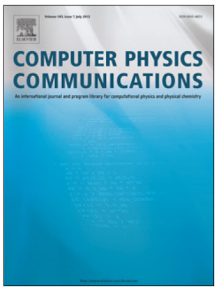


ZEFR: A GPU-accelerated high-order solver for compressible viscous flows using the flux reconstruction method

J. Romero, J. Crabill, J.E. Watkins, F.D. Witherden, A. Jameson

PII: S0010-4655(20)30022-9
DOI: https://doi.org/10.1016/j.cpc.2020.107169
Reference: COMPHY 107169

To appear in: *Computer Physics Communications*

Received date : 4 September 2019
Revised date : 12 December 2019
Accepted date : 7 January 2020

ZEFR: A GPU-accelerated high-order solver for compressible viscous flows using the Flux Reconstruction method

J. Romero, J. Crabill, J. E. Watkins, F. D. Witherden[*] and A. Jameson

Department of Aeronautics & Astronautics, Stanford University, CA 94305

Department of Ocean Engineering, Texas A&M University, TX 77843

Abstract

In this work we present ZEFR, a GPU accelerated flow solver based around the high-order accurate flux reconstruction (FR) approach. Written in a combination of C++ and CUDA, ZEFR is designed to perform scale resolving simulations within the vicinity of complex geometrical configurations. A unique feature of ZEFR is its support for overset grids; a feature which greatly expands the addressable problem space compared with existing high-order codes. The C++ implementation of FR in a manner which is suitable for modern hardware platforms is described in detail. Additionally, an overview of the input deck used by ZEFR is included. Validation results are presented for a range of steady and unsteady flow problems including Couette flow, the Taylor–Green vortex, and flow around an SD7003 aerofoil. Single node performance on a NVIDIA V100 GPU is analysed where it is shown that all of the kernels in ZEFR attain a high proportion of peak memory bandwidth. Moreover, multi-node performance is also assessed with strong scalability being demonstrated from 60 to 3840 NVIDIA V100 GPUs.

[*]Corresponding author; e-mail fdw@tamu.edu.

Program Description

Authors

Program title ZEFR v1.0

Licensing provisions 3 clause BSD license

Programming language C++ and CUDA

Computer Variable, up to and including GPU clusters

Operating system Recent version of Linux and macOS

RAM Variable, from hundreds of megabytes to gigabytes

Number of processors used Variable, code is multi-GPU and multi-CPU aware.

External routines/libraries Eigen, HDF5, METIS, MPI, and TIOGA.

Nature of problem Compressible Euler and Navier–Stokes equations.

Solution method High-order direct flux reconstruction approach suitable for curved, mixed, unstructured grids.

Unusual features Code incorporates support for overset grids.

Running time Many small problems can be solved on a recent workstation in minutes to hours.

1 Introduction

Over the past decade there has been an increasing desire amongst industrial practitioners of computational fluid dynamics (CFD) to conduct high-fidelity scale-resolving simulations (SRS) of compressible flows [1]. Such simulations, however, are typically several orders of magnitude more expensive than the Reynolds averaged Navier–Stokes simulations which represent the mainstay of industrial CFD. To this end there has been significant interest in the potential of high-order methods on unstructured grids to reduce the cost of SRS [2]. Examples of such schemes include the discontinuous Galerkin (DG) method [3, 4], the spectral volume method [5], and spectral difference (SD) methods [6, 7], amongst others.

Of particular interest is the flux reconstruction (FR) approach of Huynh [8]. Proposed in 2007 the FR approach is appealing for it unifies several existing high-order schemes, including nodal DG and SD methods, within a single framework. In addition to offering high-order accuracy on unstructured mixed grids, FR schemes are also compact in space, and thus when combined with explicit time marching offer a significant degree of element locality. These attributes enable FR to admit a particularly efficient implementation on both conventional and modern hardware platforms. An example of such a solver is PyFR [9, 10] which was specifically designed to leverage the synergies between high-order FR and modern hardware.

However, one difficulty facing high-order codes has been how to effectively simulate cases which involve the relative motion of several distinct bodies. This difficulty has prevented FR codes from being applied to otherwise tractable problems such as rotor-stator interaction in gas turbines and store separation on aircraft. Within the context of finite volume methods the main computational approach for such simulations is the *overset* method. Here, each body has its own grid which are patched together into a coherent whole via a domain connectivity algorithm. This approach has the advantage of greatly simplifying the grid generation process. However, the problem becomes how to perform this domain connectivity reliably and efficiently, and how to interpolate data between grids without impacting the overall accuracy of the simulation. Traditional methods relied on largely serial algorithms to find the regions of overlap between the grids, and then used linear interpolation to transfer data between grids. Recently, Crabill et al. [11] developed a parallel direct cut algorithm to efficiently leverage modern hardware to perform the hole-cutting stage of the domain connectivity process, and showed that leveraging the high-order polynomial solution representation within each element for inter-grid interpolation retains high-order accuracy in the presence of overset boundaries.

Our objective in this paper is to present ZEFR, an open-source CFD code for solving unsteady problems on unstructured overset domains using the FR approach. Written primarily in C++ ZEFR is capable of running on both CPU and NVIDIA GPU platforms. In addition, using the message passing interface (MPI) ZEFR is able to scale out on leadership class computing clusters. The remainder of this paper is structured as follows. In Section 2 we present the compressible Euler and Navier–Stokes equations and provide a high level overview of the FR approach. Moreover, we also outline how it can be extended to support overset domains. Our GPU accelerated implementation of FR, called ZEFR, is outlined in Section 3. In Section 4 we use a range of benchmark flow problems to validate the accuracy of ZEFR. Single node performance is assessed in Section 5 and the scalability of ZEFR is demonstrated in Section 6. Finally, in Section 7 conclusions are drawn.

2 Theory

2.1 Compressible Euler and Navier–Stokes equations

The governing systems of interest are the compressible Euler equations, which govern inviscid flow, and the compressible Navier–Stokes equations, which govern viscous flow. In conservative form the three dimensional Euler equations can be expressed as

$$\frac{\partial u}{\partial t} + \boldsymbol{\nabla} \cdot \mathbf{f} = 0, \tag{1}$$

where

$$u = \begin{Bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ E \end{Bmatrix}, \qquad \mathbf{f}(u) = \mathbf{f}^{(\text{inv})} = \begin{Bmatrix} \rho v_x & \rho v_y & \rho v_z \\ \rho v_x^2 + p & \rho v_y v_x & \rho v_z v_x \\ \rho v_x v_y & \rho v_y^2 + p & \rho v_z v_y \\ \rho v_x v_z & \rho v_y v_z & \rho v_z^2 + p \\ v_x(E+p) & v_y(E+p) & v_z(E+p) \end{Bmatrix}, \tag{2}$$

here ρ is the mass density of the fluid, $\mathbf{v} = (v_x, v_y, v_z)^T$ is the fluid velocity vector, E is the total energy per unit volume and p is the pressure. For a perfect gas the pressure and total energy can be related by the ideal gas law

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho \|\mathbf{v}\|^2, \tag{3}$$

where $\gamma = c_p/c_v$ is the ratio of specific heats.

The more general Navier–Stokes equations can be written as an extension of the Euler equations through the inclusion of viscous terms. Within the presentation outlined above the flux now takes the form of $\mathbf{f}(u, \boldsymbol{\nabla} u) = \mathbf{f}^{(\text{inv})} - \mathbf{f}^{(\text{vis})}$ where

$$\mathbf{f}^{(\text{vis})} = \begin{Bmatrix} 0 & 0 & 0 \\ \mathcal{T}_{xx} & \mathcal{T}_{yx} & \mathcal{T}_{zx} \\ \mathcal{T}_{xy} & \mathcal{T}_{yy} & \mathcal{T}_{zy} \\ \mathcal{T}_{xz} & \mathcal{T}_{yz} & \mathcal{T}_{zz} \\ \sum_i v_i \mathcal{T}_{ix} + \Delta \partial_x T & \sum_i v_i \mathcal{T}_{iy} + \Delta \partial_y T & \sum_i v_i \mathcal{T}_{iz} + \Delta \partial_z T \end{Bmatrix}. \tag{4}$$

In the above we have defined $\Delta = \mu c_p / P_r$ where μ is the dynamic viscosity and P_r is the Prandtl number. The components of the stress-energy tensor are given by

$$\mathcal{T}_{ij} = \mu(\partial_i v_j + \partial_j v_i) - \frac{2}{3}\mu \delta_{ij} \boldsymbol{\nabla} \cdot \mathbf{v}, \tag{5}$$

where δ_{ij} is the Kronecker delta. Using the ideal gas law the temperature can be expressed as

$$T = \frac{1}{c_v} \frac{1}{\gamma - 1} \frac{p}{\rho}, \tag{6}$$

with partial derivatives thereof being given according to the quotient rule.

2.2 Direct Flux Reconstruction in one dimension

In this section we will provide an overview of how the FR approach can be used to solve an advection-diffusion problem in one dimension. In order to simplify the presentation we will describe a variation of FR known as *direct* FR (DFR). DFR was introduced by Romero, Asthana, and Jameson [12] and proven to recover the nodal DG form of the standard FR scheme in one dimension under constraints on solution point locations specified in the cited work.

Consider a scalar conservation law of the form

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0, \tag{7}$$

defined within a one-dimensional domain $\Omega \in \mathbb{R}$ where x is the spatial coordinate, t is time, $u = u(x,t)$ is a scalar quantity and $f = f(u, \frac{\partial u}{\partial x})$ is a flux function which depends on the scalar and its derivative. For advection-diffusion, the flux can be split as $f = f_{\text{adv}} + f_{\text{dif}}$, with $f_{\text{adv}} = f_{\text{adv}}(u)$ corresponding to the convective flux which depends only on the scalar u, and $f_{\text{dif}} = f_{\text{dif}}(u, \frac{\partial u}{\partial x})$ corresponding to the diffusive flux which depends on the scalar and its derivative. Eq. (7) can be split into a system of first-order equations as

$$\frac{\partial u}{\partial t} + \frac{\partial f(u,q)}{\partial x} = 0, \tag{8}$$

$$q - \frac{\partial u}{\partial x} = 0, \tag{9}$$

with the introduction of a new variable, q, which is referred to as the auxiliary variable.

Next, consider partitioning the domain Ω into N non-overlapping elements, such that $\Omega = \cup_{n=1}^{N} \Omega_n$, with each element defined as $\Omega_n = \{x | x_n \leq x < x_{n+1}\}$. Define an approximate element local system

$$\frac{\partial u_n}{\partial t} + \frac{\partial f_n}{\partial x} = 0, \tag{10}$$

$$q_n - \frac{\partial u_n^C}{\partial x} = 0, \tag{11}$$

where $u_n = u_n(x,t)$, $f_n = f_n(x,t)$, $q_n = q_n(x,t)$, and $u_n^C = u_n^C(x,t)$ are piecewise polynomial functions defined within Ω_n, taking the value of zero elsewhere in the domain Ω. The global approximations to solution u and flux f are defined as

$$u \approx \sum_{n=1}^{N} u_n \quad \text{and} \quad f \approx \sum_{n=1}^{N} f_n. \tag{12}$$

As a requirement for conservation, f must be at least C^0-continuous at element boundaries. The approximate solution u is allowed to be discontinuous at element boundaries; however, to facilitate the computation of q_n, an additional global quantity $u^C = \sum_{n=1}^{N} u_n^C$, which is referred as the *continuous* solution, is introduced that maintains at least C^0-continuity at element boundaries.

To simplify the formulation, the elements Ω_n are each transformed to standard reference elements $\tilde{\Omega}_n = \{r | -1 \leq r \leq 1\}$ via a mapping function $x = \theta_n(r)$ with associated geometric

Jacobian, $J_n = \frac{\partial \theta_n}{\partial r}$ where r is the reference coordinate. Applying this mapping to Eq. (10) and Eq. (11) yields a transformed system within each $\tilde{\Omega}_n$

$$\frac{\partial \tilde{u}_n}{\partial t} + \frac{1}{J_n}\frac{\partial \tilde{f}_n}{\partial r} = 0, \tag{13}$$

$$\tilde{q}_n - \frac{1}{J_n}\frac{\partial \tilde{u}_n^C}{\partial r} = 0, \tag{14}$$

where

$$\tilde{u}_n = \tilde{u}_n(r,t) = u_n(\theta_n(r),t), \tag{15}$$

$$\tilde{f}_n = \tilde{f}_n(r,t) = f_n(\theta_n(r),t), \tag{16}$$

$$\tilde{q}_n = \tilde{q}_n(r,t) = q_n(\theta_n(r),t), \tag{17}$$

$$\tilde{u}_n^C = \tilde{u}_n^C(r,t) = u_n^C(\theta_n(r),t). \tag{18}$$

To begin, define a set of $P+1$ *solution* points $\{r_1, r_2, \ldots, r_{P+1}\}$ in the interior of the standard element. Next, define a set of $P+3$ *flux* points $\{r_0, r_1, r_2, r_3, \ldots, r_{P+1}, r_{P+2}\}$ which includes the previously defined solution points as well as additional points on the element boundaries, $r_0 = -1$ and $r_{P+2} = 1$. The number of solution points and number of flux points support polynomial interpolation of degree P and $P+2$ respectively. Typically, the solution point locations are collocated with the zeros of the Legendre polynomial of degree $P+1$, also referred to as the Gauss-Legendre quadrature points. This choice of solution points was shown by Romero et. al. to recover the nodal DG variant of the standard FR scheme [12]. Note that by definition, the flux points within the interior of the element and solution points are collocated.

The functions $\tilde{u}_n$, $\tilde{q}_n$, and $\tilde{f}_n$ are represented using Lagrange interpolating polynomials as

$$\tilde{u}_n = \sum_{j=1}^{P+1} u_{j,n}\ell_j, \tag{19}$$

$$\tilde{q}_n = \sum_{j=1}^{P+1} q_{j,n}\ell_j, \tag{20}$$

$$\tilde{f}_n = f^I_{n,0}\bar{\ell}_0 + \sum_{j=1}^{P+1} f_{j,n}\bar{\ell}_j + f^I_{n,P+2}\bar{\ell}_{P+2}, \tag{21}$$

where $u_{j,n}$ and $q_{j,n}$ are the solution and auxiliary variable values sampled at the solution point r_j, $f_{j,n} = f_{\text{adv}}(u_{j,n}) + f_{\text{dif}}(u_{j,n}, q_{j,n})$ are the flux values computed at the solution point r_j, and $f^I_{n,0}$ and $f^I_{n,P+2}$ are imposed common interface flux values at the left and right element boundary flux points respectively. ℓ_j are Lagrange basis polynomials of degree P defined using the solution points and $\bar{\ell}_j$ are Lagrange basis polynomials of degree $P+2$ defined using the flux points such that

$$\ell_j(r_i) = \delta_{i,j} \quad \text{for } i,j = 1,2,...,P+1, \tag{22}$$

$$\bar{\ell}_j(r_i) = \delta_{i,j} \quad \text{for } i,j = 0,1,...,P+2, \tag{23}$$

where $\delta_{i,j}$ is the Kronecker delta which takes the value of one if $i = j$ and zero otherwise. The procedure to compute the auxiliary variable values $q_{j,n}$ to complete the definition of Eq. (20) will be discussed in the subsequent section.

The common interface flux values $f^I_{n,j}$ are imposed at the boundary flux points in Eq. (21) to meet continuity requirements of the flux function for conservation. The common interface flux value can be split into two components

$$f^I_{n,j} = F_{\text{adv}}(u_L, u_R) + F_{\text{dif}}(u_L, u_R, q_L, q_R), \tag{24}$$

where the subscripts $\square_L$ and $\square_R$ refer to values at the flux point computed using information from the left and right elements at the boundary respectively, and $F_{\text{adv}}(u_L, u_R)$ and $F_{\text{dif}}(u_L, u_R, q_L, q_R)$ are common flux functions associated with the advective and diffusive fluxes respectively. A typical choice for F_{adv} is the Rusanov flux function [13], which is also referred to as the Lax–Friedrichs flux [4] for linear fluxes, and a typical choice for F_{dif} is the local discontinuous Galerkin (LDG) [3] flux function.

The Rusanov flux function is of the form,

$$F_{\text{adv}}(u_L, u_R) = \frac{1}{2}\big[f_{\text{adv}}(u_L) + f_{\text{adv}}(u_R)\big] + \frac{c}{2}[u_L - u_R], \tag{25}$$

where c is an approximation of the wavespeed. The LDG flux function is of the form,

$$\begin{aligned} F_{\text{dif}}(u_L, u_R, q_L, q_R) =& \frac{1}{2}\big[f_{\text{dif}}(u_L, q_R) + f_{\text{dif}}(u_R, q_R)\big] \\ &+ \beta\big[f_{\text{dif}}(u_L, q_L) - f_{\text{dif}}(u_R, q_R)\big] \\ &+ \tau\big[u_L - u_R\big], \end{aligned} \tag{26}$$

where β is a switch parameter controlling upwinding, and τ is a penalty parameter.

With the common flux values defined, substitution of Eq. (21) into Eq. (13) yields the semi-discrete update equation for the DFR scheme

$$\frac{\mathrm{d}\tilde{u}_n}{\mathrm{d}t} + \frac{1}{J_n}\left[f^I_{n,0}\frac{\mathrm{d}\bar{\ell}_0}{\mathrm{d}r} + \sum_{j=1}^{P+1} f_{j,n}\frac{\mathrm{d}\bar{\ell}_j}{\mathrm{d}r} + f^I_{n,P+2}\frac{\mathrm{d}\bar{\ell}_{P+2}}{\mathrm{d}r}\right] = 0. \tag{27}$$

Eq. (27) describes a system of ordinary differential equations in t which can be marched forward in time using a number of time integration schemes, for example, the classical fourth order Runge–Kutta (RK4) scheme.

To complete the definition of the method in one dimension, a procedure to compute the auxiliary variable values $q_{j,n}$ must be discussed. Since dependence on this variable exists only for the diffusive flux, the preceding sections provide a complete description of the direct FR method for a pure advection problem.

To compute the auxiliary variable, a continuous solution function $\tilde{u}^C_n$ is defined using Lagrange interpolating polynomials as

$$\tilde{u}^C_n = u^I_{n,0}\bar{\ell}_0 + \sum_{j=1}^{P+1} u_{j,n}\bar{\ell}_j + u^I_{n,P+2}\bar{\ell}_{P+2}, \tag{28}$$

where $u^I_{n,0}$ and $u^I_{n,P+2}$ are imposed common solution values at the left and right boundary flux points respectively. These common solution values are imposed at element boundaries to enforce desired continuity properties of $\tilde{u}^C$. The common solution values are computed as

$$u^I_{n,j} = U(u_L, u_R), \tag{29}$$

where $U(u_L, u_R)$ is a common solution function, typically associated with the chosen common viscous flux function F_{dif}. For the LDG flux, the common solution function takes the form,

$$U(u_L, u_R) = \frac{1}{2}[u_L + u_R] - \beta[u_L - u_R]. \tag{30}$$

Differentiation of Eq. (28) yields

$$\frac{\partial \tilde{u}^C_n}{\partial r} = u^I_{n,L}\frac{\mathrm{d}\bar{\ell}_0}{\mathrm{d}r} + \sum_{j=1}^{P+1} u_{j,n}\frac{\mathrm{d}\bar{\ell}_j}{\mathrm{d}r} + u^I_{n,R}\frac{\mathrm{d}\bar{\ell}_{P+2}}{\mathrm{d}r} \tag{31}$$

which leads to the definition of the auxiliary variable values $q_{j,n}$ as

$$q_{j,n} = \frac{1}{J_n}\frac{\partial \tilde{u}^C_n}{\partial r}\bigg|_{r=r_j} \tag{32}$$

Substitution of values from Eq. (32) into Eq. (20) define function $\tilde{q}_n$ and completes the description of the DFR method in one dimension for advection-diffusion type problems.

2.3 Extension to multiple dimensions

The direct FR approach can be readily extended to quadrilaterals and hexahedra by way of a tensor-product construction. Further details can be found in [8, 14, 15, 16] and the references therein. An example arrangement of solution and flux points resulting from such a construction can be seen in Fig. 1. Extensions to simplex elements have also been developed [17].

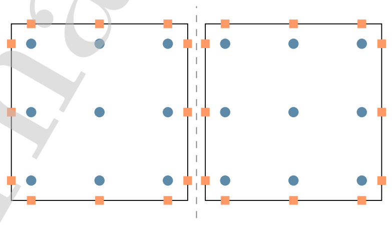


Figure 1: Example arrangement of solution points (blue circles) and boundary flux points (orange squares) inside a pair of $p = 2$ quadrilateral elements.

2.4 Extension to overset domains

In order to extend FR to overset domains we employ the parallel direct cut approach, which is a variation on the *artificial boundary* method. Here, the outer boundaries of

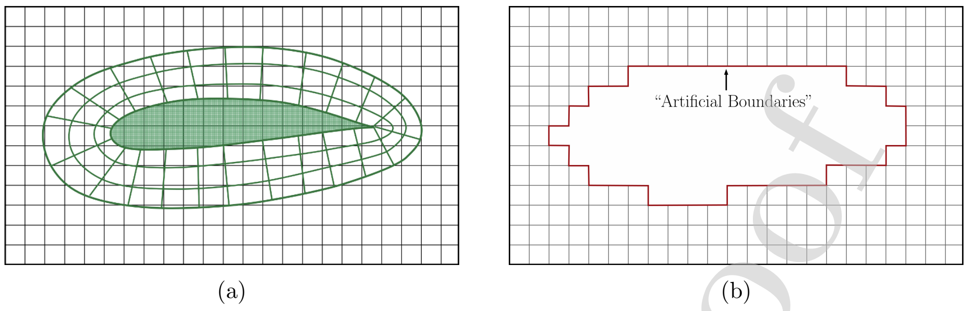
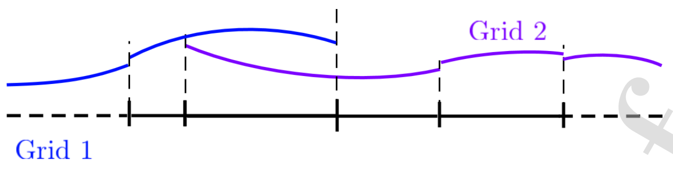


Figure 2: Example of the basic hole cutting and artificial boundary creation process on a simple airfoil grid. (a) Simple body grid/background grid overset system. (b) The artificial boundaries which would be created from the grid system.

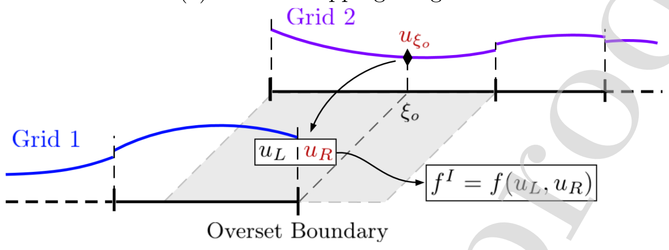
body-fitted grids—along with the surfaces of the bodies themselves—are used to 'cut' holes in other grids in the system. A simple two-grid example in two dimensions is shown in Figure 2. After the overlapping elements have been removed from a grid, the surface of the cut is termed an artificial boundary (AB) and this is where all inter-grid communication occurs. While traditional overset methods lose substantial accuracy at the boundaries between grids through the use of simple low-order interpolation methods [18], the artificial boundary approach maintains high–order accuracy across overset boundaries [19, 20, 21, 11, 22]. The authors have also shown using linear numerical analysis [22] that, while the artificial boundary method introduces anti-dissipation in 1D, the magnitude is insignificant when considering the accuracy of the solver on viscous problems. These factors make it possible to simulate complex, vortex-dominated flows such as those around rotorcraft, high-lift wing systems, and even spinning golf balls [23].

Other works have used high-order finite difference [24, 18, 25, 26] or strand [27, 28] methods on overset grids for the same purpose. The downside, however, arises with the domain connectivity step. Finite difference and finite volume overset methods require layers of 'fringe' elements or points in order to interpolate data from an active portion of one grid to another. Each additional order of accuracy requires at least two extra layers of fringe points to be added to the system—one on each side of the overset region. Finding valid interpolation stencils for these fringe points is a research area in and of itself. Several studies have combined these classical high-order methods with high-order FR or DG methods [29, 27, 30, 28]; others have used DG for all grids in the overset system [31]. All, however, have required some form of volume interpolation, which necessitates a minimum overlap requirement for data interpolation. The artificial boundary method sidesteps the issue by using the high-order solution polynomials present in every element to provide data interpolation to any points which lie within or on its boundary. Additionally, as Duan and Wang have shown, the artificial boundary method incurs less error than traditional volume interpolation for FR or DG methods [29].

In one dimension, FR on an overset domain using the artificial boundary method contains only one modification over the single-grid version. Considering the common flux function at an artificial boundary interface, where before the left and right solution val-



(a) Two overlapping 1D grids.



(b) AB interpolation procedure for the blue grid

Figure 3: Schematic of AB method in 1D, as seen from Grid 1's overset boundary.

ues u_L and u_R were extrapolated to the boundary from the elements on either side, now one side (say, u_L) is extrapolated as normal, but the other side (u_R) is the value of the solution interpolated from the opposite grid at the location of the artificial boundary, as shown in Figure 3. This necessitates mapping the physical location of the (flux) point x_i to the reference-domain position ξ_i inside of the overlapping element. In one dimension this mapping can be done quite simply with an analytic mapping, but in multiple dimensions using general curved elements, this mapping consumes a considerable amount of the effort involved in the domain connectivity. Prior works go into further details about fast, reliable methods to perform this connectivity on both CPU and GPU architectures [11, 22].

In two dimensions, the process is shown in Figure 4. First, the flux points on all artificial boundary interfaces are collected. Next, the donor element for each point is found, typically by building an alternating digital tree [32] of all possible donors and searching this tree for every point. Finally, once a potential donor is found, Newton iterations are used on the isoparametric spatial mapping of the donor element to find the reference coordinates (ξ, η) of the point (x, y) within the donor.

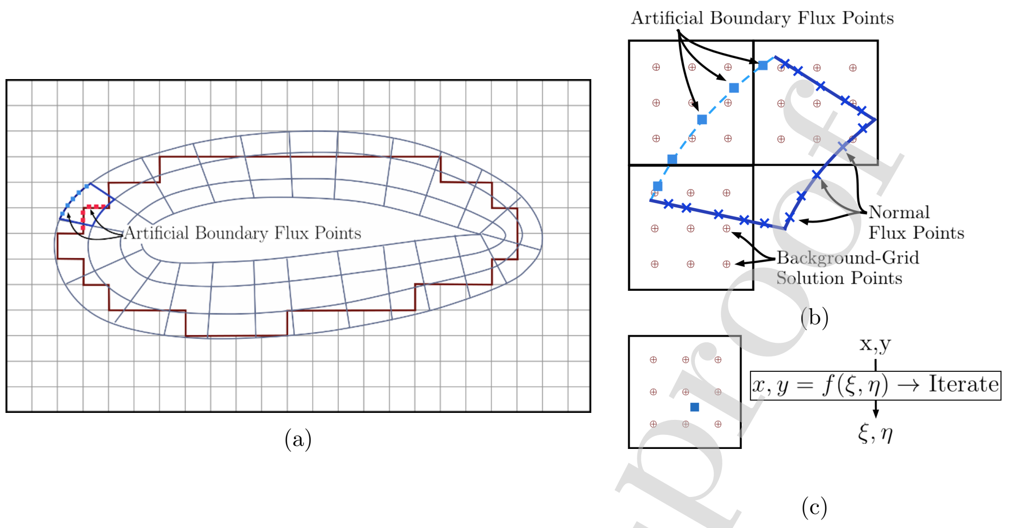

Artificial Boundary Flux Points
(a)
Artificial Boundary Flux Points
Normal Flux Points
Background-Grid Solution Points
(b)
x,y
$x, y = f(\xi, \eta) \rightarrow$ Iterate
ξ, η
(c)

Figure 4: Example of AB fringe point identification and connectivity. (a) Several fringe nodes identified from the grid system in Figure 2. (b) Closeup of (a), highlighting the airfoil-grid flux points tagged as receptor points. (c) The inverse-isoparametric mapping iterative method is used to find the reference location for each fringe point.

3 Implementation

The methods described in Section 2 are implemented in ZEFR using a combination of C++ and CUDA. ZEFR is based on the DFR method and includes the key functionality summarized in Table 1.

The main operations in ZEFR can be split into two categories: matrix-matrix products and pointwise operations. The pointwise operations required by DFR, are equivalent to those used for FR and comprehensive details on their implementation on GPUs can be found in the works of Castonguay et al. [33], Witherden et al. [9], and the references therein. Our focus in this section will be on the modifications to the algorithm as a result of using the DFR scheme of Section 2.

3.1 Data Array Layout

To begin, the details of the fundamental data arrays used to store relevant data for the DFR scheme will be provided. For the DFR scheme, storage is required for both scalar data and vector data, located at either the internal solution points of the element or at the boundary flux points. Given this, we can define four specific array dimensions corresponding to each of these scenarios, listed in Table 2, where N_S is the number of solution points, N_V is the number of solution variables, N_E is the number of elements, N_D is the number of spatial dimensions, and N_{FB} is the number of boundary flux points. In this table, the arrays [$\texttt{U}_\texttt{S}$], [$\texttt{F}_\texttt{S}$], [$\texttt{dU}_\texttt{S}$], and [$\texttt{divF}_\texttt{S}$] are used to store the solution, flux, solution gradient, and flux divergence at the solution points respectively for all elements in the domain. Similarly,

Table 1: Key functionality of ZEFR.

Dimensions	2D, 3D
Spatial orders	Arbitrary
Time steppers	Adaptive explicit Runge–Kuta
Boundary conditions	Isothermal wall, Adiabatic wall, Characteristic Riemann invariant, Supersonic inflow, Supersonic outflow
Platforms	CPUs, NVIDIA GPUs
Inter-node communication	MPI w/CUDA awareness
Governing systems	Euler, compressible Navier–Stokes
Mesh formats	Gmsh, PyFR
Solution formats	Native, PyFR, VTK

Table 2: Array dimensions used for DFR scheme. Arrays are arranged in a row-major format.

Data type	Location	Dimension	Arrays in this category
Scalar	Solution Points	$N_S \times N_V \times N_E$	[$\mathtt{U_S}$], [$\mathtt{divF_S}$]
Vector	Solution Points	$N_D \times N_S \times N_V \times N_E$	[$\mathtt{F_S}$], [$\mathtt{dU_S}$]
Scalar	Boundary Flux Points	$N_{FB} \times N_V \times N_E$	[$\mathtt{U_F}$], [Ucomm], [Fcomm]
Vector	Boundary Flux Points	$N_D \times N_{FB} \times N_V \times N_E$	[$\mathtt{dU_F}$]

the arrays [$\mathtt{U_F}$], and [$\mathtt{dU_F}$] are used to store the solution and the solution gradient at the boundary flux points for all elements in the domain. The remaining arrays [Ucomm] and [Fcomm] are used to store the common solution and numerical flux values computed from the double-valued solution and gradient states defined at each flux point.

An important aspect to note is that these arrays are organized in a structure of arrays (SoA) format, where data corresponding to the same field is contiguous in memory. This is in contrast to an array of structures (AoS) format, where data corresponding to the same element is contiguous in memory. This data format is chosen as it leads to code that can be easily vectorized, a critical requirement for high-performing software on modern platforms which depend heavily on coalesced memory access patterns for performance.

The data arrays described are organized by element, meaning that values corresponding to specific elements and solution or flux points can be easily accessed. However, in order to compute common solution or flux values, a means of accessing left and right solution and gradient states at each boundary flux point must be established. This can be accomplished via a pre-processed lookup table organized by flux point pairs within the domain. The lookup table is comprised of pointers to base memory locations corresponding to the paired flux point data stored within the established element data arrays [$\mathtt{U_F}$] and [$\mathtt{dU_F}$], as well as integer strides to enable the access of other field variables and dimensional components. The ordering of flux point pairs in the lookup table can be arbitrarily established, enabling the grouping of flux point pairs into internal flux points, boundary flux points, and, during parallel execution, partition boundary flux points. Beyond this, the flux point pairs can

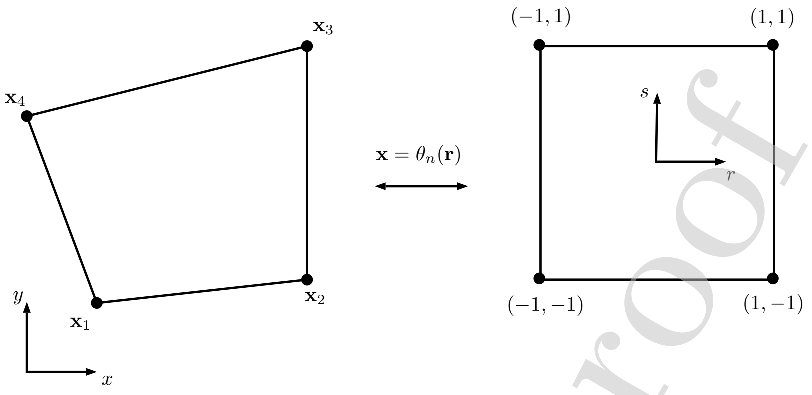


Figure 5: Transformation to standard quadrilateral element

be sorted to improve cache locality. This helps to further reduce the amount of memory indirection—something which is a common performance hurdle for unstructured CFD codes.

3.2 Definition of Matrix Operators

Many of the major operations of the DFR scheme can expressed via matrix operations. For efficiency, the operators for the DFR scheme are formulated to operate within a transformed space on *standard* elements, as opposed to operating on elements as physically defined. This transformation is described for one dimension in Eqs. (13) and (14), where the original elements are transformed to standard line elements, $\tilde{\Omega}_n = \{r| -1 \leq r \leq 1\}$. Similar transformations exist for elements in two and three dimensions. For example, on a quadrilateral mesh, each element Ω_n is transformed to standard reference square element $\tilde{\Omega}_n$ via an isoparametric mapping of the form $\boldsymbol{x} = \theta_n(\boldsymbol{r})$ with associated geometric jacobian matrix, $\boldsymbol{J}_n$, where $\boldsymbol{r} = (r, s)$ is the reference coordinate. Fig. 5 contains a visual depiction of this transformation.

Using standard elements allows for the definition of matrix operators that are constant for all elements of the same type, with physical geometry information preserved in element-wise geometric Jacobian matrices. These constant operators enable computations across the entire set of elements of a given type via large matrix-matrix products. This is in contrast to a formulation using physically defined elements, where the matrix operators will typically be uncommon across elements of the same type, requiring individual per element matrix-vector products to perform the same computation.

While the specific element type defines the actual content of these operators, the formulation algebraically remains fairly consistent.

The first matrix operator to be defined is [OppE]

$$[\texttt{OppE}]_{i,j} = \phi_j(\boldsymbol{r}_i^F) \qquad \dim[\texttt{OppE}] = N_{FB} \times N_S, \tag{33}$$

where $\phi_j(\boldsymbol{r})$ is the interpolating basis function associated with internal solution point $\boldsymbol{r}_j$ for the particular element type and $\boldsymbol{r}_i^F$ is the location of boundary flux point i. For example, for a quadrilateral element, $\phi_j(\boldsymbol{r})$ would take the form of a product of one-dimensional Lagrange interpolating polynomials, $\phi_j(\boldsymbol{r}) = \phi_{(k,l)}(\boldsymbol{r}) = \ell_k(r)\ell_l(s)$. The purpose of this operator is to extrapolate the solution values from the internal solution points to the boundary flux points.

Next, operators are defined which relate to computing gradients of scalar fields. The matrix operators to be defined are $[\texttt{OppD}_S]^q$ and $[\texttt{OppD}_F]^q$

$$[\texttt{OppD}_S]^q_{i,j} = \frac{\mathrm{d}}{\mathrm{d}q}\tilde{\phi}_j(\boldsymbol{r}^S_i) \qquad \dim\,[\texttt{OppD}_S]^q = N_S \times N_S, \tag{34}$$

$$[\texttt{OppD}_F]^q_{i,j} = \frac{\mathrm{d}}{\mathrm{d}q}\tilde{\phi}_j(\boldsymbol{r}^F_i) \qquad \dim\,[\texttt{OppD}_F]^q = N_S \times N_{FB}, \tag{35}$$

where $\tilde{\phi}_j(\boldsymbol{r})$ is the extended Lagrange interpolating basis function associated with internal solution point $\boldsymbol{r}^S_j$ for the particular tensor-element type and q is the spatial dimension. Operators are defined for $q = r$ and $q = s$ in two-dimensions, and $q = r$, $q = s$, and $q = w$ in three-dimensions. For a quadrilateral element, $\tilde{\phi}_j(\boldsymbol{r})$ would take the form of a product of the extended one-dimensional Lagrange interpolating polynomials, $\tilde{\phi}_j(\boldsymbol{r}) = \tilde{\phi}_{(k,l)}(\boldsymbol{r}) = \tilde{\ell}_k(r)\tilde{\ell}_l(s)$. $[\texttt{OppD}_S]^q$ computes the contribution to the gradient in the q-direction from values located at the solution points. Similarly, $[\texttt{OppD}_F]^q$ computes the contribution to the gradient in the q-direction from values located at the boundary flux points. To compute the full solution gradient using single larger matrix operators, the direction specific gradient operators are vertically concatenated to form $[\texttt{OppD}_S]$ and $[\texttt{OppD}_F]$
For two-dimensions:

$$[\texttt{OppD}_S] = \begin{bmatrix}[\texttt{OppD}_S]^r\\ [\texttt{OppD}_S]^s\end{bmatrix} \qquad \dim\,[\texttt{OppD}_S] = 2N_S \times N_S, \tag{36}$$

$$[\texttt{OppD}_F] = \begin{bmatrix}[\texttt{OppD}_F]^r\\ [\texttt{OppD}_F]^s\end{bmatrix} \qquad \dim\,[\texttt{OppD}_F] = 2N_S \times N_{FB}. \tag{37}$$

For three-dimensions:

$$[\texttt{OppD}_S] = \begin{bmatrix}[\texttt{OppD}_S]^r\\ [\texttt{OppD}_S]^s\\ [\texttt{OppD}_S]^w\end{bmatrix} \qquad \dim\,[\texttt{OppD}_S] = 3N_S \times N_S, \tag{38}$$

$$[\texttt{OppD}_F] = \begin{bmatrix}[\texttt{OppD}_F]^r\\ [\texttt{OppD}_F]^s\\ [\texttt{OppD}_F]^w\end{bmatrix} \qquad \dim\,[\texttt{OppD}_F] = 3N_S \times N_{FB}. \tag{39}$$

The final matrix operators that need to be defined are those related to computing the divergence of vector fields, denoted $[\texttt{OppDiv}_S]$ and $[\texttt{OppDiv}_F]$ which compute contributions to the divergence from data at the solution points and flux points respectively.

For tensor-product elements, the divergence operators take similar form to the already established gradient operators. For the solution point operator, the direction specific gradient operators are horizontally rather than vertically concatenated. For the flux point operator, the direction specific gradient operators are added together. Specifically,

For two-dimensions:

$$[\texttt{OppDiv}_S] = \left[[\texttt{OppD}_S]^r, \quad [\texttt{OppD}_S]^s\right] \qquad \dim\,[\texttt{OppDiv}_S] = N_S \times 2N_S, \tag{40}$$
$$[\texttt{OppDiv}_F] = [\texttt{OppD}_F]^r + [\texttt{OppD}_F]^s \qquad \dim\,[\texttt{OppDiv}_F] = N_S \times N_{FB}. \tag{41}$$

For three-dimensions:

$$[\texttt{OppDiv}_S] = \left[[\texttt{OppD}_S]^r, \quad [\texttt{OppD}_S]^s, \quad [\texttt{OppD}_S]^w\right] \qquad \dim\,[\texttt{OppDiv}_S] = N_S \times 3N_S, \tag{42}$$
$$[\texttt{OppDiv}_F] = [\texttt{OppD}_F]^r + [\texttt{OppD}_F]^s + [\texttt{OppD}_F]^w \qquad \dim\,[\texttt{OppDiv}_F] = N_S \times N_{FB}. \tag{43}$$

3.3 Procedure for computing $\boldsymbol{\nabla_r \cdot F}$

With the data arrays and various matrix operators defined, the procedure to compute $\nabla_r \cdot \boldsymbol{F}$, the fundamental computation of the DFR scheme, can be specified. First, we decompose the full solution domain into several partitions and assign each partition to individual GPUs. The DFR solution procedure commences on each GPU independently, with communication of data between flux points on partition boundaries implemented using MPI communication. To achieve good parallel efficiency, the MPI communication is carried out asynchronously using non-blocking sends and receives. Conveniently, the operators used to compute the solution gradients, $[\texttt{OppD}_S]$ and $[\texttt{OppD}_F]$, and the flux divergence, $[\texttt{OppDiv}_S]$ and $[\texttt{OppDiv}_F]$, are split and allow contributions from data at internal solution points and data from element boundary flux points to be applied separately. Since MPI communication only ever involves data at boundary flux points, it is possible to hide MPI communication behind the applications of contributions from the internal solution point data. To further illustrate this, a description of the parallel procedure is given in Algorithm 1. Note that indentations indicate operations that overlap with MPI data movement.

There are a few additional details in the parallel implementation that are specific to GPUs. Typically, MPI communication requires data to be resident in CPU memory. Since the CPU and GPU do not share memory, all MPI communication must be preceded by some data movement between the GPU and CPU memories. This data movement normally occurs over the PCIe bus which connects the CPU to GPU which has fairly limited bandwidth. As such, additional efforts must be taken to hide this data movement as well. This can be accomplished in a number of ways. The first method involves using asynchronous memory copies between the CPU and GPU and hence can run concurrently with other GPU computation. The second method, which is becoming more common, is to use a *CUDA-aware* MPI implementation. These more modern MPI implementations can operate directly on data resident in GPU memory, handling all required data movement between CPU and GPU in the background. An additional benefit is that these implementations can also utilize alternative means of communication between GPU devices, like direct GPU memory access via RDMA, or direct peer-to-peer communication over PCIe or NVLINK.

3.4 Input deck

ZEFR requires two items to run a simulation: a mesh and an input file. ZEFR supports the Gmsh and PyFR mesh file formats. The input file is specified in a space-delimited format

```
[InputName] [InputValue]
```

Algorithm 1: Compute $\nabla_r \cdot \boldsymbol{F}$. Note that steps marked with an asterisk are those that require matrix-matrix products

```
1. Extrapolate solution at solution points to boundary flux points*:
   $[\mathtt{U_F}] = [\mathtt{OppE}][\mathtt{U_S}]$
2. Commence asynchronous communication of $[\mathtt{U_F}]$ between partitions via MPI:
       3. Compute common solution at internal partition flux points. Store in $[\mathtt{UComm_F}]$.
       4. Compute solution point contribution to transformed gradient*:
          $[\mathtt{dU_S}] = [\mathtt{OppD_S}][\mathtt{U_S}]$
5. Compute common solution at partition-boundary flux points. Store in $[\mathtt{UComm_F}]$.
6. Complete transformed gradient computation by adding flux point contributions*:
   $[\mathtt{dU_S}] \mathrel{+}= [\mathtt{OppD_F}][\mathtt{UComm}]$
7. Convert transformed gradient to physical gradient in place and compute flux at
   solution points. Store in $[\mathtt{F_S}]$.
8. Extrapolate physical gradient to boundary flux points*:
   For each dimension dim:
       $[\mathtt{dU_F}]_{\mathtt{dim}} = [\mathtt{OppE}][\mathtt{dU_F}]_{\mathtt{dim}}$
9. Commence asynchronous communication of $[\mathtt{dU_F}]$ between partitions via MPI
       10. Compute common flux at internal partition flux points. Store in $[\mathtt{FComm}]$
       11. Compute solution point contribution to flux divergence*:
           $[\mathtt{divF_S}] = [\mathtt{OppDiv_S}][\mathtt{F_S}]$
12. Compute common flux at partition-boundary flux points. Store in $[\mathtt{FComm}]$
13. Complete flux divergence computation by adding flux point contributions*:
    $[\mathtt{divF_S}] \mathrel{+}= [\mathtt{OppDiv_F}][\mathtt{FComm}]$
```

Table 3: Boundary condition abbreviations understood by the ZEFR input file.

Condition	Abbreviation
Isothermal wall	`wall_ns_iso`
Adiabatic wall	`wall_ns_adi`
Characteristic Riemann invariant	`char`
Supersonic inflow	`inlet_sup`
Supersonic outflow	`outlet_sup`
Periodic pair	`periodic`
Outer boundary of inner overset grid	`overset`

Sample mesh and input files can be found in the `examples` directory in the ZEFR release. Given a labeled mesh boundary, the boundary conditions given in Table 1 can be specified as

```
mesh_bound [MeshBoundary] [BoundaryConditionAbbr]
```

with the applicable abbreviations being Table 3.

The time stepping scheme is given by the input name `dt_scheme` and an input value. For example, `RK44` may be specified for an explicit four stage, fourth order Runge-Kutta method. The number of steps and a time step itself may be specified using the input names `n_steps` and `dt`, respectively. Alternatively, an adaptive error-based time step may be enabled using `adapt_dt 1` and the final time may be specified using the input name `tfinal`.

ZEFR solves the Euler or Navier–Stokes equations using the input `equation EulerNS` and by disabling or enabling the input name `viscous`. The properties of these equations are defined by the number of dimensions, `nDims`, the Mach number, `mach_fs`, and the Reynold's number, `Re_fs`.

The solution is printed into files using the VTK format with the input `write_paraview 1` or the PyFR format with the input `write_pyfr 1`. The output name and frequency are specified using `output_prefix` and `write_freq`, respectively. A complete list of all inputs can be found with the repository along with various examples which exercise the capabilities of the code.

4 Validation and verification

4.1 Couette flow

The Couette flow problem considers the flow between two parallel plates of infinite extent along the x-z plane separated by a distance H in the y direction. The upper plate moves at a constant velocity u_w in the x direction while the lower plate remains stationary, and each plate is held at the same fixed temperature T_w. For a fixed viscosity μ, the steady state solution is given by

$$\rho(\bar{y}) = \frac{\gamma}{\gamma - 1}\frac{2p_0}{2C_pT_w + P_r u_w^2 \bar{y}(1-\bar{y})}, \tag{44}$$

$$u(\bar{y}) = u_w \bar{y}, \tag{45}$$

$$v = 0, \tag{46}$$

$$w = 0, \tag{47}$$

$$p = p_0, \tag{48}$$

where $\bar{y} = y/H$ and p_0 is a constant pressure.

The problem is solved on a finite three-dimensional domain $\Omega \in [-1,1] \times [0,1] \times [-1,1]$ with periodic boundary conditions imposed in the x and z directions. Isothermal boundary conditions are imposed on the lower and upper and upper boundaries such that

$$T(x, y = 0, z, t) = 0, \tag{49}$$

$$T(x, y = 1, z, t) = T_w. \tag{50}$$

Additionally, a no-slip condition is imposed on the lower and upper boundaries such that

$$u(x, y = 0, z, t) = 0, \qquad v(x, y = 0, z, t) = 0, \qquad w(x, y = 0, z, t) = 0, \tag{51}$$

$$u(x, y = 1, z, t) = u_w, \qquad v(x, y = 1, z, t) = 0, \qquad w(x, y = 1, z, t) = 0, \tag{52}$$

which enforces zero velocity on the stationary plate, and a fixed x-velocity of u_w on the moving plate.

For this case we consider the L^2 norm of the error in the total energy, defined as

$$\sigma(t)^2 = \int_\Omega \left(E(x,y,z) - E^\delta(x,y,z,t)\right)^2 \mathrm{d}\mathbf{x}, \tag{53}$$

where E corresponds to the analytic energy and E^δ corresponds to the numerical energy. This integral can be evaluated using a Gaussian quadrature on an element-wise basis.

The initial conditions for the simulations are taken as $\rho(\bar{y}, t = 0) = \langle\rho\rangle$, $u(\bar{y}, t = 0) = u_w$, $v = 0$, $w = 0$, and $p = p_0$, where $\langle\rho\rangle$ denotes the average of the analytical density computed over the domain. This ensures that the total amount of mass within the periodic domain is consistent with that of the analytical solution. As Couette flow is a steady state problem it is necessary to define a convergence criterion. We regard the simulation as being converged when the residual of the y component of the momentum equation reaches a tolerance of 1.0×10^{-13}.

Given the extent and number of mesh elements inside Ω it is possible to obtain the characteristic mesh spacing h. Moreover, by observing how the converged L^2 energy error σ changes as a function of h it is possible to numerically ascertain the numerical order of accuracy of the solver. This can be accomplished by determining the slope of the best-fit line $\log h \propto \log \sigma$.

In order to enable us to verify the order of accuracy of ZEFR simulations were run on three structured hexahedral grids. Although two simulations are technically sufficient the addition of a third simulation provides us with a means of estimating the standard error. The grids themselves can be seen in Fig. 6 with the resulting errors and orders of accuracy being tabulated in Table 4. Looking at the table we observe that ZEFR achieves the expected $P + 1$ order of accuracy.

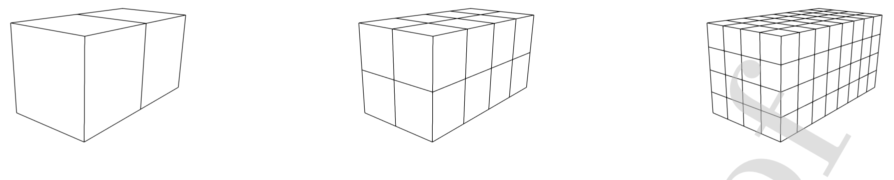


Figure 6: The extruded hexahedral grids employed for the Couette flow problem.

Table 4: L^2 energy errors and orders of accuracy for the Couette flow problem on three extruded hexahedral meshes.

	$\sigma\,/\,\mathrm{J\,m^{-3}}$		
Hexes	$P = 1$	$P = 2$	$P = 3$
2	7.68×10^{-2}	1.45×10^{-4}	2.09×10^{-5}
16	1.89×10^{-2}	2.13×10^{-5}	1.23×10^{-6}
128	4.70×10^{-3}	2.74×10^{-6}	8.38×10^{-8}
Order	2.015 ± 0.004	2.86 ± 0.06	3.98 ± 0.06

4.2 Taylor–Green vortex

Simulation of the Taylor–Green vortex breakdown using the compressible Navier–Stokes equations has been undertaken for the comparison of high-order numerical schemes. The problem has been utilized as a standard validation test case by a number of other studies including van Rees et al. [34] and within the AIAA International Workshops on High-Order CFD Methods. It is specified inside of a fully periodic box $\Omega = [-\pi L, \pi L]^3$, where L is the characteristic length, at Reynolds number $Re = 1\,600$ with an effectively incompressible Mach number of $M = 0.1$ [35]. The initial conditions are given as

$$u(x, y, z, t = 0) = U_0 \sin\frac{x}{L}\cos\frac{y}{L}\cos\frac{z}{L}, \tag{54}$$

$$v(x, y, z, t = 0) = -U_0 \sin\frac{x}{L}\cos\frac{y}{L}\cos\frac{z}{L}, \tag{55}$$

$$w(x, y, z, t = 0) = 0, \tag{56}$$

$$p(x, y, z, t = 0) = p_0 + \frac{\rho_0 U_0^2}{16}\Big(\cos\frac{2x}{L} + \cos\frac{2y}{L}\Big)\Big(\cos\frac{2z}{L} + 2\Big), \tag{57}$$

$$\rho(x, y, z, t = 0) = \frac{p(x, y, z, t = 0)}{RT_0}, \tag{58}$$

$$\tag{59}$$

where ρ_0 is the unperturbed density, p_0 is the unperturbed pressure, RT_0 is the product of the specific gas constant with a constant initial temperature, and $U_0 = M\sqrt{\gamma RT_0}$ is the unperturbed velocity. The test case is run to a final non-dimensional time of $t = 20t_c$ where $t_c = L/U_0$.

We are interested in the time histories of the total kinetic energy and enstrophy integrated over the domain, defined as

$$E_k = \frac{1}{\rho_0 U_0^2 |\Omega|} \int_\Omega \rho \frac{\mathbf{v} \cdot \mathbf{v}}{2} \,\mathrm{d}\mathbf{x}, \tag{60}$$

$$\epsilon = \frac{1}{\rho_0 U_0^2 |\Omega|} \int_\Omega \rho \frac{\boldsymbol{\omega} \cdot \boldsymbol{\omega}}{2} \,\mathrm{d}\mathbf{x}, \tag{61}$$

respectively, where $|\Omega| = (2\pi)^3$ is the volume of the domain, $\mathbf{v}$ is the velocity vector, and $\boldsymbol{\omega} = \boldsymbol{\nabla} \times \mathbf{v}$ is the vorticity. The enstrophy is of interest as, for incompressible flows, it is related to the kinetic energy decay rate via

$$\frac{\mathrm{d}E_k}{\mathrm{d}t} = 2\mu\epsilon/\rho_0.$$

Two simulations were performed, a run with a single grid, and an overset run with an inner grid. All simulations were performed with $p = 4$ solution polynomials on structured hexahedral grids. The primary grid—which also serves as the background grid for the overset case—has 52^3 elements. This equates to approximately 5×256^3 DOFs. For the overset case the inner grid sized to be 40% of the background grid and meshed with 24^3 elements. The inner grid is centred on the origin of the domain.

The kinetic energy decay rate and enstrophy for the simulations as a function of time can be seen in Figs. 7 and 8, respectively. Here we observe good agreement between the ZEFR simulations and the reference DNS results of van Rees et al. [34]. We remark here that the slight underprediction of the enstrophy compared is consistent with the $p = 4$ results of Vermeire et al. [2] who also employed FR with a similar number of DOFs.

4.3 Viscous flow over a SD7003

Finally, we investigate transitional and turbulent flow over an SD7003 aerofoil at Reynolds number 60 000. This problem is simulated within a cuboid shaped domain with dimensions $[-30, 70] \times [-30, 30] \times [0, 0.2]$. Within the domain, there is an extruded airfoil surface with a chord length, $c = 1$, located with the leading edge along the line $(0, 0, z)$, oriented at an angle of attack of 8 degrees. The Mach number is taken to be $M = 0.2$, the ratio of specific heats is $\gamma = 1.4$, and the Prandtl number is $Pr = 0.72$. The free-stream velocity U_∞ is set from the Mach number. Boundaries perpendicular to the z-axis are periodic, while the airfoil surface is treated with a no-slip, adiabatic wall condition. The remaining far-field boundaries are treated with Riemann-invariant characteristic conditions.

The domain was meshed using 203 532 hexahedral elements. This was generated by extruding an unstructured quadrilateral grid for 12 layers along the z-axis. The grid near the airfoil is structured, containing 15 layers normal to the surface of the airfoil. Additionally, there are two rectangular regions of increased resolution in the vicinity of the airfoil to capture flow near the airfoil and within the wake. In order to accurately capture the geometry of the airfoil elements on the surface were curved quadratically. Cross sections of the mesh can be seen in Fig. 9.

The simulation itself was run with $p = 4$ solution polynomials and adaptive Runge–Kutta time-stepping. From this we expect the simulation to be fifth order accurate in

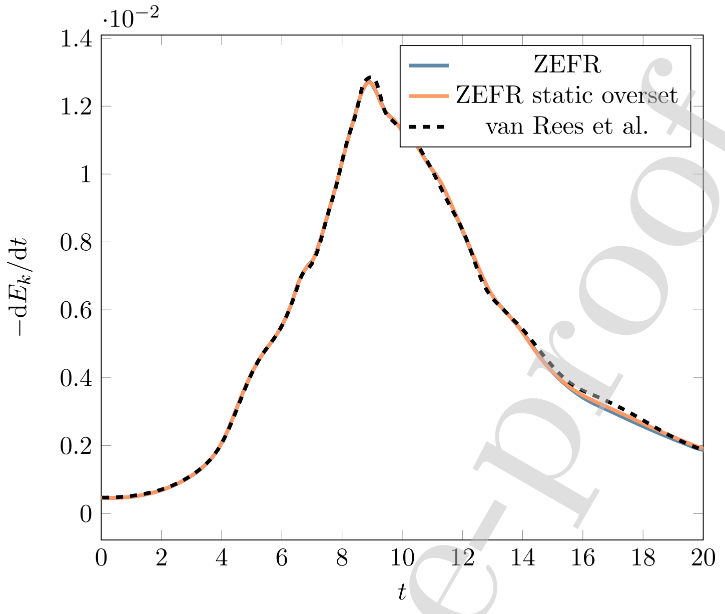


Figure 7: Kinetic energy decay rate for the Taylor–Green vortex simulations compared against the reference DNS results of van Rees et al. [34].

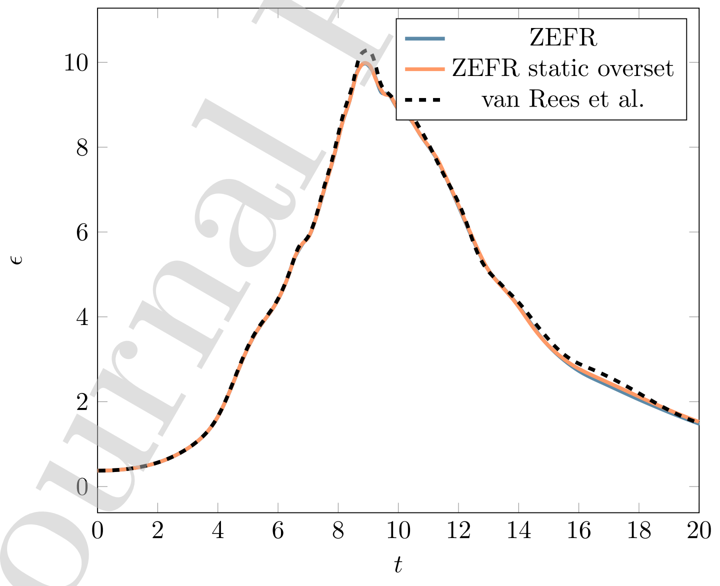


Figure 8: Enstrophy for the Taylor–Green vortex simulations compared against the reference DNS results of van Rees et al. [34].

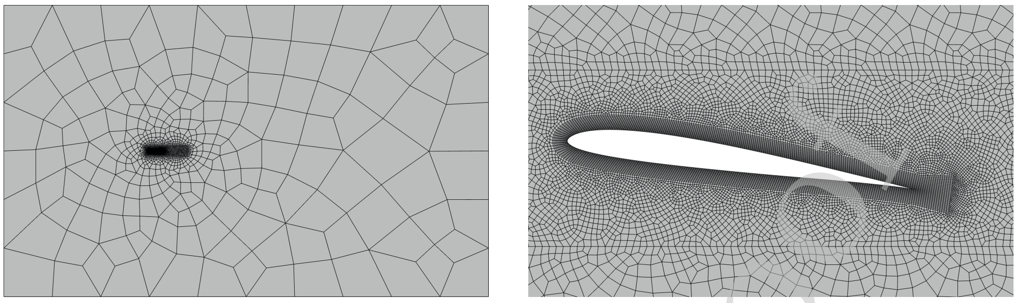

(a) (b)

Figure 9: Cross sections in the x-y plane of the grid used for the SD7003 test case.

Table 5: Comparison of average lift coefficient, drag coefficient, and separation point results for SD7003 test case.

Study	Method	C_L	C_D	x_s/c
Current	5th order FR	0.942	0.046	
Vermeire et al. [2]	5th order FR	0.941	0.049	0.315
Beck et al. [36]	4th order DG	0.923	0.045	0.310
Garmann et al. [37]	6th order FD	0.969	0.039	0.259

space. The simulation was run for a total of $30t_c$ where $t_c = c/U_\infty$ with time-average statistics being collected between $t = 20t_c$ and $t = 30t_c$. Table 5 shows the lift, drag, and separation points against those tabulated in the literature. Looking at the table we observe that the ZEFR results are consistent with those of previous numerical studies. A plot of the time averaged pressure coefficient C_P along the $z = 0.75$ plane can be seen in Fig. 10. Looking at the figure we observe excellent overall agreement with the results of Vermeire et al. [2].

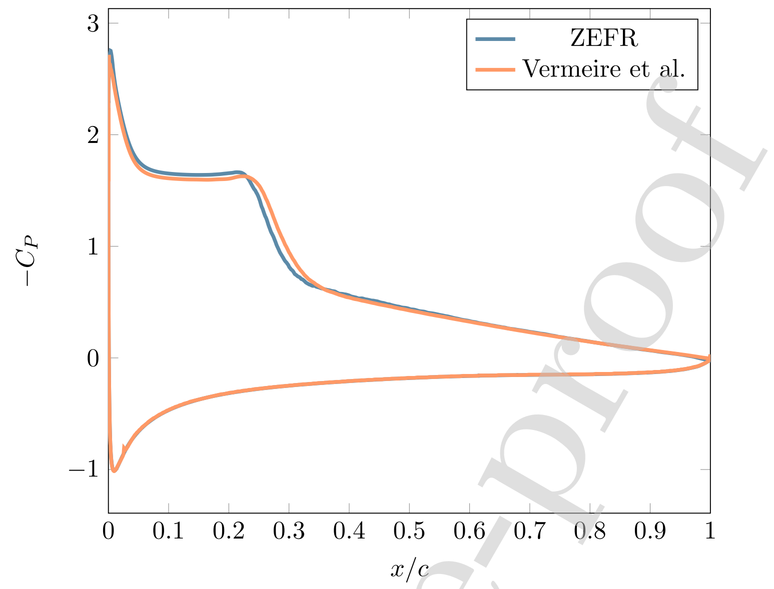


Figure 10: Pressure coefficient C_P as a function of x/c for the SD7003 simulation compared against the reference results of Vermeire et al. [2].

5 Single node performance

The single node performance of ZEFR has been evaluated on an NVIDIA V100 GPU. This accelerator has a peak memory bandwidth of ∼900 GB/s.

All performance tests were conducted using the regular (non overset) Taylor–Green vortex test case at orders $p = 3, 4, 5$. It is well known that FR on tensor product elements is a memory bandwidth bound algorithm [38]. Hence, our performance analysis will focus on measuring the amount of useful memory bandwidth sustained by ZEFR. Following Witherden et al. [9] we start by categorizing kernels as either sparse matrix multiplication (MM) kernels, pointwise kernels with a direct memory access pattern (PD), or pointwise kernels with some amount of indirect memory access (PI). The resulting breakdown of wall time across these kernels is shown in Fig. 11. Looking at the figure we observe that, at all three polynomials orders, the majority of wall time is spent performing matrix multiplications. Moreover, the fraction of wall time spent performing matrix multiplications tends to increase as a function of the polynomial order. These results are broadly inline with previous studies [9].

Using the NVIDIA profiler it is also possible to ascertain the amount of global memory traffic generated by each kernel. The average memory bandwidth sustained by the various kernel types in ZEFR can be seen in Fig. 12. We note that when generating the plot care was taken to suitably weight each kernel by its associated run time. We also verified the accuracy of these profiling results against hand-computed analysis of memory reads and writes of select indirect memory kernels. From the plot we observe that all three kernel

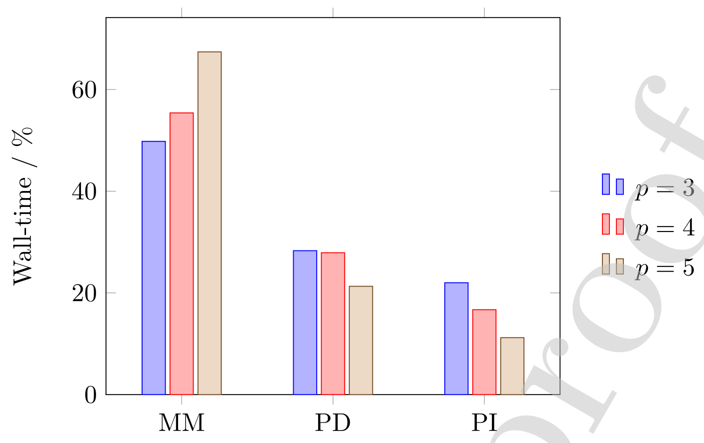


Figure 11: Wall time breakdown for the various kernel types in ZEFR on an NVIDIA V100 GPU.

Table 6: Time per DOF per right hand side evaluation for a Taylor–Green vortex simulation on an NVIDIA V100 GPU.

	Time per DOF / 10^{-9} s		
Case	$p = 3$	$p = 4$	$p = 5$
No overset	0.60	0.65	0.83
Static overset	0.81	0.88	1.10

types sustain a similar amount of bandwidth.

As an absolute measure of performance we consider the wall time required per degree of freedom to evaluate $\boldsymbol{\nabla} \cdot \mathbf{f}$. This quantity can be used to evaluate the performance of ZEFR relative to other codes. In over to evaluate this quantity we consider the Taylor–Green vortex at $p = 3, 4, 5$ on grids with 48^3, 40^3, and 34^3 hexahedra, respectively. These grids are sufficient to fully utilise all of the compute units on an NVIDIA V100 GPU. For overset an inner grid consisting of 24^3 hexahedra was employed. This grid was run on a separate NVIDIA V100 GPU with the simulation being split across two MPI ranks.

Time per DOF per right hand side evaluation can be seen in Table 6. Looking at the table we see that higher orders are slightly more expensive on a DOF basis than lower orders. Further, we see that the overhead associated with static overset is ∼35%.

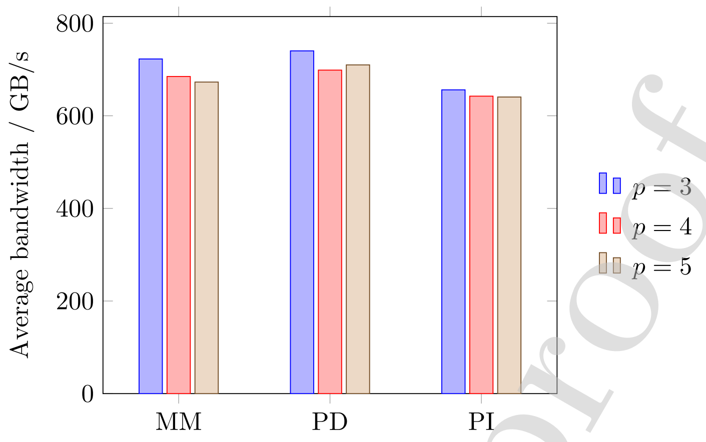


Figure 12: Average memory bandwidth sustained by the various kernel types in ZEFR on an NVIDIA V100 GPU.

6 Multi node performance

The scalability of ZEFR has been evaluated on the Summit supercomputer at Oak Ridge National Laboratory. Summit consists of 4 608 nodes with each node containing two 22 core IBM POWER9 CPUs and 6 NVIDIA Volta GPUs. Each GPU has 16 GiB of high bandwidth memory. Nodes are connected with dual-rail EDR InfiniBand.

Given that FR codes have a proven track record when it comes to weak scalability we decided to focus our efforts towards assessing the strong scalability of ZEFR. The chosen test case for our strong scaling study is an over-resolved Taylor–Green vortex simulation consisting of 195^3 hexahedral elements. The polynomial order is taken as $p = 4$. Such a configuration is sufficient to completely load 60 V100 GPUs (10 nodes). Moreover, this element count and polynomial order is typical of what one could expect to employ when performing an LES. In order to ensure that the scaling numbers are representative of what one could expect for a real simulation it was decided to *not* take advantage of the structured nature of the domain when partitioning the mesh. Instead, the generic METIS library [39] was employed.

Strong scaling results for ZEFR, both with and without CUDA aware MPI support enabled, can be seen in Table 7. Overall we can see that ZEFR continues to scale even when resources are increased by a factor of 64; albeit with a reduced parallel efficiency. Looking at the table it is also clear that that turning on CUDA aware MPI support *always* results in a *decrease* in runtime. Moreover, the benefit from CUDA aware MPI support appears to be similar at all scaling levels. As such the overall speedup numbers for ZEFR with CUDA aware MPI enabled are similar to those where it is disabled.

Table 7: Strong scalability of ZEFR for the Navier–Stokes equations. 'E' corresponds to using MPI with explicit host-to-device transfers whereas 'C' corresponds to leveraging CUDA aware MPI. All speedups are relative to the corresponding runtime on 60 GPUs.

		# V100s						
		60	120	240	480	960	1920	3840
E	Runtime	1.000	0.530	0.320	0.185	0.109	0.066	0.046
	Speedup	1.0	1.9	3.1	5.4	9.2	15.3	21.6
C	Runtime	0.917	0.496	0.295	0.170	0.098	0.059	0.042
	Speedup	1.0	1.8	3.1	5.4	9.3	15.6	21.8

7 Conclusions

In this paper we have described ZEFR, a GPU accelerated high-order accurate flow solver. The numerical approach employed by ZEFR has been detailed along with an overview of its extension to overset domains. The structure of the code has also been described. We have demonstrated the accuracy on ZEFR across a range of test cases including Couette flow, the Taylor–Green vortex, and flow over an SD7003 aerofoil. Single node performance has also been examined. When running on an NVIDIA V100 GPU we have shown that all bandwidth-bound kernels within ZEFR sustain a substantial fraction of the theoretical memory bandwidth. Finally, the strong scalability of ZEFR has been assessed on the Summit supercomputer with scaling demonstrated from 60 to 3840 NVIDIA V100 GPUs.

Acknowledgements

The authors would like to acknowledge the Army Aviation Development Directorate (AMRDEC) for providing funding for this research under the oversight of Roger Strawn and the Air Force Office of Scientific Research for their support through grant FA9550-14-1-0186 under the oversight of Fariba Fahroo. We would also like to thank the Oak Ridge Leadership Computing Facility for access to Summit under a director's discretion project award. J. Romero would like to acknowledge support from the Morgridge Family Stanford Graduate Fellowship. J. Watkins would like to acknowledge support from the National Science Foundation Graduate Research Fellowships Program.

References

[1] Freddie D Witherden and Antony Jameson. Future directions in computational fluid dynamics. In *23rd AIAA Computational Fluid Dynamics Conference*, page 3791, 2017.

[2] Brian C Vermeire, Freddie D Witherden, and Peter E Vincent. On the utility of GPU accelerated high-order methods for unsteady flow simulations: A comparison with industry-standard tools. *Journal of Computational Physics*, 334:497–521, 2017.

[3] Bernardo Cockburn and Chi-Wang Shu. The local discontinuous Galerkin method for time-dependent convection-diffusion systems. *SIAM Journal on Numerical Analysis*, 35(6):2440–2463, 1998.

[4] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: Algorithms, analysis, and applications*, volume 54. Springer Verlag New York, 2008.

[5] Zhi Jian Wang. Spectral (finite) volume method for conservation laws on unstructured grids. Basic formulation. *Journal of computational physics*, 178(1):210–251, 2002.

[6] David A Kopriva and John H Kolias. A conservative staggered-grid Chebyshev multidomain method for compressible flows. *Journal of computational physics*, 125(1):244–261, 1996.

[7] Yen Liu, Marcel Vinokur, and Zhi Jian Wang. Spectral difference method for unstructured grids i: Basic formulation. *Journal of Computational Physics*, 216(2):780–801, 2006.

[8] H T Huynh. A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods. *AIAA paper*, 4079, 2007.

[9] Freddie D Witherden, Antony M Farrington, and Peter E Vincent. PyFR: An open source framework for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. *Computer Physics Communications*, 185(11):3028–3040, 2014.

[10] Freddie D Witherden, Brian C Vermeire, and Peter E Vincent. Heterogeneous computing on mixed unstructured grids with PyFR. *Computers & Fluids*, 120:173–186, 2015.

[11] Jacob A Crabill, Freddie D Witherden, and Antony Jameson. A parallel direct cut algorithm for high-order overset methods with application to a spinning golf ball. *Journal of Computational Physics*, 2018.

[12] Joshua Romero, Kartikey Asthana, and Antony Jameson. A simplified formulation of the flux reconstruction method. *Journal of Scientific Computing*, 67(1):351–374, 2016.

[13] Vladimir Vasil'evich Rusanov. The calculation of the interaction of non-stationary shock waves and obstacles. *USSR Computational Mathematics and Mathematical Physics*, 1(2):304–320, 1962.

[14] Freddie D Witherden, Peter E Vincent, and Antony Jameson. High-order flux reconstruction schemes. In *Handbook of numerical analysis*, volume 17, pages 227–263. Elsevier, 2016.

[15] Abhishek Sheshadri and Antony Jameson. On the stability of the flux reconstruction schemes on quadrilateral elements for the linear advection equation. *Journal of Scientific Computing*, 67(2):769–790, 2016.

[16] Gianmarco Mengaldo, Daniele De Grazia, Peter E Vincent, and Spencer J Sherwin. On the connections between discontinuous Galerkin and flux reconstruction schemes: extension to curvilinear meshes. *Journal of Scientific Computing*, 67(3):1272–1292, 2016.

[17] Joshua Romero, Freddie D Witherden, and Antony Jameson. A direct flux reconstruction scheme for advection–diffusion problems on triangular grids. *Journal of Scientific Computing*, 73(2-3):1115–1144, 2017.

[18] Andrew Wissink. An overset dual-mesh solver for computational fluid dynamics. *7th International Conference on Computational Fluid Dynamics (ICCFD7)*, 2012.

[19] Marshall C Galbraith. *A Discontinuous Galerkin Overset Solver*. PhD thesis, University of Cincinatti, 2013.

[20] Marshall C. Galbraith, John A. Benek, Paul D. Orkwis, and Mark G. Turner. A discontinuous Galerkin scheme for chimera overset viscous meshes on curved geometries. *Computers and Fluids*, 119:176–196, 2015.

[21] Jacob A Crabill, Antony Jameson, and Jay Sitaraman. A high-order overset method on moving and deforming grids. *AIAA Modeling and Simulation Technologies Conference*, (AIAA2016-3225), 2016.

[22] Jacob A Crabill. *Towards Industry-Ready High-Order Overset Methods on Modern Hardware*. PhD thesis, Stanford University, 2018.

[23] Jacob A Crabill, Freddie D Witherden, and Antony Jameson. High-order computational fluid dynamics simulations of a spinning golf ball. *Sports Engineering*, 22(1):9, 2 2019.

[24] Thomas H. Pulliam. High order accurate finite-difference methods: as seen in OVERFLOW. *20th AIAA Computational Fluid Dynamics Conference*, 2011.

[25] Jayanarayanan Sitaraman, Matthew Floros, Andrew Wissink, and Mark Potsdam. Parallel domain connectivity algorithm for unsteady flow computations using overlapping and adaptive grids. *Journal of Computational Physics*, 229:4703–4720, 2010.

[26] Beatrice Roget and Jayanarayanan Sitaraman. Robust and efficient overset grid assembly for partitioned unstructured meshes. *Journal of Computational Physics*, 260:1–24, 2014.

[27] Aaron Katz and Dalon Work. High-order flux correction/finite difference schemes for strand grids. *Journal of Computational Physics*, 282:360–380, 2015.

[28] Dolan Work. *Evaluation of Flux Correction on Three-Dimensional Strand Grids with an Overset Cartesian Grid*. PhD thesis, Utah State University, 2017.

[29] Zhaowen Duan and Z. J. Wang. A high order FR/CPR method for overset strand and cartesian meshes and moving boundaries. *23rd AIAA Computational Fluid Dynamics Conference*, 2017.

[30] Michael J. Brazell, Andrew C. Kirby, and Dimitri J. Mavriplis. A high-order discontinuous-Galerkin octree-based AMR solver for overset simulations. *23rd AIAA Computational Fluid Dynamics Conference*, 2017.

[31] Michael J Brazell, Jayanarayanan Sitaraman, and Dimitri Mavriplis. An overset mesh approach for 3D mixed element high order discretizations. *Journal of Computational Physics*, 322:33–51, 2016.

[32] Javier Bonet and Jaime Peraire. An alternating digital tree algorithm for 3D geometric searching and intersection problems. *International Journal for Numerical Methods in Engineering*, 1991.

[33] Patrice Castonguay, David Williams, Peter Vincent, Manuel Lopez, and Antony Jameson. On the development of a high-order, multi-GPU enabled, compressible viscous flow solver for mixed unstructured grids. In *20th AIAA Computational Fluid Dynamics Conference*, page 3229, 2011.

[34] Wim M van Rees, Anthony Leonard, DI Pullin, and Petros Koumoutsakos. A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers. *Journal of Computational Physics*, 230(8):2794–2805, 2011.

[35] Z J Wang, Krzysztof Fidkowski, Remi Abgrall, Francesco Bassi, Doru Caraeni, Andrew Cary, Herman Deconinck, Ralf Hartmann, Koen Hillewaert, H T Huynh, Norbert Kroll, Georg May, Per-Olof Persson, Bram van Leer, and Miguel Visbal. High-order CFD methods: current status and perspective. *International Journal for Numerical Methods in Fluids*, 72(8):811–845, 2013.

[36] Andrea D Beck, Thomas Bolemann, David Flad, Hannes Frank, Gregor J Gassner, Florian Hindenlang, and Claus-Dieter Munz. High-order discontinuous Galerkin spectral element methods for transitional and turbulent flow simulations. *International Journal for Numerical Methods in Fluids*, 76(8):522–548, 2014.

[37] Daniel J Garmann, Miguel R Visbal, and Paul D Orkwis. Comparative study of implicit and subgrid-scale model large-eddy simulation techniques for low-Reynolds number airfoil applications. *International Journal for Numerical Methods in Fluids*, 71(12):1546–1565, 2013.

[38] Bartosz D Wozniak, Freddie D Witherden, Francis P Russell, Peter E Vincent, and Paul HJ Kelly. GiMMiK—generating bespoke matrix multiplication kernels for accelerators: Application to high-order computational fluid dynamics. *Computer Physics Communications*, 202:12–22, 2016.

[39] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Declaration of Interest Statement

Declaration of interests

☐ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Author Contributions Section

Author Contributions Section

All authors had an equal contribution to the manuscript, including collecting data, writing code, authoring the manuscript itself, and handling revisions.

Revised version of Program Description

Program Description

Program title: ZEFR v1.0

Program files doi: `http://dx.doi.org/10.17632/wzy83bscxd.1`

Licensing provisions: BSD 3-clause

Programming language: C++ and CUDA

External routines/libraries: Eigen, HDF5, METIS, MPI, and TIOGA.

Nature of problem: Compressible Euler and Navier–Stokes equations.

Solution method: High-order direct flux reconstruction approach suitable for curved, mixed, unstructured grids.

Unusual features: Code incorporates support for overset grids.