SIAM J. Sci. Comput.
Vol. 42, No. 5, pp. B1302–B1327

A HOLISTIC ALGORITHMIC APPROACH TO IMPROVING ACCURACY, ROBUSTNESS, AND COMPUTATIONAL EFFICIENCY FOR ATMOSPHERIC DYNAMICS*

MATTHEW NORMAN† AND JEFFREY LARKIN‡

Abstract. Atmospheric weather and climate models must perform simulations very quickly to be useful. Therefore, modelers have traditionally focused on reducing computations as much as possible. However, in our new era of increasingly compute-capable hardware, data movement is now the prohibiting expense. This study examines the computational benefits of a new algorithmic approach to modeling atmospheric dynamics on scales relevant to weather and climate simulation. Rather than minimizing computations, this new approach considers the larger problem more holistically, including spatial accuracy, temporal accuracy, robustness (i.e., oscillations), on-node efficiency, and internode data transfers together at once. Numerical experiments demonstrate how computations can be strategically *increased* to simultaneously address each of these constraints while reducing data movement to adapt to modern accelerated hardware. The new algorithm can achieve at times up to 80% peak floating point throughput in single precision on the Nvidia Tesla V100 GPU, where the traditional approach is shown to only achieve single-digit floating point efficiency. Further, the new algorithm is twice as fast as a standard Runge–Kutta time integrator, and high-order accuracy with Weighted Essentially Non-Oscillatory (WENO) limiting came at less than 30% additional runtime cost on a GPU, thus increasing the accuracy per degree of freedom.

Key words. climate, PDEs, fluid dynamics, GPUs, WENO, high-order

AMS subject classifications. 35Q30, 68W10, 65Y20

DOI. 10.1137/19M128435X

1. Introduction.

We benefit from weather and climate simulation in many aspects of society, including flood and drought prediction, severe weather prediction, future climate projections, and sea level rise estimates. In both weather and climate contexts, atmospheric simulations must be performed quickly to be of practical use, anywhere from $100\times$ to $2,000\times$ real-time depending upon the context. Weather must simulate significant lead time with tens of ensembles to better understand inherent model uncertainty, and climate must simulate multiple campaigns of hundreds of years to properly constrain future projections.

This fast throughput constraint is the single greatest computational challenge for atmospheric modeling. Consider, for instance, that in production simulations of the Energy Exascale Earth System Model (E3SM) [7] at 28km grid spacing on 5,400 nodes of the Oak Ridge Leadership Computing Facility's (OLCF's) Titan computer, more

*Submitted to the journal's Computational Methods in Science and Engineering section August 29, 2019; accepted for publication (in revised form) July 8, 2020; published electronically October 27, 2020.
https://doi.org/10.1137/19M128435X
Funding: This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC05-00OR22725. This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy (DOE). The U.S. government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).
†Oak Ridge National Laboratory, Oak Ridge, TN 37831-6016 USA (normanmr@ornl.gov, https://mrnorman.github.io).
‡Nvidia, Santa Clara, CA 95051 USA (jlarkin@nvidia.com).

than 90% of the dynamics runtime is spent in point-to-point Message Passing Interface (MPI), and roughly 50% of the overall model runtime is spent in load imbalance and parallel overheads. Even on the Summit supercomputer, which has a better network and lower node count, the dynamics still spend half of the time in MPI waiting even using a generous MPI task layout. Note also that the spectral element atmospheric dynamical core is among the most scalable global atmospheric dynamics implementations available due to low per-time-step data transfer requirements and a large stable time step (due to the spatial operator and the large time step Runge–Kutta method) [10, 5].

Earth system simulations are traditionally significant users of supercomputing resources because when it comes to prediction and projection, resolution is crucial [19]. Severe weather occurs on scales of hundreds of meters with microphysical forcing occurring on the scale of microns. While some processes (such as microphysics) cannot be resolved and must be approximated through parameterization, there is much to be gained from higher explicit resolution. Orography, land-sea interfaces, weather fronts, monsoons, moist clouds, and differences in land surface properties all represent discontinuous or highly variable forces that need to be explicitly resolved as well as possible to achieve greater physical fidelity.

However, resolution alone is not the complete story. For any method that is higher-than-1st-order accurate, small-wavelength, high-amplitude oscillations develop in the presence of discontinuous data. This is known as Runge or Gibbs phenomenon, depending upon the spatial approximations used [13, 4]. In linear contexts, these oscillations can contaminate the solution, but in nonlinear contexts, the oscillations cause energy to accumulate at small wavelengths and cause the model to crash [29]. Further, oscillations can cause highly unphysical behavior in reactive chemistry and physics routines [17]. Numerical algorithms must resolve discontinuous or highly variable features without admitting oscillations or performing unphysical modes of mixing.

Since some means of damping must be employed to limit oscillations, limiting oscillations and achieving high spatial resolution are directly competing constraints. This is because damping, by its very nature, acts to reduce gradients in the flow, some of which are physical, and we do not want to damp. It is important to damp only what needs to be damped with a high quality limiter so that accuracy is not overly reduced.

The traditional approach to increased resolution at high throughput has been grid refinement, using algorithms with as little computation as possible. While this sufficed for past computing architectures, it is no longer an effective approach for high-resolution climate. At the very least, it is asymptotically unsustainable for all simulation efforts that require a certain throughput. Each $2\times$ grid refinement requires a $2\times$ smaller time step [3], meaning each grid point requires twice as much work per unit time. To keep the same throughput, one must strong scale to twice as many nodes to compute each time step twice as fast. In the end, this means that each $2\times$ grid refinement results in cutting the workload per node in *half*. That is assuming perfect scaling. Realistically, one ends up with less than half the per-node workload to account for suboptimal scaling. This, in turn, leads to relatively larger parallel data transfer overheads and less on-node work to feed an accelerator. Because of this, straightforward grid refinement is asymptotically unsustainable because once parallel overheads dominate, the simulation cannot proceed faster.

Parallel-in-time integrators seek to allow the model to strong scale more by parallelizing some of the calculations in the temporal dimension [8]. Current implementations, while parallel, require more passes through the data per time step, which ultimately acts to increase the total memory bandwidth requirements. Also, the speed-ups obtained tend to be low compared to the additional amount of resources required. Recently, a new and simple parallel-in-time technique has emerged that allows the multiple stages of computation to be computed concurrently [6]. One could think of these schemes as block-stepped multistage integrators that incorporate multistep information as well.

Making algorithmic choices that minimize computations works against the strengths of modern accelerators. Typical atmospheric simulations use algorithms that perform few floating point operations per data fetched from memory (this will be demonstrated in section 5). For this reason, even with all available parallelism exposed, traditional atmospheric simulation algorithms rarely if ever enjoy speed-ups over traditional CPUs beyond the increased memory bandwidth we typically see on accelerators. The goal of this study is to demonstrate that there are better algorithmic approaches for atmospheric simulation that strategically *increase* computations in a manner that significantly improves accuracy and runtime while also improving computational efficiency on accelerators beyond the memory bandwidth improvement.

In [23], an algorithm was introduced that provides arbitrarily high-order accuracy in space and time with a large stable time step. It also maintains nonoscillatory properties out to a larger time step than existing strong stability preserving temporal operators [12]. More to the point of this study, the algorithm gathers a single stencil of values and performs significant amounts of local work before writing fluxes back to global memory, playing to the strengths of modern accelerators.

2. Algorithmic choices and mathematical formulation.

2.1. The Euler equations. The Euler equations form the core of all atmospheric modeling (as well as many other fluids applications). On weather and climate scales, the compressible, nonhydrostatic form of the Euler equations must typically be used to account for the full range of relevant motions. The full equations are detailed in Appendix A, and they express the conservation of mass, momenta, and a thermodynamic variable (potential temperature in this case). While the atmosphere's dry dynamics also include some dissipation, the magnitude of dissipation in the atmosphere is relatively small, dominated by viscosity from large eddies. Also, discontinuities arise in many forms in atmospheric modeling that must remain sharply resolved regardless of dissipation. To treat atmospheric dynamics accurately, one must also be able to approximate the inviscid Euler equations accurately. Also, subgrid-scale processes are typically handled in separate code usually called the "physics" packages, which the Euler solver essentially treats as right-hand side source terms.

2.2. Temporal discretization. There are certain aspects of the Euler equations that influence the temporal operator. First, the Euler equations form a "hyperbolic" system of conservation laws, meaning they essentially describe packets of information propagating at finite speeds [18]. The speeds that emerge from a characteristic decomposition are twofold: the speed of wind and the speed of sound. There also exist gravity waves emerging from the buoyant source term and the pressure gradient in stratified flow, but they typically propagate at speeds in between the speed of wind and the speed of sound. In the global atmosphere, the maximum wind speed in jet streams is only about 4–5$\times$ smaller than the maximum speed of sound, meaning there isn't much scale separation between the two.

This fact limits the effectiveness of time-implicit (or semi-implicit) solvers in a strong scaled regime, which enjoy larger time steps than their time-explicit counterparts because they enforce global data dependence over a single time step [26, 27]. Since we care about dynamics at the scale of wind speeds, the time step cannot be increased very far past the point of resolving acoustic scales. In significantly strong-scaled regimes (particularly with climate, which must simulate at 2,000$\times$ realtime), there is rarely enough per-node work to amortize the global data movement (e.g., `MPI_AllReduce` or `MPI_AllToAll`) associated with time-implicit and spectral solvers [9], even with more advanced treatments (e.g., [21]). This is why this study chooses a time-explicit operator. Four cycles of nearest neighbor communication is faster at the largest computing scales than multiple calls to `MPI_AllReduce` (of "v-cycles" of prolongation and restriction in multigrid solvers) in the extreme scaling limit. At a certain point in strong scaling, a time-explicit solver will inevitably become more efficient for *global atmospheric* flow. Oceanic models and limited area atmospheric models can exhibit significantly greater acoustic stiffness than global atmospheric flow, and slower throughput models can allow more work per node to amortize time-implicit MPI overheads. In those applications, globally dependent methods are far more effective.

The Euler equations by their nature relate the fluid state's temporal derivative to its spatial derivatives, thus relating spatial variation and temporal variation: i.e., $\partial_t \mathbf{q} + \partial_x \mathbf{f}(\mathbf{q}) + \partial_y \mathbf{g}(\mathbf{q}) + \partial_z \mathbf{h}(\mathbf{q}) = \mathbf{s}(\mathbf{q})$, where $\mathbf{q}$ is the vector containing the fluid state, $\mathbf{f}(\mathbf{q})$, $\mathbf{g}(\mathbf{q})$, and $\mathbf{h}(\mathbf{q})$ are the vectors of fluxes in the x-, y-, and z-directions, respectively, and $\mathbf{s}$ is the vector of source terms (which in this case is just gravity). As described in more detail in [23, 31], the "ADER" (Arbitrary DERivatives in space and time) ideology is chosen to directly apply this relation of space and time to gain time derivatives of the fluid state from easily obtainable spatial derivatives. In this study, Differential Transforms (DTs) are used to transform the PDE into a simple recurrence relation to obtain these derivatives efficiently (see Appendix D). This is the same technology used in the Automatic/Algorithmic Differentiation of the ubiquitous Stochastic Gradient Decent stage of Deep Learning [32, 1].

The advantage of the ADER-DT algorithm is that it cheaply computes the temporal evolution of the fluid over the time step entirely locally based only on a small stencil of data without intermittent stages of data transfers like multistage temporal operators (e.g., Runge–Kutta) do. The ADER-DT algorithm plays to the strengths of modern accelerated hardware. Further, it maintains any nonoscillatory properties of an underlying spatial interpolation out to a large CFL value of 1 at any nonlinear temporal order of accuracy desired. By contrast, existing semidiscrete temporal operators are generally (though not always) limited to an effective CFL value of 1/2 or less to maintain nonoscillatory properties [12].

2.3. Spatial discretization. Popular spatial operators for weather and climate include element-based operators [10, 11, 22] and Finite-Volume (FV) operators [25, 30]. Element-based operators have the advantages that they are easy to map to complex geometries; they are cheaper per Degree of Freedom (DOF) in nonlimited contexts; and when appropriately designed, they have minimal parallel data transfer requirements per time step. However, they have the disadvantages that it is harder to limit oscillations on a per-DOF basis; the time step decreases rapidly with increasing order of accuracy; and existing limiting mechanisms require additional neighboring information and must usually be done after the forward step rather than in the same step [5, 14]. Also, since weather and climate atmospheric models often use orthogonal limited area projections or logically rectangular nonorthogonal grids [28, 30], the need for FV stencils for interpolation is not particularly arduous.

FV methods have the advantage that they suffer no time step reduction with increasing order of accuracy, making high-order accuracy more usable. While for some temporal operators, higher-order spatial accuracy does tend to reduce the maximum stable CFL value, for ADER-DT, it does not. To justify this, we refer the reader to the results section of this paper where we ran stably at a constant CFL value of 0.9 from 3rd-order to 9th-order spatial accuracy without any added dissipation or limiting. We do not provide a more rigorous mathematical justification for this behavior in this paper. It's worth noting that higher-order methods are more computationally dense than lower-order methods (which will be demonstrated in section 5). Also, when paired with Weighted Essentially Non-Oscillatory (WENO) limiting [20], FV methods perform the limiting at the same time as the spatial reconstruction and the ADER-DT temporal discretization [23], all based only on a small local stencil of data without intermittent data transfers to and from main memory.

WENO limiting interpolates the spatial variation within a cell using several different subsets of local stencil data and chooses the smoothest interpolations to reduce oscillations. When the data is smooth, the high-order accurate approximation is recovered, meaning WENO preferentially damps less smooth regions of the data while leaving the smoother regions undamped. The WENO implementation in [23] is more accurate than the traditional approach of [20]. Some alternative limiting approaches include hyperviscosity [5, 30] and Flux Corrected Transport [34], each of which cluster computations less densely than WENO limiting. A more visual description of WENO limiting is given in Appendix C. Regarding structured and unstructured meshes, the ease of applying WENO limiting follows the same logic as the ease of using stencil-based methods. It is easy to apply in structured meshes and is more difficult to apply in unstructured meshes (e.g., [15]).

The FV method implemented here achieves high-order accuracy by reconstructing the variation inside a single cell by constraining a polynomial to match the cell averages in the surrounding stencil of data. For instance, a 5th-order accurate method needs five values to constrain a quartic polynomial within the cell (the cell in question and two cells on either side). When WENO limiting is applied, it uses the same high-order reconstruction, but it also adds multiple lower-order reconstructions and then weights them according to how much they vary in space (to get a smoother resulting interpolation). Then, whether WENO limiting is applied or not, the resulting polynomial is sampled at GLL (Gauss–Legendre–Lobatto) points across the cell for the ADER-DT time integration to use. Finally, time-averaged flux estimates at the cell edges are computed.

The FV algorithm here is run in a dimensionally split manner, meaning each of the dimensions is handled in succession rather than all at once. This creates a 2nd-order error term in time when handled via an alternating Strang splitting, the constant of which is quite small in orthogonal coordinates. In nonorthogonal coordinates, there are other dimensionally split approaches that also result in low splitting error (e.g., [16, 2]), and often, an unsplit scheme is warranted [23]. A more formal description of the FV method is given in Appendix B.

2.4. Computational intensity. Computational intensity is best defined as the number of floating point operations performed for each movement of data. The first part of this metric is clear, but the second can have multiple contexts: moving data between nodes over network fabric, moving data to and from main system Dynamic Random Access Memory (DRAM), or moving data to and from different levels of cache. Each of these contexts of data movement is successively faster. The context of data movement this study focuses on is movement to and from DRAM, though the algorithm presented here also reduces frequency and volume of off-node data movement as well (mainly via the large time step).

To explain why this concept is so important, consider the disparity between the memory bandwidth and the computational throughput of the Nvidia Tesla V100 GPU. It can perform about 15 trillion single-precision operations each second, but it's only capable of moving about 225 billion single precision values to and from DRAM each second. Given that most arithmetic operations involve two reads and one write, that means the V100 GPU can compute about 130 times faster than it can move data. You could think of it in these terms as well: once you fetch two floating point values, you can perform 130 floating point operations on them in the time it'll take to fetch two more values. Therefore, it is imperative to do as much with moved data as you can before moving it again.

High-order accuracy and WENO limiting both significantly improve computational intensity. High-order accuracy does so because of two things. First, the computational complexity of an Nth-order-accurate spatial reconstruction (sometimes called recovery) operator in D dimensions is N^{D+1}, where $D = 1$ for this study because of the dimensional splitting. Therefore, as the order of accuracy increases, the computations increase superlinearly. Second, spatial reconstruction is done in-place on a single stencil of data and isn't interrupted by further data movement to and from DRAM. Therefore, high-order accuracy performs N^{D+1} floating point operations with only N floating point values read from DRAM or cache.

WENO limiting further improves computational intensity (by a large factor) because it also uses only N stencil values, it can be performed at the same time as high-order reconstruction without additional accesses to DRAM, and it requires additional lower-order reconstructions and expensive Total Variation estimates as well. It is shown later that WENO limiting increases the number of floating point computations by 16$\times$ at 9th-order accuracy, while requiring no additional DRAM accesses beyond the N values read in for high-order reconstruction.

2.5. Algorithmic summary. For a more complete description of the WENO-limited ADER-DT FV space-time operator, please see [23]. For each dimension, the algorithm can be summarized as follows, where N is the spatial order of accuracy, and N_T is the temporal order of accuracy ($N_T \leq N$):

For each dimension (x, y, or z):

- For each cell:
    1. Project[1] a stencil of N cell averages onto N polynomial coefficients.
    2. Perform WENO limiting on the polynomial coefficients according to the WENO algorithm in [23].
    3. Project N limited polynomial coefficients onto N_T GLL (Gauss–Legendre–Lobatto) points.
    4. Computing $N_T - 1$ time derivatives at each of the N_T GLL points using Differential Transforms of the PDE (see Appendix D).
    5. Integrate time derivatives into a time-averaged state.
    6. Store cell-edge GLL point time-averages of the state and flux vectors into global memory.

  For each cell interface:

[1]By "project" we mean to represent one set of degrees of freedom using another set of basis functions in an optimal way (e.g., Vandermonde-type interpolation)

7. Compute a single-valued numerical flux from discontinuous estimates from adjacent cells using the linear upwind Godunov state [23].

For each cell:

8. Compute the time-averaged tendency as the flux divergence over the cell.
9. Update the fluid state using the time-averaged tendency for the cell as shown below:
   x**-direction**: $\overline{q}_{n+1,i} = \overline{q}_{n,i} - \Delta t\big(\widehat{f}_{i+1/2} - \widehat{f}_{i-1/2}\big)/\Delta x$,
   y**-direction**: $\overline{q}_{n+1,j} = \overline{q}_{n,j} - \Delta t\left(\widehat{g}_{j+1/2} - \widehat{g}_{j-1/2}\right)/\Delta y$,
   z**-direction**: $\overline{q}_{n+1,k} = \overline{q}_{n,k} - \Delta t\big(\widehat{h}_{k+1/2} - \widehat{h}_{k-1/2}\big)/\Delta z + \Delta t\widehat{\overline{s}}_k$,
   where $\widehat{f}$ is the time-averaged flux, $\overline{q}$ is the cell-averaged fluid state, and $\widehat{\overline{s}}$ is the time-averaged and cell-averaged source term.

While the algorithm is complex to describe in detail (especially the DT portion), the publicly available code provides a practical means of describing the numerical approach and exactly how it is implemented.

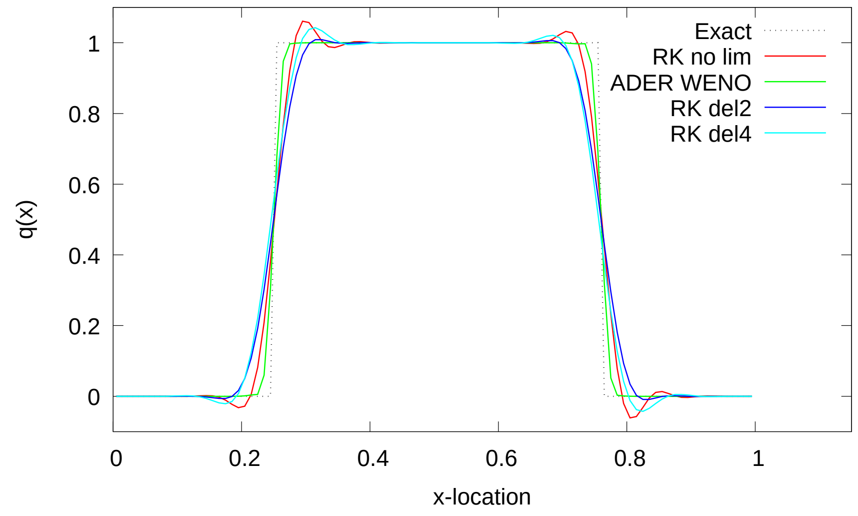
3. Improved accuracy.

It was mentioned earlier that the algorithm described here improves physical fidelity, and this section gives evidence toward that statement. Note that all examples and references in this section use the exact same ADER-DT WENO FV method presented in the last section in different contexts. It can be difficult to demonstrate accuracy in nonlinear turbulent regimes, but the properties exhibited in simpler contexts carry over into the full three-dimensional (3-D) Euler model. Given the lack of shocks in the global atmosphere, the hyperbolic dynamics are essentially transport in characteristic space. Whether other aspects of solving the Euler equations such as the Riemann solver or the gravity source term overshadow the transport properties is hard to say, but higher-order accuracy is guaranteed to improve the scheme's resolution of intracell variation of the fluid state. What makes the nonlinear flow difficult to test in a quantitative way is the fact that small differences grow rapidly in time leading to substantially different resulting states (thus the inherent limits to weather predictability).

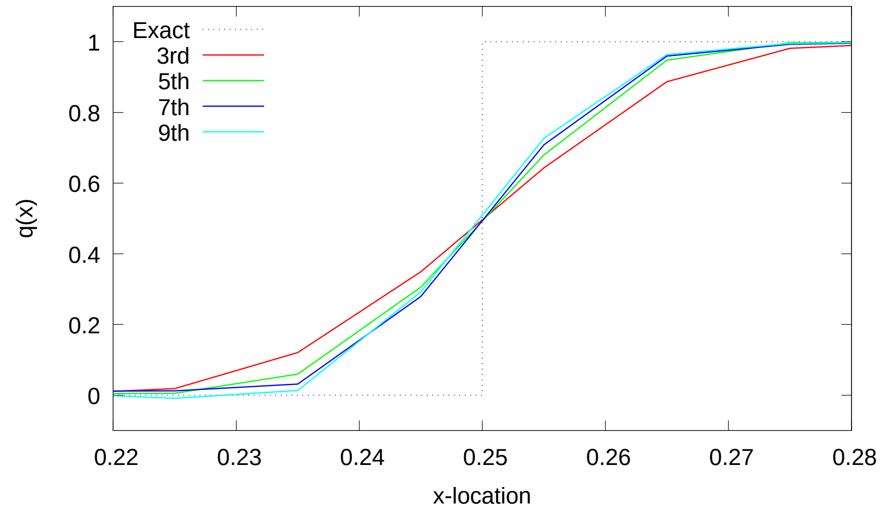
Resolution of sharply varying features is extremely important in weather and climate modeling, but that resolution must be done in such a manner that oscillations do not render the model unstable or lead to unphysical behavior in the rest of the model's physics. There are two primary ways to increase resolution: (1) use more DOFs ("h-refinement") or increase the order of accuracy per DOF ("p-refinement"). h-refinement does not improve on-node hardware utilization because the number of computations per data fetch does not change. p-refinement, however, *does* improve hardware utilization because more computation is performed per data fetch (see Table 1). The reason is that computations scale linearly with DOFs but superlinearly with order of accuracy.

Therefore, the focus here is on increasing order of accuracy and improving the limiting. But it remains to be seen whether increased order of accuracy and better limiting actually improve the solution. In fact, many believe that once limited, a higher order of accuracy gives no improvement to the solution if discontinuities are present. While one cannot experience high-order convergence at discontinuities, that is not the same thing as saying the solution cannot improve at higher-order accuracy. The constant of the error term still matters.

To demonstrate that higher-order methods do actually improve the solution when limited, the one-dimensional (1-D) uniform transport equation, $q_t + q_x = 0$, is sim-



(a) Comparison of ADER with WENO limiting ("ADER WENO") and Runge–Kutta solutions with no limiting ("RK no lim"), 2nd-order viscosity ("RK del2"), and 4th-order viscosity ("RK del4").

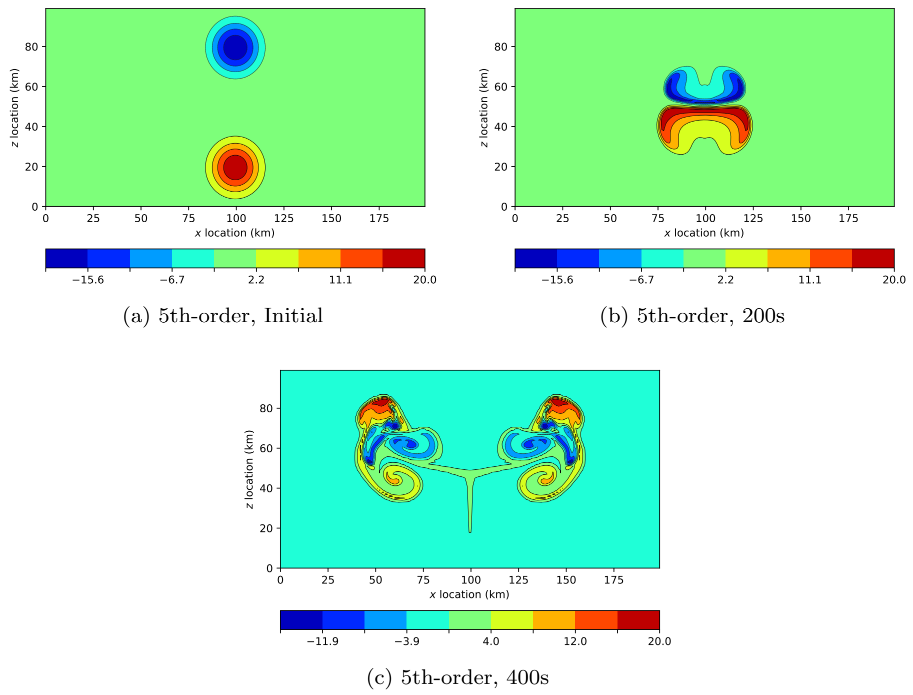


(b) Comparison of different orders of accuracy for ADER with WENO limiting zooming in on the left discontinuity.

FIG. 1. *Method comparisons for uniform advection of a square wave for one revolution with a CFL value of* 0.9 *using* 100 *cells.*

ulated with discontinuous initial data of $q_0(x) = 1$ for $0.25 < x < 0.75$ ($q_0 = 0$ otherwise) on the domain $x \in [0, 1]$ with periodic boundary conditions. The results of one revolution of transport about the domain are shown in Figure 1. In Figure 1b, it is clear that the steepness of the resolved discontinuity increases monotonically with increasing order of accuracy without admitting oscillations.

In Figure 1a, a spatially 5th-order accurate WENO-limited method with 3rd-order-accurate ADER time stepping is compared against a spatially 5th-order-accurate



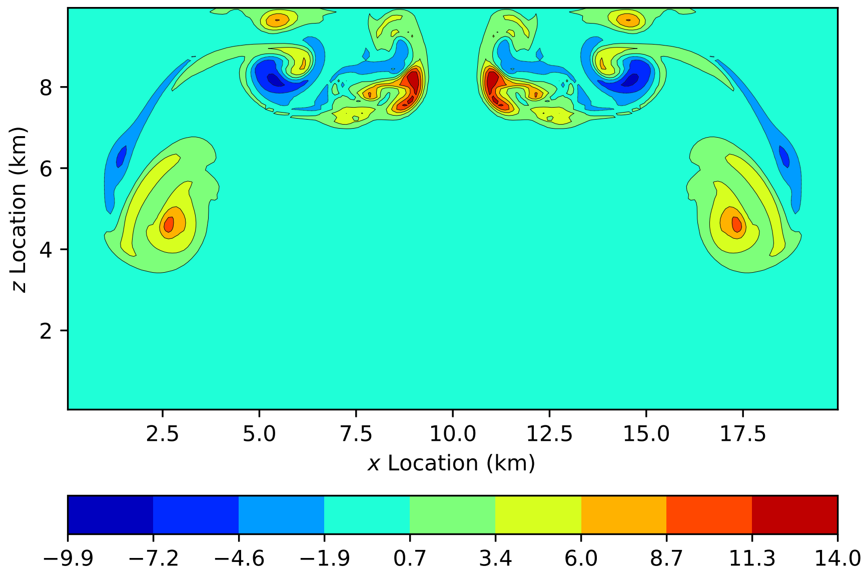
(a) 5th-order, Initial (b) 5th-order, 200s (c) 5th-order, 400s

Fig. 2. *Contours of potential temperature for colliding warm and cold thermals using ADER and WENO with a CFL value of* 0.9 *using* 200 x 100 *cells.*

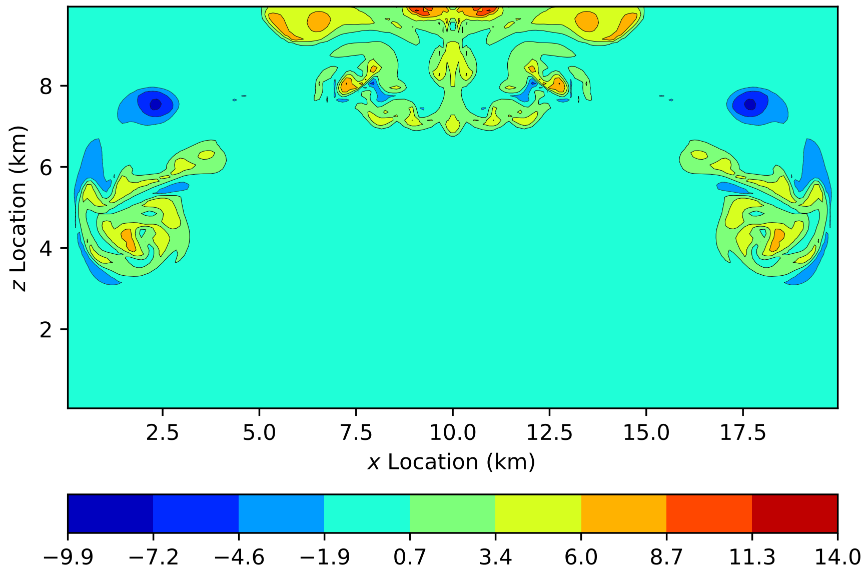
method with a 3rd-order-accurate Strong Stability Preserving Runge–Kutta (SSP-RK) time stepping [12] and no WENO limiting. The SSP-RK method uses 2nd- and 4th-order viscosity operators for damping rather than WENO limiting. It is clear from the figure that not only does the ADER WENO scheme resolve the discontinuity significantly better than the other methods, but it does so with significantly decreased oscillations.

Figure 2 shows results for a two-dimensional (2-D) version of the compressible, nonhydrostatic model simulating cold and warm thermals colliding into one another in a neutral background atmosphere. This test is useful because the flow quickly becomes discontinuous as the thermals meet, and then it becomes turbulent afterward. It shows resolved scales and resolved turbulent mixing visually. In Figure 3, the increased resolving power when increasing from 3rd-order to 9th-order accuracy is evident.

To observe how WENO limiting affects the actual resolution of the model, Figure 4 shows the Kinetic Energy (KE) spectral power density as a function of spatial wave number for the colliding thermals test case after 700s of simulation with and without WENO limiting. What this shows is essentially the amplitude of KE at different resolved wavelengths in the model. Perfect resolution would mean maintaining a roughly constant slope in KE power as the wavenumber increases if the inertial range is fully developed. Schemes will often exhibit a strong reduction from the linear slope at anywhere from $4\Delta x - 10\Delta x$. We show this plot to demonstrate that (1) our scheme has excellent resolution on the grid, and (2) the WENO limiter eliminates KE accumulation at the smallest scales without affecting any other scales appreciably. Also, consider how the kinetic energy spectra in Figure 4 remain linear until exactly



(a) 3rd-order



(b) 9th-order

FIG. 3. *Contours of potential temperature after* 700*s of simulation with colliding warm and cold thermals using ADER and WENO with a CFL value of* 0.9 *using* 200 x 100 *cells.*

the $2\Delta x$ scales. Note, however, that the noise inherent to a KE spectra plot at a single point in time makes it difficult to draw firm conclusions beyond the pile-up of energy at the $2\Delta x$ scale.

For a transport application with ADER-DT and WENO limiting in nonorthogonal spherical coordinates [23], this same algorithm with an added positivity filter was

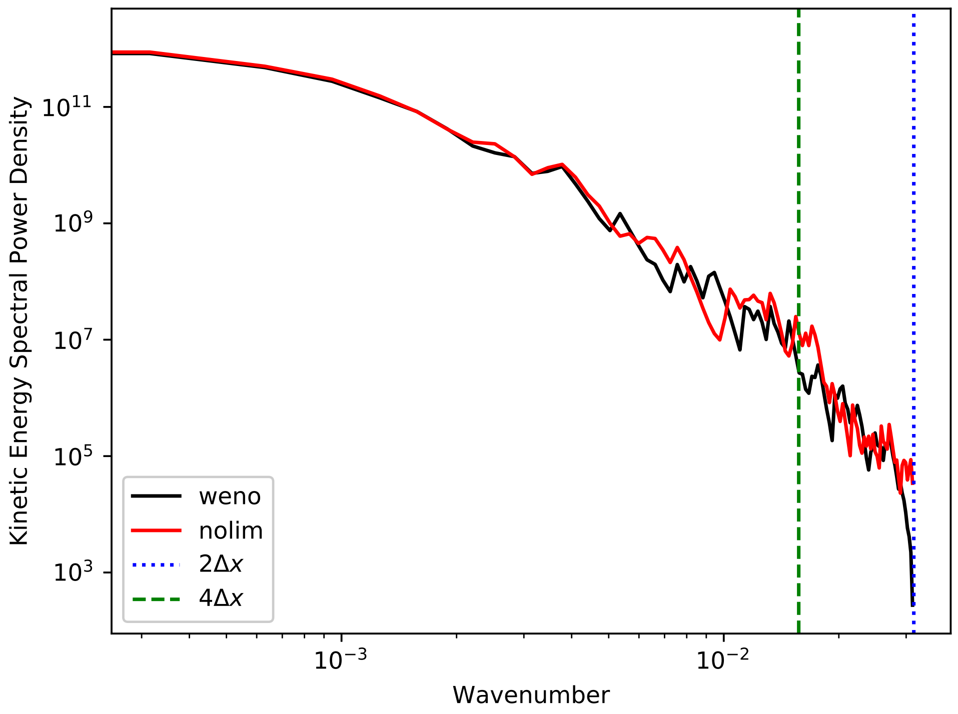


FIG. 4. *KE spectra as a function of spatial wave number after* 700*s for the colliding thermals simulation at* 5*th-order accuracy and CFL* = 0.9 *with and without WENO limiting using* 200 × 100 *cells.* $2\Delta x$ *and* $4\Delta x$ *wavelengths are marked on the plot.*

shown to effectively limit oscillations for highly complex flow fields with and without discontinuities. This algorithm preserves nonlinear relationships between multiple tracer fields well without adding unphysical mixing, a constraint critical for atmospheric chemistry reactions in particular [17]. Not only did the accuracy increase with increasing order even while limited, but WENO limiting always *increased* the accuracy compared to running without a limiter, which is quite rare for a limiter. Since limiters invoke an implicit amount of damping, they typically decrease accuracy at discontinuities in error norms compared to nonlimited methods.

This evidence demonstrates that the algorithm in this study significantly improves the physical fidelity of the model compared to the more typical, cheaper numerics long favored by atmospheric modelers. Also regarding enhanced physical fidelity with WENO limiting, consider a quote from a recent study on large eddy simulation of stratocumulus clouds: "using weighted essentially non-oscillatory (WENO) numerics for all resolved advective terms and no explicit SGS [Sub-Grid-Scale] closure consistently produces the highest-fidelity simulations" [24]. Section 5 investigates whether it also improves computational efficiency on modern hardware.

3.1. What order is best? In terms of designing an algorithm for a particular application, it is natural to consider what order of accuracy is best. Modal expansions (particularly polynomial expansions) tend to have a point of diminishing returns where increasing variation in the basis functions no longer improves the answer. Therefore, there is going to be a limit to the order of accuracy on a physical basis alone. This

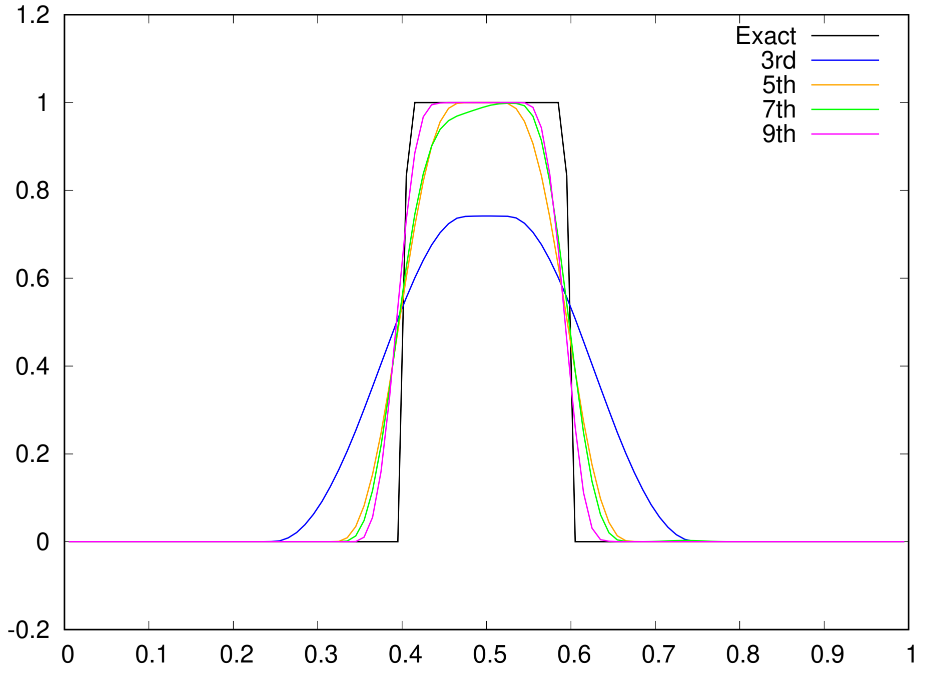


FIG. 5. *Linear transport of a square wave for* 1,000 *revolutions with* 100 *cells at a CFL value of* 0.9 *for varying orders of accuracy with ADER time stepping.*

study has shown results for orders of accuracy up to 9th-order. While the improvement in Figure 1b might seem minute between 7th- and 9th-order, consider Figure 5, which transports a square wave over a domain 1,000 times instead of just one. As the length of time integration increases, the value of high-order accuracy becomes more apparent. Clearly, simulating at 5th-order accuracy at a minimum is wise for longer simulations.

Regarding the best choice for order of accuracy, there are a number of things to consider. Once the sheer number of computations becomes large enough that overall runtime on the GPU increases unacceptably, this forms a cutoff for the order of accuracy. As will be shown in section 5, at 9th-order accuracy with WENO limiting, the overall simulation takes 30% longer than using 3rd-order accuracy with no limiting. For some applications, this may be unacceptable. Also, at 9th-order accuracy, the compute throughput on the GPU is saturated. That means any additional computations will result in a linear increase in runtime cost past 9th-order. For applications in the extreme strong scaling limit, MPI overheads consume more time, meaning there is relatively lower impact on runtime to associated with using high-order accuracy. Finally, if there is other work in the model that consumes significantly more time than the Euler solver, then it makes sense to increase the order more than you might have otherwise simply because its relative impact is going to be low.

4. Implementation details. The equations in Appendix A are implemented using the FV method described in Appendix B, specifically using the algorithm detailed in section 2.5. The algorithm is split into kernels as follows:

- *[MPI transfer of stencil data]*
- **Kernel 1**:

  - 1. Project stencil onto polynomial coefficients.
  - 2. Perform WENO limiting.
  - 3. Project polynomial onto N_T GLL points.
- **Kernel 2**:
  - 4. Compute time derivatives.
  - 5. Integrate time derivatives to time-averages.
  - 6. Store cell-edge fluxes.
- *[MPI transfer of edge flux time-averages]*
- **Kernel 3**:
  - 7. Compute single-valued numerical fluxes.
- **Kernel 4**:
  - 8. Compute tendency.
- **Kernel 5**:
  - 9. Apply tendency.

Technically, Kernels 4 and 5 could be merged for the ADER-DT algorithm, but if the code is to be general and include other time integrators, then the computation of tendencies must be separated from the application of tendencies. Kernel 1 is the kernel of most interest in this study, since the vast majority of the floating point operations occur in that kernel.

4.1. CPU software design. The Euler model is coded in C++ and uses the YAKL library,[2] using the `Array` class for multidimensional arrays and the `parallel_for` function and parallel reduction classes for all parallel work. YAKL is a simplified C++ portability library that uses a syntax similar to Kokkos.[3] In this approach, what would normally be a loop body is passed to a kernel launcher (`parallel_for`) as a class object called a functor, along with the number of loops to parallelize and the bounds of those loops. The `parallel_for` launcher then dispatches the loop body code on multiple hardware backends to provide portability. Figure 6 gives an example of what this looks like.

All functors passed to YAKL are created via C++ lambdas for convenience. While lambdas are harder to profile because of their somewhat convoluted function names, the number of variables to capture is typically quite large, making the convenience of lambdas more appealing. For the sake of code readability, we always leave the original loops associated with each YAKL `parallel_for` commented in the code. The code also uses a static array class, `SArray`, (templated on the type and dimension sizes) for low-overhead, small, local variables. Static array class dimensions are always associated with spatial or temporal order of accuracy or the number of fluid variables, each of which must be a compile-time constant. This was found to be a reasonable accommodation for sake of performance, since it does not need to change throughout the simulation.

Any function that is called within a parallel region is prefaced with `YAKL_INLINE` to tell the compiler to allow it to run in GPU or host memory and to inline it. This is a critical optimization strategy for GPU codes with many small, sequential function calls because true GPU function calls are costly.

The SageMath Python-based symbolic mathematical software[4] is used to generate C++ code for a class of functions that perform various transformations from one set

[2]https://github.com/mrnorman/YAKL
[3]https://github.com/kokkos/kokkos
[4]http://www.sagemath.org

```
void computeTend_X(real4d const &flux, real4d &tend) {
  // for (int l=0; l<numState; l++) {
  //   for (int k=0; k<nz; k++) {
  //     for (int j=0; j<ny; j++) {
  //       for (int i=0; i<nx; i++) {
  parallel_for( Bounds<4>(numState,nz,ny,nx) ,
    YAKL_LAMBDA (int l, int k, int j, int i) {
      tend(l,k,j,i) = -(flux(l,k,j,i+1)-flux(l,k,j,i))/dx;
  });
}
```

FIG. 6. *Code to compute time tendencies in the x-direction.* `real4d` *is a* `typedef` *for a YAKL* `Array<float,4>`.

of DOFs to another. For instance, the `Transform` class can transform cell averages to GLL points or polynomial coefficients to an estimate of Total Variation for WENO limiting. Each function in the `Transform` class works on static multidimensional array classes that are each templated on their array dimensions. This provides a convenient method of overloading transformation function names for each order of accuracy.

One surprising software engineering effort (coming from the perspective of a Fortran programmer) was the need to constrain floating point literals in the code. In C / C++, a nonsuffixed floating point literal is by default in double precision. In Fortran, compiler flags to control the inherent precision of floating point literals are common. However, the GNU C++ compiler has no such flag. Fortran also has language-standard suffixes for floating point literals. The absence of these in C++ is problematic because if one desires single precision, every floating point literal must be suffixed with an `f`. However, doing this explicitly means double precision code will use single precision literals, which is not desirable if one wishes to change the code's precision. After some searching, the most general solution appears to be the user-defined literals feature of the 2011 C++ standard. This feature allows the user to provide a custom suffix to all literals that can switch them to the desired precision at compile time. Regrettably, this leads to compile-time warnings for passing a `double` to a `long double` type (user-defined literals on floats must accept a `long double` type according to the standard).

4.2. GPU refactoring. By virtue of expressing loops as a `parallel_for` function launching the loop bodies as lambdas, the code can perform well on different architectures. When switching to the CUDA backend in YAKL, we had to create local references for all local class data and all global namespace data, e.g., `auto &classVar=this->classVar;` and `auto &globVar=::globVar;`. The reason for this is that C++ lambdas only automatically capture data in the *local* scope. By creating local references, the C++ lambdas then automatically capture all of the data used in the kernel by value so that it is accessible in GPU memory.

In the Euler model implemented for this study, all YAKL `Array` objects are by default placed in device memory. Therefore, when the `-D__USE_CUDA__` flag is passed to the compiler, these arrays automatically allocate the data in GPU device memory (i.e., using `cudaMalloc`). For MPI communication, the data is transferred to `Array` objects in the host memory space before sending them over MPI, and then they are transferred to device memory after the MPI messages are received. The code achieves perfect weak scaling both on-node and off-node if the problem size is sufficiently large (see Table 5), which is to be expected for a time-explicit fluids code.

4.3. Performance evaluation. In the 3-D code, originally, the WENO reconstruction and the ADER time stepping were in the same GPU kernel. But it turned out that the model ran significantly faster when these operations were separated into two kernels because of reduced register pressure. What this demonstrates is that while increasing the continuous amount of work improves computational intensity in theory, it can also increase register pressure on the GPU to the point that too many registers are allocated, and the GPU occupancy is too low for good efficiency. Therefore, there is an optimal middle ground for the amount of code in a single GPU kernel.

Because of this, the WENO reconstruction is identical between the ADER and SSPRK3 methods. For this reason, Table 1 only reports the percentage of peak flops with and without WENO limiting for the ADER method. The computation of the time average with ADER-DT consistently achieved 15% peak flops in all ADER cases because it does not include the WENO limiting, and it consistently uses three GLL points (for 3rd-order temporal accuracy).

We also compare the overall model end-to-end runtime for the following three cases to isolate the effects of WENO limiting and ADER time stepping separately on the overall runtime:

- E1: WENO limiting and ADER-DT time stepping.
- E2: WENO limiting and Runge–Kutta time stepping.
- E3: No limiting and Runge–Kutta time stepping.

For Runge–Kutta time stepping, the optimal three-stage, 3rd-order-accurate Strong Stability Preserving Runge–Kutta (SSPRK3) method from [12] is used. It has an SSP CFL value of 1, just like the ADER method, but it requires three stages to reach that CFL value whereas the ADER-DT method only requires one stage. Note that we use SSPRK3 not because it has the best SSP CFL value but because it is arguably the most common SSP RK integrator used. To ensure the comparison is best against best, the RK time stepping directly projects the N cell averages in the stencil onto two GLL points at the cell edges to compute the cell-edge fluxes. By contrast, the ADER-DT algorithm must project it onto $N_T = 3$ GLL points to compute the temporal operator up to 3rd-order accuracy. All simulations except the scaling benchmarks and power usage benchmark use a single Nvidia V100 GPU on OLCF's Summit computer. Floating point throughputs and operation counts on the GPU are obtained with Nvidia's "nvprof" tool.

5. Improved computational efficiency.

5.1. On-node performance. Table 1 shows the percentage of peak flops for the reconstruction GPU kernel with and without WENO limiting for varying orders of accuracy using 200 $\times$ 200 $\times$ 100 cells, and CFL value of 0.9, and simulating for 100 model seconds. It is evident that WENO limiting significantly improves the GPU utilization, as does a higher order of accuracy. Also notice that the flop/s decrease monotonically in going from the x- to the y- to the z-directions. This is likely because the initial stencil and final fluxes are being placed in increasingly strided memory locations (only the x-dimension is contiguous). At 9th-order accuracy with WENO limiting, however, there is enough work between these strided accesses that the decreased performance is negligible. Also note that the compute throughput is mostly maxed out at $> 75\%$ of peak flop/s on the V100 at 7th-order. Therefore, the increase in computations required by 9th-order accuracy will result in an increase in kernel runtime for that specific kernel.

Table 2 shows the end-to-end wallclock times of the 3-D model for varying configurations and orders of accuracy using 200 $\times$ 200 $\times$ 100 cells, a CFL value of 0.9,

Table 1

Percentage of peak flop/s ("% peak") and total number of floating point instructions ("Total") on the $V100$ for the reconstruction kernel of ADER and $SSPRK3$ with and without WENO limiting using $200 \times 200 \times 100 = 4$ million cells and a CFL value of 0.9 for 100 model seconds. Third-order accuracy in time is used for each simulation, but the temporal order of accuracy does not affect the flops for this particular kernel. "x-dir" specifies values for the x-direction sweep in the dimensional splitting approach, and likewise for "y-dir" and "z-dir."

Method	Order	x-dir		y-dir		z-dir	
		% peak	Total	% peak	Total	% peak	Total
Nolim	3	3.2	3.8e8	3.1	3.8e8	2.7	3.8e8
	5	6.1	6.2e8	5.8	6.2e8	4.3	6.2e8
	7	9.9	8.6e8	9.1	8.6e8	6.1	8.6e8
	9	11	1.1e9	10	1.1e9	5.6	1.1e9
WENO	3	29	3.1e9	29	3.1e9	25	3.1e9
	5	53	6.1e9	50	6.1e9	38	6.1e9
	7	76	1.1e10	75	1.1e10	58	1.1e10
	9	79	1.8e10	79	1.8e10	71	1.8e10

Table 2

End-to-end walltimes for the entire 3-D model for $SSPRK3$ without limiting, $SSPRK3$ with WENO limiting, and ADER with WENO limiting for 100 model seconds using $200 \times 200 \times 100 = 4$ million cells with a CFL value of 0.9 using a single GPU. Third-order accuracy in time is used for each simulation, but the temporal order of accuracy does not affect the flops for this particular kernel.

Order	3	5	7	9
RK nolim	11.5779	11.7512	11.8192	11.6403
RK WENO	11.6118	11.7393	11.801	12.4925
ADER WENO	4.86667	5.31531	5.65836	6.32869

and simulating for 100 model seconds. In all cases, the ADER-DT method is at least twice as fast as the SSPRK3 method. Also, WENO creates almost no overhead compared to the unlimited case. Because of the increased compute intensity, the extra computations are almost free. With WENO limiting, there is a somewhat larger jump in runtime in going from 7th-order to 9th-order, and that is mainly because the compute throughput on the GPU is saturated for the main computational kernel at around 80% of peak flop/s.

At 7th-order accuracy, running with WENO limiting requires over 13× more floating point operations than running without limiting. Yet, the runtime is virtually identical, meaning the extra work basically comes for free. This shows how well a GPU handles compute-intense workloads. Further, at 9th-order accuracy with WENO limiting, the main kernel requires almost 50× more floating point operations than 3rd-order accuracy with no limiting. Yet the overall runtime is only about 8% slower at 9th-order accuracy with WENO limiting on the V100 GPU (using the RK time integrator).

5.1.1. Power usage. Also, regarding power consumption, we consider the Thermal Design Power of the IBM Power9 CPU and the Nvidia V100 GPU on the Summit supercomputer in a node-level comparison. A 9th-order-accurate WENO-limited simulation with ADER with $400 \times 400 \times 200 = 32$ million cells for 10 model seconds consumed 3.8 kWatt-seconds of energy on the six V100 GPUs and 55.2 kWatt-seconds of energy on the two Power9 CPUs, meaning the GPUs were 14.5× more power efficient. A 3rd-order-accurate, nonlimited simulation with the same parameters consumed 2.8 kWatt-seconds of energy on the six V100 GPUs and 30.9 kWatt-seconds on the two

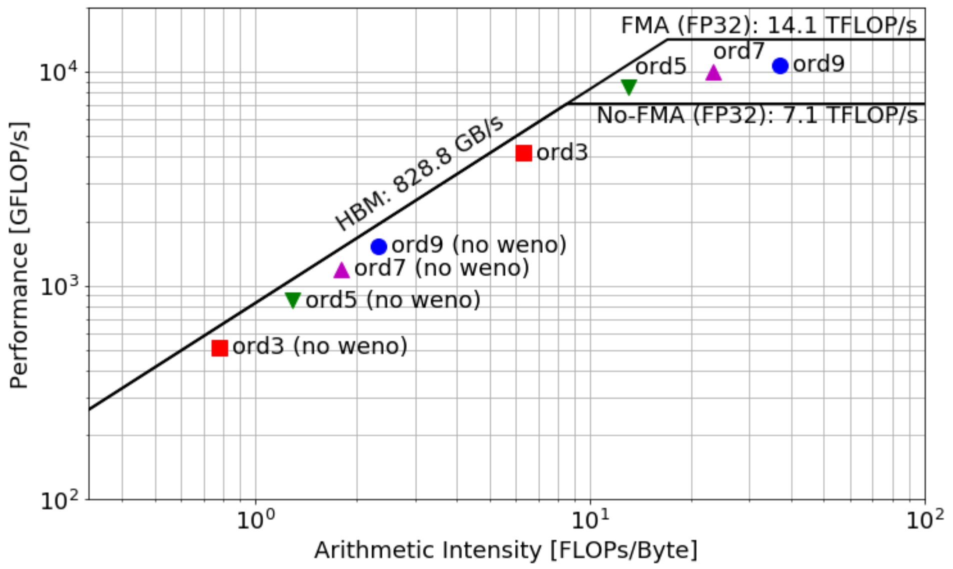


Fig. 7. *"Roofline" plot for the x-direction main reconstruction kernel for varying orders of accuracy with and without WENO limiting using* $200 \times 200 \times 100 = 4$ *million cells with 3rd-order accuracy in time. The solid, horizontal black line shows the maximum possible performance in billions of floating point operations per second ("GFLOP/s") both with and without use of Fused-Multiply-Add (FMA) instructions. The solid black, diagonal line represents a bandwidth-bound code (performance is limited by memory bandwidth). Where the two lines meet the code transitions to a compute-bound regime (performance is limited by floating point throughput). The closer a marker is to the solid black line, the greater the % of the peak performance along that performance limiter. FMA represents a special instruction that fuses a successive multiply and add operation together in higher precision. We compiled with FMA turned on. "FP32" stands for 32-bit (or "single precision") floating point operations. HBM stands for High-Bandwidth Memory for the bandwidth-bound portion of the scaling curve.*

Power9 CPUs, meaning the GPUs were 11.1× more power efficient.

5.1.2. Computational and arithmetic intensity. Finally, to gain a more detailed understanding of arithmetic intensity and its affects on the performance of the kernel, Figure 7 shows a "roofline" plot of the performance of the main reconstruction kernel for varying orders of accuracy with and without WENO limiting using $200 \times 200 \times 100 = 4$ million cells at 3rd-order temporal accuracy. The Roofline Model [33] is a technique of evaluating how well a particular operation is performing in comparison to the two main system limiters, bandwidth and computation. As the computational intensity of an operation increases, operations will naturally move up the bandwidth line until becoming limited by the FLOPs line. Codes that fall well below either line are typically limited by some other performance limiter, such as memory latency.

Whereas section 2.4 argues the reasons why WENO limiting increases the computational intensity of the operations, Figure 7 backs up these claims with performance measured using the NVIDIA Nsight Compute profiler on the Summit supercomputer. Notice that the four kernels without WENO limiting each fall along, but slightly below, the diagonal bandwidth line. Due to their low computational intensity they will never be able to fully saturate the floating-point performance of the GPU, although with each increase in order they move closer to the compute-bound regime.

As explained in section 2.4, WENO limiting significantly increases the computational intensity of the reconstruction kernel, and by 5th-order accuracy, the kernels have moved firmly into the compute-bound region of the plot. These kernels land between the lines representing 32-bit floating point performance with and without FMA instructions, indicating that the actual kernels likely use a mixture of FMA and non-FMA instructions. This gives a strong indication that if a future accelerator has an increase in FP32 performance, these kernels would likely see a commensurate increase in performance.

In section 2.4, it was stated that over 100 floating point operations can be performed for every fetch of two single precision floating point values from DRAM on the V100 GPU. This might seem to contradict Figure 7, but note that the x-axis of Figure 7 is FLOPs per *byte* (and two single-precision floating point values contain eight bytes).

5.1.3. Fully 2-D reconstruction. In nonorthogonal coordinates, often it isn't possible to do a straightforward dimensional splitting without incurring unacceptable error. In these settings the developer's only choice is to reconstruct intracell variation in a fully 2-D manner in the horizontal direction. Therefore, this section investigates GPU performance for a high-order WENO-limited reconstruction kernel. The algorithm for this kernel, which uses a tensored approach, is as follows:

- For each cell:
  - Compute the average row of the 2-D $N \times N$ stencil of cell averages.
  - Compute WENO weights for the average row.
  - For each of the N rows within the 2-D stencil:
    - Reconstruct N WENO coefficients for this row's stencil using the average row WENO weights.
    - Project N coefficients onto N_T GLL points for this row.
  - Compute the average column of the 2-D $N_T \times N$ stencil (GLL points in the x-direction, and cell averages in the y-direction).
  - Compute WENO weights for the average column.
  - For each of the N_T columns within the 2-D stencil:
    - Reconstruct N WENO coefficients for this column's stencil using the average column WENO weights.
    - Project N coefficients onto N_T GLL points for this column.
    - Store into $N_T \times N_T$ GLL points.

Previous experience has shown that using a single set of WENO weights per dimension in fully multidimensional reconstruction gives smoother results. That is why the WENO weights are computed on the average and then shared for each row and column. It also reduces the total amount of required computation.

Table 3 gives performance metrics for this kernel using $200 \times 200 \times 100 = 4$ million cells on a V100 GPU for varying orders of accuracy with and without WENO limiting. It follows a very similar pattern as seen in the dimensionally split (1-D) reconstruction kernels. The point of saturated compute throughput on the GPU is also at 7th-order accuracy in the 2-D reconstruction kernel, but the number of computations is up to $10\times$ larger than the 1-D case for this kernel. It may seem odd that the runtime actually goes *down* with increasing workloads in a lot of cases, but this kind of behavior is fairly normal with kernels that are more compute bound than bandwidth bound. Seemingly minute changes in the code can lead to noticeable changes in the GPU runtime, and it can be hard to predict. Notice how the runtime increases by a factor of $2\times$ when moving from 7th-order to 9th-order accuracy, which

Table 3

Performance metrics for a 2-D reconstruction kernel with WENO limiting using $200 \times 200 \times 100 = 4$ million cells on a $V100$ GPU. "% peak" means the percentage of peak flop/s on the $V100$, "Total" means the total number of floating point instructions, and "Runtime" is the kernel runtime (for a single kernel call) in milliseconds.

Limiter	Order	% peak	Total	Runtime (ms)
None	3	3.2	2.16e9	5.04
	5	6.95	4.80e9	4.66
	7	11.92	8.40e9	4.5
	9	11.22	1.30e10	7.75
WENO	3	19.64	1.33e10	4.34
	5	46.84	3.85e10	5.27
	7	77.58	8.79e10	7.24
	9	74.59	1.71e11	14.64

Table 4

Total model time per time step per cell ("Time") and percentage of peak flops for the x-direction reconstruction for varying workloads using a CFL value of 0.9 after 100 model seconds of simulation using 9th-order accuracy and WENO limiting using a single GPU.

# cells	Time / cell (10^{-9}s)	% peak flop/s
$200 \times 200 \times 100$ (4 million)	4.07	79%
$100 \times 100 \times 50$ (500K)	4.77	73%
$50 \times 50 \times 25$ (63K)	10.5	48%
$30 \times 30 \times 15$ (14K)	44.1	19%
$16 \times 16 \times 8$ (2K)	286	4.2%

is nearly identical to the factor by which the total number of computations increased. This demonstrates that once the compute throughput is saturated, all extra work is forced to increase the runtime linearly.

5.1.4. Small workload investigation. It is worth investigating at what point (in terms of decreasing workload) the high flop rate of the WENO-based reconstruction operator begins to tail off. There are 5,120 single precision floating point units on a single V100 GPU, and therefore, anything below this level has no chance of properly utilizing the GPU. In reality, one needs significantly more threads than there are floating point units in order to begin to properly hide latency from DRAM fetches. The more local work there is per data fetch, the more "free" the extra local flops become, lending a potential advantage to the new algorithm presented in this study.

Table 4 shows the total model runtime per time step per cell (using a CFL value of 0.9 at 9th-order accuracy with ADER and WENO) and the floating point percentage of peak for the x-direction's WENO reconstruction kernel for varying workloads ranging from 4 million cells to 2,000 cells on the V100 GPU in single precision.[5] For a point of reference, current climate workloads (climate uses smaller per-node workloads than weather because it must run faster) on Summit with the Energy Exascale Earth System Model (E3SM) [7] run with a minimum of 10^5 GPU threads per V100. For the algorithm in this study, that is roughly enough threads to get a sizeable percentage of peak flops.

5.2. Off-node performance. The WENO algorithm has several beneficial properties in parallel. First, while a stencil of data must be transferred, if the MPI

[5]Note that while some weather and climate physics parameterizations cannot run in single precision, different precisions can be used for different model components

Table 5

Weak scaling up to four Summit nodes with $200 \times 200 \times 100 = 4$ *million cells per GPU for* 200 *seconds of simulation at* 9*th-order accuracy with ADER and WENO limiting.*

# GPUs	Walltime (s)
1	13.63
2	13.19
4	13.44
6	13.4
12	13.38
24	13.41

Table 6

Strong scaling up to 24 *GPUs on four Summit nodes for* 100 *seconds of simulation using a problem size of* $200 \times 200 \times 100$ *cells at* 9*th-order accuracy with ADER and WENO limiting.*

# GPUs	Cells per GPU	Walltime (s)	Efficiency
1	4.0×10^6	32.64	—
2	2.0×10^6	17.79	92%
4	1.0×10^6	9.274	88%
6	6.7×10^5	6.682	81%
12	3.3×10^5	3.725	73%
24	1.7×10^5	2.258	60%

overheads are dominated by message latency (which they are for very strong-scaled applications with little data per message), the extra data packed into messages for higher-order will not be noticed. Further, with WENO limiting, all oscillation limiting can be done with this same stencil in the same time step without additional data movement. By contrast, post hoc limiters such as Flux-Corrected Transport, viscous operators, and optimization-based limiters not only require additional parallel data traffic but also must be done after the fact in a separate step of computations, meaning the computations are not densely clustered between periods of data movement.

The ADER-DT algorithm also has benefits in parallel because it can run with time steps that are 2–3 $\times$ larger than most SSP semidiscrete operators [12]. This is, in essence, avoiding data communication by increasing local computations, which is exactly what modern, accelerated hardware thrives at. This is true both on-node and between nodes.

In Table 5, the weak scaling of this model is evaluated using $200 \times 200 \times 100$ cells per GPU from one to 24 GPUs (four Summit nodes) using 9th-order accuracy, WENO, and ADER for 200 model seconds, and perfect weak scaling is achieved. In Table 6, the strong scaling is evaluated with the same configuration except with a fixed problem size of $300 \times 300 \times 150$ cells. The code achieves 60% scaling efficiency in going from one GPU to 24 GPUs on four nodes. Note that in Table 4, we see that for the strong scaled problem size on 24 GPUs, there is no significant on-node GPU performance degradation due to lack of threading. Therefore, we know that the degradation in scaling efficiency is because of the `cudaMemcpy` and MPI data transfer times alone.

6. Conclusions and future directions. In section 1, the constraints of atmospheric modeling in weather and climate were introduced. A new algorithmic approach using ADER time stepping with Differential Transforms and WENO limiting that improves both accuracy and computational efficiency in a realistic context were detailed in section 2. The goal is to consider as many aspects of the problem as possible together at once with regard to overall efficiency. Then, the algorithm's improvements to accuracy were detailed in section 3, where it was shown for simple and complex problems that high-order accuracy improved resolution and that WENO limiting removed oscillations and noise without degrading resolution appreciably. The experimental setup and implementation details were given in section 4.

The algorithm's improvements to computational efficiency were demonstrated in section 5. The use of WENO limiting improved the GPU utilization significantly compared to the nonlimited case, and the new algorithm achieves up to 79% peak on the Nvidia Tesla V100 GPU for the reconstruction kernel. Notably, the ADER-DT algorithm can operate at effective CFL value three times larger than the optimal 3rd-order-accurate Strong Stability Preserving Runge–Kutta method. This algorithmic approach simultaneously improves time to solution, accuracy, and hardware utilization by strategically increasing local, densely clustered computations in a manner that avoids data communication and better utilizes GPU hardware. End-to-end, the ADER time discretization was twice as fast as the SSPRK3 baseline in the 3-D benchmark. Further, even though 9th-order accuracy with WENO limiting requires almost $50\times$ more floating point operations than 3rd-order accuracy without limiting, the overall runtime increases by less than 30% because GPUs handle compute intensity very well.

The newly developed algorithm improves the runtime compared to using traditional time integrators, but it's important to note that this is only because ADER has a larger maximum effective CFL value than the SSPRK3 integrator. For methods with the same maximum effective CFL value, each pass through the data comes at a relatively fixed cost. The goal, then, is to do as much with each pass through the data as you can (thus the idea of compute intensity). The algorithm presented here has demonstrated that significantly more computations can be done in each pass through the data without incurring much extra cost, particularly on modern GPU architectures. This provides more resolution and better behavior per Degree of Freedom, and this may, in turn, allow a user to use fewer Degrees of Freedom in their model.

For future work, there are plans to add microphysics and radiation to the dynamical core to create a fully functional cloud model. This method can be expanded to cubed-sphere nonorthogonal coordinates for global simulation, but if ADER-DT is to be used, the horizontal discretization must become fully 2-D rather than dimensionally split. Fortunately, with the 2-D reconstruction kernel implemented and tested in this study, it appears the same properties of 1-D reconstruction also carry over to 2-D reconstruction.

Appendix A. 3-D compressible, nonhydrostatic Euler equations. The 3-D compressible, nonhydrostatic Euler equations used in this study are as follows:

$$\frac{\partial \mathbf{q}}{\partial t}+\frac{\partial \mathbf{f}\left(\mathbf{q}\right)}{\partial x}+\frac{\partial \mathbf{g}\left(\mathbf{q}\right)}{\partial y}+\frac{\partial \mathbf{h}\left(\mathbf{q}\right)}{\partial z}=\mathbf{s},$$

$$\mathbf{q}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho\theta\end{bmatrix},\quad \mathbf{f}=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uw\\ \rho u\theta\end{bmatrix},\quad \mathbf{g}=\begin{bmatrix}\rho v\\ \rho vu\\ \rho v^2+p\\ \rho vw\\ \rho v\theta\end{bmatrix},$$

$$\mathbf{h} = \begin{bmatrix} \rho w \\ \rho w u \\ \rho w v \\ \rho w^2 + p - p_H \\ \rho w \theta \end{bmatrix}, \quad \mathbf{s} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -(\rho - \rho_H)\, g \\ 0 \end{bmatrix},$$

where ρ is density, u, v, and w are winds in the x-, y-, and z-directions, respectively, θ is potential temperature related to temperature, T, by $\theta = T\,(P_0/P)^{R_d/c_p}$, $P_0 = 10^5\,\mathrm{Pa}$, $g = 9.8\ \mathrm{m\,s^{-2}}$ is acceleration due to gravity, $p = C_0\,(\rho\theta)^\gamma$ is pressure, $C_0 = R_d^\gamma p_0^{-R_d/c_v}$, $R_d = 287\,\mathrm{J\,kg^{-1}\,K^{-1}}$, $\gamma = c_p/c_v$, $c_p = 1004\,\mathrm{J\,kg^{-1}\,K^{-1}}$, $c_v = 717\,\mathrm{J\,kg^{-1}\,K^{-1}}$, and $p_0 = 10^5\,\mathrm{Pa}$. The hydrostatic density and potential temperature are related through

$$\frac{dp_H}{dz} = C_0 \frac{d\,(\rho_H \theta_H)^\gamma}{dz} = -\rho_H g.$$

Appendix B. Finite-volume discretization. Since the equations are dimensionally split, consider the general case of a 1-D PDE with a source term. For the Finite-Volume discretization, the 1-D equation, $\partial_t \mathbf{q} + \partial_x \mathbf{f}\,(\mathbf{q}) = \mathbf{s}$, is integrated over a 1-D local cell domain, $\Omega_i \in \left[x_{i-1/2}, x_{i+1/2}\right]$, where $x_{i\pm 1/2} = x_i \pm \Delta x_i/2$, and Δx_i is the grid spacing for the cell of index i. This gives rise to the following semidiscretized equations:

$$\frac{\partial \overline{\mathbf{q}}_i}{\partial t} + \frac{1}{\Delta x_i}\left(\mathcal{F}_{i+1/2} - \mathcal{F}_{i-1/2}\right) = \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} \widetilde{\mathbf{s}}_i\,(x,t)\,dx,$$

$$\mathcal{F}_{i-1/2}\,(t) = \mathcal{R}\left[\widetilde{\mathbf{f}}_{i-1}\left(x_{i+1/2}, t\right), \widetilde{\mathbf{f}}_i\left(x_{i-1/2}, t\right)\right],$$

$$\overline{\mathbf{q}}_i \equiv \frac{1}{\Delta x_i}\int_{x_{i-1/2}}^{x_{i+1/2}} q\,(x,t)\,dx,$$

where $\widetilde{\mathbf{f}}_i\,(x,t)$ and $\widetilde{\mathbf{s}}_i\,(x,t)$ are spatial reconstructions of the vectors $\mathbf{f}$ and $\mathbf{s}$, respectively, over the ith cell domain, Ω_i (the time dimension is left continuous for now), and $\mathcal{R}\,(\mathbf{f}^-, \mathbf{f}^+)$ is a Riemann solver to reconcile the multivalued flux at a cell interface due to samples from disparate reconstructions at the same location in space.

Appendix C. WENO limiting. WENO limiting is depicted visually in Figure 8 with a discontinuity at $x = 1/2$. In this figure, a 5th-order-accurate polynomial is formed using five cell averages, and three quadratic polynomials are formed as well. The resulting polynomial is less oscillatory than the high-order polynomial because the candidate polynomials weights are inversely proportional to their Total Variation, giving the smoothest polynomials the largest weights.

Appendix D. Differential Transforms for the Euler equations. This appendix gives the Differential Transforms (DTs) of the Euler equations (see Appendix A) used to compute time derivatives and ultimately time averages of the flux and source terms over a time step. Like any transform, DTs transform the PDE into a recurrence relation and has an inverse. The kth DT of a function is the coefficient to the kth term in the Taylor series of that function. The inverse transform is the Taylor series itself:

$$F\,(k) = \frac{1}{k!}\left.\frac{\partial^k f\,(t)}{\partial t^k}\right|_{t=t_n},$$

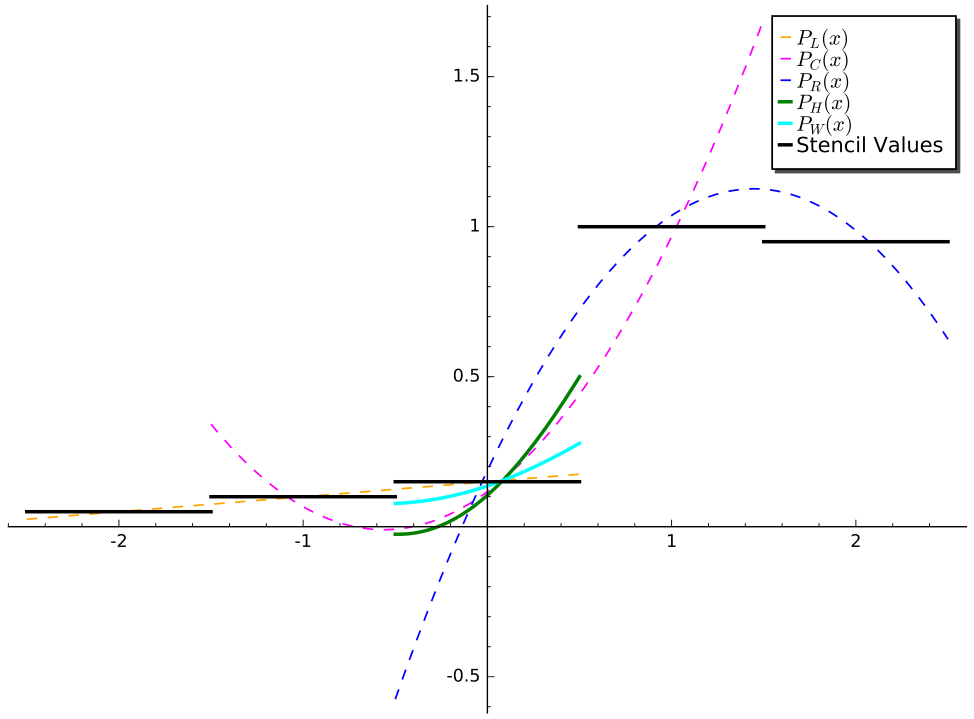


Fig. 8. *Example of WENO limiting, assuming a grid spacing of* 1*, where* $P_L(x)$*,* $P_C(x)$*, and* $P_R(x)$ *are the left-biased, centered, and right-biased lower-ordered polynomials, respectively; and* $P_H(x)$ *and* $P_W(x)$ *are the high-order and WENO polynomials over the center cell's domain. Thick black lines denote the stencil average values for this example. Note the domains of* P_L*,* P_C*, and* P_R *all overlap in the center cell.*

$$f(t) = \sum_{k=0}^{N-1} F(k)\,(t-t_n)^k .$$

In our case, we only perform DTs in the temporal dimension. Therefore, spatial indices are neglected for simplicity. Recalling that the multidimensional PDE is solved in a dimensionally split manner, the transformed PDEs for each split direction in the x-, y-, and z-directions, respectively, are as follows:
***x*-direction DT:** $\mathbf{Q}(k_t+1) = -\partial_x \mathbf{F}(k_t)/(k_t+1)\,,$
***y*-direction DT:** $\mathbf{Q}(k_t+1) = -\partial_y \mathbf{G}(k_t)/(k_t+1)\,,$
***z*-direction DT:** $\mathbf{Q}(k_t+1) = -\left[\partial_z \mathbf{H}(k_t) - \mathbf{S}(k_t)\right]/(k_t+1)\,,$
where a capital letter denotes the transformed function in time and k_t denotes the (k_t)th-order DT in time.

Before computing the DTs of the flux and source terms, the following auxiliary functions are defined: $\phi_{ij}(t) = q_i(t)\,q_j(t)/q_1(t)$ and $p^\star(t) = q_5(t)^\gamma$. The symmetry $\phi_{ij} = \phi_{ji}$ can be employed to reuse previously computed auxiliary functions instead of recomputing the same DT twice. Given these functions, the following are the temporal DTs of the flux and source vectors in the Euler equations:

$$\mathbf{F}(k_t) = \begin{bmatrix} Q_2(k_t) \\ \Phi_{22}(k_t) + P(k_t) \\ \Phi_{23}(k_t) \\ \Phi_{24}(k_t) \\ \Phi_{25}(k_t) \end{bmatrix}, \quad \mathbf{G}(k_t) = \begin{bmatrix} Q_3(k_t) \\ \Phi_{32}(k_t) \\ \Phi_{33}(k_t) + P(k_t) \\ \Phi_{34}(k_t) \\ \Phi_{35}(k_t) \end{bmatrix},$$

$$\mathbf{H}(k_t) = \begin{bmatrix} Q_4(k_t) \\ \Phi_{42}(k_t) \\ \Phi_{43}(k_t) \\ \Phi_{44}(k_t) + P(k_t) \\ \Phi_{45}(k_t) \end{bmatrix}, \quad \mathbf{S}(k_t) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -gQ_1(k_t) \\ 0 \end{bmatrix},$$

$$\Phi_{ij}(k_t) = \frac{1}{Q_1(0)} \sum_{r_t=0}^{k_t} \left[Q_i(r_t) Q_j(k_t - r_t) - Q_1(r_t) \Phi_{ij}(k_t - r_t) \right],$$

$$P(k_t) = \frac{C_0}{2} P^\star(k_t) \quad \forall k_t > 0,$$

$$P^\star(k_t) = \frac{1}{Q_5(0)} \left[\gamma P^\star(0) Q_5(k_t) + \frac{1}{k_t} \sum_{r_t=0}^{k_t-1} \mathcal{P}(r_t, k_t) \right],$$

$$\mathcal{P}(r_t, k_t) = \left[(k_t - r_t) \left(\gamma P^\star(r_t) Q_5(k_t - r_t) - Q_5(r_t) P^\star(k_t - r_t) \right) \right].$$

These functions are initialized as follows. Given that the zeroth-order temporal DT of a function is simply the values themselves at the spatial GLL point in question,

$$\Phi_{ij}(0) = Q_i(0) Q_j(0) / Q_1(0),$$

$$P(0) = C_0 P^\star(0),$$

$$P^\star(0) = Q_5(0)^\gamma.$$

The summations used to compute the DTs for $\Phi_{ij}(k_t)$ and $P^\star(k_t)$ include the actual values $\Phi_{ij}(k_t)$ and $P^\star(k_t)$ themselves, even though they are not technically defined by that point. In these cases when a function's DT depends on itself, it is assumed to be zero in the summation.

Implementation specifics. Given the initial DTs, $\mathbf{Q}(0)$, which are simply the values of $\mathbf{q}$ at a given spatial GLL point, one can compute the zeroth-order DTs of the flux and source terms. From there, a spatial derivative of the zeroth-order flux DTs must be computed, and then one can compute $\mathbf{Q}(1)$. Then, iteratively, higher-order DTs of the state, flux, and source vectors can be computed accordingly. All that is left is creating an efficient means of computing a spatial derivative, $\partial_x \mathbf{F}(k_t)$.

To compute a single spatial derivative of $\mathbf{F}(k_t)$, an $N_T \times N_T$ matrix, $\mathcal{M}_D$, is created that transforms the N_T GLL points over the cell into N_T polynomial coefficients, differentiates the polynomial, and then casts the differentiated polynomial coefficients back into N_T GLL points across the cell. To compute the matrix, $\mathcal{M}_D$, let us first define an $(N_T - 1)$th-order polynomial as $p(x) = \sum_{i=0}^{N_T-1} a_i x^i$, where the polynomial is defined over a local coordinate $x \in [-1/2, 1/2]$. Then, define two vectors of constraints:

$$a'_i = \begin{cases} i a_{i+1} & \text{if } 0 \le i \le N_T - 1, \\ 0 & \text{if } i = N_T, \end{cases}$$

$$c_{GLL,i} = \frac{1}{\Delta x} p\left(\xi_i\right) \quad \forall i \in \left\{0, \dots, N_T - 1\right\},$$

where ξ_i is the ith GLL point on the domain $[-1/2, 1/2]$. The matrix that transforms GLL points to coefficients, differentiates those coefficients, and transforms them back into GLL points is defined as

$$\mathcal{M}_D = \frac{\partial \boldsymbol{c}_{GLL}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}'}{\partial \boldsymbol{a}} \left(\frac{\partial \boldsymbol{c}_{GLL}}{\partial \boldsymbol{a}} \right)^{-1}.$$

This matrix is precomputed using symbolic mathematical software before the simulation is run.

Algorithmic simplifications. Traditionally, ADER methods perform a Riemann solve not only for the flux, but also for the temporal derivatives as well. This sets up high-order Derivative Riemann Problems that are typically solved using the locally constant linear upwind state. In this implementation, however, the algorithm is simplified by computing a single, traditional Riemann solve on the *time-averaged* state. This has two advantages over traditional ADER approaches: (1) only one Riemann solve needs to be performed per cell edge per time step and (2) less data needs to be communicated between parallel nodes.

Online resources. The code that produced all results in this paper is available publicly online at https://github.com/mrnorman/awflCloud. The code serves as a more practical documentation of exactly what is being done in the algorithms described in this paper and how they are implemented on accelerated hardware.

REFERENCES

[1] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: A survey*, J. Mach. Learn. Res., 18 (2018), 153.

[2] Y. Chen, H. Weller, S. Pring, and J. Shaw, *Comparison of dimensionally split and multi-dimensional atmospheric transport schemes for long time steps*, Q. J. R. Meteorol. Soc., 143 (2017), pp. 2764–2779.

[3] R. Courant, K. Friedrichs, and H. Lewy, *On the partial difference equations of mathematical physics*, IBM J. Res. Develop., 11 (1967), pp. 215–234.

[4] S. J. Day, N. Mullineux, and J. Reed, *Developments in obtaining transient response using Fourier transforms: Part* I: *Gibbs phenomena and Fourier integrals*, Int. J. Elect. Engng. Educ., 3 (1965), pp. 501–506.

[5] J. M. Dennis, J. Edwards, K. J. Evans, O. Guba, P. H. Lauritzen, A. A. Mirin, A. St-Cyr, M. A. Taylor, and P. H. Worley, *CAM-SE: A scalable spectral element dynamical core for the Community Atmosphere Model*, Int. J. High. Perform. Comput. Appl., 26 (2012), pp. 74–89.

[6] A. Ditkowski, S. Gottlieb, and Z. Grant, *Explicit and implicit error inhibiting schemes with post-processing*, Comput. & Fluids, 208 (2020), 104534.

[7] E3SM Project, *Energy Exascale Earth System Model (E3SM)*, [Computer Software], April 2018, https://doi.org/10.11578/E3SM/dc.20180418.36.

[8] M. Emmett and M. Minion, *Toward an efficient parallel in time method for partial differential equations*, Commun. Appl. Math. Comput. Sci., 7 (2012), pp. 105–132.

[9] K. J. Evans, R. K. Archibald, D. J. Gardner, M. R. Norman, M. A. Taylor, C. S. Woodward, and P. H. Worley, *Performance analysis of fully explicit and fully implicit solvers within a spectral element shallow-water atmosphere model*, Int. J. High. Perform. Comput. Appl., 33 (2019), pp. 268–284.

[10] A. Fournier, M. A. Taylor, and J. J. Tribbia, *The spectral element atmosphere model (SEAM): High-resolution parallel computation and localized resolution of regional dynamics*, Mon. Weather Rev., 132 (2004), pp. 726–748.

[11] F. X. Giraldo and M. Restelli, *A study of spectral element and discontinuous Galerkin methods for the Navier–Stokes equations in nonhydrostatic mesoscale atmospheric modeling: Equation sets and test cases*, J. Comput. Phys., 227 (2008), pp. 3849–3877.

[12] S. Gottlieb, *On high order strong stability preserving Runge-Kutta and multi step time discretizations*, J. Sci. Comput., 25 (2005), pp. 105–128.

[13] S. Gottlieb and C.-W. Shu, *Total variation diminishing Runge-Kutta schemes*, Math. Comp., 67 (1998), pp. 73–85.

[14] O. Guba, M. Taylor, and A. St-Cyr, *Optimization-based limiters for the spectral element method*, J. Comput. Phys., 267 (2014), pp. 176–195.

[15] C. Hu and C.-W. Shu, *Weighted essentially non-oscillatory schemes on triangular meshes*, J. Comput. Phys., 150 (1999), pp. 97–127.

[16] K. K. Katta, R. D. Nair, and V. Kumar, *High-order finite-volume transport on the cubed sphere: Comparison between* 1*D and* 2*D reconstruction schemes*, Mon. Weather Rev., 143 (2015), pp. 2937–2954.

[17] P. Lauritzen, W. Skamarock, M. Prather, and M. Taylor, *A standard test case suite for two-dimensional linear transport on the sphere*, Geosci. Model Dev., 5 (2012), pp. 887–901.

[18] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*, Cambridge Texts Appl. Math., Cambridge University Press, 2002.

[19] F. Li, W. D. Collins, M. F. Wehner, D. L. Williamson, J. G. Olson, and C. Algieri, *Impact of horizontal resolution on simulation of precipitation extremes in an aqua-planet version of Community Atmospheric Model (CAM*3*)*, Tellus A, 63 (2011), pp. 884–892.

[20] X.-D. Liu, S. Osher, and T. Chan, *Weighted essentially non-oscillatory schemes*, J. Comput. Phys., 115 (1994), pp. 200–212.

[21] E. H. Müller and R. Scheichl, *Massively parallel solvers for elliptic partial differential equations in numerical weather and climate prediction*, Q. J. R. Meteorol. Soc., 140 (2014), pp. 2608–2624.

[22] R. Nair, H.-W. Choi, and H. Tufo, *Computational aspects of a scalable high-order discontinuous galerkin atmospheric dynamical core*, Comput. & Fluids, 38 (2009), pp. 309–319.

[23] M. R. Norman and R. D. Nair, *A positive-definite, WENO-limited, high-order finite volume solver for* 2*-D transport on the cubed sphere using an ADER time discretization*, J. Adv. Model. Earth Syst., 10 (2018), pp. 1587–1612.

[24] K. G. Pressel, S. Mishra, T. Schneider, C. M. Kaul, and Z. Tan, *Numerics and subgrid-scale modeling in large eddy simulations of stratocumulus clouds*, J. Adv. Model. Earth Syst., 9 (2017), pp. 1342–1365.

[25] W. Putman and S.-J. Lin, *A finite-volume dynamical core on the cubed-sphere grid*, in Numerical Modeling of Space Plasma Flows: Astronum-2008, ASP Conference Series, Vol. 406, 2009, pp. 268–276.

[26] A. Robert, T. L. Yee, and H. Ritchie, *A semi-Lagrangian and semi-implicit numerical integration scheme for multilevel atmospheric models*, Mon. Weather Rev., 113 (1985), pp. 388–394.

[27] A. J. Robert, *The integration of a low order spectral form of the primitive meteorological equations*, J. Meteor. Soc. Japan., 44 (1966), pp. 237–245.

[28] C. Ronchi, R. Iacono, and P. S. Paolucci, *The "cubed sphere": A new method for the solution of partial differential equations in spherical geometry*, J. Comput. Phys., 124 (1996), pp. 93–114.

[29] W. C. Skamarock, *Evaluating mesoscale NWP models using kinetic energy spectra*, Mon. Weather Rev., 132 (2004), pp. 3019–3032.

[30] W. C. Skamarock and J. B. Klemp, *A time-split nonhydrostatic atmospheric model for weather research and forecasting applications*, J. Comput. Phys., 227 (2008), pp. 3465–3485.

[31] E. F. Toro and V. A. Titarev, *Derivative Riemann solvers for systems of conservation laws and ADER methods*, J. Comput. Phys., 212 (2006), pp. 150–165.

[32] I. Tsukanov and M. Hall, *Fast Forward Automatic Differentiation Library (FFADLib): A User Manual*, Spatial Automation Laboratory, Technical report SAL 2000-4, University of Wisconsin-Madison, 2000.

[33] C. Yang, T. Kurth, and S. Williams, *Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-*9 *Perlmutter system*, Concurr. Comput., 32 (2020), e5547, https://doi.org/10.1002/cpe.5547.

[34] S. T. Zalesak, *Fully multidimensional flux-corrected transport algorithms for fluids*, J. Comput. Phys., 31 (1979), pp. 335–362.