the main memory array, and that if the spare element has a fault, it cannot be used to repair faulty elements in the memory.

Given a fault map within a memory, which is typically obtained using a memory test, various algorithms are available to determine how a spare row/column can be used to replace the row/column in which the fault lies. In our experiments, we apply the widely used essential spare pivoting (ESP) redundancy analysis algorithm [70] for this purpose.

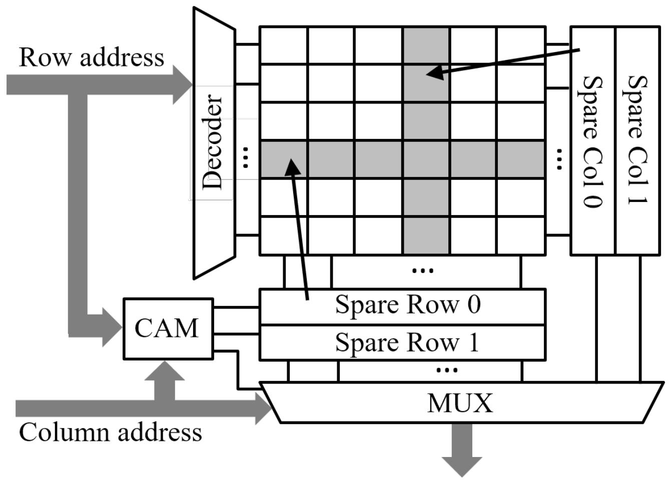


Figure 3.2: Two-dimensional redundant repaired memory.

Segmented memory repair

When an entire row (or column) is used to repair a faulty row (column) containing only one or a small number of faulty cells, a large number of spare rows and columns may be required to ensure full repair. Further flexibility may be provided by dividing the spare rows and columns into segments [20], as illustrated In Fig. 3.3. If a memory column has a single fault in a specific row, then only the segment of a spare column corresponding to that row is used, and other segments may be deployed to repair other faults.

3.2.2 Repair for approximate memories

The schemes in Section 3.2.1 were originally designed to replace faulty memory cells with spare functional cells with the goal of achieving full repair with high probability. However, for error-tolerant applications, full repair is not necessary and partial or approximate repair may be adequate. In this section, we present a set of schemes

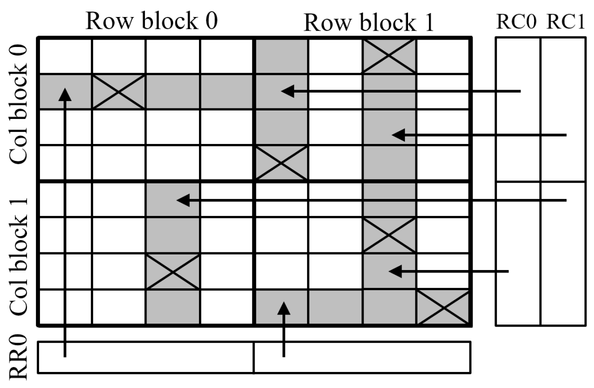


Figure 3.3: An example illustrating segmented redundancy for memory repair, where faults correspond to crossed-out cells.

that can be used to partially repair faults to provide an approximate memory for such applications.

For image compression applications, we will evaluate these schemes by using the power signal-to-noise ratio (PSNR) as a quality metric. The key idea of nonuniform protection is to protect some bits more carefully than others. For example, the role of higher-order bits is more significant than that of lower-order bits in determining a quality metric such as PSNR.

Limited spare rows and columns scheme

One extreme end of this partial redundancy continuum corresponds to *unprotected memory* with no redundancy whatsoever. This approach has zero overhead and can be expected to provide the lowest-quality result. At the other extreme is the notion of full repair, as discussed in Section 3.2.1. This approach will have the best quality, but at the highest cost. Intermediate points correspond to different cost-quality trade-offs. For approximate applications, we can limit the number of spare rows and columns of conventional repair schemes to reduce the area and delay overhead in the memory, while delivering adequate accuracy to maintain the quality metric.

k-MSB repair scheme

The k-MSB repair scheme hybridizes the redundant repair scheme with the notion of nonuniform protection. This scheme is based on the observation that the cost of providing protection increases with the number of bits, but its effectiveness in improving the system quality metric goes down steeply after the first few bits. Therefore, this method applies more spare elements to protect the first k MSBs of the byte possibly even achieving full repair for these bits. A lower level of protection could still be applied to the LSBs by using some of the spare elements since they may still be vulnerable to clustered faults. The number of available spare elements is determined by the probability of the clustered faults. As for the implementation, different threshold values of the ESP algorithm can be set for different levels of protection. For a low level of protection, a larger threshold allows the faulty rows, faulty columns, and other types of cluster faults to be prioritized for repair using these limited spare elements. As we will show, this scheme can greatly reduce the overhead with minimal quality impact.

Bit-shuffling scheme

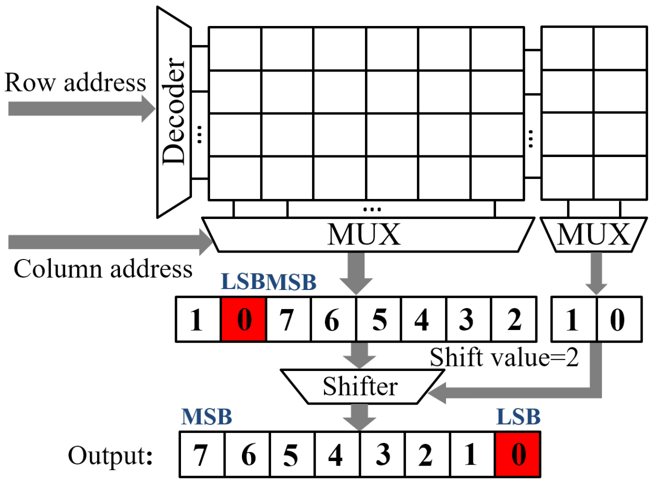


Figure 3.4: A schematic of the bit-shuffling scheme for the read operation.

The bit-shuffling scheme was proposed in [6]. Like our k-MSB method, it leverages the notion that higher-order bits are more critical to a computation than lower-order bits. This mechanism rotates the bits in a word in case of a fault to ensure that higher-order bits are protected. When data is read from or written into memory, it is rotated so that the higher-order bits are stored in a fault-free location, and the fault is effectively moved to a lower-order location.

For our image compression application, the length of each item of image data is 1 byte. Given a fault map, if the n^{th}-most significant bit has a fault, then the method uses a circular shifter to rotate the data to the left by n places as it is stored into memory. This operation implies that the LSB of this byte is placed into a faulty cell. A variation of the basic scheme is characterized by a parameter, n_{FM}, which decides the maximum number of shifts that are permitted. The largest value of a shift is limited to $2^{n_{FM}} - 1$, and therefore the $(2^{n_{FM}} - 1)$ highest-order bits are protected. For a byte, a value of 1, 2, or 3 for n_{FM} corresponds to a shift of up to 1 bit, 3 bits, or 7 bits. A lower value of n_{FM} reduces the area and delay overheads of the scheme but limits its correction capability.

Fig. 3.4 illustrates an implementation of this scheme. Here each row contains b bytes, and there are bn_{FM} columns to the right of the main storage array. For the image application, this configuration allows each byte to be rotated to preserve the correctness of its MSB. The MUX at right, below this auxiliary array, chooses the rotation requirements of the byte of interest within a line, and its select input is the column address. In this example, $n_{FM} = 2$, which enables the three MSBs to be protected. If the second MSB has a fault, then the shift value is two ($(10)_2$). While storing data, a right-circular shift translates the data by two bits to place the LSB in the faulty cell, which is shown as the stored value. The figure illustrates a read operation, where the opposite operation – a left circular shift – is performed to restore the MSB to its rightful location.

Compressed bit-shuffling scheme

The bit-shuffling scheme in [6] was implemented using an additional set of n_{FM} bits for each line. For the image compression application, the relevant RGB data is only a byte long, implying the overhead of $(b \cdot n_{FM})$ per line is prohibitive. The discussion in [6] did not encounter this problem since the word size was substantially larger, which allowed the overhead to be amortized over a larger number of data bits.

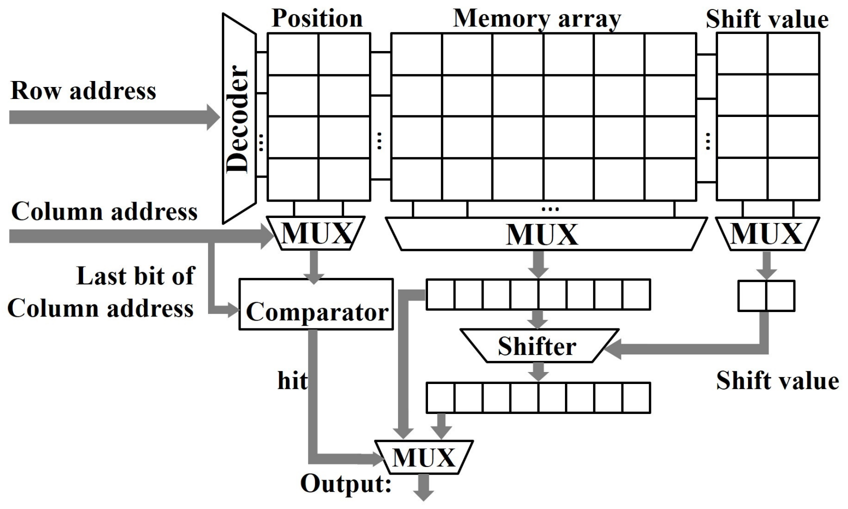


Figure 3.5: An improved implementation of the bit-shuffling scheme.

To overcome this large overhead, we propose a modification to this bit-shuffling scheme. In a realistic scenario where the number of faults is not large, most of the rows and columns will require no correction, implying that the rotation requirements stored in n_{FM} will be zero in most cases. Therefore, we propose a storage scheme with a compression rate of 2^q where a single shift value is shared by a set of 2^q bytes. As a result, the overhead of storing the rotation bits is reduced from bn_{FM} per line to $(b/2^q)n_{FM}$. The drawback of this case is that, if there are two faults within a block of 2^q words that share a shift value, only one can be corrected.

The hardware implementation of this scheme is illustrated in Fig. 3.5. An additional array is placed to the left of the memory array, storing the position within the 2^q-bit block of the word that has a fault, if any. As before, its shift value is stored to the right of the memory array. The column address, minus the bottom q bits, is applied to the MUXes connected to the *Position* and *Shift Value* arrays. The lower order bits represent the location of the byte within the line. If they match the position for that line, the corresponding shift value is used to rotate the stored data, if necessary.

Hybrid bit-shuffling and redundant repair scheme

A major drawback of bit-shuffling-based schemes is that they work well only for isolated faults. In reality, we may see row, column, or clustered faults [20,23] where faults strike multiple adjacent cells. The probability of having more than one fault per word is then greatly increased, even when words are interleaved, and bit-shuffling can be ineffective.

Redundant repair schemes with spare rows or columns are well suited to handle such errors, however. If there is a row/column error, the entire row/column is replaced. Therefore, we combine the redundancy and the bit shuffling schemes to maximize the repair efficiency. A low level of redundant repair protection is applied by using a limited number of spare elements to repair row, column, or cluster faults. Bit-shuffling is additionally used to effectively deal with isolated faults. As for the implementation, the address is sent to a CAM to take redundant repair and shift value part to take bit-shuffling in parallel, shown in Fig. 3.6. Since this scheme is a combination of these two schemes, the overhead is the sum of the individual overheads of the two constituent schemes.

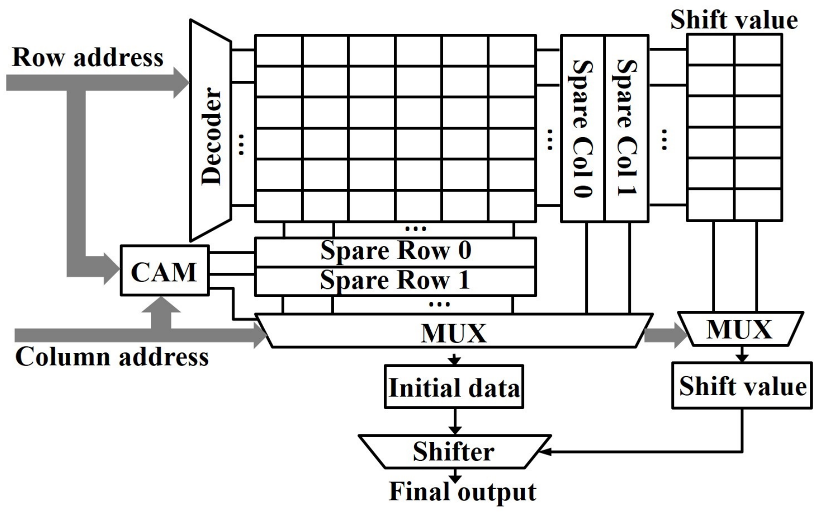


Figure 3.6: Implementation of the hybrid scheme.

3.3 Area and Delay Overheads

3.3.1 Area model

The area model for each scheme is based on the transistor count of the structure. For both the 1T1C DRAM and the 1T1MTJ STT-MRAM, the area is proportional to the transistor count. Note that the MTJ cell is dominated by the access transistor for an STT-MRAM. Therefore, a similar area model can be used for both memory types.

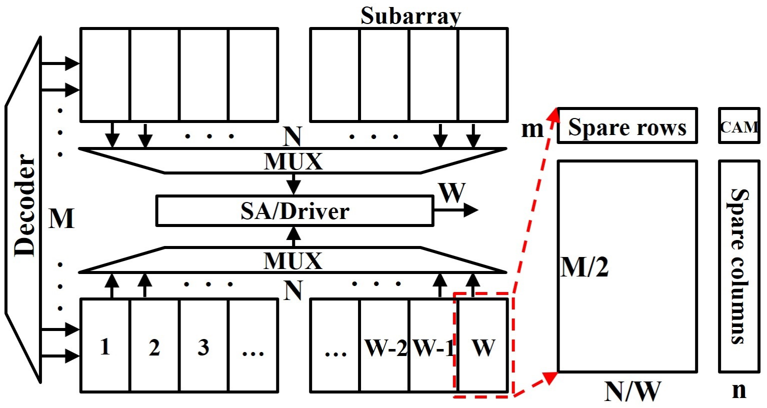


Figure 3.7: Composition of a mat for redundant repair.

Basic structure

As described in Section 3.1, the memory is eventually divided into several identical mats, each of which has a grid of 2×2 subarrays. Consider a mat with M rows, N columns with an output width of W, as shown in Fig. 3.7. Each mat consists of [3]:

- A row decoder and wordline drivers with M outputs, with an area cost of $6M$, corresponding to M NAND gates and M inverters. Note that the cost of the predecode blocks is negligible.
- The memory array, consisting of $M \times N$ one-transistor cells.
- Two column MUXes, each consisting of N pass transistors and a column decoder consisting of N NAND gates, for a total cost of $10N$.
- W sense amplifiers/drivers, each with 10 transistors, for a cost of $10W$.

Therefore, the total base area of the memory array is $6M + MN + 10N + 10W$ units.

Area models for various schemes

2D redundant repair The overhead of the 2D redundancy technique is primarily related to the cost of the spare elements and the required peripheral circuitry, as illustrated in Fig. 3.2. After the ESP, or any other algorithm replaces the faulty regions by a subset of the available spare rows or columns, the address of the replacement is stored in a content-addressable memory (CAM). When a memory address is applied, the CAM determines whether the address matches a faulty region. If so, the access is redirected to the appropriate spare element. A multiplexer, which is omitted from the figure for simplicity, is used to select from either the main memory array or the spares, based on the output of the CAM block. The area overhead of the scheme corresponds to the spare rows/columns, the additional complexity in the column multiplexer, and the CAM. The W bits in each word are interleaved in memory. For each bit, we consider a subarray, magnified in Fig. 3.7, of size $(M/2) \times (N/W)$. Since there are $2W$ such subarrays in a mat, as shown in the figure, the overhead for redundant repair with m spare rows and n spare columns is as follows:

- The spare rows require $m \times (N/W)$ cells per subarray, so that for $2W$ subarrays, the overhead is $2mN$.
- The spare columns require $n \times (M/2)$ cells per subarray. Over all of the $2W$ subarrays, the total overhead is nMW.
- The reconfiguration addresses are stored in a CAM of size $\log(M/2)$ and $\log(N/W)$, respectively, for the rows and the columns. Since each CAM cell consists of 10 transistors and uses a sense amplifier with 4 transistors, the number of transistors in the CAM for each subarray is $T_{CAM}(m, n) = m \times (10 \log(M/2) + 4) + n \times (10 \log(N/W) + 4)$.
- The MUXes for the spare columns require n pass transistors, for total of $2nW$ transistors.

The overhead for this scheme is thus $2mN + nMW + T_{CAM}(m, n) + 2nW$ units.

Segmented memory repair For the segmented repair model, if the rows and columns are both divided into P parts, then the overhead is the same as 2D redundant repair, except that the cost of the CAM is now $P \times T_{CAM}(m,n)$, where $T_{CAM}(m,n)$ is as defined above.

k-MSB repair For k-MSB repair, the arrays containing k MSBs of the byte use m spare rows and n spare columns, while other arrays use fewer spare elements, with m_{less} rows and n_{less} columns. These modifications are made to the first two terms of 2D redundant repair, and the overhead of the CAM is modified to $k \times T_{CAM}(m,n) + (W-k) \times T_{CAM}(m_{less}, n_{less})$.

Bit-shuffling and compressed bit-shuffling The overhead of bit shuffling is based on the hardware scheme in Fig. 3.4. The compressed bit-shuffling scheme is parameterized by n_{FM} and the compression rate R, where R is the number of bytes in a group that share a shift value marker.[1] For an $M \times N$ mat, $M \times N/(8R)$ bytes have their own shift values with $\log R$ bits to locate their position. For the structure in Fig. 3.5, the transistor overhead is shown below:

- The shift values for a byte need n_{FM} bits of storage and are shared over R bytes. For M rows and N columns, this results in a total cost of $(M \times N/(8R)) \times n_{FM}$.
- The identity of which of the R bytes should be shifted is stored in a small array with $\log R$ bits per group. Its transistor overhead is $(M \times N/(8R)) \times \log R$.
- $(n_{FM} + \log R) \times N/(8R)$ pass transistors are required.
- The 8-bit circular shifter has $n_{FM}+1$ layers, with 8 MUXes per layer, each with 6 transistors. This leads to a transistor count of $48(n_{FM}+1)$.

The total overhead for this scheme is thus $(M \times N/(8R)) \times (n_{FM} + logR) + (n_{FM} + \log R) \times N/(8R) + 48(n_{FM}+1)$ units.

Hybrid bit-shuffling and redundant repair The overhead can be deduced from the above discussion, adding the cost of bit-shuffling to the overhead of redundant repair.

[1] Note that the degenerate case of $R = 1$ corresponds to the scheme in [6].

3.3.2 Delay model

The delay model is based on the delay associated with the CAM, the addressing delay, the precharge delay, and the cell read-write time. For the 1Gb DDR3 DRAM in [71], $t_{RCD} = 13.91ns$ and $t_{CL} = 13.91ns$. Thus, the delay to read from memory is $t_{RCD}+t_{CL}$, which includes opening a row-by-row decoder and accessing the memory array and the active column MUX to get the data out. For the 16Mb STT-MRAM in [72], the read cycle time is $35ns$.

All of the redundant repair schemes – 2D redundant repair, segmented redundant repair, and k-MSB redundant repair – need a CAM to store the reconfiguration address for the spare elements. The matchline delay of the CAM is determined by the precharge and evaluation time [73], where $t_{pre} = 2.2R_{EQpre}C_{ML}$, $t_{eval} = 0.69R_{ML}C_{ML}$, R_{EQpre} is the equivalent resistance of the precharge transistor, R_{ML} is the equivalent resistance in matchline, and C_{ML} is the equivalent capacitance. The delay of the MUX used to select the data from the spare elements or the main memory array can be estimated as three FO4 delays. All of the corresponding parameters for the CAM and FO4 come from CACTI [3] for the DRAMs case and NVsim [74] for the STT-MRAMs to compute maximum timing overhead ($Max\ delay_{CAM} < 0.2ns, Max\ delay_{MUX} < 0.1ns$).

For the bit-shuffling related schemes, the chief delay overhead is the circular shifter due to delay in read shift value operation partially overlaps delay in memory read or write operation. The shifter can be implemented using $n_{FM} + 1$ MUX layers. For our case, the maximum value is four MUX layers, giving a total delay of 12 FO4 ($< 0.4ns$).

Based on maximum timing overhead for various parts of the array, it is clear that, compared to the original memory access time, the delay overhead of the additional circuitry is minimal and *can be neglected*.

3.4 Experimental results

We evaluate the performance of each repair technique for a $128M \times 8$ banks DDR3 DRAM similar to [71] and a $1M \times 16$ bit STT-MRAM similar to [72] using the CACTI memory organization. For each memory type, various faults are injected with the mean failure rate, P_{cell}, set to 0.1%, 0.25%, 0.5%, and 0.75%.These failure injection rates are significantly higher than those typically used in prior memory repair studies since we are evaluating error-resilient applications which can use memories that are much more error-prone and, hence, less expensive. The simulations use 10^5 Monte Carlo samples corresponding to the distribution in Section 3.1.2. Each of these samples corresponds to the total number of faults in a memory array. The probabilities that these faults map on to row faults, column faults, cluster faults, and single cell faults are set to 1%, 10%, 2%, and 87%, respectively [67]. Once the fault region is fixed for a failure, the fault type is set to one of options described in Section 3.1.2. For purposes of repair, the precise type of fault is immaterial since the existence of a fault would trigger the use of a repair strategy. The impact of the fault on the quality of the JPEG compressed image does not depend on the type of memory, i.e., DRAM or STT-MRAM, but rather on the total number of faults and their spatial distribution.

We use the PSNR to measure the quality of the images stored in the memory arrays as various faults are applied. Our experiments (details omitted due to space limitations) show that the distribution of the PSNR for the Monte Carlo simulations is always very tightly spread about the mean so that the variance is negligible. Therefore, it is sufficient to characterize the PSNR performance results using only the mean values. To compare the different schemes, we define the *quality degradation* (measured in dB) and the *percentage quality degradation* as the change in PSNR compared to the full quality image. We evaluate the different schemes by showing the changes in the quality degradation metric as a function of the area overhead, which is expressed as a percentage of the total area of an equivalent size memory array with no repair capability plus its supporting circuitry.

Fig. 3.8 shows the quality-overhead trade-offs for the bit shuffling schemes. Within each chart, the different bars within a group represent the various cell failure probabilities. The different charts within each column show the results for various compression rate of compressed bit-shuffling scheme. In reality, the memory arrays have row faults, column faults, and cluster faults. In this case, the quality degradation goes up significantly for all the bit-shuffling schemes. Fig. 3.9 shows the quality-overhead trade-offs for the hybrid scheme that combines bit-shuffling with redundant repair and uses the term "H-Compression rate" to represent this hybrid scheme with compressed bit-shuffling. The four curves correspond to various cell failure probabilities and the points on a curve represent $n_{FM} = 1, 2, 3$. For the same n_{FM}, the number of spare elements added for redundancy repair vary between the failure probabilities, resulting in different area overheads. As compression rate increases, the overhead will be reduced with minimal quality degradation. The hybrid schemes significantly reduce the quality degradation for cluster faults with only a small increase in overhead compared to only bit-shuffling schemes. Thus, the following comparisons only include the hybrid schemes.

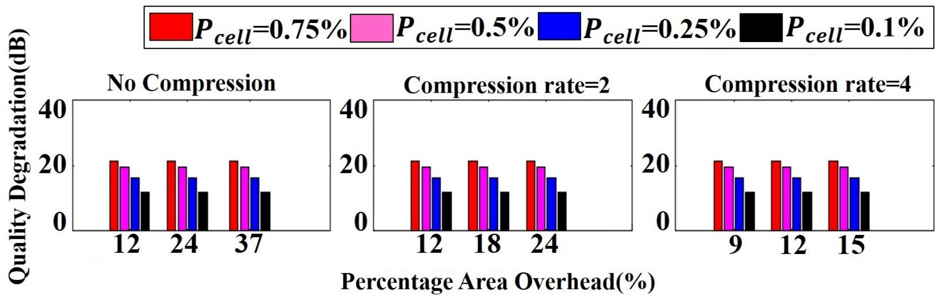


Figure 3.8: Quality-overhead trade-offs for bit-shuffling with clustered faults and the points on the x-axis correspond to $n_{FM} = 1, 2$ and 3.

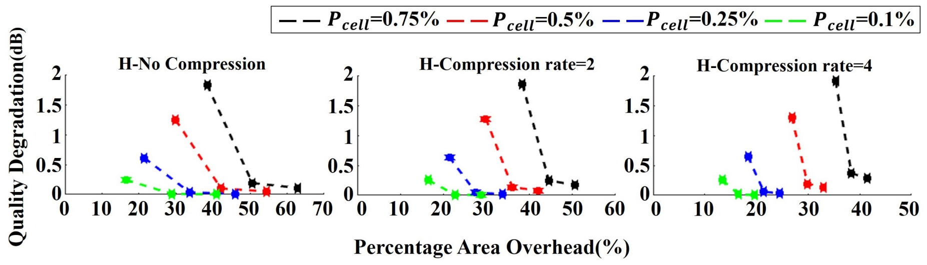


Figure 3.9: Quality-overhead trade-offs for the hybrid scheme with clustered faults.

Fig. 3.10 compares all approximate memory repair schemes. To provide another perspective, this figure exchanges the horizontal and vertical axes used in prior graphs. The x-axis groups various ranges of quality degradation, expressed as a percentage increase in degradation compared to the perfect memory. The y-axis shows the area overhead. Each plots shows the results for a different fault probability. These plots allow a designer to determine the cost of a specific scheme given a maximum allowable quality degradation or, given a cost budget, to determine the expected quality degradation of each scheme.

To evaluate the schemes that use limited spare rows and columns, we first choose the number of spare rows and columns that are required to ensure full repair over all of the Monte Carlo samples for 2D redundant repair and segmented repair. This value then is adjusted from 100% for a fully repaired system to various reduced values, each of which corresponds to reduced overhead at the cost of some quality degradation. Fig. 3.10 shows the level of quality degradation for various schemes and the corresponding overhead. In our simulation, we segment the memory into 2, 4 and 8 blocks and the results show that the segmented repair scheme works best with only two blocks. This is because, as the number of segments increases, the number of spare cells goes down, but the number of high cost CAM cells will increase. However, both the 2D and segmented repair schemes always have the highest overhead for a given level of quality degradation since these schemes are not very efficient.

It is clear that the performance of the k-MSB scheme is substantially better than the 2D and segmented repair schemes providing lower overhead and lower quality degradation, even for $k = 1$. Furthermore, decreasing k tends to reduce the overhead for a given level of quality degradation by improving the efficiency, but lower values of k may be unable to achieve the best quality. The hybrid bit-shuffling with redundant repair scheme shows the best performance as the probability of a fault increases. However, because every byte (for no-compression version) needs extra space to store the shift value, the hybrid scheme has relatively high overhead compared to the other schemes when the probability of a failure is low. Moreover, the hybrid scheme with compression can reduce the area overhead with lower quality degradation compared to no compression. In summary, we find that the k-MSB and hybrid schemes generally provide the best results across all failure probabilities.

3.5 Conclusion

This chapter has shown how partially repaired memories can be effectively used for error-resilient image applications. We find that the nature of the fault and the type of memory matters less than the number and spatial distribution of the faults. For image compression, the proposed k-MSB and hybrid bit-shuffling and redundant repair schemes provide large reductions in overhead compared to the fully-repaired case and prior repair schemes with little quality loss. Our new approaches perform well in the presence of row, column, and clustered faults, where the previously proposed basic bit-shuffling method may experience large quality degradation.

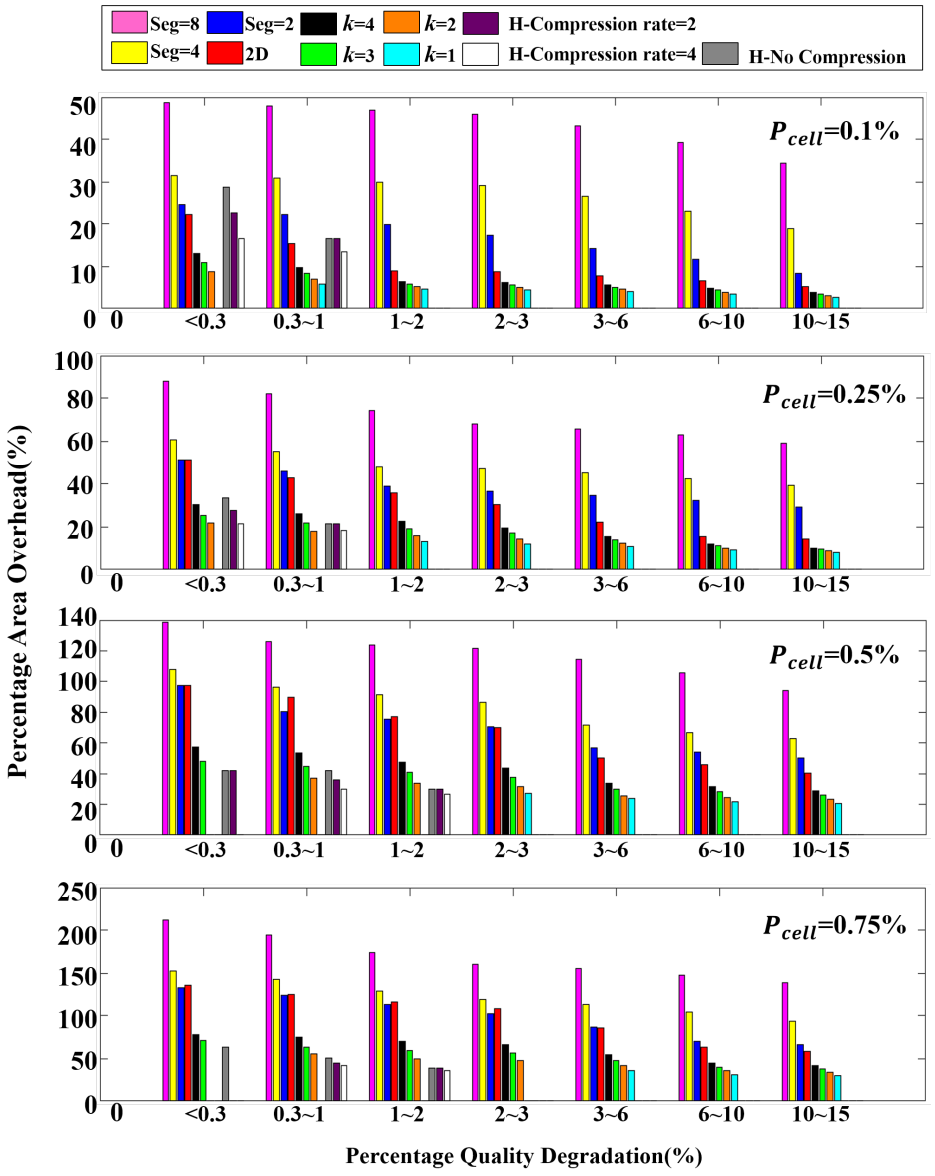


Figure 3.10: Overhead-quality comparisons of the various approximate memory repair schemes for different failure probabilities. Missing bars (e.g. k=1 for $< 0.3\%$ degradation under $P_{cell} = 0.1\%$) show configurations that are impossible to actually build for the given overhead-degradation combination.

Chapter 4

Adaptive-length Coding of Image Data for Approximate Storage

In this chapter, we focus on placing the image data with adaptive-length coding (ALC) scheme into approximate storage. ALC can provide a balance between the reliability of fixed-length coding schemes and the storage-efficiency of Huffman coding schemes [75]. We compare our JPEG-based ALC scheme only with the JPEG-based VLC scheme in the evaluation. Our target is to place the data into approximate storage, while maintaining an acceptable quality and compression efficiency.

4.1 Preliminaries and motivation

4.1.1 Cost-reliability trade-offs for approximate storage

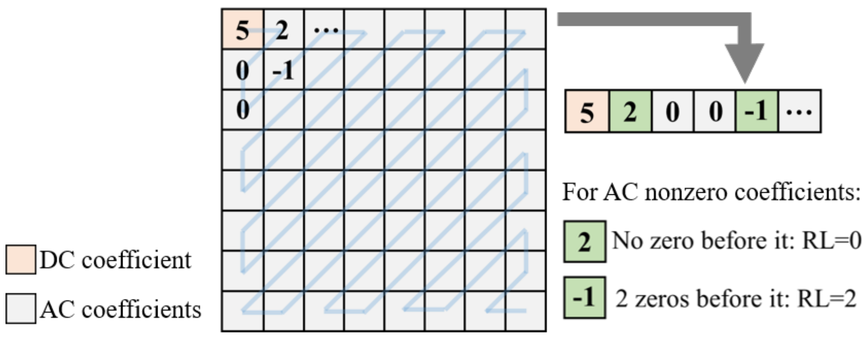
There exists a clear trade-off between the cost of a storage device and its reliability. For instance, inexpensive HDDs have lower reliability than more expensive enterprise-class devices. As a result, the less expensive devices are typically used in a RAID configuration to improve the system reliability. There is a similar trade-off for SSD devices and for other types of storage technologies. Furthermore, we can make other trade-offs besides reliability and price cost. For example, there is a trade-off between the cost of adding ECC to a memory system and the resulting reliability. This trade-off has been used in prior work to build an approximate storage system [1,7]. There also exists a trade-off in NAND flash technology between the cell storage efficiency (bits/cell) and reliability [76]. For the emerging STT-MRAM technology, the reliability is related to the energy required to change the device's state. A lower energy circuit will provide lower reliability [8, 9].

The goal of this work is to develop a methodology for storing a compressed image in approximate storage. We propose a general methodology and presenting the notion of the trade-off under various cost scenarios. As a result, we assume only two types of storage - reliable and approximate - that have different cost ratios. The specific implementation of these two types of storage is beyond the scope of this work.

4.1.2 Fundamentals of JPEG

JPEG is a commonly used technique that employs the discrete cosine transform (DCT) to perform lossy compression on digital images [77]. The DCT divides an image into a set of 8×8 pixel frames, converting the sub-image in each frame from the spatial domain into an 8×8 matrix of coefficients in the frequency domain, corresponding to various spatial frequencies. The zero-frequency coefficient of the DCT is referred to as its DC term, which is the average of the pixel values. It provides a baseline value for the encoding. The remaining elements for the AC terms provide information about the successively higher frequency components, which represent color changes across the block. Since the human eye is insensitive to high-frequency spatial variations, a quantization step is used to divide each coefficient by the non-uniform entry in the quantization matrix (higher resolution for DC and low-frequency components), and an integer result is stored. As larger divisors are used at higher frequencies, and DCT coefficients tend to be lower at high frequencies, many coefficients go to zero. Only nonzero DCT coefficients are stored, and thus the volume of compressed data is greatly reduced from the raw image. The original image can be reconstructed from this data with little or no discernible loss in quality. The entries of the quantization matrix, for a quality factor Q, range from 1 to 100, where a higher number corresponds to higher quality ($Q = 90$ is widely used). The quantization matrix for any value of Q can be derived using the procedure in [77] that is based on standard quantization matrix for $Q = 50$.

The top left entry in the 8×8 DCT coefficient matrix is the DC coefficient. The DC coefficients store the basic information for the image, and employ differential coding,



5
2
...
0
-1
0
5
2
0
0
-1
...
For AC nonzero coefficients:
2 No zero before it: RL=0
-1 2 zeros before it: RL=2
DC coefficient
AC coefficients

Figure 4.1: The zigzag pattern that orders the DCT coefficients for storage.

whereby the difference of the DC coefficients between successive blocks is saved. This enables more compact storage, but allows an error to affect multiple blocks. The remaining entries are the AC coefficients. In practice, since the nonzero AC coefficients tend to cluster near the top left entry of the matrix, the quantized coefficients are stored in a zigzag sequence, as shown in Fig. 4.1. Then DC and AC coefficients are encoded separately based on different Huffman tables specified in the JPEG standard [78].

Table 4.1: VLC-encoded AC coefficients in Fig. 4.1.

Nonzero coefficient	RL	1's complement representation	CAT	Codeword
2	0	10	2	01 10
-1	2	0	1	11100 0

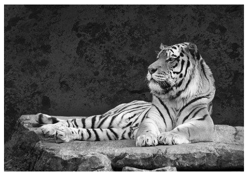
In VLC, the value of a coefficient is stored using one's complement representation[1] , where the sign is encoded by ensuring that the MSB for a positive [negative] number is 1 [0]. For example, consider a differentially coded DC coefficient of 5, represented in one's complement notation as 101, in Fig. 4.1. Then the category (CAT) of this coefficient will be encoded based on the Huffman table for the DC coefficient, which represents the number of bits required to store this value. Therefore, for this example, CAT = 3, which is translated to 100 after Huffman coding. The encoded DC coefficient then concatenates these to obtain the codeword 100 101. In VLC, the maximum value of CAT for DC coefficients is 11 bits.

[1] two's complement representation can also be applied, but it will not introduce any meaningful changes in the results.

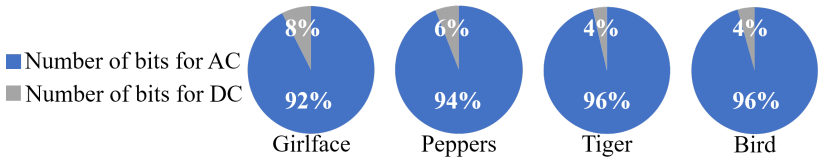
Table 4.1 encodes the AC coefficient values in Fig. 4.1. The sequence of coefficients is translated to symbols that encode the run-length (RL) and the category (CAT) of the coefficient. Here, RL corresponds to the number of zeros preceding the coefficient, e.g., for the entry of -1, RL $= 2$ as it is preceded by two zeros. Each (RL, CAT) symbol is represented by a variable-length codeword based on Huffman tables, assigning shorter codes to more common symbols, e.g., in Table 4.1, (RL $= 0$, CAT $= 2$) is represented by the symbol 01. In VLC, RL is limited to 4 bits and CAT to 10 bits for AC coefficients; in case of larger run-lengths, symbols with (RL $= 15$, CAT $= 0$) can be concatenated.

(a) Girlface

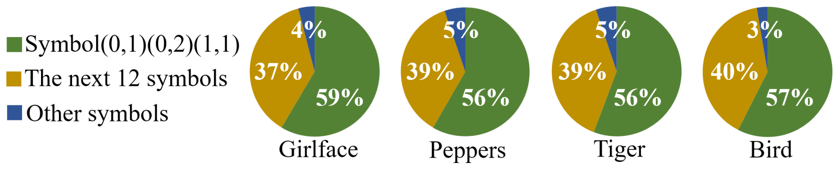
(b) Peppers


(c) Tiger

(d) Bird



(e) Number of bits required to store the DC and AC coefficients using VLC



(f) Distribution of the (RL, CAT) symbols for the AC coefficients

Figure 4.2: A set of sample images and the distribution of the symbols representing their AC and DC coefficients.

For the block in Fig. 4.1, the precise bit string that is stored is a concatenation of the codewords for the DC coefficient and all AC coefficients in Table 4.1, i.e.,

$$1001010110111000\ldots1010$$

where the ellipsis represents the nonzero coefficients that are not shown in the figure. The last symbol, 1010, corresponds to end of block (EOB), a unique symbol that indicates the last nonzero coefficient of a block. The EOB symbol acts as a separator between consecutive blocks.

As shown in Fig. 4.2(e), the storage overhead is dominated by the AC coefficients. Further, among these AC coefficients, the distribution of the symbols (RL, CAT) is quite unbalanced. For example, in the images, Girlface, Peppers and the two additional images Tiger, Bird, from the Imagenet dataset [79], the first 15 symbols can be seen to cover over 95% of the coefficients, with (0,1), (0,2) and (1,1) corresponding to over half of all 161 symbols, as shown in Fig. 4.2. This motivates the use of Huffman coding in VLC, but we will soon see how VLC is sensitive to errors.

4.1.3 Error resilience limitations of JPEG storage schemes

The error tolerance for traditional VLC-encoded JPEG, where the storage scheme uses a Huffman-based coding algorithm, is quite limited, and even a single error can bring a dramatic degradation, as illustrated in Fig. 1.1(a). This error occurs due to a misalignment caused by an erroneous symbol, which maps to a keyword with a different (RL, CAT) value and disrupts symbol boundaries. This creates noisy data in the blocks that follow. Moreover, some EOB symbols are left undetected, and the image is interpreted to have fewer blocks, with all-zero coefficients in the last few blocks, causing them to be black.

One way to avoid such scenarios is to annotate each block with the number of bits that it contains [43]. This improves the error resilience by reducing the possibility of error propagation across blocks, but under a larger number of errors, as shown in Fig. 1.1(b).

Another alternative is to use fixed-length coding (FLC) [51]. Instead of a variable-length codeword, FLC may use the maximum number of bits for RL (4 bits) and CAT (10 bits) if coding AC coefficients. For a constant number of CAT bits, it is essential to also explicitly store a sign bit since the sign can no longer be inferred from the leading bit of the data. In other words, to represent signed binary AC coefficients, the sign-magnitude representation is needed for FLC instead of only using the one's complement representation in VLC. This implies that the length of each codeword in FLC can be $4+10+1 = 15$ bits. This fixed codeword length allows the boundary between codewords in this scheme to be determined easily, so that errors no longer propagate to subsequent codewords, but this potential improvement in error resilience is accompanied by a severe degradation in compression efficiency. Furthermore, the error resilience may not actually be improved, since the large increase in the number of storage bits raises the likelihood of errors that cannot be corrected for.

4.2 Adaptive-length coding

In this section, we propose the adaptive-length coding (ALC) scheme to provide a balance between the reliability of fixed-length coding schemes and the storage-efficiency of Huffman coding schemes. The ALC method is described in three steps: symbol classification for the AC coefficients (Section 4.2.1), adaptation for additional bits (Section 4.2.2), and EOB identification (Section 4.2.3).

It is worth noting that the number of AC coefficients significantly exceeds the number of DC coefficients, as shown in Fig. 4.2(e). Meanwhile, as we will see in Section 4.6.4, the DC coefficients contain the important characteristics of an image and are not very tolerant to errors. Therefore, we develop the ALC scheme to compactly store the AC coefficients while the DC coefficients are still encoded using VLC.

4.2.1 Step 1: Symbol classification

Our ALC scheme places all symbols for the AC coefficients into one of two classes. Class I corresponds to a shorter fixed codeword length for the most frequent symbols, (0,1), (0,2), and (1,1). Class II consists of all other symbols, which are encoded using a longer fixed-length.

The structure of the codewords is summarized in Fig. 4.3. The MSB indicates whether the symbol belongs to Class I or II. In case of Class I, all of the remaining bits are used to store the coefficient. For Class II, if RL = 0, the second-MSB is set to 0, the third bit is the sign bit, and four bits are used to save the coefficient magnitude. If RL $\neq$ 0, then the second-MSB is 1, followed by three bits for RL, the sign bit, and 1 bit for the coefficient magnitude. Similar to FLC, the sign-magnitude representation is used for the AC coefficients in ALC. Next, we justify the choice of the number of bits used to store RL and the coefficient.

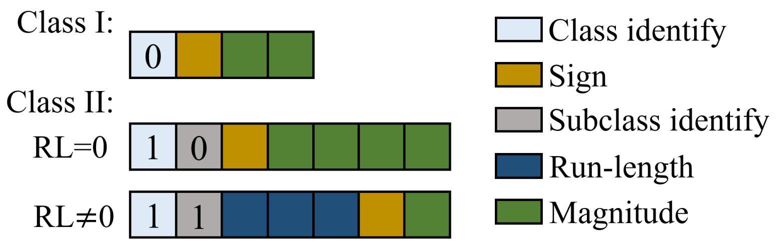


Figure 4.3: Bit assignment for Class I and Class II symbols in the ALC scheme.

Precise representation for Class I

Table 4.2 shows our scheme for uniquely representing Class I codewords using a 4-bit fixed-length code. For Class I, the MSB is set to 0. The next bit indicates the sign, and is 0 for a positive value and 1 for a negative value. The last two bits denote the magnitude of the coefficient. Since the only symbols in Class I are (RL, CAT) = {(0,1), (0,2), (1,1)}, there are four cases without considering the sign bit, as shown in Table 4.2. Thus, the magnitude and RL of Class I can be precisely represented using a 2-bit fixed length field. For comparison, the corresponding VLC codes are shown, which have similar output length to the ALC codes.

Table 4.2: Encoding Class I in ALC, and a comparison with VLC.

(RL, CAT)	Magnitude	ALC Codeword		VLC Codeword		Output length	
		Positive	Negative	Positive	Negative	ALC	VLC
(0, 1)	1	0 0 00	0 1 00	00 1	00 0	4	3
(0, 2)	2	0 0 01	0 1 01	01 10	01 01	4	4
	3	0 0 10	0 1 10	01 11	01 00	4	4
(1, 1)	1	0 0 11	0 1 11	1100 1	1100 0	4	5

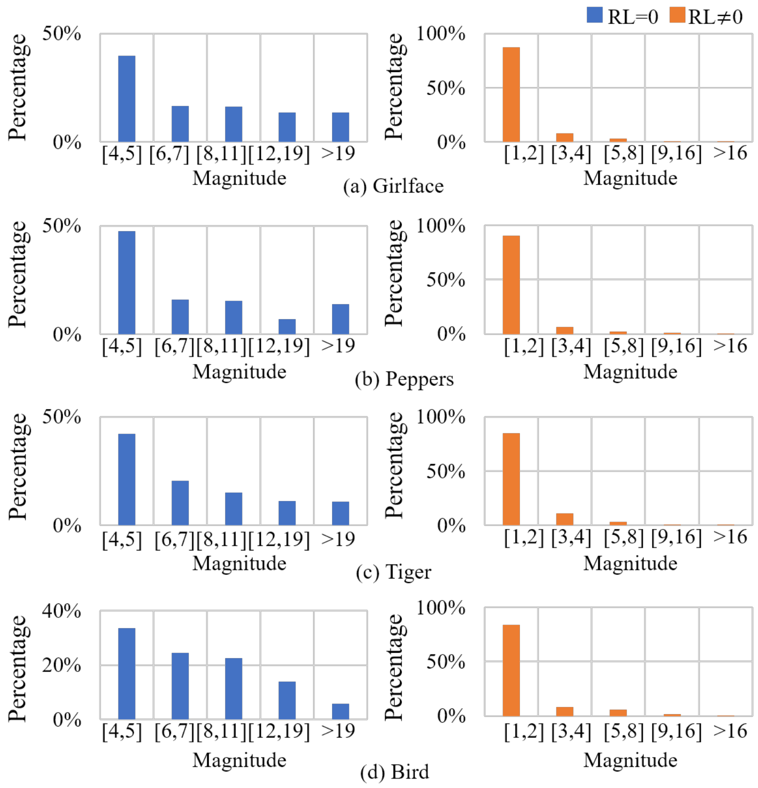


Figure 4.4: A histogram of coefficient magnitudes for Class II symbols for a representative set of images, for RL $= 0$ and RL $\neq 0$. The values in the x-axis for each class group several magnitude values when one more bit is added in the Class II magnitude field.

Approximate representation for Class II

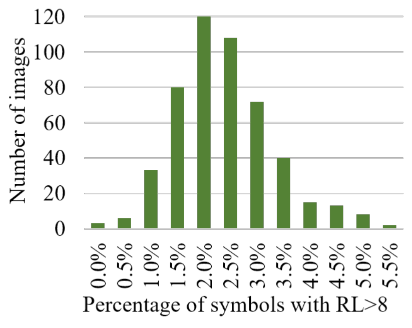
For elements in Class II, Fig. 4.4 shows the distribution of the DCT coefficient magnitudes without considering the symbols in Class I. This distribution determines the number of bits required for Class II. Four representative images are analyzed in Fig. 4.2(a)–4.2(d), separating the scenarios where RL $= 0$ and RL $\neq 0$. All results show the same trend: when RL $\neq 0$, CAT is small for almost all possible values, and is larger only when RL $= 0$. Thus, finding symbols with large values of both RL and CAT is improbable. Based on this observation, the use of the fixed bit length for Class II symbols varies with RL. When RL $= 0$, all bits are used to store the coefficient, and when RL $\neq 0$, the first few bits represent RL while the rest correspond to the coefficient, as illustrated in Fig. 4.3.

Selecting the number of bits for RL $\neq$ 0 Table 4.3 examines the symbols that cannot be represented by three RL bits, i.e., with RL $= 1, \cdots, 8$. Only a small fraction of symbols have RL > 8 and these tend to have small magnitudes. Thus, for this case, 3 bits are sufficient to represent RL. Any symbols with RL > 8, and all coefficients beyond this symbol, are discarded with little quality loss.

Table 4.3: Distribution of symbol magnitudes for RL > 8

		Girlface	Peppers	Tiger	Bird
Fraction of symbols with RL > 8		1.67%	5.18%	2.83%	0.42%
Distribution of the coefficient magnitude in RL > 8 symbols	Magnitude = 0	7.53%	13.29%	7.31%	3.28%
	Magnitude = 1	90.75%	86.30%	92.26%	96.72%
	Magnitude = 2	1.54%	0.41%	0.24%	0.00%
	Magnitude > 2	0.18%	0.00%	0.19%	0.00%

The observation from Table 4.3 can be extended to a larger set of images. Fig. 4.5 shows the distributions generated with 500 images in the dataset [79]. Figure. 4.5(a) shows that the fraction of symbols that require RL>8 is quite small. Thus, three bits for RL is sufficient for most situations. In Figure. 4.5(b), only a few parts of the symbols with RL>8 have magnitude larger than 2, which means the information can be discarded with little impact on the final results.



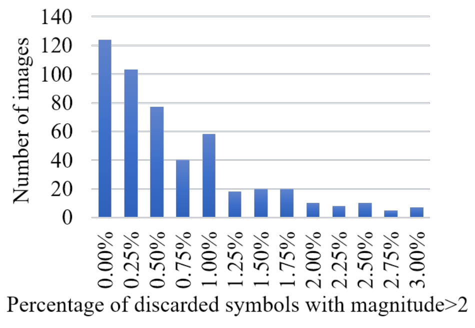


Figure 4.5: The distribution of images based on the different percentage of symbols that cannot be covered by Class II representation for RL (a) Discard the symbols with RL>8. (b) The discarded symbols with the magnitude larger than 2.

Selecting the number of bits for the coefficient magnitude According to the Huffman table for the AC coefficients, the magnitudes can be represented precisely using 10 bits. However, from Fig. 4.4, numbers that use all 10 bits are seldom encountered. Therefore, we reduce the number of magnitude bits while still covering the vast majority of scenarios. The figure shows that the distribution of magnitudes for RL $= 0$ has a wider spread than for RL $\neq 0$, but even here, the number of coefficients whose values exceed 19 is small. Therefore, our ALC encoding represents the magnitude using 4 bits. Since any value of RL > 8 is treated as an EOB, it is not necessary to store coefficient values of 0 to store long run lengths, and Class II has no magnitudes of 1 and 2 when RL $= 0$. Therefore four bits can be used to store magnitudes ranging from 4 to 19. If the actual coefficient magnitude exceeds 19, a value of 19 is stored. Similarly, for RL $\neq 0$, the vast majority of coefficient magnitudes are ≤ 2, so that ALC uses 1 bit to store coefficients[2] of 1 or 2. Coefficient magnitudes of 3 or larger are capped at 2. This is a large reduction over FLC, which requires 10 bits to represent the coefficient magnitude.

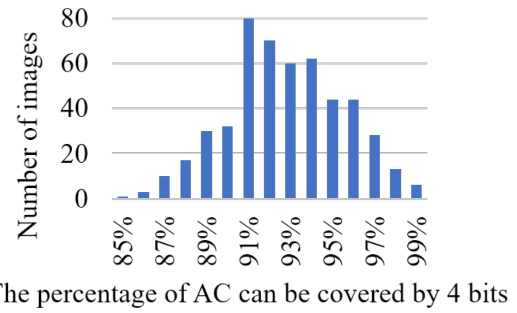
The distribution of these four images are representative and the number of bits covering most of the scenarios works for a larger set of images. Fig. 4.6 shows the distributions generated with 500 images in the dataset [79]. When RL=0, most images have over 90% of the AC coefficients that can be covered by 4 bits for magnitude; when RL$\neq$0, most images have over 85% of the AC coefficients that can be covered by 1 bit. This parameter selection is verified on a larger dataset of images in Section 4.6.

4.2.2 Step 2: Adaptation per image

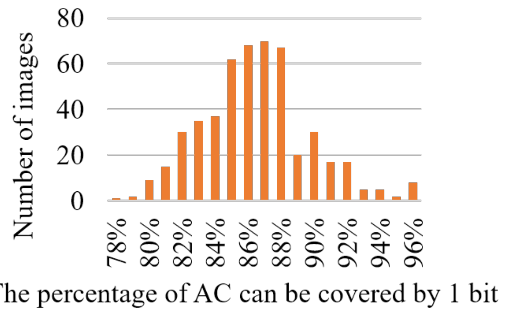
The basic ALC scheme described above combines the spirit of the Huffman coding scheme in VLC with the error-resilience of FLC. However, if used directly, only using the few bits in Section 4.2.1 to represent the magnitude of Class II is not enough and it can lead to a significant degradation of the image quality.

To better understand the reason, we study the average magnitudes of all the blocks in a representative image, Girlface, (other images show similar overall trends) for each AC frequency band. Fig. 4.7 presents a color-coded map of these magnitudes for each

[2] When RL=1, the magnitude representation can be extended to encode 2 or 3 since (RL, CAT)=(1,1) is already encoded in Class I.



(a) RL=0



(b) RL≠0

Figure 4.6: The distribution of images based on the different percentage of AC coefficients can be covered by Class II representation for magnitude.

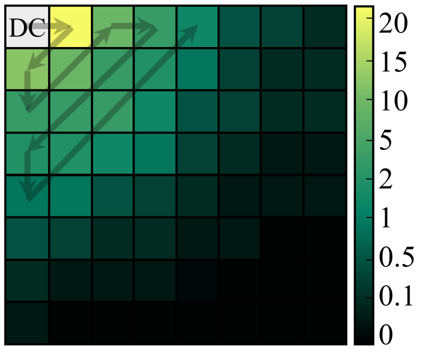


Figure 4.7: The average AC coefficient magnitudes, over all frames, of all the blocks in Girlface. A lighter color represents a higher magnitude.

element in the 8×8 coefficient matrix. The figure indicates that larger-magnitude coefficients are located near the beginning of the zigzag storage sequence. In particular, the first few lower-frequency elements of the sequence have larger coefficients with more important information than higher-frequency components. For instance, we can see that there is an important AC frequency band whose average value is about 20, but this results in an overflow in our basic ALC scheme, which only allows a maximum four bits to store a coefficient.

We alter the basic scheme to improve quality by selectively adding extra bits to the codewords of Class II located in lower-frequency positions, which allows storing these high magnitude coefficients more precisely. In order to develop a computationally simple methodology, the number of additional bits and the number of codewords with these additional bits are the same for all blocks in one image. Therefore, adding one more bit can result in a large change in output length. To ensure high compression efficiency, rather than choosing a worst-case value over all images, we employ image-specific adaptation, where the number of additional bits is chosen based on the characteristics of a specific image.

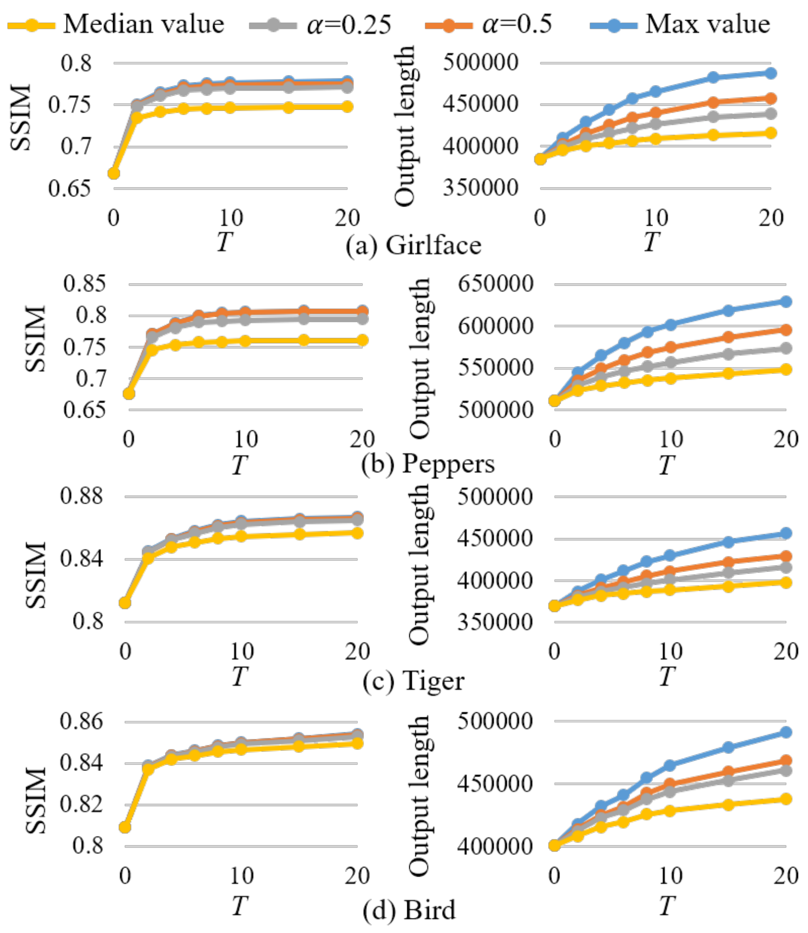


Figure 4.8: The impact of using different methods to decide the number of bits to be added to the first T AC codewords.

The number of additional bits required to precisely represent the first T AC codewords is determined by all block samples from an image, where T is an optimization

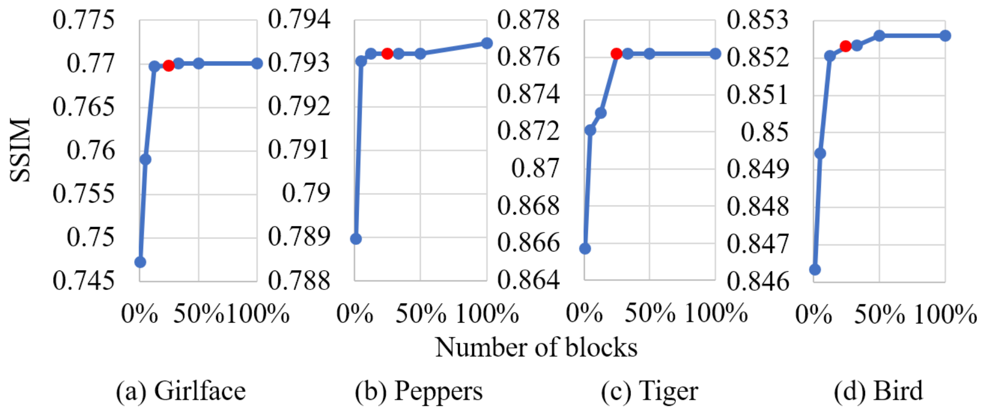


Figure 4.9: The impact of using different number of blocks to decide the number of bits to be added to the first T AC codewords.

parameter, in the zigzag pattern based on this data. We choose only the first T AC codewords because the AC codewords near the end of the block normally have lower magnitudes. Then we find the maximum and median values of the required additional bits among all block samples. If we use the Structural Similarity Index (SSIM) [80] as the quality metric (defined in Eq. (4.3) in Section 4.5.3), in the absence of error, we find that using the maximum value of the required number of additional bits provides the best quality, but at highest overhead (because this value may be rare). Another alternative is to use the median value, which has relatively low overhead, but this still results in large quality degradation, as illustrated in Fig. 4.8.

We find that using $median\_value + \alpha \times (max\_value - median\_value)$ provides an effective trade-off between the goal of reducing the overhead while delivering adequate quality, where α can be set to any value from 0 to 1. Fig. 4.8 shows the results for $\alpha = 0.25$ and $\alpha = 0.5$. In our experiments, $\alpha = 0.25$ is used.

Moreover, as shown in Fig. 4.8, the effectiveness in improving the quality goes down steeply after the first few codewords. Thus, in our experiments, we only test the value of T from 0 to 10 to satisfy different quality requirements.

Considering the high correlation of content among adjacent blocks, we uniformly select a subset of the blocks from an image and determine the number of additional bits required to precisely represent the first T AC codewords. From Fig. 4.9, it can be seen that once a subset of the blocks are sampled, a representative value of T can be chosen.

The red dot shows the SSIM value when 25% of the blocks are chosen, and empirically this is seen to be a safe threshold. In principle, this also depends on the order in which the blocks are sampled, but our experiments find the 25% threshold to be safe.

4.2.3 Step 3: EOB identification

In the ALC scheme, since the codeword lengths are fixed, we do not use the EOB symbol, described in Section 4.1.2. Instead, for each block, we record the number of codewords in the block for block alignment. Fig. 4.10 shows the number of blocks with up to k codewords, for various values of k along the x-axis. This indicates that if we limit the number of codewords in each block to 32, over 95% of all blocks can be covered. Therefore, we choose to use 5 bits to record the number of codewords in a block. If the number of codewords is larger than 32, the higher coefficients are discarded, and we empirically observe that this results in minimal quality degradation.

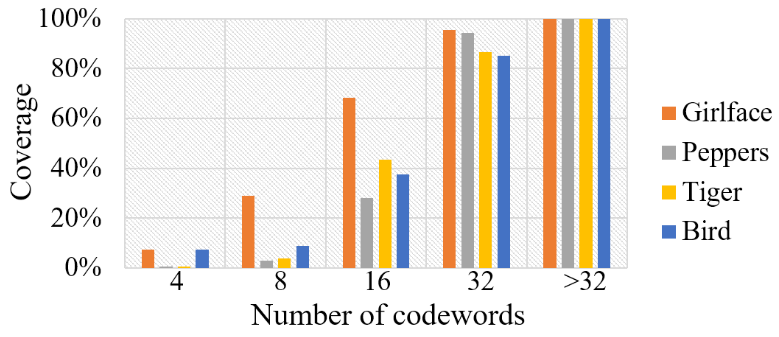


Figure 4.10: The cumulative distribution function of the number of codewords in each block.

Note that recording the number of codewords can prevent errors propagating to other blocks. For example, a bit flip in the RL field could shift the coefficients to an incorrect frequency band. However, since we store the number of codewords per block, and each codeword has a known length, such an error can be limited to the block and will not corrupt subsequent blocks.

4.3 Partitioning data between reliable and approximate storage

The image data is partitioned into two parts, one into reliable storage and the other into the approximate storage. An error in the more important data, such as the DC coefficients, could cause a critical failure. Consequently, this data is placed in the reliable storage using the conventional VLC coding. Critical data for the AC coefficients, such as the number of codewords in each block, also are placed in reliable storage. However, the remaining AC coefficient data, which requires a large number of bits, can be placed in the less expensive approximate storage. The decision to place specific data into the reliable or approximate storage areas is based on the quality requirement and the cost of the approximate storage relative to the reliable storage.

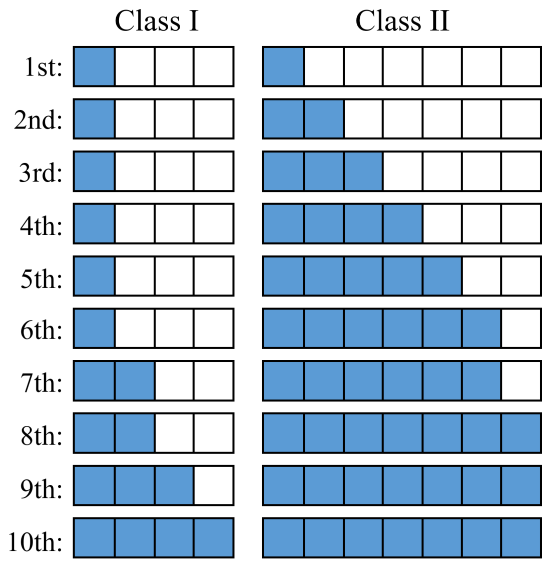
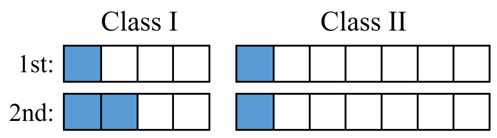


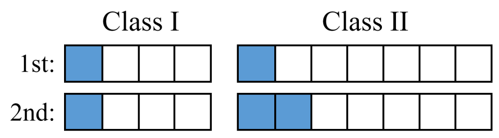
Figure 4.11: The sequence of data partitioning patterns based on the importance order of bits in Class I and Class II, where the blue and white bits are stored in reliable and approximate storage, respectively.

Fig. 4.11 shows ten different patterns that partition data bits between reliable and approximate storage. As we go from the first to the tenth pattern, the quality of the image will improve, at the cost of increased storage costs. In our approach, the final data partitioning pattern between reliable and approximate storage is determined by sequentially trying the patterns in Fig. 4.11 until the quality requirement is met. Once the pattern has been determined, it will be applied to all codewords in Class I and Class II. To restore an image, we have to access both reliable and approximate storage to get the bits of each codeword according to the data partitioning pattern.

Typically, the MSBs of a codeword are more likely to be placed in reliable storage, and the ten patterns successively place larger numbers of MSBs in Class I and Class II in reliable storage. The precise sequence is obtained based on the importance order of bits in Class I and Class II, which have different impacts on output quality, as tested on the sample images, Girlface, Peppers, Tiger and Bird. For example, if we want to generate the second pattern, one more bit should be added in reliable storage compared with the first pattern. As shown in Fig. 4.12, there are two cases: one is placing the first two MSBs in Class I and MSB in Class II in reliable storage for all codewords; the other is placing the MSB in Class I and the first two MSBs in Class II in reliable storage. Under a given error rate (1% error rate is used in this work), the case with higher quality will be chosen as the second pattern. The sequence of data partitioning patterns is derived following this process.



(a) For Class I



(b) For Class II

Figure 4.12: The two cases for geneating the second pattern: one more bit should be added in reliable storage, either for Class I or for Class II, compared with the first pattern.

For data partitioning patterns, we assume the length of codewords in Class II are fixed to simplify the complexity. As explained in Section 4.2.2, only the first few codewords have variable length due to additional bits. The lengths of the codewords are adaptively changed for different images. Thus, the data partitioning patterns should be varied for different codewords and images. The data partitioning process will be less complicated if we assume the lengths of all codewords are fixed. Ignoring these additional bits in the data partitioning pattern will lead to always placing them in approximate storage. However, the errors that occur in these additional bits have a small impact on the final results, since these additional bits are used to compensate for the least significant bits of the magnitude. Therefore, it is likely to put these additional bits in approximate storage and our simplification for the length of codewords is acceptable.

4.4 The impact of dividing the symbols into more classes

If we divide the symbols into three classes and have three fixed lengths as shown in Figure. 4.13(a): Class I corresponds to the shortest fixed codeword length for the most frequent symbols, (0,1), (0,2), and (1,1); Class II corresponds to the middle fixed codeword length for the following most frequent symbol (0,3) and a part of symbol (0,4); Class III consists of all other symbols, which are encoded using a longer fixed-length. We optimize the number of bits when using these three classes in the same manner as the previous sections. For Class I, the encoding process is the same as Table II. For class II, the three bits for the magnitude can use 000 111 to represent the values from 4 to 11. For class III, when RL=0, the four bits can be used to store magnitudes ranging from 12 to 27. If the actual coefficient magnitude exceeds 27, the value 27 is stored. When RL$\neq$0, the process is the same as the case with two classes.

However, if we compare the three-class scheme with the two-class scheme in the previous section, the improvement is quite limited. First, one more bit is required for class identification. In our methodology, the bit for class identification of a codeword is always placed in reliable storage to make sure the size of the codeword is always known accurately. Therefore, the amount of reliable storage needed to store the class identification bits increases as more classes are used. Second, adding the additional class for symbol (0,3) and a part of symbol (0,4) comes at the expense of adding one more bit for the other symbols in Class III. Fig. 4.13(b) shows that the distribution of images with different improvements in the total number of bits. The x-axis represents the improvement of the number of bits compared with using only the two-class scheme. It shows that the benefit of adding the additional class to reduce the total number of bits is very limited.

Considering that we obtain good improvement with only two fixed lengths, and that

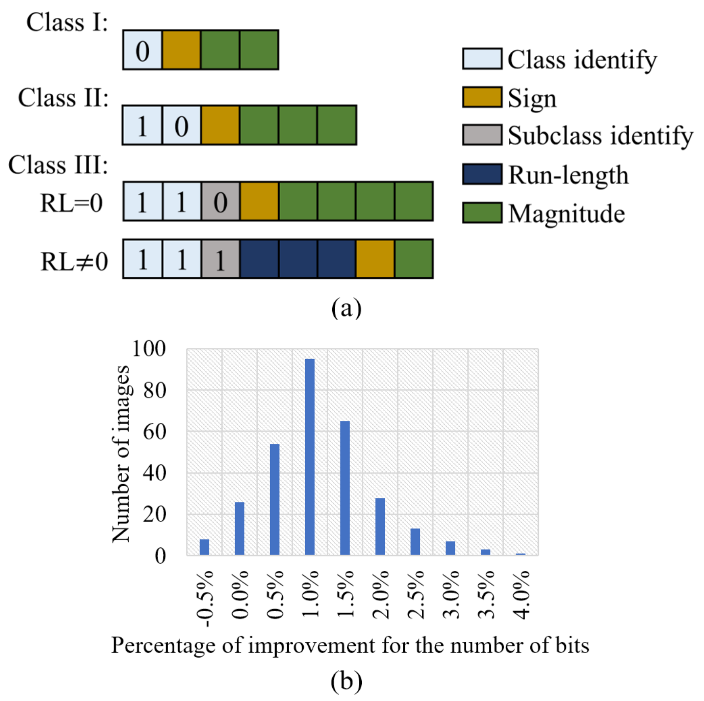


Figure 4.13: (a) Bit assignment for three-class scheme (b)The distribution of images with different improvement in the total number of bits.

adding more classes will not produce a large benefit, especially for low-cost ratio, we see that there is no benefit in using three or more lengths.

4.5 Evaluation methodology

To evaluate the performance of ALC-based storage compared to traditional JPEG-based storage, the following error injection and error correction models for storage are developed and the criteria for the quality of the result and the storage cost are also described.

4.5.1 Error model

The data array organization is based on the structure in NAND Flash memories and is a reasonable representation for a storage system built from any technology. A logical page is the smallest addressable unit for reading and writing, and each page is typically made up of a main area for data and a spare area for ECC [81].

We apply an error model based on [82] where the errors are randomly distributed in one page, which follows the random bit-error characteristics of NAND Flash memory. If the error rate of a storage bit-cell is p, then the probability that n cells fail in a storage array with size M follows the binomial distribution:

$$P\left(N=n\right)=\binom{M}{n}p^{n}\left(1-p\right)^{M-n} \tag{4.1}$$

4.5.2 Protection of error correction codes

Bose-Chaudhuri-Hocquenghem (BCH) codes are commonly used in storage [82] because they are efficient in correcting single-bit errors. If the BCH error correction capability is t bits, and the number of failed cells is n, which follows the distribution in Eq. (4.1), then the failure probability of error correction can be defined as:

$$P\left(n>t\right)=\sum_{n=t+1}^{M}\binom{M}{n}p^{n}\left(1-p\right)^{M-n} \tag{4.2}$$

The BCH scheme can construct a code with length 2^m-1, which includes data bits and parity bits, over the Galois Field $GF(2^m)$. The number of parity bits required for BCH can be computed as mt with error correction capability t [83]. In our case, the errors can occur in both data bits and parity bits with the same error rate, p.

4.5.3 Evaluation criteria

The quality of the image retrieved from the storage system, and the total cost in bits of the storage system, are used to compare the different coding schemes. In our evaluation, SSIM is used to measure the quality of the decompressed images. The SSIM is an index measuring the structural similarity between two images. It is a well-known objective image quality metric [84] [85]and is defined as

$$SSIM\left(x,y\right)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)} \tag{4.3}$$

where μ_x is the average of x, μ_y is the average of y, σ_{xy} is the covariance of x and y, σ_x^2 is the variance of x, σ_y^2 is the variance of y, c_1 and c_2 are two variables to stabilize the division with a weak denominator. It is valued between -1 and 1. When two images are nearly identical, their SSIM is close to 1 [80]. The baseline image for comparison is a conventional JPEG-compressed image with a quality factor of $Q = 90$, which we denote as $\text{SSIM}_{Q_{90}}$. If the quality of the image using our scheme is given by $\text{SSIM}_{\text{image}}$, then the percentage degradation is:

$$\text{Quality Degradation} = \frac{\text{SSIM}_{Q_{90}} - \text{SSIM}_{\text{image}}}{\text{SSIM}_{Q_{90}}} \times 100\% \tag{4.4}$$

Our evaluation places a user-specified limit on the maximum acceptable percentage quality degradation. Image data is stored in both inexpensive approximate storage and more expensive reliable storage. Therefore, C_{total}, the total cost of storing data, is the sum of the total cost associated with reliable and approximate storage. The cost of each of these components can be defined as the product of the cost per bit and the number of bits used in this type of storage. In other words, if we store N_r (N_a) data bits and E_r (E_a) ECC bits in reliable (approximate) storage, then the total cost is given by:

$$(N_r + E_r) \times C_r^{bit} + (N_a + E_a) \times C_a^{bit} \tag{4.5}$$

where C_r^{bit} and C_a^{bit} are, respectively, the cost per bit for reliable and approximate storage. Since our objective is to minimize Eq. (4.5), it is the relative cost between these components that is important, and therefore this minimization is equivalent to minimizing the function:

$$C_{total} = (N_r + E_r) + (N_a + E_a) \times r \tag{4.6}$$

where the cost ratio r can be defined as

$$r = \frac{C_a^{bit}}{C_r^{bit}} \tag{4.7}$$

We now consider the cost of various types of image storage schemes:

Conventional VLC-based JPEG image storage method: This uses VLC to store Huffman-coded compressed JPEG images. The high sensitivity to error (demonstrated in Fig. 1.1) implies that all data must be stored in reliable storage, i.e., $N_a = E_a = 0$, and E_r is determined by the number of BCH protection bits that are used.

Our basic approximate storage scheme: We use ALC encoding to enhance the error-resilience of data stored in approximate memory. This implies that N_r is relatively small and most of the stored image data corresponds to N_a. Specifically, the bits stored in reliable storage correspond to:

- The number of additional bits and the number of codewords with these additional bits, which are the same for all blocks.
- The DC coefficients, stored using VLC.
- Five bits per block that represent the number of symbols in the block, as described in Section 4.2.3.
- Critical data for the AC coefficients, as described in Section 4.3.

For a given image with cost $C_{total,\text{image}}$, we define the percentage cost improvement relative to the $C_{total,Q_{90}}$, the cost of conventional VLC-based JPEG image storage with $Q = 90$ as:

$$\text{Cost Improvement} = \frac{C_{total,Q_{90}} - C_{total,\text{image}}}{C_{total,Q_{90}}} \times 100\% \tag{4.8}$$

4.6 Experimental results

We evaluate the performance of the proposed ALC algorithm using the basic structure of a generic NAND flash memory [81]. The main area of a page contains 2KB data and it will be further split into four subpages. Each subpage contains 4096 bits of data. ECC is applied to each subpage. The organization of a page is depicted in Fig. 4.14. The four subpages are continuous in a page and their corresponding ECC parity bits are located in spare area at the end.

The error rate, p, of reliable storage is 10^{-6} and the common storage industry reliability requirement corrects bit errors to reach a failure probability of 10^{-15} [86]. To satisfy this reliability requirement, eq. (4.2) indicates that we need an error correction capability of $t = 4$. In our experiments, we evaluate approximate storage at error rates of $p = 0.1\%$, 0.5%, 1% and 1.5%. These error rates are much higher than normal storage

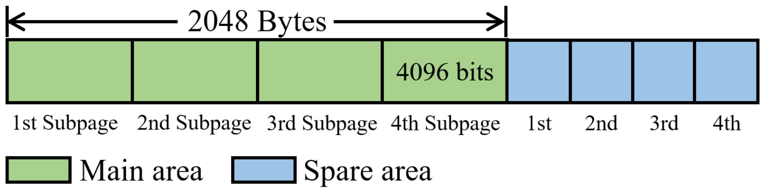


Figure 4.14: Organization of a page with the large page divided into small subpages to better implement ECC.

as our scheme investigates the use of low-cost, low-reliability storage in order to reduce the overall cost. If the approximate storage tries to reach 10^{-15} failure probability by using ECC, the number of required parity bits becomes prohibitive. For example, when $e = 1\%$, there is a 10^{-15} probability that the number of bit flips will be greater than 123 based on Eq. (4.2). In other words, we should set the error correction capability parameter to $t = 123$ to ensure the reliability of approximate storage. Based on Section 4.5.2, for each 4096 bits of data ($GF(2^{13})$ and $m = 13$), $123 \times 13 = 1599$ parity bits are required, corresponding to an overhead of 39% for ECC (i.e., 71% efficiency). In contrast, for the same value of e, in our method, reliable storage requires 52 parity bits, with an overhead of 1% (i.e., 99% efficiency). At a cost ratio of $r = 0.9$, our unreliable storage requires no ECC bits for this error rate, and thus has an overhead of 0% (i.e., 100% efficiency); when $r = 0.3$, the overhead is 12% (i.e., 89.5% efficiency).

Therefore, only weak ECC protection is applied to approximate storage. If the number of errors is beyond the ECC error correction capability, all the errors in this subpage cannot be corrected. For traditional SSD, once this situation happens, the operating system will be notified that these data are corrupted. However, in our approximate storage scheme, we allow errors to happen and the operating system does not need to be notified [7]. Therefore, for approximate storage, various values of t for BCH map to different failure probabilities of each subpage, as shown in Fig. 4.15, which is generated using Eq. (4.1) and Eq. (4.2). The weak ECC protection results in a higher failure probability than high-reliability storage. For example, when $e = 0.5\%$, a choice of $t = 20$ can correct 40% of the errors in the subpages successfully. For different cost ratios, r, defined in Eq. (4.7), the capability t is varied to find the minimum total cost, as described in Section 4.5.2.

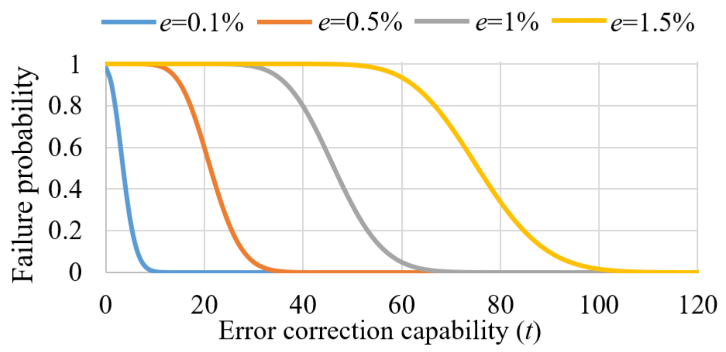


Figure 4.15: Failure probability of error correction for different error rates, p, as a function of the error correction capability, t.

Our simulations inject errors into 500 image samples from the Imagenet dataset [79]. The resolutions of these image samples range from 1600×520 to 144×144. Since we used the images Girlface, Peppers, Tiger and Bird to set the parameters of the ALC scheme, for a fair evaluation, these images are excluded from the set of 500 images that are used in our evaluation.

We compare our JPEG-based ALC scheme only with the JPEG-based VLC scheme in the evaluation. Our objective is to place the data into approximate storage, while maintaining an acceptable quality and compression efficiency. Thus, our objective is fundamentally different from prior methods described in Chapter 1, such as [9], which stores raw images (instead of JPEG), or [7], which uses the relatively rarely-used PTC compression format. Moreover, the impact of the block-based errors is large when techniques such as [1] [42–49], described in Chapter 2, are used. Therefore, only traditional JPEG and JPEG with a reduced Q [27] are compared in this section.

4.6.1 Quality degradation of ALC algorithm

Inherent quality degradation

As illustrated in Section 4.2, ALC-based storage results in inherent quality degradation due to approximations in representation and the impact of some discarded coefficients. Fig. 4.16(a) shows the distribution of quality degradation for the above set of 500 images, considering only ALC-inherent information loss. We also show the quality in terms of peak signal-to-noise ratio (PSNR) as the comparison for the SSIM degradation results, as shown in Fig. 4.16(b). In the absence of errors, we use $median\_value + \alpha \times (max\_value - median\_value)$ with $\alpha = 0.25$ to find the additional bits and apply it to the first $T = 10$ AC codewords to achieve the minimum degradation for each image. This is the largest value in our experiments, as described in Section 4.2.2. The figures indicate that the image quality histograms for these images are concentrated around the small values of the degradation metric or large value for PSNR, so that the ALC algorithm can effectively maintain most of the information for each image.

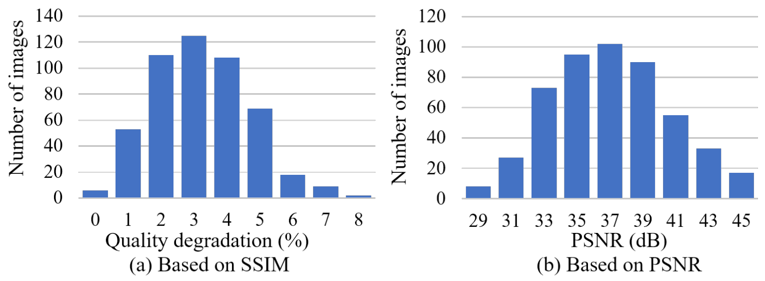
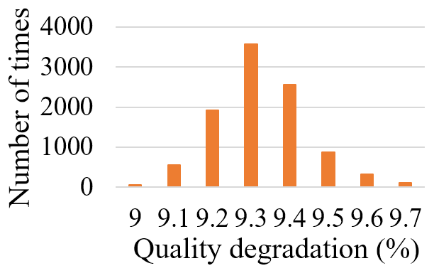


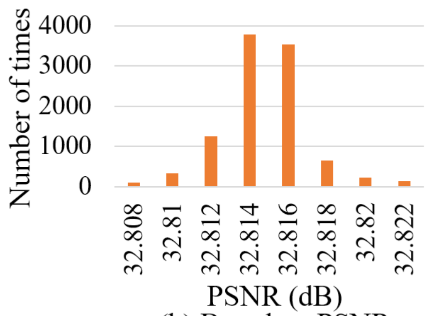
Figure 4.16: The distributions of ALC inherent quality degradation based on SSIM and based on PSNR caused only by the approximate representation for 500 different images.

Quality degradation for various error locations

Additionally, for the same number of errors caused by approximate storage and a fixed data partition between reliable and approximate storage, we perform 10^4 Monte Carlo samples on one randomly selected image (other images show similar results). The difference between the samples is in the error locations, which are randomly distributed. In our simulation, we use the expected value of the number of errors with 1% error rate, so that the number of errors in each sample is fixed. Each sample uses the original image and the errors will be randomly distributed again. The quality degradation for each sample will be computed and we get the histograms of these degradation values, as shown in Fig. 4.17. The quality degradation of these samples are seen to vary within a small range, as shown in Fig. 4.17. Therefore, unlike traditional VLC-encoded JPEG, the ALC algorithm is less sensitive to the location of the errors that occur in less important data. In other words, the quality degradation of an image in ALC can be controlled.



(a) Based on SSIM



(b) Based on PSNR

Figure 4.17: The distributions of 10^4 Monte Carlo samples on one image with different error locations caused by approximate storage.

Visual output of ALC scheme

Fig. 4.18 shows the results of using the ALC scheme to store the Girlface image under a 1% error rate. The three images correspond to different data partitions between reliable and approximate storage, resulting in different levels of quality degradation, as computed using Eq. (4.4), of approximately 5%, 10% and 20%. The values of PSNR also decrease for the larger degradation and the quality performance in terms of PSNR shows the same trend compared with SSIM. Therefore, only SSIM degradation is shown in the following parts. It can be seen from the figures that the errors in the former two cases are barely discernible. Even for the latter case, due to the built-in error-resilience of ALC, the errors affect the quality only in a localized manner instead of affecting the image globally, as in the VLC cases shown in Fig. 1.1. As the degradation increases, more obvious errors appear in the image, as shown in the zoomed-in details. For our other test images, Peppers, Tiger and Bird, the quality degradation results (not shown here) are similar, leading to the conclusion that a quality degradation of at most 10% is acceptable.

Compatibility with various quantization table JPEG techniques

More sophisticated JPEG techniques have been proposed, that use image-specific quantization tables [36] or that directly reduce the quality factor Q of the quantization table [27] to achieve further compression efficiency. Our ALC-based scheme is orthogonal to these techniques. Fig. 4.19 shows the quality degradation of ALC and JPEG

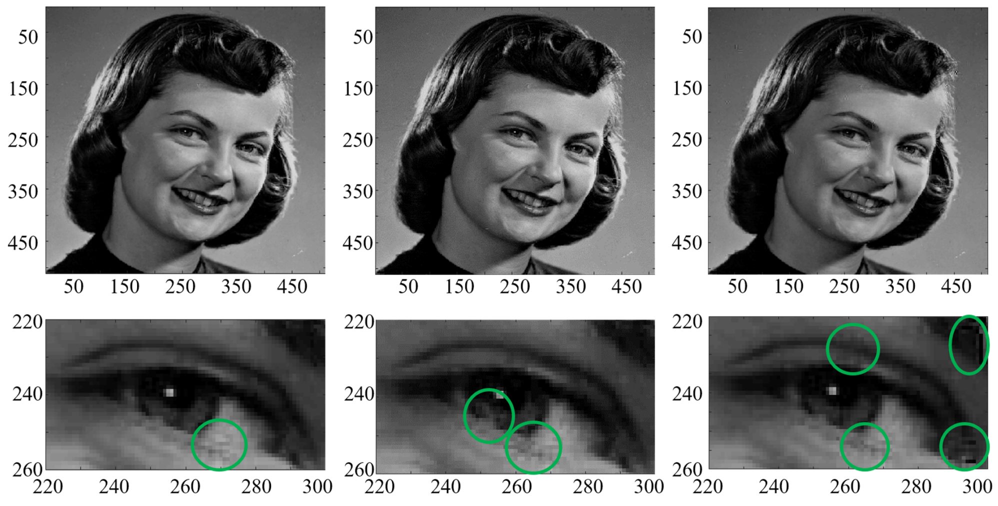


Figure 4.18: Visual output of the ALC scheme using 1% error injection under different quality degradations in the original form (upper) and zoomed in on the details (lower). Due to space constraints, the upper row of figures is shrunk from the original size of the image. However, the errors can be observed at original size and more obvious errors occur in the figures with larger quality degradation.

schemes for various quality factor Q of the quantization table. These results show the average value of all 500 images from the dataset. For each image, the number of bits for the ALC scheme is constrained to be nearly the same but no larger than JPEG-based scheme. Our goal in this section is to quantify the quality degradation caused by quantization, and therefore, the impact of errors in approximate storage is not considered. For large quality factor Q, the ALC scheme has greater quality degradation than JPEG due to approximations in representation of the values. However, thanks to the additional bits adaptively added to compensate for quality loss, the degradation of JPEG can be followed by the ALC scheme closely. As the quality factor Q decreases, the limited length of codewords for the ALC scheme can cover most situations and the degradation of the ALC scheme is dominated by the quantization. Therefore, the number of bits for ALC-encoded images can be further reduced by decreasing the quality factor, as is done with JPEG.

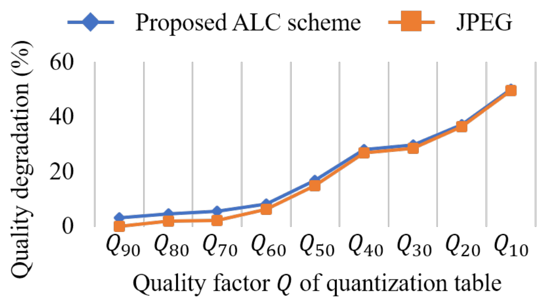


Figure 4.19: Quality degradation for ALC scheme and JPEG with the same number of bits.

4.6.2 Impact of the error rate on ALC-based storage

For the case of a mainstream non-volatile storage technology, at different error rates, p, in approximate storage, Fig. 4.20 shows the number of bits is placed in reliable and approximate storage using the ALC algorithm and JPEG with a same quality degradation. The results are generated based on a representative image out of the 500 evaluated testcases in Imagenet. Other images show similar trends. Due to the sensitivity of VLC to errors, all of the data for JPEG must be saved in reliable storage. Thus, the number of bits for JPEG is independent of the error rates in approximate storage and remains constant for all cases.

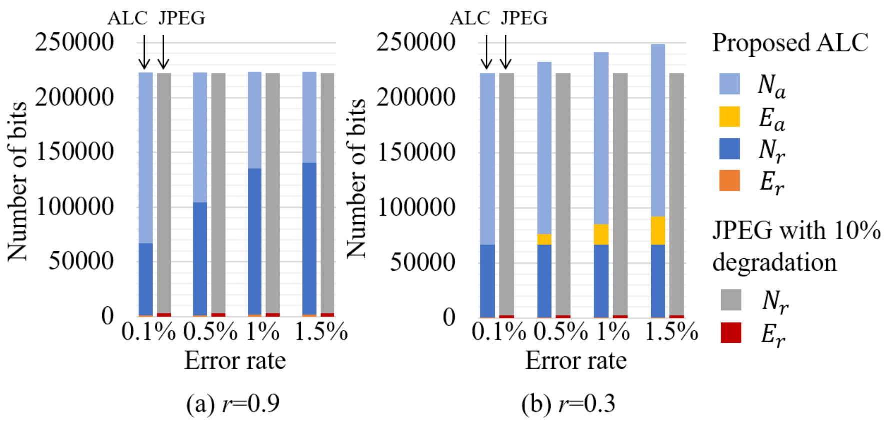


Figure 4.20: Number of bits in reliable and approximate storage for the ALC scheme, as compared with VLC-encoded JPEG using reliable storage.

We use a stacked bar chart to show the different fraction of the number of bits in these two schemes with various colors, where $E_r + N_r$ correspond to the total number of bits in reliable storage, and $E_a + N_a$ correspond to the bits in approximate storage. For each subfigure, the left stacked bar represents our ALC scheme and the right one represents the JPEG scheme. With a degradation specification of 10%, evaluated according to Eq. (4.4), the total number of bits for our ALC-based scheme is nearly the same as JPEG at $r = 0.9$ and not more than 11% compared with JPEG in the worst case at $r = 0.3$. Moreover, the increase for the total number of bits is caused by E_a for the low-cost ratio $r = 0.3$ scenario.

When cost ratio r is large, for example $r = 0.9$, it is difficult to obtain a cost benefit for the weak ECC in the approximate storage compared to directly placing the data in reliable storage. Therefore, the limited or even no ECC is applied when r is large. For different error rates, the various data partitioning patterns are used to satisfy the quality requirement instead of changing the capability of the weak ECC. As the error rate decreases along the x-axis, the total number of bits stays nearly the same and the percentage of bits that can be placed in reliable storage decreases from 68.0% at $e = 1.5\%$ to 29.9% at $e = 0.1\%$, as shown in Fig. 4.20.

When the cost ratio r is small, for example $r = 0.3$, the weak ECC protection allows fewer data to placed in reliable storage and the large cost difference between approximate storage and reliable storage can compensate for the cost of the parity bits for the weak ECC. As shown in Fig. 4.20(b), for different error rates, the number of bits placed in reliable storage remains the same, using only the first data partitioning pattern defined in Section 4.3. As the error rate decreases, the number of parity bits required for the weak ECC decreases, so the number of bits placed in the approximate storage decreases.

4.6.3 Impact of the cost ratio r on ALC-based storage

For any non-volatile memory technology, the relationship between the error rate, p, and the cost ratio, r, are strongly context-dependent. Even for the same type of storage technology, this relationship may vary from manufacturer to manufacturer, or from year to year. Meanwhile, the target of our paper is providing a methodology to allow the image to be compressed and placed in approximate storage. To preserve the generality of the approach and to provide a clear view of the landscape, we do not use the specific cost ratio or price, so that anyone can use any ratio when implementing our methodology. Therefore, rather than using a fixed function that maps p to r, we evaluate the proposed ALC-based storage scheme for general non-volatile solid-state storage by conducting a sweep of the value of the cost ratio, r, for different error rates, p.

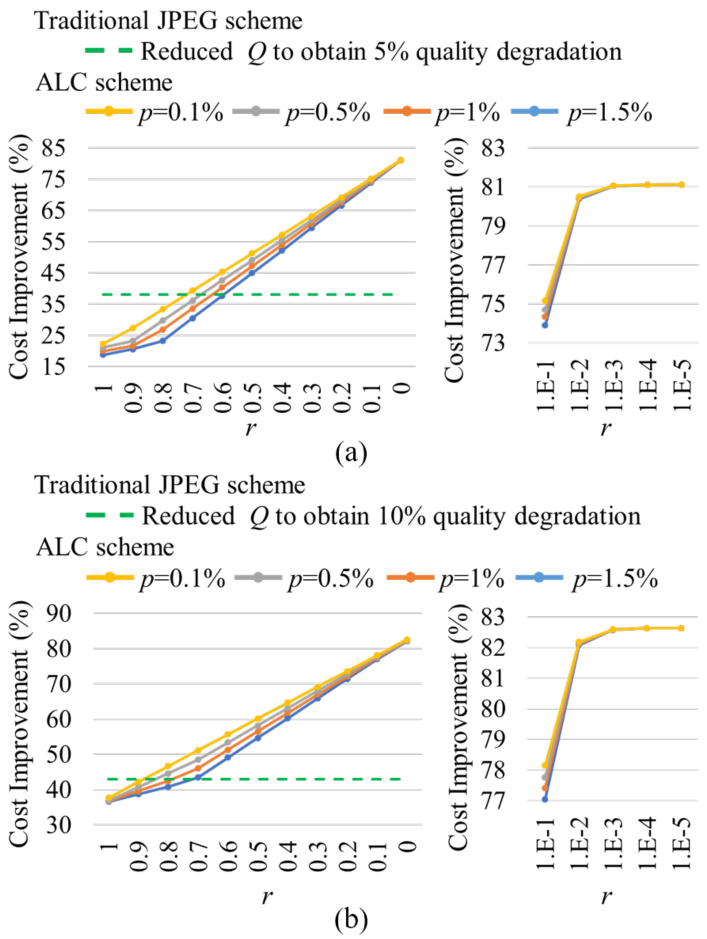


Figure 4.21: Cost improvement for various values of r and p for the ALC scheme (left). The right graph expands the range from 10^{-1} to 10^{-5}.

Using the original JPEG scheme as the baseline, Fig. 4.21 shows the cost improvement for our approach compared to the VLC-based JPEG approach that uses a reduced Q value, which has a similar concept in [27]. In both cases, the quality degradation is constrained to be under 5% (Fig. 4.21(a)) and 10% (Fig. 4.21(b)). The results show the average of all 500 images from the Imagenet dataset. We sweep the value of r from 1 to 0 and also zoom in on the range from 10^{-1} to 10^{-5} to show the details for these smaller possible values. The dashed green line represents the conventional JPEG scheme with the adaptively reduced Q factor. The remaining lines show the ALC scheme with different error rates, p.

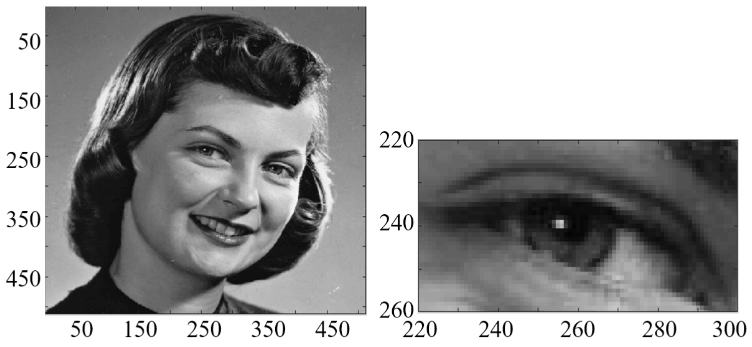
The JPEG version with a reduced Q value lowers the cost compared to the baseline due to its reduced storage requirements. This improvement is independent of r and p since all data are placed in reliable storage. When the degradation is constrained to be less than 5% and 10%, the results of reducing the Q value in the JPEG encoding changes the texture of the image, as shown in Fig. 4.22, but the image stills appears acceptable. The other test images, Girlface and Peppers, show similar results and are not included here due to space limitations.

For our ALC-based storage scheme, when r is large, there is no advantage compared to using JPEG with a smaller Q value since the data volume is similar or even larger than the reduced-Q JPEG case. However, as the cost ratio, r, gets smaller, the benefit of using approximate storage increases due to its lower cost, as shown in Fig. 4.21(a) and Fig. 4.21(b).

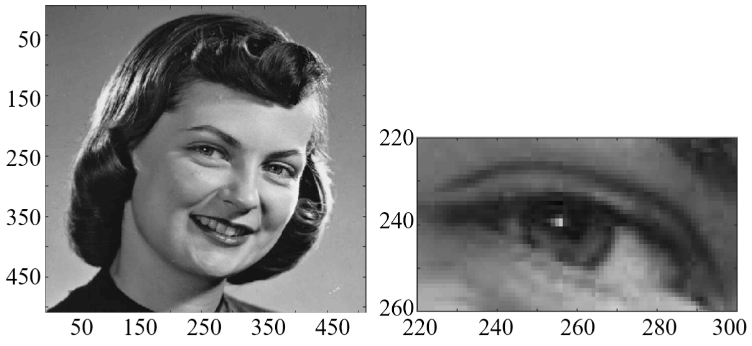
Section 4.6.2 showed that, when r is close to 1, there is no advantage to partitioning the data between reliable and approximate memory since they cost about the same. When r is close to 0, the weak ECC protection in the approximate storage allows the first pattern of data partitioning for any error rate, which means that the number of bits placed in reliable storage is fixed. When the cost of the approximate storage is very low relative to the reliable storage, the total cost of the storage system is dominated by the number of bits in reliable storage. Therefore, the cost improvement converges as r approaches 0 so that all of the curves for the different error rates nearly coincide with each other in Fig. 4.21.

4.6.4 Limit to ALC-based storage

According to the analysis in Section 4.6.3, when the cost ratio is small, the entire cost of the storage system is dominated by the number of bits stored in reliable storage, which contains all of the DC coefficients, the block information, and critical data for



(a) $Q = 85$, 4.61% degradation



(b) $Q = 75$, 9.61% degradation

Figure 4.22: Visual output of the JPEG encoded images with various Q factor reductions showing different quality degradations (Eq. (4.4)) in the original form (upper) and zoomed in on the details (lower). There is a slight degradation from (a) to (b), but both are acceptable.

the AC coefficients. For each image, the block information is fixed. When the cost ratio is small, the first pattern of data partitioning is always used, which fixes the critical data for the AC coefficients. To further reduce the cost, the only thing we can do is decrease the number of bits used to store the DC coefficients in the reliable storage.

In our case, all DC coefficients are still stored in reliable storage encoded using VLC, but the total number of bits allocated to the DC coefficients is decreased by reducing the resolution of the values. This requires a custom quantization matrix in which only the first entry (1, 1) is changed to a larger value compared to the default matrix. If we increase only the first entry while keeping the other entries the same as the $Q = 90$ matrix, the quality will degrade and the number of bits used to store the DC coefficients

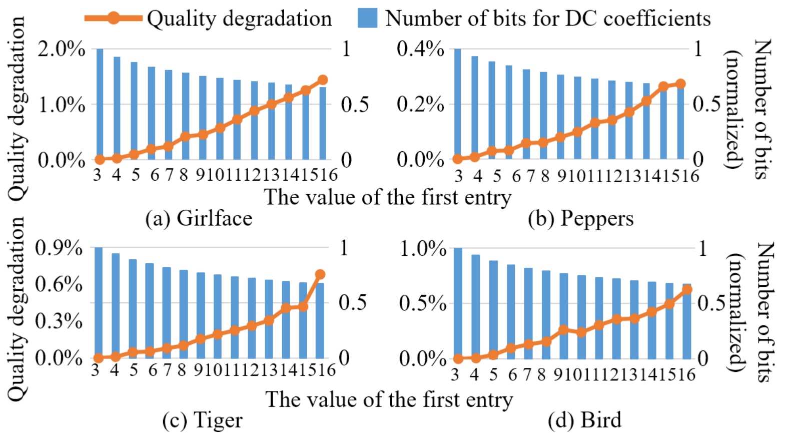


Figure 4.23: Impact of the value of the first entry of the quantization matrix on the image quality and the number of bits for the DC coefficients.

will decrease, as shown in Fig. 4.23. The first entry in the default quantization matrix with $Q = 90$ is 3, so the simulation starts from 3. This point has no quality degradation and requires the largest number of bits to store the DC coefficients. This figure shows that, when the value of the first entry increases to 16, the quality degrades less than 2% for all four images, while the number of bits decreases at least 34%. Fig. 4.24 shows that, when the first entry is 16, the visual output maintains an acceptable quality with only limited texture loss.

We conclude, directly increasing the size of the first entry of the quantization table to decrease the total number of bits used for the DC coefficients can further improve the cost of the ALC encoding.

4.7 Conclusion

This chapter has introduced a new adaptive-length coding (ALC) algorithm that can be used to reduce the cost of a long-term "cold storage" system for image data by taking advantage of low cost memory devices that can have extremely high bit error rates. The ALC scheme partitions the bits used to store the encoded image into two classes – the most significant data that needs to be highly reliable, and the less important data that can tolerate some errors. Due to the built-in error-resilience of the ALC scheme, the errors that occur in the less important data, which is placed in low-cost approximate

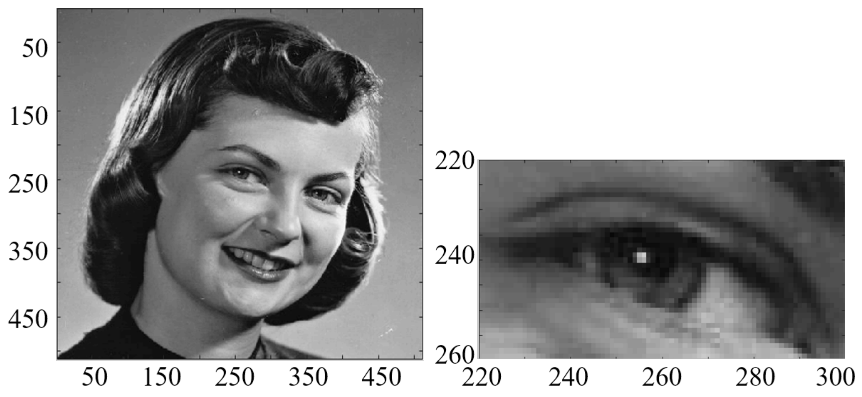


Figure 4.24: The visual output for changing the first entry of the $Q = 90$ quantization matrix to 16.

storage, affects the quality of the retrieved image in only a localized manner so that any changes in the images due to errors are barely discernible. Furthermore, when the difference in cost between the reliable storage and the unreliable (approximate) storage is large, the fraction of each encoded block that should be placed in the approximate storage depends on the error rate of the memory and the quality degradation limit desired by the user. Finally, the cost improvement of the ALC encoding increases as the cost ratio reduces and, when larger quality degradation can be tolerated, ALC can achieve even greater cost benefits.

The target of our paper is providing a methodology to allow the image to be stored in a compressed format and placed in approximate storage. Thus, we develop a new error resilient coding scheme. In this work, we examine only the fundamental idea instead of addressing system-level issues. Our objective is to make it clear that there is a viable trade-off between storage cost and image quality. Details of the implementation are specific to the precise storage scheme used and involve many subtleties beyond what can be considered in a single paper. These system-level concerns are a topic for future work.

Chapter 5

DCT-based Approximate Communication to Improve MPI Performance in Parallel Clusters

In this chapter, we study several representative MPI-based error-resilient applications. The DCT-based approximate communication scheme is described in this chapter. To compute the DCT coefficients more efficiently, subband decomposition and a fast recursive DCT with piecewise-constant approximation scheme are proposed [87]. We exploit the error tolerance of these MPI-based applications to improve communication efficiency by substantially reducing message lengths on a real cluster system.

5.1 MPI-based error-resilient applications

Approximate communication can be applied to error-resilient applications for speedup while maintaining the accuracy of the output at an acceptable level. In our study, eight representative MPI-based error-resilient applications were used for the evaluation of the proposed method. These applications spanned different domains: physical simulation, machine learning, and image processing. Their computing tasks either did not aim at an exact numerical answer or they had inherent resilience to output error. Table 5.1 summarizes these applications. The definitions of the evaluation metrics in Table 5.1

are shown below:

Table 5.1: Summary of applications.

Application	Description	Input	Evaluation Metric
FE [88]	Finite element method	300^3 for length of 3D	Relative residual norm
LULESH [89]	Unstructured Lagrange explicit shock hydrodynamics	30^3 for length of mesh	Mean relative difference
KNN [90]	K-nearest neighbors classification	Skin segmentation dataset	Similarity of predicted labels
BP [91]	Backpropagation neural network learning	CMU face image dataset	Accuracy of test cases
Sweep3D [92]	Models a wavefront propagating communication pattern	100^3 for length of mesh	Mean value range-based relative difference
Halo3D [92]	Models nearest neighbor communication pattern	100^3 for length of mesh	Mean value range-based relative difference
Edge [79]	Edge detection in image	1200×800 images	Structural similarity index
Blur [93]	Image blur filter	11500×11500 images	Structural similarity index

For FE, an iterative method is used for solving the linear equation $Ax = b$. The error tolerance of the results is defined by the user as a stopping criterion for the iterative process. In this study, the error tolerance was based on the relative residual norm and defined as $\|r_j\|_2 \,/\, \|r_0\|_2$, where $r_j = b - Ax_j$ of the jth iteration.

For LULESH, the final origin energy and the three measures of symmetry were calculated after the simulation [89]. The mean relative difference was compared with these variables using the non-compression scheme.

For KNN, similarity can be defined as n_{same}/n_{total}, where n_{same} is the number of the test cases with the same predicted labels generated by the approximate communication scheme and the non-compression scheme, and n_{total} is the total number of test cases.

For BP, the accuracy is defined as $n_{correct}/n_{total}$, where $n_{correct}$ is the number of the test cases with correct recognition, and n_{total} is the total number of test cases.

For Sweep3D and Halo3D, the value-range-based relative difference can be defined as $e_i = (x_i - \widetilde{x_i})/(x_{max} - x_{min})$, where x_i is the original value and $\widetilde{x_i}$ are the reconstructed data. x_{max} and x_{min} are the maximum and minimum values in the original data. In our evaluation, the average value E of e_i was used as error metric for these two applications.

For Edge and Blur, the SSIM [80] was used to measure the quality of the results, which is defined in Equation (4.3) in Section 4.5.

When using MPI, the messages are passed between computing nodes as arrays of data. The type of data can be double, int, char and so on. The patterns of the messages in these applications were impacted by their own algorithms or the input

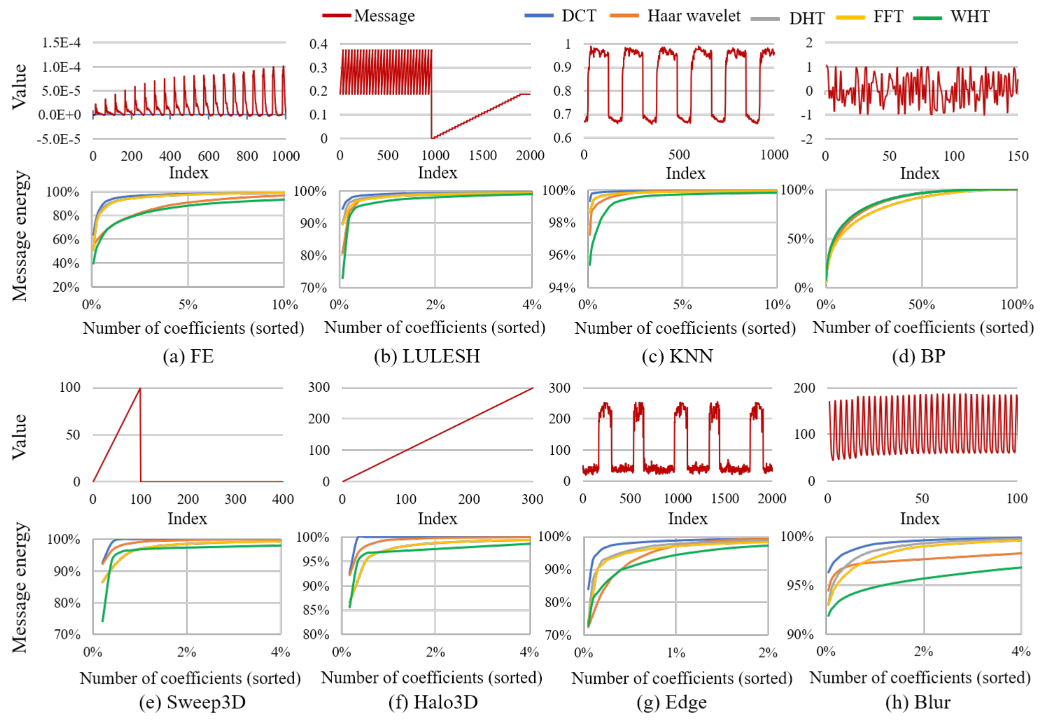


Figure 5.1: Representative messages extracted from various applications (upper one in each subfigure) and energy compaction for various transform-based compression methods with different applications (lower one in each subfigure).

data, which showed different levels of randomness. The first row of each subfigure in Fig. 5.1 shows a representative part of the message extracted from the different applications. In these subfigures, the x-axis represents the index of this data array and the y-axis represents the corresponding value of the data. For example, the pattern of FE application has a non-random characteristic and is visibly periodic, as shown in Fig. 5.1(a). This characteristic of the message pattern allows the transform-based compression algorithm to compress the message efficiently. For BP application, its pattern exhibits a more random characteristic, as shown in Fig. 5.1 (d). The non-random characteristic of the message pattern can be exploited, and more details are discussed in the next section.

5.2 Message energy compaction of DCT

In signal processing, the energy of a message is measured by the sums of squares of the coefficients in the frequency domain, defined as

$$E = \sum_{n=0}^{N} |X(n)|^2 \tag{5.1}$$

where $X(n)$ is the frequency domain transform of message $x(k)$ with length N. When the messages of an application have a non-random pattern as shown in Section 5.1, transform-based compression algorithms can maintain as much message energy as possible with few coefficients. Only a part of coefficients in the frequency domain is used to represent a message to implement compression. Using coefficients with higher message energy can yield more accurate results compared with the original message.

The DCT compression algorithm delivers higher energy compaction than other compression algorithms. Our target in applying transform-based compression is to use few coefficients to represent as much information regarding a message as possible. In other words, most of the energy in a message is concentrated in a few coefficients. DCT is the best choice of transformation algorithm for this among such competitors as discrete wavelet transform (e.g., Haar wavelet), discrete Hartley transform (DHT), fast Fourier transform (FFT), and the Walsh–Hadamard transform (WHT).

In the second row of Fig. 5.1, we apply the above compression algorithms to representative messages extracted from the MPI-based applications listed in Table 5.1. The x-axis represents the number of coefficients expressed as a percentage. These coefficients are sorted in descending order based on their absolute values. A larger absolute value of a coefficient in the frequency domain represent higher energy of a message, as shown in Equation (5.1). Therefore, using the first n of the largest coefficients to represent a message can effectively capture a large fraction of the message energy. The y-axis represents the energy of the message. Compared to energy compaction using the same number of coefficients, the results show that the DCT can pack the energy of the spatial sequence into as fewer frequency coefficients than other compression algorithms. For instance, in Fig. 5.1(a), only 1% of the coefficients can represent 95% of the energy of a message using the DCT.

Some applications have highly random message patterns that cannot be effectively compressed by using a transform-based compression algorithm. For example, 45% of the coefficients were required to represent 95% of the message energy in the BP application as shown in Fig. 5.1(d). Our approximate communication scheme is not a good choice for this type of message pattern. The number of coefficients used to represent high-level message energy can thus be used as a criterion to decide whether to use our scheme. As illustrated in Algorithm 1, a message is extracted from the application and, if it can represent sufficiently large energy ($E_i \geq \theta_e$) using a small number of DCT coefficients ($i/n \leq \theta_n$), our proposed scheme is a good candidate for approximate communication, where θ_n and θ_e are empirically determined threshold values.

Algorithm 1: Selection Algorithm for Proposed Scheme

```
Input: $M$ and $n$: message and message length;
$\theta_n$: number of coefficient thresholds; $\theta_e$:energy threshold;
Output: $flag_s$: Decision on selection of our scheme;
compute DCT coefficients $C_M$ of $M$;
initial $C_L = \Phi$; $flag_s = 0$; $i = 1$;
while $i/n \leq \theta_n$ do
    push the $i$th largest coefficient of $C_M$ to $C_L$;
    compute the message energy $E_i$ using $C_L$;
    if ($E_i \geq \theta_e$) then
        $flag_s = 1$; break;
    end
    $i = i + 1$;
end
return $flag_s$;
```

5.3 DCT-based approximate communication scheme

For the conventional DCT algorithm, the nth DCT coefficient of a sequence $x(k)$ with length N is defined as

$$X(n) = \sum_{k=0}^{N-1} x(k) \, cos\left(\pi (2k+1) \frac{n}{2N}\right) \tag{5.2}$$

where n ranges from 0 to $N-1$. To develop the runtime compression of MPI messages among nodes, low time overheads for compression and decompression are necessary.

We propose the DCT-based approximate communication scheme in three steps: a subband decomposition of the message (Section 5.3.1), fast recursive DCT with piecewise-constant approximation (Section 5.3.2), and zero run-length coding (Section 5.3.3). The proposed approximate communication scheme can pack the energy of the message into a few coefficients while substantially reducing the time needed for compression compared with the conventional DCT for large messages.

5.3.1 Subband decomposition

The subband decomposition of a message [39] can be applied to the message compression to reduce the time overhead in the first step. Fig. 5.2 shows the absolute value of coefficients of the DCT in a representative message of a FE application generated by the conventional DCT. As shown in Fig. 5.2, the lower-frequency band concentrates coefficients with larger absolute values. Based on the definition in Equation (5.1), it is reasonable to assume that most energy of the message is located in the first half of the coefficients belonging to the lower frequency. Once a message satisfies the above assumptions, subband decomposition can be used on it while slightly sacrificing the quality of the result. In Section 5.3.2, Fig. 5.5 further shows that our DCT-based approximate communication scheme with subband decomposition has a limited impact on the energy of the message for most applications that we evaluated.

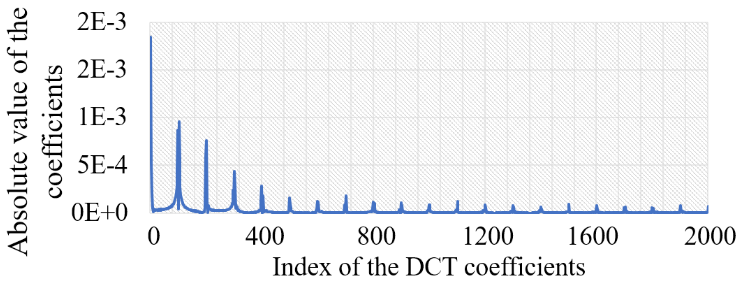


Figure 5.2: The absolute value of DCT coefficients generated by the conventional DCT scheme.

To implement subband decomposition, sequence $x(k)$ with length N can be decomposed into a low-frequency band $x_L(k)$ and a high-frequency band $x_H(k)$, given by

$$\begin{aligned} x_L(k) &= \frac{1}{2}\left(x\left(2k\right) + x\left(2k+1\right)\right) \\ x_H(k) &= \frac{1}{2}\left(x\left(2k\right) - x\left(2k+1\right)\right) \end{aligned}$$

where k ranges from 0 to $N/2-1$. Based on the derivation in [39], when n varies from $N/2$ to N, $X(n) = 0$; when n varies from 0 to $N/2-1$, the DCT coefficients can be approximately defined by

$$\begin{aligned} X(n) &\approx 2cos\left(\frac{\pi n}{2N}\right)\sum_{k=0}^{\frac{N}{2}-1} x_L(k)cos\left(\pi(2n+1)\frac{n}{N}\right) \\ &= 2cos\left(\frac{\pi n}{2N}\right)X_L(n) \end{aligned} \tag{5.3}$$

The original DCT algorithm with length N $(X(n))$ can be approximately computed by using DCT $(X_L(n))$ of length only $N/2$, which reduces the complexity of the algorithm by half. Moreover, this approximation based on subband decomposition can be used repeatedly to achieve $N/4$ subband-approximate DCT and narrower subband DCTs. To maintain an acceptable quality of results, $N/2$ subband-approximate DCT was applied here.

In the proposed DCT-based approximate communication scheme, $x_L(k)$ is first computed based on the original sequence $x(k)$. The fast recursive DCT with piecewise-constant approximation described in Section 5.3.2 is then used to compute $X_L(n)$. Finally, $X(n)$ can be obtained based on Equation (5.3). Moreover, to reduce the computational complexity of this process, the piecewise-constant approximation is applied to item $2cos(\pi n/2N)$, as described in Section 5.3.2.

5.3.2 Fast recursive DCT with piecewise-constant approximation

To compute $X_L(n)$ efficiently as described in Section 5.3.1, a fast recursive DCT with piecewise-constant approximation scheme is proposed here.

While not computing the DCT exactly, approximations of it can provide meaningful estimations at low complexity to reduce the time overhead. Different techniques of DCT approximations have been considered, such as the integer DCT [94] [95], signed

DCT [96], and rounded DCT [97] [98]. The approximate DCT transform matrix needs to be computed and stored in advance for them. When they are applied to a small length of the DCT, e.g., eight-point DCT, the size of the matrix occupies a negligibly small amount of space. However, the fast approximate DCT for the long message is required because nearly no time can be saved by compressing a small message for MPI communication. Thus, the threshold to apply compression here was set to 4 KB, as was used in [60]. Moreover, the lengths of the messages are likely not the same, so that it is hard to prepare the DCT transform matrix for these various lengths in advance. A fast recursive DCT compression algorithm combined with piecewise-constant approximation is proposed to speed-up the DCT process with a small space overhead, especially for long DCT transformations.

The recursive DCT algorithm [99] is used as basis for our proposed scheme. It is a fast 1D exact DCT algorithm that uses fewer arithmetic operations than the conventional DCT. The recursive DCT provides stable generalization for longer DCTs and a simple format of the transformation compared with other fast DCT algorithms [100] [101]. In the recursive DCT algorithm [99], the DCT coefficients can be divided into even and odd parts, for n from 0 to $N/2-1$,

$$X(2n) = G(n) \tag{5.4}$$

$$X(2n+1) = H(n) - X(2n-1),\ (X(1) = H(0)/2) \tag{5.5}$$

where $G(n)$ and $H(n)$ are the DCT coefficients of $g(k)$ and $h(k)$, respectively. For k from 0 to $N/2-1$, $g(k)$ and $h(k)$ are defined as

$$g(k) = x(k) + x(N-1-k) \tag{5.6}$$

$$h(k) = q(k) \times (x(k) - x(N-1-k)) \tag{5.7}$$

where $q(k)$ is defined as

$$q(k) = 2cos\left(\frac{(2k+1)\pi}{2N}\right) \tag{5.8}$$

The computation of the DCT coefficients is now decomposed into two half-length DCT computations.

Fig. 5.3 shows the process of the recursive DCT algorithm. The left part of the figure represents the sequence in a different recursive call and the right part represents the DCT coefficients of the sequence on the same line. The input to the algorithm is $x(k)$ and the output is $X(n)$. As shown in Fig. 5.3, the half-length sequences $g(k)$ and $h(k)$ are first computed based on the original sequence $x(k)$. This process can be further divided until the length of the sequence reaches one, shown as the last line in the left part of Fig. 5.3. Based on the definition of the DCT in Equation (5.2), the DCT coefficient is equal to the original value of the sequence if the sequence contains only one point. Therefore, the DCT coefficients of all one-point sequences in the last line are available. Based on Equations (5.4) (5.5), the DCT coefficients can be computed recursively from bottom to top until the target $X(n)$ with length N is reached. In this process, we assume that the length of the input sequence is always a power of two number. Otherwise, zero padding is applied at the end of the message.

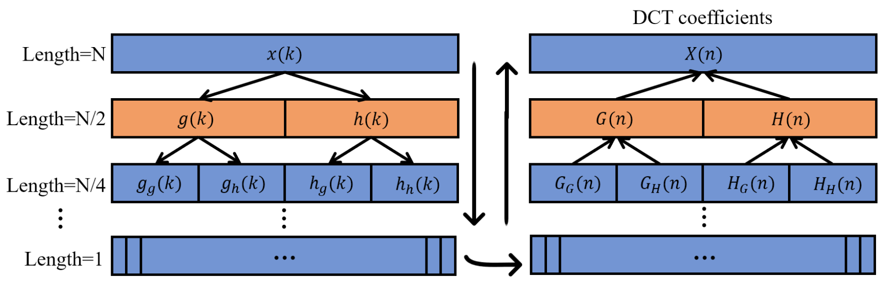


Figure 5.3: Process of the recursive DCT algorithm.

In the recursive DCT algorithm, the calculation of $h(k)$ (including $q(k)$) requires multiplication, which takes the most time in the algorithm [99]. The target of our scheme is to replace the multiplication operations by bit operations and additions/subtractions to reduce the compression overhead.

Piecewise-constant approximation can be applied to $q(k)$ defined in Equations (5.8) to achieve null multiplicative complexity in the computation of $q(k)$. We first analyze a simple format of $q(k)$ as $cos(\alpha\pi)$ with range of α from 0 to 0.5, as shown by the blue line in Fig. 5.4. In this example, the outputs of the function can be only 1, 1/2, and 0 after piecewise-constant approximation, as shown by the red line in Fig. 5.4. The boundaries of the interval are $th1$ and $th2$, and can be computed by the middle-point values 3/4 and 1/4. Because this is a monotonic function for this range of α, we can get the approximate output based on the value of α.

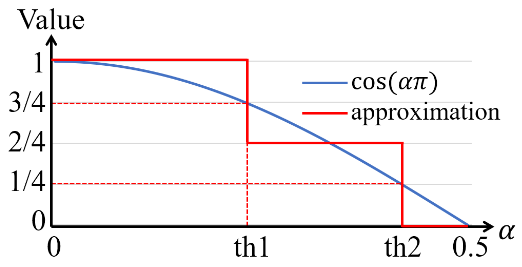


Figure 5.4: Applying piecewise-constant approximation to a cosine function.

Similarly, we can easily know the output of $q(k)$ after piecewise-constant approximation by checking only the value of k. Let $\alpha = (2k+1)/(2N)$, and we know that k ranges from 0 to $N/2-1$. The threshold values of α, $th1$, and $th2$, are easy to compute, and those of k can be computed based on the relation $k = \alpha \times N - 0.5$.

Furthermore, we can eliminate the multiplication during the process by using α to compute the threshold of k based on the relation $k = \alpha \times N - 0.5$. The threshold is approximated by dyadic values because they can be implemented by bit operations and additions/subtractions. For example, we can compute $th1$ as 0.23 in Fig. 5.4, and it can be approximately represented as $1/4 - 1/32 + 1/128$, where divisions by a power of two numbers can be also performed by bit operations.

We use more pieces for the output value of the cosine function to generate more accurate results. The possible output values are 0, 1/16, 2/16, ... , 15/16, and 1. Their thresholds of α, the corresponding approximate versions, and their absolute differences are listed as α Thr., Approx., and Diff columns, respectively, in Table 5.2.

Table 5.2: Configuration of the approximate threshold format.

α Thr.	Approx.	Diff	α Thr.	Approx.	Diff
0.0798	$\frac{1}{16}+\frac{1}{64}$	0.0017	0.3447	$\frac{1}{2}-\frac{1}{8}-\frac{1}{32}$	0.0010
0.1389	$\frac{1}{8}+\frac{1}{64}$	0.0017	0.3668	$\frac{1}{2}-\frac{1}{8}-\frac{1}{128}$	0.0004
0.1803	$\frac{1}{4}-\frac{1}{16}-\frac{1}{128}$	0.0007	0.3883	$\frac{1}{2}-\frac{1}{8}+\frac{1}{64}$	0.0023
0.2146	$\frac{1}{4}-\frac{1}{32}-\frac{1}{128}$	0.0036	0.4093	$\frac{1}{2}-\frac{1}{8}+\frac{1}{32}$	0.0030
0.2447	$\frac{1}{4}-\frac{1}{128}$	0.0025	0.4298	$\frac{1}{2}-\frac{1}{16}-\frac{1}{128}$	0.0001
0.2721	$\frac{1}{4}+\frac{1}{32}-\frac{1}{128}$	0.0013	0.4501	$\frac{1}{2}-\frac{1}{16}+\frac{1}{64}$	0.0031
0.2976	$\frac{1}{4}+\frac{1}{16}-\frac{1}{64}$	0.0008	0.4701	$\frac{1}{2}-\frac{1}{32}$	0.0014
0.3217	$\frac{1}{4}+\frac{1}{16}+\frac{1}{128}$	0.0014	0.4901	$\frac{1}{2}-\frac{1}{128}$	0.0021

Given the piecewise-constant values of $q(k)$, $h(k)$ can also achieve null multiplicative complexity because one of $h(k)$'s multipliers ($q(k)$) can be represented using bit operations. As mentioned above, the possible output values of $q(k)$ here are 0, $2 \times 1/16$, $2\times 2/16$, ... , $2\times 15/16$, and 2. They can all be represented by a set of dyadic values as 0, 1/8, 1/4, ... , $2-1/8$, and 2. Considering that multiplication by a power of two numbers can be performed by bit operations, the multiplication in $h(k)$ can be eliminated. The same approximate version of the thresholds listed in Table 5.2 can be applied to item $2cos(\pi n/2N)$ of Equation (5.3) in the subband decomposition process.

Table 5.3 shows the difference between the conventional DCT and our proposed DCT scheme based on the total time needed for the compression and decompression processes. The results are the average values of the time overhead in the FE application with various input sizes. The time increased for larger input sizes, which generated longer messages. Moreover, the proposed DCT reduced the time needed substantially compared with the conventional DCT.

Table 5.3: Reduction in the time needed for compression between the conventional DCT and the proposed scheme.

Size	200^3	300^3	400^3	450^3	500^3	550^3
Speedup	101.18	239.06	350.70	563.74	675.85	571.12

For all applications except for that of the BP, the proposed DCT scheme maintained the good energy compaction characteristics of the DCT. Fig. 5.5 shows the comparison of energy compaction for conventional exact DCT and the proposed DCT schemes with FE application and BP application. The x-axis and y-axis are the same as in Fig. 5.1, and represent the number of coefficients and the energy of the message, respectively. For FE application, proposed scheme was prevented from reaching 100% signal energy mainly because some high-frequency coefficients were discarded, as shown in Fig. 5.1(a). For these error-resilient applications, the transmission of the exact message or maintaining 100% signal energy is not necessary. Other applications except for BP (Fig. 5.1(b)) show the similar trends.

5.3.3 Zero run-length coding (RLC)

As a consequence of the message, only a few coefficients are maintained after DCT compression, and contain most of the energy of the message. The value of the remaining

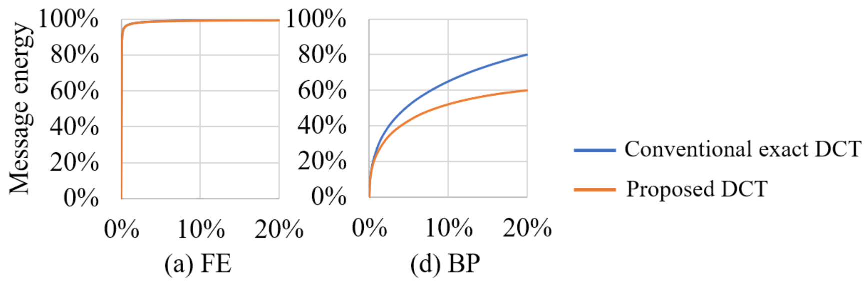


Figure 5.5: Comparison of message energy compaction between conventional exact DCT and the proposed DCT schemes.

coefficients is set to zero, and there are a large number of consecutive zero coefficients in a message. We exploit this by run-length coding the consecutive zeros to achieve high compression efficiency, as shown in Fig. 5.6. We encode each non-zero coefficient by pair first and the number of consecutive zeros preceding that coefficient, followed by the coefficient itself. Consecutive zeros with a maximum run length of 255 are represented using an eight-bit number. The non-zero coefficients and zero run-length values are then arranged in two parts in the output message, a data part and a run-length part, as shown in Fig. 5.6.

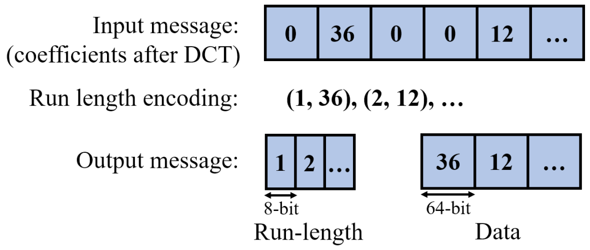


Figure 5.6: Example of the zero run-length scheme. The input message contains the coefficients after DCT compression and the output message contains data and the zero run-length parts.

5.4 Differential analysis of messages

For some applications, there are similarities among messages in the same node. For example, in two consecutive iterations of the FE application, the changes in messages to be transmitted to other nodes were not large. Therefore, most differences between

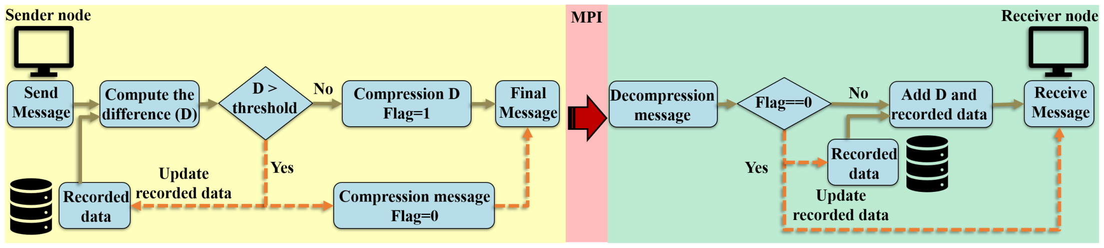


Figure 5.7: Overview of the proposed strategy between sender node and receiver node. The orange dashed lines and gray solid lines represent two methods of compression.

a given message and the previous message were small. Instead of transmitting the entire message, the difference can be applied to compression and transmitted over the network [102]. An overview of the proposed strategy is shown in Fig. 5.7 and the details are given below.

This diagram can be divided into three parts: a sender node part that generates the message and sends it, as shown in the yellow area; a receiver node part that demands the message from the sender node, as shown in the green area; and the MPI that is the interface used to transfer the message, represented by the red area in the middle.

For the sender side, the mean value range-based relative difference D is computed based on the given message x and recorded data rd, defined as

$$D = \frac{\frac{1}{N}\sum_{k=0}^{N} |x(k) - rd(k)|}{x_{max} - x_{min}}$$

where x_{max} and x_{min} are based on the given message x and N is its length. D is compared with the threshold and determines the content to be compressed. The definition of the recorded data rd is shown in the following paragraph.

For the transmission of the first message, there are no recorded data, and we can directly set D to be higher than the threshold. The full message is compressed and maintains high message energy to maintain high quality of the message. This message updates the recorded data to be used as future message reference as shown in Fig. 5.7 using the dashed orange lines. For the next message in the following iteration, D is computed. If D is lower than the threshold, which means that the recorded data can be a good baseline for the given message, only the difference $x - rd$ is compressed. In this process, the requirement of message energy in compression is relaxed, which allows for fewer coefficients to be transmitted over the network as shown in Fig. 5.7

using the solid gray lines. An additional flag is used to indicate the content of the compression. Therefore, based on differential analysis, the recorded data are updated with high quality only in case of full message compression.

At the receiver, the message is first decompressed. A flag then indicates whether the received message is a full message or only a difference message. If it is a full message, it is used to update the recorded data and the receiving process concludes. Otherwise, the reference in the recorded data needs to be added back to this difference data to get the final message.

Once an application starts the process of differential analysis, we can count the number of times the difference D is higher than the threshold. For the first several iterations, if D is always higher than the threshold, we can use this as a criterion to determine if an application has no similarities among messages. In later iterations, the original message is directly compressed without any differential analysis.

5.5 Experimental results

We evaluated our DCT-based approximate communication scheme on a distributed HP Linux cluster [103] with up to 360 nodes consisting of Intel Haswell E5-2680v3 processors and this system provided 711 Tflop/s of peak performance. Other lossy compression-based approximate communication schemes that have been used in HPC-related applications were also implemented for comparison. These state-of-the-art lossy compression algorithms were SZ [62], ZFP [63], and SSEM [65]. In the implementation of SSEM, quantization was applied to both low- and high-frequency bands instead of only to the latter to achieve better compression efficiency. All lossy compression algorithms were directly applied to OpenMPI implementation. The eight MPI-based error-resilient applications—FE, LULESH, KNN, BP, Blur, Edge, Sweep3D, and Halo3D—described in Section 5.1 were used to evaluate the impact of approximate communication on performance. This section compares the reduction in total execution time induced by our proposed scheme and the other schemes on the applications. The communication time, which included compression overhead, was further analyzed. Finally, approximate communication with differential analysis is evaluated for some applications.

5.5.1 Speedup of total execution time

The goal of approximate communication is to reduce the total time needed to execute an application. We evaluated the speedup of total execution time compared with that in the non-compression scheme. It is defined as $T_{noncomp}/T_{app}$, where $T_{noncomp}$ is the total execution time using the non-compression scheme, which does not feature compression and decompression processes. T_{app} is the total execution time using approximate communication.

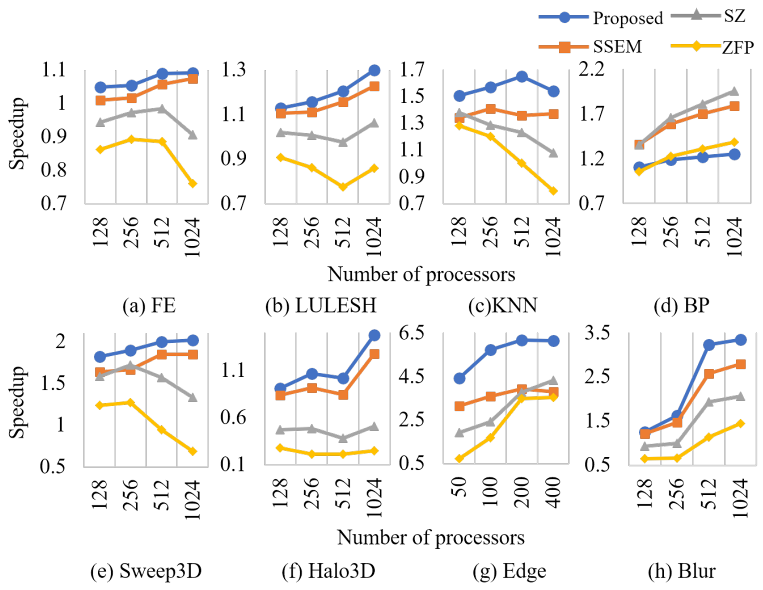


Figure 5.8: The reduction in total execution time for varying number of processors.

Fig. 5.8 shows the reduction in execution time for varying number of processors. The x-axis represents the different numbers of processors and the y-axis represents the reduction in total execution time compared with the original non-compression scheme. The various colored lines represent the results for different lossy compression schemes. All approximate communication schemes with different lossy algorithms maintained the same accuracy, the evaluation matrix for which is defined in Table 5.1. Specifically, the relative residual norm of FE was 10^{-5}; the mean relative difference of LULESH was less than 10%; the accuracy of KNN and BP was 90%; the mean value range-based relative difference between Sweep3D and Halo3D was no greater than 10%; and the SSIM of

Blur and Edge, defined in Equation (4.3), was maintained at 0.9.

Our DCT-based approximate communication scheme outperformed all other lossy compression schemes on most applications (except on the BP application), and achieved a speedup as high as 6.5x compared with the original non-compression scheme. SSEM was second best, and used the other commonly used transform-based method: wavelet transformation. Details of the evaluation of our scheme are described in Sections 5.5.2 and Sections 5.5.3.

5.5.2 Communication and compression overhead

The compression-based approximate communication algorithm is intended to reduce the overhead incurred by the time needed for communication. However, the compression produced an overhead as well. Therefore, the target of the approximate communication is then to reduce the overhead due to communication and compression. The proposed scheme strikes a good balance between the overhead in time incurred due to compression and a reduction in communication by substantially reducing the size of messages.

The execution times needed for communication and compression in all applications are illustrated in Fig. 5.9. Each bar in the figure for each application represents a lossy compression algorithm or non-compression scheme. It shows the communication and compression overhead (including decompression time) in orange and gray, respectively. All results were generated using 256 processors. The total execution time of an application can be divided into computation time, communication time, and compression overhead. For a given application, we maintained the same computation time for all compression algorithms and the non-compression scheme for better comparison. All schemes with different lossy algorithms maintained the same accuracy as described in Section 5.5.1.

As shown in the results in Fig. 5.9, the bottleneck of ZFP and SZ was the large overhead due to the time taken for compression. The communication time was positively correlated with the length of the message. The compression ratio of the SSEM-based approximate communication scheme was limited compared with other schemes. Therefore, the reduction in communication in the SSEM scheme was not as substantial as in the others. The low compression overhead and good compression ratio of the proposed DCT-based approximate communication scheme helped it record the shortest execution

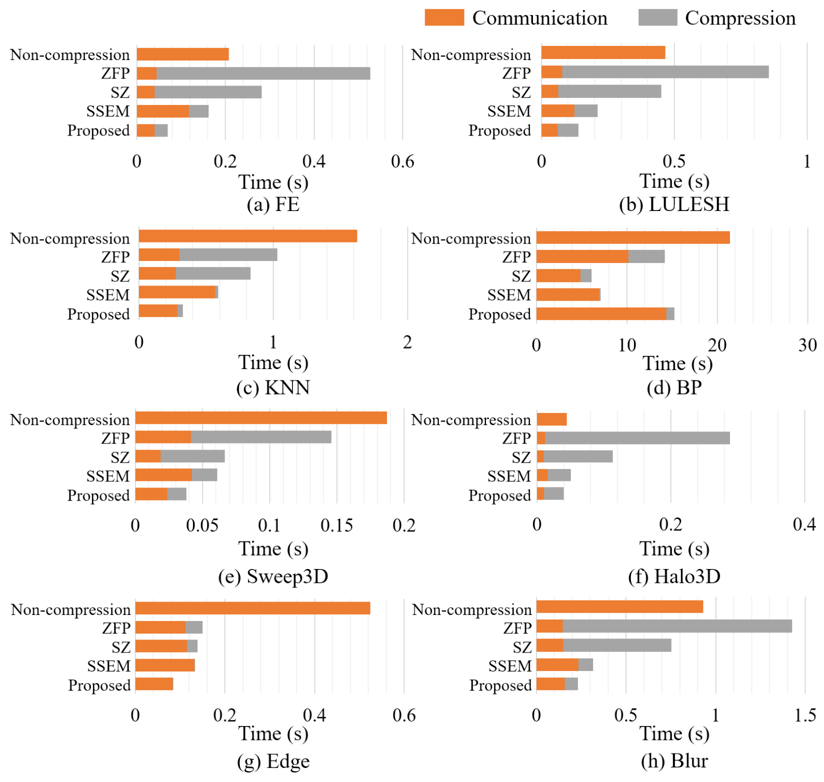


Figure 5.9: The execution times for communication and compression for various lossy compression algorithms on different applications.

time on most applications. On the BP application, the prediction-based compression algorithm SZ delivered better performance than the other three transform-based compression algorithms because of highly random messages in this application, as seen in Fig. 5.1(d). It was challenging to compress this message in the frequency domain with few coefficients.

5.5.3 Fraction of execution time used for communication

The large fraction of communication overhead in the total execution time limits the scalability of parallel applications. Therefore, reducing the fraction of communication is also an aspect we concerned for approximate communication.

The fraction of each part of the total execution time for all MPI applications is shown in Fig. 5.10. The percentage-stacked column charts are used to represent the fraction of total execution time. The x-axis represents four scenarios with different numbers of processors. For each scenario, the results of five schemes—our proposed scheme, non-compression, SSEM, SZ, and ZFP—are listed from left to right. As shown in Fig. 5.10, the fraction of communication increased in the non-compression scheme as it used a large number of processors for all applications. Considering the results for speedup given in Fig. 5.8, for applications with larger communication fractions, e.g., Edge, approximate communication achieved higher speedup with the same or even a smaller reduction in communication.

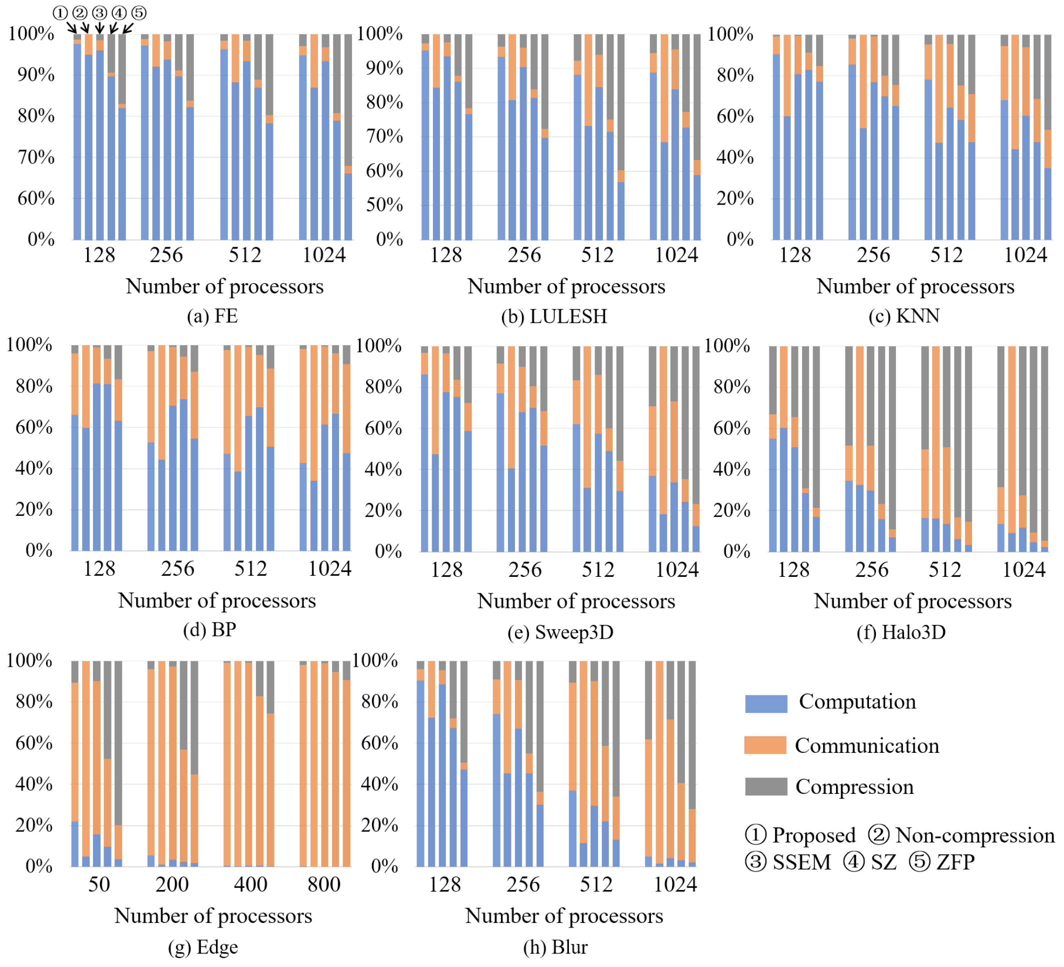


Figure 5.10: The fraction of total execution time spent on computing, communicating, and compressing as the number of processors was varied.

Our scheme can significantly reduce computation time by reducing more time needed for communication than the other lossy compression schemes. For Sweep3D with 256 processors, the percentage of total execution time devoted to communication decreased from 59% to 23%, where this time included the computational overhead required to compress the messages. Therefore, our DCT-based approximate communication scheme can significantly reduce communication and effectively improve the scalability.

5.5.4 Approximate communication with differential analysis

As described in Section 5.4, differential analysis of messages can be used in some applications. In this section, finite element and image blur applications were used to evaluate this strategy.

For FE application, Fig. 5.11 shows the total execution time for different values of error tolerance based on the relative residual norm. In FE, the result became more accurate with increasing number of iterations, but also took longer to execute. We continued running the application and recorded the execution times for different relative residual norms. The results were generated with 256 processors at an input size of 300^3. When a large error can be tolerated, e.g. 10^{-5}, the DCT-based approximate communication reduced total execution time compared with that in the non-compression scheme. However, the approximate communication lost its advantage once a strict error tolerance was applied, e.g. 10^{-11}. Moreover, because of the similarity among the messages, the benefit in terms of reducing time is greater when using differential analysis.

For Blur application, Fig. 5.12 shows the performance of the proposed scheme with and without differential analysis based on SSIM and execution time. The size of the original image was 11500×11500, and thus only part of the image is shown in Fig. 5.12. The execution times of the proposed scheme with and without differential analysis were maintained. The value of SSIM with differential analysis was higher and yielded results of better quality. Therefore, the differential analysis strategy can be used in certain applications.

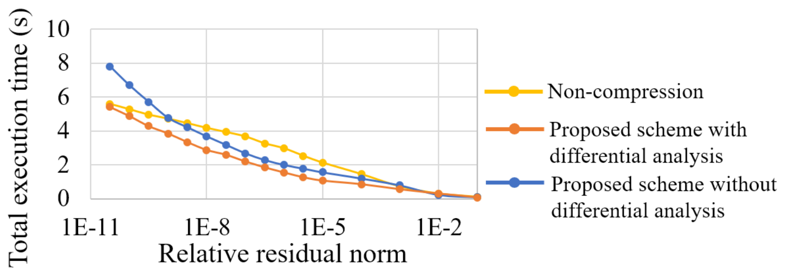


Figure 5.11: The total execution times for different requirements of the relative residual norm for various compression schemes with FE application.

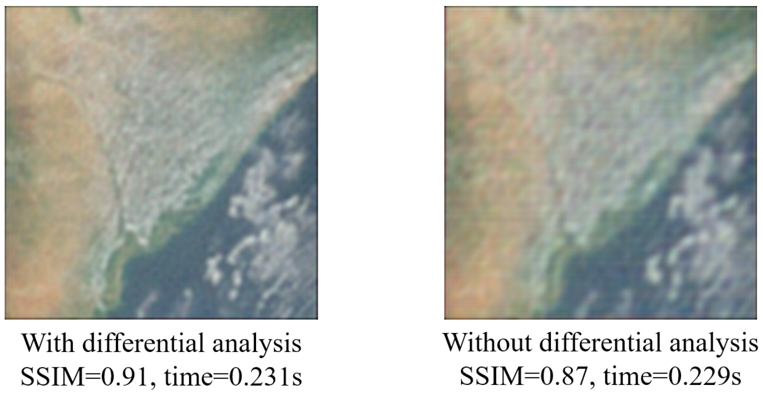


Figure 5.12: Visual output and execution time of DCT-based approximate communication scheme with and without differential analysis for Blur application.

5.6 Conclusion

In this chapter, we proposed a DCT-based approximate communication scheme to reduce communication overhead. The proposed compression scheme provides a better balance between compression speed and compression ratio compared than state-of-the-art lossy compression schemes for non-random message patterns, and can significantly reduce communication time without a considerable loss in the quality of the result, particularly for applications with large communication overhead.

Chapter 6

Conclusion

This thesis has developed novel techniques for the use of approximate memory, approximate storage, and approximate communication. By relaxing the need for a fully precise result, an approximate strategy shows great promise for implementing energy-efficient and error-tolerant systems. This thesis has reviewed several representative error-resilient applications for evaluation and all of the strategies we proposed trade off accuracy of results for gains in performance and efficiency for different components of the system.

For approximate memory repair mechanisms, we evaluated conventional redundant repair schemes and proposed new nonuniform protection-based repair schemes. In this thesis, our comprehensive models of the area and delay overheads of each approach are listed. The tradeoff relation between area overhead and quality degradation is explored under different error rates.

For approximate image data storage, adaptive-length coding of image data is proposed in this thesis to replace the error-sensitive Huffman coding in JPEG compression. ALC can overcome the error sensitivity of VLC with comparable compression efficiency. Therefore, it can be used as a methodology for storing a compressed image in approximate storage.

For approximate communication in a parallel cluster, we presented a new DCT-based approximate scheme by reducing the length of the messages in communication. To develop the runtime compression of MPI message among nodes, low time overhead of compression and decompression process are necessary. The proposed scheme in this thesis provides a good balance between the speed of the compression process and the compression ratio compared with existing lossy compression schemes used in HPC applications.

In summary, in this thesis, we mainly focus on the new approximation techniques that trade-off the accuracy of memory, storage, and communication for gains in efficiency and performance in large-scale, high-capacity computing systems. The partial-repaired memory scheme, error-resilient ALC scheme, and DCT-based approximate communication scheme are proposed in this thesis and enhance the performance of the system while delivering adequate accuracy.

References

[1] D. Jevdjic, K. Strauss, L. Ceze, and H. S. Malvar. Approximate storage of compressed and encrypted videos. In *Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 361–373, 2017.

[2] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, S. Williams, and K. Yelick. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office, Tech. Rep*, 15, 2008.

[3] CACTI tools. `http://www.hpl.hp.com/research/cacti/`.

[4] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *Proceedings of the IEEE European Test Symposium*, pages 1–6, 2013.

[5] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. In *Proc. MICRO*, pages 25–36, 2013.

[6] S. Ganapathy, G. Karakonstantis, A. Teman, and A. Burg. Mitigating the impact of faults in unreliable memories for error-resilient applications. In *Proc. DAC*, 2015.

[7] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar. High-density image storage using approximate memory cells. *ACM SIGPLAN Notices*, 51(4):413–426, 2016.

[8] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan. Approximate storage for energy efficient spintronic memories. In *Proceedings of the ACM Conference on Design Automation Conference*, page 195, 2015.

[9] H. Zhao, L. Xue, P. Chi, and J. Zhao. Approximate image storage with multi-level cell stt-mram main memory. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 268–275, 2017.

[10] B. Li, H. Najafi, and D. J. Lilja. Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier. In *Proceedings of the 2016 ACM International Symposium on Field-Programmable Gate Arrays*, pages 36–41, 2016.

[11] B. Li, H. Najafi, and D. J. Lilja. An FPGA implementation of a restricted boltzmann machine classifier using stochastic bit streams. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 68–69, 2015.

[12] B. Li, Y. Qin, B. Yuan, and D. J. Lilja. Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function. In *Proceedings of the IEEE International Conference on Computer Design*, pages 97–104, 2017.

[13] B. Li, M. H. Najafi, B. Yuan, and D. J. Lilja. Quantized neural networks with new stochastic multipliers. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 376–382, 2018.

[14] B. Li, Y. Qin, B. Yuan, and D. J. Lilja. Neural network classifiers using a hardware-based approximate activation function with a hybrid stochastic multiplier. *ACM Journal on Emerging Technologies in Computing Systems*, 15(1):12, 2019.

[15] B. Li, J. Hu, M. H. Najafi, S. Koester, and D. J. Lilja. Low cost hybrid spin-CMOS compressor for stochastic neural networks. In *Proceedings of the ACM on Great Lakes Symposium on VLSI*, pages 141–146, 2019.

[16] B. Li, M. H. Najafi, and D. J. Lilja. Low-cost stochastic hybrid multiplier for quantized neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, 15(2):18, 2019.

[17] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: a large-scale field study. *ACM Sigmetrics Performance Review*, 37(1):193–204, 2009.

[18] C. Slayman. Soft error trends and mitigation techniques in memory devices. In *Proc. RAMS*, pages 1–5, 2011.

[19] J.-F. Li, J.-C. Yeh, R.-F. Huang, and C.-W. Wu. A built-in self-repair design for RAMs with 2-D redundancy. *IEEE T. VLSI Syst*, 13(6):742–745, 2005.

[20] S.-K. Lu, C.-L. Yang, Y.-C. Hsiao, and C.-W. Wu. Efficient BISR techniques for embedded memories considering cluster faults. *IEEE T. VLSI Syst*, 18(2):184–193, 2010.

[21] Y. Xie. *Emerging memory technologies*. Springer, Boston, MA, 2014.

[22] I. Koren and Z. Koren. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, 1998.

[23] M. Horiguchi and K. Itoh. *Nanoscale memory repair*. Springer Science & Business Media, 2011.

[24] B. Li, M. Minglani, and D. Lilja. Ps-code: A new code for improved degraded mode read and write performance of RAID systems. In *Proceedings of the IEEE International Conference on Networking, Architecture and Storage*, pages 1–10, 2016.

[25] B. Li, M. Yang, S. Mohajer, W. Qian, and D. J. Lilja. Tier-code: An XOR-based RAID-6 code with improved write and degraded-mode read performance. In *Proceedings of the IEEE International Conference on Networking, Architecture and Storage*, pages 1–10, 2018.

[26] Mary meeker's annual internet trends report. `https://techcrunch.com/2019/06/11/internet-trends-report-2019/`, 2019.

[27] E. Yan, K. Zhang, X. Wang, K. Strauss, and L. Ceze. Customizing progressive jpeg for efficient image storage. In *USENIX Workshop on Hot Topics in Storage and File Systems*, 2017.

[28] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: Facebook's photo storage. In *USENIX Symposium on Operating Systems Design and Implementation*, volume 10, pages 1–8, 2010.

[29] Apple worldwide developers conference: High efficiency image file format. `https://developer.apple.com/videos/play/wwdc2017/513/`, 2017.

[30] D. R. Horn, K. Elkabany, C. Lesniewski-Laas, and K. Winstein. The design, implementation, and deployment of a system to transparently compress hundreds of petabytes of image files for a file-storage service. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 1–15, 2017.

[31] W^{3}techs reports. `https://w3techs.com/technologies/overview/image_format/all`, 2018.

[32] W. B. Pennebaker and J. L. Mitchell. *JPEG: Still image data compression standard*. Van Nostrand Reinhold, New York, NY, 1992.

[33] Google photos: High quality. `https://photos.google.com/settings`, 2018.

[34] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[35] C.-H. Huang, Y. Li, and L. Dolecek. Acoco: Adaptive coding for approximate computing on faulty memories. *IEEE Transactions on Communications*, 63(12):4615–4628, 2015.

[36] Jpegmini. `https://www.jpegmini.com/main/technology`, 2018.

[37] MPI: A message passing interface standard. `https://www.mpi-forum.org/`, 2018.

[38] F. Betzel, K. Khatamifard, H. Suresh, D. J. Lilja, J. Sartori, and U. Karpuzcu. Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems. *ACM Computing Surveys*, 51(1):1, 2018.

[39] S.-H. Jung, S. K. Mitra, and D. Mukherjee. Subband DCT: Definition, analysis, and applications. *IEEE Transactions on Circuits and systems for Video Technology*, 6(3):273–286, 1996.

[40] S. Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4):62, 2016.

[41] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: saving dram refresh-power through critical data partitioning. *ACM SIGPLAN Notices*, 47(4):213–224, 2012.

[42] D. W. Redmill and N. G. Kingsbury. The EREC: An error-resilient technique for coding variable-length blocks of data. *IEEE Transactions on Image Processing*, 5(4):565–574, 1996.

[43] Y. Yoo and A. Ortega. Constrained bit allocation for error resilient JPEG coding. In *Proceedings of the IEEE Signals, Systems & Computers*, volume 2, pages 985–989, 1997.

[44] Y. Fang. Erec-based length coding of variable-length data blocks. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(10):1358–1366, 2010.

[45] Y.-S. Lee, K.-K. Ong, and C.-Y. Lee. Error-resilient image coding (eric) with smart-idct error concealment technique for wireless multimedia transmission. *IEEE transactions on circuits and systems for video technology*, 13(2):176–181, 2003.

[46] L.-W. Kang and J.-J. Leou. An error resilient coding scheme for jpeg image transmission based on data embedding and side-match vector quantization. *Journal of Visual Communication and Image Representation*, 17(4):876–891, 2006.

[47] Y.-S. Lee, C.-M. Yu, W.-S. Chang, et al. Hvlc: Error correctable hybrid variable length code for image coding in wireless transmission. In *Proceedings of the IEEE Acoustics, Speech, and Signal Processing*, volume 4, pages 2103–2106, 2000.

[48] H. Cai, B. Zeng, G. Shen, Z. Xiong, and S. Li. Error-resilient unequal error protection of fine granularity scalable video bitstreams. *EURASIP Journal on Advances in Signal Processing*, 2006(1):045412, 2006.

[49] Z. Xue, K. K. Loo, J. Cosmas, M. Tun, L. Feng, and P.-Y. Yip. Error-resilient scheme for wavelet video codec using automatic roi detection and wyner-ziv coding over packet erasure channel. *IEEE Transactions on Broadcasting*, 2010.

[50] S. Jakubczak and D. Katabi. Softcast: One-size-fits-all wireless video. *ACM SIGCOMM Computer Communication Review*, 41(4):449–450, 2011.

[51] S. Roman. *Introduction to Coding and Information Theory*. Springer-Verlag, New York, NY, 1997.

[52] H. Hashempour, L. Schiano, and F. Lombardi. Error-resilient test data compression using tunstall codes. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 316–323, 2004.

[53] J. Meng, A. Raghunathan, S. Chakradhar, and S. Byna. Exploiting the forgiving nature of applications for scalable parallel execution. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, 2010.

[54] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the ACM International Symposium on Microarchitecture*, pages 13–24, 2013.

[55] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the ACM SIGPLAN Notices*, volume 49, pages 35–50, 2014.

[56] B. Dickov, M. Pericas, G. Houzeaux, N. Navarro, and E. Ayguadé. Assessing the impact of network compression on molecular dynamics and finite element methods. In *Proceedings of the IEEE International Conference on High Performance Computing and Communication*, pages 588–597, 2012.

[57] J. Ke, M. Burtscher, and E. Speight. Runtime compression of MPI messanes to improve the performance and scalability of parallel applications. In *Proceedings of the IEEE Conference on Supercomputing*, page 59, 2004.

[58] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt. Integrating online compression to accelerate large-scale data analytics applications. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1205–1216, 2013.

[59] C. Krintz and S. Sucu. Adaptive on-the-fly compression. *IEEE Transactions on Parallel and Distributed Systems*, 17(1):15–24, 2006.

[60] R. Filgueira, J. Carretero, D. E. Singh, A. Calderon, and A. Núñez. Dynamic-CoMPI: Dynamic optimization techniques for MPI parallel applications. *The Journal of Supercomputing*, 59(1):361–391, 2012.

[61] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C.-S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. ISABELA for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

[62] D. Tao, S. Di, Z. Chen, and F. Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pages 1129–1139, 2017.

[63] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[64] L. Fischer, S. Götschel, and M. Weiser. Lossy data compression reduces communication time in hybrid time-parallel integrators. *Computing and Visualization in Science*, 19(1-2):19–30, 2018.

[65] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka. Exploration of lossy compression for application-level checkpoint/restart. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pages 914–922, 2015.

[66] Q. Fan, S. S. Sapatnekar, and D. J. Lilja. Cost-quality trade-offs of approximate memory repair mechanisms for image data. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pages 438–444, 2017.

[67] T.-H. Wu, P.-Y. Chen, M. Lee, B.-Y. Lin, C.-W. Wu, C.-H. Tien, H.-C. Lin, H. Chen, C.-N. Peng, and M.-J. Wang. A memory yield improvement scheme combining built-in self-repair and error correction codes. In *Proc. ITC*, pages 1–9, 2012.

[68] A. Van De Goor. *Testing semiconductor memories: theory and practice.* John Wiley & Sons, Inc., New York, NY, 1991.

[69] C.-T. Huang, J.-R. Huang, C.-F. Wu, C.-W. Wu, and T.-Y. Chang. A programmable BIST core for embedded DRAM. *IEEE Des. Test*, 16(1):59–70, 1999.

[70] S.-K. Lu, Y.-C. Tsai, C.-H. Hsu, K.-H. Wang, and C.-W. Wu. Efficient built-in redundancy analysis for embedded memories with 2-d redundancy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(1):34–42, 2006.

[71] DDR3-SDRAM data sheet. `https://www.micron.com/products/dram/ddr3-sdram/`.

[72] STT-MRAM data sheet. `https://www.everspin.com/`.

[73] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE J. Solid-St. Circ.*, 41(3):712–727, 2006.

[74] NVsim tools. `www.nvsim.org/`.

[75] Q. Fan, D. J. Lilja, and S. S. Sapatnekar. Adaptive-length coding of image data for low-cost approximate storage. *IEEE Transactions on Computers (under review for publication)*, 2019.

[76] G. Dong, Y. Pan, N. Xie, C. Varanasi, and T. Zhang. Estimating information-theoretical nand flash memory storage capacity and its implication to memory system design space exploration. *IEEE Transactions on VLSI Systems*, 20(9):1705–1714, 2012.

[77] K. Cabeen and P. Gent. Image compression and the discrete cosine transform, Math 45 Project, College of the Redwoods, Eureka, CA. `https://mse.redwoods.edu/darnold/math45/laproj/Fall98/PKen/dct.pdf`, 1998.

[78] M. Ghanbari. *Standard Codecs: Image Compression to Advanced Video Coding.* The Institution of Engineering and Technology, London,UK, 3rd edition, 2011.

[79] Imagenet large scale visual recognition challenge. `http://www.image-net.org/challenges/LSVRC/`, 2016.

[80] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[81] R. Micheloni, A. Marelli, and K. Eshghi. *Inside Solid State Drives (SSDs).* Springer Dordrecht, New York, NY, 2012.

[82] H. Choi, W. Liu, and W. Sung. Vlsi implementation of bch error correction for multilevel cell nand flash memory. *IEEE Transactions on VLSI Systems*, 18(5):843–847, 2010.

[83] R. Micheloni, A. Marelli, and R. Ravasio. *Error correction codes for non-volatile memories.* Springer Dordrecht, New York, NY, 2012.

[84] S. S. Channappayya, A. C. Bovik, and R. W. Heath Jr. Rate bounds on ssim index of quantized images. *IEEE Transactions on Image Processing*, 17(9):1624–1639, 2008.

[85] S. Wang, A. Rehman, Z. Wang, S. Ma, and W. Gao. Ssim-motivated rate-distortion optimization for video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(4):516–529, 2012.

[86] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *Proceedings of the IEEE International Conference on Computer Design*, pages 94–101, 2012.

[87] Q. Fan, D. J. Lilja, and S. S. Sapatnekar. Using DCT-based approximate communication to improve MPI performance in parallel clusters. In *Proceedings of the IEEE International Performance Computing and Communications Conference*, 2019.

[88] The Mantevo project. `https://mantevo.org/`.

[89] I. Karlin. LULESH programming model and performance ports overview. Technical report, Lawrence Livermore National Lab, Livermore, CA, United States, 2012.

[90] Skin segmentation data set, UCI machine learning repository. `https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation#`.

[91] CMU face images data set. `http://archive.ics.uci.edu/ml/datasets/cmu+face+images`.

[92] A. Bhatele. Evaluating trade-offs in potential exascale interconnect topologies. Technical report, Lawrence Livermore National Lab., Livermore, CA, United States, 2018.

[93] NASA earth observatory. `https://earthobservatory.nasa.gov/`.

[94] C.-K. Fong and W.-K. Cham. LLM integer cosine transform and its fast algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(6):844–854, 2012.

[95] Z. Chen, Q. Han, and W.-K. Cham. Low-complexity order-64 integer cosine transform design and its application in HEVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(9):2407–2412, 2018.

[96] T. I. Haweel. A new square wave transform based on the DCT. *Signal processing*, 81(11):2309–2319, 2001.

[97] R. J. Cintra and F. M. Bayer. A DCT approximation for image compression. *IEEE Signal Processing Letters*, 18(10):579–582, 2011.

[98] R. J. Cintra, F. M. Bayer, and C. Tablada. Low-complexity 8-point DCT approximations based on integer functions. *Signal Processing*, 99:201–214, 2014.

[99] C.-W. Kok. Fast algorithm for computing discrete cosine transform. *IEEE Transactions on Signal Processing*, 45(3):757–760, 1997.

[100] B. Lee. A new algorithm to compute the discrete cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(6):1243–1245, 1984.

[101] C. Loeffler, A. Ligtenberg, and G. S. Moschytz. Practical fast 1-D DCT algorithms with 11 multiplications. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 988–991, 1989.

[102] Z. Chen, S. W. Son, W. Hendrix, A. Agrawal, W.-k. Liao, and A. Choudhary. NUMARCK: machine learning algorithm for resiliency and checkpointing. In *Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 733–744, 2014.

[103] Mesabi compute cluster, Minnesota Supercomputing Institute. `https://www.msi.umn.edu/content/mesabi`.