Using neural networks to reduce communication in numerical solution of partial differential equations

Laurent White
Advanced Micro Devices, Inc.
laurent.white@amd.com

Ganesh Dasika
Advanced Micro Devices, Inc.
ganesh.dasika@amd.com

Saketh Rama
Advanced Micro Devices, Inc. *
rama@seas.harvard.edu

Abstract

High-performance computing (HPC) applications are frequently communication-bound and so are unable to take advantage of the full extent of compute resources available on a node. Examples abound in scientific computing, where large-scale partial differential equations (PDEs) are solved on hundreds to thousands of nodes. The vast majority of these problems rely on mesh-based discretization techniques and on the calculation of fluxes across element boundaries. The mathematical expression for those fluxes is based on data that is available in the local memory and on neighboring data transferred from another compute node. That data transfer can account for a significant percentage of the simulation time and energy consumption. We present algorithmic approaches for replacing data transfers with local computations, potentially leading to a reduction in simulation cost and avenues for kernel acceleration that would otherwise not be worthwhile. The communication cost can be reduced by up to 50%, with limited impact on physical simulation accuracy.

1 Introduction

Many high-performance computing applications are often communication-bound and therefore are unable to take full advantage of the compute resources available on a node. A broad and important class of such applications includes large-scale simulations of physical phenomena (e.g., fluid dynamics [11, 6], ocean modeling [3] weather and climate modeling [1, 10, 7, 9, 2], seismic wave propagation [4, 8, 12], and molecular dynamics [5]). These applications have benefited from decades of improvement in computing performance and, more recently, have leveraged increasing amounts of parallelism to produce higher-fidelity predictions. However, increasing parallelism exacerbates the amount of data movement and it is common for these applications to operate below 10% of peak performance [8, 10, 6]. The trend towards more parallelism will only make this number worse unless algorithmic changes are made to mitigate this issue, that is, make more effective use of compute resources and reduce the communication overhead.

*Work done at AMD while on an internship.

Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS 2021).

We propose algorithmic approaches to reduce communication overhead in numerical algorithms that rely on mesh-based discretization techniques for solving partial differential equations [8, 3, 10, 12]. These mesh-based numerical algorithms are used in a wide range of applications, and the concepts presented here can be transferred to other classes of problems. In a mesh-based discretization technique, the domain of interest is divided into elements. For a given mesh, a numerical approximation can be obtained by using a discretization method such as the finite-volume method and the discontinuous-Galerkin finite-element method. One of the core computational kernels of these discretization methods is the flux calculation, which serves as the basis for the transfer of information from one element to a neighboring element. Fluxes are computed across element boundaries and their mathematical expressions depend on local data (within the current element) and neighboring data (within the neighboring element). When performing computations on mesh partitions that span multiple computing devices, some flux calculations inevitably require communication. Figure 1 illustrates a simple case with four partitions where the data of each partition would reside in the memory of a distinct computing device. For clarity, a small number of partitions is shown. Realistic computations involve thousands of partitions [12].

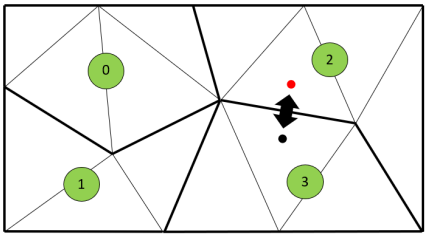

Figure 1: Mesh split into four partitions, each one made up of multiple triangular elements and delineated by a bold black line. The flux calculation (black arrow) requires values from two adjacent partitions whose data typically reside on two separate computing devices, requiring communication.

1.1 Current approach and limitations

At the beginning of each time step of the simulation, data must be transferred between neighboring partitions so that the values needed to compute the fluxes become available locally. The message-passing protocol (MPI – Message-Passing Interface) is the most widely used technique for transferring data across a cluster of computing devices [4]. When the numerical algorithm can be structured into a component that only depends on data residing in local memory and another component that depends on remote data, the communication overhead can be partially hidden by using non-blocking communication routines, which allows for overlapping computation and communication. While state-of-the-art, this approach suffers from three weaknesses: (1) Overlap is enabled only if the MPI implementation allows for it, which may not be guaranteed. (2) The cost of data transfer often remains high enough that aggressive kernel optimization is not worthwhile (there would not be enough computational work to hide the communication overhead). In other words, communication remains the bottleneck. (3) It can be difficult to restructure a computational algorithm into local and non-local components.

2 Proposed methods

We propose techniques that eliminate some of the data transfers by predicting flux values from local data. We seek to eliminate data transfers for a fraction of all time steps. This scenario is illustrated in Figure 2, where data transfer occurs every second step (b) instead of every step (a).

In Figure 2, scenario (b) shows the elimination of data transfers every second time step, which reduces the time taken to execute two time steps by reducing time spent on communication (some red boxes are eliminated) and by allowing further kernel optimization (the green boxes have shrunk) that would not be worthwhile in scenario (a) because communication remains a bottleneck.

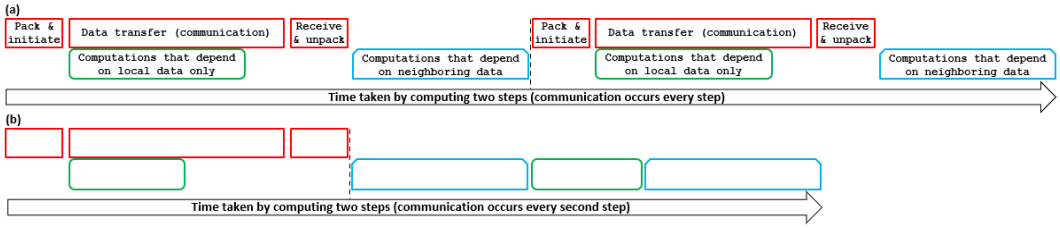

Figure 2: Wall time for computing two steps (separated by a vertical dashed line) under two scenarios. (a) Communication (red, straight-cornered boxes) occurs every step. (b) Communication is eliminated every second step. Green, rounded-cornered boxes represent computational kernels that depend on local data only. Blue, snipped-cornered boxes represent flux computations that depend on neighboring data. Boxes are illustrative and their sizes do not reflect actual wall times.

2.1 Linear extrapolation

At a given location in space, flux values are highly correlated in time, as shown in Figure 3 (Appendix). At a given time when a new flux must be calculated based on local and neighboring data, we may instead use an alternative way of predicting the flux based on a linear extrapolation from the previous two flux values. This linear extrapolation uses local data only, and therefore, does not require any data transfer.

2.2 Neural network

We now seek to increase the predictive power of our approximation by leveraging the relationship between the flux value and the local values of state variables and their spatial derivatives, in addition to only using the previous value of the flux, which was the approach outlined in the previous section. The traditional flux value, F, is obtained by evaluating a known function, f, of local and neighboring values and can be written generically as

$$F = f(\mathbf{p}^{in}, \mathbf{p}^{ex}),$$

where the superscripts in (interior) and ex (exterior) refer to local and neighboring values (communicated values), respectively, and $\mathbf{p}$ refers to vector of state variables that are solutions to the equations. Instead, we seek to approximate the flux by using only local variables, that is, we seek to approximate f as follows:

$$F \simeq \tilde{f}(\mathbf{p}^{in}, \nabla\mathbf{p}^{in}, \text{previous flux value})$$

where all function arguments are local variables (they reside in local memory). While the function f is known, the function $\tilde{f}$ is unknown and must be determined based on existing data. We propose to use a deep neural network (NN) to approximate $\tilde{f}$ based on accurate simulation data. That is, after training, the NN will take local data as input and will provide an approximation to the flux values (details in Section A.2).

3 Results

We first report on the ability of our flux approximations (linear extrapolation and NN) to provide accurate simulation results when used in the simulator as a replacement for the closed-form flux expression. We then assess the ability of our approximations in reducing communication cost when running a large-scale simulation. All results are based on a three-dimensional discontinuous finite-element acoustic wave simulator, developed internally, written in C, and enabled for large-scale MPI runs. Internode and intranode communications are both handled by MPI (instead of a hybrid OpenMP/MPI approach).

3.1 Accuracy

We consider a case of moderate size to assess the accuracy. The three-dimensional computational mesh of a unit cube is made up of 13,824 elements and all simulations are executed on 8 cores of

a workstation-class 24-core CPU. The problem geometry and initial conditions are simple enough that an analytical solution to the wave equation is known and errors can be computed (L_2 norm of pressure error at final time). In Table 1, we show the simulation errors under different scenarios. The baseline scenario relies on the closed-form flux expression that depends on local and neighboring data and is used everywhere in the mesh. In that baseline scenario, MPI communication takes place at every time step. In the extrapolation and NN scenarios, we replace the closed-form expression by the approximations when computing fluxes on inter-element faces that share two adjacent MPI partitions, which also allows for skipping MPI communication. The number between parentheses indicates the interval, in number of steps, between successive approximations (e.g., (3) means that the approximation is used every third step and communication is skipped every third step).

Table 1: Simulation errors against analytical solution under different approximation scenarios.

Scenario	**Error vs. analytical solution**
Baseline	7.54×10^{-5}
Extrapolation (3)	7.54×10^{-5}
Extrapolation (2)	NaN (unstable)
Neural network (3)	7.53×10^{-5}
Neural network (2)	7.59×10^{-5}

3.2 Communication reduction

A larger test case is considered for assessing the effect of skipping some of the MPI data transfers on overall communication cost. The domain of interest is the same unit cube. A much finer mesh of 2,097,152 elements is considered. All simulations are run on 32 nodes, each one composed of two 64-core server-class CPUs. Timing is reported in Table 2. We note 20% and 48% reductions in communication time when adopting the extrapolation and NN approximation strategies, respectively, compared with the baseline simulation strategy that uses the closed-form flux expression every time step. The increased computational time in the flux calculation (second column) for the extrapolation case stems from the requirement to save flux values after they are computed to compute the linear extrapolation in time. Approximating the flux with a NN gives rise to a large increase in computational time for the flux computation. However, the neural flux computation numbers are quite pessimistic as the implementation of the neural network does **not** currently take advantage of any hardware acceleration. Despite this increase in cost for the flux computation, the overall simulation time for the NN-based strategy is 18% lower than the baseline.

Table 2: Wall time (s) for computing fluxes (2nd column), for MPI communication (3rd column), and total simulation for various scenarios. Average of 10 runs. Standard deviation in parentheses.

Scenario	**Flux**	**Comm.**	**Total**
Baseline	7.60 (0.85)	91.75 (6.45)	131.50 (6.49)
Extrapolation (3)	9.40 (0.52)	73.02 (3.35)	116.57 (2.43)
Neural network (2)	24.80 (0.95)	48.16 (4.82)	107.86 (7.42)

4 Discussion

Through numerical experiments, we demonstrated that substituting local computations for communication led to a reduction in communication overhead by 20% and 48%, respectively, when using the extrapolation strategy every third step and the NN every second step. While the respective flux calculations are more expensive upon adopting either one of these strategies, the reduction in communication overhead more than offsets this increase and leads to faster simulation times overall. The NN-based strategy uses more input data to predict the flux, yet the data is more local in time. This locality allows for using the NN more often and allows for an even lower communication overhead than when adopting the extrapolation method. We anticipate that the increased cost associated with the neural-network inference can be decreased to a large extent through hardware acceleration, which

is ongoing work and will be reported in the future. We intend to take advantage of the flexibility of NNs to conduct a hardware-aware neural architecture search to determine the best architecture to use.

Two differences between the physics-based simulation and the local-approximation-based simulation will be difficult to overcome. First, in a physics-based simulation, every element face is associated with a unique flux value, which ensures conservation (of mass, energy, etc.). In our strategies, where local computations approximate the flux value on each side of a face shared by two MPI partitions, there is no guarantee of unique flux value (each local computation will result in slightly different flux values on each side). Second, using the local flux approximation gives simulation results that vary slightly depending on how the problem is parallelized. If the user's application demands exact (to machine precision) reproducibility of simulation results independently of how the problem is parallelized, using approximations like the ones presented here might not be appropriate.

Broader impact

The proposed algorithms constitute an important step towards addressing the increasing communication overhead in a large class of high-performance computing applications. While more work is needed to refine the methods, we believe that these communication-reduction algorithms could pave the way to a much more effective use of compute resources at the node level, leading to overall application speedup and higher energy efficiency.

References

[1] D. S. Abdi, L. C. Wilcox, T. C. Warburton, and F. X. Giraldo. A GPU-accelerated continuous and discontinuous Galerkin non-hydrostatic atmospheric model. *The International Journal of High Performance Computing Applications*, 33(1):81–109, 2019.

[2] P. Bauer, T. Quintino, N. Wedi, A. Bonanni, M. Chrust, W. Deconinck, M. Diamantakis, P. Düben, S. English, J. Flemming, P. Gillies, I. Hadade, J. Hawkes, M. Hawkins, O. Iffrig, C. Kühnlein, M. Lange, P. Lean, O. Marsden, A. Müller, S. Saarinen, D. Sarmany, M. Sleigh, S. Smart, P. Smolarkiewicz, D. Thiemert, G. Tumolo, C. Weihrauch, C. Zanna, and P. Maciel. The ECMWF Scalability Programme: Progress and Plans. Technical Memorandum 857, European Center for Medium-Range Weather Forecasts, February 2020.

[3] O. B. Fringer, M. Gerritsen, and R. L. Street. An unstructured-grid, finite-volume, nonhydrostatic parallel coastal ocean simulator. *Ocean Modeling*, 14:139–173, 2006.

[4] B. F. Jamroz and R. Flofkorn. Asynchronous communication in spectral element and discontinuous Galerkin methods for atmospheric dynamics - a case study using the High-Order Methods Modeling Environment. *Geoscientific Model Development*, 9:2881–2892, 2016.

[5] W. Jia, H. Wang, M. Chen, D. Lu, L. Lin, R. Car, W. E, and L. Zhang. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In *SC '20: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2020.

[6] C. Kato, Y. Yamade, K. Nagano, K. Kumahata, K. Minami, and T. Nishikawa. Toward realization of numerical towing-tank tests by wall-resolved large eddy simulation based on 32 billion grid finite-element computation. In *SC '20: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2020.

[7] J. F. Kelly and F. X. Giraldo. Continuous and discontinuous galerkin methods for a scalable three-dimensional nonhydrostatic atmospheric model: Limited-area mode. *Journal of Computational Physics*, 231(24):7988–8008, 2012.

[8] D. Komatitsch and J. Tromp. Spectral-element simulations of global seismic wave propagation - I. Validation. *Geophysical Journal International*, 149:390–412, 2002.

[9] S. Marras, J. F. Kelly, M. Moragues, A. Müller, M. A. Kopera, M. Vázquez, F. X. Giraldo, G. Houzeaux, and O. Jorba. A review of element-based galerkin methods for numerical weather prediction: Finite elements, spectral elements, and discontinuous galerkin. *Archives of Computational Methods in Engineering*, 23:673–722, 2016.

[10] R. Nair, H.-W. Choi, and H. Tufo. Computational aspects of a scalable high-order discontinuous Galerkin atmospheric dynamical core. *Computers & Fluids*, 38:309–319, 2009.

[11] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *SC '10: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.

[12] L. C. Wilcox, G. Stadler, C. Burstedde, and O. Ghattas. A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media. *Journal of Computational Physics*, 229:9373–9396, 2010.

A Appendix

A.1 Linear extrapolation in time of flux values

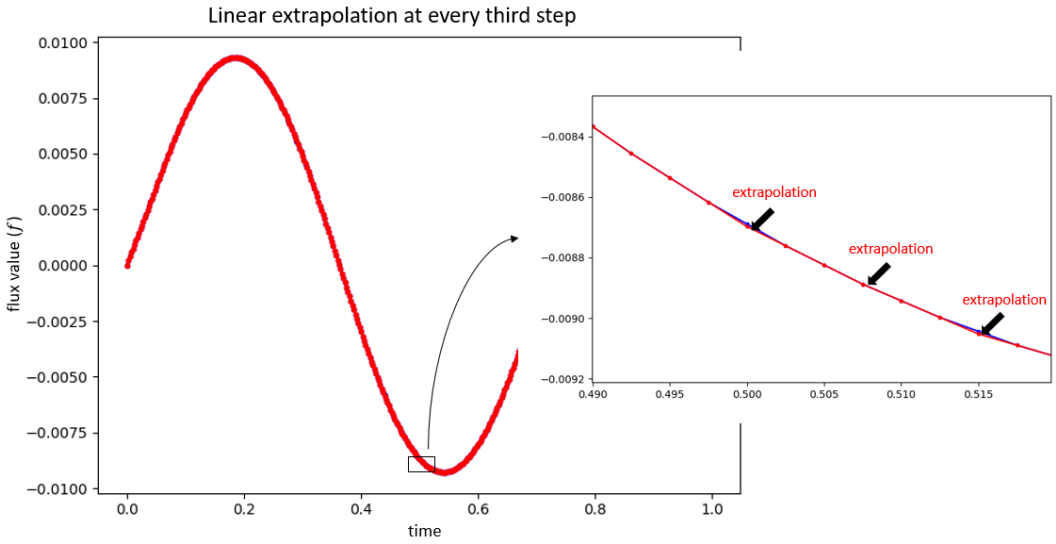

Figure 3: At a given location, flux values are highly correlated in time for smooth problems, which allows linear extrapolation (shown in red) to be a good predictor of future flux values based on the prior two values.

A.2 Neural network architecture and training

The neural network consists of 3 hidden layers of 10 units each. The leaky (slope of 0.01) rectified linear unit activation function (leaky ReLU) is used throughout. The loss function is the root mean square error between the true flux values as provided by simulation data and the predicted values. We note that variations of the chosen neural architecture could be used, which we are currently investigating. The training data is generated by running a numerical simulation for 1,000 time steps to a final time of 1 s and saving flux values every 50^{th} time step across the entire computational domain. Ninety percent of the training data is used to optimize the neural network and the remaining data is used to test it. Training was done for 100 epochs, with batch size of 1,024 and the Adam optimizer (step is 5×10^{-4}). The full dataset has 1,866,240 training samples.