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

Qianqian Fan, David J. Lilja, and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN 55455
e-mail: {fanxx297, lilja, sachin}@umn.edu

***Abstract*—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 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 seen that the fraction of total execution time devoted to communication reduces from 59% to 23%, even accounting for the computational overhead required for DCT encoding. For many 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.**

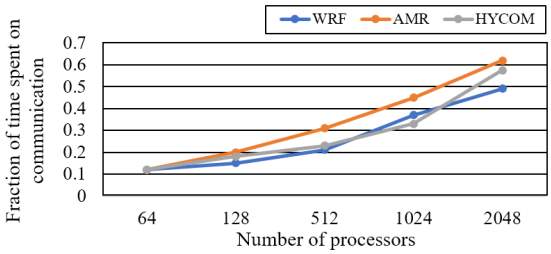


Fig. 1: Fraction of time spent on communication as the number of processors increase for representative high-performance computing applications [3].

I. Introduction

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 [1].

In large-scale parallel systems, efficient communication is a major challenge for scalability [2]. Communication-intensive parallel applications transfer a large amount of data among nodes of a cluster via an interconnection network. In [3], the fraction of time spent on communication increased significantly with the number of processors for representative applications of HPC, as shown in Fig. 1. This large communication overhead limits the scalability of parallel applications.

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 [2] 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.

This paper proposes 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 [4] 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: 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. The percentage of total execution time devoted to

This work was supported in part by U.S. National Science Foundation grant no. CCF-1438286. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

communication decreased from 59% to 23%, even accounting for the computational overhead required for DCT encoding. 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 results.

II. Related Work

To improve the efficiency of communication in parallel applications, several works have proposed some techniques [2] to reduce the total number of messages and the size of each message.

A. 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 [12]. Thread fusion was used in [13]. 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 [14] 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.

B. 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 [15] [16].

1) Lossless compression in parallel applications: By taking advantage of the similarities between spatial and temporal neighbors, Ref. [17] evaluated a method of lossless compression on a large-scale climate simulation dataset. Because different compression algorithms deliver varying performance, the adaptive compression system [18] [19] 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.

2) Lossy compression algorithm in parallel applications: 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 [20] 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 [21] 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 [22] develops an orthogonal block transform-based compression algorithm to compress 3D floating-point data. Wavelet transformation has also been applied to HPC applications [23] [24]. SSEM [24] 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.

III. 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

TABLE I: Summary of applications.

Application	Description	Input	Evaluation Metric
FE [5]	Finite element method	300^3 for length of 3D	Relative residual norm
LULESH [6]	Unstructured Lagrange explicit shock hydrodynamics	30^3 for length of mesh	Mean relative difference
KNN [7]	K-nearest neighbors classification	Skin segmentation dataset	Similarity of the predicted labels
BP [8]	Backpropagation neural network learning	CMU face image dataset	Accuracy of test cases
Sweep3D [9]	Models a wavefront propagating communication pattern	100^3 for length of mesh	Mean value range-based relative difference
Halo3D [9]	Models nearest neighbor communication pattern	100^3 for length of mesh	Mean value range based relative difference
Edge [10]	Edge detection in image	1200×800 images	Structural similarity index (SSIM)
Blur [11]	Image blur filter	11500×11500 images	Structural similarity index (SSIM)

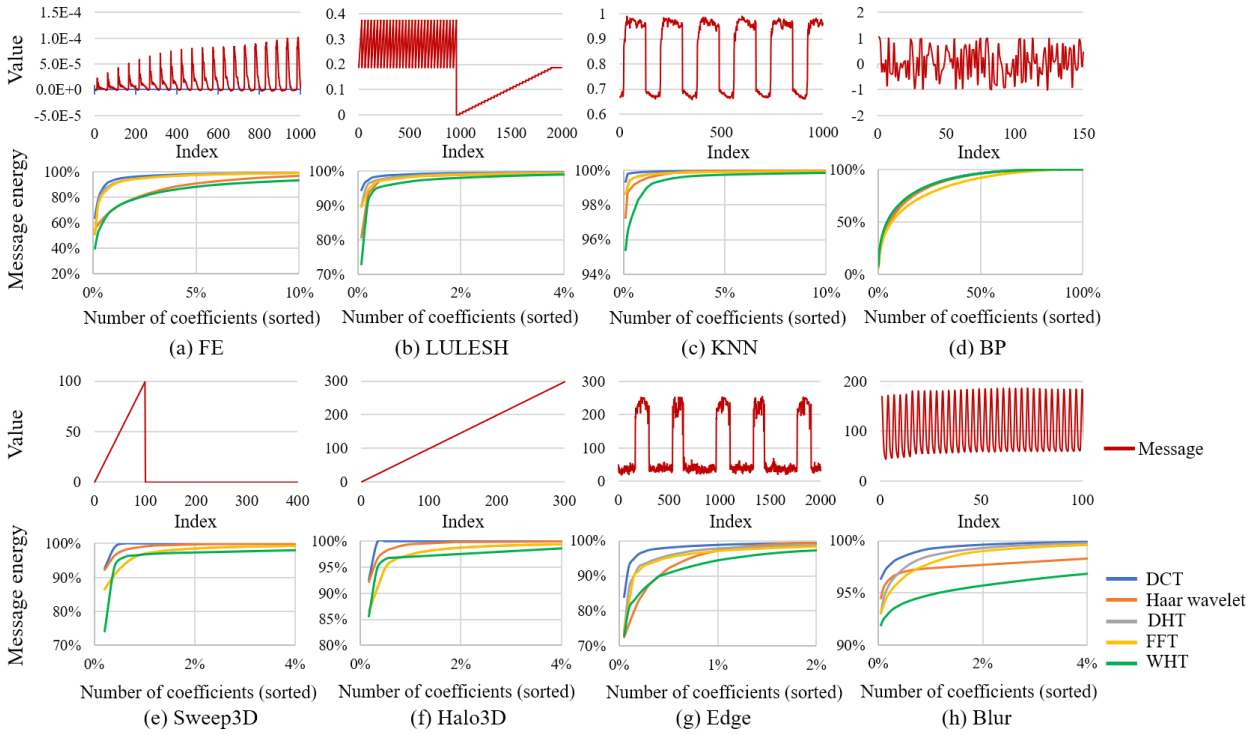


Fig. 2: Representative messages extracted from various applications (upper one in each subfigure) and energy compaction for various transform-based compression methods with different applications (lower one in each subfigure).

resilience to output error. Table I summarizes these applications. The definitions of the evaluation metrics in Table I are shown below:

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 [6]. 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 [25] was used to measure the quality of the results. It is an index that measures the structural similarity between images, and a well-known objective image quality metric defined as

$$SSIM\,(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (1)$$

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, and c_1 and c_2 are variables to stabilize the division with a weak denominator. SSIM is valued between -1 and 1. When two images are nearly identical, their SSIM is close to 1.

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 data, which showed different levels of randomness. The first row of each subfigure in Fig. 2 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. 2 (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. 2 (d). The non-random characteristic of the message pattern can be exploited, and more details are discussed in the next section.

IV. 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

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

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 III, 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. 2, we apply the above compression algorithms to representative messages extracted from the MPI-based applications listed in Table I. 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 (2). 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. 2(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. 2 (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;
θ_n: number of coefficient thresholds; θ_e:energy threshold;
Output: flag_s: Decision on selection of our scheme;
compute DCT coefficients C_M of M;
initial C_L = Φ; flag_s = 0; i = 1;
while i/n ≤ θ_n do
    push the ith largest coefficient of C_M to C_L;
    compute the message energy E_i using C_L;
    if (E_i ≥ θ_e) then
        flag_s = 1; break;
    end
    i = i + 1;
end
return flag_s;
```

V. 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

$$X(n) = \sum_{k=0}^{N-1} x(k) \cos\left(\pi (2k+1) \frac{n}{2N}\right) \tag{3}$$

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 V-A), fast recursive DCT with piecewise-constant approximation (Section V-B), and zero run-length coding (Section V-C). 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.

A. Subband decomposition

The subband decomposition of a message [4] can be applied to the message compression to reduce the time overhead in the first step. Fig. 3 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. 3, the lower-frequency band concentrates coefficients with larger absolute values. Based on the definition in Equation (2), 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 V-B, Fig. 6 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.

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

$$x_L(k) = \frac{1}{2}\left(x(2k) + x(2k+1)\right)$$

$$x_H(k) = \frac{1}{2}\left(x(2k) - x(2k+1)\right)$$

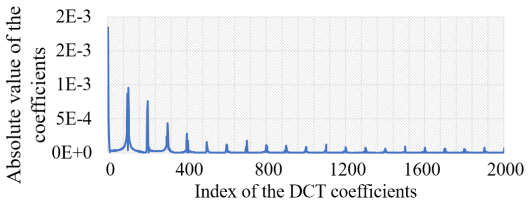


Fig. 3: The absolute value of DCT coefficients generated by the conventional DCT scheme.

where k ranges from 0 to $N/2-1$. Based on the derivation in [4], 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

$$\begin{aligned} X(n) &\approx 2cos\left(\frac{\pi n}{2N}\right)\sum_{k=0}^{\frac{N}{2}-1} x_L(k)cos\left(\pi(2n+1)\frac{n}{N}\right) \\ &= 2cos\left(\frac{\pi n}{2N}\right)X_L(n) \end{aligned} \tag{4}$$

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 V-B is then used to compute $X_L(n)$. Finally, $X(n)$ can be obtained based on Equation (4). 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 V-B.

B. Fast recursive DCT with piecewise-constant approximation

To compute $X_L(n)$ efficiently as described in Section V-A, 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 [26] [27], signed DCT [28], and rounded DCT [29] [30]. 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 [19]. 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 [31] 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 [32] [33]. In the recursive DCT algorithm [31], the DCT coefficients can be divided into even and odd parts, for n from 0 to $N/2-1$,

$$X(2n) = G(n) \tag{5}$$

$$X(2n+1) = H(n) - X(2n-1),\ (X(1) = H(0)/2) \tag{6}$$

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

$$g(k) = x(k) + x(N-1-k) \tag{7}$$

$$h(k) = q(k) \times (x(k) - x(N-1-k)) \tag{8}$$

where $q(k)$ is defined as

$$q(k) = 2cos\left(\frac{(2k+1)\pi}{2N}\right) \tag{9}$$

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

Fig. 4 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. 4, 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. 4. Based on the definition of the DCT in Equation (3), 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) (6), 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.

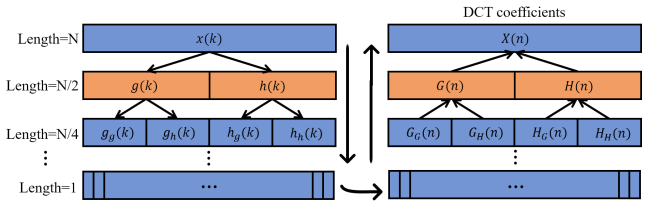


Fig. 4: Process of the recursive DCT algorithm.

In the recursive DCT algorithm, the calculation of $h(k)$ (including $q(k)$) requires multiplication, which takes the most

time in the algorithm [31]. 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 (9) 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. 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. 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.

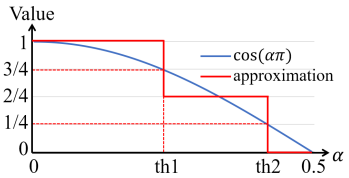


Fig. 5: Applying piecewise-constant approximation to a cosine function.

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, 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 II.

TABLE II: 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 II can be applied to item $2cos(\pi n/2N)$ of Equation (4) in the subband decomposition process.

Table III 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 III: 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. 6 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. 2, 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. 2 (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. 2 (b)) show the similar trends.

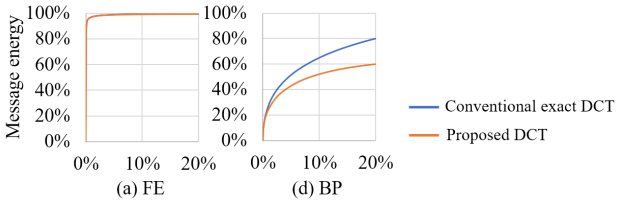


Fig. 6: Comparison of message energy compaction between conventional exact DCT and the proposed DCT schemes.

C. 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 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. 7. 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. 7.

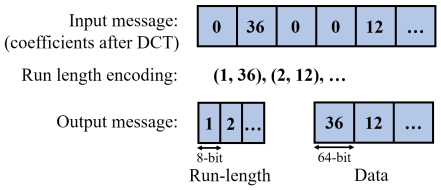


Fig. 7: Example of the zero run-length scheme. The input message contains the coefficients after DCT compression and the output message contains data and the zero run-length parts.

VI. 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 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 [34]. An overview of the proposed strategy is shown in Fig. 8 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. 8 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. 8 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.

VII. Experimental Results

We evaluated our DCT-based approximate communication scheme on a distributed HP Linux cluster [35] 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 [21], ZFP [22], and SSEM [24]. 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

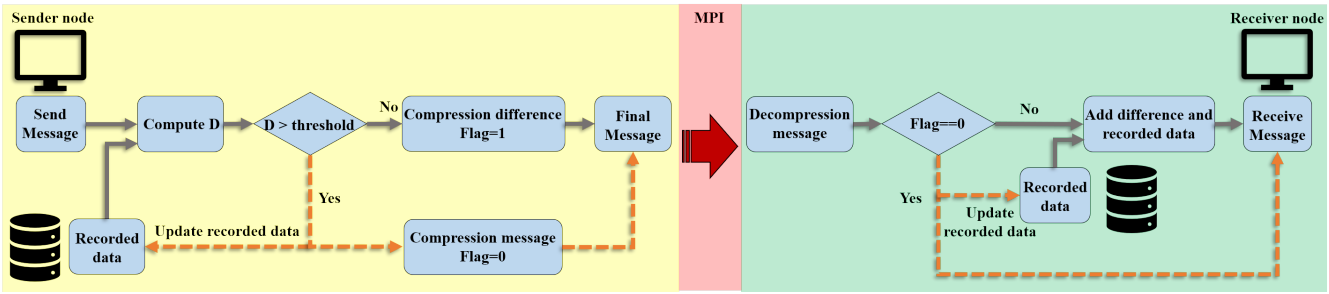


Fig. 8: Overview of the proposed strategy between sender node and receiver node. The orange dashed lines and gray solid lines represent two methods of compression.

OpenMPI implementation. The eight MPI-based error-resilient applications—FE, LULESH, KNN, BP, Blur, Edge, Sweep3D, and Halo3D—described in Section III 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.

A. 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.

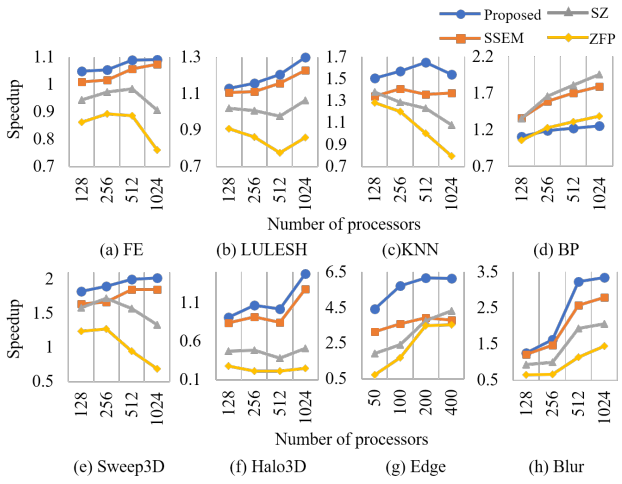


Fig. 9: The speedup of total execution time for varying number of processors.

Fig. 9 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 I. 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 (1), 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 VII-B and VII-C.

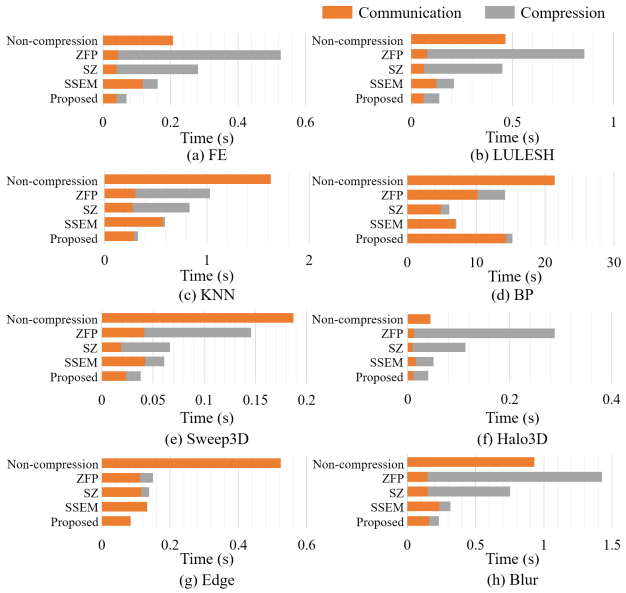


Fig. 10: The execution times for communication and compression for various lossy compression algorithms on different applications.

B. 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. 10. 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 VII-A.

As shown in the results in Fig. 10, 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

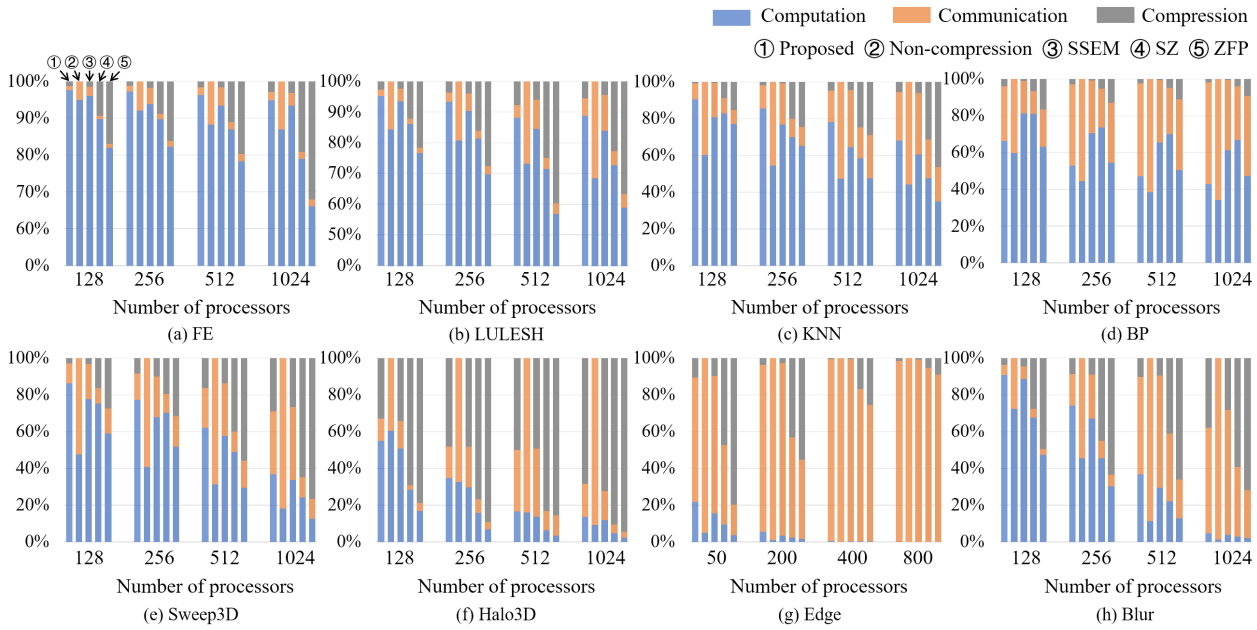


Fig. 11: The fraction of total execution time spent on computing, communicating, and compressing as the number of processors was varied.

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 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. 2 (d). It was challenging to compress this message in the frequency domain with few coefficients.

C. Fraction of 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. 11. 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. 11, 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. 9, for applications with larger communication fractions, e.g., Edge, approximate communication achieved higher speedup with the same or even a smaller reduction in communication. 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.

D. Approximate communication with differential analysis

As described in Section VI, 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. 12 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. 13 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. 13. 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.

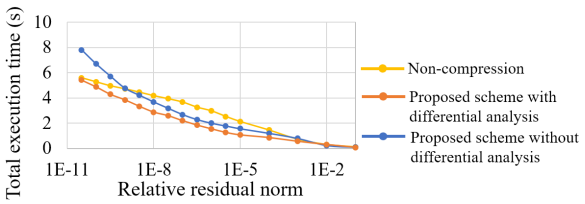
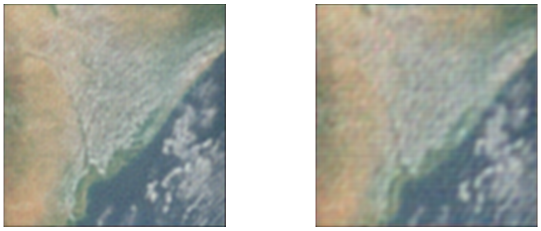


Fig. 12: The total execution times for different requirements of the relative residual norm for various compression schemes with FE application.



With differential analysis
SSIM=0.91, time=0.231s

Without differential analysis
SSIM=0.87, time=0.229s

Fig. 13: Visual output and execution time of DCT-based approximate communication scheme with and without differential analysis for Blur application.

VIII. Conclusion

In this paper, 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.

References

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

[2] F. Betzel, *et al.*, "Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems," *ACM Computing Surveys*, vol. 51, no. 1, p. 1, 2018.

[3] K. Bergman, *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office, Tech. Rep*, vol. 15, 2008.

[4] S.-H. Jung, *et al.*, "Subband DCT: Definition, analysis, and applications," *IEEE Transactions on Circuits and systems for Video Technology*, vol. 6, no. 3, pp. 273–286, 1996.

[5] "The Mantevo project." https://mantevo.org/.

[6] I. Karlin, "LULESH programming model and performance ports overview," tech. rep., Lawrence Livermore National Lab, Livermore, CA, United States, 2012.

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

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

[9] A. Bhatele, "Evaluating trade-offs in potential exascale interconnect topologies," tech. rep., Lawrence Livermore National Lab., Livermore, CA, United States, 2018.

[10] "ImageNet large scale visual recognition challenge." http://www.image-net.org/challenges/LSVRC/.

[11] "NASA earth observatory." https://earthobservatory.nasa.gov/.

[12] J. Meng, *et al.*, "Exploiting the forgiving nature of applications for scalable parallel execution," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–12, 2010.

[13] M. Samadi, *et al.*, "Sage: Self-tuning approximation for graphics engines," in *Proceedings of the ACM International Symposium on Microarchitecture*, pp. 13–24, 2013.

[14] M. Samadi, *et al.*, "Paraprox: Pattern-based approximation for data parallel applications," in *Proceedings of the ACM SIGPLAN Notices*, vol. 49, pp. 35–50, 2014.

[15] B. Dickov, *et al.*, "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*, pp. 588–597, 2012.

[16] J. Ke, *et al.*, "Runtime compression of MPI messanes to improve the performance and scalability of parallel applications," in *Proceedings of the IEEE Conference on Supercomputing*, p. 59, 2004.

[17] T. Bicer, *et al.*, "Integrating online compression to accelerate large-scale data analytics applications," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pp. 1205–1216, 2013.

[18] C. Krintz and S. Sucu, "Adaptive on-the-fly compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 1, pp. 15–24, 2006.

[19] R. Filgueira, *et al.*, "Dynamic-CoMPI: Dynamic optimization techniques for MPI parallel applications," *The Journal of Supercomputing*, vol. 59, no. 1, pp. 361–391, 2012.

[20] S. Lakshminarasimhan, *et al.*, "ISABELA for effective in situ compression of scientific data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 524–540, 2013.

[21] D. Tao, *et al.*, "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*, pp. 1129–1139, 2017.

[22] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[23] L. Fischer, *et al.*, "Lossy data compression reduces communication time in hybrid time-parallel integrators," *Computing and Visualization in Science*, vol. 19, no. 1-2, pp. 19–30, 2018.

[24] N. Sasaki, *et al.*, "Exploration of lossy compression for application-level checkpoint/restart," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pp. 914–922, 2015.

[25] Z. Wang, *et al.*, "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[26] C.-K. Fong and W.-K. Cham, "LLM integer cosine transform and its fast algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 6, pp. 844–854, 2012.

[27] Z. Chen, *et al.*, "Low-complexity order-64 integer cosine transform design and its application in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 9, pp. 2407–2412, 2018.

[28] T. I. Haweel, "A new square wave transform based on the DCT," *Signal processing*, vol. 81, no. 11, pp. 2309–2319, 2001.

[29] R. J. Cintra and F. M. Bayer, "A DCT approximation for image compression," *IEEE Signal Processing Letters*, vol. 18, no. 10, pp. 579–582, 2011.

[30] R. J. Cintra, *et al.*, "Low-complexity 8-point DCT approximations based on integer functions," *Signal Processing*, vol. 99, pp. 201–214, 2014.

[31] C.-W. Kok, "Fast algorithm for computing discrete cosine transform," *IEEE Transactions on Signal Processing*, vol. 45, no. 3, pp. 757–760, 1997.

[32] B. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1243–1245, 1984.

[33] C. Loeffler, *et al.*, "Practical fast 1-D DCT algorithms with 11 multiplications," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,*, pp. 988–991, 1989.

[34] Z. Chen, *et al.*, "NUMARCK: machine learning algorithm for resiliency and checkpointing," in *Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 733–744, 2014.

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