Implications of Reduced Communication Precision in a Collocated Discontinuous Galerkin Finite Element Framework

Marcin Rogowski  
Extreme Computing Research Center  
King Abdullah University of Science and Technology  
Thuwal, Saudi Arabia  
marcin.rogowski@kaust.edu.sa

Lisandro Dalcin  
Extreme Computing Research Center  
King Abdullah University of Science and Technology  
Thuwal, Saudi Arabia  
dalcinl@gmail.com

Matteo Parsani  
Extreme Computing Research Center  
King Abdullah University of Science and Technology  
Thuwal, Saudi Arabia  
matteo.parsani@kaust.edu.sa

David E. Keyes  
Extreme Computing Research Center  
King Abdullah University of Science and Technology  
Thuwal, Saudi Arabia  
david.keyes@kaust.edu.sa

Abstract—Compute capability of high-performance hardware has been growing at immense rates, increasing over 130x in the last decade. Communication bandwidth, however, only grew by a factor of 6x in the same time, leading to a significant decrease in the byte-to-flop metric. This trend leads us to the situation where, in many cases, computation is virtually free, and the dominant cost of a parallel application comes from its communication cost. We expect this trend to continue and, hence, the parallel application wall-clock time to be increasingly correlated with the amount of data transferred between the nodes involved. In order to alleviate this communication bottleneck, we test several communication-reducing schemes based on the idea of using higher precision for the inner cells and lower precision communication. For every approach, we report the resulting network traffic and weigh it against the decreased accuracy. We perform our experiments in a collocated Discontinuous Galerkin finite element method framework (DG-FEM) applied in Computational Fluid Dynamics (CFD). First, we present a parametric study using the method of manufactured solutions on a 3D compressible Navier–Stokes supersonic cube. Using this method allows us to quantify communication reducing schemes’ impact on the error in test cases representing a range of solution polynomial degrees and problem sizes. Finally, we verify the findings on a full-scale CFD problem, flow around the delta wing, and report on methods’ consistency as the number of processes and the number of halo elements change.

Index Terms—communication precision, single-precision, MPI, halo exchange, Discontinuous Galerkin, finite element method, Computational Fluid Dynamics, High-Performance Computing

I. INTRODUCTION

A. Communication-to-computation ratio

In recent years, High-Performance Computing (HPC) has seen large investments as a part of US, European, Japanese, and Chinese exascale computing programs. As a result, the computational capability of hardware has grown immensely; between 2010 and 2020, the average peak performance of top 10 ranked supercomputers grew as much as 137×. However, the node bandwidth has lagged behind and increased only 6.1× in the same timeframe. In particular, 2018 marks the beginning of a new era that saw the advent of a drastic architectural change of supercomputers: Summit replaced Sunway TaihuLight as the #1 supercomputer and hence, GPUs displaced manycore CPUs. At that moment, the year-to-year performance has increased by 60%, but the byte-to-flop ratio, measuring the communication performance, has decreased as much as 8× [1].

In the other segment of the market, HPC cloud emerged, as all the major Cloud Service Providers (CSPs) try to attract HPC applications to their solutions. HPC cloud has already displaced some of the on-premise HPC solutions, and it is expected to grow at a compound annual growth rate of over 2.5× of that of on-premise HPC spending during 2019-2024 timeframe [2]. Even though there are TOP500-ranked supercomputers operated by CSPs, the cloud’s main advantage comes from its pooling of geographically distributed resources. This property not only limits the available bandwidth between nodes but also makes it varied across nodes, even if only due to the distance. In one notable example perfectly illustrating this characteristic, [3] pooled resources from many geographic regions across three service providers and performed a photon propagation simulation on a peak of over 51,000 GPUs. While in this particular work, the application seems to be embarrassingly parallel and the communication is minimal, it is evident how increasing communication or synchronization requirements would greatly inhibit the application’s performance.

We expect both trends to continue - byte-to-flop ratio decreasing on top supercomputers and the ongoing shift to cloud-based solutions. With those assumptions, it can be expected
that communication will increasingly be the bottleneck and a problem gaining significance for large-scale simulations. Therefore, in this paper, rather than report the commonly used wall-time of the application, we focus on the total number of bytes sent through the Message Passing Interface (MPI). We use this approach to emphasize the cost of communication and shift the focus away from the efficiency of the implementation, hence allowing us to test more strategies even if with unnecessary computational overhead.

B. Communication costs in CFD

As in many parallel scientific applications, also in our CFD framework, the great majority of the communicated data volume comes from the halo exchange. This exchange of values on the interfaces between the neighboring cells allocated to different processes is necessary due to domain partitioning in parallel applications. For each such cell, every unknown, typically represented with double-precision floating-point, has to be communicated over the network. In the case of the compressible Navier–Stokes equations, the gradients of the variables also have to be transmitted. Therefore, for a three-dimensional problem, the volume of exchanged data quadruples, exacerbating the communication bottleneck. It can also be noted here that, unlike in high-order finite-difference quadruples, exacerbating the communication bottleneck. It can also be noted here that, unlike in high-order finite-difference methods, in high-order DG-FEM, the width of the halo does not grow as the order of the scheme increases.

In our study, we start by analyzing the $L^2$ error of the density field ($\rho$) as we change the solution polynomial degree ($p$) and problem dimensions ($N$) using a supersonic manufactured solution with the default scheme, using double-precision floating-point numbers uniformly (FP64). We analyze the results as a function of bytes transferred through the network. Then, we repeat the same experiment but using single-precision floating-point numbers (FP32). After that, we evaluate mixed schemes – first, we leave the application in FP64 but force the halo communication to be in FP32, halving the data traffic; then, we again use FP64 for the inner elements but reduce the data traffic by using half-precision floating-point values (FP16).

To further explore the mixed schemes’ behavior, we look at the FP64+FP32 approach’s effects on consistency and determinism by varying the number of processes used and therefore changing the number of elements communicated in a different precision. Finally, we validate the results on a realistic test case simulating the flow around the delta wing based on a NASA wing geometry.

II. RELATED WORK

Communication challenges have been well-recognized and studied in fields such as deep neural networks. Due to the nature of the application (stochastic training), it has been accepted that small errors introduced by lossy compression do not interfere with convergence. Reference [4] presents a comprehensive survey of compression methods.

In scientific computing, however, high precision is typically considered a necessity. This notion, combined with high-speed interconnects used in HPC centers running those applications, resulted in relatively little prior work addressing the communication requirements in this field.

Reference [5] introduces transprecision. The authors use 16-, 21-, 32-, and 64–bit floating-point numbers in different parts of their low-order finite-element solver. They report a 1/4 reduction in internode communication as they use half-precision for communication in parts of their solver. The same solution was obtained with less than a 5% difference in the number of iterations compared to single-precision communication.

More work has been published on the topic of MPI compression. A survey of mixed precision methods [6, Section 3] briefly mentions communication compression. The authors present preliminary results using zfp compression in MPI’s All2All and report a speedup of over 2×. They are especially optimistic about the possibility of compression on GPU and overlapping of compression and decompression with the communication. Reference [7] uses a compression technique using a predictor based on earlier message contents. The resulting lossless message compression yielded speedup on a 100 Mbit/s Ethernet network that authors evaluated. The authors of [8] integrated five compression algorithms into MPI and reported improved scalability of the application. In a follow-up publication, [9] turns the compression on and off adaptively and chooses the most suitable compression algorithm based on the characteristics of each message. In both [8] and [9], however, the authors limit their study to lossless compression.

In work the most similar to that presented in this paper, [10] evaluates the use of lossless and lossy compression within the Community Atmospheric Model. The authors report 66.2% message size reduction and a speedup of over 20×. However, it has to be noted that the tests were performed on a relatively slow, by today’s standards, 1 Gbit/s interconnect, which allowed for such an improvement. The authors found that the information loss due to compression was within acceptable limits.

III. BACKGROUND

A. Compressible Euler and Navier–Stokes Equations

The compressible Navier–Stokes equations in Cartesian coordinates read

$$\frac{\partial \mathbf{q}}{\partial t} + \sum_{m=1}^{3} \frac{\partial \mathbf{f}_{x_m}^{I}}{\partial x_m} = \sum_{m=1}^{3} \frac{\partial \mathbf{f}_{x_m}^{V}}{\partial x_m}, \quad \forall (x_1, x_2, x_3) \in \Omega,$$

$$\mathbf{q}(x_1, x_2, x_3, t) = g^{(B)}(x_1, x_2, x_3, t), \quad \forall (x_1, x_2, x_3) \in \Gamma,$$

$$\mathbf{q}(x_1, x_2, x_3, 0) = g^{(0)}(x_1, x_2, x_3), \quad \forall (x_1, x_2, x_3) \in \Omega,$$

where $t \geq 0$, $\mathbf{q}$ are the conserved variables, $\mathbf{f}_{x_m}^{I}$ are the inviscid fluxes, and $\mathbf{f}_{x_m}^{V}$ are the viscous fluxes. Herein, to close the system of equations (1), we use the perfect gas model.

B. Supersonic manufactured solution

We use the method of manufactured solutions [11] to evaluate the error versus the communicated bytes for a supersonic
compressible viscous flow. To do so, we consider a cubic domain $\Omega = [0,1]^3$ with the following manufactured solution

$$
\begin{align*}
\rho &= \rho^0 + \rho^1 \sin \left( \frac{\pi}{L} x \right) + \rho^2 \cos \left( \frac{2\pi}{L} x \right) + \rho^3 \sin \left( \frac{3\pi}{L} x \right), \\
\vec{U}_1 &= \vec{U}_1^0 + \vec{U}_1^1 \sin \left( \frac{\pi}{L} x \right) + \vec{U}_1^2 \cos \left( \frac{2\pi}{L} x \right) + \vec{U}_1^3 \sin \left( \frac{3\pi}{L} x \right), \\
\vec{U}_2 &= \vec{U}_2^0 + \vec{U}_2^1 \cos \left( \frac{\pi}{L} x \right) + \vec{U}_2^2 \sin \left( \frac{2\pi}{L} x \right) + \vec{U}_2^3 \cos \left( \frac{3\pi}{L} x \right), \\
\vec{U}_3 &= \vec{U}_3^0 + \vec{U}_3^1 \sin \left( \frac{\pi}{L} x \right) + \vec{U}_3^2 \cos \left( \frac{2\pi}{L} x \right) + \vec{U}_3^3 \sin \left( \frac{3\pi}{L} x \right), \\
\vec{P} &= \vec{P}_0 + \vec{P}_1 \sin \left( \frac{\pi}{L} x \right) + \vec{P}_2 \cos \left( \frac{2\pi}{L} x \right) + \vec{P}_3 \sin \left( \frac{3\pi}{L} x \right),
\end{align*}
$$

where $\rho$ denotes the density, $\vec{U} = [\vec{U}_1, \vec{U}_2, \vec{U}_3]^T$ is the velocity vector, $\vec{P}$ is the thermodynamic pressure, and $L$ is the length of the side of the cubic domain.

### C. Flow around the delta wing

A delta wing model was constructed based on a NASA wing geometry for the Second International Vortex Flow Experiment (VFE-2). We consider the flow past a rounded leading edge with a medium radius ($r_{LE}/c = 0.0015$); see Fig. 1. The simulations are carried out for an angle of attack of $\text{AoA} = 13^\circ$, a Mach number $Ma = 0.07$ and a Reynolds number of $Re = 10^6$, based on the mean aerodynamic chord.

The grid consists of $\approx 9.209 \times 10^4$ hexahedral cells. The assigned solution polynomial degree is $p = 5$ for the region surrounding the wing and its support, and $p = 2$ in the farfield. Intermediate cells are assigned $p = 3$. Overall, the computations are carried out for $\approx 17.063 \times 10^6$ Legendre–Gauss–Lobatto nodes.

We compare the distribution of the pressure coefficient, $C_p$, computed numerically with the experimental data reported in [12].

![Fig. 1: Average flow field past the 65° swept delta wing: streamlines (left) and mean axial velocity (right) at selected cross sections. Wing surface is colored based on the average pressure coefficient, $C_p$.](image)

### IV. METHODOLOGY

#### A. Precision modification

As we aim to test the effects of lowering the precision of communicated data on the error obtained for each test case, it is important to distinguish between two types of elements:

- Inner elements – where the entire data is owned by a given process and no communication is required at interfaces.

- Halo elements – where the neighboring element is owned by another process and communication is necessary at the interface. For each halo element, we need to communicate 5 unknowns and their gradients in 3 dimensions, for a total of 20 variables.

Our methods treat those elements in different ways:

- Double-precision – baseline; all the computation and communication is performed in FP64.

- Single-precision – all the computation and communication is performed in FP32.

- Double-precision with single-precision halo exchange – all the computations are performed in FP64, inner elements remain in FP64, but halo elements are converted to FP32 buffer before being communicated. As after receiving they are multiplied with FP64 values, they are effectively being cast back to FP64.

- Double-precision with half-precision halo exchange – all the computations are performed in FP64, inner elements remain in FP64, but halo elements are converted to FP16 and stored in a short int buffer before being communicated. On the receiving end, they are converted to FP32 and multiplied with FP64 values. We used the IEEE 754 half-precision binary floating-point format with a 5-bit exponent and a 10-bit significant.

In this study, we treat the unknowns and their gradients in the same manner, i.e., use the same precision for both.

#### B. Test case setup

For the central part of our study, we use the supersonic manufactured solution test case described in Section III-B. This problem allows us to test many configurations easily, and thanks to the method of manufactured solutions, we can reliably calculate the error. The configurations we used are generally the combination of:

- Problem dimensions, $N$ – describing the number of elements per side of a cube. Here, we always use $N^3$ elements and vary $N$ between 4 and 64.

- Solution polynomial degree, $p$ – defining the number of Legendre-Gauss-Lobatto nodes per element ($(p+1)^3$ nodes per element). We use $p = 1, 2, 3, 7$. Degrees of $p = 1, 2, 3$ are chosen as typical of high-order schemes in CFD with the current computational technology. The solution polynomial degree $p = 7$ is selected as a very high variant, which additionally results in a number of solution nodes well-aligned with cache lines.

As the benefits of reduced communication will be the most visible at scale, we use an approach similar to weak scaling to determine the number of processes on which to run each case. For a three-dimensional problem, the number of degrees of freedom (DOF) is calculated as $N^3 \times (p+1)^3$, and we use a constant load of 256 DOF per process (in this case, equivalent to a CPU core). We have then eliminated configurations that would result in using more than 65,536 CPU cores ($64^3$ problem at $p = 7$ results in need for 524,288 cores at 512 DOF/process.}
C. Reporting

As we motivated in the introduction, we focus on the volume of data being moved over the network. To collect the statistics, we used CrayPat from the Cray Developer Toolkit (CDT) 20.06, which allowed us to intercept all the MPI calls. We use the “MPI Message Bytes” metric, which includes all point-to-point and collective communication. Since it describes the average number of bytes communicated per process, we multiply it by the number of processes used in a particular run. Our modifications do not influence collective communication, and our measurements show that point-to-point communication and halo exchange, in particular, accounts for almost all the data being exchanged (by volume). As the number of bytes exchanged is a deterministic measure and is not subject to the operating system or network noise, we can base our statistics on a single run for each configuration, and no averaging is necessary.

To evaluate the impact of accuracy changes on the problem being solved, we use the $L^2$ error of the density field ($\rho$). Note that we consider the $L^2$ norm because the solver used in this work is provably non-linearly stable in this norm [13]. Most of the plots presented throughout this paper show the total number of bytes transferred using MPI on the X-axis and $L^2(\rho)$ on the Y-axis.

D. Software and hardware

The experiments reported in this paper are performed using the entropy-stable collocated Discontinuous Galerkin algorithm implemented in the $hp$-adaptive, unstructured, curvilinear grid framework – SSDC [13]. SSDC’s core algorithms are developed on top of the Portable and Extensible Toolkit for Scientific computing (PETSc) [14]. SSDC is used regularly to perform large-scale simulations using Shaheen II, a Cray XC40 supercomputer hosted by King Abdullah University of Science and Technology (KAUST) [15]. All the results obtained for this paper were also performed on this 6,174-node supercomputer. Each of its nodes contains 2 Intel Haswell (Xeon E5-2698v3) CPUs with 16 cores each. Shaheen’s nodes are connected using a Dragonfly network topology. Only MPI was used for parallelization with no threading and 32 MPI processes per node. Hyper-threading was turned off. Nodes were assigned by the scheduler in an exclusive mode, but the network was shared with other users.

V. RESULTS

A. Double-precision

As a baseline, we perform a modified experiment of [16, Section 4.1] described in Section IV-B and show bytes transferred vs. the $L^2$ error on the density field. The results are shown in Fig. 2. We observe that, very consistently, increasing the solution polynomial degree ($p$) and increasing the problem dimensions ($N$) decreases the error ($L^2(\rho)$). Increasing either $p$ and $N$ also has the consequence of increasing the required network traffic, and implicitly - the number of cores used for the computation. The results we obtained are very similar to those of [16, Section 4.1], showing a strong correlation between the network traffic and wall-clock time. Once $L^2(\rho)$ reaches approximately $2 \times 10^{-14}$ (see horizontal dashed line in Fig. 2), further refining of the domain fails to decrease the error.

Fig. 2: Double-precision. The number on top of each point is the number of elements per side of the cube; only powers of 2 are annotated for readability.

B. Single-precision

In our first experiment, we used single precision in the entire solver. The results are presented in Fig. 3. Remarkably, for many configurations using FP32 did not affect the solution accuracy at all. The line representing $p = 1$ at FP32 is shifted to the left from FP64 (parallel dashed line) – showing a $2 \times$ reduction in network traffic while keeping the same error. We can notice the same trend for some of the higher-order configurations ($p = 2$, $N \leq 28$ and $p = 3$ $N \leq 8$); however, once the observed error decreases below approximately $1.5 \times 10^{-5}$ (denoted with a horizontal dashed line in the figure), the FP32 scheme fails to improve the results further.

Fig. 3: Single-precision. The number on top of each point is the number of elements per side of the cube; only powers of 2 are annotated for readability.
C. Double-precision with single-precision halo exchange

Further, we used FP64 for inner elements, but halo elements are copied to an FP32 buffer before being communicated. After receiving, they are multiplied with FP64 values and hence, cast back to FP64. In the results presented in Fig. 4, we can see that this method allowed us to achieve a much lower error than the uniform application of FP32. In fact, we see a virtually identical error to that of FP64 down to $1.5 \times 10^{-7}$. This represents two orders of magnitude improvement over the FP32-only scheme. The communicated data volume is expected to be minimally higher, as this time, some data – for example, in reduction operations – is still transferred in FP64. However, the communication is so dominated by the halo exchange that the improvement is measured at $1.998 \times$. Therefore, the FP64-FP32 scheme allows to reach much lower error than using FP32 uniformly while preserving all its benefits of reducing network traffic.

D. Double-precision with half-precision halo exchange

In the last, most communication-austere experiment, we used FP64 for inner elements, but halo elements were converted to FP16 according to IEEE 754 format and stored as short int before sending. This further halved the resulting network traffic as shown in Fig. 5; however, for a vast majority of problems, the resulting error was too high. We identified two cases ($p = 1$, $N = 4^3$ and $p = 1$, $N = 8^3$), where using such a scheme resulted in the same error as that of higher precision methods. Once the error decreased below $8 \times 10^{-3}$, the FP64+FP16 scheme did not produce satisfying results.

E. Consistency - the effect of varying the number of processes

The number of domains is determined by the number of processes assigned to the problem. As some of our methods lower the precision at the domain interfaces, it might prove troublesome – as the locations and the number of halo cells will depend not only on the problem but also the number of CPU cores used to solve it. To verify whether this will pose problems to reproducibility and determinism, we repeated the experiment communicating the halo in FP32 (Section V-C) while varying the number of processes. Rather than always using 256 DOF per process, we use 256, 512, 1,024, 2,048, 4,096, and 8,192. As illustrated in Fig. 6, the results are consistent, and the $L^2$ error of a given $p, N$ configuration is stable regardless of the number of processes assigned. All the significant differences we observed are below the $1.5 \times 10^{-7}$ $L^2(\varrho)$ line (compare with Fig. 4), where the method failed to decrease the error regardless.

VI. RESULT VALIDATION

In order to validate the obtained results, we used the described schemes to simulate the flow around a delta wing, which is a realistic test case described in Section III-C. The results obtained using SSDC have been shown to match the observed data closely [13], and in this paper, we focus on
the comparison between different schemes. We analyze the average pressure coefficient, $C_p$, measured on the surface of the wing. Fig. 7 shows the difference between FP64, FP32 $^1$, and FP64 with FP32 halo schemes. From the very beginning, the norm of the residual of the FP64+FP16 scheme was increasing, and we ruled it out. As it can be seen in the figure, the results obtained using the FP64+FP32 approach are especially promising. The largest measured absolute $C_p$ error is $7 \times 10^{-3}$, and the vast majority of differences are below 0.1%. The FP32 approach led to larger localized differences, and its error is typically within 5%.

Fig. 7: Pressure coefficient ($C_p$) difference on the surface of the delta wing.

VII. Future work

While we consider the obtained results promising, there are plenty of other directions to consider. For simple casting to lower precision, statistical analysis of the communicated data is important. A better understanding of the range of values being sent would allow tailoring the data type to the purpose. Especially in precision lower than FP32, there are multiple choices. For example, Google’s Brain Floating Point Format ($bfloat16$: 8-bit exponent, 7 bits fraction) aims to maintain the range of FP32 (same 8-bit exponent) but sacrifices the accuracy. As we only use lowered precision for storage, but the computations are still performed on CPU’s FP64 execution units, we can also use less common floating-point variations, such as a 19-bit TensorFloat-32 (8-bit exponent, 10-bit fraction). Another option is to use a custom, fit for purpose, format like the authors of [5], who, in parts of their code, used 21-bit floating-point values to store three such values within a single FP64 variable.

We are not limited to fixed-storage floating-points, either. If we allow for a greater overhead, many options in lossless and lossy compression become available. The compression and decompression overhead and the complexity of having to send the data of unknown – and changing – size will allow achieving a more significant reduction in required storage per value. In one example, [6, Section 3] reported preliminary results showing that, in their application, $zfp$ allows for FP32 precision with FP16 storage.

In another possible extension, it can be noted that the communicated data does not have to be compressed uniformly. In our particular application, the variables and their gradients could be treated using different schemes. More sophisticated approaches like varying the compression method with time or space are also possible and interesting directions.

Having drafted possible additional compression schemes, we would also like to point out possible enhancements in methodology. Investigation on a range of test cases representing different problems is necessary to confirm obtained results. Once several promising compression methods are identified, they should be implemented in an efficient manner, and wall-clock time, as the ultimate metric, should be analyzed.

VIII. Conclusions

In this paper, we have investigated the effects of lowering communication precision on the volume of data transferred over the network and the resulting error. We report on the results of using a double-precision scheme and convert it to single-precision uniformly. Then, we analyze mixed methods – using double-precision for inner cells with single-precision halo communication and double-precision mixed with half-precision communication.

Our experiments are performed in a collocated Discontinuous Galerkin finite element method framework, and the problem we used for the core part of the study is supersonic compressible viscous flow. Using it with the method of manufactured solutions allowed us to analyze the schemes’ performance on various problem sizes and solution polynomial degrees. We identify scenarios in which each scheme is beneficial and those in which the resulting error no longer improves. Our results show that in this particular problem, the FP64 scheme allows us to achieve error as low as $\approx 2 \times 10^{-14}$. FP32 halves the communication cost but works well only if the resulting error is $\approx 1.5 \times 10^{-5}$ and higher. Mixing FP64 with FP32 halo retains the communication-reducing benefits of FP32 and improves its minimum error by two orders of magnitude, to $\approx 1.5 \times 10^{-7}$. Finally, mixing FP64 with FP16 in the halo only requires communicating the quarter of the data needed by the FP64 approach but performs satisfactorily only if the resulting error is not lower than $\approx 8 \times 10^{-3}$.

We make no claim that our findings are general for DG-FEM. However, our results indicate that lower precision can be selectively employed in a representative case without any accuracy penalty. Ultimately the decision regarding the number of elements used, solution polynomial degree, as well as precision used should be made based on a particular problem and the magnitude of the allowed error.

\(^1\)As even the required timestep size was below the FP32 epsilon, only variable and gradient arrays were truncated to FP32 in this experiment.
ACKNOWLEDGMENT

The research reported in this paper was funded by King Abdullah University of Science and Technology. We are thankful to the Supercomputing Laboratory and the Extreme Computing Research Center at King Abdullah University of Science and Technology for their computing resources.

REFERENCES


