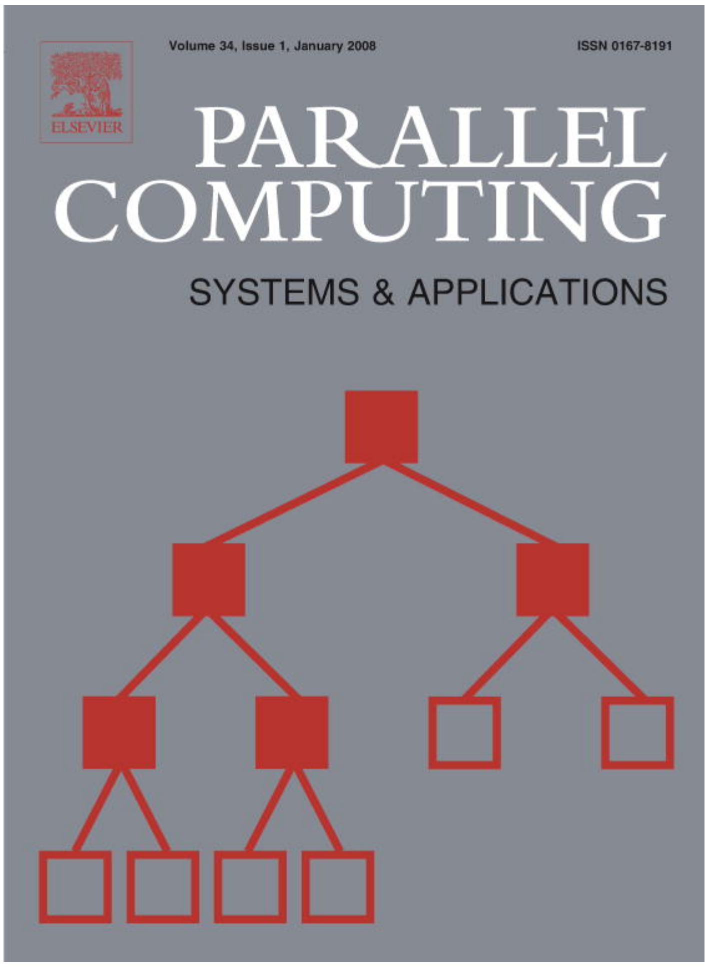

Volume 34, Issue 1, January 2008
ISSN 0167-8191
ELSEVIER
PARALLEL COMPUTING
SYSTEMS & APPLICATIONS

Available online at www.sciencedirect.com

ScienceDirect

Parallel Computing 34 (2008) 1–16

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Impact of message compression on the scalability of an atmospheric modeling application on clusters

V. Santhosh Kumar [a], R. Nanjundiah [c], M.J. Thazhuthaveetil [a,b], R. Govindarajan [a,b,*]

[a] *Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore 560 012, India*
[b] *Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India*
[c] *Center for Atmospheric and Oceanic Sciences, Indian Institute of Science, Bangalore 560 012, India*

Received 4 August 2006; received in revised form 22 May 2007; accepted 16 October 2007
Available online 24 October 2007

Abstract

In this paper, we study the scalability of an atmospheric modeling application on a cluster with commercially available off-the-shelf interconnects. It is found that interconnects with large latency and low bandwidth are major bottlenecks for performance scalability. Response curves for latency shows that for large message sizes latency is extremely sensitive to the size of the message. Thus, decreasing the message size could reduce the latency and hence improve the scalability.

We propose both lossless and lossy (*i.e.*, with loss of some information) compression schemes to reduce message sizes. These compression techniques are investigated for the Community Atmospheric Model (CAM), which is a large scale parallel application used for global climate simulation, on a IBM Power 5 Cluster with Gigabit interconnect. This is a floating point intensive application which involves both point-to-point and collective all-to-all communication of large messages (>128 KB). Floating point data which constitute the messages in CAM application results in 14.8% compression when lossless compression is employed and the speedup improves by about 18% on 32 processors. We further evaluate three lossy compression schemes with very low overheads (0.15%). We study the acceptability criteria for information loss in the lossy compression schemes using a perturbation growth test procedure. The lossy compression schemes achieve a message size reduction of 66.2% and an execution time speedup of up to 20.78 on 32 processors. We also look at the criteria for acceptability of loss of information in lossy compression techniques.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Cluster computing; Communication layer; Message compression; Parallel application; Atmospheric modeling

1. Introduction

Message passing based parallel systems have become the platform for main stream supercomputing due to their highly scalable nature. However, the performance of the parallel applications run on them is heavily

[*] Corresponding author. Address: Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore 560 012, India. Tel.: +91 80 2360 0654; fax: +91 80 2360 2648.

E-mail addresses: gvsk@hpc.serc.iisc.ernet.in (V.S. Kumar), ravi@caos.iisc.ernet.in (R. Nanjundiah), mjt@serc.iisc.ernet.in (M.J. Thazhuthaveetil), govind@serc.iisc.ernet.in (R. Govindarajan).

0167-8191/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2007.10.002

dependent on the communication subsystem. This dependence on the interconnect is even more critical in clusters where the interconnect is a commercially available off-the-shelf hardware such as a Gigabit ethernet which suffer from high latency and relatively low-bandwidth. The bottleneck becomes even more severe when the interconnection network is saturated with a large number of processors sending large messages. As will be shown in Section 3, when the network is saturated, communication latency increases drastically with increase in message size. In order to achieve better performance, it is important to reduce the message latencies. Message latencies can be reduced by reducing the message size.

One of the techniques commonly used for reducing message size is message compression. We investigate two techniques of message compression, viz., lossless compression and lossy compression. Lossless compression, as the name indicates, involves no loss of information and enables perfect reconstruction of data after reception. Lossy compression, on the other hand, involves some acceptable loss of information. As in all lossy techniques the threshold of acceptability could be a matter of concern and debate. It would largely depend on the type of problem being solved. Techniques of data compression have been widely studied. Ke et al. use lossless compression schemes for compressing large messages of float values to improve large message latency and hence application performance [8]. Their approach implements message compression and decompression in cMPI, a modified version of MPI, using a value prediction scheme which computes the difference between the actual and predicted data values and encodes the difference using leading zero count (LZC) method to compress the messages.

In this work, we study the communication performance of Community Atmospheric Model (CAM) application and investigate how different message compression schemes can be used to improve the communication performance of CAM application, thereby delivering better speedup and scalability for the application. Community Atmospheric Model is a global atmospheric model developed at National Center for Atmospheric Research (NCAR) useful for global weather and climate prediction. In this work, we use the data parallel implementation of CAM application Version 3 (CAM-3) based on message passing interface (MPI) [9] running on IBM Power5 cluster interconnected with Gigabit Ethernet.

An excellent work in improving the scalability of CAM by optimizing the computation and communication stencils has been done by Worley [14]. Our approach is complementary to that of [14]. We do not modify their stencils but rather try to improve scalability by improving the efficiency of message passing. In essence the problem being addressed by us can be summarized as

> Given a complex computational code (such as CAM3) and a cluster with slow interconnects, can the scalability of the code be improved by using message compression techniques?

In comparison with [14], the stencil used by us is in the optimal range. Our initial studies on the IBM cluster show that scalability of CAM application is limited by its communication requirements.[1] The communication in CAM involves both point-to-point and collective communication of messages larger than 128 KB. This leads to poor performance on clusters with larger number of processors.

In this paper, we use compression to reduce the size of messages in CAM. The messages that are communicated in CAM consists of double precision floating point data. First, we investigate lossless compression techniques for messages in CAM. We then propose the use of simple lossy compression of messages, such as truncation of least significant mantissa bits in the floating point data or converting the double precision data in messages to single precision. However, lossy compression can affect the accuracy of the results generated and care must be exercised to ensure that both the numerical stability and the accuracy of the results is not affected. For this particular application we look at the acceptable threshold of information loss using the validation technique suggested by Rosinski and Williamson [11]. Our experimental evaluation shows that lossy compression schemes leads to significant improvement in the scalability of the CAM application, improving the speedup from 13.53 to 20.78 on 32 processors.

The main contributions of this paper are

- Development and evaluation of the performance of lossless and lossy message compression for message passing.

[1] We tested the same code on an IBM Regatta, a SMP, and it shows excellent scalability.

- Application of these techniques to the complex computational code CAM and their impact on the scalability on a cluster with slow interconnects such as the IBM Power5 cluster.

The rest of the paper is organized as follows: In Section 2, we give a description of the Community Atmospheric Model (CAM). In Section 3, we motivate how message compression can reduce the message latency and hence improve the speedup. Section 4 discusses our lossless compression schemes and its performance results. Following this, we discuss our lossy compression schemes in Section 5 and its validation in Section 6. In Section 7, we discuss the results of lossy compression schemes. Section 8 discusses the related work and Section 9 provides concluding remarks.

2. Community atmospheric model (CAM)

In this section we describe the CAM model and its parallel implementation. We have used Community Atmospheric Model Version 3 (CAM3) developed by National Center for Atmospheric Research (NCAR), Boulder [5]. It is a widely used atmospheric model and contains components contributed by the atmospheric modeling community from all over the world. Its source code is freely available under Open Public License.

The CAM model solves equations of momentum, energy, mass and moisture conservation. We have used the model at T42 resolution. This corresponds to a horizontal grid resolution of 64 latitudes and 128 longitudes [7], corresponding to a horizontal resolution of about 280 kms and 28 levels in the vertical direction. Of the three options of dynamical core (Eulerian, Semi-Lagrangian and Finite Volume), we have used the Eulerian version in our current study. Moisture transport, however, is Semi-Lagrangian. A detailed description of CAM3 is available in [5]. For computation of model physics we use a ‘pcol’ value of 16. This is the default value in CAM3 and comparison with Worley [14] shows that this is in the optimal range for most platforms.

The parallel implementation of the model has options for both OpenMP and MPI. It uses the domain decomposition technique to distribute load amongst the processing elements. In the current implementation, a one-dimensional domain decomposition along the latitudinal (north–south) direction has been used. The physical core of the model, involving radiation, boundary-layer parameterization, cloud physics, etc. is largely unaffected by this decomposition, as these computations are oriented in the vertical direction, have no horizontal dependencies and hence parallelize trivially. The dynamical core, however, involves substantial communication between processing elements. In the present method of domain decomposition, computations of Legendre Transform and Semi-Lagrangian advection of moisture require such communication. Fast Fourier Transform (FFT), which is in the zonal or east–west direction, is unaffected by this decomposition. The communication structure of the Semi-Lagrangian advection is of the nearest-neighbor form while that of Legendre Transform is global. This means that the coefficients are spread over all the nodes and hence involve all-to-all communication to collect the values. The model is run in a number of time steps over the duration for which the climate simulation is required. In each time step the physical core and the dynamical core are executed. For the rest of the paper, we will refer to these cores of CAM as the Physics, which is computationally intensive, and the Dynamics, which is communication intensive.

3. Motivation for message compression

In this section, we motivate the need for message compression. We studied the behavior of the CAM code on an IBM Power5 cluster with Gigabit Ethernet interconnect. Table 1 shows results from our initial performance study of the CAM code for 1–32 processors. Observe that the speedup increases linearly with increasing number of processors up to 4. The speedup improvement drops beyond 4 processors and the speedup achieved for 32 processors is only 13.53.

To understand this behavior we measured the execution time in the two major phases of CAM namely the Physics core (computationally intensive) and the Dynamics core (communication intensive). These numbers are reported in the columns 4 and 5 of Table 1. Observe that the physics phase of CAM shows near-linear improvements in speedup with increasing number of processors, with a speedup of 26 for 32 processors.

Table 1
CAM performance: T42 resolution, 60 days

Number of processors	Execution time (s)	Speedup	Physics (s)	Dynamics (s)
1	34,044	1.00	29,496	3467
2	18,001	1.89	15,270	2229
4	9745	3.49	7842	1648
8	5827	5.84	4108	1577
16	3447	9.88	2180	1180
32	2516	13.53	1127	1317

Table 2
Number of double precision floating point values communicated by routines of CAM

Number of processors	bndexch	realloc4a	realloc4b	scan2	realloc7
2	17,030	2,87,584	2,91,712	1440	2656
4	17,030	2,15,688	2,18,784	720	1328
8	17,030	1,25,818	1,27,624	360	664
16	17,030	67,402	68,320	180	332
32	17,030	34,825	35,324	90	166

The execution time of the dynamics core, on the other hand reduces by only 2.63 for 32 processors. Thus, it is the dynamics core that limits the overall scalability of the parallel CAM application.

We next analyzed the communication behavior in the dynamics phase of the CAM application, identifying the communication patterns and the sizes of messages being communicated. We found that during each iterative time step in the CAM computation, the communication is performed in 5 routines – `bndexch`, `realloc4a`, `realloc4b`, `realloc7` and `scan2`. The `bndexch` routine performs a point-to-point communication using `MPI_Isend` and `MPI_Irecv` routines. `Realloc4a` and `realloc4b` perform collective communication using calls to `MPI_Alltoallv`, while `realloc7` and `scan2` call `MPI_Gatherv`. Table 2 shows the number of double precision floating point values communicated by these routines. It can be seen that the `bndexch`routine communicates 17,030 doubles or 133 KB, while for the other routines the size roughly halves as the processor count doubles. `Realloc4` routine communicates a minimum of 34,825 doubles or 272 KB while `scan2` and `realloc7` communicate a maximum of 1440 doubles or 11 KB and 2656 double or 21 KB, which are relatively small messages.

In order to understand the performance impact of communicating such large messages, we measured the latency incurred by these MPI routines on our cluster, using the micro benchmark *SKaMPI* [12]. SkaMPI is a micro benchmark for evaluating MPI implementations performing a number of runs to minimize standard error in execution time. It also measures the time for collective operations accurately based on the global time [15]. In Figs. 1 and 2, we plot the latencies for messages of various sizes, both for pingpong and `MPI_Alltoallv` type of communication on an 8-node cluster.[2] It can be observed that the communication latency increases sharply beyond a message size of 32 KB for pingpong and 64 KB for `MPI_Alltoallv`. Observe that for pingpong a reduction in message length from 128 KB to 64 KB causes 50% reduction in latency. For `MPI_Alltoallv` a reduction in message length from 1 MB to 512 KB reduces the latency by 60%. Thus, for the message sizes used in CAM, a significant reduction in message latency will result from reduced message sizes.

Hence, we use *message compression* to reduce the size of messages being communicated in CAM in order to improve the application performance. By message compression we mean that messages are compressed before being transmitted to the receiver. The receiver decompresses the message to regenerate the original message.

[2] The IBM cluster used in our experiments consists of IBM e720 series SMP nodes, each with four Power5 processors operating at 1.65 GHz, and sharing 4 GB memory and 1 Gigabit NIC for communication. Communication across processors within a node does not use shared memory as the MPICH implementation is compiled with `noshared` flag. In this experiment, we used communication amongst eight processors on eight different nodes.

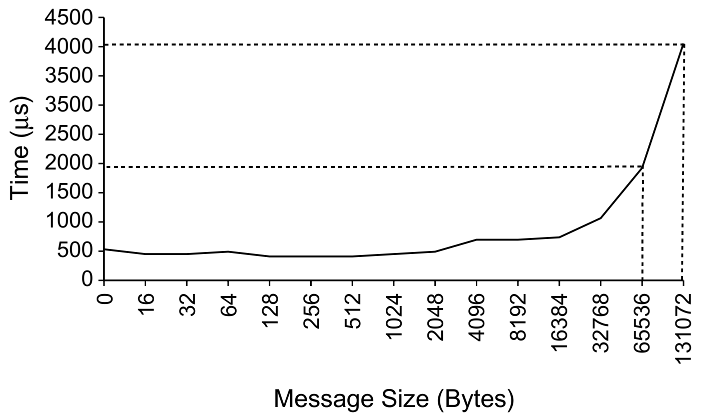


Fig. 1. Pingpong latency on IBM Power5 Gigabit Ethernet cluster.

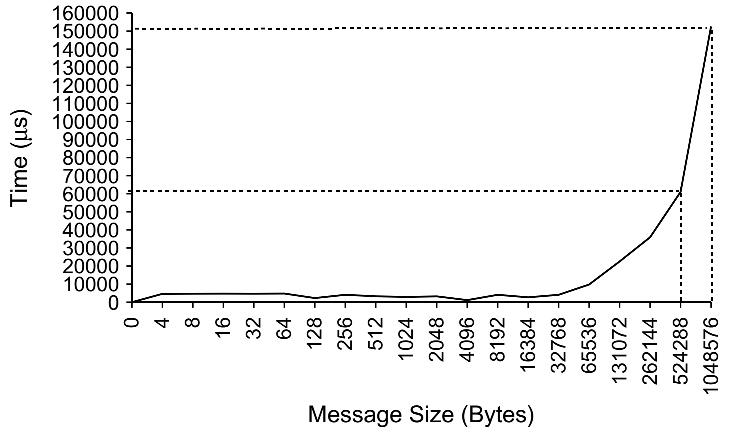


Fig. 2. `MPI_Alltoallv` latency on IBM Power 5 Gigabit Ethernet cluster.

4. Lossless compression schemes for CAM

We first study the effect of lossless compression of messages on the CAM application performance. Several compression techniques, *e.g.*, compress [13], gzip [6], leading zero count (LZC) [8], have been proposed in the literature. Under lossless compression, when the compressed message is decompressed, the original message is recreated exactly, without any loss of information. Thus, the use of lossless compression guarantees that the correctness of the results is not affected and is identical to that produced when no message compression is employed.

The CAM application executes in iterative time steps and in each time step it communicates messages containing arrays of double precision floating point values represented in the 64 bit IEEE 754 floating point format with 1 sign bit, 11 exponent bits and 52 mantissa bits. We modify CAM so that these arrays of floating point value are compressed before transmission and decompressed upon reception. We study two types of lossless compression schemes: intra-step and inter-step. Both these schemes rely on our observation that the values in the arrays being communicated are correlated, explained in Section 4.1. The intra-step compression scheme compresses the message array based on the correlation of data values in the array within each time step, whereas the inter-step compression scheme compresses the array based on the correlation of data values across time steps.

4.1. Intra-step lossless compression

We have observed that, for the CAM application, the most significant bits of successive double values in a message that is transmitted in a time step are correlated. By this we mean that successive values tend to differ

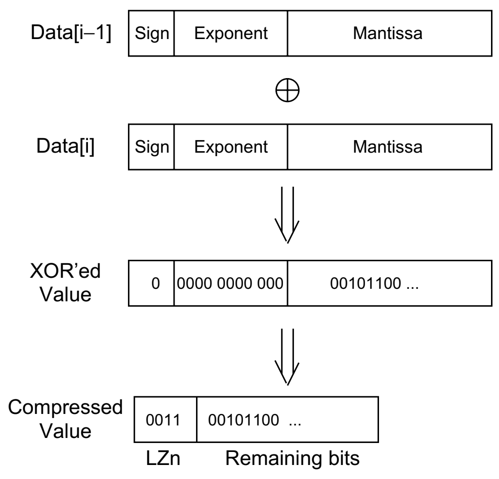


Fig. 3. Intra-step lossless compression.

in only a few most significant bits, and hence performing a bit-wise XOR produces a value with leading zeros. The number of leading zero nibbles[3] can be encoded and hence compressed. We use the nibble as the unit of compression as in [8]. We define a compressibility metric, LZ nibble ratio, as the ratio of the number of leading zero nibbles to the total number of nibbles sent by a single routine. This metric reflects the amount of compressibility that can be achieved by using the nibble as the unit of compression. The LZnibble ratios for the routines `bndexch`, `realloc4a`, and `realloc4b` are 15.14%, 6.08% and 5.91% in these routines, respectively.

Fig. 3 shows how our intra-step lossless compression scheme works on an array of double precision floating point values in a message. Successive double values (*Data[i-1]* and *Data[i]*) in a message are XOR-ed to produce the value *Xor_Val*. The leading zero nibbles in *XOR_Val* is encoded using a 4-bit count and the remaining bits are left uncompressed. Thus, each double precision value can be encoded as $(64 - 4 * LZn) + 4$ bits, where LZn is the leading zero nibble count. These compressed values are then packed and transmitted as a message. Decompression involves extracting the LZn from the message and then forming the double precision number by perpending $4 * LZn$ zeros in the MSB to the next $(64 - 4 * LZn)$ bits from the message. This is then XOR-ed with the previous value (*Data[i-1]*) to generate the decompressed number (*Data[i]*). Note the when LZn is zero, the compression scheme can actually result in an increase in size due to the LZn count bits.

4.2. Performance of CAM with intra-step lossless compression

We evaluate the performance benefits of intra-step compression the CAM application on an IBM Power5 based cluster with Gigabit Ethernet interconnection network. Each node in the cluster has four 1.65 GHz Power5 processors with 4GB of RAM and runs SUSE Linux 9.0 operating system. We used MPICH-1.2.7 as the message passing library for the CAM application with `-noshared memory` option. This means that communication between processes in a single node does not use shared memory. The compression and decompression routines are implemented as a library and are called by the application before message transmission and after message reception, respectively. We ran CAM using datasets of T42 resolution [7] for 60 days. When the CAM application is run, it reports the total timing for each process along with the time taken for physical core and dynamical core. We perform replications of our CAM timing runs so that the standard error in the total execution time is less that 1%. Although each node in the cluster has four processors, for comparison of our different compression schemes we run only 1 process per cluster node. This is to ensure that other effects such as multiple processes sharing the shared memory or NIC does not influence the comparative of results. However, the performance results with 2 processes per cluster node are also reported in Section 7.2.

[3] A nibble consists of 4 bits. A zero nibble is `0000`.

Table 3
CAM performance T42: speedup for intra-step lossless compression scheme

Number of processes	Original execution				With intra-step lossless execution			
	Total execution time (s)	Speedup	Physics execution time (s)	Dynamics execution time (s)	Total execution time (s)	Speedup	Physics execution time (s)	Dynamics execution time (s)
1	34,044	1.00	29,496	3467	–	–	–	–
16	3447	9.89	2180	1180	3370	10.10	2146	1141
32	2516	13.53	1127	1317	2419	14.02	1134	1234

Table 3 shows the speedup and breakup of execution time spent in physics and dynamics cores for the original CAM code execution and execution with our intra-step lossless compression scheme. The speedup is calculated with respect to the execution time of a sequential run. We find that intra-step compression results in a marginal reduction in execution time and hence a marginal improvement in speedup, from 9.89 to 10.1 for 16 processes and from 13.53 to 14.02 for 32 processes. We also note that the reduction in the time spent in dynamics is only by 6% for 32 processes. Since dynamics core is communication intensive and the message sizes directly influence the communication time, we measured the reduction in message sizes due to our intra-step compression, shown in Table 4. Note that while `bndexch` routine achieves a reduction in size of data communicated of up to 8.9%, the `realloc4a` and `realloc4b` routines achieve no compression at all. In fact, there is a small increase in the message size in these routines due to LZn overhead. Thus, the reduction in execution time of dynamics core or the overall improvement in speedup is only marginal.

Since this scheme does not result in good compression we next look at another lossless compression scheme, which takes advantage of the correlation between floating point values across iterative time steps.

4.3. Inter-step lossless compression

In this scheme we exploit the fact that data values from successive time steps are correlated, because they represent the temporal variation of the same parameter over the same sub-domain. We refer to our compression scheme based on this phenomenon as inter-step lossless compression. In this scheme, *Data[i]* of the current time step is XORed with *Data[i]* of the previous time step, denoted as *Prev_Data[i]*. The LZnibble ratios for the routines `bndexch`, `realloc4a` and `realloc4b` were found to be 21.06%, 19.68% and 19.31%, respectively. We observe that the ratios are higher than those for intra-step lossless compression and we expect higher compression ratios and better application performance.

Fig. 4 illustrates how inter-step compression is done. *Prev_Data[i]* and *Data[i]* from the previous and current time steps respectively are XOR-ed to produce the *Xor_Val*. The leading zero nibbles are encoded using the leading zero nibble (LZn) count and the remaining bits are left uncompressed as described in Section 4.1. These compressed data are then packed, transmitted and decompressed at the receiving end. Note that the inter-time step compression requires not only the data for the current time step but also the data for all the messages in the previous time step. Since the message sizes in CAM application are large, this scheme requires considerable space overheads.

Table 4
Reduction in data size due to intra-step lossless compression scheme

Number of processors	Routine	Bytes sent (GBytes)	Compressed bytes sent (GBytes)	Reduction in data size (%)
16	Bndexchg	49.34	45.00	8.8
	realloc4a	34.72	34.94	−0.6
	realloc4b	35.22	35.37	−0.4
32	Bndexchg	100.88	91.91	8.9
	realloc4a	35.88	35.95	−0.2
	realloc4b	36.39	36.53	−0.3

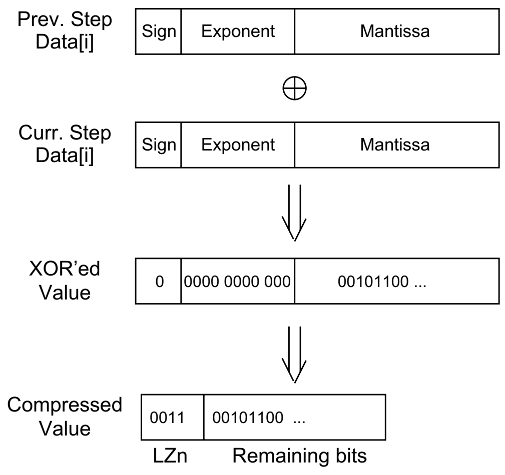


Fig. 4. Inter-step lossless compression.

Table 5
CAM performance T42: speedup for inter-step lossless compression scheme

Number of processes	Original execution				Inter-step lossless execution			
	Total execution time (s)	Speedup	Physics execution time (s)	Dynamics execution time (s)	Total execution time (s)	Speedup	Physics execution time (s)	Dynamics execution time (s)
1	34,044	1.00	29,496	3467	–	–	–	–
16	3447	9.89	2180	1180	3034	11.22	2185	758
32	2516	13.53	1127	1317	2130	15.99	1166	895

4.4. Performance of CAM with inter-step lossless compression

Table 5 shows the speedup and breakup in execution time for original CAM and the execution time with our inter-step compression scheme. We obtain a speedup of 11.22 and 15.99 for 16 and 32 processes, respectively, higher than under the intra-step lossless compression scheme. It can be seen that the time spent in dynamics has reduced by 32% for 32 processes, as compared to a reduction of 6% under intra-step compression. The reduction in message sizes due to inter-step compression scheme are shown in Table 6. We observe that inter-step compression results in reduction of message sizes by 13.4% in `realloc4a` and `realloc4b` routines and 14.8% in `bndexch`. This leads to a good reduction in communication time and hence higher speedups. However, the best reduction achieved is still only 14.8%.

We measured the overhead in compressing and decompressing all messages across all instances of `bndexch` and `realloc4` routines. Since the compression and decompression are performed at runtime, it is desirable that these overheads are as low as possible. We found that the compression and decompression overheads are 1.88% of total execution time.

As the maximum speedup achieved by lossless compression schemes for 32 processors is still only about 16, we next look at the possibility of using lossy compression schemes to achieve higher reduction in message sizes along with low overheads and hence deliver better application performance. However, as lossy compression involves loss of information, it is necessary to find the acceptable threshold for information loss in the CAM application.

5. Lossy compression schemes for CAM

The lossy compression scheme we propose here is less sophisticated than lossless schemes discussed in the earlier section or the ones used by Ke et al. [8]. Yet, the proposed scheme achieves a good compression ratio.

Table 6
Reduction in data size due to inter-step lossless compression scheme

Number of processors	Routine	Bytes sent (GBytes)	Compressed bytes sent (GBytes)	Reduction in data size (%)
16	Bndexch	49.33	42.04	14.8
	realloc4a	34.71	30.06	13.4
	realloc4b	35.21	30.62	13.0
32	Bndexch	100.86	85.93	14.8
	realloc4a	35.87	31.06	13.4
	realloc4b	36.38	31.65	13.0

Further, the simplicity of the scheme ensures that the resulting overhead in compression and decompression is minimal.

In lossy compression, some information is lost during compression, and hence decompression does not result in a message that is identical to the original message. This can lead to loss of accuracy in the results generated by the application. For CAM application we try to find an acceptable threshold for this information loss. This threshold of acceptability could be problem specific and depends on the acceptable loss in accuracy. If a higher degree of precision in communication is considered necessary then the user should suitably change the compression limit.

In the CAM model, the highest precision is required in the calculation of Gaussian latitudes, Gaussian weights and the Legendre polynomials. Errors in computation of these can lead to significant loss of accuracy and computational instability. Hence, these computations should still be conducted in double precision and be unaffected by the proposed lossy compression method. However, the temporally varying spectral coefficients which are multiplied with the polynomial are less sensitive to precision and hence lossy compression can be applied on them. We design our lossy compression scheme to affect only the least significant mantissa bits in the double precision values of temporally varying spectral coefficients. Thus, the accuracy of computations is not likely to be seriously affected. To confirm this, we ran CAM with double precision computations and lossy communication and found that the in-built error criteria was not violated. Additionally, using the validation techniques suggested by Rosinski and Williamson [11], we found that the RMS differences are within acceptable limits, as discussed in Section 6. When we tried to run the model using single precision computation in-built error criteria within the code was violated. Thus, it appears that lossy compression of messages could be a viable option for improving scalability of CAM.

We have experimented with three lossy compression schemes on double precision floating point values in CAM. During message transmission, we transmit only the most significant bits of double precision values required to maintain the accuracy of the results generated by the application. In the following subsections, we elaborate on our three lossy compression schemes.

5.1. Double precision to lossy$_N$ (DP-Lossy$_N$)

In the DP-Lossy$_N$ method of lossy compression we leave out the least significant N bits and transmit only the remaining $(64 - N)$ bits of the double precision floating point number. Fig. 5 shows an example where N is

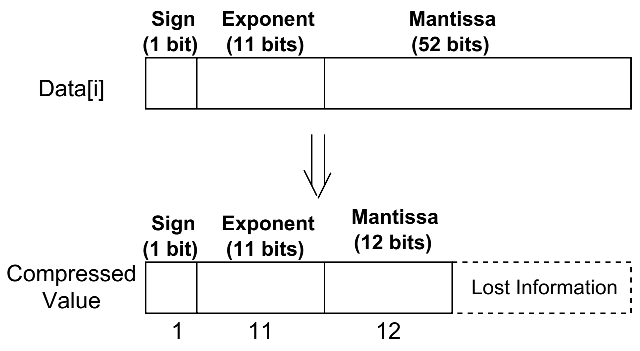


Fig. 5. DP-Lossy$_{40}$.

40. We choose N such that the most significant $(64 - N)$ bits always include the sign and exponent bits and some of the most significant bits of the mantissa. Further N must be chosen such that the numerical stability of the CAM application is not affected. Thus, the DP-Lossy$_N$ involves packing the most significant $(64 - N)$ bits of each data value in the message, which is then transmitted. When the message is received, each $(64 - N)$ bits is unpacked into a 64 bit double value, appending N zeros in least significant bit positions. In our work we try values of $N = 32$ and $N = 40$. It may be noted that DP-Lossy$_{40}$, where the least significant 40 bits are not transmitted, results in a straight 62.5% reduction is message size.

5.2. Double precision to single precision (DP-SP)

In the DP-SP lossy compression method, we typecast each double precision floating point data in a message to a single precision floating point value. The number of bits in the resulting conversion is thus fixed to 32 bits as dictated by the IEEE Single Precision floating point format. It can be seen that the DS-SP scheme gives a message size reduction of 50% with a fixed accuracy, while the DP-Lossy$_N$ scheme allows us to trade off between accuracy and message size reduction, by choosing an appropriate value for N.

5.3. Lossy with lossless scheme

In this method, we use lossless compression along with either of the lossy compression schemes, namely DP-Lossy$_N$ and DP-SP. More specifically, one of the lossy compression scheme is applied first and the inter-step lossless compression is applied subsequently on the truncated values. Note that the lossy and lossless compression schemes are orthogonal to each other in the sense that lossy compression removes the least significant bits while lossless compression takes advantage of the correlation in the most significant bits. We refer to the combined lossy and lossless compression scheme as Lossy + Lossless in general; DP-Lossy$_N$ + Lossless or DP-SP + Lossless indicate the specific lossy scheme used in the combined scheme.

6. Validation of lossy compression schemes in CAM

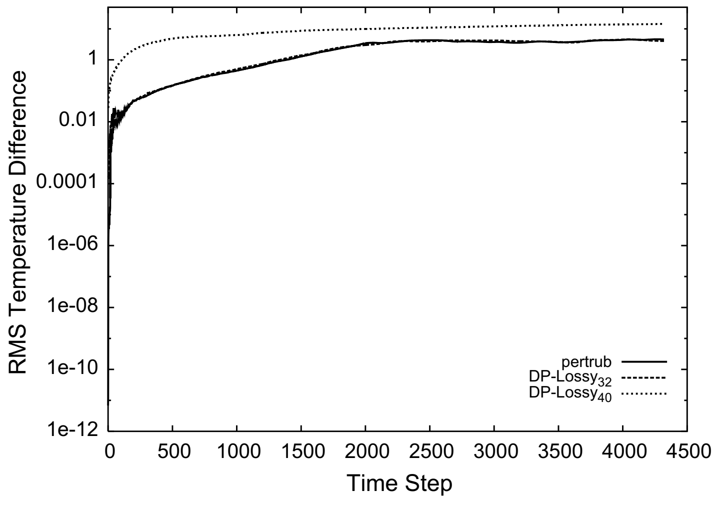
When we use lossy compression for transmitting messages in the CAM application, we must ensure that the accuracy of results generated by the application is not compromised. To check the impact of our lossy compression schemes on the correctness of CAM, we used a validation procedure called the perturbation growth test suggested by Rosinski and Williamson [11]. Recently, Jablonwski and Williamson [16] have suggested a set of rigorous tests to test the dynamical core of a GCM. Use of these tests would require major modifications to the code and would be outside scope of the present work. In their analysis, for quantitative comparisons they suggest the comparison of L2 norm for surface pressure (Eq. (16) of their paper). We compare the L2 norm for temperature. We think temperature is a better variable for comparison as it is directly affected by both physics and dynamics of the model while surface pressure is indirectly affected by the physics. In addition, truncation of messages would not only affect the dynamical core of the model but also the computation of physics. Hence we have used the procedure of Rosinki and Williamson [11]. The steps involved in this test are described briefly below. Additional detail can be found in [10]. The perturbation growth test for lossy compression schemes involves comparing the error growth in the output of runs of CAM with and without our lossy message compression enabled. Thus, the perturbation growth test consists of three runs of CAM:

1. control run of original code,
2. perturbation run of original code, and
3. run with lossy compression enabled.

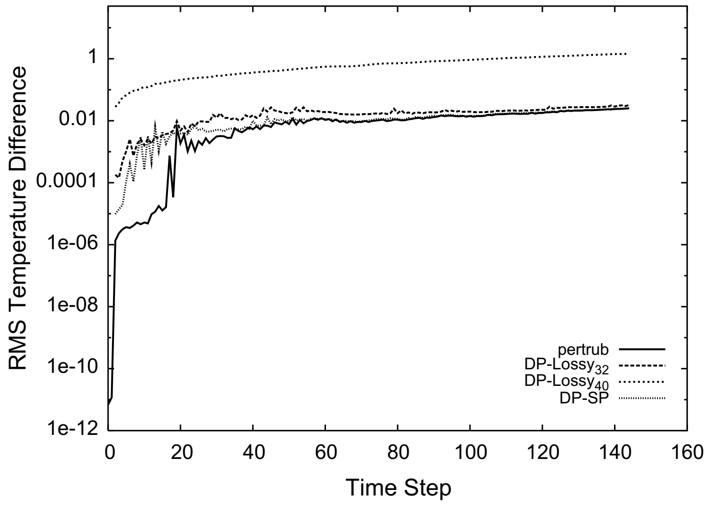
During these runs the output temperature field is written to a file. The perturbation run of the original code consists of applying a small perturbation to the initial temperature field. This is equivalent to introducing a small error in the lowest order bits of the floating point values. The perturbation introduces is typically equal to the machine epsilon. The output of this perturbation run, when compared with that of the original run, represents the error growth due to perturbation. One has to check whether this error growth is acceptable for the CAM application [11]. We use the error growth of the perturbation run and compare it with the error growth of the runs with lossy compression. If the error growth due to our lossy compression is no worse than that of the perturbation run, then the output of CAM with lossy compression of messages is considered to be acceptable.

Fig. 6 shows the comparison of error growth for the lossy compression schemes we have proposed. The y-axis represents the RMS Temperature difference and the x-axis represents the simulated time in Time Steps. Each time step simulated in CAM at T42 resolution corresponds to 20 min of actual physical time. Fig. 6a shows the error growth of the RMS Temperature difference for 60-days run, *i.e.*, 4320 ($=3 * 24 * 60$) time steps. Observe that the error growth of DP-Lossy$_{32}$ is no higher than the perturbation error growth rate. We found that DP-SP exhibits similar behavior. On the other hand, the error growth rate of DP-Lossy$_{40}$ is at least an order of magnitude higher than that observed with ‘perturb’.

Further, the error growth rate for all runs is highest during the initial 2 days and stabilizes after that. Hence in Fig. 6b we plot the RMS temperature difference for the initial 2 days (*i.e.*, 144 time steps) to show the difference in RMS during the initial period. Observe from Fig. 6b that the initial growth rate of error for DP-SP is less than that for DP-Lossy$_{32}$ because DP_SP includes more mantissa bits than DP-Lossy$_{32}$. However, after 2 days both of them show the same stable behavior. This suggests that CAM application could use a lossy



(a) 4320Time Steps (60Days)



(b) 144 Time Steps (2Days)

Fig. 6. Comparison of error growth for different lossy compression schemes with that of perturbation run of CAM: (a) 4320 time steps (60 days); (b)144 time steps (2 days).

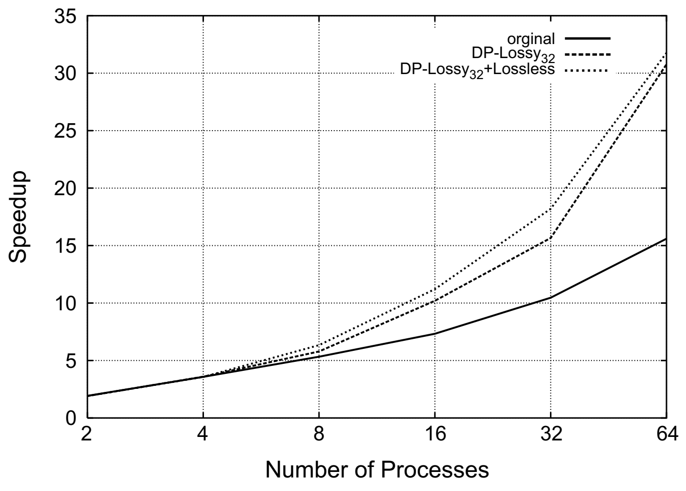


Fig. 7. Speedup for two process per node.

compression of messages (of up to 32 least significant bits of double precision values) and still generate acceptable output. So, we evaluate the performance of lossy compression schemes DP-Lossy$_{32}$ and DP-SP for the CAM application Fig. 7.

7. Performance of CAM with lossy compression schemes

As described in Section 4.2, each node of the cluster is an SMP (Symmetric MultiProcessor) containing four Power5 processors, with these processors sharing memory and network interface card. In such cluster of SMPs, the performance scalability is likely to be affected by the number of processes assigned to each node (SMP). In our first set of experiments, we assign one process per node (SMP) and report the benefits of our Lossy Compression Schemes. Subsequently in Section 7.2, we report results with 2 processes per node, where each process is assigned to a different processor in the node. This enables us to make a uniform comparison of the different compression schemes, with the same number of processes per node.

7.1. Performance of CAM with 1 process per node

Table 7 reports the execution time of CAM application under different compression schemes for 1–32 processes. It also shows the speedup up to 32 processes relative to single processor execution time. We observe that for 32 processes DP-Lossy$_{32}$ and DP-SP schemes achieve a speedup of 20.78 and 20.57, respectively, which is more than 50% higher than the speedup achieved by the original CAM code without any message compression. In comparison, the lossless compression scheme only achieves a speedup of 15.99, which is an improvement of 18% over the original scheme. We notice that the Lossy + Lossless compression does not give

Table 7
CAM performance: execution time (secs) and speedup up to 32 processes relative to single processor

Scheme	Number of processes											
	1		2		4		8		16		32	
	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speedup	Time	Speed
Original	34,044	1.00	18,001	1.89	9745	3.49	5827	5.84	3447	9.88	2516	13.53
Inter-step lossless	34,044	1.00	18,007	1.89	9748	3.49	5449	6.25	3034	11.22	2130	15.99
DP-Lossy$_{32}$	34,044	1.00	17,667	1.93	9407	3.62	5111	6.66	2727	12.48	1638	20.78
DP-SP	34,044	1.00	17,691	1.92	9380	3.63	5092	6.69	2762	12.32	1655	20.57
DP-Lossy$_{32}$ + lossless	34,044	1.00	17,933	1.90	9459	3.60	5131	6.64	2834	12.01	1693	20.11

Table 8
CAM performance: breakup of time in physics and dynamics for 1, 2, 4, 8, 16 and 32 processes

Scheme	Physics (s)						Dynamics (s)					
	1	2	4	8	16	32	1	2	4	8	16	32
Original	29,496	15,270	7842	4108	2180	1127	3467	2229	1648	1577	1180	1317
Inter-step lossless	29,496	15,231	7864	4111	2185	1166	3467	2272	1615	1196	758	895
DP-Lossy$_{32}$	29,496	15,220	7847	4086	2118	1258	3467	1943	1273	883	527	313
DP-SP	29,496	15,273	7863	4084	2165	1260	3467	1917	1256	867	513	318
DP-Lossy$_{32}$ + lossless	29,496	15,263	7852	4068	2115	1201	3467	2170	1347	922	628	414

any additional benefits compared to the lossy schemes. In fact, there is a slight degradation in speedup compared to lossy schemes.

Table 8 shows the time spent in the physics and dynamics cores of CAM. As before, the time spent in the physics core decreases linearly with increasing number of processes and almost to the same extent. The dynamics core shows a better improvement in execution time with increasing number of processes for the lossy compression schemes, reducing from 3467 s (for original execution on a single process) to 313 for 32 processes with DP-Lossy$_{32}$ or DP-SP schemes. The execution time for the dynamics core with 32 processes under lossy compression scheme is lower by a factor of 4 compared to that in the original execution time. This reduction directly translates to improved speedup of the CAM application.

Table 9 shows the reduction in message sizes obtained with the DP-Lossy$_{32}$ + Lossless compression scheme. It can be observed that the message size reduction is between 64.0% and 66.2% which is better than the 14.8% improvement achieved in inter-step lossless compression (Table 6), resulting in a speedup of 20.11 for 32 processes. We measured the overhead in compression and decompression of messages and report them as percentages of total execution time. The maximum overhead from lossy compression is 0.15%, while that for lossless compression is 2.35%. The lossless compression overhead is higher due to more housekeeping work, *i.e.*, determining count of leading zeros, packing of bits are arbitrary bits positions, whereas the lossy compression overhead is significantly lower because of the low overhead of bit masking required for DP-Lossy$_N$ scheme. Thus, the lossy compression schemes are very attractive as they result in good compression ratios and reduce the message communication time significantly, while incurring low compression overheads.

7.2. *Performance of CAM with 2 process per node*

As mentioned earlier, the performance of the CAM application on a cluster of SMPs, such as the one used in our experiment, for a fixed number of processes may vary somewhat depending on the number of processes assigned per node (or SMP). For example, the execution time of CAM under the original scheme for 32 processes with 1, 2 and 4 processes on 32, 16 and 8 nodes are 2516 s, 3255 s and 4401 s, respectively. This increase in execution time is possibly due to the interaction of processes in a node and the sharing of network interface card (NIC) by all processes in the same node. Further, since the processes in CAM execute in a synchronized manner with barrier synchronization, the processes in a node may compete for the Gigabit NIC at the same time. Thus, more processes per node may result in an increase in execution time. Hence, the performance of CAM application for a fixed number of processes, say 32 processes, with 1 process per node cannot be

Table 9
Reduction in data size due to DP-Lossy$_{32}$ + lossless compression scheme

Number of processors	Routine	Bytes sent (GBytes)	Compressed bytes sent (GBytes)	Reduction in data size (%)
16	Bndexch	49.33	16.68	66.2
	realloc4a	34.71	12.37	64.4
	realloc4b	35.21	12.67	64.0
32	Bndexch	100.86	34.08	66.2
	realloc4a	35.87	12.79	64.3
	realloc4b	36.38	13.11	64.0

compared to that with 2 or more processes per node. However, it is still useful to study the performance of the compression schemes on CAM where 2 or more processes are assigned to processors in a node, as this would enable us to understand the effects of compression schemes on larger number of processes and using the system to a better capability. We evaluated performance scalability of the CAM application for up to 64 processes, assigning two processes to processors in a single node. We concentrate on the DP-Lossy$_{32}$ and the DP-Lossy$_{32}$ + Lossless schemes in this experiment. While the base scheme achieves a speedup of only 15.6 on 64 processes, DP-Lossy$_{32}$ and DP-Lossy$_{32}$ + Lossless schemes achieve speedups of 30.77 and 31.17, which is an improvement by almost a factor of 2. Thus, we observe that the compression schemes bring in greater benefits especially for larger number of processors.

8. Related work

The work presented in this paper focuses on reducing communication time in message passing based parallel application using compression. We evaluate the performance of both lossless and lossy compression schemes on the CAM application.

Ke et al. [8] evaluate the effect of lossless message compression in the context of parallel scientific applications based on MPI. They use value prediction based on DFCM [4] to compute the difference between the actual and predicted data values which is then encoded using leading zero count (LZC) method to compress the messages. Ke et al. implement message compression in a modified version of MPI, known as cMPI, which requires that the predictor table state should be kept updated and consistent at both sender and receiver. In contrast, our lossy compression scheme neither requires any additional storage nor incurs considerable computational overhead. Further, we exploit the ability of the application to tolerate certain loss in accuracy by adopting lossy compression schemes. This enables our approach to achieve a reduction of 50–66.2% in message size or a compression ratio of 2–2.96 with a compression overhead of 0.15%. In contrast, Ke's method achieves a compression ratio of 1.36 (for the BT benchmark from NAS parallel benchmark suite) with a compression overhead of up to 1.8%. While we consider the results of lossy compression to be acceptable (based on the verification procedure of Rosinski and Williamson) the threshold of acceptability could vary with the problem being addressed. We caution the user to use an appropriate criteria of acceptability.

Compression has been used in processor caches. Alameldeen et al. [2] evaluate a compressed cache design, where the data in level-2 cache are compressed, which increases the effective cache capacity and reduces off-chip bandwidth. Their technique is based on Frequent Pattern Compression (FPC), which encodes the frequent data patterns (*e.g.*, zero runs, sign extensions, etc.) found in cache lines using lesser number of bits. The advantage of this hardware scheme is that the compression/decompression overheads are extremely low. FPC gives higher compression ratios (up to 2.4) for integer benchmarks than for floating point benchmarks [3]. Alameldeen et al. [1] use lossless compression in the context of Chip Multiprocessors (CMP). In link compression, data messages being transferred from a CMP and off-chip memory are compressed by the L3/Memory controller. This is important in CMPs, as increased processor cores per chip increases the demand for off-chip memory bandwidth. Link compression greatly reduces this off-chip bandwidth demand. However, they do not consider compression of messages across nodes of the CMPs.

9. Conclusions and future work

In a cluster using commercially available off-the-shelf communication interconnects, the slowness of links proves to be a major bottleneck in scalability. A method to overcome this is to compress the messages. Experiments with off-the-shelf interconnects show that message latency exhibits a threshold behavior with the size of message. More specifically, latency does not vary much for small messages; however, beyond a threshold (about 64 KB), the latency increases significantly with message size and is extremely sensitive to the size of the message. Thus, by reducing the message size, the latency of message passing can be reduced and this can lead to improved scalability.

We have studied two techniques of message compression, viz., lossless and lossy compression technique messages. In lossless compression, only those bytes which are changed between iterations are exchanged, while lossy compression involves some loss of information. While lossless compression results in perfect reconstruction, lossy compression can be applied only as far as the loss in information does not exceed a threshold of acceptability.

We have analyzed compression within (intra-step) and across (inter-step) time steps. Our experiments show that inter-step compression is more profitable than intra-step compression, although the former has additional space overheads. We have applied lossless and lossy compression techniques of message compression to a complex computational code, viz., CAM, as we found that the scalability of CAM is limited by the communication performance due to communication of large messages. Lossless compression resulted in a message size reduction of about 14.8% and a consequent gain of 18% improvement in scalability from 13.53 to 15.99 for 32 processors.

We have also used lossy compression of messages and used the method of Rosinski and Williamson [11] to determine the threshold of acceptability for information loss. Using lossy compression schemes reduces message size by 50% with very low overhead for compression and decompression (0.15%). The lossy compression schemes improve the speedup to 20.78 for 32 processes. Further, the lossy compression schemes bring higher benefits for larger number of processes, increasing the speedup from 15.6 to 30.77 on 64 processes.

The improvements in scalability is specific to a fixed stencil of CAM. We have not looked at varying the computational and communication stencils as done by Worley [14]. Comparison with his work shows that the configuration used by us is in the optimal range for most computational platforms. Since our method is complementary to his work, we believe that the gains obtained by better configurations can easily be enhanced by message compression especially on a cluster such as ours, which has slow interconnects. In conclusion, we would also like to highlight that for other computational codes in which messages vary even less between iterations than in CAM, lossless compression could be a very promising technique for enhancing scalability.

In the future, we would like to study the effect of performing the data compression and decompression on the programmable processor available on the network interface, so that these overheads can be eliminated from the host processor. We would also like to investigate the performance issues when there are more running processes on each node. Current studies use datasets of T42 resolution. We would like to look at the scalability at T85 resolution, which has a finer grid size.

Acknowledgements

The authors thank the reviewers for the their detailed and insightful comments which helped to improve the presentation of the paper. We acknowledge the Supercomputer Education and Research Centre (SERC), Indian Institute of Science, for providing the computing facilities used in this work. The authors acknowledge the research funding received from Intel Corporation which partly supported this work.

References

[1] A.R. Alameldeen, Using Compression to Improve Chip Multiprocessor Performance, Ph.D. Thesis, Computer Sciences Department, University of Wisconsin Madison, 2006.

[2] A.R. Alameldeen, D.A. Wood, Adaptive cache compression for high-performance processors, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, June 2004, pp. 19–23.

[3] A.R. Alameldeen, D.A. Wood, Frequent Pattern Compression: A Significance-based Compression Scheme for L2 Caches, Technical Report 1500, Computer Sciences Department, University of Wisconsin Madison, April 2004.

[4] M. Burtscher, I. Ganusov, S.J. Jackson, J. Ke, P. Ratanaworabhan, N.B. Sam, The VPC trace-compression algorithms, IEEE Transactions on Computers 54 (11) (2005) 1329–1344.

[5] W.D. Collins, P.J. Rasch, B.A. Boville, J.J. Hack, J.R. McCaa, D.L. Williamson, J.T. Kiehl, B. Briegleb, Description of the NCAR Community Atmosphere Model (CAM3.0), NCAR Report NCAR/Tn-464 + STR, NCAR, Boulder, 2004.

[6] Gzip Home Page <http://www.gzip.org>/.

[7] S.W. Hammond, R.D. Loft, J.M. Dennis, R.K. Sato, Implementation and performance issues of a massively parallel atmospheric model, Parallel Computing 21 (10) (1995) 1593–1619.

[8] J. Ke, M. Burtscher, E. Speight, Runtime compression of MPI messages to improve the performance and scalability of parallel applications, in: Proceedings of the Supercomputing, November 2004.

[9] MPI Forum, MPI: a message-passing interface standard, The International Journal of Supercomputer Applications and High Performance Computing, 8(3/4) (1994) 165–414.

[10] Port validation for CAM, ⟨http://www.ccsm.ucar.edu/models/atm-cam/port/⟩.
[11] J.M. Rosinski, D.L. Willliamson, The accumulation of rounding errors and port validation for global atmospheric models, SIAM Journal of Scientific Computing 18 (2) (1997) 552–564.
[12] R.H. Reussner, P. Sanders, L. Prechelt, M. Muller, SKaMPI: a detailed, accurate MPI benchmark, in: Proceedings of the 5th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, September 1998.
[13] T.A. Welch, A technique for high performance data compression, IEEE Computer 17 (6) (1984) 8–19.
[14] P.H. Worley, Benchmarking using the community atmospheric model, in: Proceedings of the 2006 SPEC Benchmark Workshop.
[15] T. Worsch, R. Reussner, W. Augustin, On benchmarking collective MPI operations, in: Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, September 2002.
[16] C. Jablonowski, D.L. Williamson, A baroclinic instability test case for atmospheric model dynamical cores, Quarterly Journal of the Royal Meteorological Society, 132, doi: 10.1256/qj.06.n.